CS 170 Algorithms
Fall 2014 David Wagner
MT2

PRINT your name: _____, _____
(last) (first)

SIGN your name: _____

Your Student ID number: _____

Your Unix account login: cs170-_____

The room you are sitting in right now: _____

Name of the person
sitting to your left: _____

Name of the person
sitting to your right: _____

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

There are 8 questions. The last page is page 13.

Answer all questions. Be precise and concise. Write in the space provided. Good luck!

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

Problem 1. [True or false] (9 points)

Circle TRUE or FALSE. Do not justify your answers on this problem.

(a) TRUE or FALSE: Let $(S, V-S)$ be a minimum (s,t)-cut in the network flow graph G. Let (u,v) be an edge that crosses the cut in the forward direction, i.e., $u \in S$ and $v \in V-S$. Then increasing the capacity of the edge (u,v) necessarily increases the maximum flow of G.

(b) TRUE or FALSE: All instances of linear programming have exactly one optimum.

(c) TRUE or FALSE: If all of the edge capacities in a graph are an integer multiple of 7, then the value of the maximum flow will be a multiple of 7.

(d) TRUE or FALSE: If we want to prove that a search problem X is NP-complete, it's enough to reduce 3SAT to X (in other words, it's enough to prove 3SAT $\leq_P X$).

(e) TRUE or FALSE: If we want to prove that a search problem X is NP-complete, it's enough to reduce X to 3SAT (in other words, it's enough to prove $X \leq_P$ 3SAT).

(f) TRUE or FALSE: For every graph G and every maximum flow on G, there always exists an edge such that increasing the capacity on that edge will increase the maximum flow that's possible in the graph.

(g) TRUE or FALSE: Suppose the maximum (s,t)-flow of some graph has value f. Now we increase the capacity of every edge by 1. Then the maximum (s,t)-flow in this modified graph will have value at most $f+1$.

(h) TRUE or FALSE: There is no known polynomial-time algorithm to solve maximum flow.

(i) TRUE or FALSE: If problem A can be reduced to problem B, and B can be reduced to C, then A can also be reduced to C.

(j) TRUE or FALSE: If X is any search problem, then X can be reduced to INDEPENDENT SET.

(k) TRUE or FALSE: If we can find a single problem in NP that has a polynomial-time algorithm, then there is a polynomial-time algorithm for 3SAT.

(l) TRUE or FALSE: If there is a polynomial-time algorithm for 3SAT, then every problem in NP has a polynomial-time algorithm.

(m) TRUE or FALSE: We can reduce the search problem MAXIMUM FLOW to the search problem LINEAR PROGRAMMING (in other words, MAXIMUM FLOW $\leq_P$ LINEAR PROGRAMMING).

(n) TRUE or FALSE: We can reduce the search problem LINEAR PROGRAMMING to the search problem INTEGER LINEAR PROGRAMMING (in other words, LINEAR PROGRAMMING $\leq_P$ INTEGER LINEAR PROGRAMMING).

(o) TRUE or FALSE: Every problem in P can be reduced to 3SAT.

(p) TRUE or FALSE: Suppose we have a data structure where the amortized running time of Insert and Delete is $O(\lg n)$. Then in any sequence of $2n$ calls to Insert and Delete, the worst-case running time for the nth call is $O(\lg n)$.

(q) TRUE or FALSE: Suppose we do a sequence of m calls to Find and m calls to Union, in some order, using the union-find data structure with union by rank and path compression. Then the last call to Union takes $O(\lg^* m)$ time.

Problem 2. [Short answer] (18 points)

Answer the following questions, giving a short justification (a sentence or two).

(a) If $P \neq NP$, could there be a polynomial-time algorithm for 3SAT?

(b) If $P \neq NP$, could GRAPH 2-COLORING be NP-complete?

(c) If we have a dynamic programming algorithm with n^2 subproblems, is it possible that the running time could be asymptotically strictly more than $\Theta(n^2)$?

(d) If we have a dynamic programming algorithm with n^2 subproblems, is it possible that the space usage could be $O(n)$?

(e) Suppose that we implement an append-only log data structure, where the total running time to perform any sequence of n Append operations is at most $3n$. What is the amortized running time of Append?

(f) Suppose we want to find an optimal binary search tree, given the frequency of bunch of words. In other words, the task is:

Input: n words (in sorted order); frequencies of these words: $p_1, p_2, \ldots, p_n$.

Output: The binary search tree of lowest cost (defined as the expected number of comparisons in looking up a word).

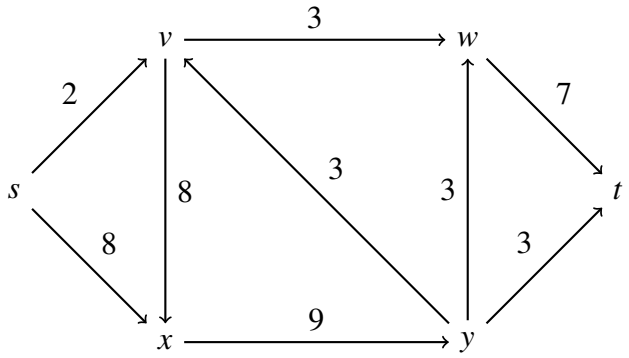
Prof. Cerise suggests defining $C(i) =$ the cost of the optimal binary search tree for words $1 \ldots i$, writing a recursive formula for $C(i)$, and then using dynamic programming to find all the $C(i)$.

Prof. Rust suggests defining $R(i, j) =$ the cost of the optimal binary search tree for words $i \ldots j$, writing a recursive formula for $R(i, j)$, and then using dynamic programming to find all the $R(i, j)$.

One of the professors has an approach that works, and one has an approach that doesn't work. Which professor's approach can be made to work? In what order should we compute the C values (if you choose Cerise's approach) or R values (if you choose Rust's approach)?

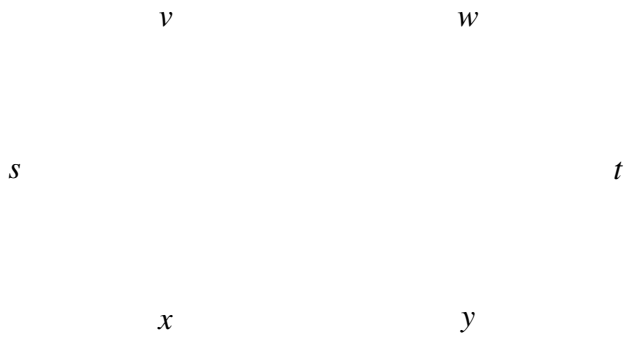
Problem 3. [Max flow] (10 points)

Consider the following graph G. The numbers on the edges represent the capacities of the edges.

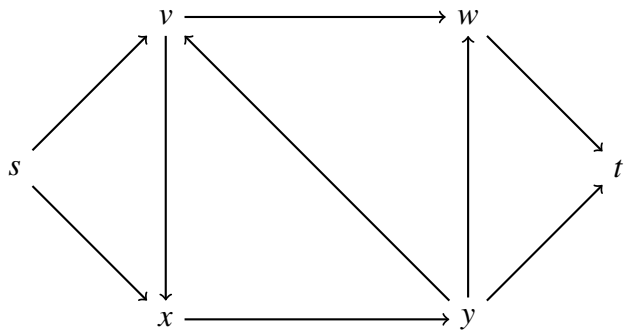


(a) How much flow can we send along the path $s \to x \to y \to v \to w \to t$?

(b) Draw the resulting residual graph after sending as much flow as possible along the path $s \to x \to y \to v \to w \to t$, by filling in the skeleton below with the edges of the residual graph. Label each edge of the residual graph with its capacity.

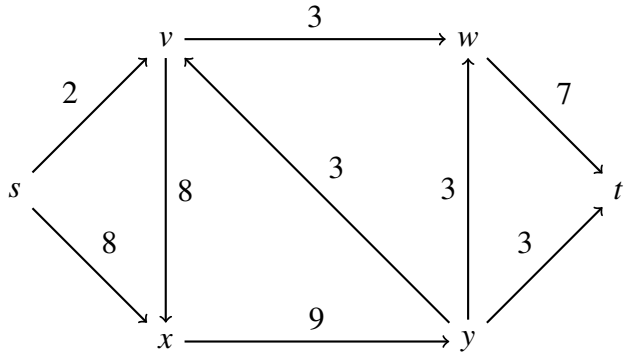


(c) Find a maximum (s,t)-flow for G. Label each edge below with the amount of flow sent along that edge, in your flow. (You can use the blank space on the next page for scratch space if you like.)



(cont. on next page)

(d) Draw a minimum (s,t)-cut for the graph G (shown below again).



Problem 4. [Max flow] (12 points)

We would like an efficient algorithm for the following task:

Input: A directed graph $G = (V, E)$, where each edge has capacity 1; vertices $s, t \in V$; a number $k \in \mathbb{N}$

Goal: find k edges that, when deleted, reduce the maximum $s-t$ flow in the graph by as much as possible

Consider the following approach:

1. Compute a maximum (s,t)-flow f for G.
2. Let G^f be the residual graph for G with flow f.
3. Define a set S of vertices by (something).
4. Define a set T of edges by (something).
5. Return any k edges in T.

(a) How could we define the sets S and T in steps 3–4, to get an efficient algorithm for this problem?

$S =$

$T =$

(b) Is there an algorithm to implement step 1 in $O(|V||E|)$ time or less? If yes, what algorithm should we use? If no, why not? Either way, justify your answer in a sentence or two.

Problem 5. [Dynamic programming] (12 points)

Let $A[1..n]$ be a list of integers, possibly negative. I play a game, where in each turn, I can choose between two possible moves: (a) delete the first integer from the list, leaving my score unchanged, or (b) add the sum of the first two integers to my score and then delete the first three integers from the list. (If I reach a point where only one or two integers remain, I'm forced to choose move (a).)

I want to maximize my score. Design a dynamic programming algorithm for this task. Formally:

Input: $A[1..n]$

Output: the maximum score attainable, by some sequence of legal moves

For instance, if the list is $A = [2,5,7,3,10,10,1]$, the best solution is $5+7+10+10 = 32$.

You do not need to explain or justify your answer on any of the parts of this question.

(a) Define $f(j) =$ the maximum score attainable by some sequence of legal moves, if we start with the list $A[j..n]$. Fill in the following base cases:

$f(n) =$

$f(n-1) =$

$f(n-2) =$

(b) Write a recursive formula for $f(j)$. You can assume $1 \leq j \leq n-3$.

$f(j) =$

(c) Suppose we use the formulas from parts (a) and (b) to solve this problem with dynamic programming. What will the asymptotic running time of the resulting algorithm be?

(d) Suppose we want to minimize the amount of space (memory) used by the algorithm to store intermediate values. Asymptotically, how much space will be needed, as a function of n? (Don't count the amount of space to store the input.) Use $\Theta(\cdot)$ notation.

Problem 6. [Greedy cards] (12 points)

Ning and Evan are playing a game, where there are n cards in a line. The cards are all face-up (so they can both see all cards in the line) and numbered 2–9. Ning and Evan take turns. Whoever's turn it is can take one card from either the right end or the left end of the line. The goal for each player is to maximize the sum of the cards they've collected.

(a) Ning decides to use a greedy strategy: "on my turn, I will take the larger of the two cards available to me". Show a small counterexample ($n \leq 5$) where Ning will lose if he plays this greedy strategy, assuming Ning goes first and Evan plays optimally, but he could have won if he had played optimally.

(b) Evan decides to use dynamic programming to find an algorithm to maximize his score, assuming he is playing against Ning and Ning is using the greedy strategy from part (a). Let $A[1..n]$ denote the n cards in the line. Evan defines $v(i, j)$ to be the highest score he can achieve if it's his turn and the line contains cards $A[i..j]$.

Evan needs a recursive formula for $v(i, j)$. Fill in a formula he could use, below.

Evan suggests you simplify your expression by expressing $v(i, j)$ as a function of $\ell(i, j)$ and $r(i, j)$, where $\ell(i, j)$ is defined as the highest score Evan can achieve if it's his turn and the line contains cards $A[i..j]$, if he takes $A[i]$; also, $r(i, j)$ is defined to be the highest score Evan can achieve if it's his turn and the line contains cards $A[i..j]$, if he takes $A[j]$. Write an expression for all three that Evan could use in his dynamic programming algorithm. You can assume $1 \leq i < j \leq n$ and $j - i \geq 2$. Don't worry about base cases.

$v(i, j) =$

where $\ell(i, j) =$

$r(i, j) =$

(c) What will the running time of the dynamic programming algorithm be, if we use your formula from part (b)? You don't need to justify your answer.

Problem 7. [Tile David's walkway] (14 points)

David is going to lay tile for a long walkway leading up to his house, and he wants an algorithm to figure out which patterns of tile are achievable. The walkway is a long strip, n meters long and 1 meter wide. Each 1×1 meter square can be colored either white or black. The input to the algorithm is a pattern $P[1..n]$ that specifies the sequence of colors that should appear on the walkway. There are three kinds of tiles available from the local tile store: a 1×1 white tile (W), a 2×1 all-black tile (BB), and a 3×1 black-white-black tile (BWB). Unfortunately, there is a limited supply of each: the tile store only has p W tiles, q BB's, and r BWB's in stock.

Devise an efficient algorithm to determine whether a given pattern can be tiled, using the tiles in stock. In other words, we want an efficient algorithm for the following task:

Input: A pattern $P[1..n]$, integers $p, q, r \in \mathbb{N}$

Question: Is there a way to tile the walkway with pattern P, using at most p W's, q BB's, and r BWB's?

For example, if the pattern P is WBWBWBBWWBWBW and $p = 5$, $q = 2$, and $r = 2$, the answer is yes: it can be tiled as follows: |W|BWB|W|BB|W|W|BWB|W|.

For this problem, we suggest you use dynamic programming. Define

$$f(j,p,q,r) = \begin{cases} \text{True} & \text{if there's a way to tile } P[j..n] \text{ using at most } p \text{ W's, } q \text{ BB's, and } r \text{ BWB's} \\ \text{False} & \text{otherwise.} \end{cases}$$

You don't need to justify or explain your answer to any of the following parts. You can assume someone else has taken care of the base cases ($j = n-2, n-1, n$), e.g., $f(n,p,q,r) = (P[n] = W \wedge p \geq 1)$ and so on; you don't need to worry about them.

(a) Write a recursive formula for f. You can assume $1 \leq j \leq n-3$.

$f(j,p,q,r) =$

(b) If we use your formula from part (a) to create a dynamic programming algorithm for this problem, what will its asymptotic running time be?

Problem 8. [Walkways, with more tiles] (13 points)

Now let's consider SUPERTILE, a generalization of Problem 7. The SUPERTILE problem is

Input: A pattern $P[1..n]$, tiles $t_1, t_2, \ldots, t_m$

Output: A way to tile the walkway using a subset of the provided tiles in any order, or "NO" if there's no way to do it

Each tile t_i is provided as a sequence of colors (W or B). Each particular tile t_i can be used at most once, but the same tile-sequence can appear multiple times in the input (e.g., we can have $t_i = t_j$). The tiles can be used in any order.

For example, if the pattern P is WBBBWWB and the tiles are $t_1 =$ BBW, $t_2 =$ B, $t_3 =$ WB, $t_4 =$ WB, then the answer is yes, $t_3 t_1 t_4$ (this corresponds to |WB|BBW|WB|).

(a) Is SUPERTILE in NP? Justify your answer in a sentence or two.

(b) Prof. Mauve believes she has found a way to reduce SUPERTILE to SAT. In other words, she believes she has proven that SUPERTILE $\leq_P$ SAT. If she is correct, does this imply that SUPERTILE is NP-hard? Justify your answer in a sentence or two.

(c) Prof. Argyle believes he has found a way to reduce SAT to SUPERTILE. In other words, he believes he has proven that SAT $\leq_P$ SUPERTILE. If he is correct, does this imply that SUPERTILE is NP-hard? Justify your answer in a sentence or two.

(d) Consider the following variant problem, ORDEREDTILE:

Input: A pattern $P[1..n]$, tiles $t_1, t_2, \ldots, t_m$

Output: A way to tile the walkway using a subset of the tiles, in the provided order, or "NO" if there's no way to do it

In this variant, you cannot re-order the tiles. For instance, if P is WBBBWWB and the tiles are $t_1 =$ BBW, $t_2 =$ B, $t_3 =$ WB, $t_4 =$ WB, then the output is "NO"; however, if the tiles are $t_1 =$ WB, $t_2 =$ B, $t_3 =$ BBW, $t_4 =$ WB, then the answer is yes, $t_1 t_3 t_4$.

Rohit's advisor asked him to prove that ORDEREDTILE is NP-complete. Rohit has spent the past few days trying to prove it, but without any success. He's wondering whether he just needs to try harder or if it's hopeless. Based on what you've learned from this class, should you encourage him to keep trying, or should you advise him to give up? Explain why, in 2–3 sentences.

You are done!

Scratch space

We won't look at anything on this page.