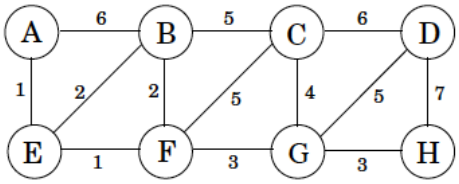
COLORADO MESA UNIVERSITY

CSCI 250. Intro to Algorithms Assignment 2 Group Task. 20 Points

Section 1- Graphs:

1. Consider the following graph.

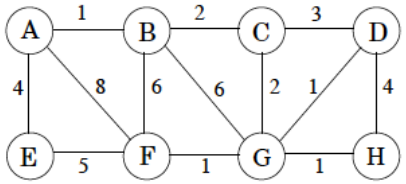


   (a) What is the cost of its minimum spanning tree?

   (b) How many minimum spanning trees does it have?

   (c) Suppose Kruskal's algorithm is run on this graph. In what order are the edges added to the MST? For each edge in this sequence, give a cut that justifies its addition.

2. Suppose we want to find the minimum spanning tree of the following graph.



   (a) Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node A). Draw a table showing the intermediate values of the `cost` array.

   (b) Run Kruskal's algorithm on the same graph. Show how the disjoint-sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is used.

3. Design a linear-time algorithm for the following task.

   Input: A connected, undirected graph G.
   Question: Is there an edge you can remove from G while still leaving G connected?

   Can you reduce the running time of your algorithm to $O(|V|)$?

Section 2 Cost / Complexity / Big O

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Section 3 Algorithmic efficiency

You are given an infinite array $A[\cdot]$ in which the first n cells contain integers in sorted order and the rest of the cells are filled with ∞. You are *not* given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in $O(\log n)$ time. (If you are disturbed by the fact that the array A has infinite length, assume instead that it is of length n, but that you don't know this length, and that the implementation of the array data type in your programming language returns the error message ∞ whenever elements $A[i]$ with $i > n$ are accessed.)

Section 4 QuickSort

i- Apply Quicksort to sort the list E, X, A,M, P, L, E in alphabetical order. Draw the tree of the recursive calls made.

ii- For the partitioning procedure outlined in Levitin's section 5.2:
Explain why if the scanning indices stop while pointing to the same element, i.e., i = j, the value t they are pointing to must be equal to p.

iii- Explain why when the scanning indices stop, j cannot point to an element more than one position to the left of the one pointed to by i.

iv- Give an example showing that Quicksort is not a stable sorting algorithm.

v- Give an example of an array of n elements for which the sentinel mentioned in Levitin's text is actually needed. What should be its value? Also explain why a single sentinel suffices for any input.

vi- For the version of Quicksort given in section 5.2 of Levitin's text:

1. Are arrays made up of all equal elements the worst-case input, the best case input, or neither?
2. Are strictly decreasing arrays the worst-case input, the best-case input, or neither?

vii- Nuts and bolts You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. **Design an algorithm** for this problem with **average-case efficiency in Θ(n log n).**