CS 5114
Solutions to Homework Assignment 8

Meghendra Singh

April 14, 2017

[30] 1. Exercise 35.1-4. Give an efficient greedy algorithm that finds an optimal vertex cover for a tree in linear time. Give pseudocode for your algorithm. Analyze its time complexity.

A tree is an acyclic and connected graph. The recursive algorithm GREEDY-VERTEX-COVER presented below computes an optimal vertex cover for a tree $T(V, E)$ in linear time. Here V is the set of all vertices in T and E is the set of all the edges present in T. We start by assuming that $u \in V$ is a randomly selected node and E' is another set that holds a copy of E, i.e. $E' = E$. We make the initial call to the algorithm as: GREEDY-VERTEX-COVER$(E', u, NULL)$. The pseudocode for GREEDY-VERTEX-COVER algorithm is as follows:

GREEDY-VERTEX-COVER(E', u, p)

```
1  FLAG = False
2  S = {}
3  // Look at all neighbors except for the parent p.
4  // So, if u is a leaf this for loop will not execute
5  for each edge (u, v) ∈ E' − {(p, u)}
6       S = S ∪ GREEDY-VERTEX-COVER(E', v, u)
7       // (u, v) will be present in E' only if v is a leaf node or a node ∉ S
8       if (u, v) ∈ E'
9            FLAG = True
10           E' = E' − {(u, v)}
11 // FLAG will be True only if u is a parent of a leaf node or a node ∉ S
12 if FLAG == True
13      // include u in the vertex cover S and
14      // remove the edge connecting it to its parent p
15      S = S ∪ {u}
16      E' = E' − {(p, u)}
17 return S
```

GREEDY-VERTEX-COVER starts by looking at each edge connected to u except for the edge between u and p (this condition will be checked only if u is not the starting node, if u is the starting node p will be $NULL$). Each neighbor connected to u (except for p i.e. the parent or node which called GREEDY-VERTEX-COVER on u) recursively calls GREEDY-VERTEX-COVER on its children until a leaf node of the tree T is reached (in this case the for loop at line 5 wouldn't execute as there will be no other edge for a leaf v other than that with the parent i.e. (p, u)) and an empty set $S = \{\}$ would be returned by the call on the leaf node (this is because the boolean variable $FLAG$ will remain $Flase$ for a leaf as the for loop would not execute).

When the call returns to the immediate parent u of a leaf node v the edge (u, v) would be removed from the set E' and $FLAG$ will be set to $TRUE$, this will lead to the parent u being added to the vertex cover S. This is correct because in a tree any internal node (parent) that connects to a leaf has to be in the vertex cover (otherwise the vertex cover would not "cover" the edge which connects the internal node to the leaf node). Also, once a node is added to the vertex cover the edge connecting it to its parent i.e. (p, u) is removed from E', this prevents the "if" block in lines 8 through 10 from being executed when the call returns to the parent node. This prevents the parent from being added into the vertex cover unless another child of the same parent was a leaf or was a node that was not included in the vertex cover, in which case $FLAG$ would be set to $True$ when the call returns to the parent node. This ensures that any internal node which is only connected to nodes included in the vertex cover, would never be included in the vertex cover itself. However, if there is at-least one child leaf node or an internal node that is not already a part of the vertex cover connected to the parent internal node, the parent internal node will be included in the vertex cover. This will happen even if one or more of its neighbors are already in the vertex cover (this is essential for the vertex cover to be valid, otherwise the edge connecting the parent internal node to the child leaf or an child internal node which was not in the vertex cover will not be covered by the solution).

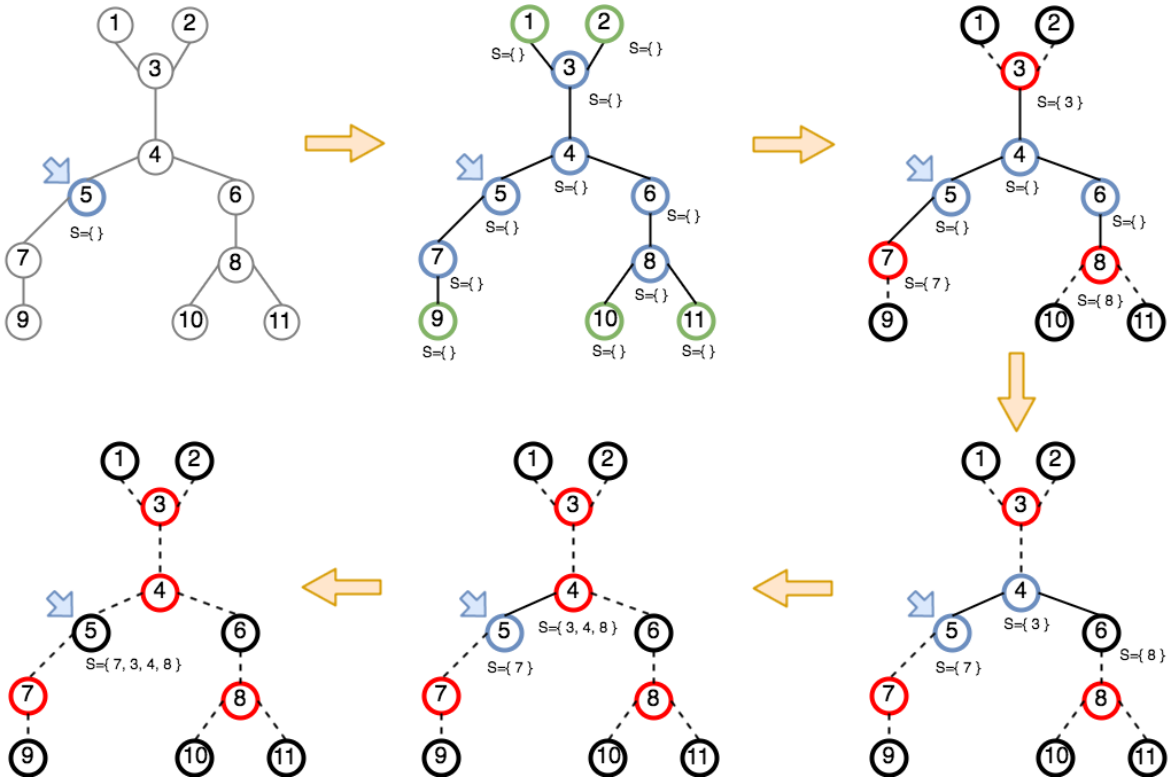

Figure 1: GREEDY-VERTEX-COVER operating on an example tree T

Consider the tree $T(V, E)$ given in the top left of figure 1 below. Here, $E = \{(1, 3), (2, 3), (3, 4),$

$(4,5),(4,6),(5,7),(7,9),(6,8),(8,10),(10,11)\}$ and $V = \{1,2,3,4,5,6,7,8,9,10,11\}$. If we call GREEDY-VERTEX-COVER$(E', 5, NULL)$, with $E' = E$ we will get $S = \{7,3,4,8\}$ as the result which is an optimal vertex cover for the tree T. Figure 1 shows how the algorithm will execute on T, starting at node 5. Blue nodes represent the internal nodes for which GREEDY-VERTEX-COVER has not finished execution (i.e. calls to all of its children have not finished execution) and green nodes represent the leaf nodes. Once GREEDY-VERTEX-COVER finishes execution on a node it is either included in the vertex cover or not included in the vertex cover and the call is returned to its parent. Red nodes represent nodes which were included in the vertex cover, the black nodes were not included in the vertex cover and a dashed edge represents an edge that has been removed from E'.

The algorithm is very similar to depth-first search, as it recursively executes on all the children of a starting node. The recursive calls would be made until a leaf node is reached and will then return to the starting node, where the set S would contain the accumulated vertex cover for the tree T. Also, any node in V can be chosen as the starting node u (even a leaf node) and the acyclic property of trees will ensure that the algorithm would finish execution and produce optimal vertex covers. In the case that there is just one node in the tree GREEDY-VERTEX-COVER would return an empty set. The algorithm executes once for each node in V, hence the running time of GREEDY-VERTEX-COVER is linear in terms of the number of nodes in T i.e. $O(|V|)$.

---

[30] 2. Exercise 35.2-2. Show how in polynomial time we can transform one instance of the traveling salesman problem into another instance whose cost function satisfies the triangle inequality. The two instances must have the same set of optimal tours. Explain why such a polynomial-time transformation does not contradict Theorem 35.3, assuming that P $\neq$ NP.

Here, the optimization problem is TSP, which is defined as follows:

> TSP
> INSTANCE: $\{\langle G, c\rangle\} : G = (V, E)$ is a complete undirected and c is a cost function from V x $V \Longrightarrow \mathbb{Z}^+ \cup \{0\}$
> SOLUTION: An ordering of the vertices $v_1, v_2, ..., v_n \in V$ such that: $c(v_1, v_n) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$ is minimized.

We need to show the following:

1. We can transform in polynomial time, one instance of TSP into another instance of TSP with a cost function that satisfies the triangle inequality.
2. The two instances of TSP have the same set of optimal tours.
3. Assuming $P \neq NP$ the polynomial-time transformation does not contradict Theorem 35.3.

1. **Polynomial time transformation:** Let I be an instance of TSP with the complete graph $G = (V, E)$ and a cost function c. Let m be the cost of the costliest edge in E, i.e. $m = \max[c(u, v)]\ \forall (u, v) \in E$. Let I' be another instance of TSP with

the complete graph $G' = G = (V, E)$ and a cost function c' such that $c'(u, v) = c(u, v) + m \ \forall (u, v) \in E$. Let e_1, e_2 and e_3 be any 3 arbitrarily selected edges in E with costs $c(e_1) = c_1, c(e_2) = c_2$ and $c(e_3) = c_3$ respectively in the instance I. By our definition of c' the costs for these three edges in the instance I' will be $c'(e_1) = c_1 + m, c'(e_2) = c_2 + m$ and $c'(e_3) = c_3 + m$. Now, we can define the following inequalities:

$$c_1 \leq m \tag{1}$$

$$c_1 + m \leq c_1 + m + c_2 + c_3 \tag{2}$$

Using (1), we can replace c_1 in the RHS of (2) with m, while preserving the inequality. So we have,

$$\begin{aligned} c_1 + m &\leq m + m + c_2 + c_3 \\ \implies (c_1 + m) &\leq (c_2 + m) + (c_3 + m) \\ \therefore \ c'(e_1) &\leq c'(e_2) + c'(e_3) \end{aligned}$$

As, G' and G are both complete graphs and e_1, e_2 and e_3 were arbitrarily selected edges, from the above result we can say that the cost function c' satisfies the triangle inequality. Also, computing c' using c can be done in linear time with respect to the number of edges $|E|$ in G (i.e. one iteration over all the edge costs in c to get m and a second iteration over all the edge costs in c to get c'). As c' is the cost function for the instance I', we have transformed a general instance of TSP i.e. I into another instance of TSP i.e. I' whose cost function c' satisfies the triangle inequality in linear time.

2. **Both the instances (i.e. I and I') have the same set of optimal tours:**
Let T be an optimal tour for the original TSP instance I. Assume that there exists a different optimal tour T' for I', i.e.

$$c'(T) \geq c'(T') \tag{3}$$

Let the cost of T in I be c_T, i.e. $c(T) = c_T$ and that of T' in I be $c_{T'}$, i.e. $c(T') = c_{T'}$. Now, we know that:

$$c_T \leq c_{T'} \text{ , as } T \text{ is an optimal tour in } I.$$

Also, by the definition of c' we know that:

$$c'(T) = c_T + m * |V| \tag{4}$$

$$c'(T') = c_{T'} + m * |V| \tag{5}$$

As, c' adds m to every edge-cost in c, a tour would incur a total additional cost of $m * |V|$. This is the additional cost added to c_T and $c_{T'}$ in (4) and (5) above. By our assumption, i.e. (3) we have the following inequalities:

$$\begin{aligned} c'(T) &\geq c'(T') \\ c_T + m * |V| &\geq c_{T'} + m * |V| \\ \implies c_T &\geq c_{T'} \end{aligned}$$

If $c_T > c_{T'}$ then this would mean that T is not an optimal tour in I, which is a contradiction to our initial assumption. However, if $c_T = c_{T'}$, this would mean that both T and T' were optimal tours in I as well as I'. Hence, both the instances of TSP will have the same set of optimal tours.

3. **Assuming $P \neq NP$ the transformation does not contradict Theorem 35.3:** Theorem 35.3 states that, if $P \neq NP$, then for any constant $\rho \geq 1$, there is no polynomial time algorithm with an approximation ratio ρ for TSP.

   By Theorem 35.2 we know that a polynomial time approximation algorithm (Approx-TSP-Tour) exists for TSP instances with a cost function satisfying the triangle inequality. We also know that the approximation ratio for this algorithm is 2, i.e. $\rho = 2$. Let H' be a solution computed by Approx-TSP-Tour for the transformed TSP instance I' and H^* be the optimal solution for I' (as seen in the last section H^* will be an optimal solution for I as well because both I and I' have the same set of optimal tours). By the definition of approximation ratio we have:

$$\frac{c'(H')}{c'(H^*)} \leq \rho$$
$$\frac{c'(H')}{c'(H^*)} \leq 2$$

We also know that,

$$c'(H') = c(H') + m * |V|$$
and,
$$c'(H^*) = c(H^*) + m * |V|$$

So we can write,

$$\frac{(c(H') + m * |V|)}{(c(H^*) + m * |V|)} \leq 2$$
$$\therefore \quad c(H') + m * |V| \leq 2(c(H^*) + m * |V|)$$
$$c(H') \leq 2(c(H^*)) + m * |V|$$

Dividing both sides by $c(H^*)$,

$$\frac{c(H')}{c(H^*)} \leq 2 + \frac{m * |V|}{c(H^*)}$$

The RHS in the above inequality corresponds to the approximation ratio for the original TSP instance I. We can see that, this ratio is no longer a constant. It would vary from instance to instance and will depend on the number of vertices times the maximum edge cost, i.e. $m * |V|$. Furthermore, if we assume that all edges in the optimal solution H^* have a cost of 1, then $c(H^*) = |V|$ and we can rewrite the above inequality as:

$$\frac{c(H')}{c(H^*)} \leq 2 + \frac{m * |V|}{|V|}$$
$$\frac{c(H')}{c(H^*)} \leq 2 + m$$

Once again we can see that, this ratio is not a constant and would depend on the maximum edge cost for an instance. Hence, the transformation does not lead to a polynomial time algorithm with a constant approximation ratio ($\rho \geq 1$), for the general TSP instance I. Hence, our polynomial time transformation from the general TSP instance I to the specific TSP instance I' (with cost function satisfying the triangle inequality), does not contradict Theorem 35.3.
