CS 124 Homework 6: Spring 2021

Your name: Sean Ty

Collaborators:

No. of late days used on previous psets: 1

No. of late days used after including this pset: 2

Homework is due Wednesday 2021-04-21 at 11:59pm ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignment must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style will count in your grades. Be sure to read and know the collaboration policy in the course syllabus. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

For all homework problems where you are asked to design or give an algorithm, you must prove the correctness of your algorithm and prove the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

1. **(15 points)** My (Professor Mitzenmacher's) RSA public key (n,e) is:

$$(46947848749720430529628739081, 37267486263679235062064536973).$$

Convert the message

I want an A

into a number, using ASCII in the natural way. (So for "A b": in ASCII, A = 65, space = 32, and b = 98; translating each number into 8 bits gives "A b" = 010000010010000001100010 in binary.) Encode the message as though you were sending it to me using my RSA key, and write for me the corresponding encoded message in decimal.

The corresponding ASCII values in base 10 are: 73 32 119 97 110 116 32 97 110 32 65. In base 2 (as 8 bits), these values are 01001001, 00100000, 01110111, 01100001, 01101110, 01110100, 00100000, 01100001, 01101110, 00100000, 01000001. Concatenating and converting to base 10 yields 88404902742541081376596033.

Raising this to e and taking the result mod n then yields 27046002102094147030583467156, so this is what will be sent.

2. **(15 points)** We have considered, in the context of a randomized algorithm for 2SAT, a random walk with a completely reflecting boundary at 0—that is, whenever position 0 is reached, with probability 1 we move to position 1 at the next turn. Consider now a random walk with a partially reflecting boundary at 0– whenever position 0 is reached, with probability 1/4 we move to position 1 at the next turn, and with probability 3/4 we stay at 0. Everywhere else the random walk either moves up or down 1, each with probability 1/2.

   Find the expected number of moves to reach n starting from position i using a random walk with a partially reflecting boundary. (You may assume there is a unique solution to this problem, as a function of n and i; you should prove that your solution satisfies the appropriate recurrence.)

   For each $0 \le i \le n$, let $T(i)$ denote the expected number of moves needed to reach n when we start from position i with the rules in the problem. Then we obtain the recursion equations

$$T(i) = \frac{1}{2}T(i+1) + \frac{1}{2}T(i-1) + 1,$$

   for $1 \le i \le n-1$. Now evidently we have $T(n) = 0$, and from the partial reflecting boundary condition, we get $T(0) = \frac{1}{4}T(1) + \frac{3}{4}T(0) + 1 \implies T(0) = T(1) + 4$. Now let $S(i) = T(i+1) - T(i)$ for $0 \le i \le n-1$. Then from the equations we obtain $T(i+1) - T(i) = T(i) - T(i-1) - 2$, and so $S(i) = S(i-1) - 2$ for $1 \le i \le n-1$. Note also that we get $S(0) = -4$ from the third equation, and so we get $S(i) = -2i - 4$ by inducting on i (base case is trivial, and if we have $S(i-1) = -2i - 2$ then $S(i) = S(i-1) - 2 = -2i - 4$ as desired), and so $T(i+1) - T(i) = -2i - 4$ for all $0 \le i \le n-1$. Now since $T(n) = 0$, for any $0 \le i \le n-1$ we obtain

$$T(i) = T(i+1) + 2(i+2) = T(i+2) + 2(i+2) + 2(i+3) = \cdots = T(n) + \sum_{j=i+2}^{n+1} 2j = (n+i+3)(n-i),$$

   where we got the last step from pairing up the first kth term with the last kth for each k and summing them up.

   So it suffices to check that this satisfies the recurrence. Indeed, we have

$$\begin{aligned}(n+i+4)(n-i-1) + (n+i+2)(n-i+1) + 2 &= 2(n^2 - i^2) + 4(n-i) - (n+i) + (n+i) + 2(n-i) + 2 \\ &= 2(n^2 - i^2) + 6(n-i) = 2(n+i+3)(n-i)\end{aligned}$$

   which proves that $T(i) = T(i+1)/2 + T(i-1)/2 + 1$, and we have $(n+3)n = (n+4)(n-1) + 4$ so $T(0) = T(1) + 4$. and clearly we have $T(n) = 0$ so this solution works.

3. **(0 points, optional)**[1] You have been given a square plot of land that has been divided into n rows and columns, yielding n^2 square subplots. Some of these subplots have rocky ground and cannot support plant growth, while others have soil and can support growing a palm tree. You would like to plant palm trees on a subset of the square subplots (at most one palm tree per subplot) so that every row and every column has exactly the same number p of palm trees. Furthermore, you would like to do this so that p is as large as possible. Devise an efficient algorithm to determine how to accomplish this. (You may give the running time in terms of the time to solve a suitable flow problem.)

[1]We won't use this question for grades. Try it if you're interested. It may be used for recommendations/TF hiring.

4. Show how to reduce the following problems to linear programming.

   - **(15 points)** Find the maximum flow from a vertex s to a vertex t in a directed graph with edge capacities—except that, at each vertex, half the flow into the vertex is lost (or kept) at the vertex, and the other half flows out. The goal is to maximize the flow that reaches the destination t.

     First note that the flow between two nodes A, B must be at most the weight of the capacity from $A \rightarrow B$ (or edge weight). Thus, we have the constraint $f_{AB} \leq c_{AB}$. Next, for every vertex A that is not a source or a sink, the total flow into A must equal the total flow out from A. That is, we have $\sum_{B \neq A} f_{BA} = 2 \sum_{B \neq A} f_{AB}$. Finally we wish to maximize the flow that reaches t, or $\sum_{t' \neq t} f_{t't}$.

   - **(15 points)** Find the maximum flow from a vertex s to a vertex t in a directed graph with edge capacities—except that, for each edge e, there is also a fixed cost c_e for each unit of flow through the edge. We need to find the maximum flow with the minimum cost. That is, there may be many possible flows that achieve the maximum flow; if there is more than one such flow, find the one of minimum cost. (Hint: you may need to use more than one linear program!)

     We first find the maximum possible flow with traditional LP– that is, we first solve the LP with the constraints $f_{AB} \leq c_{AB}$ (edge weight $a \rightarrow b$, or capacity) nonnegative, $\sum_{B \neq A} f_{BA} = \sum_{B \neq A} f_{AB}$ for all A not source or sink, and we want to maximize flow into the sink, or $\sum_{t' \neq t} f_{t't}$.

     This gives us an maximum M. Next, solve the LP with the same constraints but with the flow from s summing to M and this time we wish to minimize the total cost. That is, if we let s_{AB} be the flow we're using from A to B, then we have the conditions $s_{AB} \leq c_{AB}$ nonnegative, $\sum_{B \neq A} s_{BA} = \sum_{B \neq A} s_{AB}$ for all A not source or sink, $\sum_{t' \neq t} s_{t't} = M$, and we wish to minimize $\sum_{e \in E} c_e s_e$ where E is the set of all edges in the graph. This quantity is the total cost given the flows we use. The values obtained from this will be the desired answer.

5. **(15 points)** Consider the two-player zero-sum game given by the following matrix. (A positive payoff goes to the row player: e.g. if the row player picks the first row and the column player picks the first column, the row player scores 3 and the column player scores -3.)

$$\begin{bmatrix} 3 & 1 & 0 & -4 \\ 6 & -2 & -2 & 0 \\ -3 & 2 & 3 & -3 \\ -7 & 4 & -5 & 7 \end{bmatrix}$$

- Write down a linear program to determine the row player strategy that maximizes the value of the game to the row player, in terms of the probabilities c_1, c_2, c_3, c_4 that the column player picks column 1, 2, 3, 4. Do the same for the column player.

  Say that r_i is the probability that the row player chooses row i (so (r_1, r_2, r_3, r_4) is his strategy), so $r_1 + r_2 + r_3 + r_4 = 1$. Note that we can have $r_i = 1$ for some i–this would just be equivalent to row playing a pure strategy instead of a mixed one. Then note that if he chooses row i, his expected value is $\sum_{i=1}^4 c_i a_i$, where a_i are the respective column values in this row. Thus, by linearity of expectation, he wants to maximize his expected value which is $\sum_{i=1}^4 \left(r_i \sum_{j=1}^4 M_{ij} c_j\right)$, where M_{ij} is the ij entry (row, column) in the matrix. This is the desired linear program.

  We can solve this using an online LP solver, or we can cheat and solve it with "quick mafs." Indeed, let $e_i = \sum_{j=1}^4 M_{ij} c_j$ for $1 \le i \le 4$. Then row player wishes to maximize $S = \sum_{i=1}^4 r_i e_i$ subject to $\sum_{i=1}^4 r_i = 1$. Now if $M = \max_{1 \le i \le 4} e_i$, then we have $S \le \sum_{i=1}^4 r_i M = M$ with equality iff $r_k = 1$ for $k = \text{argmax}_{1 \le i \le 4} e_i$ and $r_k = 0$ otherwise, so row player simply chooses the row with the highest expected value, which is a pure strategy.

  Likewise, for the column player we suppose that the row player plays row i with probability r_i. Then we have $r_1 + r_2 + r_3 + r_4$ and we wish to solve for (c_1, c_2, c_3, c_4) where this will be the column player's strategy. Thus, we similarly have the following linear program: we wish to minimize $S = \sum_{i=1}^4 c_i e_i$ subject to $\sum_{i=1}^4 c_i = 1$, where $e_j = \sum_{i=1}^4 M_{ij} r_i$. Now we can again cheat by noting that if $m = \min_{1 \le i \le 4} e_i$ then we have $S \ge \sum_{j=1}^4 c_j M = M$ with equality iff $c_k = 1$ for $k = \text{argmin}_{1 \le i \le 4} e_i$ and $r_k = 0$ otherwise.

- Solve these linear programs, and give the proper strategies for both players. (You'll probably want to find an existing LP solver.)

  (According to Ed we are assuming neither knows the other's strategy). In this case, according to the notes, row's LP would be to maximize s, the lowest possible expected value for row. Let r_i be the probabilities for row to choose row i. Then evidently $r_i \ge 0$ always, and $\sum_{i=1}^4 r_i = 1$. Next, since s is the lowest possible, it follows that we have the constraints

$$s - 3r_1 - 6r_2 + 3r_3 + 7r_4 \le 0,$$

$$s - r_1 + 2r_2 - 2r_3 - 4r_4 \le 0,$$

$$s + 2r_2 - 3r_3 + 5r_5 \le 0,$$

$$s + 4r_1 + 3r_3 - 7r_4 \le 0.$$

  Similarly, column player's LP would be to maximize s given the constraints $c_i \ge 0$, $\sum_{i=1}^4 c_i = 1$, and

$$s + 3c_1 + c_2 - 4c_4 \le 0,$$

$$s + 6c_1 - 2c_2 - 2c_3 \le 0,$$

$$s - 3c_1 + 2c_2 + 3c_3 - 3c_4 \le 0,$$

$$s - 7c_1 + 4c_2 - 5c_3 + 7c_4 \leq 0.$$

We used the LP solver from this link to get that row player's best value is -0.218 which is achieved when $(r_1, r_2, r_3, r_4) = (0, 0.382, 0.455, 0.164)$. That is, row player should choose row 1, 2, 3, 4 with probabilities 0, 0.382, 0.455, 0.164 respectively.

Column player's best value is 0.218 which is achieved when $(c_1, c_2, c_3, c_4) = (0.118, 0, 0.464, 0.418)$. That is, column player should choose column 1, 2, 3, 4 with probabilities 0.118, 0, 0.464, 0.418 respectively.

- What is the value of the game? That is, should the column player pay the row player to play, or vice versa, and how much should one player pay the other to make the game fair?

  Since the best values for row, column players are -0.218 and 0.218 respectively, the value of the game is 0.218. Since column has an advantage, column should pay row 0.218 to make the game fair, so by linearity of expectation the expected best payoffs are zero for both.

6. **(15 points)** For the Harvard Tutoring Club, there are n high school students who need tutors, and n available tutors in the club to tutor them. Each student needs a tutor, and each tutor can work with at most one student. Each high school student looks at the tutors and decides on a subset of them that they would want to work with. Suppose that for any subset S of the students, the collective set of tutors $T(S)$ that they are willing to work with satisfies the condition $|T(S)| \geq |S|$. Prove that there is a way to assign every student a tutor that they are willing to work with. Hint: think of this as a flow problem.

   Consider a bipartite graph with the set of n students on the left as vertices and the set of n tutors in the right as vertices. Further, we connect a directed edge with capacity ∞ between a student to a tutor if they are willing to work with each other. Now consider adding a source vertex `Groudon` to the left of the student vertices and connect `Groudon` to each student vertex with a directed edge with capacity 1. Also, consider adding a sink vertex `Kyogre` to the right of the tutor vertices and connect each tutor to it with a directed edge with capacity 1. Note that it suffices to prove that the max flow from `Groudon` to `Kyogre` is n. By the max flow min cut theorem, it suffices to prove that the min cut has value n. Note that the min cut is at least n since we can just cut the edges from `Groudon` to all students, and we have n as the total weight.

   Now consider a cut in the flow network. Since the edges of the bipartite subgraph have capacity ∞, it follows that we should only cut off edges from `Groudon` or to `Kyogre`. Suppose that we performed this cut and after this, have `Groudon` pointing to only $T_1 \subseteq A$ and only $T_2 \subseteq B$ point to `Kyogre`. Further, let $S_1 = A - T_1$ and $S_2 = B - T_2$. Since we performed a cut, it follows that there is no edge from T_1 to any vertex in T_2, and so all edges G from T_1 must end up in S_2.[2]

   Now assume for the sake of contradiction that this cut has size less than n (if it didn't, then we would get max flow is n so the conclusion is true). Then we necessarily have $|S_1| + |S_2| \leq n-1$. So from $|S_1| + |T_1| = n$, it follows that $|S_1| + |S_2| < |S_1| + |T_1| \implies |S_2| < |T_1|$, but this is a contradiction since from earlier, we know that $T(T_1) \subseteq S_2$ and so this would give $|T(T_1)| \leq |S_2| < |T_1|$, contradicting the hypothesis in the problem.

   This argument works because we know from lecture that since the capacities are all integers, the optimal flow is also an integer.

[2]Or you know, otherwise there would be a flow from `Groudon` to `Kyogre` then this wouldn't be a cut since we'd need `Rayquaza` to save the day!
My apologies, I hope this makes grading less of a bore and writing less of a chore :) don't dock plz