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

Midterm 2

Name: Joe Solutions

SID: 12345678

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 sheets of notes. You may not consult other notes, textbooks, etc. Cell phones and other electronic devices are not permitted.

There are 5 questions. The last page is page 12.

Answer all questions. On questions asking for an algorithm, make sure to respond in the format we request. 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. (12 pts.) Dynamic Warmup

(a) The recurrence of a dynamic programming algorithm is

$$C[i] = \min\{C[i-1] + a_i, C[i-2] + b_i\},$$

initialized to $C[0] = C[1] = 0$. a_i and b_i, for $i \in 1, \ldots, n$, are constants. The desired answer is $C[n]$.

Write full *iterative* pseudocode for this algorithm; do not use recursion or memoization.

(For convenience, you can assume $C[0]$ and $C[1]$ have been set to 0).

Solution:

```
1: for i = 2..n do
2:     Set C[i] := min{C[i-1] + a_i, C[i-2] + b_i
3: return C[n].
```

(b) Fill in the edit distance table for the words *precise* and *practice*. We've already started it off for you.

Recall that entry (c, r) in an edit distance table contains the edit distance between the length-c prefix of the column word and the length-r prefix of the row word. For example, $E(3,1) = 2$, representing the edit distance between *pre* and *p*.

Then, circle the subproblems that are used to get from the base case $(0,0)$ to the solution.

		P	R	E	C	I	S	E
	0	1	2	3	4	5	6	7
P	1	0	1	2	3	4	5	6
R	2	1	0	1	2	3	4	5
A	3	2	1					
C	4	3	2					
T	5	4	3					
I	6	5	4					
C	7	6	5					
E	8	7	6					

Solution: See other version.

2. (18 pts.) Linear Programming Potpourri

(a) Call the linear program below LP_1. What is the dual?

$$\begin{aligned} \max \quad & x+y \\ \text{subject to:} \quad & x+3y \leq 20 \\ & 2x+y \leq 10 \\ & x,y \geq 0 \end{aligned}$$

Solution:

$$\begin{aligned} \min \quad & 20a+10b \\ \text{subject to:} \quad & a+2b \geq 1 \\ & 3a+b \geq 1 \\ & a,b \geq 0 \end{aligned}$$

(b) What is the optimum solution of LP_1? Give multipliers for the two inequalities of LP_1 which show that this is indeed the optimum value.

Solution: The vertices of the feasible region are (a) $x = 0$, in which case $y \leq 20/3$, yielding a value of $20/3$; (b) $y = 0$, in which case $x \leq 5$, yielding a value of 5; or (c) the intersection of these two lines: $y = (20-x)/3 = 10-2x \implies 20-x = 30-6x \implies x = 2, y = 6$, yielding a value of 8. Thus the optimum value is 8, at $x = 2$, and $y = 6$.

We know that nontrivial constraints were tight at this optimum, so we can find multipliers for them that lead the sum of the left hand side to be exactly $x+y$. These are $1/5$ and $2/5$ (which I personally found by multiplying the second by two so that I had the same number of x and y, and then dividing by 5 to normalize).

(c) Start from $(0,0)$ and perform one step of simplex in LP_1. What are the new equations? Since both x and y have positive coefficients in the objective function, we can choose either one to increase.

Choosing to increase x, we move to $(5,0)$.

The second constraint is tight so we set $y' = y$ and $x' = -y - 2x + 10$. Solving for x, we have $x = -1/2x' - 1/2y' + 5$. Substituting into LP_1, we get the new LP.

$$\begin{aligned} \max \quad & -1/2x' + 1/2y' + 5 \\ \text{subject to:} \quad & -x' + 5y' \leq 30 \\ & x' + y' \leq 10 \\ & x', y' \geq 0 \end{aligned}$$

Alternatively, choosing to increase y, we move to $(0, 20/3)$.

The first constraint is tight so we set $y' = 20 - 3y - x$ and $x' = x$. Solving for y, we have $y = 20/3 - x'/3 - y'/3$. Substituting into LP_1, we get the new LP.

$$\begin{aligned} \max \quad & 20/3 + 2x'/3 - y'/3 \\ \text{subject to:} \quad & 5x' - y' \leq 10 \\ & x' + y' \leq 20 \\ & x', y' \geq 0 \end{aligned}$$

(d) What is the value of this two-player zero-sum game? The values in the matrix indicate outcome given the corresponding actions by Row and Column; Row is the maximizer and Column is the minimizer. Briefly justify your answer.

		Column: Horse	Column: Dice
Row:	Horse	1	0
	Dice	0	1

Solution: The value is 0.5 at the mixed equilibrium in which both players pick each option with 50%

probability.

(e) You found a way to multiply 4×4 matrices with only M multiplications, and this gives rise to a divide-and-conquer algorithm that is faster than Strassen's algorithm! What is the largest value of M for which this is true? Justify briefly.

Recall that Strassen's algorithm multiplies an $n \times n$ matrix with 7 multiplications of $n/2 \times n/2$ matrices.

Solution: The recurrence for Strassen is $T(n) = 7T(n/2) + n^2$ yielding a runtime of $\Theta(n^{\log_2 7})$. This algorithm's recurrence is $T(n) = MT(n/4) + n^2$, and it has runtime $\Theta(n^{\log_4 M})$. These are the same when $M = 7^2$, so the largest M for which we are faster is $7^2 - 1$.

3. (28 pts.) True/False

Write T or F in the box.
Then, if the statement is true, justify briefly. If the statement is false, give a simple *counterexample.*

(a) ☐ In a Huffman code, if there is a leaf of depth 1 and a leaf of depth 3, then there must also be a leaf of depth 2.

Solution: False, consider the frequencies 0.5, 0.125, 0.125, 0.125, 0.125. This results in a leaf of depth 1 (0.5) and four of depth 3, and none of depth 2.

(b) ☐ In a Huffman code where all letter frequencies are distinct, the third least frequent letter has depth either equal to that of the two least frequent letters, or one less.

Solution: True. If it had a depth less than two less than that of the two least frequent letters, then we could swap it with some other letter at a greater depth, which must exist as a descendant of the sibling of the location where the two least frequent letters merge in the tree, which would result in a more efficient code, contradiction. And if it had a depth greater than the two least frequent letters, we could swap it with one of them.

(c) ☐ In an instance of SET COVER, we are given 100 elements to cover by a family of 10,000 sets. It is possible for the greedy solution to be 6 times the optimum. ($\ln 100 \approx 4.6, \ln 10,000 \approx 9.3$.)

For this question, justify briefly whether you write True or False, rather than giving a counterexample.

Solution: False, because it could be at most $\ln n$ times the optimum, where n is the number of elements. Thus it could be at most 4.6 times the optimum, and 6 is not in this range.

(d) ☐ If a linear program is unbounded, then its dual must be unbounded.

Solution: False, consider the LP: max x such than $-x \leq 1$ and x is nonnegative. The dual is to minimize y such that $-y \geq 1$ and y is nonnegative. The primal is unbounded, and the dual is infeasible.

(e) ☐ If the capacities of a max flow problem are multiples of $\frac{1}{2}$, the value of the optimum flow will also be a multiple of $\frac{1}{2}$.

Solution: True, the bottleneck edge and thus the amount Ford Fulkerson increases the flow by at each stage is a multiple of 1/2.

(f) ☐ If we add an integer $k > 0$ to all capacities of a max-flow problem, the max flow is also increased by exactly k.

Solution: False, consider S to A and S to B with capacity 0, and A to T and B to T with capacity 0. We increase max flow by 2k when we add k to all capacities.

(g) ☐ If we multiply all capacities of a max flow problem by an integer $k > 0$, the max flow is also multiplied by exactly k.

Solution: True, the Ford Fulkerson algorithm would find exactly the same flow, which would now be k times as large.

4. (12 pts.) Supercomputer Scheduling

You must schedule the operation of a supercomputer for the next n days. For each day $i \in 1,\ldots,n$, you know c_i, the number of heptacycles needed that day. The supercomputer must be rebooted after at most k consecutive days of work. It takes one day to reboot the supercomputer, and the jobs for that day are outsourced with a cost of \$1 per heptacycle. Assume that the machine was rebooted at day 0. Find the rebooting schedule that minimizes cost, which is the number of outsourced heptacycles.

For example, if $n = 11$, $k = 3$, and the c_i's are $[7,8,1,8,6,6,3,2,3,6,5]$, the optimum schedule would be to reboot at days 3, 7, and 8, at a total cost of 6.

We'll solve this problem with DP, with the following subproblems: For $i \in 1,\ldots n$, define $Q[i]$ to be the minimum cost possible for scheduling the first i days if the machine is **not** rebooted on day i. Similarly, define $R[i]$ to be the minimum cost if the machine **is** rebooted on day i.

Complete the pseudocode below.

$Q[0] = R[0] =$ _____

for $i = 1,\ldots,n$:

$Q[i] = \min$ _____

$R[i] = \min$ _____

optimum cost = _____

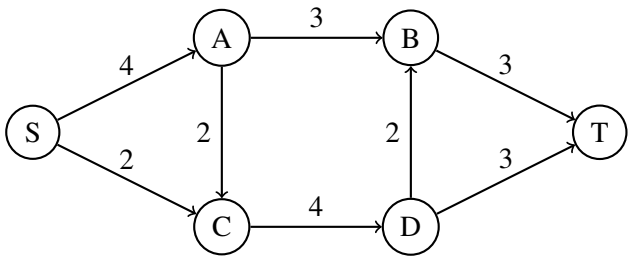
Solution: $Q[0] = R[0] = 0$

$Q[i] = \min R[j]$, where $j \geq 0$ and $i - k \leq j < i$

$R[i] = \min(Q[i-1], R[i-1]) + c_i$

optimum cost $= \min(R[n], Q[n])$

5. (25 pts.) Fantastic Flows



(a) Answer the following questions about the max flow from S to T through the graph above. No justifications are necessary.

Size of max flow:

Find a minimum cut
(specify the two sets of vertices which define it):

Is the minimum cut unique?

Solution: 6

S / ABCDT or SABCD / T

No

(b) Can the maximum flow in the above network be found with three or fewer iterations of the Ford-Fulkerson algorithm? Explain briefly.

Solution: Yes; it could send 3 units along SABT, 2 units along SCDT, and 1 unit along SACDT.

Parts (c) and (d) refer to general max flow networks, not specifically the network on the previous page.

(c) Complete the following sentence (justify briefly, assume all capacities are integers):

Decreasing the capacity of an edge by 1 decreases the flow if and only if the edge belongs to a ...

Solution: min cut. [if] This min cut has a new capacity 1 lower than its previous value, so the corresponding max flow must have also decreased by 1.

[only if] Conversely, if the edge does not belong to a min cut, then all cuts containing this edge have capacity at least 1 greater than the min cut capacity, because all edges are integral. Thus, all cuts containing this edge after its capacity is reduced have capacity at least the min cut, which is therefore unchanged.

(d) Complete the following sentence (justify briefly, assume all capacities are integers):

Increasing the capacity of an edge by 1 increases the flow if and only if the edge belongs to ...

Solution: all minimum cuts. If there is some minimum cut not containing this edge, its capacity and thus the max flow will not change as this edge's capacity is increased by 1. Conversely, if every cut not containing this edge has larger capacity, and every minimum cut, which contains this edge has its capacity increased by 1, then there will be no cut left with capacity at most the current min cut capacity. Thus the min cut's capacity will increase, which will increase the max flow.

(e) Give a polynomial-time algorithm for determining, given a flow network, whether it has a unique minimum cut. You can describe your algorithm without pseudocode if you prefer. Briefly justify correctness and polynomial running time; assume all capacities are integers.

Solution 1: Run Edmonds-Karp to find the max flow and corresponding residual graph. Set min cut A to be the minimum cut found when running BFS on the residual graph, from S. Set min cut B to be the minimum cut found when running BFS on the residual graph with edges reversed, from T. If these are the same, return True, else return False.

This runs in polynomial time because we're running Edmonds-Karp once, and BFS twice, and these all run in polynomial time. It also only takes time proportional to the number of vertices to check if two cuts are the same.

Running a BFS from S finds a set of vertices reachable from S with remaining capacity. This set of vertices exactly represents one side of a minimum cut, as all edges from this set to the rest of the graph (not reachable) must be fully utilized.

When the graph is reversed, and we run a BFS from T, we're finding a set of vertices including T that don't have any capacity to reach them from the rest of the graph. This also represents a minimum cut of the graph.

If these cuts are different, we have found two distinct minimum cuts, so the minimum cut cannot be unique.

Now, notice that any vertex V on S's side in min cut A must be on S's side in any minimum cut. There was a S-V path with excess capacity in it (hence V was reachable from S), so any cut that separates S and V would have excess capacity and thus not be a minimum cut.

Similarly, any vertex W on T's side in min cut B must be on T's side in any minimum cut. If min cut A and min cut B are the same, then the set of vertices that must be on S's side or T's side is the entire set of vertices. Thus there is only one possible cut.

Solution 2: Similar idea to Solution 1.
Find max flow. Let G be the residual graph.
Find S, the set of nodes reachable from the source s in G.
Find T, the set of nodes reachable from the source t in G^R.
If and only if $S \cup T = V$, there is a unique minimum cut.

Proof of correctness and running time are essentially the same as solution 1.

Solution 3: This solution uses part (d): increasing the capacity of an edge by 1 increases the flow iff the edge belongs to all minimum cuts.

The algorithm is: run Edmonds-Karp to determine the max flow value F. Also, find a min cut A by running a BFS from S and finding the set of all vertices reachable from S. Then, for each edge, increase its capacity by 1 and rerun Edmonds-Karp; if the flow value is $F+1$, add this edge to some set M.

Now, if M is the same set of edges that crosses min cut A, return True, else return False.

This runs in polynomial time because we're doing one run of Edmunds-Karp for each vertex, and Edmunds-Karp runs in polynomial time in the size of the graph.

Imagine M is the same set of edges that cross min cut A, but there were some other min cut. Well, this min cut would have to exclude one of the edges that cross min cut A, and then that edge would not be in M. Thus we have a contradiction, so we know that if M is the same set of edges that cross min cut A, this is a unique minimum cut.

Imagine M is a different set of edges than those that cross min cut A. If min cut A were unique, each edge that crosses min cut A would be part of all (1) minimum cuts, so it would be in M, and no other

edge would be part of any minimum cuts. Thus in this case, min cut A cannot be unique.

Blank Space

You can use this extra space for scratch work or continuing to answer a question.