**Assignment 4 specification document – Topo Sort – knapsack – floyd’s algorithms**

In this assignment, you will write java code for each of the following algorithms as discussed in class:

1. Topological Sort
2. 0-1 knapsack problem
3. Floyd’s algorithm

**Problem 1: Topological sort**: This algorithm, if possible, sorts a set of partially ordered objects. Such a set can be visualized as a digraph, where a directed edge from object A to object B means that object A must precede object B in any sorted order of the objects. Remember that there can be more than one topological sort of a set of partially ordered objects. If the set of objects cannot be sorted topologically then the conclusion is that the digraph contains a cycle.

**For additonal detials and examples see lecture 4 class notes.** Here is a pseudo-code for the algorithm (also available in Lecture 4 class notes).

//input: a digraph G with n vertices and the inDegrees of the nodes  
//output: A topological ordering v1, …,vn of the nodes of G, or that G is cyclic

Step 1: create a stack S   
Step 2: for each u, if inDegree(u) = 0 then push u on to S;   
Step 3: i = 1;  
 step 3.1: while S is not empty  
 step 3.1.1: u = pop (S) ;  
 step 3.1.2: print(u);  
 step 3.1.3: i++ ;  
 step 3.1.4: for each neighbour v of u   
 reduce the inDegree(v) by 1;   
 if inDegree(v) = 0 then push(v) on to S.;

Step 4: if i > n then return the output else declare G is cyclic;

**Your programming project - details and assumptions – Part 1.**

In part 1 of the project you will implement the Topological Sorting algorithm using the pseudocode given above.

Remember pseudocode presents the general structure of the code without regard for optimality. You should incorporate, wherever possible, optimality in your implementation.

Assume that the nodes of the digraph are labeled by integers from one onwards. Input to the algorithm is provided through a file. The input file may have zero or more lines of comments followed by an integer that represents the number of nodes in the graph followed by one edge per line. A comment line starts with ‘c’ as the first character in the first column of the line followed by a space char. A pair of integers represents a directed edge from the first integer to the second i.e. (1, 2) represents an edge from 1 to 2. Here is a sample input file:

6

1 2   
 1 6   
 1 5   
 2 3   
 2 6   
 3 6   
 3 4   
 4 6   
 4 5   
 5 6

Your algorithm will output its result to a file, optionally preceded by comments. Output the entire sorted sequence of nodes in one line. Remember that there should be n nodes in topological sort of the digraph with n-vertices. If a topological sort is not possible, output the message that the graph contains a cycle. Provide appropriate annotations to your outputs.

Note: you are allowed to reuse as much of the code as possible from the previous three assignments.

**Problem 2: 0-1 knapsack problem :** A 0-1 knapsack problem tries to solve the following problem**.** Given S = {1, 2, 3, …, n} of objects with weights w = (w1, w2, w3, …, wn) and values v = (v1, v2, v3, …, vn), and W > 0, find a subset T (of S) that will maximize Σ vi , i ε T subject to Σ wi , i ε T is <= W.

**For additional details and examples see lecture 6 class notes.** Here is a pseudo-code for the algorithm (also available in Lecture 6 class notes).

//input: 1.cardinality of the set of elements - n  
// 2. weight and value of each element i – wi, vi  
// 3. maximum sack size – W  
// output: the matrix of subproblem solutions; and, the optimal set with its value.

Knapsack (v, w, n, W ) {

Step 1: int [ n+1] [W + 1] M;  
 Step 2: for ( k = 0 to W ) M[0, k] = 0;  
 Step 3: for (i = 1 to n )   
 Step 3.1: for (j = 0 to W)   
 Step 3.1.1: M[i, j] = max { M[ i -1, j ], M[ i -1, j – wi ] + vi };   
 Step 4: return M[n, W];  
}

**Your programming project - details and assumptions – Part 2.**

In part 2 of the project you will implement the 0-1 knapsack algorithm using a modified version of the the pseudocode given above. The modification should enable you to output the optimal set and its value.

Remember pseudocode presents the general structure of the code without regard for optimality. You are welcome to incorporate optimality in your implementation.

Assume that the members of the set are labeled by integers from one onwards. Input to the algorithm is provided through a file. The input file may have zero or more lines of comments followed by three lines of integers. A comment line starts with ‘c’ as the first character in the first column of a line followed by a space character. The first line represents the cardinality of the set of elements, and the maximum size of the sack. the second line contains the weights of the elements and the third line contains the values of the elements. Here is a sample input file:

5 11

1 2 5 6 7

1 6 18 22 28

Your algorithm will output its result to a file, optionally preceded by comments. Output the entire input, the max value, and the optimal set along with its value. Provide appropriate annotations to your outputs.

**Problem 3: Floyds algorithm.**

**This algorithm finds the shortest path between any two nodes in a digraph.**

**For additional details and examples see lecture 8 class notes.** Here is a pseudo-code (also, available in lecture 8 class notes).

Algorithm Floyd(W)

//Input: the weighted adjacency matrix W of a digraph with n-vertices and no negative-length //cycle  
//Output: the distance matrix of the shortest paths’ between any two nodes. Also, output the //path and its weight between a pair of nodes.

Step 1: D = W   
Step 2: for k = 1..n do   
 Step 2.1: for i = 1..n do  
 Step 2.2: for j = 1..n do  
 dij= min{dij, dik + dkj);

Step 3: return D

**Your programming project - details and assumptions – Part 3**

In part 3 of the project, you will implement Floyd’s algorithm using a modified version of the pseudocode given above. The modified Floyd’s algorithm should enable us to construct the path between any two nodes i and j. the choicd of i and j is left to you but they should not just form an edge.

Remember pseudocode presents the general structure of the code without regard for optimality. You are welcome to incorporate optimality in your implementation.

Assume that the nodes are labeled by integers from one onwards. Input to the algorithm is provided through a file. The input file may have zero or more lines of comments followed by an integer that represents the number of nodes in the graph followed by one edge per line. A comment line starts with ‘c’ as the first character in the first column of the line followed by a space char. A triplet of integers represents a directed edge, where the third integer is the weight of the edge from the first to the second integer. i.e. an edge 1 2 3 means that the weight of the edge from 1 to 2 is 3. Here is a sample input file:

6

1 2 3   
 1 6 5   
 1 5 6   
 2 3 1   
 2 6 4   
 3 6 4   
 3 4 6   
 4 6 5   
 4 5 8   
 5 6 2

Your algorithm will output its results to a file, optionally preceded by comments. Output each of the input and output graphs as 2-D matrices in the traditional (i.e. row-colum arrangment) format without any extra punctuations. **Also, output the path (not just the edge) for a pair of nodes and its weight.** Provide appropriate annotations to your outputs.

Note: you are allowed to reuse as much of the code as possible from the previous three assignments.

**Implementation details:**

**Assignment 4 - Part 1**

1. Define a class called TopoSort<yourLastName> <section#> .
2. Write createDiGraph(File dataFile) that reads the dataFile and constructs a digraph. A digraph is represented as an adjacency list, where each list is a list of edges in no particular order. Initial inDegree array must be constructed in constant cost. When a new edge is added to the list of edges, make sure that it is not already in the list. Must use the ‘contains(Object e)’ to perform this test.
3. Write sort (File dataFile) that implements the pseudo code given in the specification part of this document. Must use an iterator or a for-each loop to process a list of edges. Sorted order must be stored in a string for eventual output.
4. Include a main(…), without any new local variabes, to run the code

**Assignment 4 - Part 2**

1. Define a class called Knapsack<yourLastName> <section#> .
2. Define createKnapsackArray(File dataFile) that reads the dataFile and initializes the variables: setSize, sackSize, weightArray,valueArray, and the knapsackArray.
3. Define knapsackSolutionSet(File dataFile) that computes and stores the solutions to the subproblems. This method also finds and prints the optimal set and its value without any method call.
4. Add any other appropriate, but minimal, code to make the class executable.

**Assignment 4 – Part 3**

1. Define a class called WarshallFloyd<yourLastName> <section#> .
2. Define constructWDAMatrix(dataFile), which reads the data file and constructs the weighted matrix of the digraph.
3. Define shortestPath(File dataFile) to compute the all-source shortest paths matrix.
4. Define String path(**int** i, **int** j) that returns the path between nodes i and j.
5. Add any other appropriate, but minimal, code to make the class executable.
6. You are allowed use of additional, but minimal, memory that will help construct the path between two nodes.

**Submission instructions**

Follow the following schedule:

Assignment 4 – Part 2: due on or before 🡪 section 1: Apr 9; section 2: Apr 10  
Assignment 4 – Part 3: due on or before 🡪 section 1: Apr 16; section 2: Apr 16

Assignment 4 – Part 1: due on or before 🡪 section 1: Apr 23; section 2: Apr 23

Assignment 4 -- Hard deadline 🡪 section 1: Apr 23; section 2: Apr 23