Second Midterm

Name:

SID:

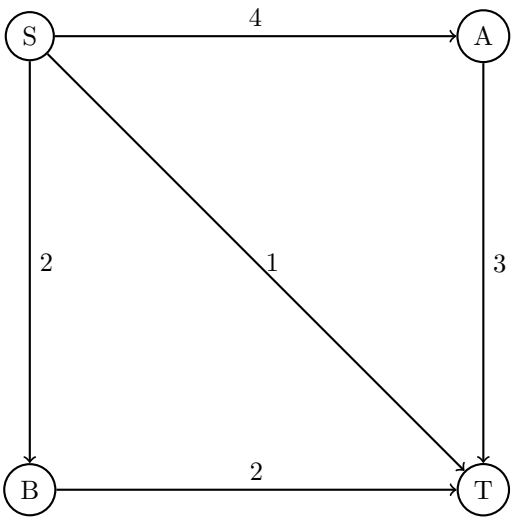
GSI and section time:

Answer all questions. Read them carefully first. Be precise and concise. The number of points indicates the amount of time (in minutes) each problem is worth spending. Not all parts of a problem are weighted equally. Write in the space provided, and use the back of the page for scratch. Box numerical final answers. Good luck!

Page 2	
Page 3	
Page 4	
Page 5	
Page 6	
Page 7	
Page 8	
Page 9	
Page 10	
Total	

1 When an explanation is required, answer with one or two short sentences. (30 points)

1. In the network below, mark all minimum S-T cuts.

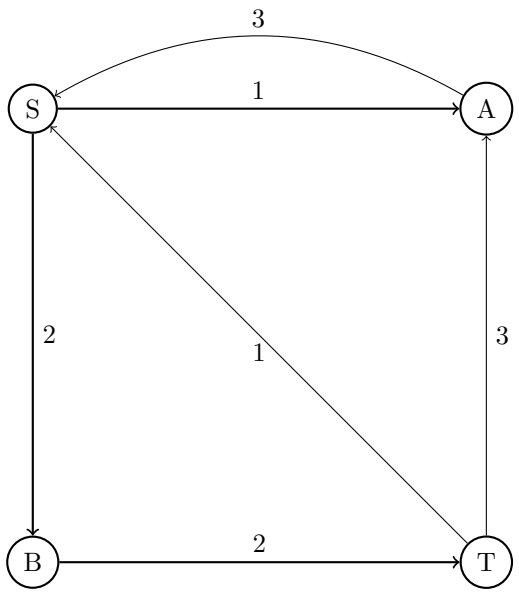


   The min ST cuts are $\{S, A, B\}$ and $\{S, A\}$. This cut has a value of 6.

2. Is there a flow of value 8? Justify your answer.

   Solution: No, since the cut is an upper bound on the value of the flow, so the maximum flow is bounded above by 6.

3. You are in the middle of running the max flow algorithm presented in class on the network above. You have increased the flow along two paths, first SAT and then ST. Draw the residual network.



4. Complete this sentence, and justify it briefly:

   In a network, increasing the capacity of an edge AB by 1 results in increasing the maximum flow if and only if... Edge AB is a bottleneck edge.

   Solution: This answer can be justified in one of these ways:

   - An edge (A, B) such that $A \in S$, $B \in T$, where S is the set of nodes reachable from s in the residual graph, and T is the set of nodes that can reach t in the final residual graph.
   - In the residual graph, there is an $s - A$ path and a $B - t$ path.
   - An edge (A, B) that is the only saturated edge in some $s - t$ path in the final residual graph.
   - An edge (A, B) that crosses all min cuts.

   Common mistakes:

   - Claiming that it suffices for (A, B) to be the unique lowest-capacity edge in some path.
   - Claiming that it suffices for (A, B) to be in some min cut, or "the" min cut.

5. We are running Multiplicative Weights with $n = 2$ experts for $T = 10,000$ days, and we choose $\epsilon = 0.01$ Is this close to the optimum choice? (Hint: $\sqrt{\ln 2} \approx .8325$)

   Solution: We need $\sqrt{\frac{\ln n}{T}} \leq \epsilon$. Plugging in what we know, we have $(\sqrt{\ln 2}/100) \approx .008325$. This is close enough (we also accepted this is not close enough).

6. In all of the first 140 days, Expert 1 has cost 0 and Expert 2 has cost 1. In the next day, with what probability will you play Expert 1? (Hint: You can assume that $0.99^{70} = \frac{1}{2}$)

   Solution: The weight assigned to expert 1 is $.99^{0 \cdot 140} = 1$, while the weight assigned to expert 2 is $.99^{1 \cdot 140} \approx 1/4$. So, the probability we play expert 1 is $\frac{1}{1+1/4} = 4/5$.

   Some partial credit was given for answers of 1/5 (if you switched the weights on experts 1 and 2), and also 1/4 or 3/4, if you had only found the weight of one of the experts.

7. Which of the following algorithms can be parallelized satisfactorily, and which cannot? No explanation required.

   - depth-first search **No** The recursive calls for DFS aren't independent, so the way it is designed currently does not allow it to be parallelized (there does exists modifications to DFS that can be parallelized, but this question was not asking about modifications).
   - $O(n^3)$ matrix multiplication **Yes** We saw this in homework.
   - Strassen's matrix multiplication **Yes** We can do the recursive calls in parallel.
   - FFT **Yes** We can do the recursive calls in parallel.
   - Kruskal's algorithm **No** The way it is designed currently does not allow parallelism
   - Boruvka's algorithm **Yes** It was designed to be parallel version of Kruskals
   - Simplex **Yes** We can parallelize the steps "find the constraint that we hit first". We also gave half credit to false answers, since the depth of the parallel algorithm for simplex is still exponential.

8. We can solve a problem in parallel time $10\log^2 n$ and work $20n^2 \log n$ using n^2 processors. Suppose that we have only $10n$ processors. What parallel time can we achieve now for the same problem? Assume that $n \geq 100$.

   Solution: According to Brent's principle we have $D' = D + W/P$, and plugging our variables in, we get $10\log^2 n = D + (20n^2 \log n)/n^2$, so $D = 10\log^2 n - 20\log n$. So, with $10n$ processors, we will get parallel time $10\log^2 n - 20\log n + (20n^2 \log n)/(10n) = 10\log^2 n - 20\log n + 2n\log n$.

2 Linear programming (20 points)

Your linear program is this:

$$\begin{aligned} &\max -2x_1 + x_2 \\ &x_2 \leq x_1 + 1 \\ &x_1 + x_2 \leq 5 \\ &x_1, x_2 \geq 0 \end{aligned} \tag{1}$$

(a) What is the dual?

Solution: The dual is:

$$\begin{aligned} &\min y_1 + 5y_2 \\ &-y_1 + y_2 \geq -2 \\ &y_1 + y_2 \geq 1 \\ &y_1, y_2 \geq 0 \end{aligned} \tag{2}$$

(b) You run simplex starting from $(0, 0)$. What is the situation after one iteration?

Solution: We first choose a variable with a positive coefficient in the objective function. The only choice we have in this case is x_2. We move x_2 as much as we can until we hit a tight constraint. Here, we can only move x_2 to one before we hit the first constraint. So, we move to the point $(0, 1)$. Now, let $z_1 = x_1 + 1 - x_2$, and write the LP in terms of z_1, x_1. We get

$$\begin{aligned} &\max -x_1 - z_1 + 1 \\ &z_1 \geq 0 \\ &2x_1 - z_1 \leq 4 \\ &x_1 \geq 0 \\ &x_1 + 1 - z_1 \geq 0 \end{aligned} \tag{3}$$

(c) What is the optimum of this LP? What are the multipliers that prove optimality?

Solution: We see from 2(b) that the simplex algorithm reaches the optimum within one iteration, and the optimum value is 1. Furthermore, by looking at the coefficients in the objective function after one iteration, we infer the multipliers to be $y_1 = 1, y_2 = 0$.

(d) Suppose that the first constraint becomes an equality: $x_2 = x_1 + 1$. Would the optimum change? Would the dual linear program change?

Solution: The optimum will not change. The dual linear program would change, in that we will drop the non-negativity constraint $y_1 \geq 0$.

(e) Is $(\frac{1}{2}, \frac{1}{2})$ the maxmin play for Row in this zero-sum game? Justify briefly.

$$\begin{matrix} 4 & 2 \\ 3 & 5 \end{matrix}$$

Solution: Yes. Here are some possible answers

- Use duality and get a strategy of $(3/4, 1/4)$ for the column player. Both ways give a value of $7/2$ for their respective LP, so must be optimal.
- Graph $4x + 3(1 - x)$ and $2x + 5(1 - x)$ on $[0, 1]$, and show that the maxmin is highest at the intersection point.

Common wrong solutions. Most students claimed that since $1/2(4) + (1/2)3 = 1/2(2) + 1/2(5)$, so the column player can't exploit this strategy. However, this doesn't necessarily mean it is the optimal strategy for the row player. A counter example is in the next part, where the column sums are equal, but the maxmin play for row is to always choose the bottom row.

(f) What is the maxmin of this game? Justify briefly.

$$\begin{matrix} 3 & 2 \\ 4 & 5 \end{matrix}$$

Solution: In this case, it is always optimal for the row player to choose the bottom row (since in each corresponding column, the value of the bottom row is larger). The column player knows this, so she will always choose the first column. Thus, the maxmin value is 4.

3 True or false? Circle the right answer. No explanation needed (25 points)

(No points will be subtracted for wrong answers, so guess all you want!)

T **F** 1. The solution of $T(n) = 6T(\frac{n}{2}) + O(n)$ is $T(n) = O(n^3)$.

True The solution is $O(n^{\log_2(6)}) \in O(n^3)$.

T **F** 2. Multiplying all weights by two does not change the shortest paths in a graph.

True Every single path in this graph will be scaled by a factor of 2, so the shortest path will remain the same.

T **F** 3. Multiplying all weights by two and adding one does not change the shortest paths in a graph.

False This is not true, as shown by the counterexample with 5 nodes A, B, C, D, E, where we have a path $A \to B \to C \to D \to E$, each of length 1, and a path $A \to E$ with weight 5. Then, the shortest path from A to E is through the first path. However, after the transformation, the first path will have cost 12, while the second path will have cost 11.

T **F** 4. We are given an array $A[1...n]$ and we want to find an i such that precisely $\lfloor \frac{n}{4} \rfloor$ elements of A are smaller than $A[i]$. $\Omega(n \log n)$ time is required for this.

False We can do quickselect in $O(n)$ time deterministically.

T **F** 5. We are given an array $A[1...n]$ and we want to find if there are $i \neq j$ such that $A[i] = A[j]$. $\Omega(n \log n)$ time is required for this.

True Under the comparison based model, this is true (similar to the proof of lower bounds for sorting). For a non comparison based model, we can use hashing, but in the worst case, where we have $n-1$ distinct integers in our array, each integer needs at least $O(\log n)$ bits to represent, so hashing will take $O(\log n)$ time per hash.

T **F** 6. The greedy algorithm for the Set Cover problem always returns the optimum solution.

False A counterexample is shown here. $S = \{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{1, 3, 5\}, \{1, 2, 4, 6, 8\}\}$. Here, we can cover everything by choosing the first two sets. The greedy will first choose the last set, the the third set, and finally the second set.

T **F** 7. The Dynamic Programming algorithm for the edit distance problem between two strings of length n takes $\Theta(n^3)$ time.

False We know of an algorithm to solve the edit distance problem only take $O(n^2)$ time.

T **F** 8. The Dynamic Programming algorithm for the polygon triangulation problem takes $\Theta(n^3)$ time.

True The book has a solution that has n^2 subproblems, each of which takes n time to evaluate.

T **F** 9. The Dynamic Programming algorithm for the edit distance problem between two strings of length n can be done in $\Theta(n)$ space.

True We only need to keep track of one row/column at a time as we go through the algorithm.

T **F** 10. The Dynamic Programming algorithm for the Knapsack problem with repetitions is a polynomial-time algorithm.

False The dynamic programming algorithm for knapsack takes exponential time (in terms of the number of input bits)

T **F** 11. The Dynamic Programming algorithm for the Traveling Salesman problem takes $n!$ time.

False The dynamic program only takes $O(2^n n^2)$ time.

T **F** 12. If a linear program has an optimum solution, then there is a vertex that is optimum.

True This is mentioned in the book.

T **F** 13. If a linear program has an optimum solution, then a feasible solution that is not a vertex cannot be optimum.

False It is possible for a feasible solution that is not a vertex to be optimum (as in the case where the objective function is zero).

T **F** 14. Simplex may perform an exponential number of iterations.

True This was also mentioned in the book.

T **F** 15. Simplex may spend exponential time on an iteration.

False Each iteration takes at most $O(nm)$ time, which is polynomial time.

T **F** 16. Simplex is a polynomial-time algorithm.

False Since simplex can take an exponential number of iterations, it is not a polynomial-time algorithm.

T **F** 17. There is no known polynomial-time algorithm for linear programming.

False Other methods like the ellipsoid method or interior point can be shown to take polynomial time.

4 Dynamic Programming (15 points)

You are given a tree with *(positive or negative)* weights on the edges, $w(u, v)$. A *chunk* is a connected subgraph of this tree. You want to find *the weight of* the chunk that has the largest total weight. You solve this problem by dynamic programming.

Fill the ...s, and justify your answers briefly.

As always with Dynamic Programming on trees, you root the tree at one of its nodes, say R, and you process the nodes bottom-up, starting from the leaves. For every node v you compute two things:

(a) $W^+(v)$ is the total weight of the heaviest chunk *containing* v in the subtree with root v;

(b) $W^-(v)$ is the total weight of the heaviest chunk *not containing* v in the subtree with root v.

If v is a leaf,

$$W^+(v) = 0 \qquad W^-(v) = 0$$

For these two cases, the connected subgraph will contain no edges, so the weight of a chunk will be zero.

If v is not a leaf, let $c_1, \ldots, c_k$ be the children of v in the rooted tree.

$$W^-(v) = \max_{1 \le i \le k} \max(W^+(c_i), W^-(c_i))$$
$$W^+(v) = \sum_{1 \le i \le k} \max(0, W^+(c_i) + w(v, c_i))$$

For the $-$ case, we know that our chunk must be fully contained within a single subtree of one of our children. So, we just take the best choice.

For the $+$ case, after we include node v, we can choose to either include each child c_i or not independently. The max with zero is there to give us the option of not taking a child.

The final answer is... $\max(W^+(R), W^-(R))$, where R is the root of our tree.

If we also wanted to output the nodes of the heaviest chunk, how would you go about doing this? Describe informally and briefly.

Solution: We can keep backpointers in our dp table to see which subproblems allowed us to get the optimal solution. More specifically, for each $W^-(r)$, we can keep track of the child that maximizes $\max(W^+(c), W^-(c))$. For each $W^+(r)$, we can keep track of the list of children whose $W^+(c) + w(r, c)$ values are positive. Then, to reconstruct the chunk, we look at our root. If $W^-(R) > W^+(R)$, then recursively get the largest chunk in the child that we point to in our backpointer for $W^-(R)$. Otherwise, include the root, and recursively include all the children that are listed in the backpointer for $W^+(r)$.

Why are we computing two quantities in this DP? Do you think that it could be done with one? Comment very briefly.

Solution: We can compute the final answer with just W^+. W^- is only here for convenience so our final answer calculation is constant time. But, we can also extract the final answer with only W^+ by just iterating through all nodes and choosing the best one.