70 no chge

Name(last, first): [redacted]

1c

U C L A Computer Science Department

[redacted]

CS 180 **Algorithms & Complexity**

Midterm **Total Time: 1.5 hours** **October 31, 2017**

Each problem has 20 points.

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

20

1 Consider a set of intervals I1, …, In. **a.** Design a linear time algorithm (assume intervals are sorted in any manner you wish) that finds a maximum subset of mutually non-overlapping intervals. **b.** Prove the correctness of your algorithm.

(a) · Let the input I be sorted by ascending interval end times.
· Let $X = [I_1]$ (or return [] if I_1 does not exist).
· For I_i from $i=2$ to n:
  · If I_i does not overlap with the last interval in X, then it does not overlap with any interval in X so add it to X.
· Return X.

(b) · This algorithm finds a subset of mutually non-overlapping intervals because it checks for overlap before adding any interval to X.
· This algorithm finds the maximum such subset. Suppose by contradiction that there is another subset of mutually non-overlapping intervals Y, $|Y| > |X|$.
  · Let X_i, Y_i be the first interval that is different between X and Y.
  · Since our algorithm picks the soonest-ending interval after the previous, we can replace Y_i with X_i in Y to obtain Y' without creating any overlaps or making Y' less optimal than Y.
  · Since $X_i = Y_i'$ and Y' is at least as optimal as Y, set $Y = Y'$ and induct.
  · But then Y converges to X for the first $|X|$ intervals, so in order for $|Y| > |X|$, Y must have non-overlapping interval(s) beyond the last interval in X.
  · But our algorithm would've added such interval(s) to X. This is a contradiction.

0

Name(last, first): [redacted]

2 . a. Design an efficient algorithm better than O (n^2) to be used in sparse graphs for finding the shortest path between two vertices S and T in a positive weighted graph. **b.** Justify the correctness of your runtime analysis.

(a) We can use BFS to find the shortest path.

- Create a BFS queue q = [s].
- While q is not empty:
  - Pop the front node from q. If it is T, then we are done. Follow the noted parents back to s to obtain the shortest path.
  - If it is not T, then add T's unvisited neighbors not already in q to q, noting that T is their BFS tree parent.

X

- T was not found.

— |V| = # nodes, |E| = # edges

(b) BFS runs in $O(|V|+|E|) < O(|V|^2)$ time because $|E| \ll |V|^2$ since the graph is sparse.

- BFS visits each node up to once – to read its neighbors.
- BFS crosses each edge up to once – to check the state of the neighbor and possibly add it to q.

X

- The backtrace also visits each node and edge no more than once.

Proof: Suppose by contradiction that BFS does not find the shortest path from S to T. That is, the length of the shortest path is less than T's level in the BFS tree. Then somewhere in this hypothetical shortest path there is an edge from level i to j, j > i+1. But BFS would have visited the node at level j immediately after the node at level i, making j = i+1. This is a contradiction.

X

20 Name(last, [redacted]

just return the maximum sum

3. Consider a sequence of positive and negative (including zero) integers. Find a consecutive subset of these numbers whose sum is maximized. Assume the weight of an empty subset is zero. **a.** Design a linear time algorithm. **b.** Prove the correctness of your algorithm.
Example: For the sequence $\overset{4}{\cancel{2}}$ -3 5 -12 the maximum sum is $\overset{6}{\cancel{4}}$.

~~be~~ e ~~be~~ e

(a)
```
• Denote the input list $L_1, \dots, L_n$.
• Let sum = 0, max = 0, begin = 1, end = 1.
• While end ≤ n:
    • sum += $L_{end}$.
    • If sum ≤ 0, set begin = end + 1 and sum = 0.
    • Else, if sum > max, set max = sum.
    • Increment end.
• Return max.
```

(b) • By contradiction, suppose there is another consecutive subset [x, y] with a greater sum than our interval [b, e]. Then one or more of the following applies:

- x < b, so [x, y] starts with a negative number.
- x > b, so [x, y] excludes a non-negative number at the beginning

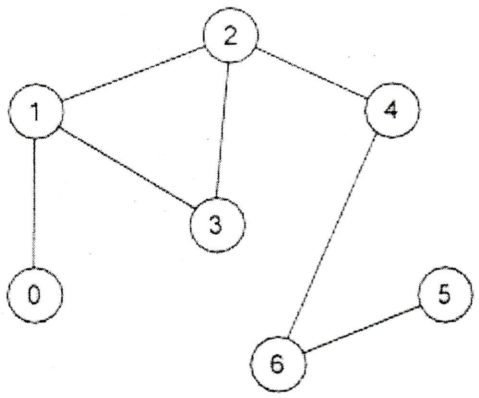
by our algorithm's checks {
- y < e, but the sum could have been made at least as large by increasing y to e.
- y > e, but the sum was not made larger by increasing e to y

In any of these cases, [x, y] does not have a greater sum than [b, e] — a contradiction.

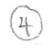
Name(last, first): [redacted]

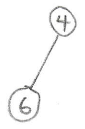
20

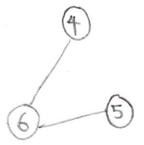
4. Consider an unweighted graph G shown below. **a.** Starting from vertex **4,** show every step of DFS along with the corresponding stack next to it. **b**. What is the run time of DFS if the graph is not connected (no proof is necessary)?

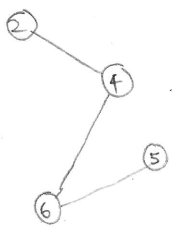


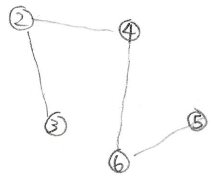
(a) graph stack

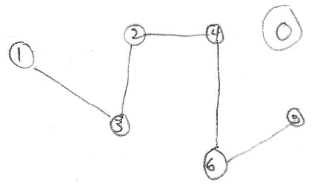
1)  2, ⑥

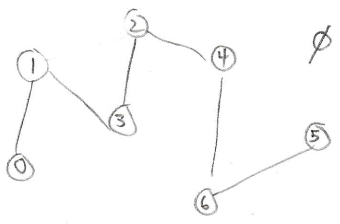
2)  2, ⑤

3)  ②

4)  1, ③

5)  ①

6)  ⓪

7)  ∅

(b) Still $O(|V| + |E|)$.

$|V| = \#$ nodes

$|E| = \#$ edges

19

Name(last, first): [redacted]

5. Consider a binary tree (it is not necessarily balanced). The tree is not rooted. Its diameter is the distance between two vertices that are furthest from each other (distance is measured by the number of edges in a simple path). Design a linear time algorithm that finds the diameter of a binary tree.

We use a DFS approach to find the longest simple path in the tree.

- Arbitrarily pick a root node s.
- If s has no neighbors (other than the node from which DFS was called on s, if applicable), then return (longest = 0, longest_extendable = 0).
- Otherwise, call this DFS on s's neighbors (other than the node from which DFS was called on s, if applicable. Return (longest = max of (any child's longest +1 or the sum of any two children's longest_extendable +2), longest_extendable = max of any child's longest_extendable + 1).
- Return the root node's longest value.

Proof:

- Suppose by contradiction there is a longer path. Then, this path uses edges beyond the maximum depth detected by DFS (impossible), or it is not simple (a contradiction).

time analysis