Sri Dhanush Reddy Kondapalli
CWID: 10476150

Final Exam CS 600

Instruction: *Answer the following questions in this document or another document and submit it in Canvas according to the Final Exam Procedure.*

1. **(12 Points) Consider a connected communication network of routers that form a free tree T. Assume the time-delay of a packet transfer from one router to another is determined by multiplying a small fixed constant by the number of communication links between the two routers. Develop an efficient algorithm, better than $O(n^3)$, that computes the maximum possible time delay in the network T.**

   Solution: Describing the greatest spanning tree algorithm. Considering the graph G, which has n vertices and m edges:

   1. Sort all of the edges in graph G in decreasing order by weight. Let S denote the set that contains the greatest weight spanning tree. Set S = ∅
   2. With the most weight, add the first edge to S.
   3. If there is no cycle in S, add the next edge to S. If there are no more edges, leave the loop.
   4. If S finishes n-1 edges, then stop and display the results in S; otherwise, continue the preceding step.

   Running time: It will take m log m time to sort the edges in decreasing order. And doing the Kruskal algorithm will take O minutes (m log n).
   As a result, it will complete in O(m log n) time.

2. **(12 Points) Suppose you are told that you have a goat and a wolf that need to go from a node s, to a node t, in a directed acyclic graph G. To avoid the wolf eating the goat, their paths must never share an edge. Design an efficient algorithm for finding two edge-disjoint paths in G, if such path exists, to provide a way for the goat and the wolf to go from s to t without risk to the goat.**
   Solution: The preceding problem is analogous to the maximum flow problem:
   1. Consider the flow network's source node s and sink node t. Each edge is assigned a unit capacity.
   2. To find the greatest flow from source to destination, use the Ford-Fulkerson algorithm.
   3. The maximum number of edge-disjoint paths is equal to the maximum flow.
   4. Reduce the capacity by one while using the Ford-Fulkerson algorithm so that this cannot be repeated.
   5. As a result, maximum flow can be equal to the greatest number of edge-disjoint paths.
      Running time: The Ford-Fulkerson algorithm has an *O(f * m)* running time, where f is the maximum flow and m is the number of edges.

3. **(12 Points) Consider a graph G and two distinct vertices, v and w in G. Define HAMILTONIAN-PATH to be the problem of determining whether there is a path that starts at v, and ends at w and visits all the vertices of G exactly once. Show that the HAMILTONIAN-PATH problem is NP-complete.**

   Solution:

To show that the HAMILTONIAN-PATH problem is NP-complete, we need to prove that it is both in NP and NP-hard.

First, let's show that it is in NP. The definition of NP is the set of decision problems for which a proposed solution can be verified in polynomial time. In the case of the HAMILTONIAN-PATH problem, if we are given a proposed path from v to w that visits all the vertices of G exactly once, we can verify in polynomial time whether the proposed path is a valid solution by checking that it starts at v, ends at w, and visits all the vertices of G exactly once.

Next, let's show that it is NP-hard. To do this, we need to prove that it is at least as hard as any other problem in NP. We can do this by showing that every problem in NP can be reduced to the HAMILTONIAN-PATH problem in polynomial time.

One way to do this is to use a reduction from the well-known NP-complete problem of finding a Hamiltonian cycle in a graph. Given a graph G and a vertex v, we can create a new graph G' by adding a new vertex w and an edge from w to v. We can then use the reduction to show that the problem of finding a Hamiltonian cycle in G is equivalent to the problem of finding a Hamiltonian path from w to v in G'. This reduction can be done in polynomial time, so the HAMILTONIAN-PATH problem is NP-hard.

Therefore, the **HAMILTONIAN-PATH** problem is **NP-complete**.

4. (12 Points) **Suppose we are given an undirected graph G with positive weights on its edges and asked to find a tour that visits the vertices of G exactly once and returns to the start so as to minimize the cost of maximum-weight edge in the tour. Assuming that the weights in G satisfy the triangle inequality, design a polynomial-time 3-approximation algorithm for this version of traveling salesperson problem.**

Solution: We can identify the tour that visits the vertices of G precisely once for an undirected graph G with positive weights on its edges:
1. Create a Maximum Spanning Tree G' from G.
2. Define V' as the set of all vertices in a set of V with regard to G'.
3. Determine the lowest cost perfect matching (PM) that belongs to E in V'.
4. To obtain an Eulerian graph, add PM to MST.
5. Locate the Eulerian tour T on the Eulerian Graph.
6. Skip the previously visited vertices on the Eulerian tour T to convert it to C.

Running Time: Polynomial time will be used to compute this method.

Note that this version of TSP is different than the 2-approximation for METRIC-TSP in Section 18.1, where G is assumed to be a complete graph.

5. **(12 Points) Suppose we have a Monte Carlo algorithm, A, and a deterministic algorithm, B, for testing if the output of A is correct. How can we use A and B to construct a Las Vegas algorithm? Also, if A succeeds with probability ½ and both A and B run O(n) time, what is the expected running time of the Las Vegas algorithm that is produced?**
Solution: A Las Vegas algorithm is a randomized algorithm that always returns the correct answer, but its running time is dependent on the probability of success of the

algorithm. To construct a Las Vegas algorithm from Monte Carlo algorithm A and deterministic algorithm B, we can do the following:

1. Run algorithm A. If A returns the correct answer, output the answer and terminate the algorithm.

2. If A does not return the correct answer, run algorithm B to verify the output of A. If B confirms that the output of A is correct, output the answer and terminate the algorithm.

3. If B indicates that the output of A is incorrect, repeat steps 1 and 2 until A returns the correct answer or B confirms the correctness of A's output.

Running Time: The expected running time of this Las Vegas algorithm is equal to the expected number of times we need to run algorithms A and B before A returns the correct answer or B confirms the correctness of A's output. Since A succeeds with probability 1/2 and both A and B run in O(n) time, the expected running time of the Las Vegas algorithm is **O(n)**.

6. (12 Points) Let S be a set of n intervals of the form [a, b], where a < b. Design an efficient data structure that can answer, in O(log n +k) time, queries of the form *contains(x)*, which asks for an enumeration of all intervals in S that contain *x*, where k is the number of such intervals. What is the space usage of your data structure?
Hint: **Think about reducing this to a two-dimensional problem.**

Solution:
The Priority range is an efficient data structure that requests an enumeration of all intervals in S that include x. Trees
Let T be a balanced binary search tree that stores n objects with two-dimensional keys that are sorted by x-coordinates. To answer range queries, T uses priority search trees as auxiliary structures. Priority range tree is the resultant data structure.

Algorithm PSTRangeSearch(x_1, x_2, y_1, y_2, v):
Input: Search keys x_1, x_2, y_1, and y_2; node v in the primary structure T of a priority range tree
Output: The items in the subtree rooted at v whose coordinates are in the x- range [x_1, x_2] and in the y-range [y_1, y_2]

```
if T.isExternal(v) then
        return ∅
if x1 ≤x(v)≤x2 then
        if y1 ≤y(v)≤y2 then
                M ← {element(v)}
         else
                M←∅
   L ← PSTSearch(x1, y1, y2, T (leftChild(v)).root())
   R ← PSTSearch(x2, y1, y2, T (rightChild(v)).root())
   return L∪ M ∪R
else if x(v) < x1 then
        return PSTRangeSearch(x1, x2, y1, y2, T.rightChild(v))
else
        return PSTRangeSearch(x1, x2, y1, y2, T.leftChild(v))
```

Running Time: The priority range-search query will take O(logn + k) time to complete, where k is the number of intervals. The priority range data structure has an O space complexity (n log n).

7. **(10 Points) Given a set P of n points, design an efficient algorithm for constructing a simple polygon whose vertices are the points of P.**
Solution:

1. The anchor point is a point an of P that is a vertex of H. Choose an anchor point in P with the lowest y-coordinate (and minimum x-coordinate if there are ties).
2. The remaining points of P (that is, P - {a}) are sorted radially around a, and S is the resultant sorted list of points. Although no explicit computation of angles is done, the points of P appear in the list S ordered counterclockwise "by angle" with regard to the anchor point a.
3. We scan over the points in S in (radial) order, keeping a list H at each step that stores a convex chain "surrounding" the points scanned thus far. When we evaluate a new point p, we run the following test:
    (a) Add p to the end of H if p makes a left turn with the final two points in H, or if H includes fewer than two points.
    (b) Otherwise, delete the final point in H and run the test again for p.
    We come to a halt when we reach the anchor point a, at which time H stores the vertices of P's convex hull in counterclockwise order.

Running Time: The running time of the algorithm is *O(n log n).*

8. **(10 Points) DNA strings are sometimes spliced into other DNA strings as a product of re-combinant DNA processes. But DNA strings can be read in what would be either the forward or backward direction for a standard character string. Thus it is useful to be able to identify prefixes and their reversals. Let T be a DNA text string of length n. Describe an O(n)-time method for finding the longest prefix of T that is a substring of the reversal of T.**

Hint: Consider using a prefix trie.
Solution:
A DNA text string of length n will be stored in Standard Tries in such a way that no T string is a prefix of another string:

a. By reversing the string, the components are inserted into the Trie S.
b. Now contrast the original DNA T with the S. take the maxstring value of 0 and the currentstring value = 0.
c. If the initial character of T matches the first character of S, then compare it to the kid in S. Continue the procedure by increasing by 1 until the character of T does not match the trie node.
d. If the initial character does not match S, match it with the child node. Continue until a match is discovered.
e. If maxstring is smaller than currentstring, set maxstring to currentstring.
f. Repeat the method until the number of tries is exhausted.

Running time: Running time of the algorithm is O(n)

Sri Dhanush Reddy Kondapalli
CWID: 10476150

9. (8 Points) Solve the following linear program using the Simplex Method:

Maximize: $18\,x_1 + 12.5\,x_2$
Subject to: $x_1 + x_2 \leq 20$
$x_1 \leq 12$
$x_2 \leq 16$
$x_1 , x_2 \geq 0.$

Solution: To find the optimal solution for the given linear programming problem, we follow these steps:

1. Convert the problem to slack form by introducing additional variables and constraints.
2. Identify the variable with the highest positive coefficient in the objective function.
3. Substitute this variable with the most restrictive constraint that it appears in.
4. Repeat steps 2 and 3 until all variables have negative coefficients in the objective function.

In this specific problem, we first convert it to slack form. The objective function becomes:

$$z = 18x_1 + 12.5x_2$$

And the constraints become:

$$x_3 = 20 - x_1 - x_2$$
$$x_4 = 12 - x_1$$
$$x_5 = 16 - x_2$$

The variable x_1 has a positive coefficient in the objective function, and the second constraint is the most restrictive with respect to x_1. We substitute x_1 using $x_1 = 12 - x_4$. The new objective function becomes:

$$z = 216 + 12.5x_2 - 18x_4$$

And the new constraints are:

$$x_3 = 8 - x_2 + x_4$$
$$x_1 = 12 - x_4$$
$$x_5 = 16 - x_2$$

Now, the variable x_2 has a positive coefficient in the new objective function, and the first constraint is the most restrictive with respect to x_2. We replace it by $x2 = 8 - x_3 + x_4$ and substitute x_2 everywhere else. The new objective function becomes:

$$z = 316 - 12.5x_3 - 5.5x_4$$

And the new constraints are:

$$x_2 = 8 - x_3 + x_4$$
$$x_1 = 12 - x_4$$
$$x_5 = 8 + x_3 - x_4$$

Since all coefficients are now negative, the basic solution with $x_1 = 12$ and $x_2 = 8$ is optimal.