SUN Yilin Algorithms
520030910361 **Homework 5** May 18, 2022

## 1

### 1.1

For any fixed input $x$, every time we run our randomized algorithm $\overline{A}$, the peformance of $\overline{A}$ on $x$ is determined. Say our algorithm $\overline{A}$ on given input $x$ has a probability of $p_i$ to perform as deterministic algorithm $A_i$, then since our given input $x$ is definite and finite, we can always find a set $S = \{A_1, A_2, \dots, A_m\}$ and some corresponding possibilities $p_1, p_2, \dots, p_m$ where $\sum_{i=1}^{m} = 1$, by which the performance of our algorithm $\overline{A}$ belongs to $S$.
So the expected running time of $\overline{A}$ on input $x$ is

$$\mathbb{E}\left[T(\overline{A}, x)\right] = \sum_{i=1}^{m} p_i T(A_i, x)$$

Then we define distribution $\mathscr{A}$ to be

$$A = A_i \quad \text{with probability} \quad p_i, \forall i \in [m]$$

Clearly

$$\mathbb{E}_{A\sim\mathscr{A}}\left[T(A, x)\right] = \sum_{i=1}^{m} p_i T(A_i, x)$$

So for any randomized algorithm $\overline{A}$ we can find a distribution $\mathscr{A}$ over $\mathcal{A}$ to represent the expected running time of $\overline{A}$
As for the expected running time of the best algorithm for $\Pi$, since it is the best algorithm, it must have the minimum expected running time, i.e. the $\mathbb{E}_{A\sim\mathscr{A}}\left[T(A, x)\right]$ for this algorithm must be the smallest. We also want the expected running time to be upper bounded by the worst input distribution, which means that we would like to minimize the running time over the maximum among $x \in \mathcal{X}$. So the overall running time of the best randomized algorithm can be optimized by finding the best distribution $\mathscr{A}$ over $\mathcal{A}$ and the best randomized algorithm should follow the distribution of $\mathscr{A}$. So the expected running time of the best algorithm for $\Pi$ is

$$\min_{\mathscr{A}\sim\mathcal{A}} \max_{x\in\mathcal{X}} \mathbb{E}_{A\sim\mathscr{A}}\left[T(A, x)\right]$$

### 1.2

To use Von Neumann's minimax theorem, we would like to show that the expected running time can be viewed as elements in a matrix.
Imagine that we have two players denoted by the Column and Row players. Our Column player has a set of algorithms to solve a question and our Row player has the set of all

possible inputs for such a question. Denote the running time of specific algorithm $a$ on given input $x$ to be $T(a,x)$ The goal of Column player is to optimize his distribution $\mathscr{A}$ so that for any distribution of inputs given by Row player, the expected running time is smallest. While the goal of Row player is to optimize his distribution $\mathscr{X}$ so that for any distribution of algorithms given by Column player, the expected tunning time is largest.
Any element of the matrix can be expressed as $T(a,x)$ where $T(a,x)$ is the running time of algorithm $a$ on input $x$. So the goal of the Column player can be expressed as

$$\min_{\mathscr{A}\sim\mathcal{A}} \max_{x\in\mathcal{X}} \mathbb{E}_{A\sim\mathscr{A}}\left[T(A,x)\right]$$

And the goal of the Row player can be expressed as

$$\max_{\mathscr{X}\sim\mathcal{X}} \min_{a\in\mathcal{A}} \mathbb{E}_{X\sim\mathscr{X}}\left[T(a,X)\right]$$

Then Von Neumann's minimax theorem says there exists mixed strategies that are optimal for both players and this mixed strategies provide the same optimal value for them, which means that

$$\max_{\mathscr{X}\sim\mathcal{X}} \min_{a\in\mathcal{A}} \mathbb{E}_{X\sim\mathscr{X}}\left[T(a,X)\right] = \min_{\mathscr{A}\sim\mathcal{A}} \max_{x\in\mathcal{X}} \mathbb{E}_{A\sim\mathscr{A}}\left[T(A,x)\right]$$

## 1.3

The worst running time for a deterministic algorithm is $n$ because at the worst case this deterministic algorithm would probe the array $n$ times to find the location we want.
By using randomization we can use a randomized algorithm which do not probe the array in a determined order, i.e. each time this randomized algorithm probe uniform randomly a position $i$ which has not been probed ever to check whether $A[i]=x$. Then the expected cost of this randomized algorithm can be calculated by summing up all the possible deterministic algorithms with probability $\frac{1}{n}$. And that is

$$\left(\sum_{i=1}^{n-1} i + n - 1\right)/n$$

which is

$$\frac{n+1}{2} - \frac{1}{n}$$

## 1.4

To prove that each randomized algorithm for the above problem costs at least $\frac{n+1}{2}-\frac{1}{n}$ in expectation in the worst case, i.e. a lower bound for the expected running time of the best

algorithm, from problem 1.2 we know that we only need to construct a distribution over all inputs of length $n$ and show that every deterministic algorithm performs bad on average when the inputs are drawn from the distribution.
Here we construct the distribution of inputs of length $n$ from the following method:
For any input array $A$, we let the number $x$ the location of which we want to find to be uniformly distributed among the $n$ positions of the array.
This means that on average every deterministic algorithm needs a time of

$$\left(\sum_{i=1}^{n-1} i + n - 1\right)/n = \frac{n+1}{2} - \frac{1}{n}$$

Note that although here we use a same equation as in problem 1.3, there meaning is different. In problem 1.3 the algorithm are randomized whereas here the algorithm is determined, what is random is the inputs. So here we are summing up all the possible inputs and illustrating that any deterministic algorithm will perform bad on average with the inputs drawing from this distribution.
Then by what we have shown in problem 1.2 each randomized algorithm costs at least $\frac{n+1}{2} - \frac{1}{n}$ in expectation in the worst case.
My algorithm in problem 1.3 matches the lower bound.

## 2

### 2.1

We first construct a weighted directed graph $H$ from $G$. Then we can find maxflow on graph $H$ to see whether there is a perfect matching in $G$.
Firstly, we let all the edges in $G$ point from $V_1$ to $V_2$ in $H$, and we assign them a weight of $+\infty$.
Then we add a source vertex $s$ and add an edge with weight 1 from $s$ to all vertices in $V_1$.
Finally we add a sink vertex $t$ and add an edge with weight 1 from all vertices in $V_2$ to $t$.
Now we have constructed the graph $H$, we can run the Network Flow algorithm to find a maxflow of $H$. If the maxflow of $H$, denoted by $f$, turns out to be $f = |V_1| = |V_2|$, then there exists a perfect matching in $G$. And the perfect matching contains exactly the edges in $G$ used by us to construct the maxflow of $H$.

### 2.2

**Proof of Necessity:**
This part is very straightforward.
Suppose there is a perfect matching $M$ from $V_1$ to $V_2$. Then for any $S \subset V_1$, for every vertex $v \in S$, there is an edge in $M$ connecting $v$ to a vertex in $V_2$. This means that there are at least as many vertices in $V_2$ that are neighbors of vertices in $V_1$ as there are vertices in $V_1$.
That is to say, for any $S \subset V_1$, $|N(S)| \geq |S|$.
**Proof of Sufficiency:**
Following the hint, we would like to prove by showing that the mincut of the graph we construct is exactly $|V_1|$(or $|V_2|$ if you like).
Firstly, any mincut can only contain the edges in $H$ which are not in $G$ because we assign edges in $G$ with a weight of $+\infty$.
Then there is a cut with capacity $|V_1|$ if we make our $S$-$T$ cut to be $S = \{s\}$, the singleton. So the capacity of mincut of $H$ is at most $|V_1|$.
Suppose there is another minimum $S$-$T$ cut where $S \backslash V_2 = \{s\} \cup (V_1 \backslash V_1')$ and $T \backslash V_1 = \{t\} \cup (V_2 \backslash V_2')$, since this is a mincut, there are no edges from $V_1 \backslash V_1'$ to $V_2 \backslash V_2'$. This means that all the neighbors of $V_1 \backslash V_1'$ must be in $V_2'$. Then by the property that $|N(S)| \geq |S|$ for any $S \subset V_1$, $|V_1 \backslash V_1'| \leq |V_2'|$.
Finally the capacity of this cut is $|V_1'| + |V_2'|$, $|V_1'| + |V_2'| \geq |V_1'| + |V_1 \backslash V_1'| = |V_1|$. This means that the capacity of any $S$-$T$ cut is at least $|V_1|$. Then by our instance where $S = \{s\}$, the mincut is indeed $|V_1|$. Finally by the Maxflow-Mincut Theorem, the maxflow of $H$ equals its mincut $|V_1|$, which means that there is a perfect matching in $G$.

## 3

We prove by showing that we can construct a new graph $H = (V', E')$ where if $(u, v) \in E'$ then $(u, v) \in E$ (they might have different weight in $G$ and $H$).
We will carry out the following process:
**1**.Ignore the directions of edges, find a cycle $C$ in the graph $G$. If no cycle, go to step **5**.
**2**.Find the edge with smallest weight in $C$ denoted by $(u, v_0) \in E$ with weight $w$.
**3**.Denote the undirected cycle by $C = (u, v_0), (v_0, v_1), (v_1, v_2), \ldots (v_n, u)$, we update the weights of edges in $C$. We delete $(u, v_0)$. If $(v_i, v_{i+1}) \in E$ then $w'(v_i, v_{i+1}) = w(v_i, vi + 1) - w$; if not, $w'(v_i, v_{i+1}) = w(v_i, vi + 1) + w$.
**4**.If there is still undirected cycle in $G$, go to step **1**. If not, go to step **5**
**5**.Now the graph $H = (V', E')$ is our new debt cycle which has no cycle, so it must contain at most $n - 1$ edges.
**Prove that the new graph $H$ is equivalent to $G$:**
In step **3**, we pick out the edge with the smallest weight and update all weights of the other edges in the cycle. In this process we will not add any new edges in the original graph. This is because we pick out the edge with smallest weight and when we modify all the weights of the other edges, they remain their original direction.
Also in this process we are not changing any amount of debt that anybody should repay somebody else. This is because in this process everyone is receiving and giving out exact the same amount of money.
By the above two arguments, by our procudure we can delete at least one edge of the original graph $G$ at each iteration. And finally we can remove all cycles in the original graph.
Finally in the new graph $G$ there is no cycle. Since it has $n$ vertices, it contains at most $n-1$ edges. Then the debt can be cleared by at most $n - 1$ person-to-person transactions.

## 4 Comments

### 4.1

About a whole day.

### 4.2

Q1 is 5 because it is too abstract. I personally think this is a mathematical analysis question instead of an algorithm question.
Q2 and Q3 both are 3 because the situations of these underlying questions are not that hard to comprehend.

### 4.3

No collaborators.