CSCI8080 Final Exam Spring 2020

Nick Palacio

Problem 1 (10 points)

Company TR's employees are organized in a strict hierarchy with the CEO as the root of the tree. The children of a node N are all supervised by N.

Each employee E in TR is assigned a positive number, EV[E], that measures how valued he/she is, in TR. We want to find a set S of employees that with the total maximum T value with the following conditions: (i) The CEO is always in the set regardless of her value, and (ii) If an employee is in the set, then her immediate supervisor is not in the set.

Design an algorithm that computes S and T given the employee hierarchy of TR using a dynamic programming based approach.

Answer

The only algorithm I can come up with to solve this problem is brute force. Here is what you would do

- Since we know that the CEO is included we can also say that all of their direct reports must be excluded.
- Starting at the 3rd level down we need to calculate 2 possibilities for every single node:
    - Calculate what my max achievable value is if I include this node (employee)
    - Calculate what my max achievable value is if I exclude this node
        - When we exclude a node we may not necessarily take all of the employees that report to them. We should calculate these same 2 possibilities for each of the employees that report to them because you may have the case where a supervisor is not taken and then only 2 of the 3 employees they supervise are taken and for the 3rd employee we actually take the people that report to them instead of themselves.
- For every path where you have made decisions on each employee you would recreate the organizational tree coloring in employees you are taking as black with the employees you are not taking as white.
    - This will leave you with 2^n different organizational trees with different sets of employees taken in each and different T values.
- **You should return the tree that has the highest calculated T value**

Problem 2 (10 points)

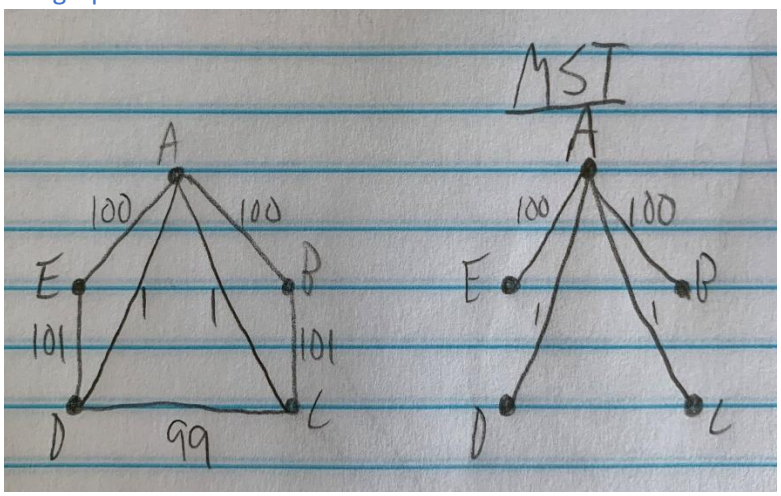
Let T be the minimum spanning tree of a graph G. Prove or disprove the following two statements.

(a) T will not contain the maximum weighted edge on any cycle in G.
(b) T will contain the minimum weighted edge of every cycle in G.

Answers

a. True, assuming this is an undirected graph with a cycle that means that there are 2 ways (edges) to get to every node. Given that every node will be included in the minimum spanning tree in the end and there are 2 ways (edges) to get to every node the MST must always use the lighter edge, by definition. This can be proven by contradiction. Let's assume we have a MST that includes the max weighted edge on a cycle in the graph. We use this edge to connect to some node N. If instead we were able to connect to node N using a lighter edge (the other edge that connects node N to the graph that we know is there because a cycle exists and is lighter because the other edge is the maximum weight edge in the cycle) then we would end up with a spanning tree that is lighter than the MST we assumed before. This obviously cannot happen meaning that any MST will not contain the maximum weighted edge on any cycle in G.

b. False, consider the graph and MST below where we have a cycle (A, B, C, D, E, A) with a minimum weighted edge of (D,C) = 99. This minimum weighted edge in that cycle is not in the MST for this graph.

a. 

Problem 3 (5 points)

Rewrite the Faster-APSP algorithm to include the predecessor matrix computation. Explain the modifications you made.

Answer

Here is the Faster-APSP algorithm from the book:

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

In order to modify the Faster APSP algorithm to include the predecessor matrix computation you would add a variable P that is the predecessor matrix with all values initialized to *nil* sometime before line 4 of Faster-All-Pairs-Shortest-Paths. You would pass that variable P into the Extend-Shortest-Paths function. Then in the Extend-Shortest-Paths function you would rewrite the for-loop iteration on line 7 to do the following

- $x = l_{ik} + w_{kj}$
- if $x < l'_{ij}$ then
    - // We can get a better path from i to j by going through k
    - $p_{ij} = k$ // lower-case-p here references an entry in the predecessor matrix P
    - $l'_{ij} = x$

There is no need for an else here because if it not an improvement to go through vertex k on the path from i to j then we just don't update the value for the shortest path weight.

Problem 4 (5 points)

Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary Boolean formula α, whether α is satisfiable. Describe and analyze a polynomial-time algorithm that either computes a satisfying assignment for a given Boolean formula or correctly reports that no such assignment exists, using the magic black box as a subroutine.

Answer

The algorithm that can satisfy a given Boolean formula in p-time using the black box described above can use the following strategy:

- Start by checking if the given Boolean formula is satisfiable. If it is not, return/report that no such assignment exists that satisfies this formula. If it is satisfiable, move onto the next step below.
- Start with 1 of the variables in the Boolean formula, set it to true. This means replacing any reference to this variable in the formula with a 'hard-coded' value of true. This produces a new boolean formula, b'.
- Use the black box to determine if b' is satisfiable.
    - If b' is not satisfiable, that means that the variable we set to true actually has to be set to false for the satisfying assignment. So set the original variable to false, this is its value in the satisfying assignment.
    - Move onto the next variable, set it to true and repeat this process on the new b'.
- Once you have assigned all variables their values for the satisfying assignment of this Boolean formula you can return that.

With the black box all we need to do is iterate over all the variables in the original Boolean expression so we can say that our algorithm runs in O(n) where n is the number of variables in the original Boolean formula.

Problem 5 (10 points)

We define the 2Sol-SAT problem as follows.

Input: A, an instance of a SAT formula (A is a conjunction of disjunctive clauses).

Output: 1 if A has at least two satisfying solutions, otherwise, 0.

(a) Show that SAT ≤p 2Sol-SAT.

(b) Show that the 2Sol-SAT problem is in NP.

Answers

a. We can show that SAT reduces to 2Sol-SAT by mapping the input of SAT to that of 2Sol-SAT in polynomial time and then showing that SAT = 1 IFF 2Sol-SAT = 1 for the original SAT input and the mapped input from that to the 2Sol-SAT problem.
    a. In order to map the input, a Boolean formula B, of SAT to 2Sol-SAT we simply add another variable, v, and the following clause to the original Boolean formula B to give us: **B AND (v OR (NOT v))**.
    b. In order to prove that SAT = 1 IFF 2Sol-SAT = 1 for the given and mapped inputs let us consider the case where SAT = 1 for B. This means there is at least 1 satisfying assignment for all the variables in B. When we map the input B as detailed above and give that to the 2Sol-SAT problem we can see that there are at least 2 satisfying assignments for the mapped Boolean formula. One with all the original assignments to B where v = True and one with all of the original assignments to B where v = False. We can also say that if there are no satisfying assignments to B then SAT = 0 and 2Sol-SAT = 0 because it doesn't matter what v is, B cannot be satisfied so that means **B AND (v OR (NOT) v)** also cannot be satisfied.
b. In order to show that the 2Sol-SAT problem is in NP we need to show that it can be verified in polynomial time. The verification algorithm for the 2Sol-SAT problem would take in the 2 satisfying solutions to input A (the Boolean formula) and verify that both satisfy the Boolean formula. This algorithm runs in p-time because all it needs to do is take both solutions, one at a time, plug in all the satisfying assignments and then compute the result and validate that the Boolean formula is satisfied. This can be done in linear time (See Lemma 34.5 that shows Circuit-SAT can be validated in linear time).

Problem 6 (10 points)

A Hamiltonian Cycle in a graph is a cycle that visits every vertex exactly once. DIRECTED-HAMILTONIANC problem checks to see if a directed graph contains a Hamiltonian cycle. UNDIRECTED-HAMILTONIANC problem does the same for undirected graphs.

(a) Describe a polynomial-time reduction from UNDIRECTED-HAMILTONIANC to DIRECTED-HAMILTONIANC.
(b) Describe a polynomial-time reduction from DIRECTED-HAMILTONIANC to UNDIRECTED-HAMILTONIAN PATH.

Answers

a. A polynomial time reduction from UNDIRECTED-HAMILTONIANC to DIRECTED-HAMILTONIANC involves mapping the inputs of UNDIRECTED-HAMILTONIANC to the inputs of DIRECTED-HAMILTONIANC in polynomial time and then showing that the outputs match for both problems given the original input and the mapping function to the other problem.
    a. You can map the input of UNDIRECTED-HAMILTONIANC (a graph, G, with edges, E, and vertices, V) to DIRECTED-HAMILTONIANC (a graph, G', with edges, E', and vertices, V') by simply taking every undirected edge (u, v) in E and creating a directed edge in both directions such that you end up with (u, v) and (v, u). The set of vertices would remain the same: G' = {E', V} such that E' contains (u, v) and (v, u) for every edge (u, v) in the undirected edges in E. This can be done in polynomial time since you only need to iterate over the edges in E.
    b. DIRECTED-HAMILTONIANC will only return true given graph G', as defined above, IFF UNDIRECTED-HAMILTONIANC will return true given graph G. This is true because the resulting directed graph, G', is equivalent to the undirected graph G since an undirected edge can be represented as 2 directed edges, one in both directions. So saying there is a Hamiltonian cycle in an undirected graph G is the same thing as saying there is a Hamiltonian cycle in another directed graph where we represent each undirected edge as 2 directed edges, one in both directions.
b. A polynomial time reduction from DIRECTED-HAMILTONIANC to UNDIRECTED-HAMILTONIANC will involve the same steps as above but going in the opposite direction. We will need to map the inputs from DIRECTED-HAMILTONIANC (a directed graph) to UNDIRECTED-HAMILTONIANC (an undirected graph) and then show that UNDIRECTED-HAMILTONIANC returns true given a graph G' IFF DIRECTED-HAMILTONIANC returns true given graph G.
    a. The only way that I can come up with to map the input here is to take the directed graph and create a set of graphs, one for each vertex. For any single vertex's graph add an edge to any vertex where, in the original directed graph, you had an edge that started at this vertex and went to another. When starting at one vertex and traveling to another you can only go to a vertex that has an edge in the graph for the vertex that we are on. When you want to see where you can go once you arrive at the destination vertex you look up that vertex's graph (tree with edges that all start from this vertex in the original directed graph). What I end up with here is mapping one directed graph to a set of undirected graphs but that was the only way I could think of to maintain the direction of the edges in the directed graph. This mapping can be done in polynomial

time since we need only iterate of the vertices of the directed graph, creating an new graph with an edge set corresponding to the directed edges that start at this vertex.

b. For proving that DIRECTED-HAMILTONIANC = 1 IFF UNDIRECTED-HAMILTONIANC = 1 I would say that for UNDIRECTED-HAMILTONIANC as long as you only travel from one vertex to another using the undirected graph for that vertex then Hamiltonian cycles would be maintained meaning UNDIRECTED-HAMILTONIANC = 1 IFF there was a Hamiltonian cycle in the original directed graph meaning DIRECTED-HAMILTONIANC = 1.