COL351: Analysis and Design of Algorithms (CSE, IITD, Semester-I-2016-17) Homework-02

Name: Shivank Goel

Entry number: 2014CS10565

There are 4 questions for a total of 75 points.

1. (20 points) Given a weighted, undirected graph G and a minimum spanning tree T of G. Suppose that we decrease the weight of one of the edges not in T. Design an algorithm for finding the minimum spanning tree in the modified graph. Give pseudocode, discuss running time, and give proof of correctness.

Algorithm

→ Let the edge whose weight has been reduced is (u, v)

→ We trace the path from u to v in T and while tracing the path we will store the vertices encountered and the weight of all the edges encountered.

→ We will identify the pair (x, y) in path from u to v in T such edge (x, y) has maximum weight.

→ If the new weight of (u, v) in G is smaller than weight of edge (x, y) then we will remove edge (x, y) & add (u, v) instead into the ~~tree~~ T. Otherwise we will keep T same.

Running Time

→ DFS on a tree takes $O(V)$ time to trace the path from u to V in T. ($O(V+E)$ but $E = V-1$)

→ Storing max edge, comparing & exchanging take $O(1)$

→ Overall time complexity is $O(V)$

Proof Of Correctness

- We know that if we join 2 vertices in a tree we get exactly 1 cycle. $u - a_1 - a_2 - \cdots - a_n - v$
- Now according to cycle theorem: Let C be any cycle in G_1 & f is max cost edge in C. Then MST of G_1 doesn't contain f.

  Proof: Let T be MST of G_1 containing f. Remove f from T. This will partition vertices of T into 2 sets V_1, V_2. The set of vertices V_1 & V_2 form a cut in G_1. Since f is a part of cycle there must exist some other edge in G_1 say e' crossing the cut. But since f had max weight among all edges in cycle $w(f) > w(e')$ and thus f can't belong to MST(G).

- So following our algorithm we are removing the max weight edge from our cycle since it can't belong to MST of G_1 & adding a possible lower weight edge(u-v) instead of it. (If it is lower than max weight edge).
- Now all the cuts except the cuts made by removing edges in path from u to v in T remain same since only the weight of edge u to v is changed. Thus all other edges in MST would remain unchanged.

Pseudo code

```
givenewtree(T, u, v)
  → visited(v) = false   ∀ v ∈ T
  → int* maxw = INT_MIN
    pair* maxe = NULL
  → DFS(u, v, maxw, maxe)
  → if (maxw > weight(u,v)) {
        T = T + (u,v)
        T = T - maxe
    }
  → return T

DFS(u, v, maxw, maxe)
  → visited(u) = True
  → For each edge (u,x){
        if (visited(x) == false){
            if( weight(u,x) > maxw) {
                maxw = weight(u,x)
                maxe = (u,x)
            }
            if (x == v) return
            DFS(x, v, maxw, maxe)
        } }
```

2. (15 points) Let T be a minimum spanning tree of a weighted, undirected graph G. Given a connected subgraph H of G, show that $T \cap H$ is contained in some minimum spanning tree of H.

→ Proof: All the vertices in H are also present in T.
Since T is MST(G) all vertices of G are also in T.
Since H is subgraph(G) vertices of H are subset of vertices of G & hence also subset of vertices of T.

→ Since H is a connected subgraph any vertex in H is reachable from any other vertex in H.

→ Let all edges common to H and T are $\{e_1, e_2, \rightarrow e_m\}$

→ We have to prove there exist a MST of H containing all the edges in $H \cap T$ i.e $\{e_1, e_2 \longrightarrow e_m\}$

→ We can show that we can include all above edges in MST(H) by using exchange argument and cut property.

→ Lets say $e_i \in (H \cap T)$ doesn't belong to MST(H). We can make a new MST(H) such that H's weight is not more than previous one & it includes e_i.

Proof

→ Remove e_i from T. It will divide T into 2 set of vertices V_{T1} and V_{T2}

→ Consider intersection of all vertices in H and V_{T_1} as V_{HT_1} and Vertices of(H) $\cap$ V_{T_2} as V_{HT_2}

→ Since $e_i \in$ edge(H$\cap$T) then V_{HT_1} & V_{HT_2} are non empty.

→ Now we have got a cut in G1 as V_{T_1} & V_{T_2} Such tha e_i is min edge weight in that cut. (Or one of the min edge weights)

→ Now we have also got a cut in H as V_{HT_1} & V_{HT_2} Such that e_i is a cut edge of H. And Since H is subgraph of G1 e_i must be min of all edges in this cut of H and hence e_i can be included in MST(H) — ⊛

→ Similarly we can prove that all $\{e_1, e_2 \text{ —— } e_m\}$ can be included in MST(H). Hence Proved. (Simultaneously by above argument)

→ Note: To support above argument We can prove e_1 belongs to some MST by ⊛. Then by induction lets say $X = \{e_1, e_2 \text{ —— } e_i\}$ $(i<m)$ belongs to some

MST of H. Then remove e_{i+1} from T & get a cut in T as well as H and via similar argument as of ⊛ $X \cup e_{i+1}$ must be a some MST of H. Hence Proved.

3. There is a currency system that has coins of value $v_1, v_2, ..., v_k$ for some integer $k > 1$ such that $v_1 = 1$. You have to pay a person V units of money using this currency. Answer the following:

   (a) (16 points) Let $v_2 = c^1, v_3 = c^2, ..., v_k = c^{k-1}$ for some fixed integer constant $c > 1$. Design a greedy algorithm that minimises the total number of coins needed to pay V units of money for any given V. Give pseudocode, discuss running time, and give proof of correctness.

Pseudocode

→ Let v[i] i=1 to k has values of coins $v_1, v_2, \ldots v_k$

```
→ mincoins (v[], V){
      Int c = 0;
      for(int i=k; i>=1; i--){
          c = c + [V/V[k]]
          V = V - ([V/V[k]] * V[k])
      }
      return c
}
```

Running Time

→ There exists one loop with k iterations and hence time complexity is order k.

Let $O_1, O_2, \ldots, O_k$ be the number of coins of denominations $v_1, v_2, \ldots, v_k$ in Optimal Solution. Let $g_1, g_2, \ldots, g_k$ be the number of coins of these denominations in our greedy Solution.

Claim1: for all $1 \le i \le k-1$ $O_i < C$ {where $C = \frac{O_i}{O_{i-1}} \forall 1 \le i \le k$}.

Pf Let $O_i \ge C$ i.e $O_i = C + \alpha, \alpha \ge 0$ for some $i \in 1$ to $k-1$

Then if we pick C coins out of those $C+\alpha$ coins we get a money worth $C \times C^{i-1}$ which can be paid by just 1 coin of denomination C^i, and hence we will be able to replace $C (>1)$ coins with a single coin. So O would not have been an optimal sdn which is a contradiction. //

Now we have to prove that $g_i = O_i \ \forall \ 1 \le i \le k$.

Pf Let m be the first index coming downwards from k such that $g_m \neq O_m$ i.e $\forall\ m+1 \le i \le k \ g_i = O_i$

Case-I Let $O_m < g_m$. Thus money made by coins of of denominations $v_1, v_2 \ldots v_{m-1}$ will be.

$$\sum_{i=1}^{m-1} O_i v_i = \sum_{i=1}^{m-1} g_i v_i + (g_m - O_m) v_m > v_m \text{ since } (g_m - O_m) v_m \text{ extra}$$

money has to be made :

But max money that can be made via $\sum_{i=1}^{m-1} O_i v_i$ $1 \le m \le k$

is $\sum_{i=1}^{m-1} (C-1) C^{i-1} = (C-1) \frac{(C^{m-1}-1)}{(C-1)} = C^{m-1} - 1 = v_m - 1 < v_m$

{Since $(0 \le O_i \le C-1 \ \forall\ i = 1 \text{ to } k-1)$ By Claim 1}

Hence this is not possible.

Case II Let $Q_m > g_m$

This is trivially not possible. Because let $V_k, V_{k-1}, \ldots V_1$ be the money left which is still need to be paid at each step. So $g_i = \left\lfloor \frac{V_i}{C_i} \right\rfloor$. Now since $Q_i = g_i$ $\forall\ m+1 \leq i \leq k$ $\therefore V_m$ must be same in both the cases.

Now $g_m = \left\lfloor \frac{V_m}{C_m} \right\rfloor$ but if you take $Q_m = \left(\left\lfloor \frac{V_m}{C_m} \right\rfloor + \alpha \right)$ where $\alpha \geq 1$

then at m^{th} step money left would have been atleast $\underbrace{\left(\left\lfloor \frac{V_m}{C_m} \right\rfloor + \alpha \right) C_m}_{\downarrow} > V_m$ which is not possible.

$$\left\{ \text{Note: } \left\lfloor \frac{V_m}{C_m} \right\rfloor C_m + \alpha C_m \right\}$$

$$V_m - C_m < \downarrow \leq V_m \qquad \downarrow \geq C_m$$

(b) (4 points) Let $c > 1$ be any fixed integer constant. Does your greedy algorithm above also work when for all $1 \leq i < k$, $\frac{v_{i+1}}{v_i} \geq c$? Give reason for your answer.

No it doesn't work since we can find a counter example. Let $c = 2$.

Let $v_1 = 1$, $v_2 = 7$, $v_3 = 15$

Let $V = 21$

Greedy soln $(6, 0, 1) \equiv 7$ coins

Optimal soln $(0, 3, 0) \equiv 3$ coins

4. (20 points) Given a list of n natural numbers $d_1, d_2, ..., d_n$, design an algorithm that determines whether there exists an undirected graph $G = (V, E)$ whose vertex degrees are precisely $d_1, ..., d_n$. That is, if $V = \{v_1, ..., v_n\}$, then degree of v_i should be exactly d_i. G should not contain multiple edges between the same pair of nodes or "loop" edges. Give pseudocode, discuss running time, and give proof of correctness.

Pseudocode

→ Sort the vertices in decreasing order of their degrees & store these degrees in an array say D[]

```
→ for (i = 1 to n) {
        int c = D[i]
        if (c+i > n) return (false)
        else {
            D[i] = 0
            for (j = i+1 to i+c) { D[j] = D[j] - 1
                                   if (D[j] < 0) return false }
        }
    }
→ return (true)
```

Running Time

→ We are running loop $n = V$ times and in each iteration of loop we are doing computation which is equal to degree of that vertex. Total computations $= \sum_{i=1}^{n} (\text{degree}_i)$

$= 2E$. So time complexity is $O(E)$

→ Initial sorting of vertices take $O(V \log V)$

→ Overall time complexity $O(V \log V + E)$.

Proof of Correctness. (For simplicity lets assume degrees are in descending order. Else we will arrange them in that way.)

(#) We will prove that a non-inc seq of degrees $(d_1, d_2, \ldots, d_n)$ is possible for n vertices iff there exist a graph G' of $(n-1)$ vertices with degrees $(d_2-1, d_3-1, \ldots, d_m-1, \ldots, d_n)$ where $m = d_1 + 1$.

Pf

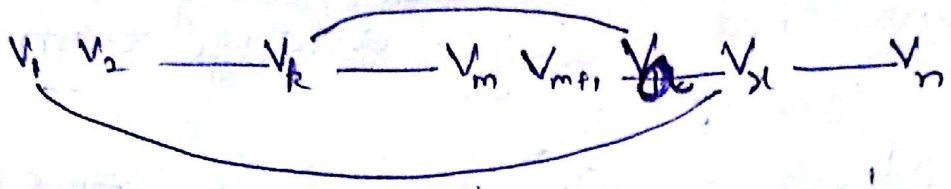
$\Leftarrow$ Let a G' exist of $(n-1)$ vertices with degrees $(d_2-1, d_3-1, \ldots, d_m-1, \ldots, d_n)$ $\cdot m = d_1 + 1$. Then we can create a new vertex with degree d_1 & join it with vertices d_i $2 \le i \le m$ to get G_1 of n vertices and degree $(d_1, d_2 \ldots d_n)$.

$\Rightarrow$ Let $(d_1, d_2, \ldots d_n)$ is possible with n vertices. If v_1 has edges with $v_2, v_3 \ldots v_m$ $(m = d_1+1)$ then we are done by simply removing v_1. Else let say V_1 doesn't have edge with vertex V_k $2 \le k \le m$. Now since degree(V_k) > degree(V_i) $k+1 \le i \le n$. Let V_1 has edge with V_x $(x > m)$

Case-I $\deg(V_k) = \deg(V_x)$ Since V_1 is connected with V_x but V_k is not connected with V_1 there is atleast 1 vertex V_a with which V_k is connected but V_x not connected.

Case-II $\deg(V_k) > \deg(V_x)$ Since V_k is connected to more vertice. $\exists$ V_a such that V_k & V_a are connected but V_x and V_a are not connected.

PTO



→ Now we can connect V_k with V_1 & disconnect V_k with V_a.

→ And then connect V_x with V_a & disconnect V_x with V_1.

→ Continuing this we can construct G_1 such that degree of all vertices remain same & V_1 is connected to $V_2, V_3 \longrightarrow V_m$ $(m = d_1 + 1)$ & then we remove V_1 to prove our statement.

Hence Proved.