CS180 Midterm Exam

KEATON HEISTERMAN

TOTAL POINTS

83 / 100

QUESTION 1

1 Q1 20 / 20

✓ **- 0 pts Correct**

- 20 pts Not understanding the problem

- 5 pts Not a rigorous proof

- 15 pts Wrong algorithm

- 5 pts Need a second pass to determine it is a majority vote.

- 10 pts Click here to replace this description.

- 10 pts Optimal time complexity should be O(n)

- 10 pts No proof or wrong proof

- 5 pts Wrong or not optimal time complexity.

- 5 pts Mix hash map and Boyer-Moore algorithm

- 5 pts It is not clear to say that "choose 2 arbitrary distinct votes". Too vague.

- 5 pts count needs to be reset/increased.

QUESTION 2

2 Q2 15 / 20

- 0 pts Correct

✓ **- 5 pts Wrong correctness proof.**

- 5 pts Wrong Time Complexty

- 10 pts Wrong algorithm.

- 5 pts Partial Problem in Algorithm

QUESTION 3

3 Q3 15 / 20

- 0 pts Correct

- 2 pts Partial problems in correctness proof

- 2 pts Partial Problems in algorithm.

✓ **- 5 pts Wrong correctness proof.**

- 5 pts Wrong time complexity analysis.

- 2 pts Partial algorithm problem. Your algorithm cannot work on disconnected graph. If you start with a single isolated node, it will terminate without visiting all other nodes.

- 10 pts Wrong algorithm.

- 2 pts Partial problem in time complexity analysis.

- 2 pts Partial problem in correctness proof.

- 20 pts No submission

QUESTION 4

4 Q4 18 / 20

- 0 pts Correct

✓ **- 2 pts Failed to adjust algorithm for target runtime**

- 5 pts Did not propose algorithm to traverse

- 5 pts Incorrect or unattempted Runtime Analysis

- 5 pts Incorrect or unattempted Correctness proof

- 5 pts Did not indicate correct final path

QUESTION 5

5 Q5 15 / 20

- 0 pts correct

- 10 pts This just shows that BFS traverses the entire tree but does not mention the levels and how it relates to the distance.

✓ **- 5 pts Need to provide the cases for both: shortest and longest path. Only one given.**

- 5 pts Proof is not clear/undefined terms

- 5 pts Does not clearly state the relation between levels and the distance.

- 2 pts Minor errors such as stating an assumption with no proof

- 20 pts incorrect

Name(last, first): Heisterman, Keaton

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** **UID : 905 837 242**

Midterm **Total Time: 90 minutes** **April 25, 2022**

Each problem has 20 points: 5 problems, 5 pages

All algorithm should be described in bullet format (with justification/proof). You cannot quote any time complexity proofs we have done in class: you need to prove it yourself.

Problem 1: Consider **m** candidates **n** votes. A majority is the person with more than **n/2** votes. Design an algorithm for finding a majority. Prove its correctness. Analyze its complexity.

Algorithm:

- You have a list with each node having a candidate, m, and each node represents a vote
- Set variable maj-cand to be the root node of the list. This variable has a counter, and a candidate stored. Set the counter to 1
- for each node n in list
  - if the maj-cand counter is zero
    - Set maj-cand candidate to n candidate and set counter to one
  - else if n candidate is equal to maj-cand candidate
    - increase maj-candidate counter by 1
  - else if candidate is not equal to maj-cand candidate
    - decrease maj-cand counter by 1
- if maj-cand counter is not zero
  - go through list again and count how many times maj-cand appears (how many votes)
  - if this is greater than n/2, return this candidate as the majority
- if you get here, there is no majority, so return no candidate (-1 to represent that)

Proof:

- This algorithm essentially removes two votes at a time. This will never destroy a majority, but could create one (shown in class). Thus after removing all votes, we are left with one potential candidate. From here we count if that candidate has a majority.

Time complexity on back

Time complexity: We loop through each vote, so this is O(n) time. Then, we count all times potential candidate is voted, so we loop again through all votes. Thus total time complexity is O(n).

1 Q1 **20 / 20**

✓ **- 0 pts** **Correct**

- 20 pts Not understanding the problem

- 5 pts Not a rigorous proof

- 15 pts Wrong algorithm

- 5 pts Need a second pass to determine it is a majority vote.

- 10 pts Click here to replace this description.

- 10 pts Optimal time complexity should be O(n)

- 10 pts No proof or wrong proof

- 5 pts Wrong or not optimal time complexity.

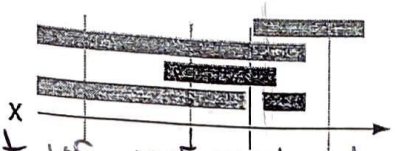
- 5 pts Mix hash map and Boyer-Moore algorithm

- 5 pts It is not clear to say that "choose 2 arbitrary distinct votes". Too vague.

- 5 pts count needs to be reset/increased.

Name(last, first): Heckterman, Keaton

Problem 2: Consider a set of intervals/tasks. A. Design an algorithm that finds the maximum number of mutually overlapping intervals/tasks. B. Analyze the time complexity of your algorithm. In the example below the answer is 3.



- Set most_overlap to zero
- Sort two lists, one on start time and one on end time.
- While lists are not empty
  - take interval w soonest finish time, f_i
  - count how many intervals start before f_i
  - if this is greater that current val in most_overlap, set most_overlap to this count
  - remove this interval from both lists, as well as any other intervals that end at the same time as f_i

Proof suppose some greater number exists.
Then, there must be more intervals that finish after some interval, i_n, while also starting

Time complexity: At most we must visit every interval for earliest finish, while also having to look at every interval that starts before that time. If we have n intervals, at most n-1 other intervals start before the earlrest finish. Thus, since we do this for all intervals, we have $O(n^2)$ time compexity

2 Q2 15 / 20

- **0 pts** Correct

✓ **- 5 pts** **Wrong correctness proof.**

- **5 pts** Wrong Time Complexty
- **10 pts** Wrong algorithm.
- **5 pts** Partial Problem in Algorithm

Name(last, first): ~~Kea~~ Hesterman, Keaton

Problem 3

Give an algorithm for determining if a graph is two-colorable, i.e. if it is possible to color every vertex red or blue so that no two vertices of the same color have an edge between them. Your algorithm should run in time O(V + E), where V is the number of vertices and E is the number of edges in the graph. You should assume that the graph is undirected and that the input is presented in adjacency-list form.

- Set all vertices to be white
- while some vertex is white
  - Select a white vertex, n
  - color n red
  - add every vertex adjacent to n to queue, and color them blue
  - Set color to red
  - while queue is not empty:
    - remove all vertices from queue and put in a process list
    - for each vertex in process list, n
      - ~~Set vertex to color~~
      - for each vertex, p, adjacent to n
        - if color of p is equal to color of n, return not two-colorable
        - Set p to color, and add it to queue
    - if color is red, set it to blue, else if color is blue, set it to red
- If you get here graph is two-colorable, so return that

Proof:

~~Proof~~ This algorithm is valid due to the structure of BFS. For a Graph to be two-colorable, for any vertex, if all ajacency vertices are another color, for all vertices, it is valid. if not, it is not. BFS will look at all possible adjacent vertices.

~~time: O(V+E) since BFS visits every edge & every~~

Time complexity: O(V+E)

BFS visits either every node or every ~~edge~~ edge, which ever is greater, initializing color variables are O(V)

Thus in total we have O(V+E) complexity.

3 Q3 15 / 20

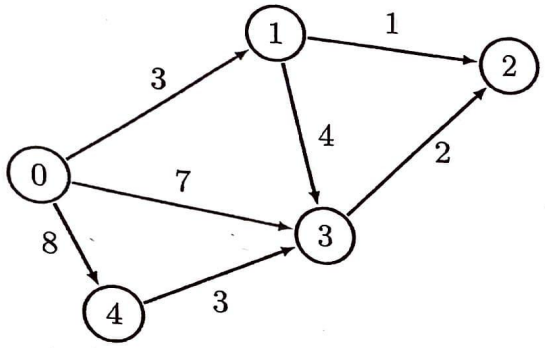
- **0 pts** Correct
- **2 pts** Partial problems in correctness proof
- **2 pts** Partial Problems in algorithm.

✓ **- 5 pts** **Wrong correctness proof.**

- **5 pts** Wrong time complexity analysis.
- **2 pts** Partial algorithm problem. Your algorithm cannot work on disconnected graph. If you start with a single isolated node, it will terminate without visiting all other nodes.
- **10 pts** Wrong algorithm.
- **2 pts** Partial problem in time complexity analysis.
- **2 pts** Partial problem in correctness proof.
- **20 pts** No submission

Name(last, first): Heisterman, Keaton

Problem 4: What is the shortest weighted path (some of the weights on the edges has to be maximized) between vertex 0 and vertex 2? Design an O(V+E) algorithm that finds the hortest path between two vertices in a connected DAG, where V is the number of vertices and E is the number of edges.



Shortest path between vertex 0 and vertex 2 is 4
The path is (0, 1, 2).

- ~~For each node, set distance to infinity and finished to false~~
- if asked for shortest from node m to n, run:
  - Shortest_path(m,n)
  - Shortest_path(n,m)
  - return whichever of these calls returns a lower distance (infinity for both means no path)
- Shortest_path(n,m) is as follows:
  - For each node, set distance to infinity and finished to false, and last vertex to null
  - Set n distance to zero and finished to true
  - while m is not found all all non-infinity distance nodes are not found
    - take vertex v with lowest distance
    - set v to finished
    - for each out edge from v not to a finished vertex y
      - set y distance to v + edge weight
      - set y last vertex to v
      - if y is equal to m, signal m is found and return the list of all previous vertices stored, as well as the distance in total
  - no path available, so return distance as infinity

Proof

This is Djiktra's algorithm for shortest path, but tweaked for directed paths (need to check both directions). We have proved this algorithm gives shortest path in class. We know this is then correct if we compare both directions

Time analysis

- initializing distances and finished visits every node, so that takes O(V) time.
- The while loop will visit at most every edge once, so that is O(E) time.
- Thus in total we have O(V+E) time complexity

4 Q4 18 / 20

- **0 pts** Correct

✓ **- 2 pts** **Failed to adjust algorithm for target runtime**

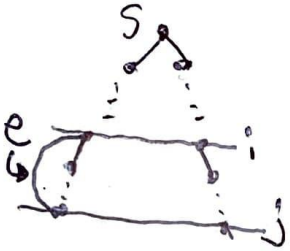
- **5 pts** Did not propose algorithm to traverse
- **5 pts** Incorrect or unattempted Runtime Analysis
- **5 pts** Incorrect or unattempted Correctness proof
- **5 pts** Did not indicate correct final path

Name(last, first): Herterman, Keaton

Problem 5: Prove that the Breadth first tree starting from a vertex s, gives you the distance between s and all other vertices.

Proof

Suppose to the contrary some shorter path exists from s to some vertex v. The breadth first tree will have levels from s, with the level representing distance from s. ~~Thus there must be some path~~ In order for there to be a shorter path, there must be an edge e from some level l_i to l_{i+j} for some $i \geq 0$ and $j \geq 2$. If this edge e were to exist, we can observe that



it is adjacent to a vertex in level l_i. BFS will then explore this edge, as it explores all edges at a given level before moving to the next. This means that this edge must be in the Breadth first tree, and this other level l_{i+j} must be level l_{i+1}. This is a logical contradiction.

Therefore since there cannot be any shorter path from s to any other vertex in the BFS tree, we have proven the BFS tree gives distance between all other vertices and s, if we define distance as its level in the tree.

5 Q5 15 / 20

- **0 pts** correct
- **10 pts** This just shows that BFS traverses the entire tree but does not mention the levels and how it relates to the distance.

✓ - **5 pts** **Need to provide the cases for both: shortest and longest path. Only one given.**

- **5 pts** Proof is not clear/undefined terms
- **5 pts** Does not clearly state the relation between levels and the distance.
- **2 pts** Minor errors such as stating an assumption with no proof
- **20 pts** incorrect