# 23F-COM SCI-180-LEC-1 Midterm

JOYCE CHEN

TOTAL POINTS

## 75 / 100

QUESTION 1

*1* **17 / 20**

- **- 0 pts** Correct
- **- 8 pts** (a) incorrect
- **- 6 pts** (b) incorrect
- **- 6 pts** (c) incorrect
- **- 4 pts** (a) Missing details
- **- 3 pts** (b) Missing details
- ✓ **- 3 pts** *(c) Missing Details*
- **- 4 pts** made-up midterm part 1: Assuming there is only one vertex with in-degree=0.
- **- 6 pts** made-up midterm part 3: time complexity incorrect

QUESTION 2

*2* **14 / 20**

- **+ 0 pts** Completely incorrect
- **+ 3 pts** Click here to replace this description.
- **+ 7 pts** small progress
- **+ 11 pts** Wrong, working for some cases
- ✓ **+ 14 pts** *Moderate error*
- **+ 16 pts** Great progress towards correct algorithm
- **+ 18 pts** Minor error
- **+ 20 pts** Correct
- **+ 8 pts** made-up midterm part A: correct
- **+ 3 pts** made-up midterm part C: Final answer correct.

QUESTION 3

*3* **18 / 20**

- **+ 0 pts** Totally wrong
- **+ 9 pts** Major error
- **+ 14 pts** Moderate error
- ✓ **+ 18 pts** *Minor error*
- **+ 20 pts** Complete

QUESTION 4

*4* **6 / 20**

- **+ 20 pts** Correct.
- **+ 18 pts** Correct, missing minor details or has minor errors.
- **+ 9 pts** Correct direction, missing critical details or has major errors.

These points are for incomplete or incorrect solutions that at least recognized that a DFS like traversal was required for the problem.

- **+ 5 pts** Designed algorithm that may work but is not linear-time.
- ✓ **+ 3 pts** *Incorrect solution/algorithm does not work on all valid graphs.*
- ✓ **+ 3 pts** *Correct runtime analysis for incomplete or wrong algorithm given.*
- **+ 2 pts** Attempted.
- **+ 0 pts** Not attempted or wrong.
- **+ 15 pts** made-up midterm: Not clear on

describing algorithm.

QUESTION 5

*5* **20 / 20**

✓ **+ 20 pts** *Correct.*

**+ 18 pts** Correct, did not show each step sufficiently.

**+ 18 pts** Mostly correct, executed algorithm correctly but highlighted incorrect final answer.

**+ 18 pts** Mostly correct, minor mistake in algorithm execution.

**+ 15 pts** Mostly correct, mistake in algorithm execution.

**+ 8 pts** Major error in execution of stable matching.

**+ 8 pts** Described the stable match algorithm but did not find final solution of matches for the given input.

**+ 4 pts** Attempted with some correct ideas.

**+ 4 pts** Had med schools applying to students.

**+ 0 pts** Not attempted or incorrect.

**+ 0 pts** made-up midterm: Incorrect

Name(last, first): Chen, Joyce

# UCLA Computer Science Department

**CS 180** **Algorithms & Complexity** UID : 405935837

**Midterm** **Total Time: 90 minutes** **November 9, 2023**

Each problem has 20 points: 5 problems, 5 pages (upload ONE pdf that has at most 2 pages per problem to gradescope and then hand in your exam to me).

**For all 5 problems: algorithms should be described in bullet point format (<u>with justification/proof</u>). You need to prove the correctness of your algorithm. You need to analyze its time complexity with proof.**

**Problem 1: a.** Describe Kruskal's MST (Minimum Spanning Tree) algorithm on a given undirected graph G=(V,E) with distinct weights. **b.**Prove that it produces an MST. **c.**Prove that the height of each Union-Find trees of size k is at most O(logk).

a) - sort the edges in G by increasing weight
- while not all nodes are in the same set:
  - do a "find" on the 2 nodes the edge connects $- O(\log V)$
  - if 2 nodes are in different sets or both unvisited yet:
    - do a "union" on the 2 nodes & add the edge to MST $- O(1)$
  - if 2 nodes are in the same set:
    - a cycle will exist if we add the edge
    - don't add edge to MST

total time: $O(E \log V)$

b) - The MST theorem states that all MSTs will contain e-min edges.
- Prove that Kruskal's will always add e-min:
  - Assume we are trying to add a new edge, where node $w \in S$ and the rest of the nodes $V-w \notin S$.
  - If we haven't added an edge yet, it's because adding such an edge will connect w and some node $\in S$ which produces a cycle, which is invalid for a tree.
  - Because we are going through edges by increasing weight, we will encounter the minimum edge e-min that connects w and some node $\notin S$.
  - The algorithm works because G must have a path between any 2 vertices, and we add e-min each time.

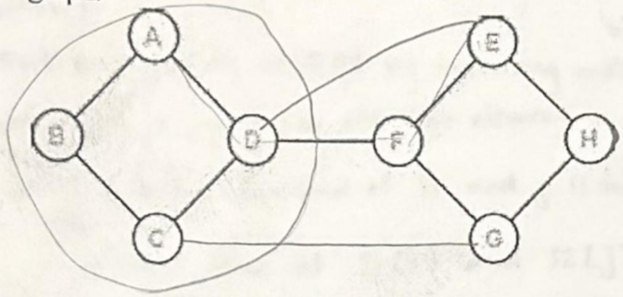
c) - Height of each union find tree of size k is not at most $O(\log k)$.
- Each time we do a union, we connect the smaller tree to the larger tree by pointing the smaller tree's root to the larger tree's.
- If a tree starts off with height k, it will either be added to a bigger tree — height still remains roughly $O(\log k)$, or a smaller tree is added to it — height is still $O(\log k)$. ><

*1* **17 / 20**

- **- 0 pts** Correct
- **- 8 pts** (a) incorrect
- **- 6 pts** (b) incorrect
- **- 6 pts** (c) incorrect
- **- 4 pts** (a) Missing details
- **- 3 pts** (b) Missing details
- ✓ **- 3 pts** *(c) Missing Details*
- **- 4 pts** made-up midterm part 1: Assuming there is only one vertex with in-degree=0.
- **- 6 pts** made-up midterm part 3: time complexity incorrect

Name(last, first): Chen, Joyce

**Problem 2:** A vertex that does not disconnect a connected undirected graph G=(V,E) is called a non-articulation-vertex (NAV). a.Describe a linear time algorithm for finding an NAV or decide that none exists. b. Prove the correctness of your algorithm. In the example below removing A or H does not disconnect the graph so they are both NAV. Removing D or F will disconnect the graph.




Example of Connected Graph

a) - we can do a partitioning on the graph. ~~and check whether a node on one side contains only edges to nodes of the same side, or multiple edges to the other side.~~

- select an arbitrary node v and place in set $S_1$. place all other nodes in set $S_2$.
- check whether there are multiple edges leading out of v connecting to a node in $S_2$.
  - if so, v is a NAV.
  - if not, add all nodes v is connected to in $S_2$ into $S_1$.

while we haven't found NAV:
  - check whether there are multiple edges from $S_1$ to $S_2$
  - if so:
    - any node in $S_1$ is a NAV.
  - if not (only 1 edge between $S_1$ and $S_2$).
    - any node that's not the node connecting $S_1$ and $S_2$ and has multiple edges is a NAV.
  - add nodes connected to in $S_2$ to $S_1$.

b) - Suppose our algorithm doesn't find a valid NAV.
- then, it finds some node that disconnects the graph G. removing such a node will remove the edge that connects the component the node is in and all other vertices.
- inside our algorithm we partition our graph such that nodes $v_1 \dots v_k$ are in $S_1$ and $v_{k+1} \dots v_n$ are in $S_2$.
- we declare that if only one edge is between $S_1$ and $S_2$, the node in $S_1$ on that edge is not NAV and "all others" are NAV, if they have multiple edges.
  - "all others" will still contain a path to nodes within S1 if they each have multiple edges. deleting one won't disconnect G ><.
  - they cannot disconnect $S_1$ and $S_2$ because they don't have an edge between them. ><

- our algorithm finds a NAV.

*2* **14 / 20**

- **+ 0 pts** Completely incorrect
- **+ 3 pts** Click here to replace this description.
- **+ 7 pts** small progress
- **+ 11 pts** Wrong, working for some cases
- ✓ **+ 14 pts** *Moderate error*
- **+ 16 pts** Great progress towards correct algorithm
- **+ 18 pts** Minor error
- **+ 20 pts** Correct
- **+ 8 pts** made-up midterm part A: correct
- **+ 3 pts** made-up midterm part C: Final answer correct.

sum=9

1 2 3 4 5 1 2 3 4 5 10

Name(last, first): Chen, Joyce

**Problem 3.** Given two sets S1 and S2 of real numbers each of size O(n) and a real number x, find whether there is a number from S1 and another number from S2 whose sum is x. An $O(n^2)$ algorithm is trivial.

A S1 B S2 $A+B=x$

Algorithm:

- sort each set S1 and S2 by increasing order.
- Maintain a pointer i to the first element in S1 and pointer j to the first element in S2.
- while i is not past end of S1 and j is not past end of S2:
  - check sum of S1[i] and S2[j], call M. if M=X, return true.
  - if M > x:
    - check sum of S1[i-1] and S2[j], if equal x, return true.
    - check sum of S1[i] and S2[j-1], if equal x return true.
    - otherwise return false.
  - if M < x:
    - increment both i and j
- If i is at end of S1 and current M < X:
  - check sum of S1[i] and every remaining element in S2
  - if we encounter sum = x, return true.
  - otherwise, return false.
- if j is at end of S2 and M < x:
  - check sum of S2[j] and every remaining element in S1
  - if sum=x, return true.
  - otherwise false.
- if M ≠ x, return false

Time Complexity

- sorting: O(NlogN)
- we are going through each set one time, and comparing the #'s we point to ⇒ O(2N) at worst ⇒ O(N).
- Total time: O(NlogN). O(N).

Proof of Correctness:

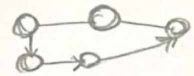
- Suppose we don't find 2 numbers that sum to x, but there are 2 such that exist.
- Then, the 2 nums $x_1 + x_2$ are either:
  - $x_1 + x_2 > x$:
    - since $x_1 + x_2$ is the min. sum so far in our algorithm, we back track and check whether $x_1$ + # before $x_2 = x$ or $x_2$ + # before $x_1 = x$.
    - if both aren't true, there is no sum $= x$ because we already checked # before $x_1$ + # before $x_2$.
    - would've returned false. ><
  - $x_1 + x_2 < x$:
    - only case is if we reach the end of algo and $x_1 + x_2 < x$.
    - however, we do a final check for whether cursum is ≠ x after checking for M<X.
    - returns false ><.

*3* **18 / 20**

- **+ 0 pts** Totally wrong
- **+ 9 pts** Major error
- **+ 14 pts** Moderate error
- ✓ **+ 18 pts** *Minor error*
- **+ 20 pts** Complete

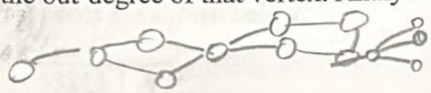
Name(last, first): Chen, Joyce

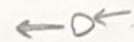
each vertex connect to
even # of vertices.

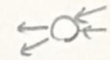


**Problem 4:**

Consider an undirected connected graph G=(V,E) where each vertex has an even degree. Design a linear-time algorithm that directs every edge (assigns a direction to each edge) such that the in-degree of every vertex is equal to the out-degree of that vertex. Analyze its time complexity (a proof is not needed).







Algorithm:

- Go through each edge $e \in E$ of G.
- Keep a map of vertex → its indegree and outdegree.
- for each edge $e_i$ that we encounter, that's not yet visited:
  - consider the 2 nodes v and w that it connects to.
  - if $\text{indegree}_v < \text{outdegree}_v$:
    - direct $e_i$ to v.
  - if $\text{indegree}_v > \text{outdegree}_v$:
    - direct $e_i$ to w.

Time complexity:

- Traversing each edge takes $O(E)$.
- Checking the indegree / outdegrees of each node takes $O(1)$ due to hashmap.
- Total: $O(E)$.

*4* **6 / 20**

**+ 20 pts** Correct.

**+ 18 pts** Correct, missing minor details or has minor errors.

**+ 9 pts** Correct direction, missing critical details or has major errors.

These points are for incomplete or incorrect solutions that at least recognized that a DFS like traversal was required for the problem.

**+ 5 pts** Designed algorithm that may work but is not linear-time.

✓ **+ 3 pts** *Incorrect solution/algorithm does not work on all valid graphs.*

✓ **+ 3 pts** *Correct runtime analysis for incomplete or wrong algorithm given.*

**+ 2 pts** Attempted.

**+ 0 pts** Not attempted or wrong.

**+ 15 pts** made-up midterm: Not clear on describing algorithm.

Name(last, first): Chen, Joyce

**Problem 5:**

Show each step of the stable matching algorithm and the final solution assuming five students (1,2,3,4,5) and five medical schools (A,B,C,D,E). You do not need a proof or a run-time analysis. Preference of the students and med schools are shown below:

| | | | | |
|---|---|---|---|---|
| 1 | CBEAD | | 35214 | A |
| 2 | ABECD | | 52143 | B |
| 3 | DCBAE | | 43512 | C |
| 4 | ACDBE | | 12345 | D |
| 5 | ABDEC | | 23415 | E |

Matching medical schools to students:

1 student 1 → C (top of 1's list)

2 student 2 → A

3 student 3 → D

4 student 4 → C (C prefers 4 to 1, A prefers 2 to 4).
student 1 becomes unmatched.

5 student 5 → A (A prefers 5 to 2)
2 becomes unmatched.

6 1 → B (next on 1's list, not taken).

7 2 → B (B prefers 2 to 1).
1 is unmatched.

8 1 → E (next on list that's unproposed to).

Final Matching:

1 → E
2 → B
3 → D
4 → C
5 → A

5 **20 / 20**

✓ **+ 20 pts** *Correct.*

**+ 18 pts** Correct, did not show each step sufficiently.

**+ 18 pts** Mostly correct, executed algorithm correctly but highlighted incorrect final answer.

**+ 18 pts** Mostly correct, minor mistake in algorithm execution.

**+ 15 pts** Mostly correct, mistake in algorithm execution.

**+ 8 pts** Major error in execution of stable matching.

**+ 8 pts** Described the stable match algorithm but did not find final solution of matches for the given input.

**+ 4 pts** Attempted with some correct ideas.

**+ 4 pts** Had med schools applying to students.

**+ 0 pts** Not attempted or incorrect.

**+ 0 pts** made-up midterm: Incorrect