Exam 1 - Rubric

Q12-21: True/False

12. Finding the *k*-th minimum element in an array of size *n* using a binary min-heap takes O(*k log n*) time. **[False]** Building heap is O(n) + removing K log n

MErging is nlogn and dividing is constant

13. We can merge any two arrays each of size *n* into a new sorted array in O(*n*). **[False]**

DAG topological sorting

14. The shortest path in a weighted directed acyclic graph can be found in linear time. **[True]**

15. Given a weighted planar graph. Prim's algorithm using a binary heap implementation will outperform Prim's algorithm using an array implementation. **[True]**

Explanation:

In Planar graph, E <= 3*V - 6. Prim's algorithm using array has the complexity of O(V^2 + E) = O(V^2). Prim's algorithm using binary heap has the complexity of O(V*logV + E*logV) = O(V*logV). Therefore, Prim's algorithm using array will run slower than Prim's algorithm using binary heap.

16. If f(n) = Ω (n log n) and g(n) = O(n^2 log n), then f(n) = O(g(n)). **[False]** Not strictly

17. If we have a dynamic programming algorithm with n^2 subproblems, it is possible that the space usage could be O(*n*). **[True]** E.g knapsack, we may require only the previous row

18. There are no known polynomial-time algorithms to solve the fractional knapsack problem. **[False]**

19. In a knapsack problem, if one adds another item, one must solve the whole problem again in order to find the new optimal solution. **[False]**

20. Given a dense undirected weighted graph, the time for Prim's algorithm using a Fibonacci heap is O(E). **[True]** Read properly, how many times deleteMin is called, which is O(V) but here it is dense so O(E)

21. In a binomial min-heap with *n* elements, the worst-case runtime complexity of finding the second smallest element is O(1). **[False]** k log N , which is 2Logn

Q22-31: Multiple Choice

22. The solution to the recurrence relation T(n) = 8 T(n/4) + O(n1.5 log n) by the Master theorem is

$O(n^{1.5} \log^2 n)$

23. There are four alternatives for solving a problem of size n by diving it into subproblems of a smaller size. Assuming that the problem can be divided into subproblems in constant time, which of the following alternatives has the smallest runtime complexity?

we solve 5 subproblems of size n/3, with the combing cost of Θ (n log n)

24. Which of the following statements is true?

If an operation takes O(n) expected time, then it may take O(1) amortized time.

25. Which of the following properties does not hold for binomial heaps?

FINDMIN takes O(log n) worst-case time

26. There are n people who want to get into a movie theater. The theater charges a toll for entrance. The toll policy is the following: the i-th person is charged k2 when i = 0 mod k (this means that i is a multiple of k), or zero otherwise. What is the amortized cost per person for entering the theater? Assume that n > k.

$\Theta(k)$

27. What is the runtime complexity of the following code:

```
void exam1(int n) {
  int count = 0;
  for (int i=n/2; i<=n; i++)
    for (int k=1; k<=n; k = 2 * k)
      for (int j=1; j<=n; j = j * 2)
        count ++;
}
```

$\Theta(n \log^2 n)$

28. Given an order k binomial tree Bk, deleting its root yields

***k* binomial trees**

29. Kruskal's algorithm cannot be applied on

directed and weighted graphs

30. Consider a recurrence $T(n) = T(a\ n) + T(b\ n) + n$, where $0 < a \leq b < 1$ and $a + b = 1$. Which of the following statements is true?

$T(n) = \Theta(n \log n)$

31. Consider an undirected connected graph G with all distinct weights and five vertices {A, B, C, D, E} and. Suppose CD is the minimum weight edge and AB is the maximum weight edge. Which of the following statements is false?

No minimum spanning tree contains edge AB

Long Questions

Q1. Amortized Cost

Consider a singly linked list data structure that stores a sequence of items in any order. To access the item in the i-th position requires i time units. Also, any two contiguous items can be swapped in constant time. The goal is to allow access to a sequence of n items in a minimal amount of time. The TRANSPOSE is a heuristic that improves accessing time if the same is accessed again. TRANSPOSE works in the following way: after accessing x, if x is not at the front of the list, swap it with its next neighbor toward the front. What is the amortized cost of a single access?

Solutions:

Aggregate method

Suppose the singly linked list is:

$$head \to a\_1 \to a\_2 \to \cdots \to a\_n$$

Considering the worst case of accessing these n items, we start access from a_n to a_1.

We first access to a_n which takes n cost. Then we swap it with a_(n-1) the next step, we want to access to the a_(n-1), again it takes n cost. Accessing a_n and a_(n-1) takes 2*n cost.

Then, we access the a_(n-2) which takes n-2 cost. Then we swap it with a_(n-2). In the next step, we want to access to the a_(n-3), again it takes n-2 cost. Accessing a_(n-2) and a_(n-3) takes 2*(n-2) cost.

Repeatedly do that, we will get the following summation of cost:

2*2 + 2*4 + 2*6 + ... + 2*(n-2) + 2*n + n(cost of swap) = 4*(1+2+3+...+n/2) + n = n(n+2)/2 + n = O(n^2)

Therefore, the amortized cost per access is O(n).

Rubrics:

Please read the textbook about the definition of amortized cost: Amortized analysis gives the average performance of each operation in the worst case.

[12 pts]

- [5 pts] State the worst case scenario is to access from a_n to a_1.We cannot perform swap along
    - The order of accessing a sequence of n elements is not the worst case: -1 pt
    - accessing order is not specified: -2 pts
    - only considering accessing one element: -2 pts
- [6 pts] Analyze and state the summation of total cost which is O(n^2)
    - Wrong explanation or no explanation: - 6pt
        - Explaining using i (this is not the worst case) -4 pt
    - Correct explanations:
        - Swap can not be performed once after each access. It swaps (i-1)th element with i th element. incorrect statement about it will get -1 pt
        - lack of sufficient explanation, for example, how to calculate the total cost: -2 pts
        - The cost of transpose is not considered: -2 pts
            - Transpose actually degrade the performance in the worst case, any statement that transpose increase the performance: - 1 pt
        - the total cost is incorrect: -2 pts
- [1 pts] Analyze and state the amortized cost per access which is O(n).
    - Answer is not O(n): -1 pt

Q2. Shortest Path Problem

Consider a network of computers represented by a directed weighted graph where each vertex is a computer, and each edge is a wire between two computers. An edge weight w(*u*, *v*) represents the probability that a network packet (unit of binary data) going from computer *u* reaches computer *v*. Our goal is to find a network path from a given source *s* to a target computer *t* such that a packet has the highest probability of reaching its destination. In other words, we are looking for a path where the product of probabilities is maximized.

Solutions:

Solution 1:
Use a modification of Dijkstra to find the maximum product of edge weights.
Use a set of unexplored edges like in Dijkstra (this may be a max-heap instead of a min-heap or a priority queue).
Initialize the source node with a 0 and all other nodes with a negative infinity.
For every iteration:

```
    Pick the node x with highest value and add it to a final_path list
    For each of the neighbors n of x:
        If node_value(n) < node_value(x)*weight(x,n):
            node_value(n) = node_value(x)*weight(x,n)
```

Keep iterating and if the picked node happens to be the target node, then we have reached the solution and we return the final_path list (can be a string, set etc as well)

Solution 2:

Take a log of all the edge weights and find the maximum length path using Dijkstra modification where we initialize the same as we did in solution 1 above (i.e with negative infinity and 0 for source node. Then we find the longest path using a modification when picking up the node as above (pick the max node value using a max heap or priority queue). The rest goes, the same as Dijkstra.

Rubric:

[+2 Marks] Using correct initialization of nodes
[+6 Marks] Correct Algorithm Description with correct node updation equation used
[+2 Marks] Returning the entire path and not just the max probability

[-1 Marks] Partially Correct Initialization

[-3 Marks] Incorrect node_value updation
[-1 Marks] Did not mention returning the entire path

Q3-7. Greedy

In this problem you are to find the best order in which to solve your exam questions. Suppose that there are *n* questions with points $p_1, p_2, \ldots, p_n$ and you need time t_k to completely answer the *k*-th question. You are confident that all your completely answered questions will be correct (and get you full credit for them), and you will get a partial credit for an incompletely answered question, in proportion of time spent on that question. Assuming that the total exam duration is *T*, design a greedy algorithm to decide on the optimal order. Solve the problem in the following five steps.

Question 3 (6 points) Describe your greedy algorithm

This is similar to the fractional knapsack problem. The greedy algorithm is as follows. Calculate $\frac{p_i}{t_i}$for each $1 \leq i \leq n$ and sort the questions in descending order of $\frac{p_i}{t_i}$. Let the sorted order of questions denoted by $s(1), s(2), \ldots, s(n)$.Answer questions in this order until $s(j)$such that $\sum_{k=1}^{j} t_{s(k)} \leq T$ and $\sum_{k=1}^{j+1} t_{s(k)} > T$. If $\sum_{k=1}^{j} t_{s(k)} = T$, then stop, otherwise partially solve the questions $s(j+1)$in the time remaining.

Rubric:

- If the greedy criteria is wrong, 0 point.
- If only stated solving the remaining problems with the highest ratio: 2 points.
- Sorting in Descending order: 4 points (keyword must occur. Not keyword, no points) Other descriptions are fine: largest to smallest, non-increasing, etc.
- Sort without order or wrong order: -2 points
- It's OK not to mention the corner case.
- Some students use the word "order", 'rank' 'arrange', 'choose' instead of "sort", which is also OK as long as they mention the order.
- Some students don't mention order after sorting, but mention in some other places such as the algorithm loop. Give full points for this case.
- Some students do the inverse version. Compute $$\frac{t_i}{p_i}$$ and sort with ascending/increasing order. Get full points for this case.
- Incorrect notations: -2 points.

Question 4 (3 points) Compute the runtime complexity of your algorithm.

Calculate the $\frac{p_i}{t_i}$for each questions: O(n)

Sorting questions in order of $\frac{p_i}{t_i}$: O(nlogn)

Processing questions: O(n)
Overall: O(2n+nlogn)=O(nlogn)

Rubric:

- considering sorting the pi/ti ratios: 1 point
- considering at least one of calculating pi/ti and processing questions: 1 point
- Final result: 1 point (must be explicitly mentioned)
- If only the final result is given and it is correct without explanation: 3 points.

Question 5 (5 points) State and prove the induction base case (use a proof by contradiction here)
Let P(i) denote an optimal solution where question s(i) is part of the solution. To show that $P(1)$ is true, we proceed by contradiction. Assume $P(1)$ to be false. Then $s(1)$ is not part of any optimal solution. Let $a \neq s(1)$ be a partially solved question in some optimal solution O' (if O' does not contain any partially solved questions then we take a to be any arbitrary question in O'). Taking time $min(t_{a,} T, t_{s(1)})$ away from question a and devoting to question $s(1)$ gives the improvement in points equal to $(\frac{p_{s(1)}}{t_{s(1)}} - \frac{p_a}{t_a}) min(t_{a,} T, t_{s(1)}) \geq 0$ since $\frac{p_i}{t_i}$ is the largest for $i = s(1)$. If the improvement is strictly positive, then it contradicts optimality of O' and implies the truth of $P(1)$. If improvement is 0, then a can be switched out for $s(1)$ without affecting optimality and thus implying that $s(1)$ is part of an optimal solution which in turn means that $P(1)$ is true.

A different way to state the base case: if the optimal solution contains problems p_{i,1}, p_{i, 2}...p_{i, k} and the greedy solution contains problems p_{j, 1}, p_{j, 2},... p_{j, m}. Assume they are all sorted by p/t in the descending order, then we show that p_{i,1}=p_{j,1}, meaning that the problem with the highest p/t ratio must be included in the optimal solution.

Rubric:

- State the base case: 3 points.
- Prove its optimality: 2 points. The idea is that there always exists another problem that can be swapped with s(1) if s(1) is not a part of the optimal solution to improve the total credits.

Common mistakes:

- It is incorrect to perform induction on the time because time is a continuous value. (0 points)
- It is also incorrect to perform induction on the total number of questions. The induction should be performed on the index of questions answered in the p/t order, not the total.
- It is incorrect to prove by showing that after completing the same number of questions, the points earned by greedy must be greater than the points earned by the optimal solution. (0 points)

- It is incorrect to argue that the "efficiency" (defined as p/t) of the greedy solution always stays ahead of the "optimal" solution. This is equivalent to using your approach to prove the optimality of your approach.
- It is incorrect to argue that the first l problems of the greedy and the optimal solution match and prove for the "l+1" problem. The only thing that we can say that there "exists" such an optimal solution that matches the greedy choice.
- When proving the inductive step, many students use "replacement" to show that the credits will increase when replacing the "new" problem with any problem that has less p/t. This is incorrect because when you do "replacement", due to the time constraints, you may replace more than one problem.

Question 6 (3 points) State the inductive hypothesis
The induction hypothesis $P(l)$ is that there exists an optimal solution that agrees with the selection of the first l questions by the greedy algorithm.

Rubric:

- Must state that our algorithm leads to optimality for the first l questions.
- 2 points deduction if not clear that our algorithm is optimal (e.g., there is only talk of an optimal algorithm).
- Hypothesis on time T is incorrect. (0 points)

Question 7 (7 points) State and prove the inductive step
Let $P(1),\ P(2),\ \ldots,\ P(l)$ be true for some arbitrary l, i.e., the set of questions $\{s(1),\ s(2),\ \ldots,\ s(l)\ \}$ are part of an optimal solution O. It is clear that Oshould also be optimal with respect to the remaining questions $\{1,\ 2,\ \ldots,\ n\} \setminus \{s(1),\ s(2),\ \ldots,\ s(l)\}$ in the remaining time $T - \sum_{k=1}^{l} t_{s(k)}$. Since $P(1)$is true, there exists an optimal solution, not necessarily distinct from O, that selects the question with the highest value of $\frac{p_i}{t_i}$in the remaining set of questions. i.e. the question $s(l + 1)$ is selected. Since $s(l + 1)$ is also the choice of the proposed greedy algorithm, $P(l + 1)$ is proved to be true.

Rubric:

- No partial credit, but equations can be paraphrased in different ways.

Q8-11. Dynamic Programming

Columbus wants to travel the Venice river as fast as possible. There are cities 1, 2, ..., n located by the river. There are some boats connecting two cities with different speeds. For each i < j, there is a boat from city i to city j which takes T[i, j] minutes to travel. Starting from city 1, help Columbus to calculate the shortest time possible to reach city n. Your proposed algorithm should have time complexity O(n^2). For example, if n = 4 and the set of boats are { T[1, 2] = 3, T[1, 3] = 2, T[1, 4] = 7, T[2, 3] = 1, T[2, 4] = 1, T[3, 4] = 3 }, then your algorithm should return 4 minutes (for the route 1 -> 2 -> 4).
Design a dynamic programming algorithm for solving this problem. Please complete the following steps.

1. Define (in plain English) the subproblems to be solved.
2. Write the recurrence relation for the subproblems. Also write the base cases.
3. Use plain English to explain how you use that recursive relation to obtain a solution. You don't need to prove the correctness.
4. State the runtime of the algorithm. Explain your answer

Solution

1. Defining sub-problem If Columbus reaches city i, then the next move is to choose the city $j > i$ to jump to after city i. So, let's define $d[i]$ as the shortest time possible to pass from city i to city n.

2. Recurrence Relation The recurrence relation is the minimum total time among all the possible selection of city $i < j$ to jump to. The base case is $d[n] = 0$.

$$d[n] = 0$$
$$d[i] = min(\{T_{i,j} + d[j] | i < j \leq n\}), 1 \leq i < n$$

3. Explanation: Based on the definition of the sub-problem, the value of $d[i]$ tells us what would be the minimum time (fastest route) to reach city n starting from city i. The base case is $d[n] = 0$, since if Columbus is at city n already, they he does not need to ride any boat and the minimum time is 0. Now assume we know the best answer for any $i + 1 \leq j \leq n$, and we want to find the best answer for i. If Columbus is located at city i, he can move to any city j after city i. If he jumps to city $j > i$, since we already know the best answer for city j and it is $d[j]$, so the total time of travel would be $T_{i,j} + d[j]$. Now, for city i for should select the next city j such that the total travel time would be minimum. That is why we have $d[i] = min(\{T_{i,j} + d[j] | i < j \leq n\}), 1 \leq i < n$.

4. Time Complexity There are n number of unique sub-problems and the update rule for each sub-problem takes $O(n)$ time. So, the total time complexity is $O(n^2)$.

```
d[n] = 0
for i=n-1 to 1:
    best_jump = t[i][n]
    for j=i+1 to n-1:
        best_jump = min(best_jump, t[i][j] + d[j])
    d[i] = best_jump
return d[1]
```

Rubric (20 points):

- Part 1 (6 points): Clear definition of subproblems
  - If they defined a 2D table for dp like OPT[i, j] -3 points (since they cannot achieve O(n^2) using this).
  - If they explain the idea but did not define OPT -4 points
- Part 2 (9 points): Correct recurrence formula (7 points) and correct base case (2 points).
- Part 3 (5 points): Explaining the reason for the recurrence relation.
- Part 4 (4 points): Correct runtime complexity.
  - -4 points if they just say time complexity is O(n^2). This is part of the question not the solution!
  - -2 points if they only mention the number of sub-problem or the time it takes to update.
  - -2 points if they use wrong reasoning of why O(n^2) (while their algorithm is O(n^3))
  - -1 point if explanation is incorrect or not enough
  - -2 point if runtime is not O(n^2)
- If they have a O(n^3) algorithm and every part of the solution is correct, they can get 15/20 points (-3 points for part 1 and -2 points for part 4).
- If they use the following idea and it was correct no reduction of points:
  - Define D[i] as the minimum time to pass from city 1 to i. And then the recurrence formula would be:

$D[1] = 0$

$D[i] = \min\{T_{j,i} + D[j] \mid 1 <= j < i \}$