79

Name(last, fi[redacted]

UCLA Computer Science Department

CS 180 — Algorithms & Complexity — October 31, 2017

Midterm — Total Time: 1.5 hours

Each problem has 20 points.

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

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.

5+10 = 15

a.
- Greedy Algorithm.
- ~~How the intervals are sorted doesn't matter,~~ X
- While there are intervals left in the set (S)
  - pick the one that ends first, I_i
  - add it to max subset
  - delete all intervals that overlap with I_i, (including I_i itself)
- We get the maximum subset of mutually non-overlapping intervals, M.
- Since we go through the set only once, Complexity is O(n) – linear

5

b.
- Assume the optimal set is different, we call it M_o
- We sort M_o & M by time
- base case: $M_o[1]$ & $M[1]$ are different, and are only elements.
  - M[1] ends before M_o[1]
- Assume first i-1 interval are the same, $M_o[i] \neq M[i]$.
  - By our Algorithm, M[i] ends before $M_o[i]$
  - So replace $M_o[i]$ by M[i] still results in optimal solution.
- By induction, we prove M results in optimal solution.

7

Name(last, [redacted]

not too many edges.

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. • Set values for vertices as shortest distance from S.

• Initially, S=0, all other vertices = ∞

• while value has not assigned to T yet. X.

• Assign value to the node tha has minimum value of any neighbor + connecting edge.

• the value assigned to T is shortest distance

-8

• From T, we traverse through vertices in decreasing value, and reach S. we would get the shortest path from S to T.

b. • We go through each vertex at most twice (one for shortest distance, one for actual path) — O(n)

• Sparse graph — not too many edges, consider edge number as constant.

• so Check the min vertex+edge value sum is also O(n)

• Thus Run time = O(n) .X

-5

Name(la[redacted]

20

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 -3 5 -12 the maximum sum is 4.

a. • weight of empty subset is 0 → so the sum should be >0 in order to have elements in the set

- Set variables max_endinghere = max_sofar = 0
- let the sequence be called A, so we have A[0] ~ A[n]
- Iterate through the current sequence, with current element as i
  - max_endinghere = max_endinghere + A[i]
  - if max_endinghere < 0
    - max_endinghere = 0
  - if max_endinghere > max_sofar
    - max_sofar = max_endinghere
- Finally, after this loop, max_sofar is the max sum (MS)

✓

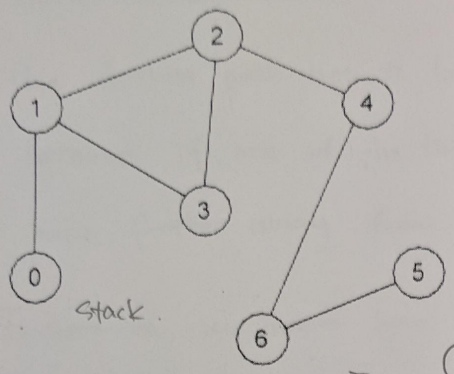
b. • Assume there exists an optimal subset that is longer than MS, we call it MSo

- Then the consecutive sequence's sum is larger than then one in MS.
- max_endinghere only resets itself to 0 when it is negative since empty set > neg
- Thus, some point in our algorithm, the max_endinghere would achieve max in MSo
- Since we set max_sofar = max_endinghere when max_endinghere > max_sofar.
- Then, whenever we achieve the max in MSo, we would set it as the Max in MS, MS = MSo.
- MS is thus optimal.

Name(last, fi[redacted]

20

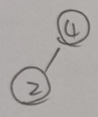
4. Consider an unweighted graph G shown below. **a.** Starting from vertex **4,** show every step of DFS along with the corresponding stack next to it. **b.** What is the run time of DFS if the graph is not connected (no proof is necessary)?

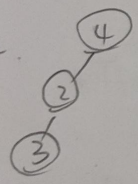


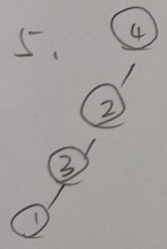
Visited . Stack .

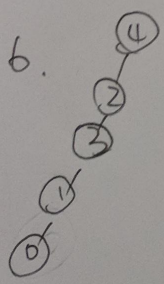
a). 1. empty | 4 |

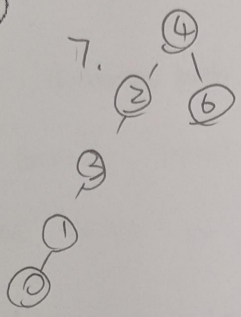
2.  | 2 | 6 |

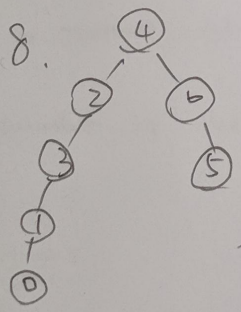
3.  | 3 | 1 | 6 |

4.  | 1 | 6 |

5.  | 0 | 6 |

6.  | 6 |

7.  | 5 |

8.  empty

The DFS order is not unique.

b. • Not connected

• if there are n vertices & m edges

• run time is O(m+n)

17 Name([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.

Observation:

- The furthest path has to be between two leaves of the tree, because if one of the two ends belongs to parent, there exists node further away from the parent
- Furthermore, no matter how we construct the binary tree, one end of the furthest path has to be one of the leaves in the bottom level, (any one of them, because their distance to source is the same)

Algorithm

- pick any leaf from bottom level BFS? O(m+n)
- use it as source to contruct BFS tree. O(m+n)
- Then the level number is the diameter of this binary tree
- Run time is O(m+n)