Assignment 4

Due: February 3, 2022

Your solutions must be typed (preferably typeset in LaTeX) and submitted as a PDF through Canvas at the beginning of class on the day its due. When asked to provide an algorithm you need to give well formatted pseudocode, a description of how your code solves the problem, and a brief argument of its correctness.

Problem 1: Highway Safety [10 points] As is well-known, America's highway infrastructure is crumbling. Yet travel must continue. Suppose you are given a map of U.S. cities and roads connecting them that shows, for every road segment, the probability of traveling down that segment *safely*, i.e. without destroying your axle in a pothole, falling into a river due to a broken bridge, etc. Design and analyze an algorithm to determine the safest route from Portland to your preferred summer vacation spot.

Stated more formally, suppose you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The safety of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.

Answer: -

Since we need the safest path, the safety probability *p(e)* should be maximum. This implies that the product of the safety probabilities of edges in the path should be maximum. If we need maximum products, we should have edges with maximum probabilities. If we modify Dijkstra's algorithm to choose edge with maximum probability every time instead of minimum, we can say that our spanning tree will always have edges with maximum weights. This will make sure that product of probabilities of edges in such a spanning tree will always be maximum, and thus it will always be one of the safest paths.

```
MAX-DIJKSTRA (G, w, s)
1       INITIALIZE-SINGLE-SOURCE (G, s)
2       S = Φ
3       Q = G.V
4       while Q ≠ Φ
5               u = EXTRACT-MAX (Q)
6               S = S U {u}
7               for each vertex v ∈ G.Adj [u]
8                       RELAX (u, v, w)
```

Note: We will multiply all the weights instead of adding them.

The two main operations involved are: EXTRACT-MAX() and RELAX(). If we use binary heap, it takes O (log V) time for EXTRACT-MAX() and RELAX(). EXTRACT-MAX() operation will be executed V times in worst case. However, RELAX() operation will be executed at most 2 E times because every edge will be considered twice, once from each side. Therefore, time complexity will be O ((V + E) log V). Now, if we use Fibonacci heap, we can execute RELAX() in O (1) time. Therefore, the overall time complexity will be O ((V log V) + E).

This algorithm can be explained as below: -

1) The first step is to mark all the nodes as unvisited.
2) Assign to every node a tentative safety value: set it to 1 for our initial node(s) and 0 for all other nodes. Set the initial node as current
3) For the current node, consider all of its unvisited neighbours and calculate their safety probabilities through the current node. Compare the newly calculated safety probabilities to the current assigned value and assign the largest one. For example, if the current node *A* is marked with a safety probability of 0.6 and the edge connecting it with a neighbour *B* has a safety probability 0.2, then the safety to *B* through *A* will be 0.6 * 0.2 = 0.12. If B was previously marked with a probability smaller than 0.12 then change it to 0.12. Otherwise, keep the current value.
4) When we are done considering all of the neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5) If the destination node (t) has been marked visited, stop.
6) Otherwise, select the unvisited node that is marked with the smallest tentative safety probabilities, set it as the new "current node", and go back to step 3.

Problem 2: Restaurant Placement A new restaurant chain is opening, and you have been given the task of selecting the restaurant locations with the goal of maximizing their total profit.

The street network is described as an undirected graph $G = (V, E)$, where the potential restaurant sites are the vertices of the graph. Each vertex v has a non-negative integer value p_v, which describes the potential *profit* of site v. Two restaurants cannot be built on adjacent vertices. You are supposed to design an algorithm that outputs the chosen set $U \subseteq V$ of sites that maximizes the total profit $\sum_{v \in V} p_v$.

For parts (a)-(c), suppose that the street network G is *acyclic*, i.e. a tree.

(a) [5 points] Consider the following *greedy* restaurant placement algorithm. Choose the highest profit vertex v_0 in the tree, breaking ties according to some ordering on vertex names, and put it into U. Remove v_0 and all of its neighbors from G. Repeat until no vertices remain. Give a counterexample to show that this algorithm does not always give a restaurant placement with maximal profit.

Answer: - For instance, if we have slightly less profit value adjacent to the node with maximum value then this program will fail.

For example, if we have maximum profit value as 50 and its adjacent node's profit values are 48 and 49, the tree has 3 elements (48-50-49), then we will select 50 and skip 48, 49 because they are adjacent to 50. Thus our max profit will be 50 rather than 97 (48+49), which is incorrect. The correct maximum profit should be 97.

(b) [10 points] Give an efficient algorithm to determine a placement with maximum profit.

1) Pick any node n and consider it as the root of the tree
2) Sort all the nodes using DFS (Depth First Search)
3) Store the sorted nodes in an array N
4) For each node n, calculate A(n) and B(n)
5) For each n in N, in the reverse order, compute A(n) and B(n)
6) max($A(n_0)$, $B(n_0)$) is the maximum profit
7) Please note $A(n) = p_n$, where p_n = profit obtained at that node and B(n) = 0 if n is a leaf node
8) If n is not a leaf node, $A(n) = p_n + \Sigma_{u \in n.children} B(u)$
$B(n) = \Sigma_{u \in n.children} \max(A(u),B(u))$

Time Complexity: - Here we use DFS to sort the nodes which will take O(n) time.
The time for computing the values of A(n) and B(n) will depend on the degree of the node we are present on. So, the total time is the sum of the degree(s) of node(s) i.e., O(|E|) where E is the number of edges.

In a tree with n vertices, there are n-1 edges which means E=n-1.

The time to compute A() and B() is O(n). We visit every node and use A() and B() to find the optimal placement which takes O(n) . This makes the time complexity for this as O(n).

(c) [10 points] Suppose that, in the absence of good data, the restaurant chain decides that all sites are equally good. The goal therefore is to simply find the placement with the maximum number of locations. Give a simple greedy algorithm for this case and argue its correctness.

Answer : -

Algorithm

1) Pick any node n_0 as the root of the tree and sort all nodes using DFS and store the sorted nodes in an array N
2) In reverse order, include each valid node and remove its parent

Here we use DFS to sort the nodes which will take O(n) time. Since each node is processed here, the time complexity comes out to be O(n).

Proof of Correctness

Because of DFS sorting, in reverse order, first element of an array must be a leaf node. It can be included and its parent excluded. When nodes from an array are processed in reverse order, each processed node is the last valid one in current N which is a leaf node. Also, optimal solution of each subtree should include leaf. Therefore, this algorithm will always return an optimal solution.

(d) [5 points] Suppose that the graph is arbitrary and not necessarily *acyclic*. Give the fastest correct algorithm you can for solving the problem.

Answer: -

1) Greedy algorithm could be: - try all possible subsets of vertices
2) Test whether every edge has only one element (we are eliminating all the neighbours)
3) Compute the total profit for this solution
4) Repeat steps 2 and 3 for all possible sets and select the set of vertices with maximum value computed at step 3

This can be explained as below: -
In this solution, we should try all the possible subsets of the given vertices. We must then test the independence property i.e., eliminating the neighbors and not taking both the nodes. This means including only one node which makes the number of edges to be n-1.
The total profit for every valid solution is taken and then the best solution is computed from among them.