CS 3510 C: Design & Analysis of Algorithms 09/21/2023

Practice Exam 2

Abrahim Ladha

- This is the CS 3510 practice exam for Exam 2. This does **not** approximate the difficulty or length of the actual exam; it serves as just a big question bank for you to practice!
- Topics include: Graphs, DFS, Topological Sort, Strongly Connected Components, BFS, Dijkstra's, MST, Kruskal's, Max Flow Min Cut
- Note that this assignment does not need to be submitted, but we highly recommend going through it to prepare for the exam!

1.) Determine whether the following statements are true or false. Give a brief explanation if true, or a counterexample if false.

(a) If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge can not be part of the MST.

◯ True

√ **False**

Consider two components which are only connected by this unique heaviest edge. Since the spanning tree must contain a path to all vertices, we are required to take this edge despite its weight.

(b) If G has a cycle with a unique heaviest edge e, then e can not be part of any MST.

√ **True**

◯ False

Suppose this unique heaviest edge $e = (u, v)$ is in the MST T. Then consider removing e, splitting the MST into two parts, S_u and S_v. Since e lies in a cycle in the original graph, there's still a path from u to v even after we remove e, meaning there's another edge (on this path, that was part of the cycle) that cuts across S_u and S_v. Since e was the unique heaviest edge in the cycle, replacing e in T with this other edge fives a spanning tree of lower weight, meaning T couldn't have been a MST.

(c) Let e be any edge of minimum weight in G. Then e must be part of some MST.

√ **True**

◯ False

Edge e must be the minimum cost edge of some cut. In this cut we know that the minimum cost edge will be included in some MST based on the cut property. Since e is the smallest edge of the entire graph, this edge must be the minimum cost of the subset of edges that cut the graph, and we will pick this edge e.

(d) If G has a cycle with a unique lightest edge e, then e must be part of every MST.

◯ True

√ **False**

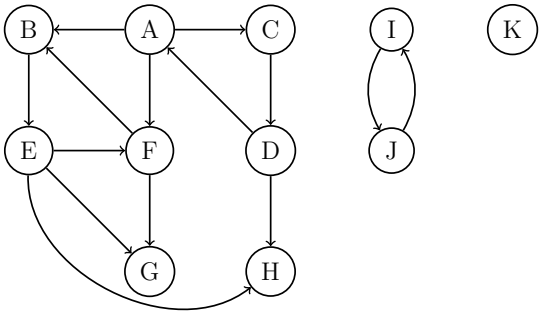
Consider a graph with vertices $\{A, B, C, D\}$, and edges with edge weights $(A, B, 5)$, $(B, C, 2)$, $(C, D, 1)$, $(D, A, 4)$, $(B, D, 3)$. Consider the cycle A -¿ B - D -¿ A, where edge (B, D) of weight 3 is the unique lightest edge. However the MST does not contain this edge.

(e) The shortest-path tree computed by Djikstra's algorithm is necessarily an MST.

◯ True

√ **False**

Consider a graph with vertices $\{A, B, C, D\}$, and edges (A, B), (B, C), and (C, D) each with a weight of 1. We also have edge (A, D) with a cost of 2. MST will take edges (A, B), (B, C), (C, D) whereas Djikstra's will take edges (A, B), (B, C), (A, D).



2.) Given the above graph, answer the following questions about depth first search algorithms. All exploration should be done in alphabetically order, and start from A unless otherwise stated.

(a) After running explore, in what order would the nodes be found.

- ◯ A, B, E, F, G
- ◯ A, B, E, F, G, H, C, D, I, J, K
- √ **A, B, E, F, G, H, C, D**
- ◯ A, B, E, F, G, H, C, D, J, I, K

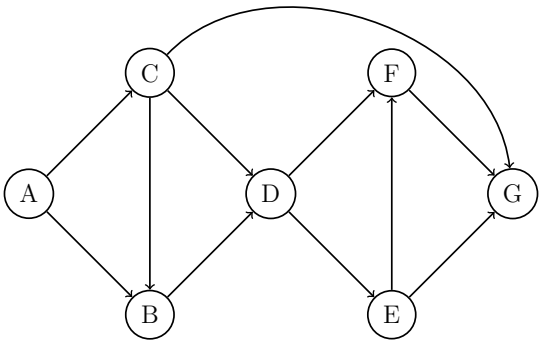
(b) After running DFS, taking note of pre/post labels, which node has the lowest post label?

- ◯ A
- √ **G**
- ◯ K
- ◯ H

(c) After running DFS, and sorting descending by post labels, which ordering would you get?

- ◯ K, I, J, A, B, E, F, G, C, D, H
- ◯ I, J, K, A, B, E, F, G, C, D, H
- ◯ A, B, E, F, G, C, D, H, K, I, J
- ◯ A, B, E, F, G, C, D, H, I, J, K

Sorry, there was a typo, the correct answer is **K, I, J, A, C, D, B, E, H, F, G**.



3.) Given the above graph, answer the following questions.

(a) In the metagraph of the above graph, which node will be in a strongly connected component which leads to the sink.

- ◯ A
- ◯ B
- √ **F**
- ◯ G

(b) If the graph was changed such that all the edges were removed, what would the number of edges be that you'd need to add such that the entire graph is strongly connected.

- ◯ 6
- √ **7**
- ◯ 8
- ◯ 9

With seven nodes, six edges would make some kind of tree. A minimal but strongly connected graph will look like a cycle, which takes seven.

(c) What is the minimum number of edges which would need to be added to have two strongly connected components.

- ◯ 0
- √ **1**
- ◯ 2
- ◯ 3

4.) Let $G = (V, E)$ be a directed graph in which each vertex $u \in V$ is from the set $\{1, 2, ..., |V|\}$. For each vertex in $u \in V$, let $R(u)$ be all the vertices that are directly reachable from u. $\min(u)$ is defined as the vertex in $R(u)$ which is minimum, i.e., $\min(u)$, for $u \in V$ is the vertex v such that $v = \min\{v \in R(u)\}$. Design an algorithm in $\mathcal{O}(|V| + |E|)$ time that computes $\min(u)$ for all vertices $u \in V$. Justify its correctness and show that its runtime is linear. **Hint: since the vertex labels are integers, think of linear time sorting algorithms.**

Solution: Take the reverse of the graph as G^R. Sort the vertices in ascending order by their label using a linear-time sorting algorithm. Run DFS starting from the node with the smallest label with the following modification: when explore is called, record the number that is being explored from, and set all the new vertices seen to have the minimum be that number. Do all exploration in ascending order. This ensures that any vertex u which is reachable from the smallest labels first will be updated accordingly, and larger vertices will not overwrite $R(u)$.

This algorithm reverse the graph which takes linear time and runs DFS which is linear time.

5.) Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

Algorithm:

1. First run `SCC` and verify that the number of strongly connected components is n, that is, there are no cycles. If any cycles are found, then it is impossible to complete the curriculum since a course would be its own prerequisite.

2. In the case no cycles are found, create a new vertex s. Add a new edge from s to v for every source vertex v and run Breadth-first `explore` from s.

3. The depth of the last vertex visited is the same as the minimum number of semesters needed to graduate.

Running Time:

Running `SCC` takes linear time, determining the source vertices can be done in linear time, and running breadth-first `explore` takes linear time. Altogether this algorithm takes linear time.

6.) Suppose you want to find the shortest path from u to v in a graph G that passes through a specified vertex w. Design an algorithm for each of the following settings.

(a) G is an undirected weighted graph.
Algorithm:

1. We run Dijkstra's algorithm once, starting from w, and then get the total weight/path by looking at $dist[u]$ and $dist[v]$, since the shortest path from u to w is the same as the shortest path from w to u in an undirected graph.
2. Then add the distances $dist[u]$ and $dist[v]$. That should be the shortest distance between u and v passing through w.

Running Time:
This has running time $O((|V| + |E|)log|V|)$, since we just call Dijkstra's once.

(b) G is a directed weighted graph.
Algorithm:

1. Now that the graph is directed, the shortest path from u to w isn't the same as the shortest path from w to u.
2. Instead, we run Dijkstra's twice: once starting at u and once starting at w.
3. Then, we concatenate the shortest path from u to w (from the first run) with the shortest path from w to v (from the second run).

Running Time:
This still has running time $O((|V| + |E|)log|V|)$, since we just call Dijkstra's twice.

(c) G is a directed weighted graph, and instead of a single u and v, we have k queries $(u_1, v_1), (u_2, v_2), ..., (u_k, v_k$ where for each one we want to find only the weight of the shortest path that passes through w (which is fixed).
Algorithm:

1. Since we want to answer queries quickly, repeating the solution from b.) wouldn't be very fast, as it requires us to run Dijkstra's again for each u_i.
2. Instead, we run Dijkstra's twice: once on G, starting from w, and once on G_R, the reverse graph, also starting from w.
3. Now, we can efficiently find the shortest path from u_i to w for any u_i by looking at the shortest path from w to u_i in the reverse graph.
4. Then, we add $dist[u]$ (from the result of Dijkstra's on G^R starting at w), and $dist[v]$ (from the result of Dijkstra's on G starting at w) for each pair of (u_i, v_i).

Running Time:
Thus, we can answer each query as $dist_{G^R}[u_i] + dist_G[v_i]$ in $O(1)$ after $O((|V| + |E|)log|V|)$ preprocessing, for total running-time $O((|V| + |E|)log|V| + k)$.

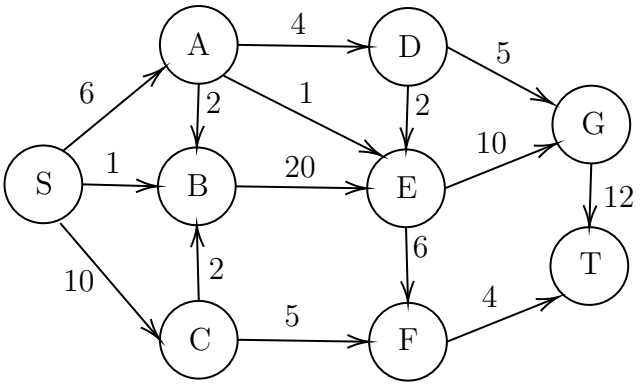
7.) Suppose you live in a village with n houses and you want to build wells and lay pipes to connect water to all houses with the minimum cost necessary. You're given a list of the costs of building wells for each house $w_1, w_2, ..., w_n$. You can also lay bidirectional pipes between certain houses given as a list of the two houses and the cost to lay the pipe. For example, one such entry in the list would be (h_1, h_2, c_{12}), which tells you that laying a pipe between house 1 and house 2 would cost c_{12}. Give an efficient algorithm to find the minimum total cost to connect water to all houses, and show its runtime.

Solution: Layout a graph with vertices as houses and an additional vertex labeled as water. Now create edges between houses where pipes can be built with weight as the cost to build the pipe, and create edges from the water vertex to houses where the weight is the price to build a well for that house. Run Kruskal's algorithm and give the MST.

The goal is to create the least weighted graph to connect all houses to water; by our construction this will happen because all the vertices in the graph will be connected by either building a pipe or a well to the house. No more connections are used than needed because it is a spanning tree, and the weight or cost is minimum because an MST was created.

This algorithm takes $\mathcal{O}(|E| \log |E|)$.

8.) Consider the network shown below.



(a) What is the maximum flow from S to T? List the flow associated with each edge in the following format: (u, v, f_e) where $u \in G, v \in G$ and f_e is the flow along the edge $u \to v$.

The maximum flow from S to T is of size 13. Here are the edges corresponding to the max flow (there are multiple possibilities):
(S, A, 6), (S, B, 1), (S, C, 6), (A, B, 2), (C, B, 2), (A, D, 4), (A, E, 0), (B, E, 5), (C, F, 4), (D, E, 0), (D, G, 4), (E, F, 0), (E, G, 5), (F, T, 4), and (G, T, 9).

A few other possibilities:
Other edges same as above, except (A, B, 1) and (A, E, 1), (B, E, 4).
Other edges same as above, except (A, D, 3), (A, E, 1), (D, G, 3) and (E, G, 6).
Other edges same as above, except (A, D, 3), (A, E, 1), (D, E, 2), (D, G, 1) and (E, G, 8).
Other edges same as above, except (A, D, 3), (A, E, 1), (D, E, 1), (D, G, 2) and (E, G, 8).

(b) Find a matching minimum cut. Clearly list the edges in the minimum cut.

The edges in the minimum cut are $S \to A$, $S \to B$, $C \to B$, and $F \to T$.

9.) You've become the head industrial engineer at a company operating on n cities numbered 1 to n along a one-way road. The ith city produces $p_i \in P$ units of goods, and can buy $s_i \in S$ units of goods from other cities. For each pair of cities i and j, such that $1 \leq i \leq j \leq n$, you can transport at most c units of goods from city i to city j. Design an efficient algorithm to determine the maximum number of goods that can be sold in total to all cities after a sequence of transportations given n, c, P, and S. You do not have to prove its runtime.

Solution: Construct a network where there is a source, sink and node for each city. For each city, have one edge from the source with capacity p_i, and one edge to the sink with capacity s_i. For each city draw an edge from that city to every city who's label is greater with capacity c. Run Ford-Fulkerson, and give the maximum flow.

The algorithm is correct because there are limits on how much cities can produce and consume based on p_i and s_i, and how much they can transport. Then we care about the maximum which running Ford-Fulkerson will give us.