97

Name(last, first): Name Withheld

UPE-Certified Name Withholder™

Discussion

ID Withheld

UPE-Certified ID Withholder™

U C L A Computer Science Department

CS 180 **Algorithms & Complexity**

Midterm **Total Time: 1.5 hours** **October 27, 2016**

Each problem has 20 points.

All algorithm should be described in English, bullet-by-bullet

1 **a.** Describe Depth First Search on an undirected and un-connected graph.
b. Analyze the time complexity of DFS when there are no cycles.

(a) DFS (node)

(1) Mark node as visited.

(2) If node has unvisited neighbours, pick an arbitrary neighbour of node, "neighbour", and perform DFS(neighbour).

+15

(3) If not, and there are still unvisited nodes, pick an arbitrary unvisited node, "unvisited", and perform DFS(unvisited).

(This ensures that the algorithm will backtrack and will process every component.)

(b) Let us charge the operations of visiting vertices in step 2 to the edges. Each edge is charged once, so DFS globally takes O(e).

+5

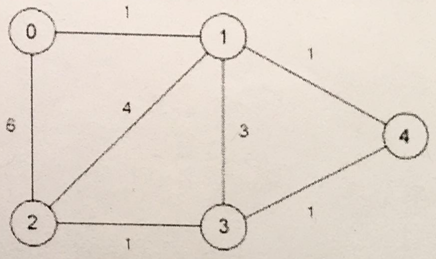
The number of components in the graph can be as high as n - 1, so it is O(n). The algorithm performs this many "jumps" in step 3.

Therefore, DFS has a time complexity of O(e + n).

20

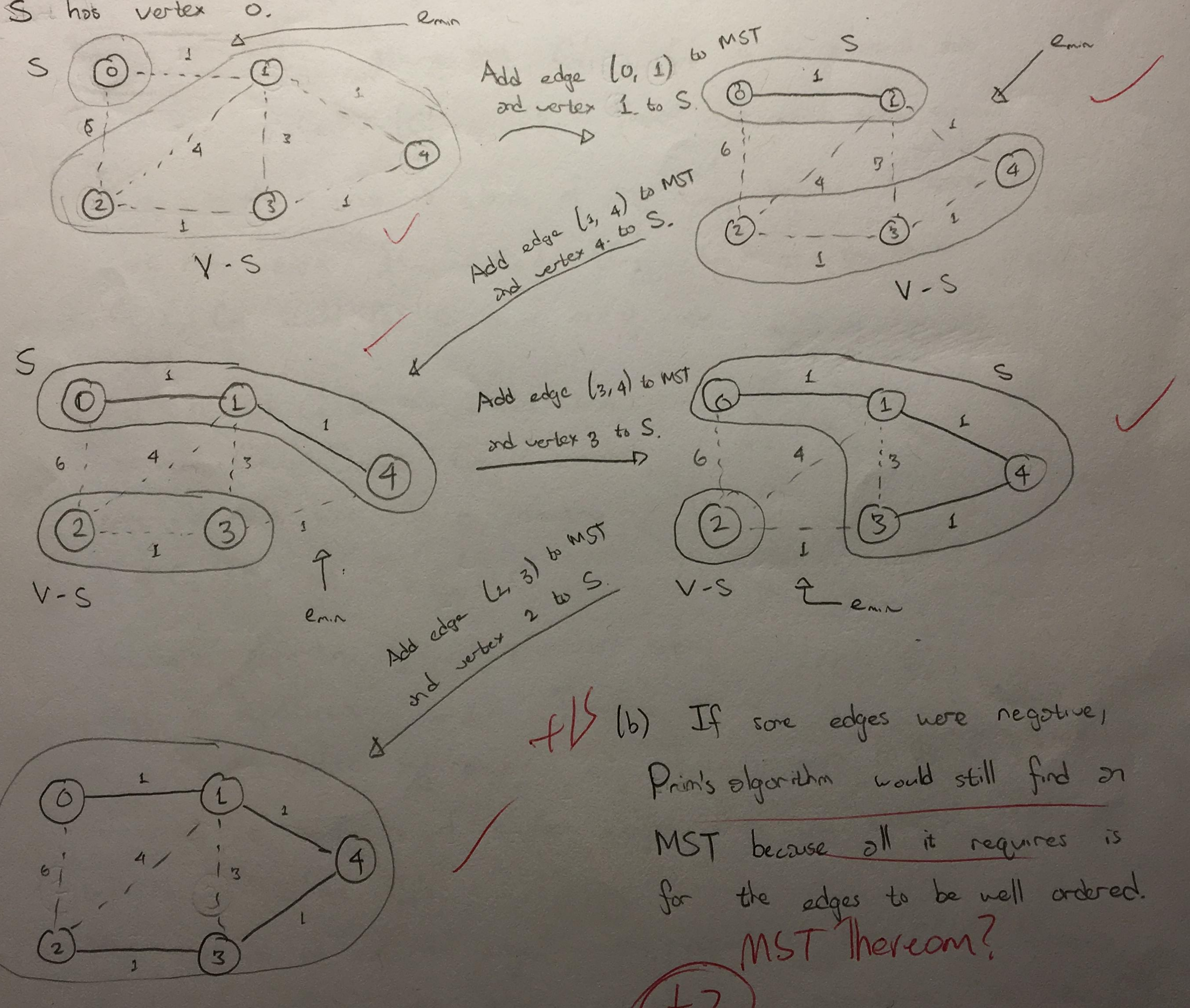
Name(last, first): Name Withheld UPE-Certified Name Withholder™

2 . a. Use Prim's MST algorithm to find an MST in this graph. Show each step on the graph shown below.
b. If some edges were negative would the algorithm still find an MST. Why?



Prim's MST algorithm starts with set S consisting of one vertex and V - S consisting of all others.

S has vertex 0.



S
V - S
e_min
Add edge (0, 1) to MST and vertex 1 to S.
Add edge (1, 4) to MST and vertex 4 to S.
Add edge (3, 4) to MST and vertex 3 to S.
Add edge (2, 3) to MST and vertex 2 to S.

+15 (b) If some edges were negative, Prim's algorithm would still find an MST because all it requires is for the edges to be well ordered.

MST Thereom?

+2

(b

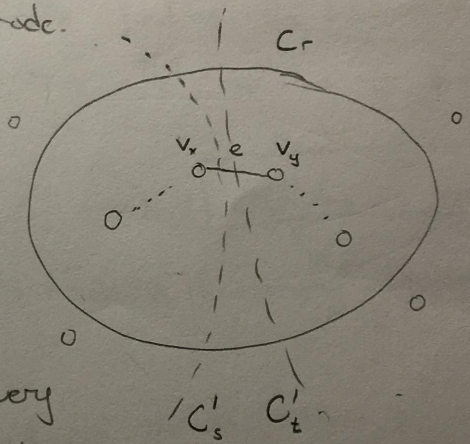
17

Name(last, first): Name Withheld UPE-Certified Name Withholder™

20

3. Prove that solving the k-clustering problem (described in class and in the book) using Kruskal's MST algorithm, produces an optimal clustering. That is, it will produce an optimal set of clusters C1, C2, …, Ck with Maximum cluster distances. (Use a figure to better describe your proof: as was done in class / book).

Let $C_1, C_2, \dots, C_k$ be the clusters produced using Kruskal's MST algorithm, stopping after $n-k$ edges are added. We wish to show that this is an optimal set. By contradiction, assume that $C_1', \dots, C_k'$ is an optimal set and $\{C_1, \dots, C_k\} \neq \{C_1', \dots, C_k'\}$. There must exist some subset C_r which is not a subset of any C_i', because if there were not, either $\{C_1, \dots, C_k\} = \{C_1', \dots, C_k'\}$ or the clusters $C_1, \dots, C_k$ would fail to cover every node.



Let C_s' and C_t' be two clusters which intersect C_r. Because C_r is connected, there must exist some edge e connecting $v_x \in C_s' \cap C_r$ and $v_y \in C_t' \cap C_r$.

Kruskal's algorithm greedily processes every edge in nondecreasing order. Therefore, because e is added before any external edge (with respect to the C_i's) is considered, e must be of smaller weight than any external edge with respect to the C_i's.

But e is an external edge with respect to the C_i''s. Because it is of smaller weight, this contradicts our assumption that the C_i''s are an optimal clustering (for which the minimum external edge must be maximized) by definition.

Therefore, Kruskal's algorithm produces an optimal k-clustering. ∎

20

Name(last, first): Name Withheld
UPE-Certified Name Withholder™

4. Consider a sequence of n real numbers X = (x1, x2, …, xn).
a. Design an algortihm to partition the numbers into n/2 pairs. We call the sum of each pair S1, S2, …, Sn/2. The algorithm should find the partitioning that minimizes the maximum sum.
b. Analyze the time comlpexity of your algorithm.

(a) Algorithm:

10

```
1. Sort X by nondecreasing order.
2. Let i = 1.
3. Take the minimum and maximum in X and put them in partition i.
4. Increment i.
5. If there are still numbers remaining, go back to step 3.
```

5

(b) We can use an efficient divide-and-conquer algorithm like mergesort to perform step 1 in $O(n \log n)$ time.
The operations in the "loop", specifically finding the minimum and maximum, can now be done in constant time. The loop runs $\frac{n}{2}$ times, or in $O(n)$.
Therefore, the algorithm takes $O(n \log n + n) = \boxed{O(n \log n) \text{ time.}}$

5

(c) Consider the following greedy algorithm A', which is functionally equivalent to the one given above: (call it A):

```
A': 1. Let i = 1.  and put them in partition i.
    2. Pick the minimum and maximum
    3. Increment i.
    4. If there are still numbers remaining, go back to step 3.
```

A' outputs a solution $S = (S_1, \ldots, S_n)$. Consider an optimal solution $O = (O_1, \ldots, O_n)$, $S \neq O$. There exists some first k such that $S_k \neq O_k$. Let $S_k = (a_k, b_k)$ and $O_k = (a_k, b_\ell)$. There must be some S_ℓ, $\ell > k$, that contains b_ℓ, $S_\ell = (a_\ell, b_\ell)$. Because S adds pairs in increasing minimum/decreasing maximum order, it must be the case that either $a_k + b_\ell > a_k + b_k$ or $a_k + b_\ell > a_\ell + b_\ell$. Thus $O_k > S_k$ or $O_k > S_\ell$, and S must be at least as optimal as O is. Therefore, S is an optimal solution and A is a correct algorithm. ∎

2.

Name(last, first): Name Withheld

UPE-Certified Name Withholder™

5. Let G be a DAG and let K be the maximal number of edges in a path in G.
a. Design an algorithm that divides the vertices into at most k+1 groups such that for each two vertices v and w in the same group there is no path from v to w and there is no path from w to v.
b. Analyze the time complexity of your algorithm.

(a) 1p

1. Compute the in-degree of every vertex.
2. Let i = 1.
3. Find all sources and add them to partition i.
4. Remove the sources and update the source list.
5. Increment i.
6. If there are still vertices remaining, go to Step 2.

(b) 5

Let us use an array of n integers to maintain the in-degree of each node.
To compute the in-degrees, we process each edge once in constant-time operations. This step takes O(e) time,

Finding all sources takes O(n).
Globally, updating the source list takes O(e), for if we charge each edge for the updating operation, each edge is charged for constant time.
Therefore, the algorithm takes $\boxed{O(e+n) \text{ time.}}$

(c) 5

We prove that no two vertices v, w, with a path between them are in the same partition. At any point in the execution, we choose all sources and put them in partition i. If v and w have a path between them, then one of them must have in-degree at least 1, and cannot be a source.
We prove that there are at most k + 1 groups. Consider the longest path. The first vertex is a source and is added to partition 1. Then the next becomes a source and is added to partition 2. Because K is the maximal number of edges in a path, the last vertex is added to partition k + 1. If there were more partitions, this contradicts the assumption that the longest path is of length K. ■