(100)

Name(last, first): ____________

UCLA Computer Science Department

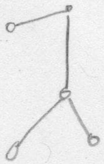
CS 180 **Algorithms & Complexity** **ID :** ____________

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) - push starting node to a stack (last in, first out)
starting node: if given, choose that node, if not, choose one arbitrarily
- while the stack is not empty:
  - pop the top node off the stack
  - push all of this node's unvisited neighbors onto the stack
- if we still have unvisited nodes, pick one arbitrarily as our new starting node and push onto stack, repeat previous while-loop steps

+15

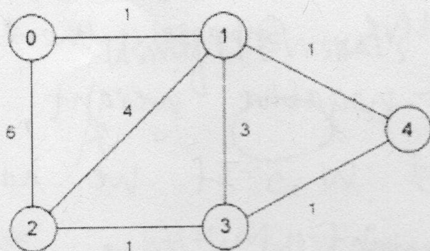
(b) We will process each node once when we pop it off the stack, so this is $O(n)$, and check each edge twice - once for each node that's adjacent to it, so this is $O(2e) = O(e)$. If there are no cycles, then there are only a maximum of $e = (n-1)$ edges. Thus the time complexity is $O(n+e) = O(n+n-1) = O(2n-1) = O(n)$.

+5

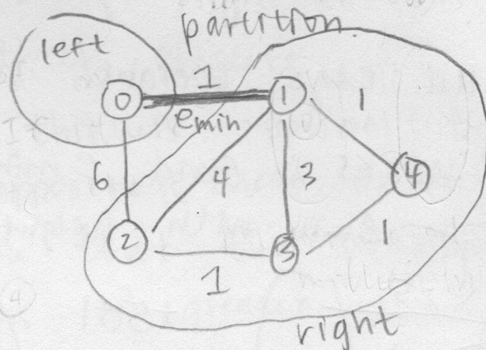
(20)

Name(last, first): ___

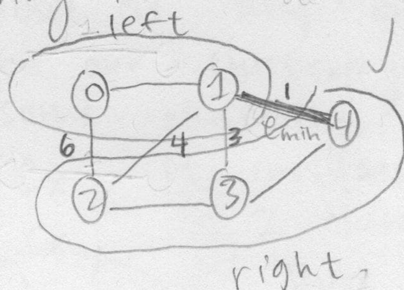
2 . a. Use Prim's MST algorithm to find an MST in this graph. Show each step on the graph shown below.
b. If some edges were negative would the algorithm still find an MST. Why?



(a) - create 2 partitions. arbitarily choose 1 node and put it in the left partition, adding it to our MST. Put all other nodes in the right partition



- find emin between any node in left and any node in right. By MST theorem, emin must be in the MST. In this case, emin = (0, 1) with weight 1 so we add emin and node ① to our MST, moving it to the left partition.

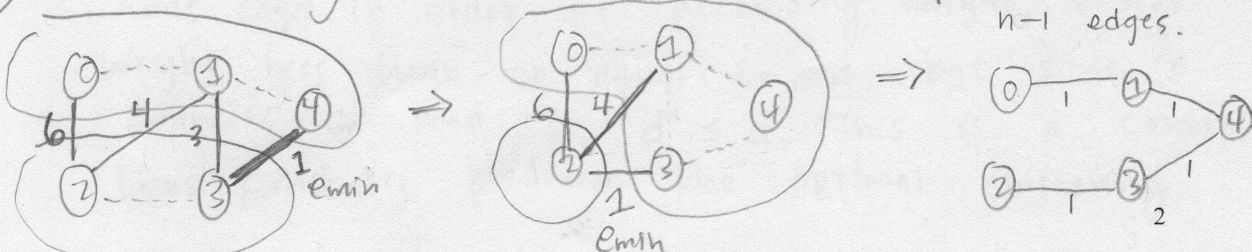


+15

- repeat previous step. we find emin, and if it does not create a cycle, add emin to the MST and move the adjacent node to left partition.

repeat until we've added n-1 edges.

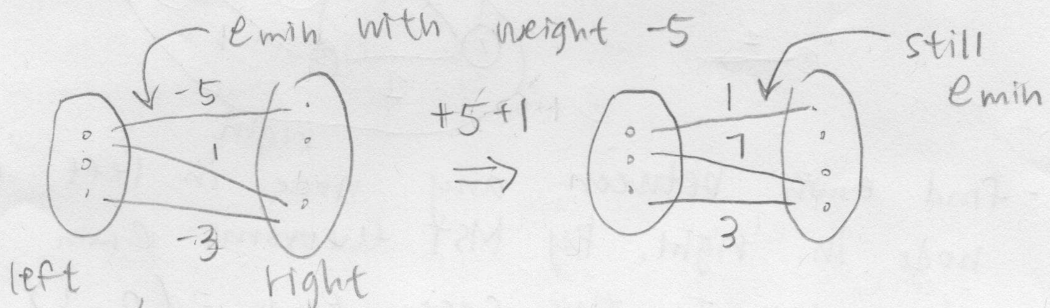
+20

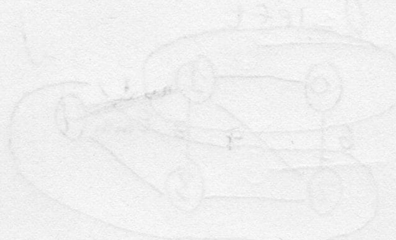


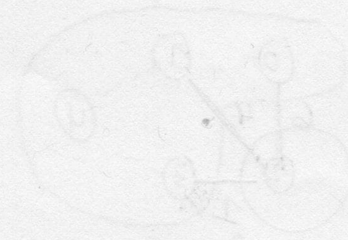
(back)

time complexity = $O(\log(e))$ to insert each edge into minheap so that it's $O(1)$ to find min. e edges, so $O(e \log e)$. Process each edge once, so we have $O(e + e\log e) = O(e \log e)$.

(b) Prim's still works because e_{min} doesn't change when the weights are negative. Suppose there is some minimum edge with negative weight, call this edge e with weight w. If we add $[(-w)+1]$ to every edge weight in the graph, all edge weights are positive and Prim's works, and e_{min} never changed.







Name(last, first): ________________

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).

Using Kruskal's MST, we find some clustering

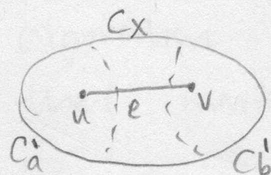
A = C_1 C_2 … C_K

with the maximum minumum distance d between clusters.

We prove that this is optimal by contradiction. Assume the opposite: there exists a better clustering such that the maximum min distance d' between clusters is greater than d. $d' > d$

B = C_1' C_2' … C_K'

If this clustering is different, there must exist at least one cluster $C_x \in A$ such that it is not a subset of any one $C_x' \in B$. If this were not true, then our clustering either omitted nodes or is the same clustering. We see that for this cluster C_x, it includes nodes from at least 2 clusters C_a' and C_b'.



There is some edge e in C_x such that $u \in C_a'$, $v \in C_b'$, and $e = (u, v)$. Since Kruskal's adds edges in order of ascending weight, e has weight less than or equal to d. But since e connects C_a' and C_b', $d' \leq d$. This is a contradiction. Thus. Kruskal's produced the optimal clustering.

20

Name(last, first): _______, _______

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 X using merge sort
- create pair (min(X), max(X))
remove min(X) and max(X) from X.

10

- repeatedly create pairs (min(X), max(X)) until we have $\frac{n}{2}$ pairs.

Proof: We have maximum sum $S = x_i + x_j$ for some $x_i, x_j \in X$. We prove by contradiction.
Assume the opposite. Suppose there is some pairing such that the max sum $S' < S$.

5

Then x_i must have been paired with some $x_k < x_j$, and x_j must have been paired with some $x_e < x_i$. But x_i was min(X) and x_j was max(X) at the iteration on which they were paired. Then we cannot find an $x_e < x_i$, since there are no elements $< x_i$. This is a contradiction. Thus our algorithm correctly finds the partitioning that minimizes the max sum.

(b) mergesort takes $O(n\log n)$
Since the set is sorted, we can take the first and last elements in $O(1)$. We do this $n/2$ times, so it's $O(\frac{n}{2})$.

5

Thus our total time complexity is
$O(n\log n + n/2) = O(n\log n)$.

2 /

Name(last, first): ______, ______

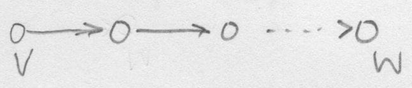
5. Let G be a DAG and let K be the maximal number of edges in a path in G.
a. Design an algorithm that divides the vertices into at most k+1 groups such that for each two vertices v and w in the same group there is no path from v to w and there is no path from w to v.
b. Analyze the time complexity of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

(a) - Run topological sort
- at each iteration, take all the nodes that are sources and put them in a group.

more explanation for next time plz :)

(1)

Proof: We run our algorithm and find some grouping with $\leq k+1$ groups. We want to show there are no paths between edges in the same group. We prove by contradiction and assume the opposite. There is some group A such that there is a path between 2 nodes:



But if there is a path from V to W, W must have in-degree of at least 1. Then W was not a source at this iteration and is not in the same group as V. This is a contradiction, thus there are no paths between V and W.

(5)

We get k+1 groups at most because there are k+1 nodes in the longest path, and at each iteration one node in the path will be a source - otherwise there would be a longer path.

(5)

(b) topological sort takes $O(n+e)$ so total time complexity is $O(n+e)$.