CS 124: Problem Set 6

Chao Cheng
Collaborators: Hao Wang

February 4, 2021

Problem 1. Let $m(i)$ be the expected number of moves to reach n starting from i. At any given position i, $m(i)$ will depend on $m(i+1)$ and $m(i-1)$, with $\frac{1}{2}$ probability of moving to i from $i+1$ and $\frac{1}{2}$ probability of moving from $i+1$. Since it takes one move from either direction, we have that $m(i) = \frac{m(i+1)}{2} + \frac{m(i-1)}{2} + 1$.

However, since there is a boundary at $i = 0$, we have that $m(0) = \frac{m(0)}{2} + \frac{m(1)}{2} + 1$. Solving this recursion gives $m(0) = m(1) + 2$. Finally, if $i = n$ then we are done, and $m(n) = 0$. Thus, we have the following recursion:

$$m(i) = \begin{cases} 0 & i = n \\ \frac{m(i+1)}{2} + \frac{m(i-1)}{2} + 1 & 1 \le i < n \\ m(1) + 2 & i = 0 \end{cases}$$

From manually calculating $m(i)$ for the first few i, suppose that $m(i) = n^2 + n - i^2 - i$. We will prove that this proposition works for each case

The first case is $m(n)$. From the recursion, we have $m(n) = 0$, and from the proposition we have $m(n) = n^2 + n - n^2 - n = 0$, so this case works.

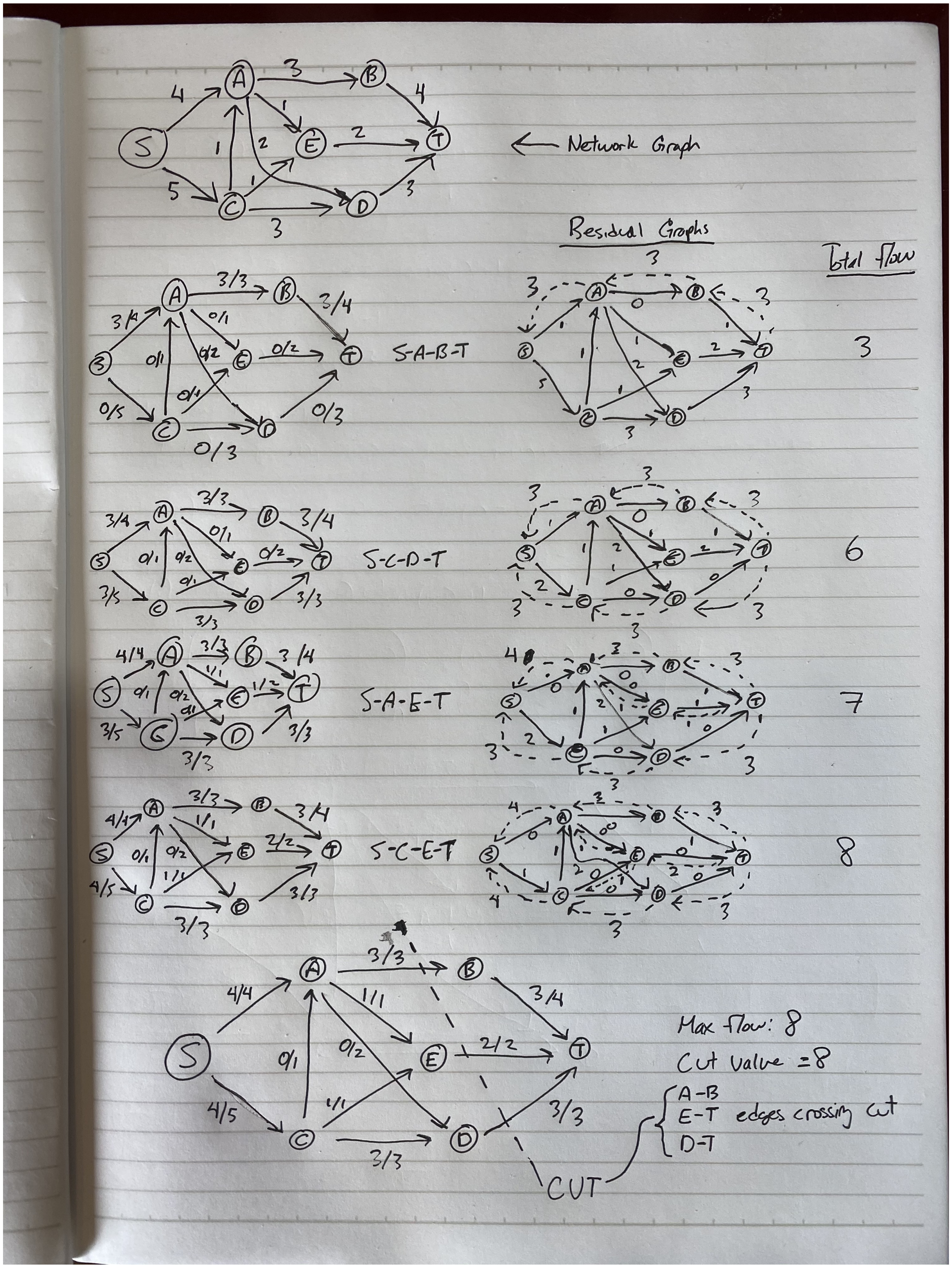
The next case is $m(0)$. We have $m(0) = n^2 + n$, and from the recursion we also have $m(0) = m(1) + 2 = n^2 + n - 1 - 1 + 2 = n^2 + n$, so $i = 0$ holds as well.

Then for i between 0 and n we have from the recursion that

$$\begin{aligned} m(i) &= \frac{m(i+1)}{2} + \frac{m(i-1)}{2} + 1 \\ &= \frac{n^2 + n - (i+1)^2 - (i+1)}{2} + \frac{n^2 + n - (i-1)^2 - (i-1)}{2} + 1 \\ &= n^2 + n - \frac{i^2 + 2i + 1}{2} - \frac{i+1}{2} - \frac{i^2 - 2i + 1}{2} - \frac{i-1}{2} + 1 \\ &= n^2 + n - i^2 - 1 - i + 1 \\ &= n^2 + n - i^2 - i \end{aligned}$$

so the proposition is correct. Thus, the proposition holds for all three cases of the recursion, and therefore $m(i) = n^2 + n - i^2 - i$.

Problem 2. Image attached below.



Problem 3. This problem can be turned into a network flow problem as follows. Let the land be represented by an $n \times n$ matrix. First, determine the row or column that can support the smallest number of palm trees by iterating over the matrix. Suppose this number is p; then any row or column can support at least p palm trees, and the optimal solution to the problem is a configuration in which each row and each column has exactly p palm trees. If p is any larger, then there will be some row or column that cannot support that many palm trees, so this is the optimal solution because p is as large as possible.

We can construct a graph as follows. We begin with a source node S, n nodes $R_1, R_2, \dots R_n$ representing the rows, n nodes $C_1, C_2, \dots C_n$ representing the columns, and a sink node T. We draw an edge with capacity p from S to each R_i, and an edge with capacity p from each C_i to T. Finally, we draw an edge with capacity 1 between R_i and C_j if the i, j-th position in the matrix of the land can support a palm tree.

Running the Ford-Fulkerson algorithm on this network will result in an optimal flow configuration with maximum flow from S to T that can be translated into the optimal solution of the original problem by taking any edge between the row nodes and the column nodes that has a flow of 1 going through it, and planting a palm tree at the corresponding position in the land.

Because each node's input and output must be equal, each row node is limited to p output edges because the input edge from the source has capacity p, and similarly, each column node is limited to p input edges since the output edge to the sink has capacity p, fulfilling the requirement that no row or no column has more than p trees planted. Additionally, the total output of all row nodes is equal to the output from S. This output is then distributed to the column nodes, which means that the total input to all column nodes is equal to the total output of all row nodes, so it is also equal to the total input of all row nodes. Finally, each column node outputs to the sink node, and therefore, the input to the sink node must be equal to the output of the source node, which in turn means each row node and each column node has its maximum capacity p flowing through it.

Thus, running Ford-Fulkerson on the network graph will result in finding the maximum flow from the source node to the sink node. Since there are n row nodes and the edge capacity from the source to each row node is p, the maximum output of the source node is np, so the maximum flow of the graph is np. This corresponds to the optimal solution of the original problem, because the optimal solution has a total of np palm trees planted since each row and each column has exactly p palm trees.

The runtime of this algorithm is $O(n^3 p)$ or $O(n^4)$, considering the worst case where p is as large as n. Since we have at $2n + 2$ nodes (vertices) and at most $n^2 + 2n$ edges in the graph, running Ford-Fulkerson using DFS requires $O(Ef^*) = O((n^2 + 2n)(np)) = O(n^3 p)$ time and running it using BFS requires $O(VE^2) = O((2n + 2)(n^2 + 2n)^2) = O(n^5)$ time; thus we use Ford-Fulkerson with DFS. Finally, determining p initially by iterating over the matrix takes $O(n^2)$ time.

Problem 4. (a) Let s be the source vertex and t be the sink vertex. Also, let $f_{i,j}$ denote the flow from vertex i to vertex j, and let $c_{i,j}$ denote the maximum flow capacity from vertex i to vertex j. Then we have the following constraints:

(1) $0 \leq f_{i,j} \leq c_{i,j}$ for any edge (i,j). In other words, the flow between vertices cannot exceed the maximum capacity.

(2) $\displaystyle\sum_{(x,v)\in E} f_{x,v} = 2 \sum_{(v,y)\in E} f_{v,y}$ for any vertex v except for s and t. In other words, the sum of flow entering a vertex must be two times the sum of flow exiting the vertex.

Since we want to maximize the flow from s to t, the objective function is

$$\max\left\{\sum_{(z,t)\in E} f_{z,t}\right\} \qquad \text{which is equivalent to} \qquad \max\left\{\sum_{(s,w)\in E} f_{s,w}\right\}$$

or in other words, the sum of flow entering the sink or the sum of flow exiting the source.

(b) To solve this problem, we use two linear programs.

The first program maximizes the flow through the network, with the following constraints (using similar notation as defined in the previous part):

(1) $0 \leq f_{i,j} \leq c_{i,j}$ for any edge (i,j).

(2) $\displaystyle\sum_{(x,v)\in E} f_{x,v} = \sum_{(v,y)\in E} f_{v,y}$ for any vertex v except for s and t.

The objective function is $\max\left\{\sum_{(s,w)\in E} f_{s,w}\right\}$. Let the result of the first program (i.e. the maximum flow) be denoted as M.

The second program will use the same constraints as the first with an additional constraint that the sum of flow leaving the sink, or the maximum flow, must be equal to M. Thus the constraints are:

(1) $0 \leq f_{i,j} \leq c_{i,j}$ for any edge (i,j).

(2) $\displaystyle\sum_{(x,v)\in E} f_{x,v} = \sum_{(v,y)\in E} f_{v,y}$ for any vertex v except for s and t.

(3) $\displaystyle\sum_{(s,w)\in E} f_{s,w} = M$.

Since we want to minimize the total cost for this network flow, the objective function is $\min\left\{\sum_{e\in E} f_e c_e\right\}$ where f_e is the flow through an edge and c_e is the cost per unit of flow through the edge. Thus, using these two linear programs we can find the maximum flow with the minimum cost.

Problem 5. Suppose the row player adopts the strategy (x_1, x_2, x_3, x_4) where x_i denotes the probability of choosing strategy i, and let z denote the guaranteed expected outcome. Then s/he has the following constraints:

(i) $x_1 + x_2 + x_3 + x_4 = 1$

(ii) $x_1, x_2, x_3, x_4 \geq 0$

(iii) $z \leq 3x_1 + 6x_2 - 3x_3 - 7x_4$

(iv) $z \leq x_1 - 2x_2 - 2x_3 + 4x_4$

(v) $z \leq -2x_2 + 3x_3 - 5x_4$

(vi) $z \leq -4x_1 - 3x_3 + 7x_4$

and their objective function is $\max\{z\}$.

Similarly, suppose the column player adopts strategy (y_1, y_2, y_3, y_4) and let w denote the guaranteed expected outcome. Then s/he has the following constraints:

(i) $y_1 + y_2 + y_3 + y_4 = 1$

(ii) $y_1, y_2, y_3, y_4 \geq 0$

(iii) $w \geq 3y_1 + y_2 - 4y_4$

(iv) $w \geq 6y_1 - 2y_2 - 2y_3$

(v) $w \geq -3y_1 - 2y_2 + 3y_3 - 3y_4$

(vi) $w \geq -7y_1 + 4y_2 - 5y_3 + 7y_4$

and their objective function is $\min\{w\}$.

The results of using a linear programming solver on the above programs are as follows:

$$z = -\frac{224}{583}, \quad x_1 = \frac{8}{53}, \quad x_2 = \frac{161}{583}, \quad x_3 = \frac{221}{583}, \quad x_4 = \frac{113}{583}$$

$$w = -\frac{224}{583}, \quad y_1 = \frac{81}{583}, \quad y_2 = \frac{11}{53}, \quad y_3 = \frac{234}{583}, \quad y_4 = \frac{147}{583}$$

Or, in decimal form:

$$z = -0.384, \quad x_1 = 0.151, \quad x_2 = 0.276, \quad x_3 = 0.379, \quad x_4 = 0.194$$

$$w = -0.384, \quad y_1 = 0.139, \quad y_2 = 0.208, \quad y_3 = 0.401, \quad y_4 = 0.252$$

Thus, the value of the game for the row player is -0.384. Since this is negative (meaning the row player is expected to pay 0.384 to the column player), the column player should pay 0.384 to the row player to make the game fair.