/

94

Name(last, first): Lindberg, Tyler

Discussion. 1A

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **ID :** 204447717

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)
- Choose an arbitrary unvisited node a as the root
- If a is the target node return a
- Add all neighbors of a to a node stack and mark the nodes as visited

+15

- Pop a node b off the stack
    - If b is the node were searching for return b
    - Otherwise add all neighbors of b that have not yet been visited to the stack, and mark them as visited ~~[illegible]~~
- Repeat until stack is empty or target node is found
- If stack is empty and we have not visited all n nodes repeat

b) With no cycles we have an undirected tree

Running DFS on each component of the tree $C_1, C_2, \ldots, C_k$ will take $O(n_i)$ time based on the number of nodes in each component. It will not take $O(e)$ because there are no cycles, meaning we have $e = n-1$ edges.

+5

In addition it will take $O(k)$ time to search all connected components, and $k \leq n$.

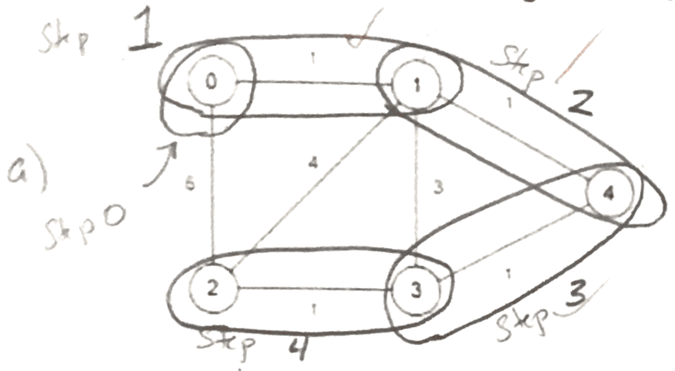
Since $\sum_{i=1}^{k} n_i = n$, then the complexity is $n+k$ which is $O(n)$.

20

Name(last, first): Lindberg, Tyler

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?

a)



Step 0: Add node 0

Step 1: Add node 1

Step 2: Add node 4

Step 3: Add node 3

Step 4: Add node 2

Choose arbitrary node zero as start

+15

b) Yes, Prim's algorithm would still find an MST. Since we are building a tree there are no cycles, avoiding many issues of negative edges.

+5

Take $e_{min} < 0$ for a certain two partitions of the graph. If there existed a MST not containing e_{min} then we could add e_{min} to the graph to create a cycle. In this case, there must be at least one other edge e' in the cycle with e_{min} such that $e_{min} < e'$, since we know e_{min} is the minimum edge between two partitions. If we replace e' with e_{min} we get a smaller overall weight, resulting in a smaller overall weight for the graph, contradicting the statement that the initial graph was an MST.

20

Name(last, first): Lindberg, Tyler

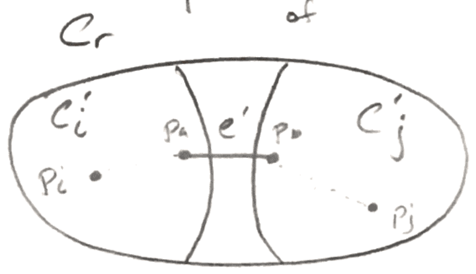
20

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).

By contradiction, if our clustering is not optimal, there must exist a clustering $C_1', C_2', \dots, C_k'$ such that $d_*' > d_*$.

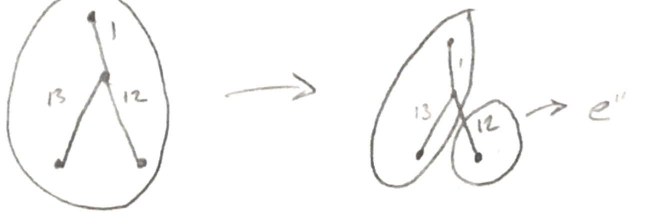
We know that at least two clusters from $C_1', C_2', \dots, C_k'$ must differ from ours with at least one element in a different cluster, otherwise it is the same as our clustering making ours optimal.

Due to this there must be at least one cluster C_r in our clustering such that it contains portions of C_i' and C_j' two clusters of the optimal set.



Take a path from p_i in C_i' to p_j in C_j', we know there must be an edge e' connecting those two clusters. e' is a part of our cluster C_r but not of any cluster in the optimal set.

However, based on our algorithm we know that whenever we split up one of our clusters into two separate ones we introduce a new edge e'' such that $e'' < d^*$, because Kruskal's picks smallest edges to be in clusters first.

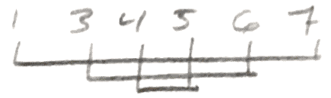


In this case e' results from splitting C_r in two, meaning with have a smaller d^*, contradicting our statement.

20

Name(last, first): Lindberg, Tyler

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) • Sort all x_i in X in non-decreasing order

• Set $i = 1$

• Make pair $S_i = \{x_i, x_{n-i+1}\}$

• Increment i by 1

• Repeat until $i = \frac{n}{2}$

5 20 21 22 30 40

10

Proof:

b) The initial sort takes $O(n \log n)$ time

Assigning the pairs take $\frac{n}{2}$ steps, since we set each pair S_i for $0 < i \leq \frac{n}{2}$, this is $O(n)$ 5

So overall, the algorithm is $O(n \log n)$

Proof: Assume there exists an algorithm more optimal than ours

Take the first S_i when our algorithm differs from the optimal S_i'.

5 This means in the optimal algorithm x_i and x_{n-i+1} are in different pairs.

Let's split X into $X_a = \{x_1, \ldots, x_{\frac{n}{2}}\}$ & $X_b = \{x_{\frac{n}{2}+1}, \ldots, x_n\}$

In our algorithm pairs must always consist of one element from X_a and one from X_b.

If ever in the optimal algorithm x_i' and x_j' are both from X_a, we know there must be another pair such that both are from X_b. If this is the case then there is a contradiction as all $x \in X_b$ are greater than $x \in X$ so x_i' and x_j' will be a larger sum than a pair where one element is instead from X_a. On back of page

Another situation is that x_i and $x_{\frac{n}{2}-i+1}$ are connected to different elements from the other set. Assume $x_i \in X_A$ and $x_{\frac{n}{2}-i+1} \in X_b$.

In this case x_i must be connected with an $x < x_{\frac{n}{2}-i+1}$, or

$x_{\frac{n}{2}-i+1}$ must be connected with an $x < x_i$.

If they are equal then it no contradiction because then both algorithms are optimal.

However, when each of these pairs occur, a larger pair is created elsewhere.

✗ For $S_1 = \{x_1, x_n\}$, if x_1 chooses a smaller $x < x_n$, then x_n must choose an $x > x_1$ because x_1 is currently the smallest x. We know that our first step can be the first step of an optimal algorithm, and by induction when regarding the subproblem of $X' = X - \{x_1, x_n\}$, we can make another optimal decision.

$\textcircled{\small 19}$ **Name**(last, first): Lindberg, Tyler

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.

K = longest path length

a) ✓ • Find all sources in G

(9) • Run topological sort on G starting at the sources we found

no explanation of how update sources

• at each step add all sources into another group g_i
• increment i by 1

• After running topological sort we will have $g_1, \dots, g_c$ such ✓ that $c \le k+1$

↳ this is because the maximal length path is K, meaning on this path there can be k+1 sources during our algorithm

(5) ✓ Proof: By contradiction

Assume at a certain step in our algorithm we have nodes v and w placed into the same group but a path connects them

However, by the definition of topological sort, if v has a path to w in G, then v will be considered as a source before w, because there is a dependency on v being reached before w. Since v must be considered before w, they would never be sources at the same time, since at least one path from v to w would make w's in-degree ≥ 1, contradicting our statement

(5) b) Topological sort is O(e+n) since G may be unconnected.
We do nothing else other than adding all nodes to groups take O(n) time.

So our algorithm is O(e+n).