Assignment 2 solutions

Note Title 08-09-2019

1. Suppose H has a cycle $C_1, C_2, \dots, C_k$ where $C_1, \dots, C_k$ correspond to SCC.

By definition of H, there are edges $e_1, e_2, \dots, e_k$ such that $e_i = (u_i, v_i)$ where $u_i \in C$ and $v_i \in C_{i+1}$ (C_{i+1} is C_1 if $i = k$).

We claim there is a path from v_1 to u_1.

Indeed $v_1 \overset{\text{in } C_2}{\rightsquigarrow} u_2 \underset{e_2}{—} v_2 \overset{\text{in } C_3}{\rightsquigarrow} u_3 \underset{e_3}{—} v_3 \; \cdots \; u_k \underset{e_k}{—} v_k \overset{\text{in } C_1}{\rightsquigarrow} v_1$ is such a path

$\Rightarrow$ u_1 and v_1 should be in the same SCC, a contradiction.

In class, we saw a linear time algorithm to find all the SCC's, ie it outputs an array A such that

$$A[v] = C_i \quad \text{where } C_i \text{ is the SCC containing } v.$$

Now it scans all the edges in G and if there is an edge (u, v) it adds an edge between $A[u]$ and $A[v]$ in H.

2. Let $C_1, C_2, \dots, C_k$ be the SCCs and let H be the graph as above. $\because$ H is a DAG, we can arrange the vertices in topological sort. Let this ordering be $C_1, C_2, \dots, C_k$.
Now we check that H has edges $(C_1, C_2), (C_2, C_3), (C_3, C_4) \dots, (C_{k-1}, C_k)$.
Clearly all these steps can be done in linear time.

<u>Correctness</u>: Sps there are edges (C_i, C_{i+1}) for $i = 1 .. k-1$ in H.
Let u, v be two vertices in G where $u \in C_j$, $v \in C_k$, with $j \leq k$.
Then it follows that there is a path from u to v.

Conversely, suppose H does not contain an edge between C_i and C_{i+1} for some i.
Let $u \in C_i$ and $v \in C_{i+1}$.

Now, we claim that there is no path from u to v or v to u.
Indeed a path from u to v will have to contain an edge which goes in the "reverse" direction of C_j's ($\because$ $C_i \to C_{i+1}$ edge doesn't exist)
Same argument if there is a path from v to u.

3. First run Dijkstra and shortest path from s to every vertex v – call this array $D[v]$.
Let $pred[v]$ be the predecessor of v in the shortest path tree.

Arange the vertices in an order $s = v_1, v_2, v_3, v_4, \ldots, v_n$ such that $pred(v)$ comes before v for all the vertices v (eg, you can do pre-order traversal of the shortest path tree, or look at the order in which the vertices are visited by Dijkstra).
We will compute the second shortest path $D'[v]$ for all v in this order.
When we come to a vertex v_i, there are two choices:

(i) The predecessor of v_i in the second shortest path is same as $pred(v_i)$

In this case, $D'[v_i] = D'[pred(v_i)] + length(pred(v_i), v_i)$

Note that we have already computed $D'[pred(v_i)]$ when we come to v_i.

(ii) The predecessor of v_i is some other vertex than $pred(v_i)$: if this vertex is w then $D'[v_i] = D[w] + length(w, v_i)$

Note that this is the shortest path. (we may not have seen w yet)

$\therefore$, we will compute $D'[v_i]$ as

$$\min\left[D'[pred(v_i)] + length(pred(v_i), v_i),\ \min_{\substack{w:\\ (w,v_i) \text{ is an edge}\\ w \neq pred(v_i)}} \{ D[w] + length(w, v_i) \} \right]$$

4

We first compute the shortest path from s to every vertex v using Bellman Ford
– let $D[v]$ be this distance.

Let E' be the set of directed edges $e = (x, y)$ such that $d[y] = d[x] + \ell(x, y)$

<u>Claim</u>: If P is a shortest path from s to t, then all it's edges lie in E'.

Pf: Let P be such a path and $e = (x, y)$ an edge in it. Then the part of P from s to x and s to y must be shortest paths as well.

Conversely if P is any s-t path in E' then it is a shortest path.
Say $P = e_1, e_2, e_3, \ldots, e_k$, where $e_i := (x_i, x_{i+1})$. Then
$e_i \in E' \Rightarrow D[x_{i+1}] - D[x_i] = l_{e_i}$

Adding this for all edges, we get $D[t] - D[s] = l(P) \Rightarrow l(P) = D[t]$.

$\therefore$ we just need to count the # of paths from s to t in $G' = (V, E')$.

Now G' is a DAG (cycle $\Rightarrow$ length of cycle $= 0$, but we are assuming all cycles have length > 0).

So we have to count # paths from s to t in a DAG.
We can assume s has in-degree 0. We write the vertices in G' in topological sort $s, v_1, v_2, \ldots, v_n$. $\because$ s has indegree 0, we can always place it at the beginning.

Now maintain an array $C[v]$ which counts # s-v paths in G'.
Initialize $C[s] = 1$;

For $i = 1, \dots, n$

$$C[v_i] = \sum_{\text{edges } (v_j, v_i)} C[v_j];$$

(5) Let R denote these edges $e_1, e_2, \dots, e_k$ where $e_i = (u_i, v_i)$.

Let H be the graph obtained by removing R.

Let $D[v]$ denote shortest path from s to v in H.
Also let $D_i[v]$ be the shortest path from v_i to v in H. $\Big\}$ computing these takes $O(km \log n) = O(m \log n)$ time. by Dijkstra

Now suppose P is a shortest path from s to t in G and say it contains $e_{i_1}, e_{i_2}, \dots, e_{i_r}$ from R in this order as we go from s to t.

So P looks like

$$s \rightsquigarrow u_{i_1} \to v_{i_1} \rightsquigarrow u_{i_2} \to v_{i_2} \rightsquigarrow \cdots u_{i_r} \to v_{i_r} \rightsquigarrow t$$

So, $\ell(P) = D[u_{i_1}] + \ell e_{i_1} + D_{i_1}[u_{i_2}] + \ell e_{i_2} + D_{i_2}[u_{i_3}] + \cdots + D_{i_r}[t]$.

Thus knowing $e_{i_1}, \ldots, e_{i_r}$ we can find $\ell(P)$ in $O(1)$ time.

So, we need to cycle over all such choices of $e_{i_1}, \ldots, e_{i_r}$ (Note that ordering also matters).

But this quantity is $\leq 2^k \cdot k! = O(1)$.

6

Let T be the tree rooted at s.

Let $D_T[v]$ be the length of the path from s to v in this tree T.

<u>Claim</u>: T is a shortest path tree iff $D_T[u] \leq D_T[v] + \ell_{(u,v)}$ for every edge (u,v) in G. (*)

<u>Pf</u> If T is a shortest path tree, then $D_T[u] = D[u]$, where $D[u]$ denotes the shortest path from s to u. We know that $D[u]$ satisfies (*)

Conversely, Suppose D_T satisfies (*). $\because D_T[u] \geq D[u]$ ($D_T(u)$ is the length of a path from s to u)

we will show that $D_T[v] \leq D[v]$ for all v. This will imply that $D_T[v] = D[v]$.

Take a shortest s–v path in G (of length $D[v]$). Let this be

$s \to u_1 \to u_2 \ldots \to v = u_k$ For each edge $e = (u_i, u_{i+1})$ here

$D_T[u_i] \le D_T[u_{i+1}] + \ell_e$. ($\because D_T$ satisfies ⊛)

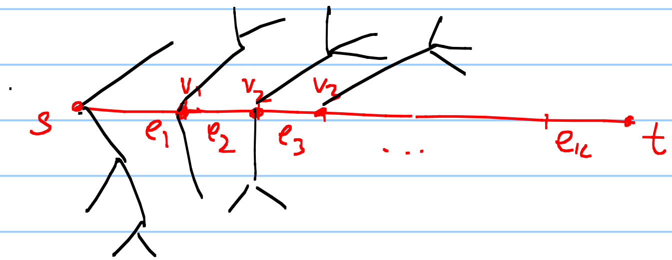
Adding this for all e in P $\Rightarrow$ $D_T[v] \le \ell(P) = D[v]$.

7 (EXTRA CREDIT ONLY)

Let P be a shortest s-t path. If we remove an edge not in P, shortest s-t path doesn't change. So enough to consider the case when $e \in P$.

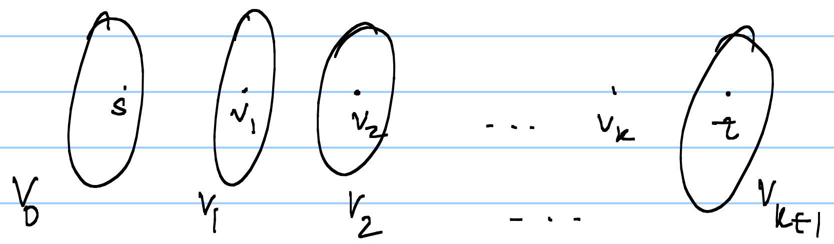
Let $D_s[v]$ be the shortest s-v path in G and $D_t[v]$ be the shortest v-t path in G. We can compute these in $O(m \log n)$ time by Dijkstra.

Let T be the shortest path tree from s.



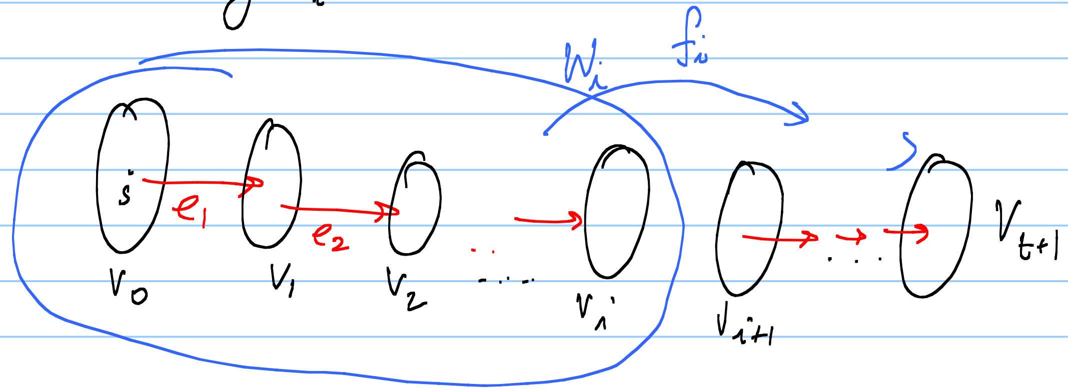
← this is how T looks like and P is the path from s to t in T.

When we remove $e_1, \ldots, e_k$ (ie all edges in P) from T the tree splits into components



Note that these are components of T and not G when P is removed.

If we remove only e_i, T looks like



Let W_i denote $V_0 \cup V_1 \cup \dots \cup V_i$

Now let P_i be the shortest path in G from s to t when we remove e_i. So, it must contain at least one edge from W_i to $\overline{W_i}$ (complement of W_i) - call this edge f_i. Say f_i goes from $x \in W_i$ to y in $\overline{W_i}$. We claim that

$\ell(P_i) = d_s[x] + \ell(f_i) + d_t[y]$ $\rightarrow$ this is where we use the fact that G is undirected.

$$\therefore \quad \ell(P_i) = \min_{\substack{e=(x,y) \\ x \in W_i,\, y \notin W_i}} \left(d_s[x] + \ell(f_i) + d_t[y] \right).$$

Now we can compute all these quantities from left to right by looking at every edge going out of V_0, then V_1, and so on.