Name: Rollno:

CS345: Design and Analysis of Algorithms (Final Exam)

17th November 2024

Total Number of Pages: 10 Time: 3 hr Total Points 100

Instructions

1. All questions are compulsory.
2. Answer all the questions in the space provided in the question paper booklet.
3. Use the space provided in the paper for rough work.
4. The symbols or notations mean as usual unless stated.
5. You may cite and use algorithms and their complexity as done in the class.
6. Cheating or resorting to any unfair means will be severely penalized.
7. Superfluous and irrelevant writing will result in negative marking.
8. Using pens (blue/black ink) and not pencils. Do not use red pens. for answering.

Helpful hints

1. It is advisable to solve a problem first before writing down the solution.
2. The questions are *not* arranged according to the increasing order of difficulty.

Question	Points	Score
1	10	
2	8	
3	12	
4	8	
5	10	
6	6	
7	9	
8	9	
9	14	
10	14	
Total:	100	

Name: Rollno:

Question 1. Short answer type questions.

(a) (1 point) The number of nodes at depth i in the Binomial tree of degree k, B_k is

(a) $\binom{k}{i}$

.

(b) (2 points) The maximum depth of a tree in a Fibonacci heap is O(n).

(c) (2 points) Suppose we have a set A consisting of n distinct elements. If we split A into $\log n$ many blocks of equal size, compute the median of each block, and then compute the median of all these median, say x. If the rank of x in A is r, then r is at least $n/4$ and at most $3n/4$.

(Assume n is a power of 2, and ignore additive constants.)

(d) (2 points) If A and B are two NP-complete problems, then does $A \leq_p B$? Give reason for your answer.

> **Solution:** Yes. Since A is NP-complete it is in NP. And since B is NP-complete, $\forall C \in$ NP, $C \leq_p B$. Hence $A \leq_p B$.

(e) (1 point) Suppose P $\neq$ NP. Does every NP-complete problem have a constant factor approximation algorithm? (Yes/No) **No**

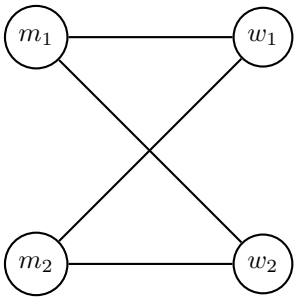
(f) (1 point) If a directed, weighted graph is allowed to have negative weight cycles, then what is the complexity of computing a shortest path in such a graph?

(f) **NP-complete**

(g) (1 point) If a man m has a woman w at the top of his preference list, and w has m at the top of her preference list, then in every stable marriage, is m married to w? (Yes/No) **Yes**

Question 2. This question is based on the stable marriage problem.

(a) (2 points) Consider the following 4 vertex complete bipartite graph.



Give an example of a set of preference lists for the four vertices, such that there are two stable matchings.

$$L(m_1) = (w_1, w_2)$$
$$L(m_2) = (w_2, w_1)$$
$$L(w_1) = (m_2, m_1)$$
$$L(w_2) = (m_1, m_2)$$

(b) (2 points) Given a set M of n men, and a set W of n women, and their respective preference lists L, design an algorithm that checks whether there is a unique stable marriage or not.

Solution: Run Gale-Shapley algorithm on (M, W) to get a stable marriage, say S_1. Again run Gale-Shapley algorithm on (W, M) to get a stable marriage, say S_2. If $S_1 = S_2$, then there is a unique stable marriage, else not.

(c) (4 points) Prove the correctness of your algorithm in Part (b).

Solution: Clearly if there is a unique stable marriage then $S_1 = S_2$.
Suppose $S_1 = S_2 = S$(say). Then S is both man optimal as well as woman optimal. For the sake of contradiction let there be some other stable marriage $T \neq S$. Hence there is a married pair (m, w) in S, who are married to different partners in T. Suppose m is married to w', and w is married to m' in T.
Since S is man optimal, m must prefer w to w'. Similarly since S is woman optimal, w must prefer m to m'. Hence (m, w) is an unstable pair in T. Hence T is not a stable matching.

Question 3. In this problem we will give a bound on the maximum degree of a node in a Fibonacci heap H of size n.

(a) (3 points) Let x be a node in H having degree k. Let $y_1, y_2, \ldots, y_k$ be the children of x, in the order they were linked to x (y_1 was earliest and y_k was latest). Then show that for all $i \geq 2$, $\deg(y_i) \geq i - 2$.

Solution: size$(y_1) \geq 1$. For $i \geq 2$, when y_i is made a child of x, all $y_1, y_2, \ldots, y_{i-1}$ are children of x, therefore $\deg(x) \geq i - 1$. Moreover when y_i is added to x, $\deg(x) = \deg(y_i)$. Hence $\deg(y_i) \geq i - 1$. Since then at most one child of y_i has been dropped, as dropping two children would remove y_i as a child of x. Therefore, $\deg(y_i) \geq i - 2$.

(b) (2 points) Let F_k be the k-th Fibonacci number. Then $F_0 = 0$, $F_1 = 1$ and for $k \geq 2$, $F_k = F_{k-1} + F_{k-2}$.
Prove that for $k \geq 0$, $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Solution: We prove by induction on k. For $k = 0$, the statement holds. Assume its true for $k - 1$. Then

$$\begin{aligned} 1 + \sum_{i=0}^{k} F_i &= 1 + \sum_{i=0}^{k-1} F_i + F_k \\ &= F_{(k-1)+2} + F_k \quad \text{(by induction hypothesis)} \\ &= F_{k+2} \end{aligned}$$

(c) (3 points) Consider the golden ratio ϕ, which is also the positive root of the quadratic equation $\phi^2 - \phi - 1 = 0$.
Prove that for $k \geq 0$, $F_{k+2} \geq \phi^k$.

Solution: We prove this by induction on k as well. For $k = 0$, the statement holds. Assume it is true for all $k \leq n-1$. Then

$$\begin{aligned} F_{n+2} &= F_{n+1} + F_n \\ &\geq \phi^{k-1} + \phi^{k-2} \quad \text{(by induction hypothesis)} \\ &= \phi^{k-2}(\phi + 1) \\ &= \phi^{k-2}\phi^2 \quad \text{(definition of } \phi) \\ &= \phi^k \end{aligned}$$

(d) (3 points) Let x be a node in H having degree k. Then prove that $\text{size}(x) \geq \phi^k$.

Solution: Let s_k be the minimum size of a subtree rooted at a node of degree k. Therefore $s_0 = 1$ and $s_1 = 2$. Now

$$\begin{aligned} \text{size}(x) &\geq s_k \\ &\geq 2 + \sum_{i=2}^{k} s_{\deg(y_i)} \\ &\geq 2 + \sum_{i=2}^{k} s_{i-2} \quad \text{(by Part (a))} \end{aligned}$$

Now we show by induction that $s_k \geq F_{k+2}$. Base case is easy to see. For inductive case,

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \quad \text{(by induction)} \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} \\ &\geq \phi^k \end{aligned}$$

(e) (1 point) Conclude that the maximum degree of a node in H is $O(\log n)$.

Solution: For any node x in H having degree k, by Part (d) we have $n \geq \text{size}(x) \geq \phi^k$. Therefore $k \leq \log_\phi n = O(\log n)$.

Question 4. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover every vertex in G.

(a) (5 points) Give a reduction from the cycle cover problem to the problem of computing a matching in a bipartite graph. In other words, given a directed graph $G = (V, E)$ output an undirected graph $H = (V_H, E_H)$ and a number t, such that G has a cycle cover if and only if H has a matching of size t.

Solution: Define $H = (V_H, E_H)$ as follows:

- For each vertex $u \in V$, add two vertices, u_A and u_B in V_H.
- For each directed edge $(u, v) \in E$, add the undirected edge , (u_A, v_B) in E_H.
- Set $t := |V|$.

(b) (1 point) What does the reduction in Part (a) tell us about the complexity of computing a cycle cover?

Solution: Since bipartite matching can be solved in polynomial time, hence cycle cover can also be solved in polynomial time.

(c) (2 points) Can we give a polynomial time reduction from vertex cover to cycle cover? What would it imply?

Solution: If vertex cover can be reduced to cycle cover in polynomial time, then by the previous part, there is a polytime algorithm for vertex cover. However vertex cover is known to be NP-complete. Hence this would imply $\mathsf{P} = \mathsf{NP}$, which is very unlikely.

Question 5. Given a network $G = (V, E)$ with source s and sink t, such that $s \neq t$, and a capacity function c that assigns a positive integral value to every edge. Also the max flow f itself is given, that is, for edge e, $f(e)$ is given. Lastly we are given a fixed edge $g = (u, v)$ in G.

(a) (5 points) If the capacity of g is decreased by 1, then give a linear time algorithm to compute the max flow in the updated network.

Solution: Given: Network $G = (V, E)$ with source s and sink t, integral capacity function c, max flow f, an edge $g = (u, v)$.

1. If $f(g) < c(g)$ then the flow remains the same. There is no change.
2. Otherwise, decrease the flow and capacity of the edge g by 1.
3. Compute a path P_1 from s to u and another path P_2 from v to t using BFS.
4. Decrease flow by 1 for all edges along P_1 and P_2, to get a new flow, say f'.
5. Compute residual network of G with respect to the flow f', say $G_{f'}$.
6. Run BFS from s to see if t is reachable.
7. If yes, then increase flow by 1 along this path to get a flow f'' and output f'' as maximum flow. Else output f' as maximum as flow.

(b) (5 points) Prove the correctness of your algorithm. Argue why the algorithm in Part (a) will run in linear time

Solution: To see that the algorithm runs in $O(n)$ note that steps 2–7 take $O(n)$ time each. Steps 1 and 2 take $O(1)$ time.

Proof of correctness: Note that if the capacity of one edge is decremented by one, then the max flow value either decreases by one or stays the same. Also if the edge g was not saturated by f, then $c(g) \geq f(g) + 1$ since the capacities are integral and hence f is integral. Therefore if g was not saturated, decrementing the capacity of g by one, does not change the max flow. If g was saturated, then there was at least one unit of flow passing through g, since $c(g) \geq 1$. Therefore there is some path from s to u and v to t in the network, through which at least one unit of flow is passing. Therefore decreasing flow by one unit along (P_1, g, P_2), gives a new valid flow f', which saturates the new capacity of g'. Now if f' is not maximum, then in the residual network $G_{f'}$, there must be some path from s to t. The residual capacity of this path must be at least one, since the capacities are integral. So incrementing the flow by one, along this path gives us a new max flow f''.

Question 6. (6 points) A string s is said to be a cyclic rotation of another string t, if they have the same length and s consists of a suffix of t followed by a prefix of t. For example *arc* and *car* are cyclic rotations of each other.

Design a linear time algorithm to check if a string s is a cyclic rotation of another string t.

Solution: Check if s and t are of the same length. If not, then reject. Apply KMP algorithm to check if t is a substring of ss. If yes, then accept, else reject.

Since KMP is linear time, and the size of ss is linear in the size of s, therefore our algorithm also runs in linear time.

If t is a cyclic rotation of s, then t begins with a suffix of s, followed by the remaining prefix of s. Hence t is a substring of ss. Now if t is a substring of ss, then either $t = s$, or t begins with a proper suffix of s, followed by the remaining prefix of s. In both cases t is a cyclic rotation of s.

Question 7. The BDST problem takes as input an undirected graph $G = (V, E)$ and a positive integer $k \leq |V| - 1$, and decides whether or not there is a spanning tree of G in which every vertex has degree at most k (i.e. each vertex has at most k edges of the spanning tree incident on it). In this question we will show that BDST is NP-complete.

(a) (2 points) Show that BDST $\in$ NP.

Solution: Certificate: A collection of edges, say s
Verifier: Given G, k and the certificate s. Let H_s be the subgraph of G, induced by the edges in s. Check if H_s is a spanning tree of G. If not then reject. Else check if the degree of every vertex in $H_s \leq k$. If yes, then accept, else reject.

(b) Choosing a known NP-complete problem: HamPath

(c) (4 points) Give the reduction.

Solution: Given G, output $\langle G, 2 \rangle$ as the instance of BDST.

(d) (1 point) What is time complexity of the reduction.

Solution: Trivially $O(n)$.

(e) (2 points) Prove the correctness of the reduction.

Solution: If G has a Hamiltonian path, then the path itself is a spanning tree and degree of every vertex in the path is at most 2. So $\langle G, 2\rangle$ is a positive instance of BDST.

On the other hand if G has a spanning tree such that every vertex in the spanning tree has degree at most 2, then the spanning tree must be a single path that spans all vertices. Therefore G has a Hamiltonian Path.

Question 8. There is a sequence of matrices $M_1, M_2, \ldots, M_n$ storing numbers. The number of operations required to multiply a $p \times q$ matrix with a $q \times r$ matrix is pqr (using the standard matrix multiplication algorithm). For each $1 < i \leq n$, the number of rows of M_i is identical to the number of columns of M_{i-1}. So the product $M_1 \cdot M_2 \ldots M_n$ is well defined. We also know that matrix multiplication is associative. That is, $(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)$. However, the number of arithmetic operations required may vary in these two possible ways. Our aim is to compute $M_1 \times M_2 \times \ldots \times M_n$ using least number of multiplication operations.

(a) (1 point) Give an example of three matrices M_1, M_2 and M_3, such that the number of operations to compute $(M_1 \cdot M_2) \cdot M_3$ is different from the number of operations to compute $M_1 \cdot (M_2 \cdot M_3)$.

Solution: Let M_1 be a 5×2 matrix, M_2 be a 2×4 matrix and M_3 be a 4×3 matrix.

No of operations required to compute $(M_1 \cdot M_2) \cdot M_3 = 5 \cdot 2 \cdot 4 + 5 \cdot 4 \cdot 3 = 100$. No of operations required to compute $M_1 \cdot (M_2 \cdot M_3) = 2 \cdot 4 \cdot 3 + 5 \cdot 2 \cdot 3 = 54$.

(b) (8 points) Design an algorithm based on dynamic programming to solve this problem in polynomial time.

Solution: Let the dimension of the matrix M_i be $d_i \times d_{i+1}$.

Let $C_{i,j}$ be the minimum number of operations required to multiply the sequence $M_i, M_{i+1}, \ldots, M_j$. Our target is to compute $C_{1,n}$, and we know for all i, $C_{i,i} = 0$. Now the recursive formulation of $C_{i,j}$ can be given as follows:

$$C_{i,j} \quad = \quad \min_{i \leq k \leq j-1} \{C_{i,k} + C_{k+1,j} + d_i d_{k+1} d_{j+1}\}$$

Therefore we need to fill the matrix diagonal-wise.

Algorithm 1: MatrixMultiplication$(d_1, d_2, \ldots, d_{n+1})$

```
for i = 1 to n do
    C_{i,i} := 0
end
for Δ = 1 to n − 1 do
    for i = 1 to n − Δ do
        Set j := i + Δ
        Set C_{i,j} := ∞
        for k = i to j − 1 do
            C_{i,j} := min{C_{i,j}, C_{i,k} + C_{k+1,j} + d_i d_{k+1} d_{j+1}}
        end
    end
end
return C_{1,n}
```

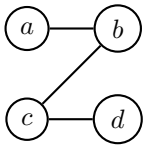
Time complexity of the algorithm is $O(n^3)$.

Question 9. Recall that in class we defined a matching in a graph as a subset of edges that do not share any common vertex. Given a bipartite graph we saw how to compute a maximum matching by reducing it to the problem of computing maximum flow in a network. However that does not give us a linear time algorithm. In this question we will design a 2-approximate linear time algorithm to compute a maximum matching.

A *maximal matching* is defined to be matching that is not a proper subset of any other matching.

(a) (3 points) Give an undirected graph G with 4 vertices where a maximal matching is not a maximum matching. In G, mention what is the maximal matching and what is the maximum matching.

Solution:



Maximum matching: $\{(a,b),(c,d)\}$. However $\{(b,c)\}$ is a maximal matching.

(b) (3 points) Given an undirected graph $G = (V, E)$, give an $O(|E|)$ time algorithm to compute a maximal matching in G.

Solution: We give a greedy strategy to compute a maximal matching. Let $e_1, e_2, \ldots, e_m$ be the edges of the graph G.

Algorithm 2: MaximalMatching$(e_1, e_2, \ldots, e_m)$

```
Set M := ∅
for i = 1 to m do
    if Both end points of e_i do not have any edge incident on them, having index < i
      then Add e_i to M
end
return M
```

(c) (4 points) Given an undirected graph G show that the size of a maximum matching in G is at most the size of any vertex cover of G.

Solution: Let M be a matching in G. Now consider any vertex cover of G, say C. For every edge in M, at least one of its endpoints must be in C, since C is a vertex cover. And since no two edges in M shares a common vertex, therefore $|M| \leq |C|$. Since M and C were arbitrary, therefore the size of the maximum matching is at most the size of the minimum vertex cover.

(d) (3 points) For every maximal matching M in $G = (V, E)$, show that there is vertex cover in G of size $2 \cdot |M|$.

Solution: Let M be a maximal matching. Consider the set of vertices,

$$S = \{u \in V \mid u \text{ is an endpoint of some edge in } M\}.$$

We claim that S is a vertex cover of G. Suppose not, then there is some edge $e \in G$, such that neither endpoint of e is in S. Hence by definition of S, e is not in M. If this is the case, then we can add e to M to get a matching of greater size. This contradicts that M is a maximal matching. Therefore S is a vertex cover.

(e) (1 point) Conclude that the algorithm in Part (b), is a 2-approximate algorithm for maximum matching.

Solution: The algorithm in Part (b) gives a maximal matching M. By Part (d), there exists a vertex cover C of size $2|M|$. By Part (c), $|C|$ is an upper bound on the size of the maximum matching. So $2|M|$ is an upper bound on the size of a maximum matching. Hence, the algorithm in Part (b) is a 2-approximation algorithm for maximum matching.

Question 10. (14 points) IIT Kanpur has built some new classrooms and wants to schedule the final exam of every course in a single classroom. Your task as an algorithm designer is to see if that is possible and if so give such a scheduling.

There are n different courses, each of which needs to schedule a final exam in one of r rooms during one of t different time slots. At most one course's final exam can be scheduled in each room during each time slot. Conversely, courses cannot be split into multiple rooms or multiple times. Moreover, each exam must be overseen by one of the p proctors. Each proctor can oversee at most one exam at a time, each proctor is available for only certain time slots and no proctor is allowed oversee more than 3 exams overall. The input to the scheduling problem consists of three arrays:

- An integer array $E[1..n]$ where $E[i]$ is the number of students enrolled in the i-th course.
- An integer array $S[1..r]$, where $S[j]$ is the number of seats in the j-th room. The i-th course's final exam can be held in the jth room if and only if $E[i] \leq S[j]$.
- A $t \times p$ boolean matrix A where $A[k, l] = 1$ if and only if the l-th proctor is available during the k-th time slot.

Design a polynomial time algorithm that either schedules a room, a time slot, and a proctor for every course's final exam, or correctly reports that no such scheduling is possible. Prove the correctness of your algorithm and show its time complexity.

[Hint: Give reduction to a suitable known problem and prove why the reduction gives the correct answer.]

Solution: We define a network $G = (V, E)$, such that V consists of the following types of vertices:

- A source vertex s'
- Vertex c_i for each course i
- Vertex r_j for each room j
- Vertex t_k for each time slot k
- Vertex p_l'for each proctor l
- A sink vertex t'.

The edges in G and their corresponding capacities are defined as follows:

- Add directed edge (s', c_i) with capacity 1 for all courses i. This corresponds to the statement that each course has at most 1 final exam.
- For all courses i and rooms j, if $E[i] \leq S[j]$, then add the directed edge (c_i, r_j) with capacity 1. This corresponds to the statement that if the course size is at most the size of the room, then the exam can be held in that room.

- For all rooms j and time slots k, add the directed edge (r_j, t_k) with capacity 1. This corresponds to the statement that at most one exam can be held in a room at a given time.
- For all time slots k and proctors l, if $A[l,k] = 1$, then add the directed edge (t_k, p_l) with capacity 1. This corresponds to the statement that each proctor can invigilate at most one exam at a time slot in which they are available.
- Add directed edge (p_l, t') with capacity 3 for all proctors l. This corresponds to the statement that each proctor can invigilate at most 3 exams.

Now each path from s' to t' corresponds to a choice of course, rooms, time slot and proctor for a final exam. Conversely each valid choice of course, room, time slot and final exam, corresponds to a path from s' to t'. Therefore if the maximum flow in the above network, has value n then a scheduling of all courses is possible, otherwise not. And each $s' - t'$ path of the flow corresponds to a valid assignment for a course.

Constructing the network G takes time quadratic in the size of the input. We know max flow can be computed in polynomial time, hence the time complexity is polynomially bounded.