CS570 Spring2024: Analysis of Algorithms Exam II

	Points		Points
Problem 1	20	Problem 5	18
Problem 2	9	Problem 6	20
Problem 3	16	Problem 7	17
	Total	**100**	

Instructions:
1. This is a 2-hr exam. Closed book. No electronic devices or internet access. One 8.5x11 cheat sheet allowed.
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) 20 pts

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

[TRUE/[✓ FALSE]]

In a flow network, the maximum flow is unique if all edge capacities are unique AND the min cut is unique.

[TRUE/[✓ FALSE]]

In a dynamic programming solution, the value of the optimal solution should always be saved for all unique sub problems.

[TRUE/[✓ FALSE]]

The dynamic programming algorithm for the Knapsack Problem runs in weakly polynomial time in the size of the input.

[[✓ TRUE]/FALSE]

Given a flow f in a flow network G, if there exists a directed path from the source node s to the sink node t in the residual graph G_f, then f is not a max-flow in G.

[TRUE/[✓ FALSE]]

Let (*A*, *B*) be a minimum *s-t* cut in a flow network. If we strictly increase the capacity of every edge that crosses the (A,B) cut, then the value of a maximum flow of the network must strictly increase.

[TRUE/[✓ FALSE]]

During the execution of the Ford-Fulkerson algorithm, if all augmenting paths are simple (acyclic), then the resulting max flow will have no flow cycles.

[TRUE/[✓ FALSE]]

If an iteration of the Ford-Fulkerson algorithm on a flow network assigns a flow of 1 to an edge (*u*, *v*), then after every subsequent iteration, the flow assigned to (*u*, *v*) must have a value greater than or equal to 1.

[[✓ TRUE]/FALSE]

It is possible for a correctly implemented dynamic programming algorithm to have an exponential running time with respect to the input size.

[TRUE/[✓ FALSE]]

Any dynamic programming solution with n^2 unique subproblems will run in $O(n^2)$ time.

[[✓ TRUE]/FALSE]

It is possible for a circulation network to not have a feasible circulation even if all edges have unlimited capacities.

2) 9 pts (3 pts each)
Circle ALL correct answers (no partial credit when missing some of the correct answers). No need to provide any justification.

i- Given a flow network *G* with *n* nodes and *m* edges, with a maximum flow of value *k*, which of the following statements are true for the running time of some flow algorithm:

a) A running time of $O(mnk)$ is weakly polynomial

b) A running time of $O(m \log k)$ is pseudo-polynomial

c) A running time of $O(n m^2)$ is strongly polynomial

d) A running time of $O(k^2)$ is pseudo-polynomial

ii- Which of the following holds for the scaled version of the Ford-Fulkerson algorithm? Assume *C* is the maximum capacity of any edge outgoing from the source, and *m* is the number of edges in the network.

a) The number of Δ-scaling phases is $O(\log C)$

b) The number of Δ-scaling phases is $O(C)$

c) The number of augmentations in a Δ-scaling phase is at most *m*

d) The number of augmentations in a Δ-scaling phase is at most 2*m*

iii- Bellman-Ford algorithm can be used to find.

a) Flow cycles in a flow network

b) Flow cycles in a circulation network

c) Negative cycles in a directed graph

d) Shortest paths in a directed graph

3) 16 pts

The Miso Good food truck produces many different lunch menu items. Suppose that, on a given day, Miso Good has produced m types of food items $b_1, \ldots, b_m$, and the quantity of each type of food item b_j is exactly q_j. Unfortunately, due to the limited quantities, they often run out of popular items, making customers sad. As a solution, Miso Good is implementing a sophisticated lunch-ordering system as follows. Suppose there are n customers $a_1, \ldots, a_n$. The customers are required to text in their acceptable choices for food items before lunchtime - each customer a_i submits a set A_i of one or more acceptable food items. Then, Miso Good will try to assign ONE food item to each customer, and the customers who cannot be assigned a food item from their acceptable set, will be given a $10 voucher as compensation for their inconvenience. Miso Good would like to minimize the number of these vouchers they give out.

a- Given the input as described above, design an efficient network flow based algorithm for Miso Good to output the optimal assignment of lunches to customers (and vouchers as necessary). (10 pts)

Construct a flow network G with source node s, sink node t, n nodes for customers, and m nodes for food items. Add directed edges from the source s to the customers each with capacity 1, edges from each customer to their preferred food item each with capacity 1, and edges from food items to the sink t with capacity q_j. Find a max-flow in the flow network with Scaled Ford-Fulkerson. For each edge from customers to food items, if the flow assigned to that edge is 1, give that customer a food item of that type. If for some customer, there is no out-going edge with assigned flow 1, give that customer a voucher.

Rubric:

7pts for flow network construction:

- 2pt (Good effort): The student's network at least includes all the following basic elements: a source node (or something that looks like a source node), nodes for customers, nodes for food items, a sink node (or something that looks like a sink node), and edge capacities for each edge.
- 2pt (Completeness): Specified completely and clearly in words, rather than only drawing a picture, their construction of the edges between customers and food items, e.g., for each customer and each food item, if that customer prefers that food item, then there's a directed edge from that customer to that food item.
- 3pt (Correctness): The student's construction is completely correct.

3pt for algorithm description:

- 1p (Completeness): The student completely and clearly describes how to assign customers to food items or vouchers after running the flow algorithm of their choice on their network.
- 2pt (Correctness): The student's description of how to assign customers to food items or vouchers after running the flow algorithm of their choice on their network yields a correct algorithm.

See next page for part b

b- Prove the correctness of your algorithm. (6 pts)

Claim: For all integers k, such that $n \geq k \geq 0$, there exists an integer-valued flow f in G with value n - k iff there exists a way to assign food items or vouchers to customers that assigns k customers to food vouchers.

Forward proof: For some integer k, such that $n \geq k \geq 0$, if there exists an integer-valued flow f in G with value n - k, then exactly n - k of the out-edges of the source is assigned a flow value of 1, and the other k out-edges of the source are assigned flow 0. For each of the k customers at the heads of the source out-edges assigned flow 0, by conservation of flow, none of the edges from that customer to the food items has assigned flow 1, so the customer will be assigned a voucher. Therefore, exactly k customers will be assigned a voucher. For each of the n - k customers at the heads of the source out-edges assigned flow 1, by conservation of flow, exactly one of the out-edges from that customer to the food items has assigned flow 1, and the others 0. By assigning customers to food items as in the algorithm, the customer will be assigned the food item corresponding to the edge with assigned flow 1. For each food item, the net in-flow is equal to the number of customers assigned to that food item. By conservation of flow, this is bounded by the capacity q_j of the out-edge from the food item to the sink. Therefore, the number of customers assigned to food item j is $\leq q_j$.

Backward proof: For some integer k, such that $n \geq k \geq 0$, if there exists a way to assign food items or vouchers to customers that assign k customers to food vouchers, then we can construct an integer-valued flow in G with value n - k as follows: for each edge from source to customer, if the customer was assigned a food item, assign a flow of value 1 to that edge, and 0 otherwise. For each edge from customer to food item, if the customer was assigned that food item, assign a flow of value 1 to that edge and 0 otherwise. For each edge from food item to sink, assign a flow of value equal to the number of customers assigned to that food item to that edge. This flow has value n - k, satisfies the conservation of flow at each node, and satisfies the edge capacities.

The number of vouchers given to the customers is equal to n minus the value of the flow. Since the value of the flow is maximized, the number of vouchers given to customers is minimized.

Rubric:
3pts for Forward Proof:

- 1pt (Premise): Begins by assuming that there exists a flow in their network with some specified property, e.g., the flow is a max-flow, or the flow has value n - k.

- 1pt (Completeness): Given such a flow, the student completely describes how to construct a corresponding assignment of customers to food items or vouchers.
- 1pt (Correctness): Provides some argument for why their constructed assignment of customers to food items or vouchers assigns total *k* vouchers to customers or why their constructed assignment of customers to food items or vouchers is optimal. To earn points for this part, the student must have also gotten the +1pt for completeness for their forward proof.

3pts for Backward Proof:

- 1pt (Premise): Begins by assuming that there exists an assignment of customers to food items or vouchers with a specified property, e.g., the assignment is optimal or assigns total *k* vouchers to customers.
- 1pt (Completeness): Given such an assignment of customers to food items or vouchers, the student completely describes how to construct a corresponding flow in their flow network.
- 1pt (Correctness): Provides some argument for why their constructed flow has value *n* - *k*, or why their constructed flow is a max-flow. To earn points for this part, the student must have also gotten the +1pt for completeness for their backward proof.

4) 20 pts

Given an array A of length n containing positive integers $a_1, \ldots, a_n$ and positive integers m and k such that $mk \leq n$, Your goal is to choose k mutually disjoint subarrays of length m from array A to maximize the total sum of elements within these subarrays. Formally, let the start and end indices of the k intervals of length m be $[L_1, R_1]$, $[L_2, R_2]$, …, $[L_k, R_k]$ such that $1 \leq L_1 < R_1 < L_2 < R_2 < \ldots < L_k < R_k \leq n$. The objective is to maximize the value of $\sum_{1 \leq i \leq k} \sum_{L\_i \leq j \leq R\_i} a_j$.

a) Define in plain English the subproblems that your dynamic programming algorithm solves. (6 pts)
b) Define a recurrence relation that expresses the optimal solution (or its value) of each subproblem in terms of the optimal solutions to smaller subproblems. (6 pts)
c) Using the recurrence formula in part b, write pseudocode using iteration to find the maximum value. Make sure you specify base cases and their values. (6 pts)
d) What is the time complexity of your solution? (2 pts)

Brief solution to the problem

a)
OPT(i, j) is the maximum sum when choosing i mutually disjoint subarrays for the first j elements of A

Rubric:
- -2 pts: the subproblem definition is not clear and well-defined
- -4 pts: the subproblem definition is clearly wrong (e.g., the variables does not appear in the subproblems definition)
- -6 pts: No/minimal attempt

b)
OPT(i, j) = max(OPT(i, j - 1), OPT(i - 1, j - m) + sum(a[j - m + 1 … j]))

Rubric:
- -2 pts: the recurrence relation contains some minor error according to the student's definition in a)
- -4 pts: the recurrence relation is clear wrong according to the student's definition in a)
- -6 pts: No/minimal attempt

c)
Initialize OPT(i, j) = 0 for all i = 0, 0 <= j <= n and for all 1 <= i <= k, 0 <= j < i * m
Initialize the prefix sum array of size (n + 1) with prefix_sum[0] = 0

```
for i = 1 to n:
    prefix_sum[i] = prefix_sum[i - 1] + $a_i$
for i = 1 to k:
    for j = (i * m) to n:
        OPT(i, j) = max(OPT(i, j - 1), OPT(i - 1, j - m) + prefix_sum[j] - prefix_sum[j - m])
return OPT(k, n)
```

Rubric:
2 pts for base cases:

- -1 pt: minor error
- -2 pts: No attempt/incorrect basecases

2 pts for the iterative pseudocode:

- -1 pt: minor error
- -2 pts: No attempt/incorrect iterative process (for/while loop is wrong)

2 pts for the return value:

- -2 pts: No/incorrect return value to this problem

d)
O(nk)

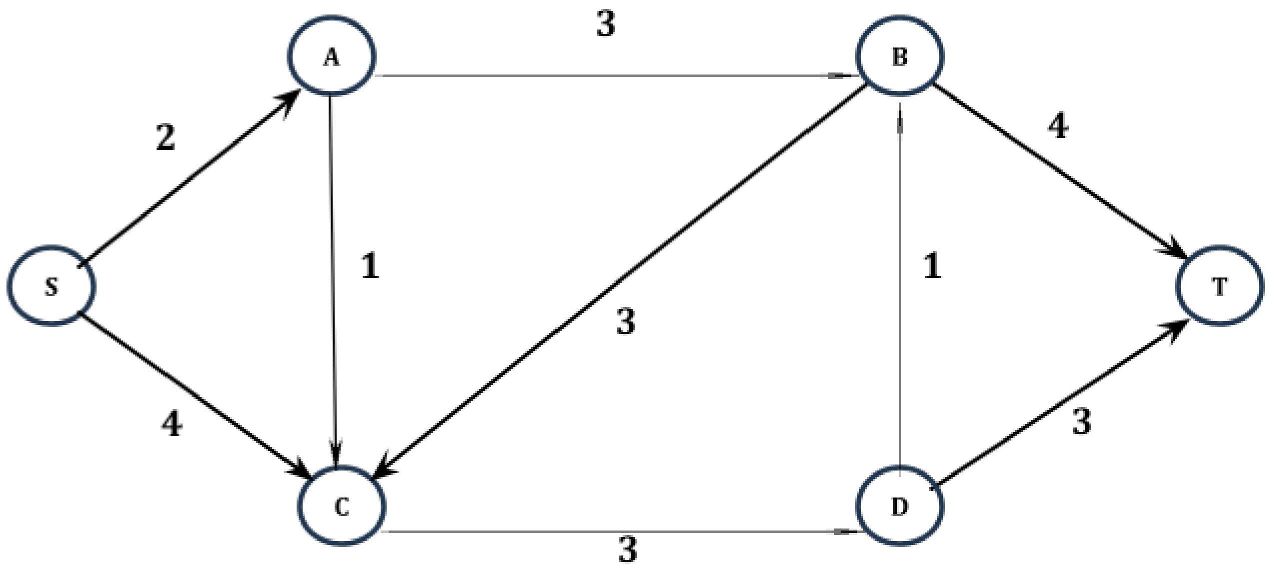
Rubric:

- -2 pts: No attempt/Incorrect according to the student's pseudocode in c)

Shawn: I think the initialization should be
Initialize OPT(i, j) = 0 for row i=0 and all columns $0 <= j <=n$
            OPT(i,j)=0 for row i>0 and columns $0 <= j <=i*m$ on that row
Please double check

5) 20 pts

In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.



a: Draw the residual networks after each of these two augmentation steps (using the above two augmenting paths in the order given). (6 pst)

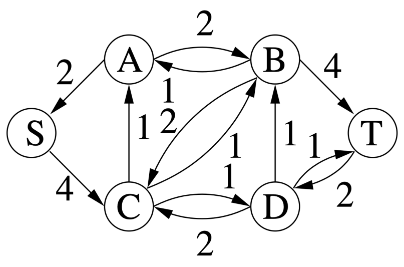
b: List all of the augmenting paths that could be chosen for the third augmentation step. (6 pts)

c: What is the value of the maximum flow? (4 pts)

Draw a dotted line through the original graph to represent the minimum cut. (4 pts)

- Answer:

a: Graph after 2nd Aug path (for brevity)



The graph after the first one is expected as well in the question.

Rubric: 3 points for each graph. Only 1.5 penalty for the first one missing if the 2nd one is completely correct.

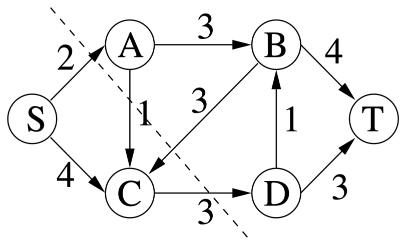
b: S-C-A-B-T, S-C-B-T, S-C-D-B-T, and S-C-D-T. Note that Ford-Fulkerson does not (need to) find non-simple augmentation paths

Rubric: -1.5 for each missing/incorrect path (also applies to incorrect paths listed in addition to the 4 correct ones)

c: 5.

Rubric: 4 points for correct value, 0 otherwise.

d:
See below figure for the minimum cut.



Rubric: 4 points for correct cut. 2.5 if cut described in a way other than shown above and the description is incomplete (e.g., listed only in terms of edges crossing the cut)

6) 17 pts

Recall the matrix chain multiplication problem: Given the dimensions of a sequence of *n* matrices $A_1 \ldots A_n$ in an array *arr*[0..*n*], where the number of rows in the matrix A_i is *arr*[*i*-1] and the number of columns *arr*[*i*], the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimized.

a) Given the notation OPT(i,j) denoting the minimum number of element multiplications to complete the product of matrices $A_i \ldots A_j$, write the recurrence formula to find the minimum number of element multiplications required. (you need to use the *arr* array to refer to the number of rows and columns) 5 pts

Remember: if A is an m by n matrix and B is n by k, the number of element multiplications in A * B is mnk.

OPT(i,j) = Min (for i <=k < j) {OPT(i, k) + OPT(k+1,j) + arr[i-1]*arr[k]*arr[j] }

Rubric:

5 pts for the recurrence formula:

- -1 pt: a minor mistake in the formula (e.g., putting arr[i-1]*..*arr[j] outside the bracket)
- -3 pts: multiple mistakes in the formula (e.g., missing the iterator k, only considers the cases k = i or j-1)
- -5 pts: incorrect answer (e.g., using more dimensions instead of OPT(i,j), such as OPT(i,j,k)) or incomplete answer

b) Given the array *arr*[0..4] = [4, 2, 3, 1, 3] representing the sizes of matrices $A_1 \ldots A_4$, perform a bottom up pass to find the minimum number of element multiplications. In other words, draw the two dimensional OPT array and fill in the terms in the array by numerically solving the bottom up pass. No code or pseudocode is necessary. (6 pts)

26	12	9	0
14	6	0	
24	0		
0			

Here i and j grow along the conventional positive x and positive y direction respectively.

Rubric:

6 pts for the table

- -1 pt: one incorrect/missing **non-zero** value in the table
- -3 pts: multiple incorrect/missing (more than one) **non-zero** values in the table
- -6 pts: incorrect answer (more than four non-zero values are incorrect)

c) Recall that the objective is to find the most efficient way to multiply these matrices together. So using the values of the optimal solutions from part b, perform the top down

pass to find the order in which the matrices should be multiplied. Again, no code or pseudocode is necessary.
Note: You need to numerically solve the top down pass and at each step show how the optimal multiplication order is revealed at that step.
(6 pts)

According to the problem statement, we will use the array and computed values in Part(b) to construct the best order to multiply the matrices.

Considering the sequence of matrices A1 * A2 * A3 * A4. In the first step of the top down pass, we are deciding whether to multiply

-	(A1 * A2 * A3) * A4	14+12, or
-	A1 * (A2 * A3 * A4)	12+24, or
-	(A1 * A2) * (A3 * A4)	9+24+36

The first step therefore indicates (A1 * A2 * A3) * A4 to be the best option

In the second step we are deciding whether to multiply

-	(A1 * A2) * A3, or	24 + 12, or
-	A1 * (A2 * A3)	6+8

The second step therefore indicates A1 * (A2 * A3) to be the best option

So the optimal order of operations will be **(A1 * (A2 * A3)) * A4**.

Rubric:
6 pts for the explanation and the correct optimal multiplication order:

- -3 pts: incomplete/mistake in the explanation and the optimal multiplication order (provided order is correct, the explanation is partially correct)
- -4 pts: no explanation (provided order is correct, explanation and OPT values are incorrect)
- -6 pts: incorrect answer (wrong order with wrong explanation/OPT values) or no multiplication order is provided

Additional Space

Additional Space

Additional Space