FINAL EXAM
CSI 503 Spring 2021

RAHUL BHENJALIA

May 15 2021

1 ANSWER: 1

In the case of adjacency list, we can efficiently iterate over all edges in $O(E)$ time, and get the the worst case complexity $O(V.E)$ in total.

.

In the case of adjacency matrix, we cannot iterate over all edges as efficiently, but we could iterate over all entries in the matrix, and check for each one if the edge exists (that is check if the distance stored is infinity), however that will require $O(V^2)$ time and therefore lead to a total complexity of $O(V^3)$.

2 ANSWER: 2

1. **TRUE**. Let's assume the contradiction, that there is not a unique light edge crossing any given cut. That situation would result in every node having at least two edges. Hence, while building a unique MST, a leaf node would have only one possible edge, but it would not be a unique MST. Hence, TRUE.

2. **FALSE**. Since Prim's algorithm is a greedy algorithm (going for the smallest edge), it does not matter what the starting node is, as long as the algorithm is applied correctly.

3. **TRUE**. We can consider following when we show that a MST has exactly $|V|-1$ edges: when a tree with only one vertex (zero edges), if we decide to add another vertex to it we need to connect then two by using an edge. So the amount of edges will always be one less than the amount of vertices.

.

.

.

.

3 ANSWER: 3

ANSWER 3
FOR $k = 0$

$D^{(0)}$	v_1	v_2	v_3	v_4
v_1	0	∞	∞	∞
v_2	∞	0	∞	∞
v_3	∞	∞	0	∞
v_4	∞	∞	∞	0

FOR $k = 1$

$D^{(1)}$	v_1	v_2	v_3	v_4
v_1	0	2	∞	∞
v_2	∞	0	-4	∞
v_3	1	3	0	-10
v_4	∞	∞	∞	0

FOR $k = 2$

$D^{(2)}$	v_1	v_2	v_3	v_4
v_1	0	2	-2	∞
v_2	∞	0	-4	∞
v_3	1	3	-1	-10
v_4	∞	∞	∞	0

FOR $k = 3$

$D^{(3)}$	v_1	v_2	v_3	v_4
v_1	0	2	$-\infty$	∞
v_2	∞	0	-4	∞
v_3	1	∞	0	-10
v_4	∞	∞	∞	0

When we reach $k = 3$, the Floyd-Warshall algorithm keeps updating the shortest path from v_1 to v_3 with even smaller weight. Therefore we can never reach $k = 4$ and such table for $D^{(4)}$ does not get filled.

.
.
.
.
.
.
.
.
.

4 ANSWER: 4

First, we construct G' by splitting each vertex v of G in three vertices: v_1: where all incoming edges are now incoming edges to v_1. v_2: where all outgoing edges are now outgoing edges from v_2. And they are joined by an edge capacity $l(v)$

More clearly, we can say $G' = (V', E')$ has capacity function which is defined as follows:

- For every $v \in V$ we are creating two vertices in $V' -> v_1, v_2$
- We add an edge (v_1, v_2) in E' with $c'(v_1, v_2) = l(v)$
- So, for every edge $(u, v) \in E$, we create an edge (u_2, v_1) in E' with capacity $c'(u_2, v_1) = c(u, v)$
- We introduce s_1 and t_2 as new source and target vertices in G'
- Clearly, from this we can say $|V'| = 2|V|$ (number of vertices)and $|E'| = |E| + |V|$ (number of edges), hence this answers our question in Part 1(b)

Now for Part 1(a), let f be a flow in G that respects vertex capacities. Now, we create a flow function f' in G' with following conditions:

- For each edge (u, v) belongs to G let $f'(u_2, v_1) = f(u, v)$.
- For each vertex $u \in V – t$, let $f'(u_1, u_2) = \Sigma_{v \in V} f(u, v)$.
- Also, let $f'(t_1, t_2) = \Sigma_{v \in V} f(v, t)$
- So without hesitation we can say that there is one to one correspondence between flows that respect vertex capacities in G and flows in G'.
- If we look at capacity constraint, every edge in G' of the form (u_2, u_1) has a corresponding edge in G with corresponding capacity and flow and thus, satisifies capacity constraint.
- Therefore, for $u \in V – s, t$ we have $f'(u_1, u_2) = \Sigma_{v \in V} f(u, v) \leq l(u) = c'(u_1, u_2)$.
- We also have $f'(t_1, t_2) = \Sigma_{v \in V} f(v, t) \leq l(t) = c'(t_1, t_2)$

.
.
.
.
.
.
.

5 ANSWER: 5

The algorithm below runs by exploring a pair vertices at a time so we can say that time complexity of the algorithm is $O(V^2)$
First the algorithm considers a pair of vertices in the graph G (a, b in this case)and make edges (u, a) and (v, b).
Then we create a transitive closure for the new graph that contains the new vertices (u, v).
Now, the only way the solution exists is if the transitive closure for the graph contains edges (u, a) and (v, b).
.
.
ALGORITHM:
UpdateClosureAfterInsert(G, G*, u, v)
1. //Input: A Graph G = (V,E)
2. for every pair (a,b) of vertices in V do
3. Let H = G (copy of original graph)
4. createEdge-in-H – > (u,a) //append to H graph
5. createEdge-in-H – > (b,v) //append to H graph
6. result = CreateTransitiveClosure(H)
creating Transitive closure of new graph H which has added edges
8. if result has edges(u,a) and(b,v) then do
9. return result
10. else do
11. return ”No Result”

6 ANSWER: 6

PART(a)

Optimization problems are those problems in which each feasible solution has an associated value, and we wish to find the feasible solution with the best value. In the case of maximum clique optimization problem, it is defined as finding a maximal clique, where every clique is contained in a maximal clique.
.
Decision problems are those where the answer is simply ”yes” or ”no”, for example the CLIQUE problem, where we have to determine whether a graph G contains a k-vertex clique, and find any such clique that it contains, using a brute force algorithm..
.
Now the fact is that NP-completeness does not directly apply to optimization problems, but it is directly applied to decision problems. Although proving that a given problem is NP-complete binds us to the area of decision problems, there is a convenient relationship between optimization problems and decision problems. We usually can cast a given optimization problem as a related decision problem by imposing a bound on the value to be optimized.

.
In the case of CLIQUE and MAX-CLIQUE-OPTIMIZATION, where applying a simple greedy strategy to CLIQUE problem can give us the MAXIMAL CLIQUE. Here's how we can do that:
.
First, we start with an arbitrary clique (for instance, any single vertex or even the empty set), continuously updating the current clique one vertex at a time by looping through the graph's remaining vertices. For each vertex v that our loop examines, add v to the clique if it is adjacent to every vertex that is already in the clique, and discard v otherwise.

PART(b)

From above example, we can say that the relationship between an optimization problem and its related decision problem works in our favor when we try to show that the optimization problem is difficult. That is because the decision problem is in a sense easier, or at least no harder.

We can state in a way that has more relevance to NP-completeness, if we can provide evidence that a decision problem is difficult, we also provide evidence that its related optimization problem is more difficult. Thus, even though it restricts our attention to decision problems, the theory of NP-completeness often has implications for optimization problems as well.

7 ANSWER: 7