73

Name(last, first): John Stucky 1D

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **ID : 704626778**

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) 1. Pick an arbitrary root node to visit.
2. At the node you are currently visiting:
(i) Look at all edges connected to the node
(ii) For each edge, follow the edge to the connected node. If this node has not been visited, call step (2) on this node. When that finishes, move onto the next node connected to the node we are currently visiting.
3. When there are no more nodes that can be visited by edges, visit an arbitrary node that has not been visited and return to step 2. Repeat step 3 until all nodes in the entire graph have been visited.

+15

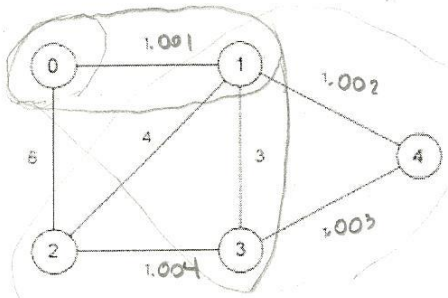
(b) During DFS, we visit each edge at most twice (once for the time we explore the edges of the first node, once for the second node). The running time is therefore O(e) in a connected graph, so O(n) since there are no cycles (at most n-1 edges). In an unconnected graph, the runtime is also O(n): in the worst case, we visit each node individually. So the runtime is O(n).

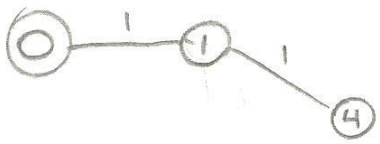
+5

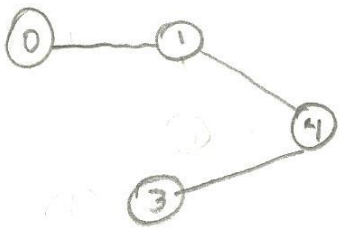
20

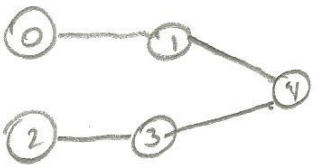
Name(last, first): John Stucky

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?









(6)

(a) Fudge weights to make weights unique.
Start with vertex labeled 0 in one partition, rest in other partition.
Apply MST Theorem.

Left	Right	min weight
0	1	1.001
	2	6
	3	∞
	4	∞

↓

Left	Right	min weight
0	2	4
1	3	3
	4	1.002

↓

Left	Right	min weight
0	2	4
1	3	1.003
4		

↓

Left	Right	min weight
0		
1	2	1.004
3		
4		

↓

Left	Right	min weight
0		
1		
2		
3		
4		

+15

15

Name(last, first): John Stucky

5

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).

Pf:

By induction:

Base Case C_i: All nodes are in one cluster. There is only one way to do this, so it must be optimal.

Inductive Step:

Assume for a fixed k that C_k is optimal. Show that C_{k+1} is optimal.

what does it mean?

Imagine C_k existing clusters that are optimal.

In Kruskal's algorithm for clustering, one additional cluster is created when the most recent edge is removed. This edge ties or has the highest weight of all edges currently added.

BWAC: If a different edge is removed, then the minimum edge between two different clusters will be less than that of the one removed by Kruskal's, so this will have smaller maximum cluster distance which is not optimal. (a contradiction).

If a cluster off by more than one edge from C_k is optimal for C_{k+1}, then multiple edges will have to be added and removed to obtain C_{k+1}. One of these removed edges will have smaller weight than the edge removed by Kruskal's so it won't be optimal (a contradiction).

20

Name(last, first): John Stucky

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 complexity of your algorithm.

(a) Greedy Approach: Pair the largest number with the smallest one. Remove these two and pair the remaining largest and smallest.

Algorithm:
Use Mergesort to
① Sort the numbers in non-decreasing order into Y ($Y_1 \leq Y_2 \leq \ldots \leq Y_n$).
② For i ranging from 1 to $\frac{n}{2}$:

10

Sum of pair $S_i = Y_i + Y_{n+1-i}$ (pair the i^{th} element with the i^{th} element from the back).

(b) Sorting can be done in $O(n \log n)$. Use Mergesort. We can do step (2) from (a) in $O(n)$ time since we are looking at each element once to add it to a pair. The total running time =

5

$$O(n \log n + n)$$

$$\Rightarrow O(n \log n)$$

Pf: BWAC
Assume there is a smaller, optimal maximum sum. Let x_j and x_k be the largest sum pair in the original algorithm. Without loss of generality, suppose $x_j \leq x_k$. we know that x_j is paired $z_k < x_k$ and x_k is paired with $z_j < x_j$ in the better solution (where z_j, z_k are $\in X$).

5

Is $z_k < x_j$?
If yes, we have $z_j, z_k < x_j \leq x_k$, but by our algorithm we know x_j or x_k would've paired with z_j or z_k instead of x_k or x_j, a contradiction.
If no, $z_j < x_j \leq z_k < x_k$, so x_k would've been paired with z_j before x_j, a contradiction.

Name(last, first): John Stucky

(13)

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) Define a source to be a vertex with no edges going into it (in-degree of 0).

① List all the sources from the graph. Add them to a new group.
② Remove all the edges leaving these sources from the graph.
③ Update the list of sources to be those not yet in a group that have an in-degree of 0.
Repeat step 1 until there are no nodes left.

(11)

(b) We can contribute time according to when a node is visited or when an edge is removed. There are O(n) nodes to visit and O(e) edges to remove. However, since the graph is acyclic, there are at most n-1 vertices. so the running time is

$= O(n)$

$O(n+e)$

(3)

no proof :(