Romil V. Shah
20008692

CS / CPE 600
Prof. Reza Peyrovian

Final Exam
Submission Date: 12 / 18 / 2022

Q1. 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.

Given a weighted directed acyclic graph and source point S, find the longest distance from S to all other vertices in the graph. For directed acyclic graphs, the longest path problem has a linear time solution.

First, initialize the distance to all vertices to negative infinity, the distance to the source point is 0, and then find the topological order. The topological ordering of the graphs represents the linear order of a graph.

When the topological order is found, all the vertices in the topological order are processed one by one. For each processed vertex, update the distance to its neighbor by using the current vertex.

Algorithm: findLongestPath()
Input: Tree roots
Output: maximum possible time delay

1. Initialize dist[] = {NINF, NINF, ...} , dist[s] = 0 . s is the source point and NINF is the negative infinity. Dist represents the longest distance from the source point to the other point.

2. Establish a topological sequence of all vertices.

3. Perform the following algorithm for each vertex u in the topology sequence
   For each adjacent point v of u
   If (dist[v] < dist[u] + weight(u, v))
   dist[v] = dist[u] + weight(u, v)

TIME COMPLEXITY:
The topological sorting order = O(V + E).
Following the topological order, a loop is used to discover all nearby points foreach vertex, with a temporal complexity of O(E).
The internal loop thus executes O(V+E) times.
As a result, the algorithm's overall time complexity is O(V+E)

Q2. 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.

Consider the directed graph 'G', which has two vertices: source's' and destination 't'. We need to identify a route that will prevent them from sharing any edges.

This may be addressed by converting the issue to a maximum flow problem using the Ford-Fulkerson technique.

1. In a flow network, consider the supplied source and destination as sources and sinks. Each edge should be assigned a unit capacity.
2. Calculate the maximum flow from source to sink using the Ford-Fulkerson procedure.
3. The maximum flow equals the number of edge-disjoint paths multiplied by the maximum flow.

The Ford-Fulkerson algorithm:

- Set flow total = 0 as the algorithm.
- Repeat until no path exists between s and t.
    - Run a depth first search from the source vertex to find a flow path to the end vertex.
    - Let f be the path's minimum capacity value.
    - Add f to total flow.
- For each of the path's edges
    - Reduce capacity from one side to the other
    - Increase edge capacity from another size to the back

Q3. 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.

Here, we need to prove that the HAMILTONIAN-PATH issue is NP-complete. For that, we must first establish that this belongs to the NP class, and then identify a known NP-complete issue that can be reduced to a Hamiltonian route.

Assume there is a graph G, and the Hamiltonian path is solved by non-deterministically selecting the edges from G that should be included in the path.

Following that, we must travel the path and ensure that each vertex is visited precisely once, which can be accomplished in polynomial time. As a result, this belongs to the NP class.

The next step is to identify an NP - complete issue that can be reduced to a Hamiltonian route. This may be done by looking for a Hamiltonian cycle in the graph, which is a route that starts and finishes at the same vertex. Because the Hamiltonian cycle is NP- complete, we can reduce this to the Hamiltonian route.

We generate a graph G0 from the graph G = (V, E) in which G has a Hamiltonian cycle if and only if G0 contains a Hamiltonian route. This is accomplished by selecting an arbitrary vertex u in G and copying it along with all its edges, u0. Then, in the graph, add vertices v and v0 and connect v with u and v0 with u0.

We assume G includes a Hamiltonian cycle, therefore if we start in v, follow the cycle we obtained from G back to u0 instead of u, and eventually end in v0, we get a Hamiltonian route in G0. Assume that G0 includes a Hamiltonian route. In such scenario, the route must have both v and v0 endpoints. This route may be converted to a G cycle.

If we ignore v and v0, the path must have endpoints in u and u0, and removing u0 causes a G cycle if we close the path back to u instead of u0. When G is a single edge, the construction will fail, hence this must be handled separately.

As a result, we've demonstrated that G includes a Hamiltonian cycle only if and only if G0 contains a Hamiltonian path, completing the proof that Hamiltonian Path is NP- complete.

Q4. 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 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.

Hint: Show that it is possible to turn a Euler-tour traversal, E, of an MST for G into a tour visiting each vertex exactly once such that each edge of the tour skips at most two vertices of E.

Sol.

Let S represent the vertices v1 to vn.

We need to check 2 things in this traveling salesperson problem:
1. Every edge in G is crossed by neighboring vertices.
2. Every vertex in G is in V(vertices), ensuring that each node has been visited.

Proof: To show that the TSP is NP hard, we need to prove that NP reduces TSP in the polynomial time. Let us prove this with the help of Hamiltonian Cycle. Every HC is NP complete, thus HC is NP hard and every problem in NP reduces to HC in polynomial time. Further if we reduce HC to TSP in polynomial time, then we will show every vertex in NP reduces to TSP in polynomial time, since the sum of two polynomials is also a polynomial.

In the solution # 3, we have shown that given a graph G = (V, E), does there exist a simply cycle in G that traverses every vertex exactly once.

Now, let us observe that a simple cycle on n vertices has n edges. To reduce HC to TSP following algorithm is used,
1. Take G = (V, E), set all edge weights equal to 1, and let k = |V| = n, that is, k equals the number of nodes in G.
2. Any edge not originally in G then receives a weight of 2 (traditionally TSP is on a complete graph, so we need to add in these extra edges)
3. Then pass this modified graph into TSP, asking if there exists a Tour on G with cost at most k. If the answer to TSP is YES, then HC is YES. Likewise, if TSP is NO, then HC is NO.

As we have mentioned that reduction takes polynomial time, now, we further need show that solutions for HC are in 1-1 correspondence with solutions to TSP using the reduction.

There are 2 directions in which this can be solved,

1. HC has a YES answer => TSP has a YES answer
If this is the case, then there exists a simple cycle C that visits every node exactly one, thus C has n edges. Every node weigh 1 and k=n given that there is a tour of weight n. Thus, this proves that TSP has YES answer.

2. HC has a NO answer => TSP has a NO answer
If this is the case, then there does not exist a simple cycle C that visits every node exactly one, thus C has n edges. Suppose that TSP has yes answer. Then there is a tour that visits every vertex once with weight at most k. Since the Tour requires every node be traversed with node weigh 1 and k=n given that there is a tour of weight n. This shows that the edges are in HC graph. And here is the contradiction of forming an HC.

Thus, TSP is both in NP and NP-Hard, we have that TSP is NP-Complete, as required.

Q5. Suppose we have a Monte Carlo algorithm, A, and 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.
A Monte Carlo algorithm, A and a deterministic algorithm, B, for testing if the output of A is correct if it outputs true and vice-versa for false based.

For Las Vegas algorithm, it can output ? or fail , but if it produces an output, it must produce the correct output.

Here we must design Las Vegas algorithm from A and B. Monte Carlo algorithm suggests true-biased. Let us assume that algorithm B is false-biased.

First run the algorithm A. If the output true, which is guaranteed to correct as A is true-biased. Otherwise, run algorithm B and if it outputs false, which is again guaranteed to be correct because B is false-biased. If A doesn't output true, and B doesn't output false, output ? , which is that the algorithm has failed to find the correct answer.

The total time this takes is, one call for A, one call for B, and some small additional time for bookkeeping.

For calculating the success probability, which is 1⁄2 given, notice that the algorithm succeeds if A outputs true, or if A outputs false and then B outputs false and vice- versa. The latter case happens with probability p and the former happens with the probability (1 - p) q.

This the running time will be $r = p + (1 - p) q$.

Q6. 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.

Let S be the n interval of the form [a, b], where a < b. For this we need to construct priority search tree using builtPST(S), S is the sequence of 2 - dimensional item sorted by x coordinates.

First, we may reduce this to a 2 - dimensional problem by saving intervals [a, b] as point (a, b), indicating that the data is 2 - dimensional.

When we search for a number x, if x >= a and x = b, x is in [a, b].

As a result, we may adapt the priority range tree (PRT) to this situation. The priority search tree T for S is defined recursively as follows:

- The topmost item p and the median x-coordinate x are stored in Root T.
- The priority search tree for SL is T's left subtree.
- The priority search tree for SR is T's right subtree.
- PRT also provides an O(log n + k) range search tree.

Q7. Given a set P of n points, design an efficient algorithm for constructing a simple polygon whose vertices are the points of P.

Sol.

We must use Graham's scan algorithm to create a simple polygon whose vertices are the points of P.

Let there be a range of 0 to n - 1 points,

Algorithm:

- To begin, locate the y - bottommost coordinate's point. If two points with the same y-coordinate value are located, choose the one with the smaller x-coordinate value. Make the lowest point A0, then place it in the first position in the output hull.
- Then, in counterclockwise sequence around points [0], sort the remaining n - 1 points by polar angle. If the polar angles of the two points are equal, the closest point should be placed first.
- After that, check to see if two or more points have the same angle. If so, choose the one that is farthest away from A0. Let the array's new size be m.
- Return if m3.
- Otherwise, make an empty stack and add the points p[0], p[1], and p[2] to it.
- After that, identify the fourth vertex by processing the remaining points.
- Remove points from the stack until it is not oriented counterclockwise. Then, a. point near to the stack's top, b. point at the stack's top, and c. push points.

The Graham’s Scan algorithm will take O(n log n) time for n input points.

Q8. 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.

Let T be a DNA text string of length n. We can construct O(n) time method for finding the longest prefix of T that is substring of the reversal of T (T') as follows,
1. First, we need to calculate the reversal of T, that is T'.
2. We should create a suffix tree (compressed trie) S for T' in O(n) time complexity.
3. Use suffix tree S to calculate the longest prefix of T.

This method can be completed in O(n) time.

Q9. 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.

Sol.

Step 1:
We can rewrite the linear program into the initial slack form as:
$z = 18x_1 + 12.5x_2$
$X_3 = 20 - x_1 - x_2$
$X_4 = 12 - x_1$
$X_5 = 16 - x_2$
$x_1, x_2, x_3, x_4, x_5 >= 0$

The solution is $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 20, 12, 16)$ and its objective value is $z = 0$. We choose to increase the value to x_1

Step 2:
$z = 216 - 18x_4 + 12.5x_2$
$X_1 = 12 - x_4$
$X_3 = 8 - x_2 + x_4$
$x_1, x_2, x_3, x_4, x_5 >= 0$

The solution is $(x_1, x_2, x_3, x_4, x_5) = (12, 0, 8, 0, 16)$ and its objective value is $z = 216$. We choose to increase the value to x_1

Step 3:
$z = 316 - 5.5x_4 - 12.5x_3$
$X_1 = 12 - x_4$
$X_2 = 8 + x_4 - x_3$
$X_5 = 8 - x_4 + x_3$
$x_1, x_2, x_3, x_4 >= 0$

The solution is $(x_1, x_2, x_3, x_4, x_5) = (12, 8, 0, 0, 8)$ and its objective value is $z = 316$.

Now that all the coefficients in the objective function are negative. Thus, the final solution is 316.