CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Final Exam

Due: Monday, May 10, 1:00pm EST

Name: Letao Zhang *NetID: 186004459*

Instructions

1. Do not forget to write your name and NetID above, and to sign Rutgers honor pledge below.
2. The exam contains 4 problems worth 100 points in total *plus* one extra credit problem worth 10 points.
3. This is a take-home exam. You have exactly 48 hours to finish the exam.
4. The exam should be done **individually** and you are not allowed to discuss these questions with anyone else. This includes asking any questions or clarifications regarding the exam from other students or posting them publicly on Piazza (any inquiry should be sent directly to the Instructor or posted privately on Piazza). You may however consult all the materials used in this course (video lectures, notes, textbook, etc.) while writing your solution, but **no other resources are allowed**.
5. Remember that you can leave a problem (or parts of it) entirely blank and receive 25% of the grade for that problem (or part). However, this should not discourage you from attempting a problem if you think you know how to approach it as you will receive partial credit more than 25% if you are on the right track. But keep in mind that if you simply do not know the answer, writing a very wrong answer may lead to 0% credit.

   The only **exception** to this rule is the extra credit problem: you do not get any credit for leaving the extra credit problem blank, and it is harder to get partial credit on that problem.
6. **You should always prove the correctness of your algorithm and analyze its runtime.** Also, as a general rule, avoid using complicated pseudo-code and instead explain your algorithm in English.
7. You may use any algorithm presented in the class or homeworks as a building block for your solutions.

Rutgers honor pledge:

On my honor, I have neither received nor given any unauthorized assistance on this examination.

Signature: Letao Zhang

Problem. #	Points	Score
1	25	
2	25	
3	25	
4	25	
5	+10	
Total	100 + 10	

Problem 1.

(a) Mark each of the assertions below as True or False and provide a short justification for your answer.

(i) If $f(n) = 2^{\sqrt{\log n}}$ and $g(n) = n$, then $f(n) = \Omega(g(n))$. **(2.5 points)**

Solution. False.
Knowing that if $f(n) = \Omega(g(n))$, then we get $2^{\sqrt{\log n}} \geq n$. Multiply $\log_2$ on both side we get $\sqrt{\log n} \geq \log_2 n$, which is not true. Therefore, the answer should be $g(n) = \Omega(f(n))$.

(ii) If $T(n) = T(n/3) + T(n/4) + O(n)$, then $T(n) = O(n)$. **(2.5 points)**

Solution. True.
Consider the recursion tree. At the root, we have $C \cdot n$ time. In the next level, we have $(1/3 + 1/4) \cdot C \cdot n = 7/12 \cdot C \cdot n$. In the level after that, we have $(1/3 + 1/4)^2 \cdot C \cdot n = (7/12)^2 \cdot C \cdot n$. In general, at level i, we have $(1/3 + 1/4)^{i-1} \cdot C \cdot n = (7/12)^{i-1} \cdot C \cdot n$.
The longest distance between a leaf and the root is $\log_3 n$ and the shortest one is $\log_4 n$. So the total runtime is upper bounded by,

$$T(n) \leq \sum_{i=0}^{\infty} C \cdot n \cdot (7/12)^i = C \cdot n \cdot \frac{1}{1 - 7/12} = O(n).$$

(iii) If P = NP, then all NP-complete problems can be solved in polynomial time. **(2.5 points)**

Solution. True.
All NP-complete problems are also in NP by definition and so if P=NP, they all can be solved in polynomial time also.

(iv) If P $\neq$ NP, then no problem in NP can be solved in polynomial time. **(2.5 points)**

Solution. False.
There are some problem in NP can be verified in polynomial time but cannot be solved in polynomial time.

(b) Prove the following statements.

(i) Suppose G is a directed acyclic graph (DAG) with a unique source s. Then, there is a path from s to v for any vertex v in G. **(7.5 points)**

Solution. We can prove it by using contradiction. Assume there is a non-empty DAG G which has at least one incoming edge. Start at any vertex v_1, and then follow an edge entering v_1 repeatedly in reverse. In the end, we will notice that we could form a cycle, which is a contradiction for DAG (DAG doesn't contain a cycle). Therefore, there is a path from s to v for any vertex v in G.

(ii) Consider a flow network G and a flow f in G. Suppose there is a path from the source to sink such that $f(e) < c_e$ for all edges of the path, i.e., the flow on each edge is strictly less than its capacity. Then, f is *not* a maximum flow in G. **(7.5 points)**

Solution. We can prove it by using contradiction. Assume there is a flow f in G which is the maximum flow. Suppose there has 3 edges from source s to sink t with capacity of a,b,c. Respectively maximum flow f will be $f(a, b, c)$, means that $f(e) = c_e$. But this violates the condition $f(e) < c_e$ that flow on each edge is strictly less than its capacity. Hence our assumption is wrong.

Finally, we get a contradiction that f is not a maximum flow. Thus, our original assumption was false, proving the statement.

Problem 2. We consider a different variant of the Knapsack problem in this question. You are given n items with integer weights $w_1, \ldots, w_n$ and integer values $v_1, \ldots, v_n$ and a target value V. Your goal is to determine the *smallest* knapsack size needed so that you can fit a set items in the knapsack with total value at least V. In other words, you want to *minimize* $\sum_{i \in S} w_i$ subject to $\sum_{i \in S} v_i \geq V$ (over the choice of S from n items).

Design an $O(n \cdot V)$ time dynamic programming algorithm for this problem.

(a) *Specification of recursive formula for the problem (in plain English)*: **(5 points)**

Solution. For any integers $0 \leq i \leq n$ and $0 \leq j \leq V$, define: $K(i, j)$: the smallest knapsack size we needed that we can fit a set items in the knapsack with total value at least V, by picking a subset of the first i items, i.e., items $\{1, ..., i\}$, when we have a knapsack of size j.

(b) *Recursive solution for the formula:* **(7.5 points)**

Solution. The recursive formula for $K(i, j)$ is as follows:

$$K(i,j) = \begin{cases} 0, & if i = 0 or j = 0 \\ min(K(i-1, j-v_i) + w_i, K(i-1, j)), & otherwise \end{cases}$$

(c) *Proof of correctness of the recursive formula:* **(7.5 points)**

Solution. Let us consider the base case of this function first: either when $i = 0$ or $j = 0$. In both cases, we have$K(i, j) = 0$ which is also the value we can achieve by using the first 0 items (i.e., no item at all or $i = 0$) or when the knapsack has no size (i.e., $j = 0$). So the base case of this function matches the specification.

We now consider the larger values of i and j. Suppose first $v_i > j$. In this case, $K(i, j) = K(i-1, j)$.This is correct because we cannot fit item i in a knapsack of size j and thus the best value we can achieve is by picking the best combination of items from the first $i - 1$ items, which is captured by $K(i - 1, j)$. Thus, whenever $v_i > j, K(i, j) = K(i - 1, j)$ precisely captures the specification we had. Finally, we have the case when $v_i \leq j$ (corresponding to the last line of the recursive formula). At this point, we have two options in front of us for maximizing the value of items.

We either pick item i in our solution which leaves us with the first $i - 1$ items remaining to choose from next and a knapsack of size $j - v_i$ but we also collected the value w_i. Hence, in this case, we can obtain the value of $K(i - 1, j - v_i) + w_i$ is the largest value we can get by picking a subset of the first $i - 1$ items in a knapsack of size $j - v_i$which is precisely the size of our knapsack after we pick item i in the solution).

By picking the smallest of these two options in the formula, we obtain the best solution for $1, ..., i$, proving correctness. We should note that since $K(i) = 0$ for either when $i = 0$ or $j = 0$, we do not need to worry as the value of K on this entry is anyway 0.

(d) *Runtime analysis:* **(5 points)**

Solution. There are n choices for i and V choices for j so there are in total $n \cdot V$ subproblems. Each subproblem also, ignoring the time it takes to do the inner recursions, takes $O(1)$ time. Hence, the runtime of the algorithm is $O(n \cdot V)$.

Problem 3. You are given a directed graph $G = (V, E)$ such that every *edge* is colored red, yellow, or green, and two vertices s and t. We say that a path from s to t is a *good* path if (1) it has *at least one* edge of each color, and (2) all the red edges in the path appear before all the yellow edges, and all the yellow edges appear before the green edges. For instance, a $(red, red, yelow, green, green, green)$ path is a good path but neither a $(red, green, green)$ path nor a $(red, green, yellow)$ path are good.

Design and analyze an $O((m + n) \cdot n)$ time algorithm that outputs the size of the *largest* collection of *edge-disjoint good* paths from s to t in a given directed graph $G = (V, E)$ with n vertices and m edges.

(25 points)

Solution. *Algorithm.* We need to find the maximum collection of edge-disjoint paths from a vertex s to a vertex t in a given graph $G = (V, E)$. Firstly, using DFS from s to t, record all the different paths from s to t and store them. After that iterate over all the paths and store the colors. Secondly, using hash-maps to check for the colors whether it is in the right order or not. Simply turn G into a network by assigning capacity 1 to every edge. Then find the maximum flow from s to t in this network and return the edges with non-zero flow as the edges of the paths.

Proof of Correctness. We prove that there is a flow of value k in G' if and only if there is a collection of k colorful paths in G. This implies that the maximum value of flow in G' is equal to the size of the largest collection of colorful paths in G.

1) Suppose there are *edge-disjoint good* paths paths in G, create a flow f of value k as follows. We could sent 1 unit of flow along good path without violating the capacity constraints. If we give each edge capacity 1, then the max flow from s to t assigns a flow of either 0 or 1 to every edge. Since any vertex of G lies on at most two saturated edges, the subgraph S of saturated edges is the union of several edge-disjoint good paths and cycles. Moreover, the number of paths is exactly equal to the value of the flow. Since in the collection of colorful paths no vertex is used more than once, we will not use any the edges above more than once also and thus this is a valid flow with the same value as number of paths in the collection, which outputs the size of the *largest* collection of *edge-disjoint good* paths from s to t.

2) Suppose now there is a flow of value k in G', create a collection of *edge-disjoint good* paths paths in G as follows. Determine that it has *at least one* edge of each color, and all the red edges in the path appear before all the yellow edges, and all the yellow edges appear before the green edges. Check for the correctness of the order of colors while using dfs and store only those paths which have the correct order of colors. Since the path should be edge-disjoint. So whenever in some other path, you will find a used node, you can simply not count that path as it won't be edge-disjoint. Moreover, since capacity of every edge (u_v, w_v) is only 1 in G', no vertex can appear in more than once of these paths. Thus, the flow paths give us *edge-disjoint good* paths in G.

Runtime Analysis. Network G' has $O(n)$ vertices and $O(m+n)$ edges, and maximum flow F in the network is at most n(as there are at most n edges of capacity 1 going out of s). Thus, by running Ford-Fulkerson algorithm for max-flow, the running of time of this algorithm is $O((m+n) \cdot F) = O((m+n) \cdot n)$ as desired.

Problem 4. Prove that the following problems are NP-hard. For each problem, you are only allowed to use a reduction from the problem specified.

(a) 4**-Coloring Problem:** Given an undirected graph $G = (V, E)$, is there a 4-coloring of vertices of G? (A 4-coloring is an assignment of colors $\{1, 2, 3, 4\}$ to vertices so that no edge gets the same color on both its endpoints). **(12.5 points)**

For this problem, use a reduction from the 3-*Coloring problem.* Recall that in the 3-Coloring problem, you are given a graph $G = (V, E)$ and the goal is to find whether there is a 3-coloring of G or not. A 3-coloring is an assignment of colors $\{1, 2, 3\}$ to vertices so that no edge gets the same color on both its endpoints

Solution. *Reduction.* If the graph is 3-colorable, we can simply use any 3-coloring of the graph as a proof. So the input to our verifier is the input graph G and a supposed 3-coloring of G. The verifier then goes over the edges of G one by one to ensure that no edge is monochromatic. Creating an instance $G' = (V, E)$ of 4-coloring as follows.

The following is a verifier V for 4-coloring is that $V = (v, e)$. 1) Check that e includes ≤ 4 colors. 2)Color each vertex of v as specified by e. 3)For each vertex, check that it has a unique color from each of its neighbors. 4) If all checks pass, *Yes*; otherwise, *No.*

Proof of Correctness. We give a polynomial-time reduction from 3-coloring to 4-coloring. The reduction maps a graph G into a new graph G' such that $G \in$ 3-coloring if and only if $G' \in$ 4-coloring. We do so by setting G' to G, and then adding a new vertex v and connecting it to each vertex in G'.

If G is 3-colorable, then G' can be 4-coloring exactly as G with being the only vertex colored with the additional color. Similarly, if it is 4-colorable, then we know that the vertex v must be the only vertex of its color – this is because it is connected to every other vertex in G'. Thus, we know that G must be 3-colorable.

Runtime Analysis. This reduction can be implemented in $O(n + m)$ time and hence a poly-time algorithm for 3-coloring problem which in turn implies P = NP (by definition of 3-coloring problem being NP-hard). Thus a poly-time algorithm for 4-coloring also implies P = NP, making this problem NP-hard.

(b) **Hamiltonian Path Problem:** Given an undirected graph $G = (V, E)$, does G contain a path that goes through all vertices, i.e., a Hamiltonian path? **(12.5 points)**

For this problem, use a reduction from the *s-t Hamiltonian Path problem.* Recall that in the s-t Hamiltonian Path problem, you are given a graph $G = (V, E)$ and two vertices s, t and the goal is to decide whether there is a s-t path in G that passes through all other vertices.

(Note that the difference between Hamiltonian Path problem and s-t Hamiltonian Path problem is that in the former problem, the path can start from any vertex and end in any vertex as long as it goes through all vertices, while in the latter it should start from s and ends at t.)

Solution. *Reduction.* We are using reduction from the *s-t Hamiltonian Path problem.* Given an instance $G = (V, E)$ of the undirected $s-t$ Hamiltonian path problem, we create an instance G' as follows.

Proof of Correctness. We show that G has a $s-t$ Hamiltonian path if and only if G' has a Hamiltonian path.

1) If G has a $s-t$ Hamiltonian path P, then G' has a Hamiltonian path. Whether a Hamiltonian Path exists in a graph is in NP, because given a set of edges, we can check in polynomial time whether we've gone through all the vertices. We can check if a potential $s-t$ path is Hamiltonian in G in polynomial time.

2) If G has a Hamiltonian path, then G has a $s-t$ Hamiltonian path. Let P be the $s-t$ Hamiltonian path in G', deciding whether there is a $s-t$ path in G' that passes through all other vertices. So a Hamiltonian path in G' visits variable vertices in order from the vertices s to v.

Runtime Analysis. This reduction can be implemented in $O(n + m)$ time and hence a poly-time algorithm for the Hamiltonian Path Problem implies a poly-time algorithm for undirected $s-t$ Hamiltonian path which in turn implies P = NP (by definition of $s-t$ Hamiltonian path being NP-hard). Thus a poly-time algorithm for Hamiltonian Path Problem also implies P = NP, making this problem NP-hard.

Problem 5. [**Extra credit**] Alice wants to throw a party and is deciding who to invite. She has n people to choose from and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to the constraint that at the party, each person should know at least five other people.

Give a polynomial time algorithm that takes as input a list of n people and the list of pairs who know each other and outputs the maximum number of guests that Alice can invite.

(+10 points)

Hint: Get creative and design an algorithm for this problem from scratch; this problem is *not* about using reductions to the problems you have already seen in this course.

Solution. We are using graph to solve this problem. Assume there is a graph $G(V, E)$ where the vertex compose of each person. If there is an edge between vertices u and v, means that two of them are knowing each other. Suppose someone does not have five people they know. Then, we should remove them from consideration because we are told to invite the person who know at least five other people. In each iteration, we delete at least one person each time. When the algorithm terminates, by definition, the subset of invitees is valid. Since we only delete people who are unlikely to be invited, we always generate as many invitees as possible. Therefore, the total runtime is $O(n)$ time.

Extra Workspace

Extra Workspace

Extra Workspace