Sketch of Solution for Tutorial on Randomized Algorithms

4. Consider the problem of finding an "approximate median" of an unsorted array A[1..n]: an element of A with rank between n/4 and 3n/4 (inclusive of both). Give a fast Monte Carlo algorithm for finding the approximate median of the array.

   [Answer] Just pick any one element at random and output it. Since you are trying to find an element that can lie anywhere within half the range of the elements, the chance that the correct approximate median is picked is ≥ 1/2.

5. Suppose that you are given a bolt and a set of n nuts of distinct size, exactly one of which fits the bolt. To find the fitting nut, a simple Las Vegas randomized algorithm is designed as follows:

   Repeat until a fitting nut is found

   Pick a random nut and try it

   If it fits the bolt stop

   If it does not fit the bolt, remove the nut from the set of nuts

   What is the exact expected number of nuts that will be compared with the bolt before the proper nut is found?

   [Answer] Let X = number of tries before the correct nut is found. Then X is a random variable with values from 1 to n-1. Note that if the correct nut is not found in n-1 tries, the only remaining nut must fit the bolt.

   P[X = k]

   = (Probability that the wrong nut was picked in each of the previous k-1 tries) × (Probability that the correct nut is picked in the k-th try)

   = (Prob. of picking a wrong nut in the 1st try) × (Prob. of picking a wrong nut in the 2nd try) ×…× (Prob. of picking a wrong nut in the (k-1)-th try) × (Prob. that the correct nut is picked in the k-th try)

   $$= \frac{n-1}{n} \times \frac{n-2}{n-1} \times \ldots \times \frac{n-k+1}{n-k} \times \frac{1}{n-k+1} = \frac{1}{n}$$

   The above is true for k < n-1. For k = n-1, there are only 2 elements left, and picking either one gives the correct nut (either the nut picked or the only other one left), so the last (1/(n-k+1)) is replaced by 1. Hence the required probability becomes

$$= \frac{n-1}{n} \times \frac{n-2}{n-1} \times \dots \times \frac{2}{3} \times \frac{1}{2} = \frac{2}{n}$$

Then $E[X] = \frac{2}{n} \times (n-1) + \sum_{k=1}^{n-2} \frac{1}{n}$

$$= \frac{2(n-1)}{n} + \frac{(n-1)(n-2)}{2n}$$

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

6. Given three n×n matrices A, B, and C, we need to check if AB = C. The simple deterministic algorithm will take $O(n^3)$ time, which can be improved to around $O(n^{2.37})$ using best known matrix multiplication algorithms. Can you design a $O(n^2)$ time Monte Carlo algorithm for the problem?

   [Hint: if you want to get the time down to $O(n^2)$, you cannot multiply two n×n matrices. But you can multiply two n×n and n×1 matrices in $O(n^2)$ time, may be more than once. But you don't have a n×1 matrix in this problem. So?]

   [Answer] The algorithm is as follows:

   Randomly choose a n×1 matrix r with elements from [0,1]
   If ABr = Cr, answer YES
   else answer NO

   The NO answer is always correct obviously. But the YES answer may be wrong.

   To find the probability of the YES answer being wrong, we need to find the probability that ABr = Cr but AB ≠ C.

   Let X = (AB – C)r = Yr where Y = AB – C

   So $X = [x_1, x_2, x_3, \dots, x_n]^T$

   Since AB ≠ C, Y has at least one non-zero element. Let it be y_{ij}.
   Then $x_i = y_{i1}r_1 + y_{i2}r_2 + \dots + y_{ij}r_j + \dots + y_{in}r_n = y_{ij}r_j + z$

   $P[x_i = 0] = P[x_i = 0 \mid z = 0] \times P[z = 0] + P[x_i = 0 \mid z \neq 0] \times P[z \neq 0]$ (by Bayes' theorem)

   $$= P[r_i = 0] \times P[z = 0] + P[r_i = 1 \text{ and } y_{ij} = -z] \times P[z \neq 0]$$
   $$\leq \frac{1}{2} \times P[z = 0] + \frac{1}{2} \times P[z \neq 0]$$
   $$= \frac{1}{2}$$

For ABr = Cr to hold, all terms of X has to be 0

So the probability that ABr = Cr but AB ≠ C is

$P[x_1 = 0 \text{ and } x_2 = 0 \text{ and } \ldots.. \text{ and } x_n = 0] \leq P[x_i = 0] = \frac{1}{2}$

7. Suppose that at each step of the min-cut Algorithm, instead of choosing a random edge for contraction, two vertices are chosen at random and are merged into a single vertex. This is repeated till only 2 vertices are left. Show that there exist inputs for which the probability that the modified algorithm finds a min-cut is exponentially small.

   [Answer] Take a graph G as follows: Two cliques U and V of n nodes each, connected by a single edge from some node in U to some node in V. So the minimum cut is the partition {U, V} with size 1 (the single edge going between U).

   Let A = event that all of U is contracted to one node and all of V is contracted into one node

   So probability of finding the minimum cut = P[A]

   Let B = event that two final vertices left each have n nodes (but from any of U and V)

   $$\begin{aligned} \text{Then } P[A] &= P[A \mid B] \times P[B] \\ &\leq P[A \mid B] \\ &= \frac{1}{\binom{2n}{n}} \\ &\leq \frac{1}{2^n} \end{aligned}$$

8. A company has a very large number of small video clips stored on disk, which can be downloaded by its subscribers. In order to improve the download time, video clips are stored in a fast cache in memory to avoid costly disk access time. However, memory being costlier and smaller, not everything can be cached, and the company only caches clips that are searched for at least twice in memory cache. First time downloads are served directly from disks. On a second download request of the same clip, it is brought to cache. On getting a download request, the company wants to use a fast way of detecting whether the same clip has already been requested or not. Suggest a bloom filter based scheme for the company to detect this. The company is willing to accept some error that can cause some clips to be put in cache even if they are accessed only for the first time.

[Answer]

Use a Bloom Filter M.

On a download request:

Check M to see if it has been requested already

If no (always correct), set bloom filter

serve from disk

Else (yes, may be false positive)

Serve from cache

Note that on "yes", for the 2nd request, it is still not in cache. Checking if the item is in cache or not is separate, I assumed it will be done using standard techniques. So the 2nd request will bring it in cache, from 3rd request onwards it will be found in cache. The bloom filter shown is only to detect if it is a first or second request.

9. There are n boxes (numbered 1 to n), with exactly one box containing Rs. 10000. The other boxes are empty. To find the money, each box will have to be opened to see if it contains the money until the money is found. Opening a box counts as one "probe". We want to find the money while minimizing the number of probes performed. The following Las Vegas algorithm is designed for it.

Select x in [0, 1] uniformly at random.
if x = 1
then Probe boxes in order 1, 2,…, n and stop if bill is located
else Probe boxes in order n, …, 1 and stop if bill is located

Show that the expected number of probes before the money is found is exactly (N+1)/2.

[Answer]

Suppose the money is in box k. Then the expected number of tries needed to find the money

= (number of tries needed if x = 0) × P[x = 0] + (number of tries needed if x = 1) × P[x = 1]

$= \frac{1}{2} \times k + \frac{1}{2} \times (n - k + 1)$

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

Since this is independent of k, it does not matter which box the money is in. So the expected number of tries to find the money is (n+1)/2.

10. Consider an experiment in which two people are asked to choose n numbers one by one randomly from a very large range of positive integers. The numbers chosen need not be distinct, so a person can choose the same number more than once also. At the end, we want to find out if they have chosen the same set of numbers (i.e., the same set of distinct numbers, each repeated the same number of times) or not. You are asked to design a Monte Carlo algorithm that outputs YES if they have chosen the same set of numbers, NO otherwise. The NO answer must always be correct but the YES answer may be wrong with a probability of at most 1/2. Your algorithm should run in O(n) time and O(n) space.

    [Answer] Use two hash tables, A and B with random hash functions. All slots initially have value 0. For person 1, hash the integers in A; if two elements hash to the same slot increment the count in that slot, no need to keep track of the exact distinct elements that hashed there. Do the same for person 2 in table B. Now compare A and B. If the counts are the same in all slots, answer YES, else answer NO.

    The NO answer is always correct obviously. But the YES answer can be wrong, so the two sets can be different but the answer can be YES. So suppose the two sets differ. Then there is an element x which appears a different number of times in the two sets. It is possible that some other element hashes to the same slot as x, thereby corrupting the count of x. If the size of the hash table is m, the expected number of elements that hash to the same slot as x is n/m. Set m > 2n. Then the number of elements that hash to the same slot as x is ½. So with probability > 1/2, no other element hashes to the same slot as x. If no other element hashes in x's slot, then the difference in the number of x's coming from the two sets cannot be cancelled out, so the difference in the sets will be detected.