CS 4150: Homework 4
More Dynamic Programming, Graphs

Submission date: Friday, Oct 25, 2024 (11:59 PM)

This assignment has 5 questions, for a total of 40 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Note. *When asked to describe and analyze an algorithm, you need to first write the pseudocode, provide a running time analysis by going over all the steps (writing recurrences if necessary), and provide a reasoning for why the algorithm is correct.* Skipping or having incorrect reasoning will lead to a partial credit, even if the pseudocode itself is OK.

Question 1: DFS and BFS in Graphs .... **[8]**

For the following graph problems, design linear time algorithms (i.e., running time $O(|V|+|E|)$ or $O(n+m)$ as we have been calling it). You should describe each algorithm in English, using around 2-3 bullets. **Do not write code.** You can use BFS and DFS as subroutines, without describing them. For each bullet, write down the running time.

In each case, assume that the graph is given to you as an adjacency list.

The goal of this question is for you to learn to think in terms of using standard graph algorithms as "primitives".

For `DFS(u)`, the output will be a boolean array `isReachable[]`, where `isReachable[v]` is true if v is reachable from u via a (directed) path and false otherwise. For `BFS(u)`, the output will be an integer array `dist[]`, where `dist[v]` denotes the minimum number of edges in a path from u to v. `dist[v]` will be `INFINITY` if v is not reachable from u.

(a) **[5]** Suppose G is an **undirected** graph, and let uv be an edge in G. Determine if there is a cycle involving (i.e., containing) the edge uv in G.

(b) **[3]** Suppose G is a **directed** graph and let u be one of the vertices. Let r be a given parameter. Determine if exist paths of length $\leq r$ from u to every other vertex in G.

Question 2: The Good Host .... **[8]**

You are hosting a party for n people (indexed $1, 2, \ldots, n$) and you have booked two banquet halls. As it turns out, there are some some pairs of people that simply do not see eye-to-eye and they refuse to be in the same hall at the same time. Given a list of pairs $\{i, j\}$ that don't get along, your goal is to find a partitioning of the n people into two halls so that within each hall, there is no pair that does not get along. (If this is impossible to achieve, your algorithm must output `impossible`.)

Give an algorithm for this problem that runs in time $O(m+n)$, where m is the number of pairs $\{i, j\}$ of people who do not get along.

Important. To receive full credit, you must describe your algorithm in pseudocode and give an explanation for why it runs in $O(m+n)$ time.

[*Hint:* Form an appropriate graph and try a BFS-like procedure.]

Question 3: Fast Ancestry .... **[10]**

It turns out that DFS is surprisingly powerful, especially if one keeps track of "visit times" for each vertex. Consider the following subroutine that uses a global variable t, in addition to the reachable[] array, and two new arrays that maintain the start and finish times for each vertex:

```
boolean reachable[n]   // dimension = #(vertices) = n
int start_time[n], finish_time[n]
int t = 0

Procedure TimeDFS(vertex u):
  set reachable[u] = true
  set start_time[u] = t
  t = t+1
  for (j in neighbors(u)):
    if (reachable[j] == false):
      TimeDFS(j)
    end if
  end for
  finish_time[u] = t
  t = t+1
end Procedure
```

(a) [**4**] Take a complete binary tree of depth 3 (so it has 7 vertices), draw the tree and write down the `start_time` and `finish_time` values for each of the vertices.

(b) [**6**] Using the procedure above, solve the following problem from the textbook: given a tree with n vertices and root node r, find a way to pre-process the tree so as to answer "ancestry" queries in $O(1)$ time. An ancestry query asks: is vertex u an ancestor of vertex v? This should return `true` if u appears on the path from v to the root and `false` otherwise. [As is standard, a tree is given as a set of vertices; for each vertex, we have the index of its parent, and a list of all its children. Of course, the parent of the root r is null.]
[**Important.** Give an (informal) explanation as to why your solution is correct.]

Question 4: Scheduling with Constraints .... **[9]**

Your company's CEO is visiting your local office, and you would like to schedule a meeting with the CEO for each of the employees at the office. Suppose the employees are numbered $1, 2, \ldots, n$. Given the social dynamics at your office, you have some constraints; each constraint has the form: person i's meeting (with the CEO) must happen before person j's.

Luckily, you know some graph theory, so you represent people in your office as vertices $1, 2, \ldots, n$, and each constraint as above is represented as a directed edge from i to j. You are given a collection of m such constraints, and **your goal** is to find a valid meeting schedule, i.e., a valid ordering of $1, 2, \ldots, n$, such that all the constraints are satisfied.

Notice that if the graph constructed as above has a cycle, it is impossible to find a valid schedule. So you can assume that the graph does not have a directed cycle.

(a) [**4**] Prove that if a directed graph has no (directed) cycle, then there must exist a vertex of *in-degree* equal to zero.

[*Hint:* Try to prove by contradiction – what happens if every vertex has in-degree ≥ 1?]

(b) [**5**] Use the observation above to design an algorithm for finding a valid schedule. [You do not need to analyze its running time, but in order to get credit, it **must not be exponential** in n.]

Question 5: Shortest Paths and Negative Edges .... [**5**]

We discussed in class that Dijkstra's algorithm requires all the edge weights to be non-negative in order to work. A student suggests a simple fix. Given a graph G with some negative weight edges, he suggests computing the least weight $w_{\min}$ (which will be negative), and then adding $|w_{\min}|$ to all the edge weights. This would make all the weights non-negative. He claims that the shortest path in this new graph also yields the shortest path in G ("because the weights are just shifted", he says..).

Is there something wrong in the reasoning above? Explain with an example.