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.

Sol:

We can treat this like a Maximum flow problems.

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.

Sol:

This problem can be solved by reducing it to maximum flow problem. Following are steps.  
**1)** Consider the given source and destination as source and sink in flow network. Assign unit capacity to each edge.  
**2)** Run Ford-Fulkerson algorithm to find the maximum flow from source to sink.  
**3)** The maximum flow is equal to the maximum number of edge-disjoint paths.

When we run Ford-Fulkerson, we reduce the capacity by a unit. Therefore, the edge can not be used again. So the maximum flow is equal to the maximum number of edge-disjoint paths.

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.

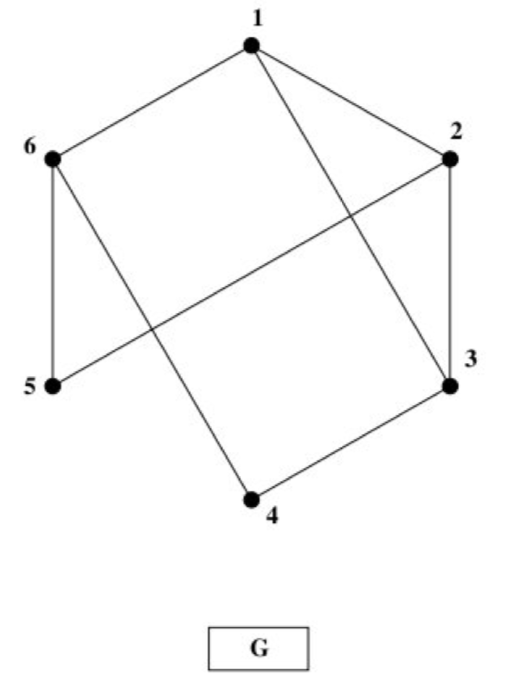
Sol：

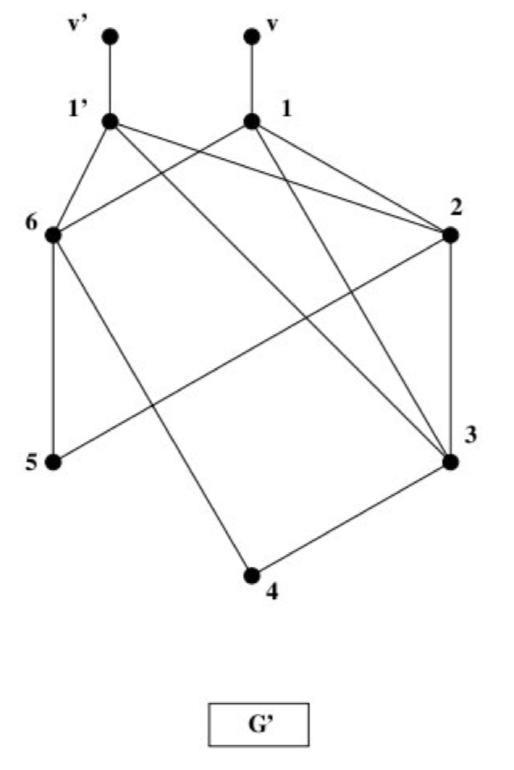
To show that this problem is NP-complete we first need to show that it actually belongs to the class NP and then find a known NP-complete problem that can be reduced to Hamiltonian Path.

For a given graph G we can solve Hamiltonian Path by non-deterministically choosing edges from G that are to be included in the path. Then we traverse the path and make sure that we visit each vertex exactly once. This obviously can be done in polynomial time, and hence, the problem belongs to NP.

Now we have to find an NP-complete problem that can be reduced to Hamiltonian Path. A closely related problem is the problem to determine whether a graph contains a Hamiltonian cycle, that is, a Hamiltonian path that begin and end in the same vertex. Moreover, we know that Hamiltonian Cycle is NP-complete, so we may try to reduce this problem to Hamiltonian Path.

Given a graph G = ⟨V, E⟩ we construct a graph G′ such that G contains a Hamiltonian cycle if and only if G′ contains a Hamiltonian path. This is done by choosing an arbitrary vertex u in 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 v′ with u′; see below is an example.





Suppose first that G contains a Hamiltonian cycle. Then we get a Hamiltonian path in G′ if we start in v, follow the cycle that we got from G back to u′ instead of u and finally end in v′. For example, consider the left graph, G, in Figure 1 which contains the Hamiltonian cycle 1,2,5,6,4,3,1. In G′ this corresponds to the path v,1,2,5,6,4,3,1′,v′.

Conversely, suppose G′ contains a Hamiltonian path. In that case, the path must necessarily have endpoints in v and v′. This path can be transformed to a cycle in G. Namely, if we disregard v and v′,the path must have endpoints in u and u′ and if we remove u′ we get a cycle in G if we close the path back to u instead of u′.

The construction won't work when G is a single edge, so this has to be taken care of as a special case. Hence, we have shown that G contains a Hamiltonian cycle if and only if G′ contains a Hamiltonian path, which concludes the proof that Hamiltonian Path is NP-complete.

1. (12 Points) Supose 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.

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.

Sol:

First,we Let 𝑇 be a tree rooted at 𝑟 with at least two vertices. we can list the vertices of 𝑇 in a sequence such that the distance (in 𝑇) between two adjacent vertices (including the first and the last) is at most 3, and furthermore the vertex following 𝑟 is adjacent to 𝑟 (in 𝑇).

Given this, Suppose that there is a Hamiltonian cycle with heaviest edge having weight 𝑤. Then there is a spanning tree whose heaviest edge has weight 𝑤. So the heaviest edge in the bottleneck spanning tree has weight at most 𝑤. Apply the claim to that tree, and consider the corresponding Hamiltonian cycle. Since the tree distance between adjacent vertices is at most 3 and thanks to the triangle inequality, the heaviest edge in the resulting cycles weighs at most 3𝑤.

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?

Sol:

Algorithm Las Vegas:

For i =1 to infinity do

solMC=A(n);// run the Monte Carlo algorithm here

answer\_from\_verifier = B(solMC)// run the deterministic algorithm here

if answer\_from\_verifier is correct

return solMC

end

end

and,the expected running time is:

=2

=

= 2O(n)/p

= 4O(n)

Which can be seen as 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.

Sol:

I will reduce this to a 2-dimensional problem, by building a k-d tree,T, with 𝑘=2 dimensions, where the interval [a,b] is represented as the point (a,b).

For the tree leaves , which are points sorted by x-coordinate.

For the internal node v,is the subset of points at the leaves in the subtree v.

With this additional information, for every query result interval 𝑞=[x𝑞,y𝑞]

we can perform a range search with query condition a < xq < x.

Firstly ,we can use an algorithm like the 1DTreeRangeSearch to find all the items that contains x in the range [a, x] in O (log n + k) time, where k is the number of answers.

Secondly, Then we traverse the search result above, compare the y𝑞 which meets the condition x <y𝑞 < b; The running time isO(k ) where k is the number of elements reported.

Therefore the running time totally is O (log n + k).

And The space needed to n points in the one-dimensional range tree 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.

Sol：

1. Choose *P*0 ∈ *S* as anchor point. It is the start of bypass.
2. Sort all other points { *S* \ *P*0} with polar angle relative *P*0. As a result, we get one of the possible polygons for a given set *S*. Sorting can be done in *O (n log n)* time
3. (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.

Sol：

1. Computing Trev, the reversal of string T, is O(n)
2. A prefix trie for Trev can be built in O(n) time.
3. Longest prefix of s in Trev can be found in O(n) time using the prefix trie.
4. (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.

To solve this linear program using the simplex method, we first we rewrite the linear

program in slack form, introducing the slack variables: x3, x4,x5

Maximize: z = 18x1 + 12.5x2

Subject to: x3=20 - x1 - x2

x4=12-x1

x5=16-x2

x3, x4, x5 0

We then choose to increase x1, as it has the largest coefficient in the objective function. The most restrictive constraint is given by x4，so we pivot x4 and x1, yielding the following new slack form with objective value , c\*= 216.

Maximize: z = 216 – 18 x4 + 12.5 x2

Subject to: x3 =8 + x4 - x2

X1 =12-x4

x5 =16-x2

Next we increase x2, as it has the largest coefficient in the objective function. So we pivot x3 and x2, yielding the following new slack form with objective value , c\*= 316.

Maximize: z = 316 – 6 x4 -12.5 x3

Subject to: x2 =8 + x4 – x3

x1 =12 - x4

x5 =16 - x4 + x3

For all the coefficients of the objective function are negative. we see that the

optimal value for this linear program is 316, with x1 = 12 and x2 = 8.