Australian National University
Research School of Computer Science

COMP3600/COMP6466 in 2016 – Assignment Two

Due: 23:55 Monday, October 24
Late Penalty: 5% per working day, cutting off date October 30

Submit your work electronically through Wattle. The total mark is 50. *Note that the mark you received from each question is proportional to the quality of your solution.*

Question 1 (**25 points**).

A complete graph is an undirected graph with an edge between each pair of vertices. A randomly weighted complete graph is a complete graph, in which each edge is assigned a weight that is a random real number uniformly distributed between 0 and 1.

Let $L(n)$ be the expected (average) sum of edge weights in a minimum spanning tree (MST) of a randomly weighted complete graph G with n vertices. Your tasks are to

(i) Calculate $L(n)$, using both Prim's and Kruskal's algorithms when $n = 10, 100, 500, 1,000$ respectively. **(12 points)**

(ii) Observe the changing trend of the value of $L(n)$ with the growth of n, and explain why this happens. **(5 points)**

(iii) The running times of both Prim's and Kruskal's algorithms when $n = 10, 100, 500, 1,000$ respectively. **(4 points)**

(iv) With the growth on the number of vertices n, observe the running time trends of both Prim's and Kruskal's algorithms, whether they grow at the same speed, or one is growing much faster than the other one, and explain why that happens (**4 points**)

More precisely, write code using one language such as C, C^{++}, Java, or Python program that does this: For each size $n = 10, 100, 500, 1,000$, generate 20 randomly weighted complete graphs with n vertices and determine the sum of the weights of edges in minimum spanning trees of the graphs as well as the running times of Prim's and Kruskal's algorithms. *Notice that the graph generation time will not be taken into account.* In your report, print out

(a) The average $L(n)$ of the sum of edge weights of 20 MSTs for each n by both Prim's and Kruskal's algorithms

(b) Explain the changing trend of the value of $L(n)$ with the growth of n.

(c) The average running times of both Prim's and Kruskal's algorithms for finding MSTs in the 20 graphs for each n

(d) Observe the running time trends of both algorithms with the growth of n, and explain why such running time gaps exist

Your program should have the following properties.

1. Do not read any input. Write one line of output for each n, and at most 5 extra lines of explanatory output.

2. Store the graph as a symmetric matrix containing the edge weights.

3. Implement Kruskal's algorithm, using subroutines of Quick Sort and Directed Forest for disjoint sets, you need coding these two subroutines by yourself. Furthermore, a data structure - the minimum priority queue will be employed in the implementation of Prim's algorithm, and you need to implement the minimum priority queue as well.

4. To measure the running time of an algorithm, refer to a timing procedure in the exercise of the 1st lab.

5. (a) If you use C, write the program as a single file `mst.c` which will compile on Linux with the command

```
gcc -o mst mst.c -lm
```

("**-lm**" selects the maths library; you might not need it). Use real random numbers (type **double**) generated using the library function **rand()** whose use is illustrated as follows.

```
/* This program generates 10 random double numbers in (0,1).
 * Note that rand() returns an integer in the range [0,RAND_MAX],
 * so you need performing some conversion to double and scaling.*/

#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    double x;
    int i;
```

```
    for (i = 0; i < 10; ++i)
    {
        x =  (rand()+0.5)/(RAND_MAX+1.0);
        printf("%11.8f\n",x);
    }
    exit(0);
}
```

(b) If you use Java, write the program as a single file `mst.java` which will compile on Linux with the command

```
javac mst.java
```

Use real random numbers (type **double**) generated using the library function **Math.random()** whose use is illustrated as follows.

```
// This program generates 10 random double numbers in (0,1).
//

class dran
{
   public static void main(String[] args)
   {
      double x;

      for (int i = 0; i < 10; i++)
      {
          x = Math.random();
          System.out.println(x);
      }
   }
}
```

(c) If you use other languages such as Python, follow the instructions similar to the mentioned languages C and Java.

What to submit:

Upload your file `mst.c` or `mst.java` containing the program to Wattle. Don't submit both these files, just one of them.

You also need to submit a report in *the PDF format* that contains the source code and its outputs, and the answers to part (i), part (ii), part (iii), and part (iv) of the question.

Question 2 (**10 points**).

Answer any **two** of the three questions Q2.(a), Q2(b), and Q2.(c). Should you answer all of them, the lower mark ones will be counted as your received mark for this question.

Q2.(a) Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course u to course v if and only if u is a prerequisite for v.

Devise *a linear running time* algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). (**5 points**)

Q2.(b) Given a telecommunication network represented by a directed graph $G = (V, E)$, each link $(u, v) \in E$ has an associated value $p(u, v)$, which is a real number in the range $0 \leq p(u, v) \leq 1$ that represents the reliability of a communication channel from node u to node v. We interpret $p(u, v)$ as the probability that channel from u to v will not fail, and we assume that these probabilities are independent.

Devise an efficient algorithm to find the most reliable path between two given vertices. Notice that the reliability of a path from node s to node t is usually not equal to the reliability of another path from node t to node s, assuming that both nodes s and t are in V. (**5 points**)

Q2.(c) Given an undirected connected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, assume that there are only three distinct weights on its edges, i.e., for each edge $e \in E$, its weight is one of the values in $\{w_1, w_2, w_3\}$, where $w_j > 0$ with $1 \leq j \leq 3$.

Devise an $O(n + m)$ time algorithm to find a minimum spanning tree in G. (**5 points**)

Question 3 (**15 points for COMP3600 and 10 points for COMP6466/Honours**)

Answer only **one** of the following two questions, not all of them. Should you answer both of them, the lower mark will be counted as your received mark for this question.

Q3.(a) A mission-critical production system has n stages that have to be performed sequentially; stage i is performed by machine M_i. Each machine m_i has a probability r_i of functioning reliably and a probability $1 - r_i$ of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the while system works is $r_1 \cdot r_2 \cdot \ldots r_n$. To improve this probability, we add redundancy by having m_i (≥ 1) copies of the machine M_i that perform in stage i. The probability

that all m_i copies fail simultaneously is only $(1-r_i)^{m_i}$, so the probability that stage i is completed correctly is $1-(1-r_i)^{m_i}$ and the probability that the whole system works is $\Pi_{i=1}^{n}(1-(1-r_i)^{m_i})$. Each machine M_i has a cost c_i, and there is a total budget B to buy machines, assuming that both B and c_i are positive integers.

Given the probabilities $r_1, r_2, \ldots, r_n$, the costs $c_1, c_2, \ldots, c_n$, and the budget B, find the redundancies $m_1, m_2, \ldots, m_n$ that are within the available budget and that maximize the probability that the system works correctly.

- Devise an efficient algorithm to determine the values of $m_1, m_2, \ldots, m_n$ under the given budget B such that the probability that the system works correctly is maximized. (**12 points for COMP3600 and 8 points for COMP6466/Honours**)
- Analyze the running time of your algorithm. (**3 points for COMP3600 and 2 points for COMP6466/Honours**)

Q3.(b) A certain string-processing language offers a primitive operation which splits a string into two pieces. Since this operation involves copying the original string, it takes n units of time for a string of length n, regardless of the location of the cut. Suppose, now that you want to break a string into many pieces. The order in which the breaks are made can affect the total running time. For example, if you want to cut a 20-character string at position 3 and 10, then making the first cut at position 3 incurs a total cost 20+17=37, while doing position 10 first has a better cost of 20+10=30.

- Devise a dynamic programming algorithm that, given the locations of m cuts in a string of length n ($m < n$), find the minimum cost of breaking the string into $m+1$ pieces. (**12 points for COMP3600 and 8 points for COMP6466/Honours**)
- Analyze the running time of your algorithm. (**3 points for COMP3600 and 2 points for COMP6466/Honours**)

Question 4 (**5 points for COMP6466/Honours only**)

A $d-$dimensional box with dimensions $(x_1, x_2, \ldots, x_d)$ *fits inside* another box with dimensions $(y_1, y_2, \ldots, y_d)$ if there exists a permutation π on $\{1, 2, \ldots, d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$.

- Describe an efficient method to determine whether or not one d-dimensional box fits inside another. (**2 points**)
- Suppose that you are given a set of n $d-$dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Describe an efficient algorithm to determine the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k}\rangle$ of boxes such that B_{i_j} fits inside $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$ and $1 \leq i_j \leq n$. Express the running time of your algorithm in terms of n and d. (**3 points**)

Bonus Questions (**5 points**)

`Warning:` *the following questions are designed for people who are capable of doing some extra research-related work. Only if you have finished all basic questions* **correctly** *and are willing to challenge yourself, you can proceed the questions.*

BQ.1: Given a 3-edge connected, weighted undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, we say that two edges $e_1 \in E$ and $e_2 \in E$ are *most vulnerable* if their removal will result in the maximum increase on the weighted sum of the edges in a minimum spanning tree of graph $G' = (V, E - \{e_1, e_2\})$.

Devise an $O(n^3)$ time algorithm to identify these two most vulnerable edges e_1 and e_2 in G. (A graph is 3-edge connected if it is still connected after the removal of any two edges from it.) (**3 points**)

BQ.2: Given a directed weighted graph $G = (V, E)$, assume that the shortest path between any two vertices in V contains no more than k edges. Devise an $O(k|E|)$ time algorithm to find a shortest path between a pair of vertices $u \in V$ and $v \in V$. (**2 points**)