CS3230: Design and Analysis of Algorithms
Semester 2, 2022-23, School of Computing, NUS

Solutions of Practice Problem Set 2

February 16, 2023

Below we use $\mathbb{E}[\cdot]$ to denote the expectation.

Question 1: Suppose you are given a set of n pair of positive integers $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, where for all $i \in \{1, \cdots, n\}$, $1 \le x_i, y_i < n^2$. Let us define the *power* of a pair (x, y) as the integer $x + n^y$. Design an $O(n)$ time algorithm that sorts the input set of pairs in non-decreasing order of their powers.

Solution: Since $1 \le x_i, y_i < n^2$ for all i, we have that for any pair (x_i, y_i), $x_i < n^{y_i}$ iff $y_i \ne 1$. Hence, we can handle the pairs with $y_i = 1$ and those with $y_i \ne 1$ separately. First, scan through the array and put those with $y_i = 1$ into array A_1 and the rest into array A_2. This takes $O(n)$ time. Next, sort A_1 directly by computing and comparing their respective powers $x_i + n$. Since these values are bounded above by $O(n^2)$, we can sort A_1 in $O(n)$ using Radix sort. To sort A_2, we can first sort by x values and then by y values using a stable sorting algorithm (similar to the idea of Radix sort). The correctness is guaranteed by $x_i < n^{y_i}$ as shown above. Since the x and y values are all bounded above by n^2, we can use Radix sort as a stable sorting algorithm to sort A_2 in $O(n)$ time. Lastly, merge the sorted A_1 and A_2 using the merge routine of merge sort, which runs in $O(n)$ time as well. The overall complexity is $O(n)$.

Note: Computing powers $x_i + n^{y_i}$ explicitly can lead to exponential complexity.

Question 2: We call an integer array A *k-mixed* if exactly k of the integers present in A are even, and all the other integers in A appear in sorted order. Design an algorithm that given a $\lceil n/\log n \rceil$-mixed array A containing n integers, sorts it in $O(n)$ time.

Solution: The key idea is to handle the odd and even integers separately. First, scan through the array A and put the odd integers into array A_O and the even integers into array A_E. We have $|A_E| = k = \lceil \frac{n}{\log n} \rceil$. This step takes $O(n)$ time. Next, since A_O is sorted by definition, we are left with A_E, which can be sorted with merge sort in $O(k \log k) = O(\frac{n}{\log n} \log \frac{n}{\log n}) = O(n)$ time. Lastly, merge the sorted A_O and A_E using the merge routine of merge sort, which runs in $O(n)$ time as well. The overall complexity is $O(n)$.

Question 3: Consider an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Order the vertices in V randomly. Let I be the set of vertices v whose all the neighbors occur after v in the above random order. Note, I is an independent set, i.e., no edges between the vertices in I (think why?). Show that expected size of I is $\sum_{v \in V} \frac{1}{d(v)+1}$, where $d(v)$ denotes the degree of v in G.

Solution: Consider any arbitrary edge $(u, v) \in E$. Suppose $u \in I$. By definition of I, v occurs after u in the order. This means that $v \notin I$. Hence I is an independent set.

Let $X_v = \mathbb{1}\{v \in I\}$. (We use this notation $\mathbb{1}\{v \in I\}$ to denote that $X_v = 1$ if $v \in I$; 0 otherwise, i.e., X_v is an indicator random variable denoting whether $v \in I$ or not.) Then $\mathbb{E}[X_v]$ denotes the probability that v occurs before all its neighbors in the order. Since v and its neighbors are ordered randomly, we have $\mathbb{E}[X_v] = \frac{1}{d(v)+1}$. By linearity of expectation, we have $\mathbb{E}[|I|] = \sum_{v \in V} \mathbb{E}[X_v] = \sum_{v \in V} \frac{1}{d(v)+1}$.

Question 4: In a fictional dice game, you repeatedly toss a fair (i.e., six sides are equally likely) die. If it ever turns to be an odd number, you lose. Otherwise, your score is the number of tosses that it takes you to get the first 6. What is the probability that your score is 1 conditioned on the event that you do not lose.

Solution: The probability of tossing a 6 on the first toss is $\frac{1}{6}$. The probability of not losing is $\sum_{i\geq 0}^{\infty}(\frac{2}{6})^i \times \frac{1}{6} = \frac{1}{4}$, where the term $(\frac{2}{6})^i$ denotes the probability of tossing a 2 or a 4 for i times before finally tossing a 6. The conditional probability required is thus $\frac{\frac{1}{6}}{\frac{1}{4}} = \frac{2}{3}$.

Note: The answer may seem counter-intuitive since in a fair 3-sided die with 2, 4 and 6, the probability of getting a 6 in the first toss is only $\frac{1}{3}$. What's going on is that the condition of not losing actually biases the number of tosses to be small.

Question 5: Given a permutation π over $\{1, 2, \cdots, n\}$, let $L(\pi)$ be the length of the longest increasing subsequence in π. (Note, a subsequence may not be contiguous. E.g., in the permutation $\pi = (1, 6, 4, 5, 2, 7, 3)$, the longest increasing subsequence is $(1, 4, 5, 7)$ and thus $L(\pi) = 4$.)

Show that for a random permutation π over $\{1, 2, \cdots, n\}$, $\mathbb{E}[L(\pi)] = O(\sqrt{n})$. (Additionally, also try to show that $\mathbb{E}[L(\pi)] = \Omega(\sqrt{n})$.)

Solution: To show $\mathbb{E}[L(\pi)] = O(\sqrt{n})$, note that for any fixed k-tuple $(i_1, ..., i_k)$, the probability that $\pi(i_1) < \pi(i_2) < ... < \pi(i_k)$ is exactly $\frac{1}{k!}$, since any of the possible orderings are equally likely to occur. Hence by union bound, we have $\Pr[L(\pi) \geq k] \leq \frac{\binom{n}{k}}{k!} \leq (\frac{ne^2}{k^2})^k$, using inequalities $\binom{n}{k} \leq (\frac{en}{k})^k$ and $k! \geq (\frac{k}{e})^k$. Further note, $L(\pi)$ can be at most n. Hence, we have

$$\mathbb{E}[L(\pi)] = \sum_{k\geq 0} k \cdot \Pr[L(\pi) = k] \tag{1}$$

$$\leq 10\sqrt{n} \cdot \sum_{k<10\sqrt{n}} \Pr[L(\pi) = k] + n \cdot \sum_{k\geq 10\sqrt{n}} \Pr[L(\pi) = k] \tag{2}$$

$$\leq 10\sqrt{n} \cdot \Pr[L(\pi) < 10\sqrt{n}] + n \cdot \Pr[L(\pi) \geq 10\sqrt{n}] \tag{3}$$

$$\leq 10\sqrt{n} + n \cdot \Big(\frac{ne^2}{(10\sqrt{n})^2}\Big)^{10\sqrt{n}} \tag{4}$$

$$= O(\sqrt{n}). \tag{5}$$

To show $\mathbb{E}[L(\pi)] = \Omega(\sqrt{n})$, let $X_i = \mathbb{1}\{\text{some entry in } (i-1)\sqrt{n}+1 \leq j \leq i\sqrt{n} \text{ satsifies } (i-1)\sqrt{n}+1 \leq \pi(j) \leq i\sqrt{n}\}$ (i.e., an indicator random variable denoting whether some entry in $(i-1)\sqrt{n}+1 \leq j \leq i\sqrt{n}$ satsifies $(i-1)\sqrt{n}+1 \leq \pi(j) \leq i\sqrt{n}$ or not). Note that we have $X_i = 0$ when all the entries in the interval $[(i-1)\sqrt{n}+1, i\sqrt{n}]$ are from the $n - \sqrt{n}$ numbers outside the interval, so we have

$$\Pr[X_i = 0] = \frac{\binom{n-\sqrt{n}}{\sqrt{n}}}{\binom{n}{\sqrt{n}}} \tag{6}$$

$$= \prod_{i=0}^{\sqrt{n}-1} \frac{n - \sqrt{n} - i}{n - i} \tag{7}$$

$$\leq (1 - \frac{1}{\sqrt{n}})^{\sqrt{n}} \tag{8}$$

$$\leq e^{-1}. \tag{9}$$

Hence, we have $\mathbb{E}[X_i] \geq 1 - e^{-1}$ and $\mathbb{E}[L(\pi)] \geq (1 - e^{-1})\sqrt{n} = \Omega(\sqrt{n})$.

Question 6: Suppose n people queue up in a movie theater which has exactly n seats. However, the first person in the queue has lost his ticket and sits in one of the empty seats uniformly at random. Subsequently, each person (everybody else has their tickets) sits either in his assigned seat, or if that seat is already occupied, seats in an empty seat uniformly at random. What is the expected number of people not sitting in their assigned seats?

Solution: Without loss of generality, assume that the first person had ticket for seat 1, the second person has ticket for seat 2, and so on. Observe that if person j finds that his seat is already occupied, then the unoccupied seats at that point are $\{1, j+1, j+2, ..., n\}$. To prove this, consider the seating process. Suppose person 1 sits in seat $i_i > 1$, now persons $2, ..., i_1 - 1$ sit in their correct seats. Person i_1 finds his seat taken but seats $\{1\} \cup \{i_1 + 1, ..., n\}$ free. The proof follows by induction.

Let p_i be the probability that the i-th person sits in the wrong seat. The expected number of people not sitting in their assigned seats is simply $\sum_i p_i$ (this could be established using indicator random variable). Clearly, $p_1 = \frac{n-1}{n}$. For $i > 1$ and $j < i$, let $p_{i,j}$ be the probability that person j sits in seat i. Then we have $p_{i,1} = \frac{1}{n}$. For $j > 1$, note that person j sits in seat i exactly if j sits in the wrong seat and happens to choose seat i. From the observation above, we know that if j sees his seat already taken, seats $\{1\} \cup \{j+1, ..., n\}$ will be free. Hence, given that person j sits in the wrong seat, he sits in seat i with probability $\frac{1}{n-j+1}$. Hence, we have $p_{i,j} = \frac{p_j}{n-j+1}$ for $j > 1$, and thus, $p_i = \frac{1}{n} + \sum_{j=2}^{i-1} \frac{p_j}{n-j+1}$ for $i > 1$.

Solving the above recurrence, we have $p_i = \frac{1}{n-i+2}$ for $i > 1$. Hence,

$$\sum_{i=1}^{n} p_i = \frac{n-1}{n} + \sum_{i=2}^{n} \frac{1}{n-i+2} \tag{10}$$

$$= 1 + \sum_{i=3}^{n} \frac{1}{n-i+2} \tag{11}$$

$$= \sum_{i=1}^{n-1} \frac{1}{i} \tag{12}$$

$$= \Theta(\log n). \tag{13}$$

Question 7: Consider a connected graph $G = (V, E)$ with n nodes (i.e., $|V| = n$) and m edges (i.e., $|E| = m$). For any (non-trivial) partitioning of $V = V_1 \cup V_2$ (with $V_1 \cap V_2 = \emptyset$ and both V_1 and V_2 are non-empty) we call the set of all the edges having one endpoint in V_1 and other in V_2 a *cutset.* Now we would like to find a partition such that the corresponding cutset is of minimum size.

To find such a partition we would adopt the following randomized procedure (Procedure $\mathcal{A}$):

1. Pick an edge of G uniformly at random, and contract its two endpoints into a single node. Repeat this process until only two nodes remain.

2. Let u_1, u_2 be the two remaining nodes at the end. Let V_1 be the set of nodes that went into u_1 (while contracting edges), and similarly V_2 be the set of nodes that went into u_2. Output V_1 and V_2 as the partition.

Just to clarify, suppose the algorithm choose an edge $u - v$ to contract, then you may think of the new node as a set uv and all the edges (except $u - v$) incident on u and v in original will now incident on this new node. Note, contraction of edges may create multiple edges between two nodes. See Figure 1. Now answer the following questions.

(a) Show that the size of a minimum cutset is at most $\frac{2m}{n}$.

(b) Probability that the cutset corresponding to the output (partition) of Procedure $\mathcal{A}$ is of minimum size, is at least $\frac{2}{n(n-1)}$.

(c) Repeat Procedure $\mathcal{A}$ for k times, and finally return the partition with minimum cutset size among k output partitions. Now this is your final algorithm. What should be the value k you will choose such that your final algorithm outputs the partition with minimum cutset, with probability at least 2/3? (Provide the detailed analysis.)

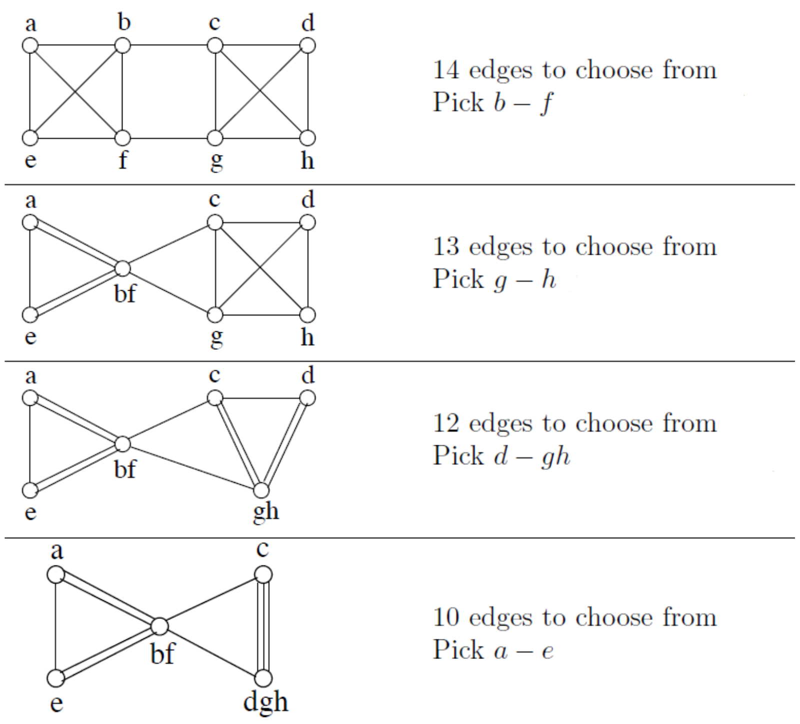

Figure 1: An example run of the algorithm

Solution:

(a) Without loss of generality assume that the graph is undirected. Let $d(v)$ denote the degree (in case of directed graph, sum of in-degree and out-degree) of a vertex v. Observe that $\sum_{v \in V} d(v) = 2m$ (because on the left size we count each edge twice). So by averaging argument there exists one vertex v such that $d(v) \leq \frac{2m}{n}$.

Now if you remove all the edges incident on a vertex then you will get a non-trivial partitioning with cutset of size equal to the degree of that vertex. Hence size of minimum cutset is at most $\frac{2m}{n}$.

(b) Let V_1 and V_2 be the partition corresponding to a minimum cut, and size of the minimum cut is s. Then by the statement of the previous question, probability that at the first step the algorithm will pick some edge from this particular cut is $\frac{s}{m} \leq \frac{2}{n}$. The stated algorithm returns the right answer as long as it never picks an edge across the minimum cut. If it always picks a non-mincut edge, then this edge will connect two nodes on the same side of the cut, and so it is okay to collapse them together.

Each time an edge is collapsed, the number of nodes decreases by 1. Therefore,

$$\begin{aligned} \Pr[\text{Final cut is a minimum cut}] &= \Pr[\text{The first selected edge is not in minimum cut}] \times && (14)\\ &\Pr[\text{The second selected edge is not in minimum cut}] \times \cdots && (15)\\ &\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})(1 - \frac{2}{n-2}) \cdots (1 - \frac{2}{3}) && (16)\\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{1}{3} && (17)\\ &= \frac{2}{n(n-1)}. && (18) \end{aligned}$$

(c) From the previous question we know that during a particular run of the algorithm probability that the output is not a minimum cut is at most $(1 - \frac{2}{n(n-1)})$. Now if you independently run the algorithm k times, then probability that at least one of the runs outputs a minimum cut is at least $1 - (1 - \frac{2}{n(n-1)})^k \geq 1 - e^{\frac{-2k}{n(n-1)}}$, which is at least $\frac{2}{3}$ for $k = \Theta(n^2)$. So the final algorithm runs in time $O(n^2 m)$ and outputs a min-cut with probability at least $\frac{2}{3}$.

Note: The algorithm that you have just seen is the famous Karger's algorithm for finding global mincut. There are some modifications of the given algorithm that lead to even faster algorithm which is referred as Karger-Stein Algorithm. Interested student may explore online on further improvements.