NATIONAL & KAPODISTRIAN UNIVERISTY OF ATHENS

DEPARTMENT OF ECONOMICS

MSc "FINANCIAL ADMINISTRATIVE AND BUSINESS INFORMATION SYSTEMS"

"A GRAPHICAL USER INTERFACE FOR TWO BASIC DYNAMIC PROGRAMMING PROBLEMS"

Paris Tsampras

Athens, February 2017

The candidate certifies that the submitted work is personal, unless reference is made to other works.

Paris Tsampras

Acknowledgement

I would like to thank Professor Stelios Kotsios, from whom I was taught the concepts of Dynamic Programming, Operational Research and Game Theory during my postgraduate studies in the Department of Economics, for his most valuable assistance and guidance.

ABSTRACT

The main purpose of this is study is to review software engineering technics that can be applied to solve Dynamic Programming problems, and secondly to design and develop a software application that solves two common dynamic programming problems upon user command. In order to achieve the above goals, this study reviews dynamic programming as a concept of solving computational problems generally but emphasizes in two well-known specific problems; the knapsack problem and the shortest route problem. Both of them present characteristics that make them ideal to be solved using dynamic programming concepts.

This study starts with an examination and comments on the background of both dynamic programming and of the two problems mentioned above. The second part consists of the software application that was developed as an integral part of this study. The software packages and their uses, the algorithms, the architectural design and pseudo code of the application are discussed thoroughly. This application was developed using the well-documented java programming language. The complete code is located on the annexes A and B. In this part there will be a presentation of the code divided in packages and procedures.

This study was conducted in Athens, Greece between October 2016 and February 2017. The technologies used to develop the software application besides the java programming language which was mentioned above are the Netbeans (version 8.2) Integrated Development Environment (IDE) installed on a Windows 7 64-bit operating system and the Java Development Kit compiler. In addition for graphs design the graph stream library was used.

Contents

1. Background

1.1. Dynamic Programming

By simply using the term Dynamic Programming we do not refer directly in software development in some programming language. The word "programming" comes from the concept of dealing with methods of combinatorial problems during the first half of the previous century, and is often misleading[1]. Since the tremendous development of informatics as a science, the word "programming" switched its meaning to software development but the term Dynamic Programming still remained in widespread use without changing its own meaning. The best way to describe what the term Dynamic Programming stands for was given by Richard Bellman, the founder of both the concept and the very term, in his principle of optimality as follows[2] : "the basic idea of the theory of dynamic programming is that of viewing an optimal policy as one determining the decision required at each time in terms of the current state of the system. Following this line of thought, the basic functional equations given below describing the quantitative aspects of the theory are uniformly obtained from the following intuitive. An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions". In other words by using the term Dynamic Programming we refer to the principle of dealing with a problem by breaking it down to smaller subproblems, compute each one once and store its solution. By doing that the overall solution of the problem comes from combining the solutions of the subproblems. The ultimate gain of dealing with a problem in this way is that every time a subproblem presents itself, it is not computed again but instead, the solution given from its first calculation is being used. To achieve that, dynamic programming algorithms make extensive use of arrays. They store the solution of each subprorblem in one or more arrays and access it whenever the same subproblem presents itself again. This has a profound effect on the time complexity of solving a problem; in fact many dynamic programming algorithms achieve time complexities that are far lower that the respective time complexities of recursive or greedy algorithms with several examples such as the problem of finding the Fibonacci sequence numbers which has an exponential time complexity when solved by recursion alone.

The most critical feature that a problem must have in order to be solved by using dynamic programming concepts is the optimal substructure, which means that the solution of a problem can be given through combining the optimal solutions of its respective subproblems. It is also important to note that the subproblems must be of an overlapping nature; if this is not the case then the solution is not given by dynamic programming but rather in a "divide and conquer" approach such as recursion. Not all problems have the above characteristics and as a consequence one must be very careful on applying dynamic programming to a specific problem. Another important property of dynamic programming is the "backward induction" which is a process of tracing the solution of a problem backwards, meaning that given a terminal state of a problem, in which supposedly the solution would be known, one can eliminate all non-optimal solutions of the various subproblems that preceded the final state. By using that method the optimal solution of all subproblems can be calculated. Backward induction may seem quite difficult to understand yet it is being used in

dynamic programming since the days of Bellman. It is actually one of the methods that were used to solve Bellman's equation[3] . Backward induction is also used extensively in game theory, a scientific field that employs the vast majority of the principles used in Dynamic Programming. Both John Von Neumann and Morgenstern, the founders of the game theory as a scientific field used the principle of backward induction to solve two-person games[4].

Dynamic Programming is all about optimization. It is one the most powerful tools of finding an optimized solution of a problem .Since the concept of optimization is applied on many scientific fields, dynamic programming methods have been used to solve problems in various quantitative sciences such as mathematics, economics, managerial decision making , biosciences and of course computer science. The most notable examples of dynamic programming usage in Bioinformatics are the RNA structure prediction and the protein DNA binding algorithms. Concerning decision making, many problems in this field can be divided in overlapping subproblems and solved by using one of shortest path finding algorithms like the Dijkstra or the Kruskal algorithm. What all the above fields have in common with respect to solving problems in a dynamic programming manner, is the approach by which the algorithms of solving the various problems are designed.

1.2. Design of Dynamic Algorithms

Before we proceed with the analysis of specific non probabilistic problems such as the Knapsack problem or the Shortest Route problem it is important to present the main features of a dynamic algorithm. To begin with, there is no standard dynamic algorithm for solving every problem but rather there are design patterns that can be applied in an algorithm in order to be characterized dynamic. The first feature of a dynamic programming algorithm is the definition of the substructure of the problem. Software wise that would be a recursive function or array, called as many times as the number of the subproblems of the main problem. The second feature is the extensive usage of tables (array data structures) to store the computations done for finding the solutions of the subproblems and finally, the third feature of a dynamic algorithm is that it must have some sort of a bottom up implementation.

The latter feature is strongly associated with the concept of backward induction which was discussed in the previous chapter. The bottom-up implementation of a dynamic algorithm is the programming tool for implementing such a concept. The bottom up implementation begins by finding the solutions of the subproblems beginning from the end of problem, that are by definition of a smaller scale, and proceed with the other subproblems backwards. In order for this technique to work, a dynamic algorithm must always define a final (or initial depending on the way we choose to call it) stage that is also by definition known.

To better understand the design of dynamic algorithms we have to present some examples algorithm and correlate their features to the ones described above. The dynamic algorithm for finding the Fibonacci sequence numbers is one of the best examples for someone to understand the design of a dynamic algorithm.

Example 1: The Fibonacci sequence is characterized by the fact that every number it contains is the sum of the last two that preceded it in the sequence[5]. The mathematical type

for this sequence is[5] $F_n = F_{n-1} + F_{n-2}$ given that $F_0 = 0$ and $F_1 = 1$. A dynamic programming algorithm that computes all the numbers of the Fibonacci sequence is the following:

```
 1. Fibonacci(int n)
 2.  if n = 0 or n = 1 then
 3.    return n
 4.  else if f[n] != -1 then
 5.    return f[n]
 6.  else
 7.    f[n] = Fibonacci (n - 1) + Fibonacci (n – 2)
 8.    return f[n]
 9.  end if
10. end
```

Algorithm 1 : Finding the n^{th} Fibonacci sequence number

The above dynamic algorithm implements the problem of finding the n^{th} number of the Fibonacci sequence. By examining it we can easily observe the definition of the substructure of the problem. It is given in a very clear way in line 7 and it contains two recursive function-calls that implement the mathematical type of the Fibonacci sequence definition. In addition the usage of tables for storing the solutions of the various subproblems can also be verified. The "f" table is a data structure used to store each calculation for a given index of the Fibonacci numbers. The bottom up implementation might be a bit harder to be observed. The concept of execution of this algorithm for a given input (for example 4) is the following:

- F[4] = F[3] + F[2]
- F[3] = F[2] + F[1]
- F[2] = F[1] + F[0]

This algorithm is using bottom-up implementation as it begins from the last index and proceeds solving subproblems towards the first index of the Fibonacci sequence. The initial conditions are by definition known as stated above and so by combining the solutions of the various subproblems the algorithm terminates giving the expected result.

Example 2: Another good example that employs a dynamic algorithm is the dice-throw problem. Imagine that you are given n dice each with m faces, numbered from 1 to m, find the number of ways to get sum X. X is the summation of values on each face when all the dice are thrown. The dynamic algorithm that implements this problem is given below:

```
 1.Dice(int m, int n, int x)
 2.for j=1 j<= m and j<=x j++
 3.   Solution[i][j] = 1
 4.end for
 5.for i=2 i<= n i++
 6.   for j=1 j<= x j++
 7.         for k=1 k<=m and k<j k++
 8.               Solution[i][j] =
 9.              Solution[i][j] +
10.               Solution[i-1][j-k]
11.         end for
12.    end for
13.end for
```

Algorithm 2 : Dice – throw algorithm

Again we can identify the features of a dynamic algorithm. First, the optimal substructure is given between lines 8 and 10. The recursive call on table "Solution" determines the solution of each subproblem by combining the solutions of the previous ones. The initial state which is by default known is given in the first "for-loop" and works as the basis for the bottom up implementation.

1.3. The knapsack Problem

The knapsack problem is one of most the famous problems in combinatorial optimization. The definition of the problem is quite simple. We are given a set of items, each having a specific weight and a specific value. We are also given a knapsack which has a specific capacity with respect to weight. The problem lies on finding what the maximum value of the items that can be carried in the knapsack is, provided that the overall weight of all items picked must not exceed the weight of the knapsack. The knapsack problem has been studied since the late seventeenth century[6] . The definition of the problem and its name was given by the mathematician Tobias Dantzig[6]. There are many known variations of the knapsack problem. The main reason for the differentiation is the restrictions applied on the solution regarding the number of copies of each item that can be carried in the knapsack or the subdivision units of the copies (in case that the knapsack can be filled with non-integer values of an item) .

On the "0 - 1" version of the knapsack problem an item can either be part of the solution or not. The term "being part of the solution" means that the specific item belongs to the optimal subset of the items that make up the optimal solution of the problem. But in this version the copies of each item that are included on the optimal subset are restricted to only one copy[7]. The mathematical definition of the "0-1" or Binary Knapsack Problem is the following[7]:

P_j = profit of item j

W_j = weight of item j

C = capacity of the knapsack

Maximize $z = \sum_{j=1}^{n} Pj * Wj$

Subject to $\sum_{j=1}^{n} Pj * xj < C$ and x_i belongs to [0, 1]

On the "bounded knapsack problem" the restriction of one-copy items only does not exist. On the other hand there is an upper bound limitation of the copies of the items that can be included in the optimal subset. This upper bound restricts the number of copies to be no more than the overall capacity of the knapsack[8] . The mathematical representation of this version is quite similar the "0-1"[8] version.

P_j = profit of item j

W_j = weight of item j

C = capacity of the knapsack

Maximize $z = \sum_{j=1}^{n} Pj * Wj$

Subject to $\sum_{j=1}^{n} Pj * xj < C$ and $0 < x_i < C$

Finally the "Unbounded knapsack problem" removes all limitations and restrictions regarding the number of copies of the items but one. The number of copies in this version must be a positive integer or zero. Again the mathematical type is given below:

P_j = profit of item j

W_j = weight of item j

C = capacity of the knapsack

Maximize $z = \sum_{j=1}^{n} Pj * Wj$

Subject to $\sum_{j=1}^{n} Pj * xj < C$ and $x_i >= 0$

There are also other variations of the knapsack problem such as the subset problem, the change making problem, the multiple knapsack problem, the generalized assignment problem and the bin packing problem but they refer to non-generic restrictions and they are not analyzed further in this study.

The knapsack problem can be solved by many types of algorithms such as the brute-force algorithm, the branch and bound algorithm[8] and the dynamic algorithm. In this study the dynamic algorithms are discussed. Regarding the first version of the knapsack problem (binary knapsack problem) the dynamic algorithm that gives the solution is given below[9]:

```
1. Knapsack(n,W,w,v)
2. begin
3. for j from 0 to W do:
4.    m[0, j] = 0
5. end for
6. for i from 1 to n do:
7.      for j from 0 to W do:
8.          if w[i] > j then:
9.              m[i, j] = m[i-1, j]
10.          else:
11.              m[i, j] = max(m[i-1, j], m[i-1, j-w[i]] +
12.                                                    v[i])
13.         end if
14.      end for
15. end for
16. end
```

Algorithm 3 : The Algorithm for the unbounded knapsack problem

All dynamic algorithms that solve knapsack problems are variations of the one given above. The analysis of this algorithm presents us with the following conclusions. Firstly the algorithm is given as input four parameters of two types, integers and arrays of integers.

- N = the number of items that make up the problem. The restrictions mentioned above apply on them.
- W= the capacity of the knapsack in terms of weight
- w= an array data structure which contains the various weights of the items in an indexed way
- v= an array data structure which contains the various values of the items indexed in the same way as the w array for consistency purposes.

By considering for simplicity that all weights and values are non-negative integers we avoid further complications from negative values. The most important local variable of the algorithm is the two-dimensional array "m". Its first dimension is visualizing the number of items and its second the number of weights. Overall each slot of the array would be interpreted as the "maximum value that the knapsack can hold given [i][j] where i and j are two counters denoting the current value of each dimension mentioned above". By doing that the slot m[n][w] contains the solution of the problem. The optimal substructure can be identified between lines 6 and 10. In these lines lies the implementation of the mathematical type of the unbounded knapsack problem. The recursive calls of the previous values of the "m" array can also be identified in these lines. This algorithm like all dynamic algorithms makes effective use of the bottom-up implementation.

The double iterations on items and weights check if the weight of the item equal to the counter of the first iteration is bigger than the one of the second. That would be interpreted as "does the current item weight more than the number of kilograms that the knapsack can hold" were the number of kilograms denotes the current value of second dimension of the "m" array. If the answer to the above question is "yes" then the item may be included in the knapsack provided that its value contributes to a larger overall value in the knapsack as compared with the already stored values. If not then it is simply not included. The decision of accepting the item in the knapsack for a given weight at a time depends on whether the overall value before its acceptance and after is bigger than the one that is already calculated as optimal by the previous iterations. The "m" array is populated by the using the mathematic formula below:

```
M[i][j] = 0 if j  = 0
M[i][j] = m[i-1, j] ,  j < w[i]
M[i][j] = max(m[i-1, j], m[i-1, j-w[i]] +  v[i]) , else
```

Equation 1 : Mathematical type for populating the knapsack table

What was described in the latter paragraph is one subproblem of the unbounded knapsack problem. The overlapping subproblems are of the exact same description with the values and weights differing accordingly. The concept of execution of this algorithm is that is

fills every slot of table “m” beginning from the [0][0] position and continues on by constantly accessing the previous slots in order to determine the optimal solution each time. The initial conditions of the problem are implemented in the first “for loop” and they are critical for the terminal state of the algorithm. During the initialization phase of the algorithm the first row of the “m” array is populated with consecutive zero or “null” values because it is obvious that with capacity from “0” to “W” units of weight and with no items available the total value of the knapsack would be zero. The counters of the loops can also be initialized in the reverse order. The slots of the array would then be populated in a reverse order and the solution of the problem would be stored on the [0][W] position of the “m” array. In terms of time complexity we make the following computations:

$$
\begin{aligned}
\text{Complexity} &= \sum_{i=0}^{n} \sum_{j=1}^{W} O(1) \;+\; \sum_{i=0}^{n} O(1) \\
&= \sum_{i=0}^{n} \sum_{j=1}^{W} O(1) \;+\; O(1) \\
&= \sum_{i=0}^{n} W \;+\; O(1) \\
&= O(n * W) \;+\; O(1) \\
&= O(n * W)
\end{aligned}
$$

Equation 2 : Calculating the complexity of the dynamic knapsack algorithm

Equation 2 proves that the complexity of the dynamic algorithm for the knapsack problem is O(nW) . The costs of the two consecutive loops make the most it. In the calculation of the complexity we considered as O(1) the cost of an assignment. In this algorithm the complexity is called pseudo-polynomial because even though it is actually polynomial in numeric values, it is exponential in the length of the input.

The knapsack problem has many applications on many fields. It is quite common on mathematics, economics and business because it arises whenever there is a problem of limited resources and optimal distribution of them based on the optimal gain measured in some way. The same principles apply also in the financial world in cases of optimal return of multiple investments[11]. The knapsack problem was also the basis for early encryption algorithms[10] . If someone had to describe the knapsack problem in one phrase, maybe the most suitable one would be the famous business phrase “Value for money” denoting that based on a given capability one must make the optimal gain. Generally speaking the knapsack problem can serve as guidebook for every resources allocation problem.

1.4 The Shortest Route Problem

The shortest route problem is another typical problem of combinatorial optimization. The problem is defined as follows: in a graph with weighted edges the shortest possible path must be found between a specific initial node "A" and a final node "Z". By using the term shortest, in this study we imply the minimum possible weight defined as the sum of the weights of the edges used by following the route from "A" node to "Z" node. The shortest route problem has many variations much like the knapsack problem. The most popular versions of this problem are the ones with directed graphs, undirected graphs, cyclical and non-cyclical graphs as well as the one with negative values. Although the various versions may differ regarding the type of the graph, in every case the problem stays always the same. The shortest route problem can be solved by many methods, again like the knapsack, such as brute force, exhaustive search and others. The historical background of the shortest route problem in terms of mathematical analysis differed significantly from other operational research problems[12] due to the fact that it did not evolve as quickly as other problems of this field. There are many algorithms that solve this problem and the vast majority of them are considered greedy, not dynamic. The main difference between the two of them is that greedy algorithms do use the trait of "memorization", which is another word for the usage of array data structures to store solutions of overlapping subproblems. The most famous and effective algorithms that solve this problem are the "Dijkstra Algorithm", the "Bellman-Ford algorithm", the Kruskal algorithm and last but not least the algorithm made by Prim. All of them have advantages and disadvantages with regards to time complexity.

Dijkstra's algorithm solves a shortest path problem with nonnegative edge weights. The algorithm can be described as follows: for every node of the graph, two numbers are stored in a array data structure. The first is the length of the shortest path from the initial node to current found so far, the second the node preceding that was previously accessed by following that path. The algorithm will construct better paths in an iterative way, improving the solution in each step. On termination, the algorithm must have found the shortest path from the initial node to all other nodes in a graph[14].

```
1.Dijkstra's_Algorithm_Shorstest_Path
2.begin
2.   while(F is missing a vertex)
3.      for each edge of v, (v1, v2)
4.        if(dist[v1] + length(v1, v2) < dist[v2])
5.            dist[v2] = dist[v1] + length(v1, v2)
6.            prev[v2] = v1
7.        end if
8.      end for
9.    end while
10.end
```

Algorithm 4: Dijkstra's shortest route problem

Algorithm 4 describes the dijkstra's method for solving efficiently the shortest route problem given the following data:

dist : array of distances from the source to each edge

prev : array of pointers to preceding nodes

i : loop counter

F : list of finished nodes

U : list or heap unfinished nodes

Bellman's algorithm is also one of the most important algorithms that solve the shortest route problem. Bellman and Ford used a slightly different approach for designing the algorithm. It is actually slower from the Dijkstra's algorithm for computing the solution but has the advantage that can deal negative weights on some edges, a feature that the latter does not have. Bellman's algorithm is more of a dynamic nature than greedy. It uses the same principle of optimality was mentioned in chapter "Dynamic Programming" of this study. Like other Dynamic Programming Problems like the example described in the chapter "design of dynamic algorithms" and the one of Knapsack problem , the algorithm calculate shortest paths in bottom-up manner. It calculates the shortest distances for the shortest paths which have at-most one edge in the path. Then, it calculates shortest paths with at-most 2 edges, and so on. After the ith iteration of outer loop, the shortest paths with at most i edges are calculated. There can be maximum V – 1 edges in any simple path, that is why the outer loop runs |v| – 1 times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most (i+1) edges[14].

```
1.bellmanFord(G, s)
2.    for all edges in G(V)
3.        D(V) = INT_MAX
4.       parent[V] = -1
5.    D(s) = 0
6.    for i = 1 to |G(V)| - 1
7.        for each edge (u, v) in G(E)
8.            if edge can be Relaxed
9.                D(v) = D(u) + weight of edge (u, v)
10.                parent[v] = u
11.           end if
12.        end for
13.    end for
14.    for each edge in G(E)
15.       if edge can be Relaxed
16.            return false
17.       end if
18.    end for
19.  return true
17. end
```

Algorithm 5: Bellman Ford algorithm for finding shortest path

2. Dynamic Programming Problems Solver

2.1. General

In the second part of this study we present a software application that was developed in order to solve basic dynamic programming problems. The application named "Dynamic Programming Problems solver" was developed with the object oriented programming language java and solves two problems that were mentioned and discussed in the first part of this study; the knapsack problem and the shortest Route problem. This application can be divided in two software units. The unit "GUI" which is mainly responsible for the human-machine interaction and contains the graphic environment and the unit "Engine" which is responsible for actually solving each problem. To put in strictly software development terms the first unit implements the "front end" component of the program and the second one the "back end"component. The general characteristics of this application can be summarized in table 1.

Operating System (O.S.)	Windows 7 , 64 bit
Programming Language	Java Development Kit, Oracle
Compiler	Javac, Oracle
IDE	Netbeans version 8.2
Graphics Design	Netbeans, jframe java library
Graphs design	Graphstream java library

Table 1 : General characteristics and dependencies

Each software unit can be divided in classes and routines. The general concept of execution is that upon clicking a button on the Human Machine Interface (HMI) a respective action will be triggered. After validation of this action either the HMI will keep asking for further data from the end user in order to populate various fields of the problem that needs to be solved, or error messages shall make their appearance on the screen, informing the end user about the respective error. In case that every field was populated in a valid way and result buttons are clicked then the solution shall appear on the screen. It is important to note that the end user has the complete control of the application in the terms of executing various actions, returning to previous menus, initializing and terminating the application. The end user has the capability to navigate through the menus of the application changing data for various fields, and computing a problem several times.

In order to compile and run the application, the installation of java[17] and java development kit is required. The application can also be compiled and run through an IDE e.x. Netbeans[16] or Eclipse. In any case, for successful compilation of the project the installation of graphstream library[15] is also needed. The application can also be executed in three ways. The first way is directly through an IDE, the second by simple double clicking the "jar" file which is the executable and the third is from command line by executing the following command:"java-jar "<Installation_Folder>\dist\Dynamic_Programming.jar"" . Below there is an example of command line execution.

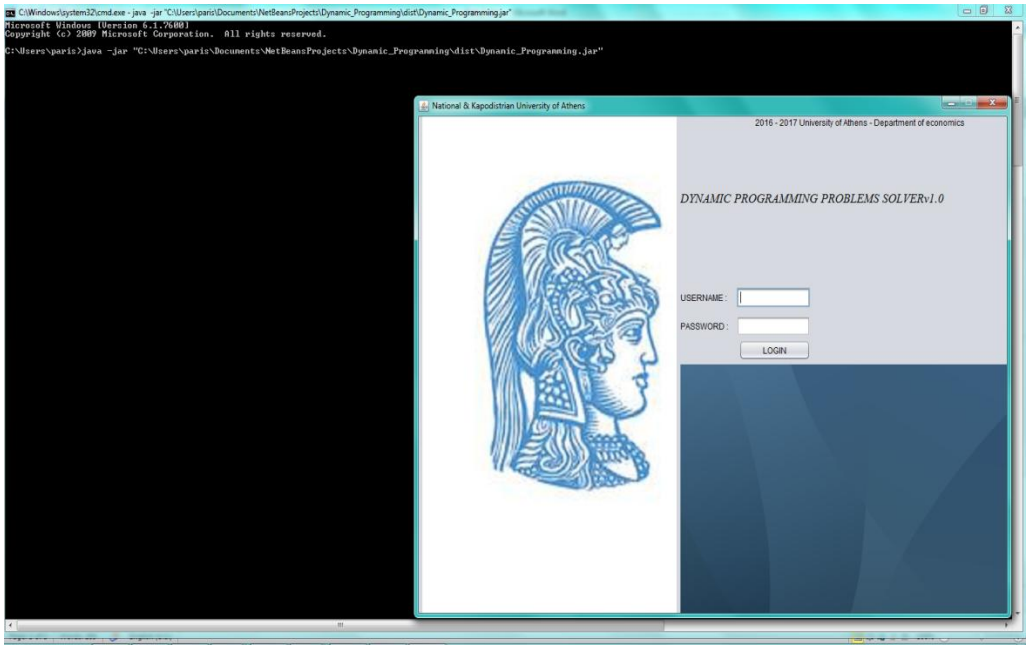

Figure 1: Example of command line execution

2.2. GUI Software Unit

The first unit of this program is named "GUI" and implements the graphics of the program as was mentioned above. The main features that all the panels of this unit have are documented below:

- The left part of the panel is occupied by the logo of the National & Kapodistrian University of Athens.
- The right part is occupied by the actions that can be performed in each panel.
- Data entry begins from the upper stages of the panel and continues downwards until the solution is described on the bottom-right part of the panel
- Upon clicking a button on a panel that triggers the appearance of another panel the panel on the screen, the old panel is set to non-visible mode.
- Upon clicking the window exit button on the upper-right position of the panel the application terminates its execution and discards all data.
- None of the panels is resizable. That means that the actual size of all panels cannot be changed with the exception of minimization. Its dimensions shall stay the same throughout the execution of the program.
- The content of each panel is not editable by the end user of the program with the exception of the various fields that designate to the end user to edit them in order to be used as input for the program.

The "GUI" software unit comprises of several subunits or classes as they are named in java, each representing a different stage of the HMI. These subunits are the "Login_Screen" subunit, the "Main_Menu" subunit, the "Knapsack Problem" subunit and the "Shortest Route Problem" and they are documented below.

2.2.1. GUI.Login Screen

Upon execution of the application by the end user, the first feature that appears on the computer screen is the "Login Screen". The main functionality of this panel can easily be deducted from its name. The end user must populate two fields, namely the "username" field and the password field. Then the only action that can be performed is the pressing of the "login" button. By clicking this button two actions can be triggered. If both the username and the password are correct, then the flow shall reach another activity, that of "main menu", otherwise a pop-up panel shall inform the end user that an error occurred during the validation of the values contained on the two fields.

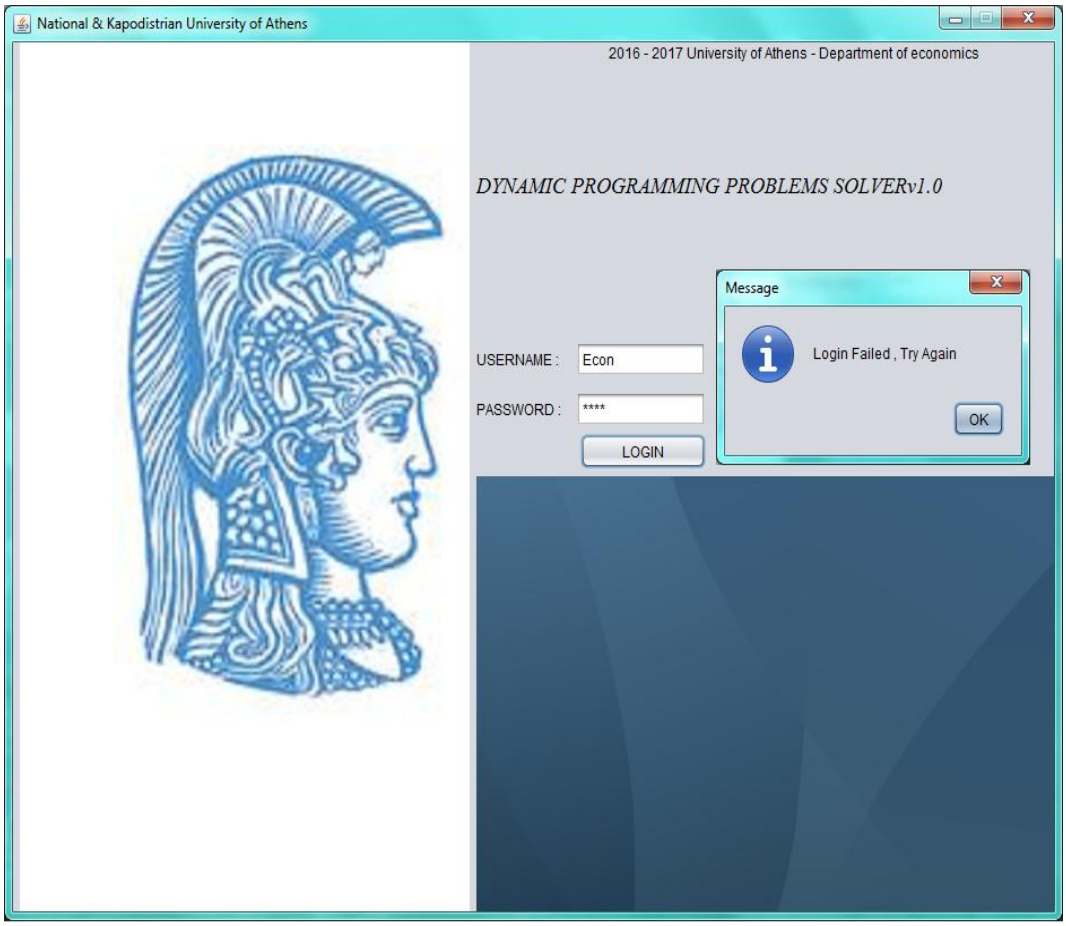

Figure 2:Login Error

The situation described above is a simple comparing of two strings and there is no need to provide an algorithm for that. The complete java code for the login_screen.java file can be found in ANNEX B.2.1 of this study.

2.2.2. GUI.Main_Menu

Upon successful execution of the login function the flow reaches the "Main menu" class. This class has several activities. On this panel the end user has three options. The first one is to terminate the program. The other one is to choose between the clicking of two buttons the

"Knapsack Problem" button or the "Shortest Route Problem" button. Each triggers a new panel to appear on the screen with the description of the respective problem.

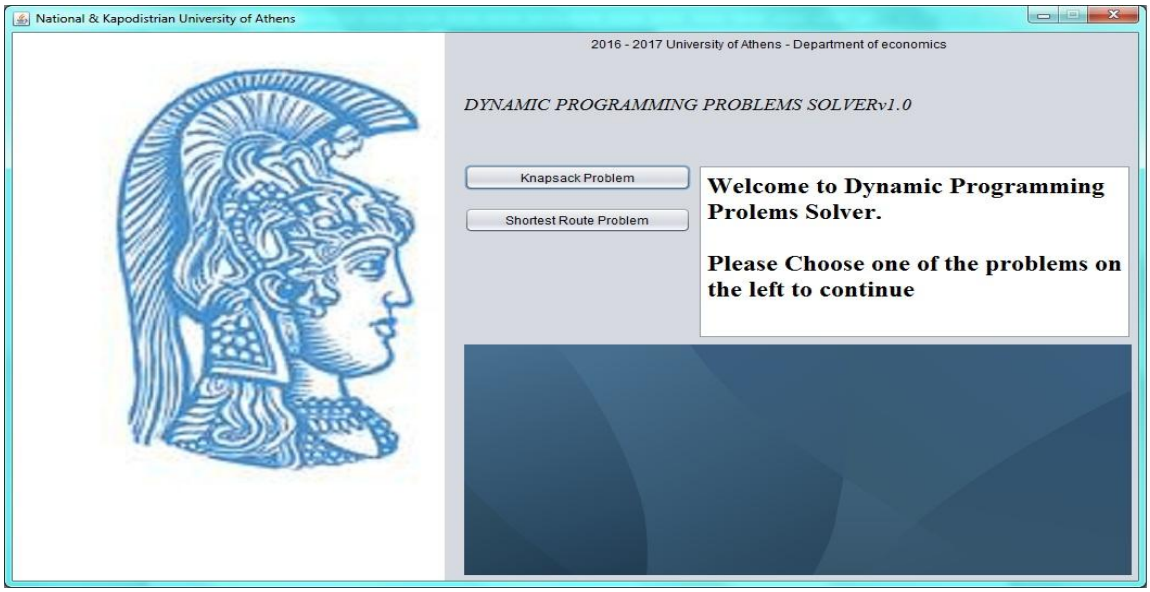

Figure 3: Main Menu

The complete java code for the Main_Menu.java file can be found in ANNEX B.2.2 of this study.

2.2.2. GUI.Knapsack_Problem

This subunit is the menu of the knapsack Problem. In this stage the end user has to populate two fields. The number of the knapsack can hold and the number of the Items that are available for input on the Knapsack. It is a design decision to limit the second number to ten items for practical purposes. By doing that, the end user has not the option of solving the knapsack problem for more than ten "Items". During the first stage of the data entry the end user has three main options. Either to press the "back to main menu button" which will lead him to the previous activity, to press the "Reset all fields" button which will erase any data input on the two fields and finally to press "continue" button. By opting for the latter, the second stage of the activity shall be initiated provided that the data input are valid. Each action described above is a local routine of this class. The pseudo-code for the continue action is described below:

```
1.  Continue_Action(Items, Kilos)
2.  Begin
3.    If Items > 10 or Items < 2
4.      Show Message("Wrong Input of Items" )
5.    Else if Kilos > 100 or Kilos < 1
6.       Show Message("Wrong Input of Items" )
7.    Else
8.        Proceed to stage_2 of Knapsack data
   entry
9.    End if
10.     End
```

Algorithm 6: Continue Action

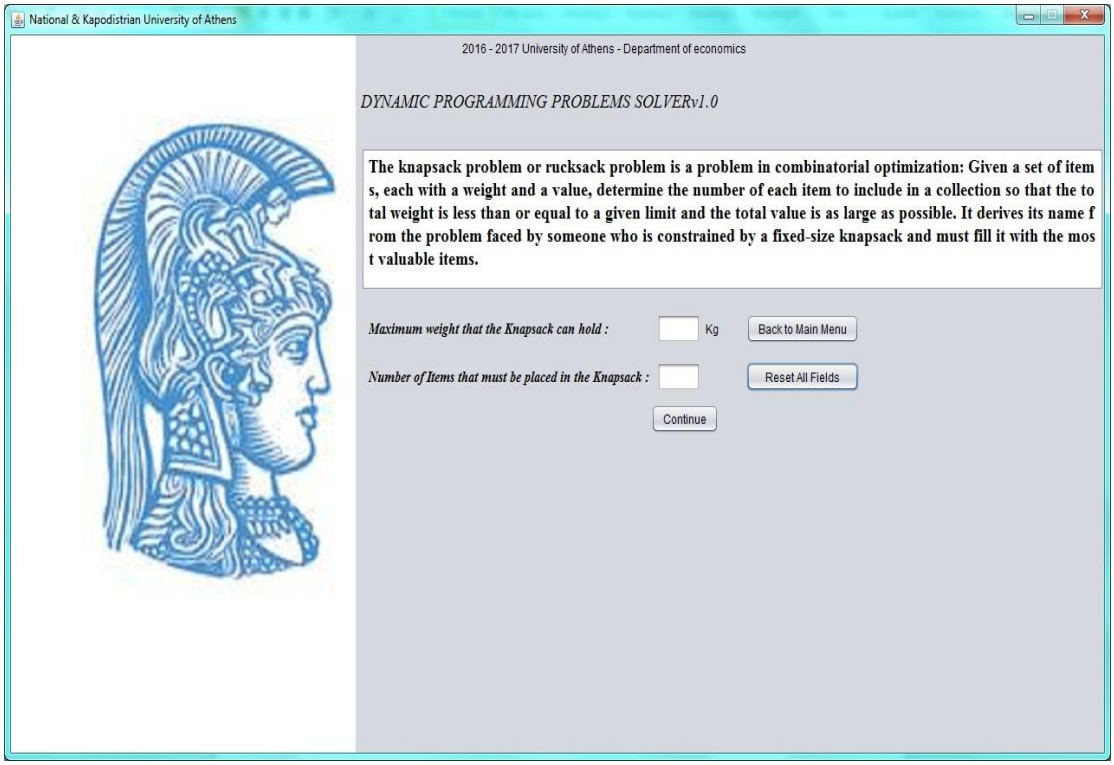

Figure 4: First stage of the Knapsack Data Entry

If the above algorithm terminates using the third option, the second stage of the knapsack problem data entry arises. Upon initialization of this stage a table emerges on the bottom of the panel and the "continue" button becomes non-clickable. The rows and columns of this table are parameterized in accordance with the values of the fields populated during the previous stage. The numbers of rows of the array represent the number of Items that the end users choose for the Knapsack. The columns represent different things each. The column named "Values" represent the value per unit of an Item as described in part one 1.3 section "The knapsack problem" and the column "Weights" represent the weight per Unit of an Item.

During this stage the end user still retains the options of returning to the main menu and resetting all fields but has another one, to press Button "Solve". Upon pressing that button the program shall proceed to the validation of all cells of the table. There are two requirements in order for that validation to work. The first is that all cells of the table must be populated and the second that all cells must contain numbers only. If at least one of these two conditions are not met the applications shall inform the end user that an error has occurred. If both conditions are met then the application shall proceed to the processing of the data and the solution of the Unbounded Knapsack problem where the data input in the table mentioned above and the values populated during the previous stage contain all information needed. The data flow goes to the "Engine" software unit, Knapsack subunit with input all the above data.

Upon successful processing by the "Engine" of the above data the solution of the Knapsack problem for the data given shall appear on the bottom-right position of the panel on another table which shall contain the exact solution of the problem as per Items, Units, Weights and the Maximum Possible Value that can be achieved.

The display of the second stage for a given example input of twenty kilos capacity and ten Items, of the Knapsack data entry can be seen in the figure below:

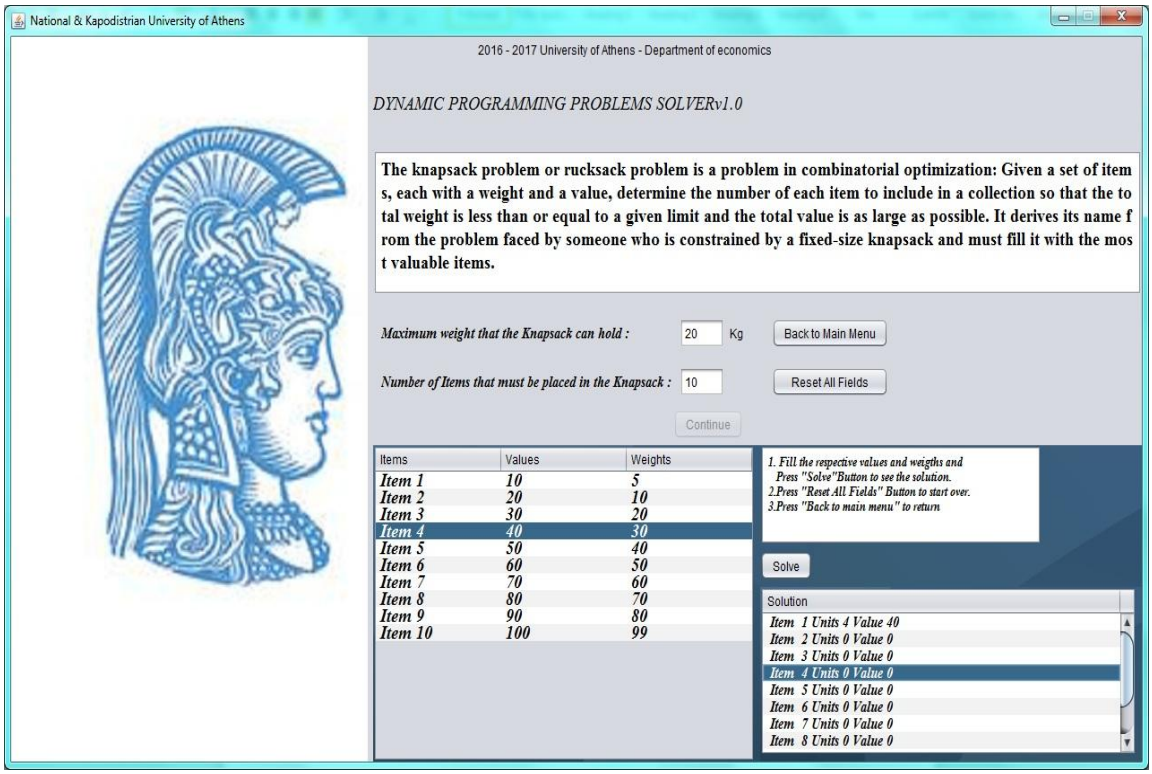

Figure 5: Second Stage of Knapsack Problem data entry and solution given

The complete java code for the Knapsack_Problem.java file can be found in ANNEX B.2.3 of this study.

2.2.3. GUI.Shortest_Route

This subunit implements the menu panel of the Shortest Route problem. During the first stage of the data entry the end user is again presented with three different options. To return on the main menu screen by pressing the “Return to main menu” button, to clear the values of every fields that is already populated by pressing the “Reset all fields” button and finally to press the “Continue” button. During this stage the only field that the end user has to insert a value is the “Number of Layers” field. The respective value shall represent the total number of layers that the graph which will be created shall have. We use the term “layers as in Bellman algorithm of finding the shortest path. For example “Layer 1” represents all routes from the initial node of the graph that has used one edge. For “Layer 2” we imply routes of two edges and so on. Upon pressing the “continue” button, the only restriction applied is that the value of the field “Number of Layers” must be a positive value. If this is not the case the application shall inform the end user with a pop-up panel of the respective error. In any other case the flow shall proceed to next stage. It is also important to note that there is no upper-bound restriction for the number of layers that the end user can feed the application with. This means that the shortest Route problem can be solved for any given input concerning number of layers.

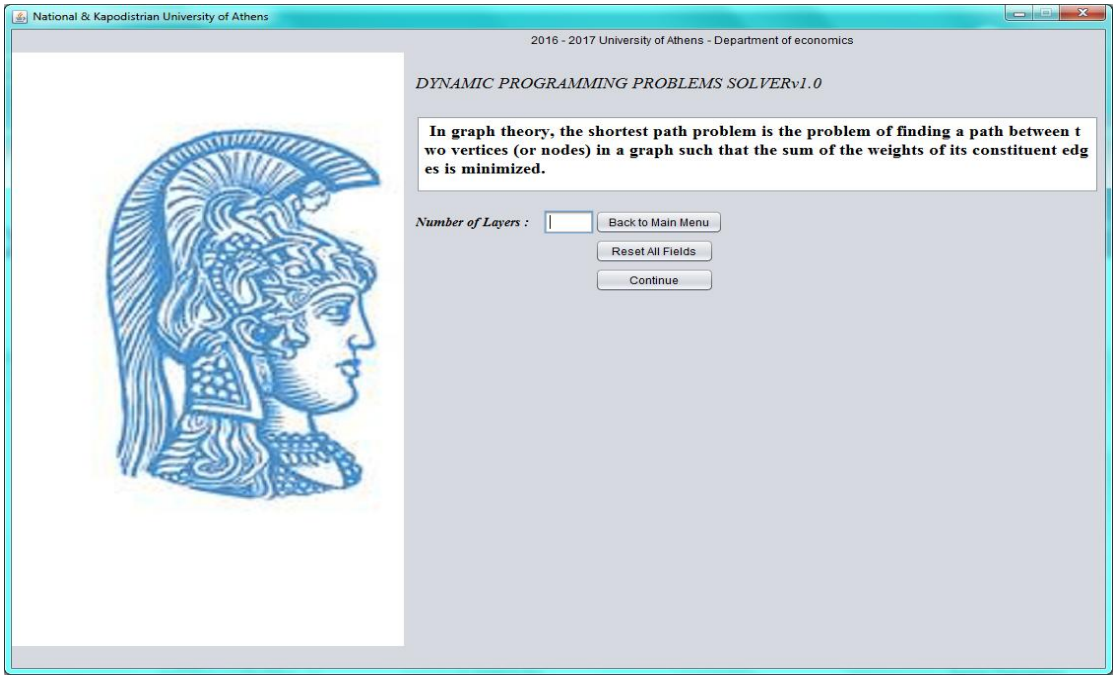

Figure 6 : First stage of Shortest Route problem menu

Upon successful execution of the first stage a table shall emerge in bottom center part of the panel. This table shall "Number of Layers + 1" rows and two columns. The right column only may be edited by the end user and it represents the number of nodes that each layer have. The end user will also notice that the first and last cells of the left column will be populated upon initialization with the value "1" and cannot be edited. This feature represents the initial and final stage of the Shortest Route problem, meaning that a graph on we which the shortest path is unknown can have only one initial node and only one final node. During this stage the end user still retains all the actions that could perform on the previous stage plus one. The latter is to press the "Next" button which will trigger the initialization of the third stage provided that the table has been populated correctly (with non-negative values). Again there is no upper bound regarding the values of the first table.

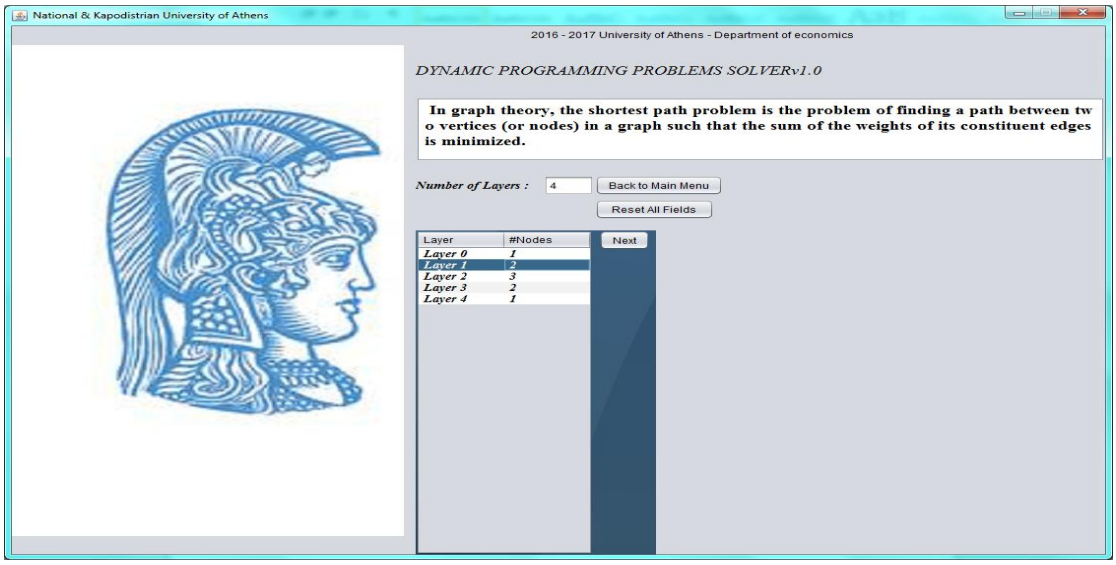

Figure 7: Second stage of the Shortest Route problem menu

Upon successful execution of the second stage a second table shall just right of the first one. This table can be edited from the end user in order to provide the graph the respective weights of every edge of the graph. On the latter table there is absolutely no restriction regarding the values of the cells. The program shall not create an edge between tow nodes if the respective edge has zero value or if it is not populated at all. Upon clicking the "Create graph" button the graphical representation of the graph shall emerge in a separate panel. The creation of the graph was implemented by using the graphstream java library from org.graphstream[17] .

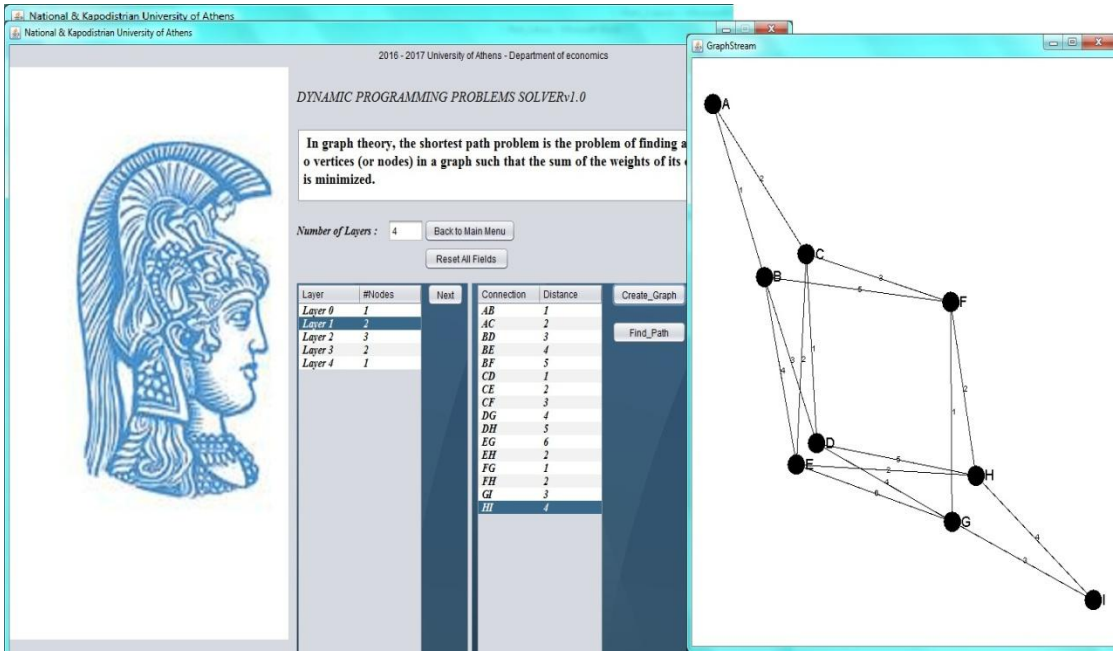

Figure 8: Third stage of Shortest Route problem menu

Finally the end user has to press the "Find Path" button. Upon pressing it, the flow of the program goes to the "Engine" software unit that implements the algorithm of finding the shortest route utilizing all the above data. When it returns the results two different things emerge on the screen. First, a third table makes its appearance on the far right of the screen containing the results of the problem. It shall inform the end user about the edges that make up the shortest path and the total weight of the path. In addition another panel shall pop-up containing another graphical representation of the graph, with one difference from the previous one. It shall indicate the shortest route by designating the edges that make up the shortest route with red color. The two panels can be compared as the first one shall be in visible mode. Throughout the process described above the end user has the capability to stop in whatever stage chooses, or restart the process using different data. The two figures below present an example of successful completion of the Shortest Route problem for a random input.

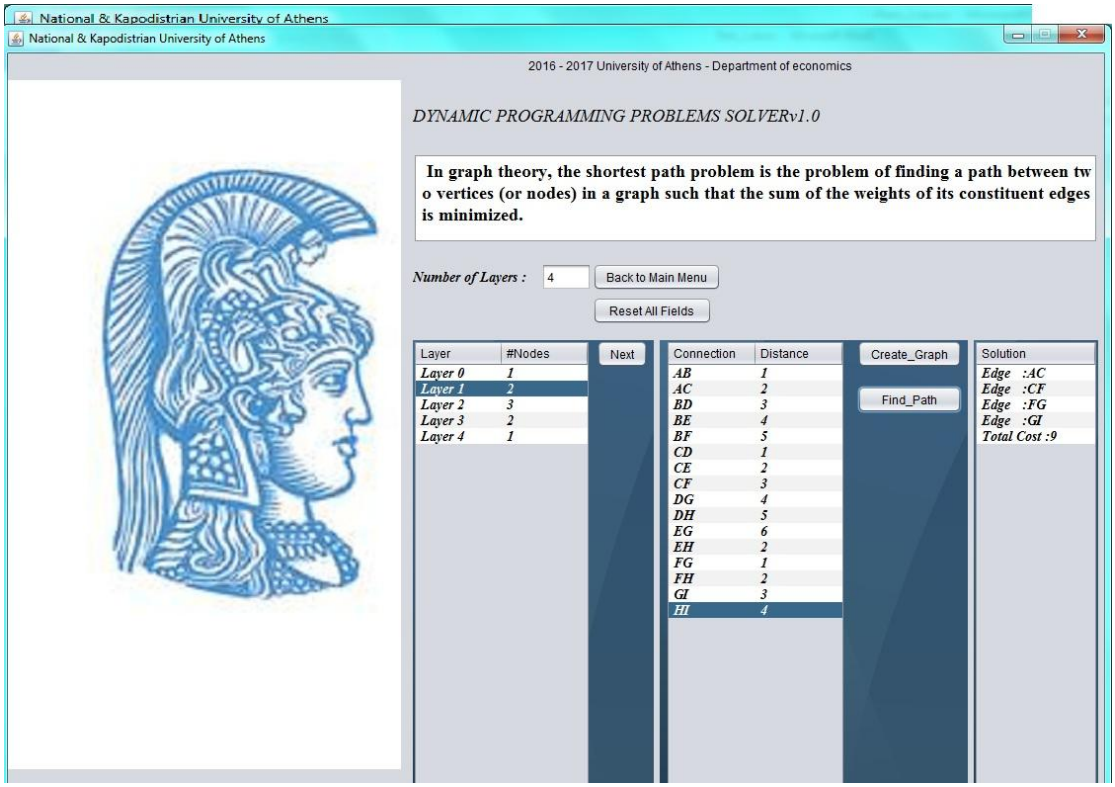

Figure 9: Final stage of the Shortest Route problem menu

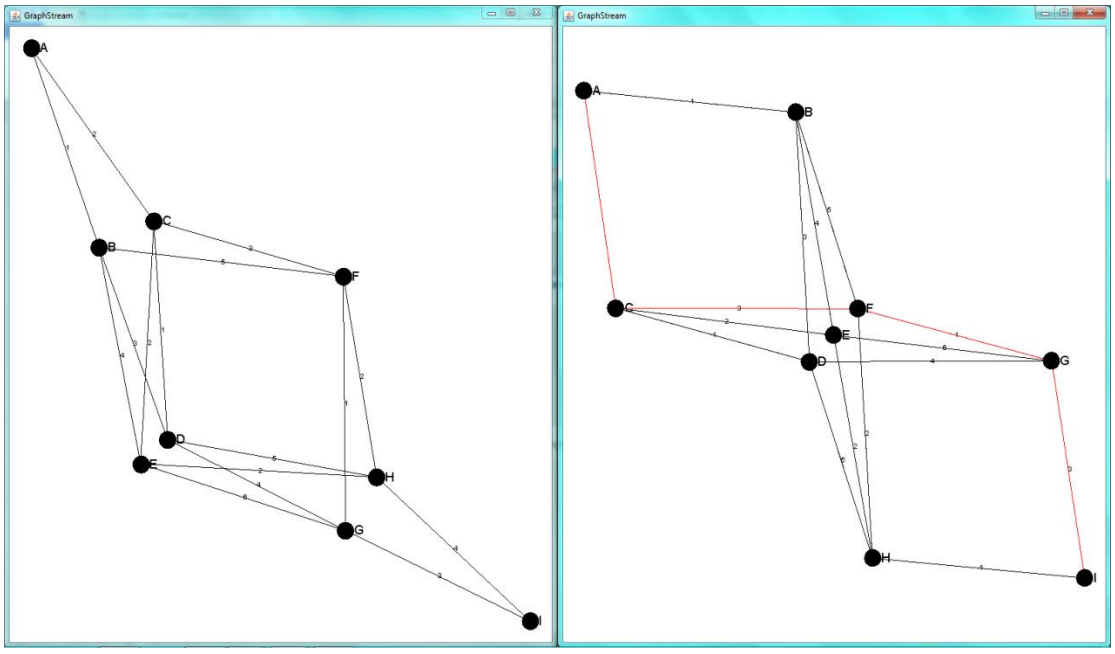

Figure 10: Comparing the graphs

The complete java code for the Shortest-Route.java file can be found in ANNEX B.2.4 of this study.

2.3. Engine Software Unit

The second software unit of the application is processes all data given as input from the respective menu and computes them in order to return the solution back to the HMI. This unit contains no graphic environment. It is composed from three files:

- The Knapsack file which solves the unbounded knapsack problem
- The Diophantine Equations file which extends the knapsack file in order to provide with utility solutions on Diophantine equations
- The shortest route file which solves the homonymous problem

2.3.1. Engine.Knapsack

The Knapsack file implements the unbounded knapsack problem. It is comprised from two procedures the "Solve Knapsack" procedure and the "Determine Max Element" procedure. The first one is the basic function for computing the maximum value that the knapsack can hold. In this study we calculate the above value by implementing a slightly different version of the knapsack problem but the basis of the algorithm is the one described in the first part of this study. The algorithm that we implemented is the following:

```
1. Solve_Knapsack(Maximum_Load,Number_of_Items,Data_Table)
2. Begin
3.   Populate Values[]  from Data_Table
4.   Populate Weights[] from Data_Table
5.   For units = 0 units < Maximum_Load units++
6.     F[last_item][unit] = Values[last_item] *
7.                  units/Weights[last_item]
8.   End for
9.   For Item = last_item item > 0 item = item -1
10.    For units = 0 units < Maximum_Load units++
11.      For x = 0 x <= units/Weight[Item] x++
12.        Temp_array[x] = Values[Item] * x +
13.                    F[Item+1][units - Weigths[Item]*x]
14.      End for
15.      F[Item][units] =Determine_Max_Element(Temp_array)
16.    End for
17.  End for
18.  Solve_Diophantine_Equation(Values,
19.                              Optimal_Value,
20.                              Weights)
21.  Return Result
22.  End
```

Algorithm 7: Solve Knapsack Algorithm

The concept of execution of the above algorithm is the following: First of all the values and weights of the items are extracted from the data table which was given as input from the GUI. knapsack file. Then the implementation of the knapsack algorithm initiates. This algorithm makes use of an array which is has "Number of Items" +1 rows and "Maximum load" +1 columns. This array named "F " keeps stored in each cell the maximum value that the knapsack can hold provided that we have in our disposal "i" Items and "j" units of weight .The initialization process takes place between the lines 5 and 8. The algorithm sets the last row of the array to the optimal value given the respective disposed units. Then backward induction initiates. For every item and for every unit of weight up to the maximum weight that the knapsack can hold, the algorithm computes the value by adding the value of the current item with the optimal value of the previous item already stored in the array, stores it on a temporary array and then compares all the cells of the temporary array. The result from that computation delivers the current optimal value that the knapsack can hold and the process continues on to the point that the algorithm has populated all the cell of the "F" array.

In order to determine the maximum value of the temporary array during each loop we use the typical max – min algorithm as follows:

```
1. Determine_Max_Element(Array  A)
2. Let the first entry to be the
   maximum
3.   For I = 1 I <= A.length i++
4.     If A[I] > max
5.       Max = A[i]
6.     End if
7.   End for
8. Return max
9. End
```

Algorithm 8: Find max element algorithm

Up to that point the "Solve Knapsack" algorithm will have given the answer to the following question: "What is the optimal value that can be contained on the knapsack given the data contained in data_table?" . It still remains to be clarified what is the exact composition of the solution. In other words the algorithm has to find the specific number of copies that were used in order to come up with the above solution. In mathematical terms the question would be the following linear Diophantine equation:

```
a1*x1 + a2*x2 + ... + an*xn = C

find xi  , i = [1,n]

where ai , i = [1,n] are Known
```

Equation 3: Linear Diophantine equation

In order to solve the above equation this unit contains the file Diophantine_Equations which has all the implementations for a specific number of items. It is a design decision to

include ten routines in this file, each solving the above equation for n = the respective number of items and not one because then we would have to compute it in a recursive way which would be catastrophic in terms of time complexity. Instead, we choose to solve the equation in an iterative manner by using the algorithm below which is a typical representation for all ten procedures:

```
1. Backtrack_Solution(C, Array A, Array B,max_load)
2. Begin
3.       For I = 0 and  I * A[0] <= C I++
4.         For J=0 and I*A[0]+J*A[1]<=C J++
5.           For K=0 and I*A[0]+J*A[1]+K*A[2]<=C K++
6.             If i*B[0]+j*B[1]+K*B[2]<=max_load and
7.                I*A[0]+J*A[1]+K*A[2]<=C
8.
9.                   Weight_Per_Item[0] = i
10.                 Weight_Per_Item[0]  = j
11.                 Weight_Per_Item[0]  = k
12.
13.               End if
14.          End for
15.        End for
16.      End for
17.      Return Weight_Per_Item
18.      End
```

Algorithm 9: Solving a linear Diophantine Equation for n = 3

The above algorithm works for n = 3 items. By augmenting the number of for loops and the size of all the arrays respectively then the algorithm can compute any given number of items. Hence the result is returned on the GUI.Knapsack from which the procedures of this file were called and portrayed on the panel. The complete java code for the Engine.Knapsack file can be found on ANNEX A.1.1 and the complete code for the Engine. Diophantine_Equations file can be found on ANNEX A.1.3 of this study.

2.3.2. Engine.Shortest_Route

The shortest Route file of the Engine unit implements the algorithm of finding the shortest path between an initial node let “A” and a final node let ”Z”. In this study we have implemented a version of the Bellman Ford algorithm described in part one and we have made the following assumptions:

- The graph shall be divided in layers. Each layer shall represent the number of edges used to reach a specific node from the initial node.
- There will be no cyclical paths on the graph.
- The nodes of each layer may be connected with the nodes of the next layer or not but they cannot be connected with the nodes of any other layer.
- There are no restrictions on the number of nodes of any layer with the exception of the first and the last that by definition must contain one node each.

This file contains two routines one for finding the path and one for computing the minimum element of an array of integers. The primary procedure that gives the solution is the first one and its algorithm is described below:

```
1. Find_Path(Layers, Array Edges, Array Nodes_Per_Payer)
2. Begin
3.   For each layer beginning from layer 0
4.     For each node of current_layer
5.       For each node of current_layer+1
6.         A[layer][node of current_layer][node of
7.                                  current_layer+1]
8.         = Edges[nodes,weight]
9.         Mapped_A[layer][node of current_layer][node
10.                                       of
11.                                       current_layer+1]
12.        = Edges[nodes,Name]
13.
14.
15.        End for
16.      End for
17.      F[Layers][0] = 0
18.      For each layer beginning from the last
19.        For each node in current layer
20.          F[Layer][node] =
21.          Determine Min Element(A[layer][node],
22.                                F[previous_layer],
23.                                Layer)
24.        End for
25.      End for
26.      End for
27.      Result = F[0][0]
28.      Return Result
29.      End
```

Algorithm 10: Find shortest path algorithm

The above algorithm makes extensive use of Bellman's principle of optimality. The most important clarifications that must be made in order to understand the above algorithm are the role of the three arrays. The two dimensional "F" array is the main array that keeps stored in each cell the shortest path value between the target (final) node and the F[i][j] node where "i" denotes the layer and "j" denotes the index of the node in its layer. The "A" array is a three-dimensional array that keeps stored the weight of an edge in the cell A[i][j][w] where "i" denotes the layer , "j" denotes the index of a node in layer, representing the beginning of the edge and "w" denotes the index of the edge of the next layer representing the index of the node to which the edge is directed. The "Mapped_A" array is identical to "A" but keeps stored the name of the edge for example "AB" denoting that edge connects node "A" with node "B". By using the backward induction we compute the minimal weight path for each node of every layer to the target node. Hence each layer has a node for which the path towards the target node is the shortest. By combining the above with the initial state of F[number of layers][0] = 0 , meaning that the distance between the target node from itself is

zero, the algorithm is able to compute the shortest route from the initial node to the target node. For determining the minimum value between an array of Integers we again use the classical max-min algorithm, as in the knapsack problem with the necessary modifications. The algorithm is described below:

```
1. Determine_Min_Element(Array  A)
2. Let the first entry to be the minimum
3.   For I = 1 I <= A.length i++
4.     If A[I] < max
5.       Min = A[i]
6.     End if
7.   End for
8. Return min
9. End
```

Algorithm 11: Determine min element in an array

The complete java code for the Engine.Shortest_Route file can be found on ANNEX A.1.2 of this study.

CONCLUSIONS

The first part of this study was a short background review of dynamic programming as general concept. It of great importance to whoever is willing to be involved with software engineering for quantitative fields, to be familiar with the basic principles of optimization and dynamic algorithms. In order to achieve that, in this study we presented some of the most famous dynamic algorithms such as the Fibonacci sequence algorithm and the dice throw algorithm. In the first part there is also a brief review of the knapsack problem and the shortest route problem. These problems are considered the milestone of their kind and therefore are the ones we chose to include in our application.

Having completed the first part, we proceed into the second one, where we apply theory into practice. We tried to develop a rather simple software application that is able to compute solutions to the two problems mentioned above. This part contains the detailed user interface for data entry, result presentation and navigation through the menus. Designing user interfaces is always proven to be a much more difficult job than it may seem. One has to take into account that the user may sometimes want to see a more detailed solution. In our case given the generic form of the problems we implemented, we decided that the user interface ought to be as much detailed as possible.

Finally the development of the code for the two problems is the core of the application. In the two annexes the complete code is given. Thought it may seem difficult to be read from someone without particular software engineering knowledge, if the reader has a good understanding of the dynamic algorithms it is more interesting than it may originally seem.

ANNEX A

A.1 PACKAGE ENGINE

A.1.1. FILE Knapsack.java

```
package Engine;

import java.util.Arrays;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;

/* @author Paris Tsampras */
public class Knapsack
{
   public static Integer Number_Of_Phases;
   public static Integer Maximum_Load;
   public static Integer[] Weigths_Per_Item;



   public static Integer Determine_Max_Element(Integer[] A  ,
                                    String Str   ,
                                    Integer Phase)
   {

      int max_element =0;
      if (Str.equals("Values"))
      {
         max_element = A[0];
         for (int index =0; index < A.length; index++)
            if (A[index] > max_element )
            {
               //Weigths_Per_Item[Phase] = index;
               max_element = A[index];

            }
         return max_element;
      }
      else
         return -1;

   }


   public static String[] Solve_Knapsack(DefaultTableModel Knapsack_Table)
   {
      System.out.println(Number_Of_Phases);
      System.out.println(Maximum_Load);
      Integer Number_of_Columns = Knapsack_Table.getColumnCount();
      Integer Number_of_Rows    = Knapsack_Table.getRowCount();
      Integer Remaining_Value   = 0;

      String[]  Result          = new String [Number_Of_Phases+1];
```

```
Integer[] Values          = new Integer[Number_Of_Phases];
Integer[] Weigths         = new Integer[Number_Of_Phases];
    Integer[][] F           = new      Integer[Number_Of_Phases][Maximum_Load+1];

for(int index = 0 ; index < Number_Of_Phases; index++ )
   Values[index]  = Integer.parseInt(
                String.valueOf(
                Knapsack_Table.getValueAt(index, 1)));
for(int index = 0 ; index < Number_Of_Phases; index++ )
   Weigths[index] = Integer.parseInt(
                String.valueOf(
                Knapsack_Table.getValueAt(index, 2)));
for(int Item = 0 ; Item < Number_Of_Phases; Item++ )
   Weigths_Per_Item[Item] = 0;


// KnapSack Algorithm

// Initialization of last phase
for (int units = 0; units <= Maximum_Load; units++  )
{
   F[Number_Of_Phases-1][units] = Values[Number_Of_Phases-1]*
                          (int)(units/Weigths[Number_Of_Phases-1]);

}
// end of initialization

// Bottom-up implementation of knapasck algorithm
// We begin from the last phase of the problem
// representing the Item which is located on the
// last row of Knapsack_Table populated by the
// previous activity
for(int Item = Number_Of_Phases-2; Item >= 0; Item-- )
{
   for (int units = 0; units <= Maximum_Load; units++  )
   {
      Integer [] temp_array = new Integer[(int)(units/Weigths[Item]) + 1];
      for(int x= 0; x <= (int)units/Weigths[Item]; x++  )
         temp_array[x] = Values[Item]*x +
                     F[Item+1][units - Weigths[Item]*x];
      F[Item][units]    = Determine_Max_Element(temp_array,"Values",Item);
   }

}
// End of KnapSack Algorithm

//print the results
for(int Item = 0 ; Item < Number_Of_Phases; Item++ )
{

   for (int units = 0; units <= Maximum_Load; units++  )
   {
       System.out.printf("%3d ",F[Item][units]);
   }
   System.out.println();
```

```
        }
        // end of printing

        // backtracking the result per Items , Units , Values
        // diofintine equation a1*x1 + a2*x2 + ... + an*xn = C
        // where ai , i = [1,n] are Known
        // solved by brute force

        if (Number_Of_Phases == 3)
           Diophintine_Equations.BackTracking_3 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 4)
           Diophintine_Equations.BackTracking_4 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 5)
           Diophintine_Equations.BackTracking_5 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 6)
           Diophintine_Equations.BackTracking_6 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 7)
           Diophintine_Equations.BackTracking_7 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 8)
           Diophintine_Equations.BackTracking_8 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 9)
           Diophintine_Equations.BackTracking_9 (Values,F[0][Maximum_Load],Weigths);
        if (Number_Of_Phases == 10)
           Diophintine_Equations.BackTracking_10(Values,F[0][Maximum_Load],Weigths);
        // end of Back tracking



        // Assign the results at the returned variable
        // of the function
        for(int Item = 0 ; Item < Number_Of_Phases; Item++ )
        {
           System.out.println(" Item  "+(Item+1)+
                              " Units "+ Weigths_Per_Item[Item] +
                              " Value "+ Weigths_Per_Item[Item]*Values[Item]);
           Result[Item] = (" Item  "+(Item+1)+
                          " Units "+ Weigths_Per_Item[Item] +
                          " Value "+ Weigths_Per_Item[Item]*Values[Item]);
        }
        Result[Number_Of_Phases] = ("Maximum Value is  "+ F[0][Maximum_Load] );

        return Result;
     }

}
```

A.1.2. FILE Shortest_Route.java

```
package Engine;

/* @author Paris Tsampras */
public class Shortest_Route
{
```

```
public static Integer Number_Of_Layers;
public static Integer Used_Edge_Index[];


public static Integer Determine_Min_Element(Integer[] A,
                                            Integer[] B,
                                            Integer Phase)
{
   int min_element =0;
   if (A[0] != null)
   {
      min_element = A[0]+B[0];
      Used_Edge_Index[Phase+1] = 0;
      for (int index =0; index < A.length; index++)
         if (A[index] != null &&
            B[index] != null)
         if (A[index]+B[index] < min_element )
         {
            min_element = A[index] + B[index];
            Used_Edge_Index[Phase+1] = index;
         }
      return min_element;
   }
   else
      return B[0];
}

public static String[] Find_Path(javax.swing.JTable Nodes_Per_Layer,
                         javax.swing.JTable Edges )
{
   String [] Result      = new String [Number_Of_Layers+1];
   Integer[][]   F       = new Integer[Number_Of_Layers+1]
                             [Number_Of_Layers+1];
   Integer[][][] A       = new Integer[Number_Of_Layers+1]
                             [Number_Of_Layers+1]
                             [Number_Of_Layers+1];
   String[][][] Mapped_A = new String [Number_Of_Layers+1]
                             [Number_Of_Layers+1]
                             [Number_Of_Layers+1];
   Integer Previous_Nodes_Count = 0 ;


   for(int index=0; index < Number_Of_Layers; index++)
   {
      //System.out.println(String.valueOf(
      //              Nodes_Per_Layer.getValueAt(index, 1)));
      for(int layer_nodes= 0;
            layer_nodes< Integer.parseInt(
                     String.valueOf(
                     Nodes_Per_Layer.getValueAt(index, 1)));
            layer_nodes++)
      {
         for(int nodes=0;
               nodes<  Integer.parseInt(
                     String.valueOf(
```

```
                    Nodes_Per_Layer.getValueAt(index+1, 1)));
                nodes++)
        {
            if(String.valueOf(
              Edges.getValueAt(nodes+Previous_Nodes_Count,1)).
              matches("[0-9]+"))
            {
                A[index][layer_nodes][nodes] =
                        Integer.parseInt(
                        String.valueOf(
                        Edges.getValueAt(nodes+Previous_Nodes_Count,1)));
                Mapped_A[index][layer_nodes][nodes] =
                        String.valueOf(
                        Edges.getValueAt(nodes+Previous_Nodes_Count,0));
            }
            else if (nodes == 0 )
                A[index][layer_nodes][nodes] = 100000 ;

        }
        Previous_Nodes_Count =Previous_Nodes_Count+ Integer.parseInt(String.valueOf(
                        Nodes_Per_Layer.getValueAt(index+1, 1)));
    }
}

// Printing 3D "A" table for debugging purposes
for(int i=0; i<= Number_Of_Layers; i++ )
{
    for(int j=0; j<= Number_Of_Layers; j++ )
    {
        for(int k=0; k<= Number_Of_Layers; k++ )
        {
            if (A[i][j][k] != null)
                System.out.printf("%2d ",A[i][j][k]);
        }
    }
    System.out.println();
}
// end of printing


// Set Initial Conditions
F[Number_Of_Layers][0] = 0;

// Initiate Shortest Route Algorithm
// We begin from the last layer of the graph
// heading backwards
for(int Phase = Number_Of_Layers-1; Phase >= 0; Phase-- )
{
    // for every node in current layer
    for(int node=0;
            node < Integer.parseInt(
                String.valueOf(
                Nodes_Per_Layer.getValueAt(Phase, 1)));
            node++)
    {
```

```
            // Compute Shortest Route from node
            // "Phase-Node" where "Phase" is the layer
            // and "Node" is the index of the layers'nodes
            // to the target node of the graph

            F[Phase][node] = Determine_Min_Element(A[Phase][node],
                                        F[Phase+1]    ,
                                        Phase        );

          // Computation done based on the assumption that the Shortest
          // Route is the edge connecting current Node to the node of the
          // next layer , for which the route to the target node is
          // the Shortest
          }

       }
       // End of Shortest Route Algorithm

       // Print out the results
       System.out.println(F[0][0]);

       for(int i=0; i<= Number_Of_Layers; i++ )
       {
          for(int j=0; j< Integer.parseInt(
                       String.valueOf(
                       Nodes_Per_Layer.getValueAt(i, 1)));
              j++ )
             System.out.printf("%2d ",F[i][j]);
          System.out.println();
       }
       // End of Printing


       for(int i=1; i< Number_Of_Layers; i++)
          Result[i] = Mapped_A[i]
                          [Used_Edge_Index[i]]
                          [Used_Edge_Index[i+1]];

       Result[0] = "A"+Result[1].charAt(0);
       Result[Number_Of_Layers] =   String.valueOf(F[0][0]);

       for(int i=0; i<= Number_Of_Layers; i++)
          System.out.println(Result[i]);

       return Result;
    }
}
```

A.1.3. FILE Diophintine_Equations.java

```
package Engine;

/* @author Paris Tsampras */
```

```
public class Diophintine_Equations extends Knapsack
{

   public static void BackTracking_3(Integer[] B,Integer Sum, Integer[] C)
   {
      Integer A[] = new Integer[Number_Of_Phases];

      for(int i=0; i< B.length; i++ )
         A[i] = B[i];

      int index =0;
      for (int i = 0; i * A[0] <= Sum; i++) {
         for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
            for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <=Sum;k++) {
               if (i * A[0] + j *  A[1] + k * A[2] == Sum
                  && i*C[0]+j*C[1]+k*C[2]<= Maximum_Load) {
                  Weigths_Per_Item[0] = i;
                  Weigths_Per_Item[1] = j;
                  Weigths_Per_Item[2] = k;
                  System.out.println(i + "," + j + "," + k);
               }
            }
         }
      }
   }

   public static void BackTracking_4(Integer[] B,Integer Sum, Integer[] C)
   {
      Integer A[] = new Integer[Number_Of_Phases];

      for(int i=0; i< B.length; i++ )
         A[i] = B[i];

      int index =0;
      for (int i = 0; i * A[0] <= Sum; i++) {
      for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
      for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
      for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3]<= Sum;l++){
          if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] == Sum
          && i*C[0]+j*C[1]+k*C[2]+l*C[3]<= Maximum_Load) {
          Weigths_Per_Item[0] = i;
          Weigths_Per_Item[1] = j;
          Weigths_Per_Item[2] = k;
          Weigths_Per_Item[3] = l;
          System.out.println(i + "," + j + "," + k +"," + l);
                  }
               }
            }
         }
      }
   }


   public static void BackTracking_5(Integer[] B,Integer Sum, Integer[] C)
   {
```

```
    Integer A[] = new Integer[Number_Of_Phases];

    for(int i=0; i< B.length; i++ )
       A[i] = B[i];

    int index =0;
    for (int i = 0; i * A[0] <= Sum; i++) {
    for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
    for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
   for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
   for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <=m; m++) {
   if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] + m*A[4] == Sum
   && i*C[0]+j*C[1]+k*C[2]+l*C[3]+ m*C[4]<= Maximum_Load) {
      Weigths_Per_Item[0] = i;
      Weigths_Per_Item[1] = j;
      Weigths_Per_Item[2] = k;
      Weigths_Per_Item[3] = l;
      Weigths_Per_Item[4] = m;
      System.out.println(i + "," + j + "," + k +"," + l+"," + m);
                    }
                 }
              }
           }
        }
     }
  }

  public static void BackTracking_6(Integer[] B,Integer Sum, Integer[] C)
  {
     Integer A[] = new Integer[Number_Of_Phases];

     for(int i=0; i< B.length; i++ )
        A[i] = B[i];

     int index =0;
     for (int i = 0; i * A[0] <= Sum; i++) {
     for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
     for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
     for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
     for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <= Sum; m++) {
     for (int o = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] <= Sum; o++)
{
                    if (i*A[0] +j*A[1] + k *A[2] +l*A[3]+m*A[4]+o*A[5] == Sum
         && i*C[0]+j*C[1]+k*C[2]+l*C[3]+ m*C[4]+o*C[5]<= Maximum_Load) {
                    Weigths_Per_Item[0] = i;
              Weigths_Per_Item[1] = j;
              Weigths_Per_Item[2] = k;
              Weigths_Per_Item[3] = l;
              Weigths_Per_Item[4] = m;
              Weigths_Per_Item[5] = o;
       System.out.println(i + "," + j + "," + k +"," + l+"," + m+"," + o);
                       }
                    }
                 }
              }
```

```
            }
          }
        }
    }

    public static void BackTracking_7(Integer[] B,Integer Sum, Integer[] C)
    {
        Integer A[] = new Integer[Number_Of_Phases];

        for(int i=0; i< B.length; i++ )
            A[i] = B[i];

        int index =0;
        for (int i = 0; i * A[0] <= Sum; i++) {
        for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
        for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
        for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
        for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <= Sum; m++) {
        for (int o = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] <= Sum; o++)
{
        for (int p = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] <=
Sum; p++) {
        if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] + m*A[4] + o*A[5] + p*A[6] == Sum
      && i*C[0]+j*C[1]+k*C[2]+l*C[3]+ m*C[4]+o*C[5]+p*C[6]<= Maximum_Load) {
            Weigths_Per_Item[0] = i;
       Weigths_Per_Item[1] = j;
       Weigths_Per_Item[2] = k;
       Weigths_Per_Item[3] = l;
       Weigths_Per_Item[4] = m;
       Weigths_Per_Item[5] = o;
       Weigths_Per_Item[6] = p;
 System.out.println(i + "," + j + "," + k +"," + l+"," + m+"," + o+"," + p);
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
        }
    }

    public static void BackTracking_8(Integer[] B,Integer Sum, Integer[] C)
    {
        Integer A[] = new Integer[Number_Of_Phases];

        for(int i=0; i< B.length; i++ )
            A[i] = B[i];

        int index =0;
        for (int i = 0; i * A[0] <= Sum; i++) {
        for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
        for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
        for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
        for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <= Sum; m++) {
```

```
      for (int o = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] <= Sum; o++)
{
      for (int p = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] <=
Sum; p++) {
      for (int q = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] <= Sum; q++) {
      if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] + m*A[4] + o*A[5] + p*A[6]+q*A[7] ==
Sum
      && i*C[0]+j*C[1]+k*C[2]+l*C[3]+ m*C[4]+o*C[5]+p*C[6]+q*C[7] <=
Maximum_Load  ) {
                  Weigths_Per_Item[0] = i;
                  Weigths_Per_Item[1] = j;
                  Weigths_Per_Item[2] = k;
                  Weigths_Per_Item[3] = l;
                  Weigths_Per_Item[4] = m;
                  Weigths_Per_Item[5] = o;
                  Weigths_Per_Item[6] = p;
                  Weigths_Per_Item[7] = q;
System.out.println(i + "," + j + "," + k +"," + l+"," + m+"," + o+"," + p+"," + q);
                                    }
                                 }
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
      }
   }

   public static void BackTracking_9(Integer[] B,Integer Sum,Integer[] C)
   {
      Integer A[] = new Integer[Number_Of_Phases];

      for(int i=0; i< B.length; i++ )
         A[i] = B[i];

      int index =0;
      for (int i = 0; i * A[0] <= Sum; i++) {
      for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
      for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
      for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
      for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <= Sum; m++) {
      for (int o = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] <= Sum; o++)
{
      for (int p = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] <=
Sum; p++) {
      for (int q = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] <= Sum; q++) {
      for (int r = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] + r* A[8] <= Sum; r++) {
        if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] + m*A[4] + o*A[5] + p*A[6]+q*A[7]+r*
A[8] == Sum
                                    && i*C[0]+j*C[1]+k*C[2]+l*C[3]+
m*C[4]+o*C[5]+p*C[6]+q*C[7]+r*C[8] <= Maximum_Load ) {
```

```
          Weigths_Per_Item[0] = i;
          Weigths_Per_Item[1] = j;
          Weigths_Per_Item[2] = k;
          Weigths_Per_Item[3] = l;
          Weigths_Per_Item[4] = m;
          Weigths_Per_Item[5] = o;
          Weigths_Per_Item[6] = p;
          Weigths_Per_Item[7] = q;
          Weigths_Per_Item[8] = r;
          System.out.println(i + "," + j + "," + k +"," + l+"," + m+"," + o+"," + p+"," + q+"," +
r);
                                    }
                                 }
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
      }
   }

   public static void BackTracking_10(Integer[] B,Integer Sum,Integer[] C)
   {
      Integer A[] = new Integer[Number_Of_Phases];

      for(int i=0; i< B.length; i++ )
         A[i] = B[i];

      int index =0;
      for (int i = 0; i * A[0] <= Sum; i++) {
      for (int j = 0; i * A[0] + j * A[1] <= Sum; j++) {
      for (int k = 0; i * A[0] + j *  A[1] + k * A[2] <= Sum; k++) {
      for (int l = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] <= Sum; l++) {
      for (int m = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] <= Sum; m++) {
      for (int o = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] <= Sum; o++)
{
      for (int p = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] <=
Sum; p++) {
      for (int q = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] <= Sum; q++) {
      for (int r = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] + r* A[8] <= Sum; r++) {
      for (int s = 0; i * A[0] + j *  A[1] + k * A[2] + l*A[3] + m*A[4] + o*A[5] +p*A[6] +
q*A[7] + r* A[8] +s*A[9] <= Sum; s++) {
      if (i * A[0] + j *  A[1] + k * A[2]  + l*A[3] + m*A[4] + o*A[5] + p*A[6]+q*A[7]+r*
A[8]+s*A[9] == Sum
                                       && i*C[0]+j*C[1]+k*C[2]+l*C[3]+
m*C[4]+o*C[5]+p*C[6]+q*C[7]+r*C[8]+s*C[9] <= Maximum_Load  ) {
                                    Weigths_Per_Item[0] = i;
                                    Weigths_Per_Item[1] = j;
                                    Weigths_Per_Item[2] = k;
                                    Weigths_Per_Item[3] = l;
                                    Weigths_Per_Item[4] = m;
```

```
                                    Weigths_Per_Item[5] = o;
                                    Weigths_Per_Item[6] = p;
                                    Weigths_Per_Item[7] = q;
                                    Weigths_Per_Item[8] = r;
                                    Weigths_Per_Item[9] = s;
                                    System.out.println(i + "," + j + "," + k +"," + l+"," + m+"," +
o+"," + p+"," + q+"," + r+"," + s);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
}
```

ANNEX B

B.2 PACKAGE GUI

B.2.1. FILE Login_Screen.java

```
package GUI;

import javax.swing.JOptionPane;

/* @author paris tsampras */
public class Login_Screen extends javax.swing.JFrame {

    public Login_Screen()
    {
        initComponents();
        setLocationRelativeTo(null);
```

```
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jTabbedPane1 = new javax.swing.JTabbedPane();
        filler2 = new javax.swing.Box.Filler(new java.awt.Dimension(0, 0), new
java.awt.Dimension(0, 0), new java.awt.Dimension(32767, 32767));
        jLabel5 = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jPasswordField1 = new javax.swing.JPasswordField();
        jTextField1 = new javax.swing.JTextField();
        Login_Button = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jDesktopPane1 = new javax.swing.JDesktopPane();

        jLabel5.setText("jLabel5");

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 100, Short.MAX_VALUE)
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 100, Short.MAX_VALUE)
        );

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("National & Kapodistrian University of Athens");
        setResizable(false);

        jLabel2.setText("USERNAME :");

        jLabel3.setText("PASSWORD :");

        jTextField1.setName("UserNameField"); // NOI18N

        Login_Button.setText("LOGIN");
        Login_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Login_ButtonActionPerformed(evt);
            }
        });

        jLabel6.setText("                    2016 - 2017 University of Athens -
Department of economics");
```

```
        jLabel7.setFont(new java.awt.Font("Times New Roman", 2, 18)); // NOI18N
        jLabel7.setText("DYNAMIC PROGRAMMING PROBLEMS SOLVERv1.0");
        jLabel7.setIconTextGap(1);

        jLabel8.setIcon(new javax.swing.ImageIcon(getClass().getResource("/GUI/uoa.jpg")));
// NOI18N

        javax.swing.GroupLayout jDesktopPane1Layout = new
javax.swing.GroupLayout(jDesktopPane1);
        jDesktopPane1.setLayout(jDesktopPane1Layout);
        jDesktopPane1Layout.setHorizontalGroup(

jDesktopPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 0, Short.MAX_VALUE)
        );
        jDesktopPane1Layout.setVerticalGroup(

jDesktopPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 0, Short.MAX_VALUE)
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel8)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 101,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 107,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE,
474, javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                                .addComponent(Login_Button,
javax.swing.GroupLayout.PREFERRED_SIZE, 111,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel3)
```

```
                        .addComponent(jLabel2,
javax.swing.GroupLayout.PREFERRED_SIZE, 82,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                        .addComponent(jTextField1)
                        .addComponent(jPasswordField1,
javax.swing.GroupLayout.DEFAULT_SIZE, 115, Short.MAX_VALUE))))
                .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE,
474, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(0, 37, Short.MAX_VALUE))
            .addComponent(jDesktopPane1)))
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel6)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 186,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(32, 32, 32)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(jLabel1, javax.swing.GroupLayout.Alignment.TRAILING)
               .addComponent(jLabel4, javax.swing.GroupLayout.Alignment.TRAILING)
               .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                  .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addComponent(jLabel2)))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(jLabel3)
               .addComponent(jPasswordField1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Login_Button)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jDesktopPane1))
         .addComponent(jLabel8, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
      );

      pack();
   }// </editor-fold>

   private void Login_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
      String Username = new String("");
      String Password = new String(jPasswordField1.getPassword());
```

```
      Username = jTextField1.getText();

      if ( Username.equals("Econ") && Password.equals("Econ") )
      {

         new Main_menu(this).run();
      }
      else
         JOptionPane.showMessageDialog(null, "Login Failed , Try Again");
   }

   public static void main(String args[]) {

       try {
          for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
             if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
             }
          }
       } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Login_Screen.class.getName()).log(java.util.logging.Leve
l.SEVERE, null, ex);
       } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Login_Screen.class.getName()).log(java.util.logging.Leve
l.SEVERE, null, ex);
       } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Login_Screen.class.getName()).log(java.util.logging.Leve
l.SEVERE, null, ex);
       } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Login_Screen.class.getName()).log(java.util.logging.Leve
l.SEVERE, null, ex);
       }
       //</editor-fold>


       java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
             new Login_Screen().setVisible(true);
          }
       });
   }

   // Variables declaration - do not modify
   private javax.swing.JButton Login_Button;
   private javax.swing.Box.Filler filler2;
   private javax.swing.JDesktopPane jDesktopPane1;
   private javax.swing.JLabel jLabel1;
   private javax.swing.JLabel jLabel2;
   private javax.swing.JLabel jLabel3;
```

```
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPasswordField jPasswordField1;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
}
```

B.2.2. FILE Main_Menu.java

```
package GUI;

/* @author Paris Tsampras */
public class Main_menu extends javax.swing.JFrame {

    public Main_menu(javax.swing.JFrame Previous)
    {
        initComponents();
        jButton3.setVisible(false);
        jButton4.setVisible(false);
        jButton5.setVisible(false);
        setVisible(true);
        setLocationRelativeTo(null);
        Previous.setVisible(false);
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jToggleButton1 = new javax.swing.JToggleButton();
        jLabel8 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        KnapSack_Problem_Button = new javax.swing.JButton();
        Shortest_Route = new javax.swing.JButton();
        jButton3 = new javax.swing.JButton();
        jButton4 = new javax.swing.JButton();
        jButton5 = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        Main_Menu_Description = new javax.swing.JTextArea();
        jDesktopPane1 = new javax.swing.JDesktopPane();

        jToggleButton1.setText("jToggleButton1");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("National & Kapodistrian University of Athens");
        setResizable(false);

        jLabel8.setIcon(new javax.swing.ImageIcon(getClass().getResource("/GUI/uoa.jpg")));
// NOI18N
```

```
        jLabel7.setFont(new java.awt.Font("Times New Roman", 2, 18)); // NOI18N
        jLabel7.setText("DYNAMIC PROGRAMMING PROBLEMS SOLVERv1.0");

        KnapSack_Problem_Button.setText("Knapsack Problem");
        KnapSack_Problem_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                KnapSack_Problem_ButtonActionPerformed(evt);
            }
        });

        Shortest_Route.setText("Shortest Route Problem");
        Shortest_Route.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Shortest_RouteActionPerformed(evt);
            }
        });

        jButton3.setText("Problem 3");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });

        jButton4.setText("Problem 5");
        jButton4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton4ActionPerformed(evt);
            }
        });

        jButton5.setText("Problem 4");
        jButton5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton5ActionPerformed(evt);
            }
        });

        jLabel6.setText("                    2016 - 2017 University of Athens -
Department of economics");

        Main_Menu_Description.setEditable(false);
        Main_Menu_Description.setColumns(20);
        Main_Menu_Description.setFont(new java.awt.Font("Times New Roman", 1, 24)); //
NOI18N
        Main_Menu_Description.setRows(5);
        Main_Menu_Description.setText("Welcome to Dynamic Programming\nProlems
Solver.\n\nPlease Choose one of the problems on\nthe left to continue");
        jScrollPane1.setViewportView(Main_Menu_Description);

        javax.swing.GroupLayout jDesktopPane1Layout = new
javax.swing.GroupLayout(jDesktopPane1);
        jDesktopPane1.setLayout(jDesktopPane1Layout);
        jDesktopPane1Layout.setHorizontalGroup(
```

```
jDesktopPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 0, Short.MAX_VALUE)
        );
        jDesktopPane1Layout.setVerticalGroup(

jDesktopPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 0, Short.MAX_VALUE)
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel8)
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE,
415, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE,
474, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jButton3,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jButton5,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(Shortest_Route,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jButton4,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(KnapSack_Problem_Button,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                            .addGap(0, 0, Short.MAX_VALUE))
                        .addComponent(jDesktopPane1))
                    .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED_SIZE, 594,
Short.MAX_VALUE)
        .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 25,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 94,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(KnapSack_Problem_Button)
                    .addGap(18, 18, 18)
                    .addComponent(Shortest_Route)
                    .addGap(21, 21, 21)
                    .addComponent(jButton3)
                    .addGap(18, 18, 18)
                    .addComponent(jButton5)
                    .addGap(18, 18, 18)
                    .addComponent(jButton4))
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
189, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jDesktopPane1)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void KnapSack_Problem_ButtonActionPerformed(java.awt.event.ActionEvent evt)
{

        new knapsac_problem(this);
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void Shortest_RouteActionPerformed(java.awt.event.ActionEvent evt) {
        new Shortest_Route(this);
    }


        public void run()
```

```
        {

        }


    // Variables declaration - do not modify
    private javax.swing.JButton KnapSack_Problem_Button;
    private javax.swing.JTextArea Main_Menu_Description;
    private javax.swing.JButton Shortest_Route;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JDesktopPane jDesktopPane1;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JToggleButton jToggleButton1;
    // End of variables declaration
}
```

B.2.3. FILE Knapsack_Problem.java

```
package GUI;

import Engine.Knapsack;
import static Engine.Knapsack.Number_Of_Phases;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

/* @author Paris Tsampras */
public class knapsac_problem extends javax.swing.JFrame {

    // global class variables
    Boolean Number_of_Items_Is_Valid ;
    Boolean Maximum_Weight_Is_Valid ;
    Integer Max_No_Of_Items;
    Integer Max_Kg_Of_Weigth;
    Integer Min_No_Of_Items;
    Integer Min_Kg_Of_Weigth;

    javax.swing.JFrame Previous_Activity;

    /* Constructor Routine*/
    public knapsac_problem(javax.swing.JFrame Previous)
    {
        initComponents();
        this.Knapsack_panel.setVisible(false);
        this.Solution_Panel.setVisible(false);
        setVisible(true);
        setLocationRelativeTo(null);
        Previous.setVisible(false);
```

```
        Previous_Activity       = Previous;
        Number_of_Items_Is_Valid = false;
        Maximum_Weight_Is_Valid  = false;
        Max_No_Of_Items         = 10;
        Max_Kg_Of_Weigth        = 200;
        Min_No_Of_Items         = 1;
        Min_Kg_Of_Weigth        = 1;
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel5 = new javax.swing.JLabel();
        jLabel1 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        Knapsack_problem_Description = new javax.swing.JEditorPane();
        Weight_Label = new javax.swing.JLabel();
        Maximum_Load_TextField = new javax.swing.JTextField();
        Items_Label = new javax.swing.JLabel();
        Number_Of_Items_TextField = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        Back_To_Main_Menu = new javax.swing.JToggleButton();
        Knapsack_panel = new javax.swing.JDesktopPane();
        jScrollPane2 = new javax.swing.JScrollPane();
        Knapsack_Data = new javax.swing.JTable();
        Solve = new javax.swing.JButton();
        jScrollPane4 = new javax.swing.JScrollPane();
        User_Guidlines = new javax.swing.JTextArea();
        Solution_Panel = new javax.swing.JDesktopPane();
        jScrollPane3 = new javax.swing.JScrollPane();
        Solution_Table = new javax.swing.JTable();
        Continue = new javax.swing.JButton();
        Reset_All_Fields = new javax.swing.JButton();

        jLabel5.setForeground(new java.awt.Color(255, 255, 255));
        jLabel5.setText("jLabel5");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("National & Kapodistrian University of Athens");
        setResizable(false);

        jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/GUI/uoa.jpg")));
// NOI18N
        jLabel1.setText("jLabel1");

        jLabel6.setText("                              2016 - 2017 University of Athens -
Department of economics");

        jLabel7.setFont(new java.awt.Font("Times New Roman", 2, 18)); // NOI18N
        jLabel7.setText("DYNAMIC PROGRAMMING PROBLEMS SOLVERv1.0");

        Knapsack_problem_Description.setEditable(false);
```

```
        Knapsack_problem_Description.setFont(new java.awt.Font("Times New Roman", 1, 18)); // NOI18N
        Knapsack_problem_Description.setText("The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.");
        jScrollPane1.setViewportView(Knapsack_problem_Description);

        Weight_Label.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Weight_Label.setText("Maximum weight that the Knapsack can hold :");

        Items_Label.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Items_Label.setText("Number of Items that must be placed in the Knapsack :");

        jLabel4.setText("Kg");

        Back_To_Main_Menu.setText("Back to Main Menu");
        Back_To_Main_Menu.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Back_To_Main_MenuActionPerformed(evt);
            }
        });

        Knapsack_panel.setBackground(new java.awt.Color(255, 255, 255));
        Knapsack_panel.setToolTipText("");

        Knapsack_Data.setBorder(new javax.swing.border.MatteBorder(null));
        Knapsack_Data.setFont(new java.awt.Font("Times New Roman", 3, 18)); // NOI18N
        Knapsack_Data.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Items", "Values", "Weights"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, true, true
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Knapsack_Data.getTableHeader().setReorderingAllowed(false);
        jScrollPane2.setViewportView(Knapsack_Data);
        if (Knapsack_Data.getColumnModel().getColumnCount() > 0) {
            Knapsack_Data.getColumnModel().getColumn(0).setResizable(false);
            Knapsack_Data.getColumnModel().getColumn(1).setResizable(false);
            Knapsack_Data.getColumnModel().getColumn(2).setResizable(false);
        }

        Solve.setText("Solve");
```

```
        Solve.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SolveActionPerformed(evt);
            }
        });

        User_Guidlines.setEditable(false);
        User_Guidlines.setColumns(3);
        User_Guidlines.setFont(new java.awt.Font("Times New Roman", 3, 12)); // NOI18N
        User_Guidlines.setRows(5);
        User_Guidlines.setText("1. Fill the respective values and weigths and\n   Press
\"Solve\"Button to see the solution.\n2.Press \"Reset All Fields\" Button to start over.\n3.Press
\"Back to main menu\" to return");
        jScrollPane4.setViewportView(User_Guidlines);

        Solution_Panel.setBackground(new java.awt.Color(255, 255, 255));

        Solution_Table.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Solution_Table.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Solution"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Solution_Table.setGridColor(new java.awt.Color(255, 51, 51));
        Solution_Table.getTableHeader().setReorderingAllowed(false);
        jScrollPane3.setViewportView(Solution_Table);
        if (Solution_Table.getColumnModel().getColumnCount() > 0) {
            Solution_Table.getColumnModel().getColumn(0).setResizable(false);
        }

        Solution_Panel.setLayer(jScrollPane3, javax.swing.JLayeredPane.DEFAULT_LAYER);

        javax.swing.GroupLayout Solution_PanelLayout = new
javax.swing.GroupLayout(Solution_Panel);
        Solution_Panel.setLayout(Solution_PanelLayout);
        Solution_PanelLayout.setHorizontalGroup(

Solution_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(Solution_PanelLayout.createSequentialGroup()
                .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 0,
Short.MAX_VALUE)
                .addContainerGap())
        );
        Solution_PanelLayout.setVerticalGroup(
```

```
Solution_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(Solution_PanelLayout.createSequentialGroup()
            .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 0,
Short.MAX_VALUE)
            .addContainerGap())
    );

    Knapsack_panel.setLayer(jScrollPane2,
javax.swing.JLayeredPane.DEFAULT_LAYER);
    Knapsack_panel.setLayer(Solve, javax.swing.JLayeredPane.DEFAULT_LAYER);
    Knapsack_panel.setLayer(jScrollPane4,
javax.swing.JLayeredPane.DEFAULT_LAYER);
    Knapsack_panel.setLayer(Solution_Panel,
javax.swing.JLayeredPane.DEFAULT_LAYER);

    javax.swing.GroupLayout Knapsack_panelLayout = new
javax.swing.GroupLayout(Knapsack_panel);
    Knapsack_panel.setLayout(Knapsack_panelLayout);
    Knapsack_panelLayout.setHorizontalGroup(

Knapsack_panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
        .addGroup(Knapsack_panelLayout.createSequentialGroup()
            .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 426,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(Knapsack_panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                .addGroup(Knapsack_panelLayout.createSequentialGroup()

.addGroup(Knapsack_panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                    .addComponent(Solve)
                    .addComponent(jScrollPane4,
javax.swing.GroupLayout.PREFERRED_SIZE, 314,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(0, 0, Short.MAX_VALUE))
            .addComponent(Solution_Panel)))
    );
    Knapsack_panelLayout.setVerticalGroup(

Knapsack_panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
        .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 0,
Short.MAX_VALUE)
        .addGroup(Knapsack_panelLayout.createSequentialGroup()
            .addComponent(jScrollPane4, javax.swing.GroupLayout.PREFERRED_SIZE, 94,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Solve)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Solution_Panel))
    );
```

```
        Continue.setText("Continue");
        Continue.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ContinueActionPerformed(evt);
            }
        });

        Reset_All_Fields.setText("Reset All Fields");
        Reset_All_Fields.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Reset_All_FieldsActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 398,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE,
858, Short.MAX_VALUE)
                            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel7,
javax.swing.GroupLayout.PREFERRED_SIZE, 415,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jLabel6,
javax.swing.GroupLayout.PREFERRED_SIZE, 474,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                .addGap(0, 0, Short.MAX_VALUE))))
                    .addGroup(layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(Knapsack_panel))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(15, 15, 15)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(Continue)
                            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(Items_Label,
javax.swing.GroupLayout.PREFERRED_SIZE, 327,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                        .addComponent(Weight_Label,
javax.swing.GroupLayout.PREFERRED_SIZE, 291,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                        .addComponent(Maximum_Load_TextField,
javax.swing.GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE)
                        .addComponent(Number_Of_Items_TextField))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jLabel4)))
                .addGap(32, 32, 32)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addComponent(Back_To_Main_Menu,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(Reset_All_Fields,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                .addGap(0, 0, Short.MAX_VALUE))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 685,
Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 25,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 137,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(22, 22, 22)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(Weight_Label)
                .addComponent(Maximum_Load_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel4)
                .addComponent(Back_To_Main_Menu)))
            .addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(Items_Label)
                .addComponent(Number_Of_Items_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(Reset_All_Fields))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(Continue)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Knapsack_panel))
      );

      pack();
   }// </editor-fold>

   private void Back_To_Main_MenuActionPerformed(java.awt.event.ActionEvent evt) {
      Maximum_Load_TextField.setText("");
      Number_Of_Items_TextField.setText("");
      Previous_Activity.setVisible(true);
      setVisible(false);
   }

   private void ContinueActionPerformed(java.awt.event.ActionEvent evt) {
   //  Upon Clicking the Continue Button
   //  Data field verification shall follow
   //  In case eligibility is contested error handling
   //  shall follow
      Continue.setEnabled(false);
      Integer Number_of_Items = 0; //default value
      Integer Weigth          = 0; //default value

      if (Number_Of_Items_TextField.getText().contains(",") ||
         Number_Of_Items_TextField.getText().contains(".") ||
         Number_Of_Items_TextField.getText().matches("[0-9]+") == false
        )
         JOptionPane.showMessageDialog(null, "Wrong Input in Number of Items , Try
Again");
      else
      {
         Number_of_Items = Integer.parseInt(Number_Of_Items_TextField.getText());
         if (Number_of_Items <= Max_No_Of_Items && Number_of_Items >=
Min_No_Of_Items)
            Number_of_Items_Is_Valid = true;
         else
            JOptionPane.showMessageDialog(null, "Number of Items should be in range (1 -
100), Try Again");
      }
      if (Maximum_Load_TextField.getText().contains(",") ||
         Maximum_Load_TextField.getText().contains(".") ||
         Maximum_Load_TextField.getText().matches("[0-9]+") == false
        )
         JOptionPane.showMessageDialog(null, "Wrong Input in Maximum Weight , Try
Again");
      else
```

```
        {
            Weigth = Integer.parseInt(Maximum_Load_TextField.getText());
            if (Weigth <= Max_Kg_Of_Weigth && Weigth >= Min_Kg_Of_Weigth)
                Maximum_Weight_Is_Valid = true;
            else
                JOptionPane.showMessageDialog(null, "Max Weight should be in range (1 - 100),
Try Again");
        }
        if (Number_of_Items_Is_Valid == true && Maximum_Weight_Is_Valid == true)
        {
            // knapsac_data_entry
            DefaultTableModel model = (DefaultTableModel) Knapsack_Data.getModel();
            for(int index=1 ; index<=Number_of_Items; index++ )
                model.addRow(new Object[]{"Item "+index,"",""});
            this.Knapsack_panel.setVisible(true);
            this.Knapsack_Data.setVisible(true);
        }
    }

    private void Reset_All_FieldsActionPerformed(java.awt.event.ActionEvent evt) {
    // Upon Clicking "Reset_All_Fields" Button All fields shall
    // be set to their respective default values
        Integer Number_of_Items = 0;                        // default value
        if (Number_Of_Items_TextField.getText().equals("")       == false &&   // if != null
            Number_Of_Items_TextField.getText().matches("[0-9]+") == true    )  // in (0-9)
range
            Number_of_Items = Integer.parseInt(Number_Of_Items_TextField.getText());
        Maximum_Load_TextField.setText("");
        Number_Of_Items_TextField.setText("");
        Continue.setEnabled(true);
        if (Knapsack_panel.isVisible())
        {
            if (Number_of_Items_Is_Valid == true && Maximum_Weight_Is_Valid == true)
            {
                DefaultTableModel model = (DefaultTableModel) Knapsack_Data.getModel();
                for(int index=Number_of_Items ; index>0; index-- )
                    model.removeRow(index-1);
            }
            Knapsack_Data.setVisible(false);
            Knapsack_panel.setVisible(false);
        }
        if (Solution_Panel.isVisible())
        {
            DefaultTableModel model = (DefaultTableModel) Solution_Table.getModel();
            for(int index=Number_of_Items+1 ; index>0; index-- )
                model.removeRow(index-1);
            Solution_Table.setVisible(false);
            Solution_Panel.setVisible(false);
        }
    }

    private void SolveActionPerformed(java.awt.event.ActionEvent evt) {

        // Validate all cells
        Boolean break_loop = false;
```

```
        Integer Number_of_Items = 0; //default value
        Integer Weigth          = 0; //default value
        Number_of_Items = Integer.parseInt(Number_Of_Items_TextField.getText());
        Weigth          = Integer.parseInt(Maximum_Load_TextField.getText());

        DefaultTableModel model = (DefaultTableModel) Knapsack_Data.getModel();
        for(int column=2; column <= 3; column++ )                       // columns
           for (int row=1 ; row <= Number_of_Items ; row++)               // rows
           {
              if (String.valueOf(model.getValueAt(row-1, column-1)).matches("[0-9]+") ==
false)
              {
                 JOptionPane.showMessageDialog(null, "Please insert Numbers Only and press
ENTER , Try Again");
                 break_loop = true;
                 break;
              }
           }
        //End of Validation

        if (break_loop == false)
        {
           String[] Result = new String[Number_of_Items+1];
           Engine.Knapsack.Maximum_Load     = Weigth;
           Engine.Knapsack.Number_Of_Phases = Number_of_Items;
           Engine.Knapsack.Weigths_Per_Item = new Integer[Number_of_Items];
           Result = Engine.Knapsack.Solve_Knapsack(model);

           Solution_Panel.setVisible(true);
           Solution_Table.setVisible(true);
           DefaultTableModel Solution_model = (DefaultTableModel)
Solution_Table.getModel();
           for(int index=1 ; index<=Number_of_Items+1; index++ )
              Solution_model.addRow(new Object[]{Result[index-1]});

        }
     }

     public void run()
     {

     }

     // Variables declaration - do not modify
     private javax.swing.JToggleButton Back_To_Main_Menu;
     private javax.swing.JButton Continue;
     private javax.swing.JLabel Items_Label;
     private javax.swing.JTable Knapsack_Data;
     private javax.swing.JDesktopPane Knapsack_panel;
     private javax.swing.JEditorPane Knapsack_problem_Description;
     private javax.swing.JTextField Maximum_Load_TextField;
     private javax.swing.JTextField Number_Of_Items_TextField;
     private javax.swing.JButton Reset_All_Fields;
     private javax.swing.JDesktopPane Solution_Panel;
     private javax.swing.JTable Solution_Table;
```

```
    private javax.swing.JButton Solve;
    private javax.swing.JTextArea User_Guidlines;
    private javax.swing.JLabel Weight_Label;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    // End of variables declaration
}
```

B.2.4. FILE Shortest_Route.java

```
package GUI;

import java.io.File;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import org.graphstream.graph.Graph;
import org.graphstream.graph.implementations.*;
import org.graphstream.ui.spriteManager.*;
import org.graphstream.ui.*;

/* @author Paris Tsampras */
public class Shortest_Route extends javax.swing.JFrame
{
    javax.swing.JFrame Previous_Activity;
    Integer Number_Of_Layers = 0;
    /* Constructor Routine*/
    public Shortest_Route(javax.swing.JFrame Previous)
    {
        initComponents();
        this.Tables_Panel.setVisible(false);
        this.Tables_Panel_B.setVisible(false);
```

```
    this.Data_Table_A.setVisible(false);
    this.Distances_Table.setVisible(false);
    this.Solution_Panel.setVisible(false);
    this.Solution_Table.setVisible(false);
    this.Find_Path.setVisible(false);
    setVisible(true);
    setLocationRelativeTo(null);
    Previous.setVisible(false);
    Previous_Activity = Previous;
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jScrollPane3 = new javax.swing.JScrollPane();
    Distances = new javax.swing.JTable();
    jLabel8 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    Shortest_Route_problem_Description = new javax.swing.JEditorPane();
    Back_To_Main_Menu = new javax.swing.JToggleButton();
    Number_of_Layers_Label = new javax.swing.JLabel();
    Number_Of_Layers_TextField = new javax.swing.JTextField();
    Reset_All_Fields = new javax.swing.JButton();
    Continue_button = new javax.swing.JButton();
    Tables_Panel = new javax.swing.JDesktopPane();
    jScrollPane2 = new javax.swing.JScrollPane();
    Data_Table_A = new javax.swing.JTable();
    Next_Button = new javax.swing.JButton();
    Tables_Panel_B = new javax.swing.JDesktopPane();
    jScrollPane4 = new javax.swing.JScrollPane();
    Distances_Table = new javax.swing.JTable();
    Create_Graph_Button = new javax.swing.JButton();
    Find_Path = new javax.swing.JButton();
    Solution_Panel = new javax.swing.JDesktopPane();
    jScrollPane5 = new javax.swing.JScrollPane();
    Solution_Table = new javax.swing.JTable();

    Distances.setModel(new javax.swing.table.DefaultTableModel(
        new Object [][] {

        },
        new String [] {
            "NODES_CONNECTION", "Distance"
        }
    ) {
        boolean[] canEdit = new boolean [] {
            false, false
        };

        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit [columnIndex];
        }
```

```
        });
        Distances.getTableHeader().setReorderingAllowed(false);
        jScrollPane3.setViewportView(Distances);
        if (Distances.getColumnModel().getColumnCount() > 0) {
            Distances.getColumnModel().getColumn(0).setResizable(false);
            Distances.getColumnModel().getColumn(1).setResizable(false);
        }

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("National & Kapodistrian University of Athens");
        setBackground(new java.awt.Color(51, 51, 255));
        setResizable(false);

        jLabel8.setIcon(new javax.swing.ImageIcon(getClass().getResource("/GUI/uoa.jpg"))); // NOI18N

        jLabel6.setText("                              2016 - 2017 University of Athens - Department of economics");

        jLabel7.setFont(new java.awt.Font("Times New Roman", 2, 18)); // NOI18N
        jLabel7.setText("DYNAMIC PROGRAMMING PROBLEMS SOLVERv1.0");


jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);

        Shortest_Route_problem_Description.setEditable(false);
        Shortest_Route_problem_Description.setFont(new java.awt.Font("Times New Roman", 1, 18)); // NOI18N
        Shortest_Route_problem_Description.setText(" In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.");
        jScrollPane1.setViewportView(Shortest_Route_problem_Description);

        Back_To_Main_Menu.setText("Back to Main Menu");
        Back_To_Main_Menu.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Back_To_Main_MenuActionPerformed(evt);
            }
        });

        Number_of_Layers_Label.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Number_of_Layers_Label.setText("Number of Layers :");

        Reset_All_Fields.setText("Reset All Fields");
        Reset_All_Fields.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Reset_All_FieldsActionPerformed(evt);
            }
        });
```

```
        Continue_button.setText("Continue");
        Continue_button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Continue_buttonActionPerformed(evt);
            }
        });

        Data_Table_A.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Data_Table_A.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Layer", "#Nodes"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, true
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Data_Table_A.getTableHeader().setReorderingAllowed(false);
        jScrollPane2.setViewportView(Data_Table_A);
        if (Data_Table_A.getColumnModel().getColumnCount() > 0) {
            Data_Table_A.getColumnModel().getColumn(0).setResizable(false);
            Data_Table_A.getColumnModel().getColumn(1).setResizable(false);
        }

        Next_Button.setText("Next");
        Next_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Next_ButtonActionPerformed(evt);
            }
        });

        Tables_Panel.setLayer(jScrollPane2, javax.swing.JLayeredPane.DEFAULT_LAYER);
        Tables_Panel.setLayer(Next_Button, javax.swing.JLayeredPane.DEFAULT_LAYER);

        javax.swing.GroupLayout Tables_PanelLayout = new
javax.swing.GroupLayout(Tables_Panel);
        Tables_Panel.setLayout(Tables_PanelLayout);
        Tables_PanelLayout.setHorizontalGroup(

Tables_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(Tables_PanelLayout.createSequentialGroup()
                .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 180,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(Next_Button)
                .addContainerGap())
        );
```

```
        Tables_PanelLayout.setVerticalGroup(

Tables_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 426,
Short.MAX_VALUE)
            .addGroup(Tables_PanelLayout.createSequentialGroup()
                .addComponent(Next_Button)
                .addGap(0, 0, Short.MAX_VALUE))
        );

        Distances_Table.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Distances_Table.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Connection", "Distance"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, true
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Distances_Table.getTableHeader().setReorderingAllowed(false);
        jScrollPane4.setViewportView(Distances_Table);
        if (Distances_Table.getColumnModel().getColumnCount() > 0) {
            Distances_Table.getColumnModel().getColumn(0).setResizable(false);
            Distances_Table.getColumnModel().getColumn(1).setResizable(false);
        }

        Create_Graph_Button.setText("Create_Graph");
        Create_Graph_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Create_Graph_ButtonActionPerformed(evt);
            }
        });

        Find_Path.setText("Find_Path");
        Find_Path.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                Find_PathActionPerformed(evt);
            }
        });

        Tables_Panel_B.setLayer(jScrollPane4,
javax.swing.JLayeredPane.DEFAULT_LAYER);
        Tables_Panel_B.setLayer(Create_Graph_Button,
javax.swing.JLayeredPane.DEFAULT_LAYER);
        Tables_Panel_B.setLayer(Find_Path, javax.swing.JLayeredPane.DEFAULT_LAYER);
```

```
        javax.swing.GroupLayout Tables_Panel_BLayout = new
javax.swing.GroupLayout(Tables_Panel_B);
        Tables_Panel_B.setLayout(Tables_Panel_BLayout);
        Tables_Panel_BLayout.setHorizontalGroup(

Tables_Panel_BLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(Tables_Panel_BLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane4, javax.swing.GroupLayout.PREFERRED_SIZE, 182,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(Tables_Panel_BLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING, false)
                    .addComponent(Create_Graph_Button,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(Find_Path, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
        Tables_Panel_BLayout.setVerticalGroup(

Tables_Panel_BLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(Tables_Panel_BLayout.createSequentialGroup()
                .addComponent(Create_Graph_Button)
                .addGap(18, 18, 18)
                .addComponent(Find_Path)
                .addGap(0, 0, Short.MAX_VALUE))
            .addComponent(jScrollPane4, javax.swing.GroupLayout.DEFAULT_SIZE, 426,
Short.MAX_VALUE)
        );

        Solution_Table.setFont(new java.awt.Font("Times New Roman", 3, 14)); // NOI18N
        Solution_Table.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Solution"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        Solution_Table.setGridColor(new java.awt.Color(255, 51, 51));
        Solution_Table.getTableHeader().setReorderingAllowed(false);
```

```
        jScrollPane5.setViewportView(Solution_Table);

        Solution_Panel.setLayer(jScrollPane5, javax.swing.JLayeredPane.DEFAULT_LAYER);

        javax.swing.GroupLayout Solution_PanelLayout = new
javax.swing.GroupLayout(Solution_Panel);
        Solution_Panel.setLayout(Solution_PanelLayout);
        Solution_PanelLayout.setHorizontalGroup(

Solution_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(Solution_PanelLayout.createSequentialGroup()
               .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 126,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addGap(0, 0, Short.MAX_VALUE))
        );
        Solution_PanelLayout.setVerticalGroup(

Solution_PanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 0,
Short.MAX_VALUE)
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
           layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
           .addGroup(layout.createSequentialGroup()
              .addComponent(jLabel8)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addComponent(Number_of_Layers_Label,
javax.swing.GroupLayout.PREFERRED_SIZE, 124,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(Number_Of_Layers_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 51,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(1, 1, 1)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                       .addComponent(Reset_All_Fields,
javax.swing.GroupLayout.PREFERRED_SIZE, 121,
javax.swing.GroupLayout.PREFERRED_SIZE)
                       .addComponent(Back_To_Main_Menu)
                       .addComponent(Continue_button,
javax.swing.GroupLayout.PREFERRED_SIZE, 121,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(0, 0, Short.MAX_VALUE))
                 .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE)
                    .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE,
474, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE,
415, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(Tables_Panel,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(Tables_Panel_B,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(Solution_Panel,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel8, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 25,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 91,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(20, 20, 20)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(Back_To_Main_Menu)
                    .addComponent(Number_of_Layers_Label)
                    .addComponent(Number_Of_Layers_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(Reset_All_Fields)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(Continue_button)
                .addGap(16, 16, 16)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(Tables_Panel)
                    .addComponent(Tables_Panel_B)
                    .addComponent(Solution_Panel)))
        );
```

```
        pack();
    }// </editor-fold>

    private void Back_To_Main_MenuActionPerformed(java.awt.event.ActionEvent evt) {
        Previous_Activity.setVisible(true);
        setVisible(false);
    }

    private void Reset_All_FieldsActionPerformed(java.awt.event.ActionEvent evt) {
        // Upon Clicking "Reset_All_Fields" Button All fields shall
        // be set to their respective default values
        if (Tables_Panel.isVisible())
        {
            DefaultTableModel model_B = (DefaultTableModel) Distances_Table.getModel();
            for(int index =Distances_Table.getRowCount(); index>0; index-- )
                model_B.removeRow(index-1);
        }
        Tables_Panel.setVisible(false);
        if (Tables_Panel_B.isVisible())
        {
            DefaultTableModel model = (DefaultTableModel) Data_Table_A.getModel();
            for(int layers =Number_Of_Layers; layers>= 0; layers-- )
              model.removeRow(layers);

        }
        if (Solution_Panel.isVisible())
        {
            DefaultTableModel model = (DefaultTableModel) Solution_Table.getModel();
            for(int index=model.getRowCount() ; index>0; index-- )
                model.removeRow(index-1);
            Solution_Table.setVisible(false);
            Solution_Panel.setVisible(false);
        }
        Tables_Panel_B.setVisible(false);

        Number_Of_Layers_TextField.setText("");
        if(!Continue_button.isVisible())
            Continue_button.setVisible(true);

    }

    private void Continue_buttonActionPerformed(java.awt.event.ActionEvent evt) {

        if (Number_Of_Layers_TextField.getText().contains(",") ||
            Number_Of_Layers_TextField.getText().contains(".") ||
            Number_Of_Layers_TextField.getText().matches("[0-9]+") == false
           )
            JOptionPane.showMessageDialog(null, "Wrong Input in Number_Of_Layers , Try
Again");
        else
        {
            Continue_button.setVisible(false);
            Number_Of_Layers = Integer.parseInt(Number_Of_Layers_TextField.getText());
            Tables_Panel.setVisible(true);
            Data_Table_A.setVisible(true);
```

```
        DefaultTableModel model = (DefaultTableModel) Data_Table_A.getModel();
        model.addRow(new Object[]{"Layer 0 ","1"});
        for(int index=1 ; index<=Number_Of_Layers-1; index++ )
           model.addRow(new Object[]{"Layer "+index,"",""});
        model.addRow(new Object[]{"Layer "+Number_Of_Layers,"1"});
     }
   }

   private void Next_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
     // Validate all cells
     Boolean break_loop = false;
     DefaultTableModel model = (DefaultTableModel) Data_Table_A.getModel();
     for(int index=1 ; index<=Number_Of_Layers+1; index++ )
     {
        //System.out.println(String.valueOf(model.getValueAt(index-1, 1)));
        if (String.valueOf(model.getValueAt(index-1, 1)).matches("[0-9]+") == false)
        {
           JOptionPane.showMessageDialog(null, "Please insert Numbers Only and press
ENTER , Try Again");
           break_loop = true;
           break;
        }
        if(!String.valueOf(model.getValueAt(0, 1)).equals("1") &&
          !String.valueOf(model.getValueAt(Number_Of_Layers, 1)).equals("1"))
        {
           JOptionPane.showMessageDialog(null, "Layer -0- and last layer "
                                    + "must contain 1 node, Try Again");
           break_loop = true;
           break;
        }
     }

     //end of validation
     if (break_loop == false)
     {
        Tables_Panel_B.setVisible(true);
        Distances_Table.setVisible(true);
        DefaultTableModel model_B = (DefaultTableModel) Distances_Table.getModel();

         char Node_Abbrev = 'A';
        // for every layer
        for(int layers =0; layers<= Number_Of_Layers; layers++ )
        {
           if (layers == Number_Of_Layers)
              break;
           else
           {
              int next_layer = layers+1;
              int layer_nodes      =
Integer.parseInt(String.valueOf(model.getValueAt(layers,1)));
              int next_layer_nodes =
Integer.parseInt(String.valueOf(model.getValueAt(next_layer,1)));
              //System.out.println(layer_nodes);
              //System.out.println(next_layer_nodes);
              // for every node of this layer
```

```
            for(int Number_of_Node=1;
                  Number_of_Node<=layer_nodes ;
                  Number_of_Node++)
            {
               // create connections to every node of the next layer
               for(int next_layer_No_Nodes = 1;
                     next_layer_No_Nodes <=  next_layer_nodes;
                     next_layer_No_Nodes++)

                  model_B.addRow(new Object[]{""+Node_Abbrev+(char)(Node_Abbrev +
                                                (char)(layer_nodes-Number_of_Node) +
                                                (char) next_layer_No_Nodes),""});
               Node_Abbrev++;
            }
         }
      }
   }
}

private void Create_Graph_ButtonActionPerformed(java.awt.event.ActionEvent evt) {

   Graph graph = new SingleGraph("Graph");
   SpriteManager sman = new SpriteManager(graph);
   DefaultTableModel model = (DefaultTableModel) Data_Table_A.getModel();

   // assign attributes to graph
   graph.addAttribute("ui.stylesheet", "node { shape: box;"
                                    + "size: 25px, 25px; "
                                    + "fill-color: black;"
                                    + "stroke-mode: plain;"
                                    + "stroke-color: blue;"
                                    + "text-mode:normal;"
                                    + "text-background-mode:none; "
                                    + "text-size: 18;"
                                    + "}");
   // end of assignement
   // Create Nodes
   char Node_Abbrev = 'A';
   for(int layers =0; layers<= Number_Of_Layers; layers++ )
   {
      if (layers == Number_Of_Layers)
         break;
      else
      {
         int layer_nodes = Integer.parseInt(String.valueOf(model.getValueAt(layers,1)));
         for(int Number_of_Node=1;
               Number_of_Node<=layer_nodes ;
               Number_of_Node++)
         {
            graph.addNode(""+Node_Abbrev);
            graph.getNode(""+Node_Abbrev).addAttribute("ui.label", ""+Node_Abbrev);
            //System.out.println(Node_Abbrev);
            Node_Abbrev++;
         }
      }
```

```
        }
        graph.addNode(""+Node_Abbrev);
        graph.getNode(""+Node_Abbrev).addAttribute("ui.label", ""+Node_Abbrev);
        //System.out.println(""+Node_Abbrev);
        //End of Nodes Creation

        DefaultTableModel model_B = (DefaultTableModel) Distances_Table.getModel();
        for(int index =0 ; index <= model_B.getRowCount()-1; index ++)
        {
           if(//String.valueOf(model_B.getValueAt(index,1)).equals("0") ||
             String.valueOf(model_B.getValueAt(index,1)).equals("") )
           {
              // Do not create edge
           }
           else
           {
              // Create Edge
              graph.addEdge(   String.valueOf(model_B.getValueAt(index,0)),
                        ""+String.valueOf(model_B.getValueAt(index,0)).charAt(0),
                        ""+String.valueOf(model_B.getValueAt(index,0)).charAt(1));
              graph.getEdge(
String.valueOf(model_B.getValueAt(index,0))).addAttribute("ui.label",
                          String.valueOf(model_B.getValueAt(index,1)));
           }
        }
        graph.display();

        Find_Path.setVisible(true);


     }

     private void Find_PathActionPerformed(java.awt.event.ActionEvent evt) {

        String[] Result = new String[Number_Of_Layers];

        Engine.Shortest_Route.Used_Edge_Index = new Integer[Number_Of_Layers+1];
        Engine.Shortest_Route.Number_Of_Layers = Number_Of_Layers;
        Result = Engine.Shortest_Route.Find_Path(Data_Table_A, Distances_Table);

        Solution_Panel.setVisible(true);
        Solution_Table.setVisible(true);
        DefaultTableModel Solution_model = (DefaultTableModel) Solution_Table.getModel();
        for(int index=0 ; index<Result.length-1; index++ )
           Solution_model.addRow(new Object[]{"Edge   :"+Result[index]});
        Solution_model.addRow(new Object[]{"Total Cost :"+Result[Number_Of_Layers]});

        // Create Final Graph
        String marked = "edge.marked {fill-color: red;}";

        Graph graph = new SingleGraph("Graph_2");
        SpriteManager sman = new SpriteManager(graph);
        DefaultTableModel model = (DefaultTableModel) Data_Table_A.getModel();

        // assign attributes to graph
```

```
graph.addAttribute("ui.stylesheet", "node { shape: box;"
                                    + "size: 25px, 25px; "
                                    + "fill-color: black;"
                                    + "stroke-mode: plain;"
                                    + "stroke-color: blue;"
                                    + "text-mode:normal;"
                                    + "text-background-mode:none; "
                                    + "text-size: 18;"
                                    + "}");

// end of assignement
// Create Nodes
char Node_Abbrev = 'A';
for(int layers =0; layers<= Number_Of_Layers; layers++ )
{
   if (layers == Number_Of_Layers)
      break;
   else
   {
      int layer_nodes = Integer.parseInt(String.valueOf(model.getValueAt(layers,1)));
      for(int Number_of_Node=1;
            Number_of_Node<=layer_nodes ;
            Number_of_Node++)
      {
         graph.addNode(""+Node_Abbrev);
         graph.getNode(""+Node_Abbrev).addAttribute("ui.label", ""+Node_Abbrev);
         //System.out.println(Node_Abbrev);
         Node_Abbrev++;
      }
   }
}
graph.addNode(""+Node_Abbrev);
graph.getNode(""+Node_Abbrev).addAttribute("ui.label", ""+Node_Abbrev);
//System.out.println(""+Node_Abbrev);
//End of Nodes Creation

DefaultTableModel model_B = (DefaultTableModel) Distances_Table.getModel();
for(int index =0 ; index <= model_B.getRowCount()-1; index ++)
{
   if(//String.valueOf(model_B.getValueAt(index,1)).equals("0") ||
     String.valueOf(model_B.getValueAt(index,1)).equals("") )
   {
      // Do not create edge
   }
   else
   {
      // Create Edge
      graph.addEdge(   String.valueOf(model_B.getValueAt(index,0)),
                  ""+String.valueOf(model_B.getValueAt(index,0)).charAt(0),
                  ""+String.valueOf(model_B.getValueAt(index,0)).charAt(1));
       for(int idx=0 ; idx<Result.length-1; idx++ )
       {
          if(String.valueOf(model_B.getValueAt(index,0)).equals(Result[idx]))
          {
```

```
graph.getEdge(String.valueOf(model_B.getValueAt(index,0))).addAttribute("ui.style", "fill-
color: red;");
                    break;
                }
                graph.getEdge(
String.valueOf(model_B.getValueAt(index,0))).addAttribute("ui.label",
                            String.valueOf(model_B.getValueAt(index,1)));
            }
        }
    }
    graph.display();

}


public void run()
{}

// Variables declaration - do not modify
private javax.swing.JToggleButton Back_To_Main_Menu;
private javax.swing.JButton Continue_button;
private javax.swing.JButton Create_Graph_Button;
private javax.swing.JTable Data_Table_A;
private javax.swing.JTable Distances;
private javax.swing.JTable Distances_Table;
private javax.swing.JButton Find_Path;
private javax.swing.JButton Next_Button;
private javax.swing.JTextField Number_Of_Layers_TextField;
private javax.swing.JLabel Number_of_Layers_Label;
private javax.swing.JButton Reset_All_Fields;
private javax.swing.JEditorPane Shortest_Route_problem_Description;
private javax.swing.JDesktopPane Solution_Panel;
private javax.swing.JTable Solution_Table;
private javax.swing.JDesktopPane Tables_Panel;
private javax.swing.JDesktopPane Tables_Panel_B;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
// End of variables declaration
}
```

REFERENCES

1. Stuart Dreyfus. "Richard Bellman on the birth of Dynamical Programming"
2. Bellman, Richard (1954), "The theory of dynamic programming", Bulletin of the American Mathematical Society.
3. Jerome Adda and Russell Cooper, "Dynamic Economics: Quantitative Methods and Applications", Section 3.2.1, page 28. MIT Press, 2003
4. John von Neumann and Oskar Morgenstern, "Theory of Games and Economic Behavior", Section 15.3.1. Princeton University Press. Third edition, 1953
5. Beck, Matthias; Geoghegan, Ross (2010), The Art of Proof: Basic Training for Deeper Mathematics, New York: Springer
6. Dantzig, Tobias. Numbers: The Language of Science, 1930
7. Silvano Martello – Paolo Toth, "Knapsack problems", algorithms and Computer Implementations, chapter 2, University of Bologna.
8. Silvano Martello – Paolo Toth, "Knapsack problems", algorithms and Computer Implementations, chapter 3, University of Bologna.
9. *Andonov, Rumen; Poirriez, Vincent; Rajopadhye, Sanjay (2000). "Unbounded Knapsack Problem: dynamic programming revisited". European Journal of Operational Research*, 123 (2)
10. Merkle, R. and M. Helman, (1978). "Hiding information and signatures in trapdoor knapsacks"
11. Kellerer, Pferschy, and Pisinger 2004, p. 449
12. Alexander Schrijver, On the History of the Shortest Path Problem
13. Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, S. R., Jr.; Petry, R. M.; Seitz, R. N. (1957). Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems. Cleveland, Ohio: Case Institute of Technology
14. E. Dijkstra, "A note on two problems in connexion with graphs," in Numerische Mathematik, vol. 1. 1959, pp. 269–271.
15. http://graphstream-project.org/download/
16. http://wiki.netbeans.org/Main_Page
17. https://java.com/en/download/