Homework 4

Prateep Mukherjee

November 26, 2013

1. (a) Let, X_i be the random variable denoting the event that a card i is drawn before queen of spades. Hence, $X_i = 1$ if card is drawn before queen of spades, 0 otherwise. So, the expected number of cards Zeus draws is :

$$\begin{aligned} E &= E(X_1 + X_2 \cdots X_{51}) \\ &= E(X_1 = 1) + E(X_2 = 1) \cdots + E(X_{51} = 1) \text{ (Linearity property)} \quad (1) \end{aligned}$$

Now, to compute $E(X_i = 1)$, assume we have 2 cards, queen of spades and card i. Therefore, the probability that i is drawn before queen of spades, is $1/2$. Using this in Eq 1, we get the total expected value to be $\frac{51}{2}$.

(b) The maximum value card that Zeus gives to Poseidon is the first card Zeus draws from the stack. Each of the 14 cards in a suit can be the first card drawn, including queen of spades. There are 4 cards of each value in the deck. Therefore, the probability of drawing a particular value card is $4/52 = 1/13$. Thus, the expected maximum value card is : E(max-value) $= 1/13 \cdot \sum_{i=2}^{14}(i) = 8$.

(c) The minimum value card that Zeus gives to Poseidon is the last card Zeus draws from the stack. Using linearity of expectation property, we consider expectation of each value of a card, and sum up all these expections to get the final desired expectation. Now, for 2 to be the last card drawn, the only condition it must satisfy is that it is drawn before queen of spades. For 3 to be the last card drawn, there are two conditions - (i) It is drawn before queen of spades; and (ii) no 2 is drawn before it. Generalizing this notion, we can say that a card of value i can be the last card drawn if no card with values $[1, i-1]$ is drawn before it. The expectation is therefore written as : E(min-value) $= \lceil 4/5 \times (1/13 + 2/13 + \cdots 13/13) \rceil = \lceil 28/5 \rceil = 6$. (Probability for each value i is computed by only picking cards from the range $[i, 13]$. Also, the probability of choosing card i before queen of spades is $4/5$, as we can pick 4 different suits of card i before queen of spades in 4 out of 5 ways.)

(d) I don't know

2. (a) The number of total permutations of set S is $n!$. Tieing a and b together to ensure that they remain adjacent, the total number of permutations is $2(n-1)!$. Therefore, the probability that a and b are adjacent is $2(n-1)!/n! = 2/n = 2/(a+1)$.

(b) Reference: `http://www.cs.duke.edu/courses/fall09/cps230/hws/hw4/headsol.pdf`

To compute the probability of node i being an ancestor of node j , one observation is that nodes with priorities lesser than i will only affect the probability of node i being adjacent to node j in the ordered sequence. So, node i is the ancestor of node j if and only if there is no node k ($k < i$) such that the search key of node k lies in between i and j in the permutation. We show that both sides of this condition is true.

Proof by contrapositive: To prove a statement "If A then B" is equivalent to proving the statement "If not B then not A".

First, let us prove that there are no nodes k ($k < i$) between i and j in the permutation if i is an ancestor of j. We prove this proof by contrapositive. Therefore, we show that node i is not an ancestor of node j if there exists a node k between i and j, which means that the search key value of node k lies in between nodes i and j. Now, since priority of node k is lesser than both i and j , node k cannot be the descendent of nodes i and j in the treap, according to the heap property. Again, as the search key of node k lies in between nodes i and j, according to the binary search tree(BST) properties of a treap, nodes i and j will lies on different branches of the treap. Therefore, node i cannot be the ancestor of node j. Since the complement holds true, the original claim also holds good. Hence, we have proved that left side of our statement.

Second, let us prove that if i is an ancestor of j then there cannot exist a node k ($k < i$) which occurs between i and j in the permutation. As before, we prove this by using proof by contrapositive. Therefore, we show that there is a node k ($k < i$) between i and j in the permutation if i is not an ancestor of j. So, i and j are on opposite sides of a higher node. Now, respecting the heap property of a treap, this higher node must have a priority lesser than i and j. Also, if we consider the BST properties of a treap, nodes in the left subtree of k have search key values lesser than k, while those on the right have search key values greater than k. Since the permutation has a bijective mapping with the search key values, this can only be true if k is between i and j in the permutation. Hence, we prove the right hand side of our statement.

Third, we ignore nodes with priorities $> i$, since the treap is a min-heap and they have no influence on the probability of node i being the ancestor of node j. Therefore, according to the statement above, i is the ancestor of j if and only if i is adjacent to j. We know the probability of the later event from (a). Therefore, the probability that i is an ancestor of j is $2/(i+1)$.

(c) The depth of a node j is the number of ancestors it has. Let, $X_{i,j}$ be an indicator random variable defined as follows:

$$X_{i,j} = \begin{cases} 1, & \text{if node } i \text{ is the ancestor of node } j, \\ 0, & \text{if node } i \text{ is not the ancestor of node } j. \end{cases}$$

The expected value of depth of node j is therefore,

$$\begin{aligned} E(depth(j)) &= \sum_{i<j} 1 \cdot Pr[X_{i,j} = 1] + 0 \cdot Pr[X_{i,j} = 0] \\ &= \sum_{i<j} Pr[X_{i,j} = 1] \\ &= \sum_{i<j} \frac{2}{i+1} \end{aligned}$$

3. (a) Expected value for $\{|N| - |F|\}$ can be computed as follows:

$$\begin{aligned} E(|N| - |F|) &= E(|N|) - E(|F|) \quad \text{(Linearity of expectation)} \\ &= n - \bigcup_{i=1}^{n} E(f(i)) \quad (F = \textstyle\bigcup_{i=1}^{n} f(i)) \\ &= n - \frac{n}{m} \quad \text{(since } p(f(i)) = \tfrac{1}{m}\text{)} \\ &= n(1 - \frac{1}{m}) \end{aligned}$$

(b) Probability that $|N| = |F|$ means that each mapping of $i \in N$ to M is unique and no collision occurs during mapping of n elements. The first element of N has m possible locations to be mapped into, the second has $m-1$ positions, third $m-2$ and so on. The total possible mappings of n elements from N to m locations in M is m^n. Therefore,

$$\mathcal{P} = p(|N| = |F|) = \frac{m \times (m-1) \times (m-2) \ldots (m-n+1)}{\underbrace{m \times m \times m \ldots m}_{\text{n times}}} \tag{2}$$

(c) In this case, we use the recursive formula for computing expectation. Let, E be the expected number of functions we try till we succeed. If we succeed in the first case, which has probability $\mathcal{P}$, then the expected number of trials is 1. Else, we increase our expected number of trials by 1 and the probability is $1 - \mathcal{P}$. Therefore, the expected number of functions we need to try is

$$\begin{aligned} E &= 1 \cdot \mathcal{P} + (1 - \mathcal{P}) \cdot (E + 1) \\ \implies E &= 1/\mathcal{P} \end{aligned}$$

(d) Since choice of functions in each trial is independent, the probability such that none of the first k functions work is $(1 - \mathcal{P})^k$.

4. (a) As the probability of each edge e in graph G is proportional to its weight, a method to choose a random edge would be to pick the edge with maximum weight in G. We can use *max-heap* data structure to choose the maximum weight edge. Inserting all the edges in the *max-heap* is $\mathcal{O}(n^2)$, where n is the number of vertices in the graph, while deleting the root each time would be $\mathcal{O}(\log n)$. Therefore, the algorithm is :

i Build a *max-heap* H of the edges based on their weights.

ii Repeat until there are 2 vertices left in G:

- Pick root e from H and collapse vertices (u, v) connected through e.
- Delete root e from H.
- Set $G = G \setminus e$

iii return the edges remaining in the graph as the min-cut set.

Therefore, at each step of the algorithm, we need $\mathcal{O}(\log n)$ time complexity to delete the root and pick the next highest weighted edge. Since there are $n-2$ steps to the algorithm, the total complexity for selecting a random edge is $\mathcal{O}(n \log n)$.

(b) **Claim:** Picking the edge(e) at the root of H at each iteration of the algorithm ensures that e does not lie in the min-cut set.

Proof: At each step of the algorithm above, we pick an edge which has the maximum weight. Therefore, its chance of being in the minimum cut set is zero. which can be proved by contradiction, because if this edge is indeed included in the min-cut, we can choose any other edge, with smaller weight, in the existing graph and use this to get a smaller min-cut. Therefore, the original method of computing the probability of GUESSMINCUT still holds true. □

(c) At each step of GUESSMINCUT, our algorithm gives the maximum weighted edge. This edge cannot be in the min-cut set, as proved in (b). Therefore, using our algorithm to choose an edge, GUESSMINCUT will compute the min-cut in at most 1 step, as the edges we are deleting will never be in the min-cut. Therefore, the probability of success is $\geq \frac{1}{n^2}$.