THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Analysis of Algorithms

(Computer Science 340b)

ASSIGNMENT 3

Due date: Tuesday, April 6, 2021, 11:55 PM EDT

1. Modify the KMP string matching algorithm to find the largest prefix of P that matches a substring of T. In other words, you do not need to match all of P inside T; instead, you want to find the largest match (but it has to start with p_1).

2. In the textbook, 15.4-2 (pp. 396).

3. In the textbook, 16.2-5 (pp. 428).

4. In the textbook, 16.3-8 (pp. 436).

5. Solve the following variation of the 0-1 knapsack problem (pp. 425): The assumptions are identical to those of the 0-1 knapsack problem, except that there is an unlimited supply of each item. In other words, the problem is to pack of maximum value with items of given weights and values in a knapsack with a given-weight, but each item may appear many times.

6. Modify minimum spanning tree algorithm to find maximum spanning tree.

7. Find a counter example with three vertices that shows Dijkstra's algorithm does not work when there is negative weight edge.

8. Let $G = (V, E)$ be a weighted directed graph with no negative cycle. Design an algorithm to find a cycle in G with minimum weight. The algorithm should run in time $O(|V|^3)$.

9. Implement the Dijkstra's single source shortest path algorithm for a weighted directed graph using a heap data structure. The time complexity of the algorithm should be $O((|V|+|E|)\log|V|)$.

A heap data structure for peiority queue (check notes "Heaps" from page 9 for priority queue) should be implemented as an abstract data type (a class in C++, Java, or Python) on a set of elements, where each element has an id and a key, with the following operations.

- $heap(keys, n)$: initializes a heap with the array $keys$ of n elements.
- $in\_heap(id)$: returns true if the element whose id is id is in the heap;
- $min\_key()$: returns the minimum key of the heap;
- $min\_id()$: returns the id of the element with minimum key in the heap;
- $key(id)$: returns the key of the element whose id is id in the heap;
- $delete\_min()$: deletes the element with minimum key from the heap;
- $decrease\_key(id, new\_key)$: sets the key of the element whose id is id to new_key if its current key is greater than new_key.

An input graph file will be available. The format of the input file is the following:

- The first line of the input file contains an integer, n, indicating the number of vertices of the input graph.
- Each of the remaining lines contains a triple "$i\ j\ w$", where $1 \leq i, j \leq n$, indicating an edge from vertex i to vertex j with weight w.

Vertex 1 is considered as the source.

The output of your program should be the following:

- The input graph in adjacency list representation: Print the adjacency lists in any reasonable format. Print each edge with its weight.
- The shortest path tree edges with shortest path weights: The edges should be listed in the order in which they are produced by the Dijkastra's algorithm. For each shortest path tree edge (i, j), print "$(i, j) : w$" in a separate line, where w is the shortest path weight from the source to vertex j.

In your gaul account, you should create a directory called "asn3" which contains your **Assignment Submission Form**, **asn3_solution.pdf** for question 1 to question 8, your **program** for question 9, the input file with name **infile**, and a **makefile**. The makefile should be written such that the command "make clean" will remove all the "*.o" files and the command "make" will generate an executable file **asn3** that can be run by typing "asn3 < infile". If you are using Java or Python, you may not need the makefile. In that case, you should have a script file, **asn3**, so that by typing "asn3 < infile" your Java or Python program will run.

You should use script command to capture the screen of the execution of your program. The **resulting file** should also be in directory "asn3".

Your programs have to be able to run on **compute.gaul.csd.uwo.ca** as our TAs will be marking your programs with this machine.

Once ready, create a tar file for directory "asn3" and submit the tar file through OWL course website assignment 3 area. Please check assignment submission guideline for details.

- To answer a question for designing an algorithm, the following three steps are needed
  1. Describe your algorithm in English (**not** with pseudo code);
  2. Show why the algorithm is correct; and
  3. Analyse the complexity of the algorithm.