CSE 3500

Exam 1
TODAY
@ 5PM
IN
ARJ 105

GREEDY ALGORITHMS.

SETTING. INPUT IS A Set of objects. The goal is to identify a Subset Subject to a set of Constraints and which OPTIMIZES

A given OBJECTIVE FUNCTION.

ANY Solution that satisfies the given constraints is Said to be FEASIBLE. A FEASIBLE Solution that optimizes the objective function is OPTIMAL.

Algorithm Greedy:

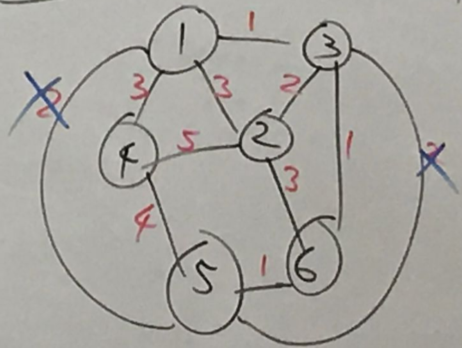
~~INPUT: $O_1, O_2, \ldots, O_n$~~

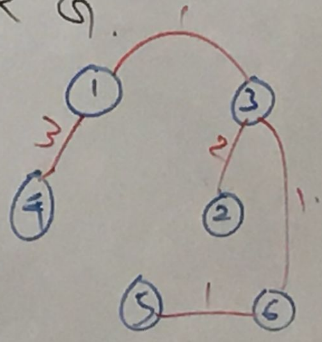
```
Solution = ϕ;
for i=1 to n do
    IF Solution ∪ {O_i}
    IS Feasible then
    Solution = Solution ∪ {O_i};
Output solution;
```

MINIMUM SPANNING TREE (MST):

INPUT: A weighted undirected graph $G(V,E)$

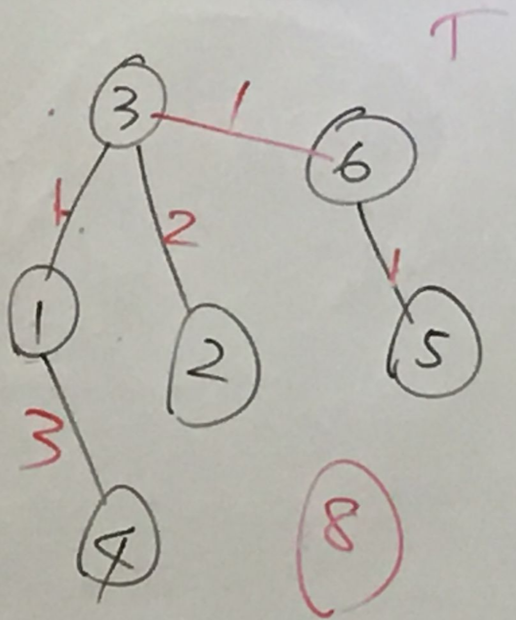
Output: A MST FOR G.

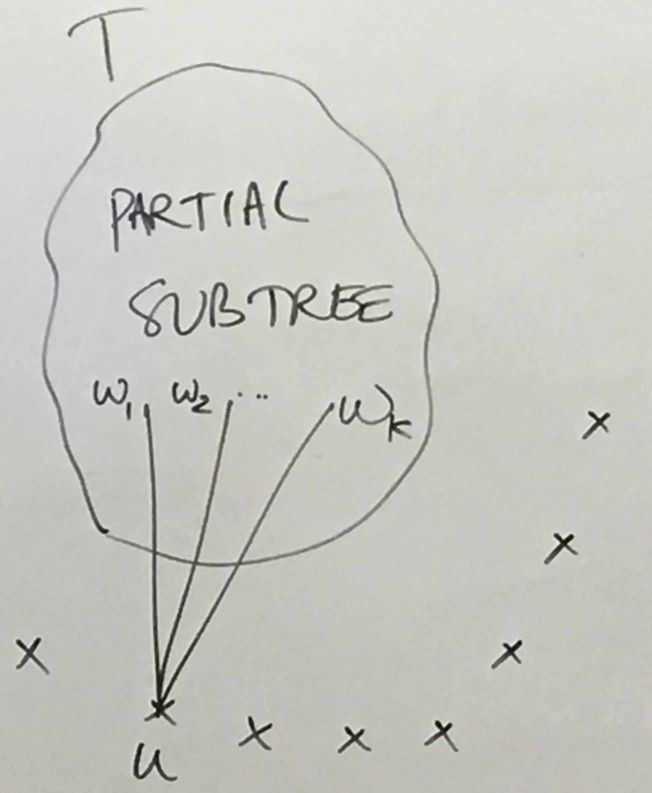




TOTAL=

PRIM'S ALGORITHM





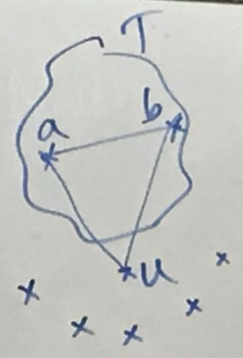
NEAR[u] = the NEAREST TREE NEIGHBOR OF u.

w_i = NEAR[u] if Cost(u, w_i) is the SMALLEST AMONG Cost(u, w_1), Cost(u, w_2), ..., Cost(u, w_k)

We use a 2-3 TREE Q to store all the nodes outside the partial tree T.

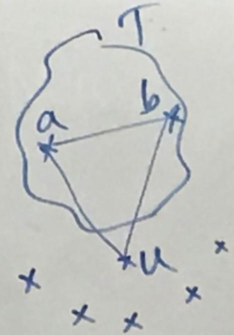
① Let (a, b) the edge with the LEAST WEIGHT in G.

② for $u \in V - \{a, b\}$ do
if Cost(u, a) < Cost(u, b) then NEAR[u] = a else NEAR[u] = b;



③ for $u \in V - \{a, b\}$ do
INSERT u into Q with a key value of Cost(u, NEAR[u]);

We use a 2-3 TREE Q to store all the nodes outside the partial tree T.



① Let (a, b) the edge with the LEAST WEIGHT in G. → $O(|E|)$

② **for** $u \in V - \{a, b\}$ **do** → $O(|V|)$

if Cost(u, a) < Cost(u, b) then NEAR[u] = a else NEAR[u] = b;

③ **for** $u \in V - \{a, b\}$ **do**

INSERT u into Q with a key value of Cost(u, NEAR[u]);

$O(|V| \log |V|)$

$n = |V|$.

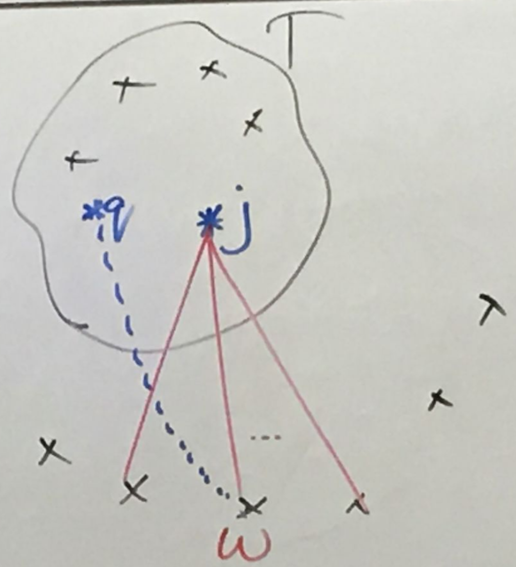
(4) <u>for</u> $i = 1$ to $(n-2)$ <u>do</u>

let j be the node in Q with the least key;

<u>for</u> $w \in Adj(j)$ <u>do</u>

if $Cost(w, NEAR[w]) >$ $Cost(w, j)$ then $NEAR[w] = j$, $O(\log|V|)$

Add $(j, NEAR[j])$ to T and delete j FROM Q;



if $Cost(w, NEAR[w]) >$ $Cost(w, j)$ then $NEAR[w] = j$

$$\sum_{w \in V} d_w = 2|E|.$$

TOTAL FOR Step 4

$$= \sum_{w \in V} d_w \log|V|$$

$$= \log|V| \sum_{w \in V} d_w$$

$$= O(|E| \log|V|)$$

d_w – DEGREE OF w.

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 15: October 18, 2016

Greedy Algorithms

- There are numerous problems for which simple greedy approaches could help in obtaining optimal solutions.
- Consider the following class of problems: Let I be a set of objects. We are interested in finding a subset S of I such that S satisfies a set of constraints and optimizes a given objective function.
- Any subset of I that satisfies the given constraints is called a *feasible solution*. Any feasible solution that optimizes the objective function is called an *optimal solution*.
- We would ideally like to get an optimal solution.
- The minimum spanning tree problem belongs to the above class of problems.

Minimum Spanning Tree (MST) Problem

- The MST problem can defined as follows. Input: A weighted directed graph $G(V, E)$; Output: A spanning tree of G that has the least total edge weight.
- A spanning tree of G is nothing but a subset of the edges of G that will induce a tree on V.
- Note that a spanning tree will have $|V| - 1$ edges.
- Thus the MST problem can be thought of as that of choosing a subset T of E such that T will induce a tree on V and the sum of weights of all the edges in T is as small as possible.

A General Greedy Algorithm

- Let I be the input set of objects. A generic greedy algorithm starts with an empty set S as the solution. It examines each object O of I at a time and makes a decision on whether to include O into S or not. When it finishes examining all the objects of I, it outputs the resultant solution.
- The order in which the objects are examined could make a difference in the quality of the output. A *selection criterion* is used to determine this order.

- A pseudocode for a generic greedy algorithm is given below.

> $Solution = \emptyset$;
> **for** $i = 1$ **to** n **do**
>   Select the next object O from I using
>   a relevant selection criterion; $I = I - \{O\}$;
>   **if** $Solution \cup \{O\}$ is feasible **then** $Solution = Solution \cup \{O\}$;
> Output $Solution$;

- Note that in a greedy algorithm once we make a decision with respect to an object we will never re-examine this decision.

Prim's Algorithm

- In the last lecture we discussed the Kruskal's algorithm. In this lecture we will explore the Prim's algorithm in detail.
- Prim's algorithm always keeps a tree T that is a subtree of a MST of G. To begin with T will have only one edge, namely, the edge of G with the least weight. Ties are broken arbitrarily.
- The algorithm starts with a tree T with one edge and this tree is grown one edge at a time. When the tree has $|V| - 1$ edges, the resultant tree is output.
- An important question here is which edge of G should be added next to T?
- Since we are interested in minimizing the sum of the weights of all the edges in T, a relevant greedy approach for selecting the next edge for T will be to choose the edge whose weight is the least from among all those edges that have one end point in T and the other end point outside T.
- To identify the edge that should enter T next, we define a data structure called *near*. For any node u outside T, we define $near[u]$ to be that node v of T such that $cost(v, u)$ is the least (across all the nodes u in T). Here $cost(i, j)$ refers to the weight of the edge (i, j) for any $(i, j) \in E$.
- To begin with we identify the edge of G with the least weight and T has only this edge at the beginning. Let (a, b) be this edge. For every node u other than a and b we compute $near[u]$ and insert u into a 2-3 tree Q with a key value of $cost(u, near[u])$.
- From thereon, we identify the node u from Q with the least key value. This nodes enters the tree T next. When we insert a new node into T, the *near* values of some of the nodes might change. Luckily the only nodes whose *near* values might change will be those that are

adjacent to u (the node that has just now been inserted into T) in G. We modify the *near* values of the nodes as needed. Followed by this we repeat the process of identifying the next node that should enter T next, and so on. Let $n = |V|$ and $m = |E|$. A pseudocode follows.

1) Identify the edge of G with the least weight. Let (a, b) be this edge;
2) $T = \{(a, b)\}$;
3) **for** every $u \in V - \{a, b\}$ **do**
    4) **if** $cost(u, a) < cost(u, b)$ **then** $near[u] = a$ **else** $near[u] = b$;
    5) Insert u into a 2-3 tree Q with $cost(u, near[u])$ as its key;
6) **for** $i = 1$ **to** $n - 2$ **do**
    7) Identify the node u in Q with the least key;
    8) $T = T \cup \{(u, near[u])\}$; Delete u from Q;
    9) **for** every $w \in Adj(u)$ **do**
        10) **if** $cost(w, u) < cost(w, near[w])$ **then** $near[w] = u$;

- **Run Time Analysis:** Step 1 takes $O(m)$ time. Step 2 takes $O(1)$ time if we keep T as a list of edges. Step 4 takes $O(1)$ time. In step 5, we perform an insert operation into a 2-3 tree. Each insert takes $O(\log n)$ time. Thus step 3 takes a total of $O(n \log n)$ time.

  In step 10, the key value of a node w in Q changes. One way of making this change will be to delete w from Q and insert it back into Q with the new key value. Thus step 10 takes $O(\log n)$ time. If the degree of the node u is d_u, then step 9 takes a total of $O(d_u \log n)$ time. Step 7 takes a total of $O(n \log n)$ time (over all values of i in the **for** loop of line 6). Step 8 takes a total of $O(n \log n)$ time (over all values of i in the **for** loop of line 6).

  The total time step 9 takes (over all values of i in the **for** loop of line 6) is $O\left(n \log n + \sum_{u \in V - \{a,b\}} d_u \log n\right) = O\left(n \log n + \sum_{u \in V} d_u \log n\right) = O(n \log n + m \log n)$, since $\sum_{u \in V} d_u = 2m$ for any undirected graph.

  In summary, the total run time of the algorithm is $O((m + n) \log n)$.