CS 170 Efficient Algorithms and Intractable Problems
Fall 2016 Christos Papadimitriou and Luca Trevisan

# Final

Name:

SID:

GSI and section time:

*Write down the names of the students on your left and right as they appear on their SID.*

Name of student on your left:

Name of student on your right:

Name of student behind you:

Name of student in front of you:

*Instructions:*

Write your name and SID on each sheet in the spaces provided.

You may consult two handwritten, double-sided sheet of notes. You may not consult other notes, textbooks, etc. Cell phones and other electronic devices are not permitted.

There are 11 questions. The last page is page 16.

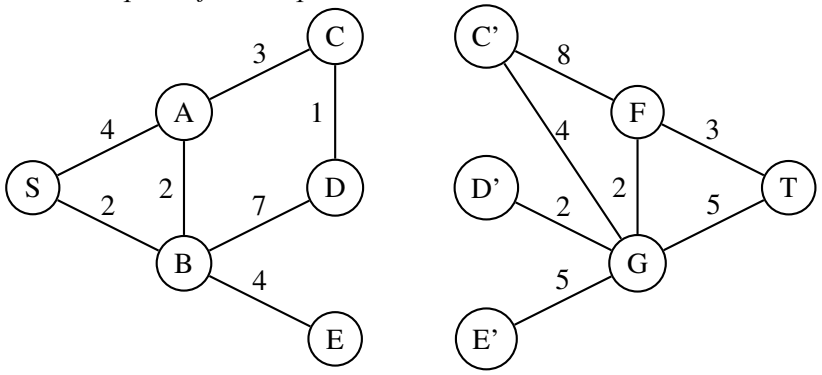
Be sure to respond in the format we request, noting when justifications are or aren't required. Write in the space provided. Good luck!

*Note:* If you finish in the last 15 minutes, please remain seated and do not leave early, to avoid distracting your fellow classmates.

Do not turn this page until an instructor tells you to do so.

**1. (8 pts.) Graph Games**

*No justifications are required for this question.*

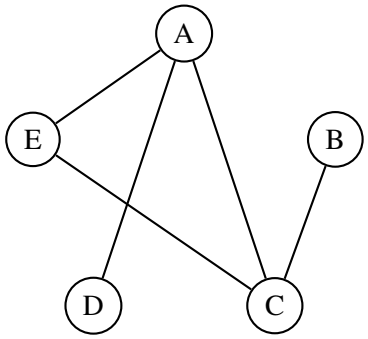



(a) In the undirected graph above, edges are labeled with their weights. $S$ and $T$ are separated by a river, but you can build a length-0 edge between $C$ and $C'$, $D$ and $D'$, or $E$ and $E'$, to cross the river and connect them. You'd like to build the bridge so that the resulting $S-T$ distance is as short as possible.

Where do you build the bridge? (answer $C$, $D$, or $E$)

What is the resulting $S-T$ length?

(b) Consider the undirected graph below:




For each question, answer with the letters of the vertices in the solution (not the size of the solution).

What is the smallest vertex cover of this graph?

What is the largest independent set of this graph?

What is the largest clique of this graph?

**2. (8 pts.) Public Funds**

Consider the following problem PUBLIC FUNDS:

> You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have $m$ bank accounts at your disposal to use to pay for your fence; each account $i$ has a balance of $b_i$. You must choose one of $n$ options for your fence; each fence $j$ costs $c_j$ dollars. You would like to withdraw from at most $k$ of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all $b_m$ dollars.)
>
> Determine whether it is possible to exactly pay for some fence $j$; that is, whether there is a $j$ between 1 and $n$ such that you can withdraw exactly $c_j$ dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$, and $k$, and if so return the corresponding choice of fence and set of bank accounts that you withdraw from.

Show that PUBLIC FUNDS is NP-Complete, by arguing it is in NP and then giving and briefly justifying a reduction that shows it is NP-hard.

**3. (32 pts.) Scheduling Meetings: Greedy, dynamic, or intractable**

You are the scheduling officer for the CS department. The department has one conference room, and every day you receive a list $L$ of $n$ requests for it, in no particular order: $L = (s_1, e_1), \ldots, (s_n, e_n)$. Each item in the list $(s_i, e_i)$ is a pair of integers representing start and end times, respectively, with $s_i < e_i$. Your goal is to schedule as many meetings as possible in one day, with the constraint that meetings cannot conflict with each other (the intervals for the scheduled meetings must be disjoint). Assume for simplicity that all of these integers are distinct.

(a) *Y*ou consider a greedy algorithm for this problem:
*Choose the meeting that starts first, and delete all conflicting meetings; repeat.*
Give a simple example with just three meetings showing that this cannot work.

(b) Now find an optimal greedy algorithm: given the list $L$ as specified above, your algorithm should return a list of non-conflicting meetings of maximum length.

(c) Briefly justify the correctness of your algorithm.

(d) What is the running time of this algorithm, given the unsorted list of $n$ meetings? Justify briefly.

(e) Now suppose each meeting also has a positive integer weight $w_i$, and you want to schedule the set of meetings that maximize the total weight. Your input is the list $L$ of $n$ requests, again in no particular order, with each element now formatted as $(s_i, e_i, w_i)$. Since the greedy algorithm no longer works, you design a dynamic programming algorithm to return the maximum weight achievable. (*Note*, you do not need to return the set of meetings that achieve it).

First, define your subproblems in words:

For $i = 1, \ldots, n$ $W[i] =$_____

Give clear pseudocode for this problem. You do not need to justify your solution.

(f) What is the running time of this algorithm, given the unsorted list of $n$ meetings? Justify briefly.

(g) Now meetings have no weights, but can have one or more *breaks.* For example, one meeting may require the conference room from 1 to 3, from 4 to 5, and 8 to 10. When you schedule a meeting, you must schedule all its parts. You want to find a list of at least $k$ meetings that can be scheduled together.

*Note:* To make this problem work, assume that the integers specifying meeting start and end times can be arbitrarily large. Perhaps we're on a planet with no rotation, so that the day is endless, or the integers can represent arbitrarily small units of time.

*Continued on the following page*

Show that SCHEDULING WITH BREAKS is NP-complete. Argue briefly that it is in NP, and then show it is NP-hard with a reduction involving a known NP-complete problem. Argue that your reduction works.

*Hint: Independent Set*

(h) You are now in an EECS department, and you get scheduling requests from both CS folks and EE folks, with breaks as above. But you know that no two CS requests ever conflict with each other, and neither do two EE requests. Again, you want to maximize the number of meetings scheduled.

Briefly explain why this problem can be solved in polynomial time.

**4. (8 pts.) Rook Programming**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4 | | | | X |
| 3 | | | X | |
| 2 | X | | X | |
| 1 | | | | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4 | | | R | X |
| 3 | | R | X | |
| 2 | X | | X | R |
| 1 | R | | | |

You are given an $n$ by $n$ chessboard that has several holes (marked with an X in the example above left). You want to place as many rooks as possible on the blank spaces on this board (rooks cannot be placed in the holes) such that no two rooks attack each other: that is, no two rooks can be in the same row or column. For example, the rook placement in the above right is an example of a solution for that instance.

Describe an efficient algorithm for this problem. *Hint:* use bipartite matching.

**5. (10 pts.) Parallelism Party**

For each part, briefly describe how to parallelize the given algorithm, and give the depth of the parallelized algorithm in $\Theta(\cdot)$ notation in terms of $n$. Assume that spawning an arbitrary number of processes in a single for loop has depth $O(1)$.

(a) The FFT, where $n$ is the length of the list of coefficients. *Note:* your solution should have polylogarithmic depth, $O(\log^k n)$ for some $k$.

depth: $\square$

(b) The edit distance DP algorithm, where $n$ is the length of both strings. *Note:* your solution should have at most linear depth, $O(n)$.

depth: $\square$

(c) Checking whether there is a cycle in a directed graph $G = (V, E)$, where $n = |V|$ and $|E| = \Theta(n^2)$. *Note:* your solution should have polylogarithmic depth, $O(\log^k n)$ for some $k$.

depth: $\square$

**6. (6 pts.) Selected Shorts**

Briefly justify each answer.

(a) Suppose that in a weighted graph with negative weights we know that the shortest path from $s$ to $t$ has at most $\log |V|$ edges. Can we find this shortest path in $O(|V|^2 \log |V|)$ time?

(b) You know that a graph with $n$ vertices has a Rudrata cycle, and you do not know anything else about it. Give the tightest possible lower and upper bounds for the size of the smallest vertex cover of the graph.

**7. (15 pts.) True or False?**

*Write T or F in the box. Unlike the midterms, you do not need to justify your answer.*

(a) ☐ The recurrence $T(n) = 3T(n/4) + n\log n$ has solution $\Theta(n\log^2 n)$.

(b) ☐ One can determine whether a weighted directed graph has a negative cycle in $O(|V|^3)$ time.

(c) ☐ In the experts/multiplicative weights update problem running for $T$ days, it is possible to design a deterministic algorithm with a non-normalized regret (total amount of your loss minus the best expert's loss) which is always less than $T$ for every adversary.

*In the following questions there are four possible answers:* True, False, True iff P = NP, True iff P $\neq$ NP. Write T, F, =, or $\neq$ in the box. No justifications are required. By "reduction" we always mean "polynomial-time reduction."

(d) [ ] There is a reduction from LINEAR PROGRAMMING to INTEGER LINEAR PROGRAMMING.

(e) [ ] There is a reduction from LINEAR PROGRAMMING to MAX FLOW.

(f) [ ] There is a known polynomial-time algorithm for 3SAT.

(g) [ ] There is a polynomial time algorithm for 3SAT. **True iff P=NP**. The above statement is synonymous with "3SAT is in P." Since 3SAT is an NP-complete problem, that is equivalent to saying P=NP.

(h) [ ] The 2-MST problem (find the minimum spanning tree so that a given vertex $v$ has degree $\leq 2$) can be solved in polynomial time.

(i) [ ] The All-2-MST problem (find the minimum spanning tree so that all vertices have degree $\leq 2$) can be solved in polynomial time.

(j) [ ] 2SAT is NP-complete.

**8. (10 pts.) MST Mishap** You found the MST of a huge graph $G = (V, E)$, but afterwards you realized that you made a small mistake: the weight of a particular edge $e$ is not $w(e)$, but instead $w'(e)$. Give a linear-time algorithm for finding the true MST. For each of the four cases, clearly describe your algorithm (with pseudocode if desired); you do not need to justify correctness.

1. $e$ is in the current MST and $w'(e) < w(e)$.

2. $e$ is in the current MST and $w'(e) > w(e)$.

3. $e$ is not in the current MST and $w'(e) > w(e)$.

4. $e$ is not in the current MST and $w'(e) < w(e)$.

**9. (8 pts.) Shortest Paths with Order Oracle**

Given a connected, directed graph $G = (V, E)$ with positive edge weights, and vertices $s$ and $t$, you want to find the shortest path from $s$ to $t$ in a graph in $O(|E| + |V|)$ time. Normally, this wouldn't be possible; however, you have consulted the Order Oracle, who gave you a list $L$ of the $V$ vertices in increasing order of the shortest-path distance from $s$.

In other words, $L[1]$ is $s$, $L[2]$ is the closest other vertex to $s$, $L[|V|]$ is the farthest vertex from $s$, etc. Find an algorithm SHORTESTPATH$(G, s, t, L)$ which computes the length of the shortest $s-t$ path in $O(|E| + |V|)$ time.

*Circle* the shortest path algorithm you're modifying.

Dijkstra's Bellman-Ford Shortest path in DAG BFS DFS

Describe, *in three sentences or less*, the modification to this algorithm. Any explanations over three sentences will be ignored.

**10. (8 pts.) Streaming**

(a) Show that, for every $\varepsilon > 0$, you can use the count-min data structure to determine the frequency of the most frequent element within an *additive* error of $\varepsilon \cdot n$, where $n$ is the number of items in the stream, with the same space complexity $O(\varepsilon^{-1}(\log n)(\log n + \log \Sigma))$ of count-min. (You do not need to rederive the space complexity analysis and correctness analysis of count-min.)

(b) If $\varepsilon$ is a constant and $|\Sigma|$ is polynomial in $n$, then the above algorithm uses space polynomial in $\log n$, but it only guarantees an additive, not multiplicative, approximation. Prove that deterministically estimating the frequency of the most frequent element within a multiplicative factor smaller than two requires linear space: specifically, the algorithm would never underestimate the frequency of an element, and would always predict less than twice the true frequency. *Hint: Recall that it takes linear space to tell if all elements are distinct.*

**11. (8 pts.) Precipitation Prognostication**

There are 1000 mysterious email addresses which have each been sending you an email every morning at 6am. (Each morning, you get 1000 emails, one from each address). Each email contains only the single word `rain` or `sun`. Spooky! You realize that these are each predictions for the day's weather, and that one of the email addresses (you don't know which) is the infallible Whether Channel, which is always correct. The other email addresses are trolls.

Every day, you must decide whether or not to take an umbrella on your commute; afterwards, you learn whether it actually rains. You have made a mistake that day if you bring an umbrella and it does not rain, or you do not bring an umbrella and it does. This process goes on forever.

Find an algorithm, which, given the history of predictions up until the current day, outputs whether or not to take an umbrella that day. Your algorithm should guarantee that, no matter how long the process goes on, you will make at most 10 mistakes, ever, despite the fact that the trolls have hacked into your computer and will know your algorithm.

Clearly describe your algorithm (with pseudocode if desired) and justify its correctness.

**Blank Space**

*You can use this page for problem solutions or as scratch space.*