CSCI-GA.1170-001/002 Fundamental Algorithms December 6, 2015

Solutions to Problem 1 of Homework 11 (13 (+2) points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 8*

(a) (2 pts) Assume directed graph G is acyclic. Show that G has at least one vertex v having no outgoing edges.

Solution:

Suppose that G is acyclic and every node has at least one outgoing edge. Pick any node u and begin following the outgoing edges from u. Since u has at least one outgoing edge (u, v) we can walk forward to v. Then since v has at least one outgoing edge (v, x) we can walk forward to x. Repeat until we visit a node, say w, twice. Let C denote the sequence of nodes encountered between successive visits to w then we can see that C is a cycle. Hence our assumption is wrong. Therefore G has at least one vertex v having no outgoing edges □

(b) (4 pts) Consider the following greedy algorithm for topological sort of a directed graph G: "Find a vertex v with no outgoing edges. If no such v exists, output 'cyclic'. Else put v as the last vertex in the topological sort, remove v from G (by also removing all incoming edges to v), and recurse on the remaining graph G' on $(n-1)$ vertices". Prove that this algorithm is correct.

Solution:

Base Case

If $n = 1$, output the single vertex as it has no outgoing edges. Hence the base case is true

Induction Hypothesis

If G is a graph consisting of $n-1$ vertices such that G has at least one vertex with no outgoing edges i.e G is acyclic then G has a topological ordering

Inductive Step

Given G, a DAG on n vertices, let v be the vertex with no outgoing edges. Remove v and all the incoming edges to v from G. Let the resulting graph be $G' = G - \{v\}$. Now G' is a DAG since deleting v from G cannot create any cycles. Therefore by Inductive Hypothesis G' has a topological ordering. Now place v at the end of the topological ordering of G'. This is valid since v has no outgoing edges □

(c) (3 pts) Using the adjacency list representation, show how to properly initialize the field $v.out$ which will contain the out-degree of each vertex $v \in V$ in time $O(n + m)$, and $n = |V|$ and $m = |E|$.

Solution:

$v.out$ will contain the out-degree of vertex v. Therefore it is equal to the size of the linked list v points to, in the adjacency list of G. To get the size of linked list, the entire list has to be traversed. Hence the time taken will be $O(deg(v))$ and we do this for each $v \in V$. Therefore the total time is $O(|E|)$ and also we visit each node in G to traverse it's corresponding list which will take $O(|V|)$ time. Therefore the total time taken to initialize $v.out$ for every $v \in V$ is $O(m+n)$ □

(d) (4+2 pts) Using part (c), initialize the FIFO queue Q which will contain all the vertices of V of out-degree 0 (i.e., $v.out = 0$; by part (a), at least one such node exists). Using this queue Q (i.e., the $Q.queue$ and $Q.dequeue$ operations), show a simple iterative algorithm to implement the greedy topological sorting algorithm described in part (b). Your algorithm should run in time no worse that $O(n^2)$. For **extra credit**, do it in time $O(m+n)$.

Solution:

```
1  Algorithm: TOPOLOGICAL-ORDERING(G)
2  CREATE-QUEUE(Q)
3  for each x ∈ V do
4  |   if x.out is 0 then
5  |   |   Q.queue(x)
6  |   end
7  end
8  num ← n
9  while Q is not empty do
10 |   v ← Q.dequeue()
11 |   T(v) ← num
12 |   num ← num − 1
13 |   for each (x,v) ∈ E do
14 |   |   x.out = x.out − 1
15 |   |   if x.out is 0 then
16 |   |   |   Q.queue(x)
17 |   |   end
18 |   end
19 end
20 Return T
```

Algorithm 1: Topological Ordering of G in $O(m+n)$ time

In the above algorithm, every vertex enters and leaves the queue exactly once. Hence this contributes to $O(n)$ time and after each vertex v is dequeued we reduce $x.out$ by 1 where $(x, v) \in E$. Therefore we visit every incoming edge into a vertex exactly once. Hence this contributes to $O(m)$ time. Therefore the total running time of the above psuedo code is $O(m+n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms December 6, 2015

Solutions to Problem 2 of Homework 11 (10 (+5) points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 8*

Assume G is an undirected graph with weight function w, and $e_1 \dots e_m$ are the m edges of G sorted according to their weight: $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$. Imagine you just ran the Kruskall's algorithm of G and it output an MST T of G. Now assume that somebody changes the weight of a single edge e_i from $w(e_i)$ to some other value w'. For each of the following 4 scenarios, describe the fastest algorithm you can think of to transform the original MST T of G to a new (and correct) MST T' of G after the edge weight change. Make sure you justify your answer, and express your running time as a function of m and n.

(a) (3 pts) Assume $e_i \in T$ and $w' < w(e_i)$ (so we decreased an MST edge).

Solution:

We decreased an MST Edge. Let the resulting tree be T'. Now T' is the new MST as it's weight is lesser than T. Therefore the running time is $O(1)$ □

(b) (4 pts) Assume $e_i \notin T$ and $w' < w(e_i)$ (so we decreased a non-MST edge).
(**Hint**: Compute the unique shortest path in T between the two end-points of e_i.)

Solution:

Add e_i to MST. So this will result in a cycle. According to the Cycle property in MST we have to remove the edge with the highest weight from that cycle.

To do this compute the shortest path in T between two end-points of e_i in $O(n)$ time (as $m = n - 1$). Let the weight of the heaviest edge in this path be w. If $w > w'$ then replace this edge with e_i in T. This will take $O(n)$ time. This will be the new MST. Therefore the running time is $O(n)$ □

(c) (3 pts) Assume $e_i \notin T$ and $w' > w(e_i)$ (so we increased a non-MST edge).

Solution:

We increased a non-MST edge. Therefore the MST will remain the same. Time taken is $O(1)$ □

(d) (5 pts) **Extra Credit:** Assume $e_i \in T$ and $w' > w(e_i)$ (so we increased an MST edge).
(**Hint**: Try to find the smallest weight edge e_j which should replace e_i under the new weight.)

Solution:

Remove e_i from T. Therefore we have two connected components now. Let them be T_1 and T_2. These connected components can be computed using BFS/DFS in $O(n)$ time. Now consider all those edges such that one end-point lies in T_1 and the other end point lies in T_2. Let the edge with least weight be e_i'. Now add e_i' to T. This will take $O(n)$ time. This resulting tree will be the MST. Therefore time taken is $O(n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms December 6, 2015

Solutions to Problem 3 of Homework 11 (13 (+5) points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 8*

(a) (3 points) Using the correctness of the Kruskal/Prim algorithm, show that if no two edges in the input graph have equal weights then the minimum spanning tree is unique.

Solution:

Assume that all the weights of the edges in G are distinct and G has two MSTs T_1 and T_2. Let e be the edge with the least weight in G that exists in T_1 but not in T_2. Now if we add e to T_2 it will result in a cycle. Let f be some edge in this cycle that is distinct from e i.e $w(f) > w(e)$. Now delete f from T_2 and let the resulting tree be T_2'. Therefore $w(T_2') < w(T_2)$ but we know that T_2 is an MST. This is a contradiction. Hence our assumption must be wrong. Therefore if no two edges in G have equal weights then the MST is unique □

(b) (5 points) Let e be the maximum weight edge on some cycle of a connected graph $G = (V, E)$. Prove that there exists an MST T of $G' = (V, E \backslash \{e\})$ which is also an MST of G. Namely, some MST of G does not include e.

Solution:

Assume that the MST T of G contains e in it. Let the end points of e be x and y. Remove e from T. Therefore T has two connected components T_1 and T_2.

T_1 and T_2 can be easily computed by using DFS/BFS on T. In G, start from the vertex x and move along the cycle until we reach y. Somewhere in between we will cross an edge $e' \neq e$ such that one end point of e' lies in T_1 and the other end point lies in T_2.

Since e is the heaviest edge in the cycle, $w(e') < w(e)$. Connect these two components T_1 and T_2 using e' after removing e from T. Let the resulting tree be T'. Then clearly $w(T') < w(T)$. But we know that T is an MST. Hence our assumption is wrong i.e e MST of G does not include e.

Therefore there exists an MST of $G' = (V, E \backslash \{e\})$ which is also an MST of G

□

(c) (5 points) Given a graph $G = (V, E)$ with edge weight function $w : E \mapsto \mathbb{N}$ such that no two edges have equal weight, give an algorithm to find a second smallest spanning tree. Assume that there exists a second smallest spanning tree with exactly one edge different from a minimum spanning tree. State the running time of your algorithm.

Solution:

Given below is the algorithm to find the second smallest spanning tree assuming that the second smallest spanning tree and MST differ by exactly one edge

Algorithm

$T \leftarrow \textsc{Mst}(G)$
$min\delta = -\infty$
For every edge $e \notin T$ do

- Add e to T. Now T has a cycle in T.
- Let e' be the maximum edge in the cycle next to e and $e' \neq e$
- Remove e' from T. Let the new tree formed be T'
- $\delta = w(e) - w(e')$. If $\delta < min\delta$ then $min\delta = \delta$ and $ResT = T'$
- Return $ResT$

In the first step we are computing MST of G. This will take $O(m \log n)$ time. The *for* loop runs $m - n$ times and finding e' will take $O(n)$ time. Therefore the total running time will be $O(mn + m \log n) = O(mn)$ □

(d) (5 points **Extra credit**) Prove the assumption you used in part (c).

Solution:

Assumption used in part (c) is that the second smallest spanning tree will differ from MST by exactly one edge.

Proof by Contradiction

Let T' be the second smallest spanning tree of G and T be the MST of G. Assume that T' differs from T by two or more edges i.e there are at least two edges in $T - T'$ and let (u, v) be one such edge with minimum weight.

Add (u, v) to T' which will result in a cycle C in T'. Let $(x, y) \in C$ and $(x, y) \in T' - T$. Now we claim that $w(x, y) > w(u, v)$. Assume that $w(x, y) < w(u, v)$. If we add (x, y) to T, it will result in a cycle C' in T. Let $(u', v') \in C'$ and $(u', v') \in T - T'$. If $w(x, y) < w(u', v')$ then we can remove (u', v') from T and we will get a new tree T'' such that $w(T'') < w(T)$ but we know that T is the MST. Hence $w(u', v') < w(x, y) < w(u, v)$. But we already know that (u, v) is the edge with minimum weight in $T - T'$. Hence our assumption is wrong. Therefore $w(x, y) > w(u, v)$.

Now that (x, y) and (u, v) are on the same cycle C, remove (x, y) from T' to get a new tree T'_1 which is different from T (as T'_1 and T' differ from each other by exactly one edge) and

$w(T_1') < w(T')$. This implies that T' is not the second smallest spanning tree which is a contradiction. Hence our assumption is wrong.

Therefore the second smallest spanning tree will differ from MST by exactly one edge $\square$

CSCI-GA.1170-001/002 Fundamental Algorithms December 6, 2015

Solutions to Problem 4 of Homework 11 (5 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 8*

In a directed graph $G = (V, E)$, each edge $(u, v) \in E$ has an associated independent value $r(u, v) \in \mathbb{R}$, $0 \leq r(u, v) \leq 1$ which represents how secure the channel from vertex u to vertex v is, i.e. the probability that the channel will not fail. Give an algorithm to find the most reliable path between two given vertices.

(**Hint**: Notice, here we need to *maximize the product*, while we know how to *minimize the sum*. To "turn" maximum into minimum, use $\max_{(a_1,\ldots a_k)\in S}(a_1 \cdot \ldots \cdot a_k) = \min_{(a_1,\ldots a_k)\in S}(\frac{1}{a_1} \cdot \ldots \cdot \frac{1}{a_k})$. To "turn" product into sum, use $\log(ab) = \log(a) + \log(b)$. Make sure you carefully explain how you follow these hints.)

Solution:

Let P be the the most reliable path between u and v and $(r_1, r_2, \ldots, r_k)$ be the reliabilities of edges in P. Therefore the product $(r_1 \cdot \ldots \cdot r_k)$ has to be the maximum. Consider the following

$$max_{(r_1,\ldots,r_k)\in E}(r_1 \ldots r_k) = min_{(r_1,\ldots,r_k)\in E}(\frac{1}{r_1} \cdot \ldots \cdot \frac{1}{r_k})$$

Take log on both sides

$$\begin{aligned} max_{(r_1,\ldots,r_k)\in E}\{\log(r_1 \ldots r_k)\} &= min_{(r_1,\ldots,r_k)\in E}\{\log(\frac{1}{r_1} \cdot \ldots \cdot \frac{1}{r_k})\} \\ &= min_{(r_1,\ldots,r_k)\in E}\{\log(\frac{1}{r_1}) + \ldots + \log(\frac{1}{r_k})\} \\ &= min_{(r_1,\ldots,r_k)\in E}\{(-\log r_1) + (-\log r_2) + \ldots + (-\log r_k)\} \end{aligned}$$

Therefore maximizing the product is equivalent to minimizing the sum of $-\log$.

Now transform G into G' such that $V' = V$, $E' = E$ and each edge $(u, v) \in E'$ has a weight a associated to it given by, $w(u, v) = -\log r(u, v)$. Note that $r(u, v) \leq 1 \implies \log r(u, v) \leq 0 \implies w(u, v) \geq 0$. This implies that every edge in G' has a non-negative weight associated to it. Hence djisktra can be called on any node of G'

Therefore call dijkstra on every node of G'. Let P be the shortest path between any two vertices $(u, v) \in V'$ and $(-\log r_1, \ldots, -\log r_k)$ be the weights of edges in P. Therefore the sum $(-\log r_1) + (-\log r_2) + \ldots + (-\log r_k)$ has to be minimum. This implies the product $(r_1 \cdot \ldots \cdot r_k)$ has to be maximum. Therefore P is the most reliable path between (u, v) in G. □

CSCI-GA.1170-001/002 Fundamental Algorithms December 6, 2015

Solutions to Problem 5 of Homework 11 (18 Points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 8*

Tucker, who lives in a node s of a weighted graph G (with non-negative weights), is invited to an exciting party located at a node h, where he will meet a girl of his dreams, Sharona. Naturally, Tucker wants to get from t to h as soon as possible, but he is told to buy some beer on the way over. He can get beer at any supermarket, and the supermarkets form a subset of the vertices $B \subset V$. Thus, starting at s, he must go to some node $b \in B$ of his choice, and then head from b to h using the shortest total route possible (assume he wastes no time in the supermarket). Help Tucker to meet Sharona as soon as possible, by solving the following sub-problems...

(a) (2 pts) Compute the shortest distance from s to all supermarkets $b \in B$.

Solution:

Call dijkstra on s which gives us the shortest distance from s to every other node in G. Return $\delta(s, b)$ for each $b \in B$ □

(b) (3 pts) Compute the shortest distance from every supermarket $b \in B$ to h. Can one simply add a new "fake" source s' connected to all supermarkets with zero-weight edges and run Dijkstra from s'?

Solution:

The newly added node s' is directly connected to each of $b \in B$. Therefore any path from s' to h has to path through at least one super market. Now running dijkstra on s' gives the shortest path and distance from s' to every other node in G.

Let P be the shortest path from s' to h. Therefore P must contain at least one super market. Now start moving from s' to h in P and let $b_1, \ldots, b_k \subset B, k \geq 1$ be the super markets that we came across while moving forward from s' to h in the that order. As we know that the sub-path of the shortest path is also a shortest path, we can get the shortest path or distance from each of $b_1, \ldots, b_k \subset B$ to h but this might not give us the shortest distance from every super market $b \in B$ to h. Therefore adding a fake source s' might not work.

However we can simply run dijkstra from h. So now we have shortest distance from h to every other node in G. Return $\delta(h, b)$ for every $b \in B$ □

(c) (4 pts) Combine parts (a) and (b) to solve the full problem.

Solution:

From part (a), we have the shortest distance from s to every other vertex in G

From part (b), we have the shortest distance from h to every other vertex in G.

Let $\delta_b(s,h)$ be the shortest distance from s to h such that the path contains at least one super market. Then $\delta_b(s,h)$ is given by

$$\delta_b(s,h) = min_{b \in B}\{\delta(s,b) + \delta(b,h)\}$$

□

(d) (4 pts) Your solution in part (c) used two calls to the Dijkstra's algorithm (one in part (a) and one in part (b). Define a new graph G' on $2n$ vertices and at most $2m + n$ edges (and "appropriate" weights on these edges), so that the original problem can be solved using a *single* Dijkstra call on G'.

Solution:

□

(e) (5 pts) Assume now that, in addition to beer, Tucker also needs to buy flowers for the beautiful Sharona, and the set of flower shops is $F \subset V$. As with supermarkets, Tucker can choose any flower shop $f \in F$, and it does not matter if he buys first beer, then flowers or vice versa, so he will naturally choose the best among all these options (which supermarket, which flower shop, and in what order). Show how to solve this problem for the poor Tucker.

Solution:

□