CS6170: Randomized Algorithms
Problem Set #1

Name: Swathi Shree Narashiman — Marks: 24

Roll No: EE22B149 — Due: February 9, 23:59

Problem 1 — 3 marks

You are writing a program to simulate the board game "The Settlers of Catan", and you need a way to simulate the throw of two dice and taking their sum. You have at your disposal a subroutine that can output a uniformly random bit. How would you use this subroutine to implement the dice throws. Explain the expected number of uniformly random bits that you will require to perform this task.

Solution:

To simulate a die, Let us draw a random bit $r_i \in \{0,1\}$, where i represents the i^{th} random bit drawn to represent a throw of the die.

A throw of die is represented with a sample space, S $= \{1,2,3,4,5,6\}$. Since, there are 6 elements in the sample space of S, we need **atleast 3 bits to represent the outcome**. Let us define the mapping between outcome of drawing a random bits as follows:

Random bit string	Outcome of die
000	Redraw random bits
001	1
010	2
011	3
100	4
101	5
110	6
111	Redraw random bits

Let X be the random variable representing number of random bits drawn at random to determine the outcome of the die,

$$E[X] = P(NoRedraw) * 3 + P(Redraw) * (E[X]+3)$$

Since 2 cases out of 8 possible random bit string require a redraw,

$$E[X] = \frac{3}{4} * 3 + \frac{1}{4} * (E[X]+3)$$

Solving the equation we get,

$$E[X] = 4$$

Therefore, the **expected number of random bits is 4 for each die**. To simulate the sum, we need a total of 8 random bits.

Collaborator: Shri Prathaa M

Problem 2 **4 marks**

Consider the following alternate algorithms for verifying the multiplication of $n \times n$ matrices.

1. **Algorithm 1**: Given A, B, C, choose a number j uniformly at random from $\{1, 2, \ldots, n\}$ and multiply A with the j^{th} column of B and check if it matches the j^{th} column of C entrywise.
2. **Algorithm 2**: Given A, B, C, choose two numbers i and j uniformly at random from $\{1, 2, \ldots, n\}$ and multiply the i^{th} row of A and the j^{th} column of B and check if the product is equal to the $(i, j)^{th}$ entry of C.

Analyze the two algorithms and explain which one is better. Are these algorithms better than Frievald's algorithm? What is the running time of this algorithm (in terms of δ) if I want to make the error probability δ? Give all the details.

Solution:

Let us analyse the error probability and running time of both the algorithms.

Algorithm 1: The error probability is given by,

$$Pr[Error_1] = Pr[AB \neq C | [AB]_j = C_j]$$

Let us upper bound the error, the worst case occurs when all columns match except 1 column,

$$Pr[Error_1] = Pr[\textit{Choosing n-1 matching columns}]$$

$$Pr[Error_1] = 1 - \frac{1}{n}$$

Hence,

$$Pr[Error_1] \leq 1 - \frac{1}{n}$$

Algorithm 2:

$$Pr[Error_2] = Pr[AB \neq C | [AB]_{ij} = C_{ij}]$$

Let us upper bound this error, the worst case occurs when 1 out of n^2 elements donot match.

$$Pr[Error_2] = Pr[\textit{Choosing } n^2\textit{-1 matching elements}]$$

$$Pr[Error_2] = 1 - \frac{1}{n^2}$$

Hence,

$$Pr[Error_2] \leq 1 - \frac{1}{n^2}$$

Clearly, $\frac{1}{n} > \frac{1}{n^2}$,

$$1 - \frac{1}{n} < 1 - \frac{1}{n^2}$$

Therefore, **Algorithm 1 is better than Algorithm 2**.

Runtime for Algorithm 1:

Let us repeat the algorithm 1 k times, to amplify the success,

$$Pr[Error_1] = Pr[\textit{Each Repetition Error Occurs}] = (1 - \frac{1}{n})^k$$

Using, $1-x \leq e^{-x}$,

$$Pr[Error_1] \leq e^{\frac{-k}{n}} = \delta$$

This gives us,

$$k = n * log_e \frac{1}{\delta} = O(n)$$

Runtime for Algorithm 2:

Let us repeat the algorithm 2 k times, to amplify the success,

$$Pr[Error_1] = Pr[Each\ Repetition\ Error\ Occurs] = (1 - \frac{1}{n^2})^k$$

Using, $1-x \leq e^{-x}$,

$$Pr[Error_1] \leq e^{\frac{-k}{n^2}} = \delta$$

This gives us,

$$k = n^2 * log_e \frac{1}{\delta} = O(n^2)$$

In **Frievald's algorithm**,

$$Pr[Error_{Frievald}] \leq \frac{1}{2}$$

Both Algorithm 1 and Algorithm 2 have worst case errors dependent on n and therefore,

$$Pr[Error_{Frievald}] < Pr[Error_1] < Pr[Error_2]$$

Hence, Both the **algorithms are worser than Frievald**, while **Algorithm 1 is better than 2.**

Collaborator: Shri Prathaa M

Problem 3 — 3 marks

An s-t-cut in a graph is a set of edges such that their removal gives a new graph which does not contain a path from s to t. Consider the following modification of Karger's algorithm to compute the smallest s-t-cut in the graph: Choose a random edge in the graph such that it is not between supernodes containing s and t, and contract it; keep continuing until the only two supernodes are the ones containing s and t, and output this as the minimum s-t-cut.

Show that there are graphs such that the success probability of this algorithm finding the minimum s-t-cut is exponentially small.

Solution:

Consider a graph G(V,E). **Let degree of each vertex $\geq$ k** .

Let C be the minimum s-t cut in the Graph G for a given s and t. **Let size of C be c**.

Let E_i be the event that the i^{th} contraction was not a cut-edge in C and Let F_i be the event that the first i-1 contractions were not cut-edges in C.

Hence, we can write the number od edges to be,

$$|E| = \frac{nk}{2}$$

, to avoid double counting we divide by 2.

Clearly,

$$Pr[E_1] = Pr[F_1] = 1 - \frac{2c}{nk}$$

After i contractions, number of vertices left is n-i. The edge density is given by,

$$|E| \leq \frac{(n-i)k}{2}$$

, we have an inequality because if a cut edge is contracted the edge density reduces otherwise it remains the same.

This gives us,

$$Pr[E_i|F_{i-1}] = 1 - \frac{2c}{(n-i+1)k}$$

Our goal is to find $Pr[F_{n-2}]$.

We can write the recursive relation,

$$Pr[F_{n-2}] = Pr[E_{n-2}|F_{n-3}]Pr[F_{n-3}]$$

$$Pr[F_{n-2}] = \prod_{i=1}^{n-2} 1 - \frac{2c}{(n-i+1)k}$$

Using the inequality $1 - x \leq e^{-x}$,

$$Pr[F_{n-2}] \leq \prod_{i=1}^{n-2} e^{-\frac{2c}{(n-i+1)k}}$$

$$Pr[F_{n-2}] \leq e^{-\sum_{i=1}^{n-2} \frac{2c}{(n-i+1)k}}$$

$$Pr[F_{n-2}] \leq e^{-\frac{2c}{k}\sum_{i=3}^{n} \frac{1}{i}}$$

Using Reimann Integral, ‘

$$\sum_{i=3}^{n} \frac{1}{i} \leq log_e n$$

$$Pr[F_{n-2}] \leq e^{-\frac{2c}{k} log_e n}$$

$$Pr[F_{n-2}] \leq n^{-\frac{2c}{k}}$$

Therefore,

$$Pr[success] \leq n^{-\frac{2c}{k}}$$

Hence, when c>>k the success probability of the algorithm is **Exponentially decay in n**. Therefore, for graphs with a cut edge size for a given s and t much greater than the degree of other vertices, the success probability of the algorithm is exponentially small.

For, $k \approx c$,

$$Pr[success] \approx n^{-2}$$

Collaborator:Shri Prathaa M

Problem 4 — 4 marks

An k-cut is a partition of the vertex set into k vertex-disjoint parts $V_1, V_2, \dots, V_k$ such that $V_1 \cup V_2 \cup \dots \cup V_k = V$. The size of the cut is the number of edges whose endpoints are in different partitions. The *min k-cut* is the k-cut of minimum size. Explain how you can modify Karger's algorithm to obtain a min k-cut in a graph. You should write the algorithm clearly, and analyze the running-time and the success probability of the algorithm. You do not need to optimize the running-time by choosing the right data structures.

Solution:

Algorithm to Find the k-cut partition

```
Input: Graph G(V,E) with n vertices
while $|V| \geq k$ :
    Choose edge $e \in E$ uniformly at random from $G$
    $G \leftarrow G \backslash e$
Output: Return the number of edges in $G$
```

Let C be the min k-cut in the graph, let **size of C be c.**

This means each vertex has a ,

$$degree \geq \frac{c}{k-1}$$

We make this argument because, if $degree < \frac{c}{k-1}$ then we can remove k-1 vertices to form k partitions with a cut-edge size < c.

$$\implies |E| = \frac{nc}{2(k-1)}$$

Let E_i be the event that the i^{th} contraction was not a cut-edge in C and Let F_i be the event that the first i-1 contractions were not cut-edges in C.

$$Pr[E_1] = Pr[F_1] = 1 - \frac{2(k-1)}{n}$$

After i contractions, number of vertices left is n-i. The edge density is given by,

$$|E| \approx \frac{(n-i)c}{2(k-1)}$$

, we have an inequality because if a cut edge is contracted the edge density reduces otherwise it remains the same.

his gives us,

$$Pr[E_i|F_{i-1}] = 1 - \frac{2(k-1)}{n-i+1}$$

Our goal is to find $Pr[F_{n-k}]$.

We can write the recursive relation,

$$Pr[F_{n-k}] = Pr[E_{n-k}|F_{n-k+1}]Pr[F_{n-k+1}]$$

$$Pr[F_{n-k}] = \prod_{i=1}^{n-k} 1 - \frac{2(k-1)}{n-i+1}$$

Using the inequality $1 - x \leq e^{-x}$

$$Pr[F_{n-k}] \leq \prod_{i=1}^{n-k} e^{-\frac{2(k-1)}{n-i+1}}$$

$$Pr[F_{n-k}] \leq e^{-\sum_{i=1}^{n-k} \frac{2(k-1)}{n-i+1}}$$

$$Pr[F_{n-k}] \leq e^{-2(k-1)\sum_{i=1}^{n-k} \frac{1}{n-i+1}}$$

Using Reimann integral

$$\sum_{i=k+1}^{n} \frac{1}{i} \leq log_e n$$

$$Pr[F_{n-k}] \approx e^{-2(k-1)log_e n}$$

$$Pr[F_{n-k}] \approx n^{-2(k-1)}$$

Therefore, success probability of 1 run of the algorithm is given by,

$$Pr[Success] \approx n^{-2(k-1)} = \Omega(\frac{1}{n^{2k-2}})$$

Let us run the algorithm λ times to amplify the success,

$$Pr[Error] \approx (1 - n^{-2(k-1)})^{\lambda}$$

$$Pr[Error] \leq e^{\frac{-\lambda}{n^{2k-2}}}$$

If the algorithm errors with δ,

$$e^{\frac{-\lambda}{n^{2k-2}}} = \delta$$

$$\lambda = n^{2k-2} log_e \frac{1}{\delta} = O(n^{2k-2} log_e \frac{1}{\delta})$$

One run of Karger contraction to k vertices takes approximately O(n^2) time,

$$Running\ time\ of\ k\ cut = O(n^{2k} log_e \frac{1}{\delta})$$

Collaborator: Shri Prathaa M

Problem 5 **5 marks**

You have a function $f : \{0,1\}^n \to \{0,1\}$ stored as a bit vector indexed by strings in $\{0,1\}^n$ - we will refer to the bit vector also by f. The function satisfies the following property: for every $x, y \in \{0,1\}^n$, $f(x \oplus y) = f(x) \oplus f(y)$, where $\oplus$ is the bitwise XOR of the strings.

You send this bit vector over a channel which corrupts at most $1/5$ fraction of the bits in the vector - the bits that are corrupted can be any of the bits and you have no control over it. We will call this corrupted vector $\widehat{f}$.

Once you receive this bit vector, you want to do the following: given a $z \in \{0,1\}^n$, compute $f(z)$. You are allowed to query the bit vector at any point of your choice - this means that given a string x, you will get $\widehat{f}(x)$. For an input z, you want to compute $f(z)$ with probability $> 1/2$ (**strictly greater than** $1/2$) with as few queries as possible. Explain how you will do this - you should describe the number of queries you will make, the algorithm to compute $f(z)$ and an analysis of the success probability of your algorithm.

Note that your algorithm should work for every z, even for those z for which $\widehat{f}(z) \neq f(z)$.

Solution:

Algorithm

- Let us select random string, $z_i \in \{0,1\}^n$
- For z_i we query $\widehat{f}(z_i)$ and $\widehat{f}(z_i \oplus z)$.
- Compute $\widehat{f}(z_i) \oplus \widehat{f}(z_i \oplus z)$
- Return the value computed as f(z)

Let us consider the pairs (z_i, z_j) such that $z_i \oplus z_j = z$. There are 2^{n-1} such pairs. Let us place these pairs in a table with 2^{n-1} rows and 2 columns.

Let the channel corrupt the bits by blackening atmost $\frac{2^n}{5}$ boxes in the table. In a given row, worst case both the boxes (columns) are blackened. Our algorithm succeeds when both boxes are white or both are blackened in a given randomly selected row.

The success probability of our algorithm is given by,

$$Pr[Success] = Pr[Both\ the\ boxes\ are\ white\ in\ a\ randomly\ picked\ row]$$

$$+Pr[Both\ the\ boxes\ are\ black\ in\ a\ randomly\ picked\ row]$$

$$Pr[success] \geq Pr[Both\ the\ boxes\ are\ white\ in\ a\ randomly\ picked\ row]$$

Number of rows with atleast 1 black box $\leq \frac{2^n}{5}$.

$$Pr[Failure] \leq \frac{\frac{2^n}{5}}{2^{n-1}}$$

$$Pr[Failure] \leq \frac{2}{5}$$

Therefore,

$$Pr[Success] \geq \frac{3}{5} > \frac{1}{2}$$

Hence, when we choose a random pair (z_i, z_j) such that $z_i \oplus z_j = z$, with a probability of atleast 1/2 , $f(z) = \widehat{f}(z_i) \oplus \widehat{f}(z_j)$. The number of queries required to compute the value of f(z) is 2.

Collaborator: Took ideas from a few people

Problem 6 **5 marks**

A *majority tree* is a complete 3-ary tree of height h where each leaf of the tree is assigned a boolean value. The value of an internal node is recursively computed by finding the boolean values at its children, and then taking their majority. The value of the majority tree is the value of its root.

Consider the following randomized algorithm for evaluating a majority tree: From the root, choose two of its children uniformly at random, and recursively evaluate the trees rooted at these children. If they return the same value, then return that as the value of the root. Else, recursively evaluate the third child and compute the majority.

Show that the expected number of leaf nodes that are queried by this algorithm is at most $n^{0.9}$, where $n = 3^h$.

Solution:

Let i denote the i^{th} level in the majority tree where $i \in \{0, 1, 2, ...h\}$.

Then, the number of nodes in the i^{th} level is given by 3^i.

Let the random variable Y_i denote the number of nodes queried in the i^{th} level of the tree, since the root node is queried ,

$$E[Y_0] = 1$$

The goal is to find, $E[Y_h]$.

The children nodes can take 8 possible values : 000, 001, 010, 011, 100, 101, 110, 111. Of which while recursively evaluating a given node , **we query only 2 children node in case of 000 or 111**. For the remaining 6 cases, **worst query complexity is 3 children nodes.**

Let us consider a majority tree which only consists of children that have a worst case query complexity 3.
For such a tree,

With probability= $\frac{1}{3}$ we get the same value while querying two children and with probability $\frac{2}{3}$ we get different values while two children.

Therefore, we define the recursive relation,

$$E[Y_i] = NumberOfChildrenQueriedPerNode * E[Y_{i-1}]$$

$$E[Y_i] = (\frac{1}{3} * 2 + \frac{2}{3} * 3) * E[Y_{i-1}]$$

$$E[Y_i] = \frac{8}{3} * E[Y_{i-1}]$$

Applying the recursion with $E[Y_0] = 1$, we get

$$E[Y_h] = (\frac{8}{3})^h * E[Y_0] = (\frac{8}{3})^h$$

Since this is the worst case complexity, for any majority tree we have,

$$E[Y_h] \leq (\frac{8}{3})^h$$

Substituting $h = ln_3 n$,

$$E[Y_h] \leq (\frac{8}{3})^{ln_3 n}$$

$$E[Y_h] \leq n^{ln_3 \frac{8}{3}}$$

$$E[Y_h] \leq n^{0.9}$$

Therefore, expected number of leaf nodes queried by this algorithm is atmost $n^{0.9}$.

Collaborator: Shri Prathaa M