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(n3), that computes the maximum possible time delay in the network T.**

Solution: Describing the algorithm for maximum spanning tree. Consider the undirected graph G with *n* vertices and *m* edges.

1. For all the edges in graph G, sort the edges in decreasing order by weight. Let S be the set comprising the maximum weight spanning tree. Set S = ∅
2. Add the first edge to S with maximum weight.
3. Add the next edge to S if there is no cycle present in S. Check if there are no edges remaining exit the loop.
4. If S completes *n-1* edges, then terminate and show results in S otherwise repeat above step.

**Running time:** Sorting the edges according to decreasing order will take m log m time. And performing Kruskal algorithm will take O(m log n).

So, together it will run in O(m log n) time.

1. **(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 above problem is similar to maximum flow problem:

1. Consider the source node s and sink t in flow network. Assigning unit capacity to each edge.
2. Apply Ford-Fulkerson algorithm to find the maximum flow from source to destination.
3. Maximum flow is equal to the maximum number of edge-disjoint paths.
4. While applying Ford-Fulkerson algorithm reduce the capacity by 1 so that this can not be repeated.
5. So maximum flow can be equal to the maximum number of edge-disjoint paths.

**Running time:** The running time ofFord-Fulkerson algorithm is *O(⏐f\*⎟ m)* where *f* is the maximum flow and *m* is the edges.

1. **(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:

Hamiltonian path is a path that contains each vertex in a graph exactly once. For a problem to be NP-Complete it must satisfies Two condition:

1. Proof that the problem is in NP.
2. Proof that the problem is NP-Hard
3. Proof that the problem is in NP.
4. Non-deterministically choosing edges from G that are to be included in path.
5. Traverse the path and make sure visiting the vertex exactly once.

As above algorithm can be done in polynomial time thus it is in NP.

Finding a problem that can be reduced to HAMILTONIAN-PATH. A closely related problem is HAMILTONIAN-CYCLE which is NP-Complete, so we can reduce this problem to HAMILTONIAN-PATH.

1. Proof that the problem is NP-Hard
2. Given a graph G(V,E), construct a graph G’ such that G contains a HAMILTONIAN-CYCLE if and only if G’ contains HAMILTONIAN-PATH.
3. This is done by choosing arbitrary vertex *u* in the G and adding a copy *u’* of it together with all its edges. Then add vertices *v* and *v’* to the graph and connect *v* with *u* and *u’* with *v’.*

**HAMILTONIAN-PATH** is NP complete.

1. (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: For an undirected graph G with positive weights on its edges: We can find the tour that visits the vertices of G exactly once. By

1. Construct a Maximum Spanning Tree G’ from G.
2. Let V’ be the set of all vertices from a set of V with respect to G’
3. Compute the minimum cost perfect matching (PM) belongs to E in V’.
4. Add PM to MST to obtain an Eulerian graph.
5. Find the Eulerian tour T of the Eulerian Graph.
6. Convert the Eulerian tour T into C by skipping the previously visited vertices.

**Running Time**: This algorithm will be computed in polynomial time.

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.

1. **(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: We can construct Las Vegas algorithm using combination of Monte Carlo algorithm A and deterministic algorithm B.

First applying the Monte Carlo algorithm which has probabilistic correctness than using the output of this applying deterministic algorithm B.

Monte Carlo will continue running for every output and then using deterministic algorithm to predict the correct output. This is the same as Las Vegas which gives the certain output every time.

**Running Time**: As the running time of the above algorithm O(2n) as running twice. As the probability of A succeeds is half the running time will be O(n). So, running time of the above algorithm is *O(n).*

1. **(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 efficient data structure which asks for an enumeration of all intervals in S that contain x is the Priority range Trees

Let T be a balanced binary search tree storing n items with two-dimensional keys, ordered according to their x-coordinates. T with priority search trees as auxiliary structures to answer range queries. The resulting data structure is called priority range tree.

**Algorithm** PSTRangeSearch(x1, x2, y1, y2, v):  
***Input:***Search keys x1, x2, y1, and y2; 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 [x1, x2] and in the y-range [y1, y2]

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)* where k is the number of intervals. The space complexity of the priority range data structure is *O(n log n).*

1. **(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. Find a point *a* of P that is a vertex of H and call it the *anchor point*. pick anchor point *a* the point in P with minimum y-coordinate (and minimum x-coordinate if there are ties).
2. We sort the remaining points of P (that is, P − {a}) radially around a, and let S be the resulting sorted list of points. In the list S, the points of P appear sorted counterclockwise “by angle” with respect to the anchor point a, although no explicit computation of angles is performed.
3. After adding the anchor point a at the first and last position of S, we *scan* through the points in S in (radial) order, maintaining at each step a list H storing a convex chain “surrounding” the points scanned so far. Each time we consider new point p, we perform the following test:
4. (a) If p forms a left turn with the last two points in H, or if H contains fewer than two points, then add p to the end of H.

(b) Otherwise, remove the last point in H and repeat the test for p.  
We stop when we return to the anchor point a, at which point H stores the vertices of the convex hull of P in counterclockwise order.

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

1. **(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:

DNA text string of length n will be stored in *Standard Tries* such that no string of T is a prefix of another string.

* 1. Inserting the elements in the Trie S by reversing the string.
  2. Now compare the original DNA T with S. take maxstring value 0 and currentstring value=0
  3. If first character of T is matching with the first character of S, then check it with the child in S. Continue the process by incrementing by 1 until character of T is not matching with the trie node.
  4. If first character is not matching with S, match it with the child node. Repeat until match is found.
  5. If maxstring value is lesser than current string value, make maxstring = currentstring.
  6. Repeat the process until end of the tries.

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

1. (**8 Points) CLRS P. 804**

**Solve the following linear program using the Simplex Method:**

**Maximize: 18 x1 + 12.5 x2**

**Subject to: x1 + x2 ≤ 20**

**x1 ≤ 12**

**x2 ≤ 16**

**x1 , x2 ≥ 0.**

Solution:

