CS570 Fall 2022: Analysis of Algorithms Exam I

	Points		Points
Problem 1	16	Problem 5	20
Problem 2	12	Problem 6	22
Problem 3	8	Problem 7	10
Problem 4	12		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Open book and notes. No electronic devices or internet access.
2. If a description to an algorithm or a proof is required, please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure, so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.
8. This exam is printed double sided. Check and use the back of each page.

1) 16 pts

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]

There exists an instance of the Stable Matching problem in which two men have the same best valid partner.

[TRUE/FALSE]

Prim's algorithm is not guaranteed to return a correct solution for graphs with negative weights.

[TRUE/FALSE]

If $T(n) = 4T(n/2) + 8\,n^2$, then $T(n) = O(n^3)$

[TRUE/FALSE]

For a weighted connected undirected graph *G* with positive weights, if the edge *e* is not part of any MST of *G*, then *e* must be the unique maximum weight edge of some cycle in *G*.

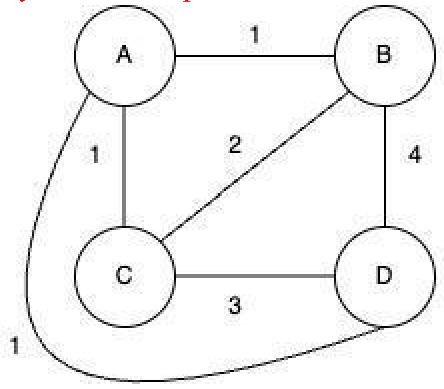
[TRUE/FALSE]

If a binomial heap consists of the 3 binomial trees B_0, B_1, and B_3, then after 4 Extract_Min operations, the binomial heap will consist of the following 3 trees: B_0, B_1, and B_2

[TRUE/FALSE]

For any cycle in a weighted connected undirected graph *G* with positive weights, if the cycle has a unique least-weight edge, then that edge is in some minimum spanning tree of *G*.

- In the graph below, the edges in an MST are AB, AC and AD. Therefore no edge in cycle BCD is present in the MST.



[TRUE/FALSE]
If Algorithm *A* has a worst-case running time of $\Theta(n^3)$ and algorithm *B* has a worst-case running time of $\Theta(n^2)$, then Algorithm *B* always runs faster than algorithm *A* on the same input.

[TRUE/FALSE]

In every undirected graph, there exists at least one path between every pair of vertices.

2) 12 pts

Circle ALL correct answers (no partial credit when missing some of the correct answers). No need to provide any justification.

i- Which of the following contradicts the statement, “The worst-case running time of the algorithm is $\Omega(n^2)$”? (3 pts)

(a) The algorithm runs in $O(1)$ steps on some types of input.
(b) For no input does the algorithm run in $O(n)$ steps.
(c) The worst case running time is $O(n \log n)$.
(d) The worst case running time is $O(2^n)$.
(e) The worst case running time is $\Omega(n^3)$.

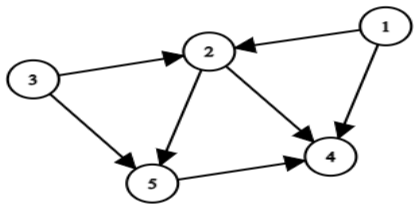
ii- Consider a binary max heap represented as an array [10, 9, 6, 8, 7, 4, 1, 2, 3]. We perform an Extract_Max followed by Decrease_Key(9,5) [meaning that the element with key value 9 will now have a key value of 5] on this heap. Which of these represents the new state of the heap? (3 pts)

(a) [8, 6, 7, 5, 4, 3, 1, 2]
(b) [8, 7, 6, 3, 5, 4, 1, 2]
(c) [8, 6, 5, 7, 4, 1, 2, 3]
(d) [8, 7, 6, 3, 5, 4, 2, 1]

iii- Which of the following algorithms solve the Minimum Spanning Tree problem? (3 pts)

(a) Kruskal's Algorithm
(b) Dijkstra's Algorithm
(c) Strassen's Algorithm
(d) Prim's Algorithm

iv- Which of the following is/are valid topological sorting(s) for the given DAG? (3 pts)



1) 1,3,2,5,4
2) 1,3,5,2,4
3) 3,1,2,5,4
4) 3,1,5,2,4
5) 4,5,2,3,1

3) 8 pts

For the given recurrence equations, solve for *T(n)* if it can be found using the Master Method (make sure to show which case applies and why). Else, indicate that the Master Method does not apply and explain why.

i) $T(n) = 8T(n/2) + n\log n - 1000n$

ii) $T(n) = 2T(n/2) + n^3(\log n)^3$

iii) $T(n) = 4T(n/2) + n^2(\log n)^2$

iv) $T(n) = 4T(n/2) - n^4(\log n)^4$

Solutions:

1. Case 1 of the Master Method $T(n) = \theta(n^3)$
2. Case 3 of the Master Method $T(n) = \theta(n^3(logn)^3)$. Condition $2f(n/2) = 2(n/2)^3(log(n/2))^3 <= cn^3(logn)^3$ for some c < 1, holds true for c = ¼
3. Case 2 General of the Master Method $T(n) = \theta(n^2(logn)^3)$
4. Master theorem cannot be applied since function $f(n)$ is not asymptotically positive

Rubric:
For each part of this question:
– 1pt for correct Case selection
– 1pt for correct final answer

4) (12 pts) A student wants to insert n elements into an empty binary heap. The student also wants to backup this heap after every fixed number of insertions. Unfortunately, the backup operation is quite costly: each backup operation takes $\Theta(n)$ time (no matter how many elements are currently in the heap).

a) What is the amortized cost of the insertion operation if backups are performed after every n/10 insertions? (6 pts)

b) What is the amortized cost of the insertion operation if backups are performed after every 10 insertions? (6 pts)

Solution:
a) Aggregate method:
Total time to insert all n elements is O(nlogn) + O(10n) = O(nlogn)
Amortized: O(nlogn)/n = O(logn).

b) Aggregate method:
Total time to insert all n elements is O(nlogn) + O(n/10*n) = O(n^2)
Amortized: O(n^2)/n = O(n).

Basic (3 pts):

- For Both part a and b:
  - -2 pts for wrong answer
  - -1 pt for incorrect binary-heap insertion run time (should be O(logn))

Reasoning (3 pts):

- For Both part a and b:
  - -3 pts if the reasoning is completely wrong
  - Aggregate method or accounting method are both acceptable
    - -1 pt for incorrect insertion run time or number of insertion steps
    - -2 pts for incorrect number of backup steps or backup cost/credit

If no attempt or no sufficient reasoning:

- For Both part a and b:
  - -2 pts for wrong answer
  - -4 pts for no sufficient reasoning

5) 20 pts

The transportation network of bus, train, and airplane routes in California can be represented as a weighted connected undirected graph $G(V,E)$ with positive weights, where each vertex $v \in V$ represents a city in California, each edge $e \in E$ represents a transportation route between two cities, and each edge weight $w(e)$ represents the length of time needed to travel via the transportation route e. Each edge $e \in E$ is either a bus route, train route, or airplane route. Denote the set of bus routes as $E_B \subseteq E$, the set of train routes as $E_T \subseteq E$, and the set of airplane routes as $E_A \subseteq E$. Note that there may be more than one transportation route type connecting two cities. For example, there may be a train route and a bus route between the same two cities.

a) Design an efficient algorithm to compute the length of time of the quickest route from a given city s to a given city t that never travels via the same mode of transportation twice consecutively. For example, if we take a train from city i to city j, then we can't take a train out of city j. (17 pts)

b) Analyze the worst-case time complexity of your algorithm. (3 pts)

- Solution I - modifying Dijkstra's directly

For each node v, keep three distance values: $d_B(v)$, $d_A(v)$, $d_T(v)$, each denoting the length of the shortest path from s to v with the last edge being transportation mode B, V or T respectively.

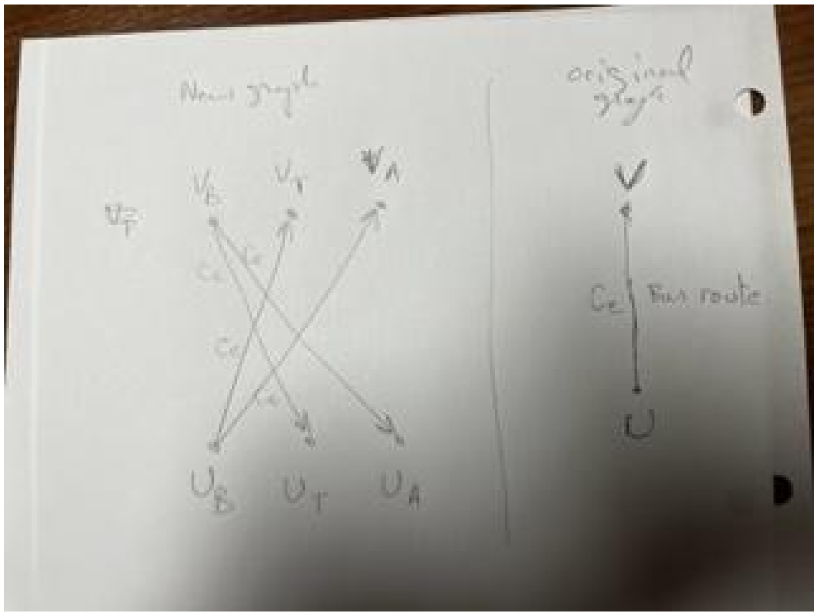
Modifications to the Dijkstra's algorithm:

- Initialize the priority queue Q by adding all these d_B, d_A and d_T values (∞ for all nodes except 0 for node s).
- For the extract_min step, say we extracted the minimum value $d_B(u)$, then we should still keep the two other values $d_A(u)$ and $d_T(u)$ in Q, and won't add u to the visited set yet.
- For the relaxation step, say we extracted the minimum value $d_B(u)$, then for each neighbor v of u:
    - Decrease_key $d_A(v)$ to $d_B(u)+w_A(uv)$ (if it becomes less)
    - Decrease_key $d_T(v)$ to $d_B(u)+w_T(uv)$ (if it becomes less)

  Similarly for other cases.
- After Q becomes empty, we return the min of $d_B(t)$, $d_A(t)$ and $d_T(t)$.

- Solution II - converting the graph

For each vertex v create three vertices v_B, v_T, v_A.
For each undirected edge $(u,v) \in E_B$ (similarly for E_A and E_T), create 4 directed edges (u_B, v_T), (u_B, v_A), (v_B, u_T), (v_B, u_A) with same edge weight, as shown in graph below.
(Alternative: have all edges in the other direction)



Create a dummy starting node with edge weight 0 connecting to s_B, s_T, s_A.
Create a dummy ending node with edge weight 0 connecting from t_B, t_T, t_A.
(Intuition: traveling from node u_B to v_A corresponds to traveling from city u to city v by bus, while leaving v_A by airplane.)

For both solutions, the runtime is dominated by running the Dijkstra's Algorithm, which is $\Theta((|E| + |V|)log|V|)$ / $\Theta(|E|log|V|)$ (using binary heap) or $\Theta(|E| + |V|log|V|)$ (using fibonacci heap).

Rubric:

a) 17 points in total

Mention Dijkstra's for finding shortest paths (+5)

- If they modify Dijkstra's and run it on the original graph:
  - A reference solution:
    - Solution I
  - Rubric:
    - Case I: Attempt to modify Dijkstra's to incorporate the constraint (+2)
    - Case II: Nearly correct with some small mistake (+10)
      - Needs to consider the three optimal routes to each node to get the points.
    - Case III: Fully correct (+12)
- If they modify the graph and run the standard Dijiskstra's:
  - A reference solution:
    - Solution II
  - Rubric:
    - Case I: Attempt to convert the problem: Create three nodes for each city (+2)
    - Case II: Nearly correct with some small mistake (+10)
      - Failure to consider undirectedness of G (e.g. only creating 2 directed edges)
      - Failure to consider directedness of newly created graph
      - Failure to add dummy starting/ending nodes
      - Missing details in description such as edge weights
    - Case III: Fully correct: correct edges between corresponding nodes and correct details (+12)

b) 3 points in total

- If the solution is based on Dijkstra's:
  - Correct analysis/description of the stated complexity (+1)
    - If the student modifies the graph or constructs a new graph, the cost needs to be discussed to get the point.
  - Running Dijkstra's
    - with $O(V^2)$ (+1)
    - with correct runtime (+2)
- If the solution is not based on Dijkstra's:
  - The time complexity is correct for the proposed solution (+2)

6) 22 pts
Assume there are n TAs for a graduate-level CS course. The TA availability on Mondays is provided in the form of two arrays S[1..n] and E[1..n]. For example, for the first TA in the list, S[1] = 8 and E[1]=13 indicates that this TA will be available from 8 AM to 1 PM.

a) Design an (announced clarification: efficient) algorithm that returns the minimum number of TAs required so that there is at least one TA available from 8 AM to 8 PM on Mondays. (12 pts)

b) Analyze the worst-case time complexity of your algorithm. (3 pts)

c) Prove that your algorithm is correct. (7 pts)

SOLUTION

a) We will construct a set of TAs whose availabilities cover the time interval 8AM-8PM as follows. As we construct our set, let t denote the latest time covered by a TA in our set. Initially, when our set is empty, t = 8AM.

Sort the TAs by their availability start times S. Iteratively add the TA with the latest availability end time E whose availability start time S is less than or equal to t.

At the end of each iteration, if $t \geq$ 8PM, return the size of our set. If we have iterated over all TAs and still $t <$ 8PM, then there is no set of TAs whose availabilities cover the time interval 8AM-8PM, in which case our algorithm (should return ∞? should return -1? should return None or null? may return anything since the behavior of our algorithm is undefined in this case?).

b) The TAs can be sorted by their availability start times S in time $O(n \log n)$.

In each iteration, we add the TA with the latest availability end time E whose availability start time S is less than or equal to t. In doing so, we iterate over each TA at most once over the course of our algorithm. Therefore, the total cost of these steps is $O(n)$.

The total time complexity therefore is $O(n + n \log n) = O(n \log n)$.

c) Let O be the set of TAs returned by our algorithm. Suppose there exists a smaller optimal set of TAs O^* that covers the time interval 8AM-8PM. Denote by S_i^* the availability start time of the TA with the ith earliest availability start time in O^*, and E_i^* that TA's availability end time. Denote by S_i the availability start time of the TA with the ith earliest availability start time in the set O returned by our algorithm, and E_i that TA's availability end time.

We must have $S_1^* \leq$ 8AM, since otherwise O^* would not cover the time interval starting at 8AM and ending at S_1^*. By construction, $S_1 \leq t =$ 8AM. Since E_1 is the latest availability end time of all TAs whose availability start time is less than or equal to $t =$ 8AM, we must have $E_1^* \leq E_1$.

Suppose that $E_i^* \leq E_i$ for all $1 \leq i \leq k$, for some integer $k \geq 1$. We must have $S_{k+1}^* \leq E_k^*$, since otherwise O^* would not cover the time interval starting at E_k^* and ending at S_{k+1}^*. Putting these inequalities together gives $S_{k+1}^* \leq E_k^* \leq E_k$. By the design of our algorithm, E_{k+1} is the latest availability end time among TAs whose availability start time is less than or equal to E_k. Since $S_{k+1}^* \leq E_k$, we then must have $E_{k+1}^* \leq E_{k+1}$.

Therefore, $E_i^* \leq E_i$ for all $i \geq 1$.

Note that the availability end times in O^* must be non-increasing. I.e., for all $i \geq 1$, we must have $E_i^* \leq E_{i+1}^*$, since otherwise $S_i^* \leq S_{i+1}^* \leq E_{i+1}^* < E_i^*$, in which case TA i+1 can be removed from O^* while still covering the time interval 8AM-8PM, contradicting our assumption that O^* is optimal.

Note that the availability end times in O must also be non-increasing. I.e., for all $i \geq 1$, we must have $E_i \leq E_{i+1}$, since otherwise there does not exist a set of TAs that covers the time interval 8AM-8PM, contradicting our assumption that O^* is a valid solution.

Let denote $m = |O|$ and $m^* = |O^*|$. Since our algorithm terminates when $E_i \geq$ 8PM, we must have $E_{m\text{-}1} <$ 8PM $\leq E_m$. Since O^* is smaller than O, we have $m^* \leq m$-1. Therefore $E_{m^*}^* \leq E_{m^*} \leq E_{m\text{-}1} <$ 8PM. Since $E_{m^*}^*$ is the last and therefore latest availability end time in O^*, we must have that O^* does not cover the time interval 8AM-8PM, contradicting our assumption that O^* was a valid solution.

Therefore, there does not exist an optimal smaller set of TAs O^* that covers the time interval 8AM-8PM. Therefore, our algorithm is optimal.

RUBRIC

Q6a (12 points)

Q6a.i (5)

(+0) Q6a.i: The student did not sort the TAs in any order.

(+2) Q6a.i: The student sorted the TAs in some order, but not the correct order.
Example: “Sort the intervals (S_i, E_i) in increasing order.”

(+5) Q6a.i: The student sorted the TAs by availability start time.

Q6a.ii (5)
(+0) Q6a.ii: The student did not give a condition for iteratively adding TAs to their constructed set.
(+2) Q6a.ii: The student constructed a set of TAs by iteratively adding the TAs to their constructed set, but did so incorrectly.

Examples:

- They started to iterate through the whole set of remaining TAs while adding a new TA to the constructed set.
- In each iteration, they selected the TA with min start time instead of setting a threshold for TAs that have start time less than previous selected TA end time

(+5) Q6a.ii: The student constructed a set of TAs by iteratively adding the TA with the latest availability end time whose availability start time is less than or equal to *t*.
Q6a.iii (2)
(+0) Q6a.iii: The student did not attempt to handle the case where no valid set of TAs exists.
(+2) Q6a.iii: The student attempted to handle the case where no valid set of TAs exists.

Q6b (3 points)
Q6b.i (1)
(+0) Q6b.i: The student did not mention that the time complexity of sorting TAs is $O(n \log n)$, or the student already received 0 pts for Q6a.i.
(+1) Q6b.i: The student mentioned that the time complexity of sorting TAs is $O(n \log n)$.
Q6b.ii (2)
(+0) Q6b.ii: The student did not mention that the total time complexity of iterating over TAs is $O(n)$, or already received 0 pts for Q6a.ii.
(+1) Q6b.ii: The student mentioned that the total time complexity of iterating over TAs is $O(n)$, but did not give sufficient reasoning.

Example: "Using pointers, we can iterate through the list in O(n) time."

(+2) Q6b.ii: The student mentioned that the total time complexity of iterating over TAs is $O(n)$, and reasoned that this is because each TA is iterated over exactly once over the course of our algorithm.

Q6c (7 points)
Q6c.i (1)
(+0) Q6c.i: The student did not begin their proof by contradiction by assuming there exists some other smaller set of TAs that covers 8AM-8PM than the one returned by the algorithm.
(+1) Q6c.i: The student began their proof by contradiction by assuming there exists some other smaller set of TAs that covers 8AM-8PM than the one returned by the algorithm.

Q6c.ii (2)
(+0) Q6c.ii: The student did not state at least one base case for their inductive proof.
(+1) Q6c.ii: The student attempted to prove at least one base case for their inductive proof, but did not give sufficient reasoning.

(+2) Q6c.ii: The student correctly proved at least one base case for their inductive proof.

Q6c.iii (2)

(+0) Q6c.iii: The student did not state their inductive hypothesis.
(+1) Q6c.iii: The student incorrectly stated their inductive hypothesis.
Example: "Assume $E_i^* \leq E_i$."
(+2) Q6c.iii: The student correctly stated their inductive hypothesis.

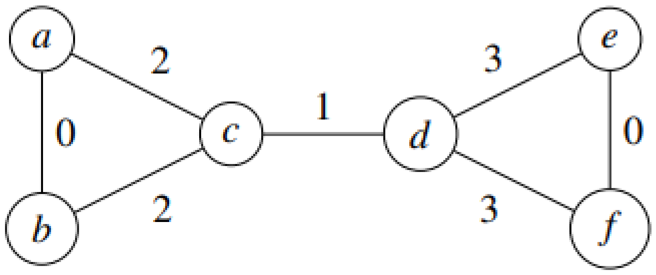
Q6c.iv (2)

(+0) Q6c.iv: Inductive step not attempted.
(+1) Q6c.iv: Inductive step attempted, but not fully correct.
(+2) Q6c.iv: Correct inductive step.

7) 10 pts
Prove or disprove the following statement:

For a weighted connected undirected graph G(V,E) *with positive weights, if for all edges* e ∈ E *there exists at most one other edge* e' ∈ E *with the same weight, then* G *has at most two distinct minimum spanning trees.*

False.
To prove, give any counterexample.
E.g., There are 4 MST's in the following graph



One MST has edges ac and ed
…ac and df
…bc and ed
…bc and df

Rubric:
(+0 points): Wrong answer (i.e. trying to prove the statement rather than disprove)
(+4 points): Correct answer but wrong counter-example
(+10 points): correct ans + counter-example

Additional Space

Additional Space

Additional Space