CS 6550: Randomized Algorithms January 17, 2019

Problem Set 1

Instructor: Eric Vigoda *STUDENT NAME*

Problem 1 Random Permutation

(a) Suppose you are required to generate a random permutation of size n. Assuming that you have access to a source of independent and unbiased random bits, suggest a method for generating random permutations of size n. Efficiency is measured in terms of both time and number of random bits. What lower bounds can you prove for this task?
[Python Implementation (which is basically pseudocode)]

$k = \text{rand}(1, n!)$
r = n-1
string = ""
list = [i for i in range$(0, n-1)$]
while $r \geq 1$:], (runs at most n times)
    while $k > r!$: (runs at most r times)
        $k = k - r!$ (O($\log_2(n!)$) = $O(n\log_2(n))$)
    string.append(list[1]) (O(n))
    list.remove(list[1]) (O(r))
    r -= 1 O($\log_2(r)$)

The number of random bits used here is bounded by $\log_2(n!) + 1$ which is roughly $O(n\log_2(n))$. Runtime is $(n-1)(O(n\log_2(n)) + O(n)) + (n-2)(O(n\log_2(n)) + O(n)) + \cdots + 2(O(2) + O(2)) = O(n^3\log_2(n))$.

(b) Consider the following method for generating a random permutation of size n. Pick n random values $X_1, \ldots, X_n$ independently from the uniform distribution over the interval $[0, 1]$. Now, the permutation that orders the random variables in ascending order is claimed to be a random permutation, and it can be determined by sorting the random values. Is the claim correct? How efficient is this scheme?
This claim is correct. We can see this by noting the following. For fixed $a_1, a_2, \ldots, a_n \in [0, 1]$, $P(X_1 = a_1 \text{ and } X_2 = a_2 \text{ and } \ldots \text{ and } X_n = a_n) = P(X_{\sigma(1)} = a_1 \text{ and } \ldots \text{ and } X_{\sigma(n)} = a_n)$, where $\sigma \in S_n$ is any permutation, because the values of these variables are chosen independently and namely the order in which we choose them (which I am really thinking of as the labels on the variables here) does not affect the probability of the outcome. Namely if we choose variable Y first (as X_1) or choose Y $\sigma(1)st$ (as $X_{\sigma(1)}$) the probability that $Y = a_1$ does not change. Sorting the random values takes $O(n\log(n))$ time using mergesort.

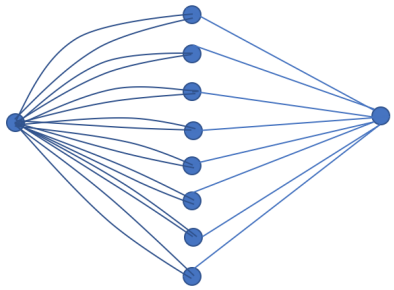
(c) Consider the following implementation of the scheme suggested in (b). The binary representation of the fraction X_j is a sequence of unbiased and independent random bits. For each j we sample the first k bits of X_j. Give a tight bound on k such that we can sort the random variables $X_1, \ldots, X_n$ by comparing just the first k bits.
We calculate a lower bound on the probability of success. Namely we calculate the probability that none of the sampled variables agree completely pairwise on the first k bits, which equals $(X_i)^k \neq (X_j)^k$ for all $1 \leq i < j \leq n$ (where $(X_i)^k$ denotes the first k bits of X_i). So, we calculate $\prod_{i=2}^{n} P((X_i)^k \neq (X_j)^k \text{ for all } 1 \leq j < i \,|\, (X_j)^k \neq (X_l)^k \text{ for all } 1 \leq j < l < i)$. This comes out to $\prod_{i=1}^{n-1} \frac{2^k - i}{2^k} = \prod_{i=1}^{n-1} 1 - \frac{i}{2^k} \geq \prod_{i=1}^{n-1} 1 - \frac{n-1}{2^k} = (1 - \frac{n-1}{2^k})^{n-1}$. So, to summarize, we have $P(\text{success}) \geq (1 - \frac{n-1}{2^k})^{n-1}$. We would like to find a function $f(n, c)$

such that whenever $k \geq f(n,c)$, we have that $P(\text{failure}) \leq \frac{1}{n^c}$. This happens exactly when $P(\text{success}) \geq 1 - \frac{1}{n^c}$. So, we set $(1 - \frac{n-1}{2^k})^{n-1} \geq 1 - \frac{1}{n^c} = \frac{n^c-1}{n^c}$. Taking the $(n-1)$st root of both sides gives $1 - \frac{n-1}{2^k} \geq (\frac{n^c-1}{n^c})^{1/(n-1)}$ or equivalently, $1 - (\frac{n^c-1}{n^c})^{(1/(n-1))} \geq \frac{n-1}{2^k}$ which happens when $2^k \geq \frac{n-1}{1-(\frac{n^c-1}{n^c})^{(1/(n-1))}}$. Finally, we note that this statement is true exactly when $k \geq \log_2(\frac{n-1}{1-(\frac{n^c-1}{n^c})^{(1/(n-1))}})$. So, we get our desired function $f(n,c) = \log_2(\frac{n-1}{1-(\frac{n^c-1}{n^c})^{(1/(n-1))}})$.

Problem 2 Karger's algorithm for s-t min-cut

Consider adapting Karger's algorithm to the problem of finding an s-t min-cut in an undirected graph. In this problem, we are given an undirected graph G together with two distinguished vertices s and t. An s-t cut is a set of edges whose removal from G disconnects s from t; we seek an s-t cut of minimum cardinality. As the algorithm proceeds, the vertex s may get amalgamated into a new vertex as a result of an edge being contracted; we call this vertex the s-vertex (initially the s-vertex is s itself). Similarly, we have a t-vertex. As we run the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex.

(a) Show that there are graphs in which the probability that this algorithm finds an s-t min-cut is exponentially small. Consider a graph similar to the one shown below in which we have k vertices in the middle (the graph shown below has k=8 (s and t are the vertices to the left and right respectively)). In such a graph, the probability of finding the unique min cut is $\prod_{i=0}^{k-1} \frac{2k-2i}{3k-2i} = \prod_{i=0}^{k-1} \frac{2(k-i)}{3(k-i)+i} \leq \prod_{i=0}^{k-1} \frac{2(k-i)}{3(k-i)} = (\frac{2}{3})^k$. Setting k arbitrarily large we can make



such a probability arbitrarily small.

(b) How large can the number of s-t min-cuts in an instance be?

Problem 3 Min 3-way cut

For an undirected graph $G = (V, E)$, a 3-way cut is a subset S of edges whose removal from G breaks the graph into at least 3 components. The size of the 3-way cut is the number of edges in S.

(a) Let k be the size of the minimum 3-way cut of G. For any two distinct vertices u, v of G, prove that $d(u) + d(v) \geq k$.
I assume we are to assume that $|V(G)| \geq 3$. Proof: Assume not. Assume there exists some pair $u, v \in V(G)$ of distinct vertices such that $d(u) + d(v) < k$. Then, construct the following set $F := \{e \in E(G) | e \cap u \neq \emptyset\} \cup \{e \in E(G) | e \cap v \neq \emptyset\}$. We see that $|F| < k$. Clearly, if the two sets whose union is F are disjoint we get some number of edges in F which is bounded by k and if these two sets are not disjoint, we get even fewer edges in F. Then, note that $G \setminus F$ has u and v as isolated vertices. Also, $G \setminus \{u, v\}$ has at least one vertex and in particular at least one component. So, $G \setminus F$ has at least 3 components, contradicting the fact that any minimum 3-way cut has $\geq k$ edges and we are done.

(b) Show that $|E| \geq kn/4$ where $n = |V|$ is the number of vertices of G.
First, if G has an even number of vertices construct a partition of them into pairs $P := \{\{u_i, v_i\} | V(G) = \coprod_{i \in [\frac{n}{2}]} \{u_i, v_i\}\}$. Otherwise, if G has an odd number of vertices let $P := \{\{u_i, v_i\} | \coprod_{i \in [\frac{n}{2}]} \{u_i, v_i\} = V(G) \setminus \{w\}$ for some $w \in V(G)\}$. Next, note that by part a we have that $2|E| = \sum v \in V(G) d(v) \geq \sum_{i \in [\frac{n}{2}]} k$ in the case of an even number of vertices or $2|E| = \sum v \in V(G) d(v) \geq d(w) + \sum_{i \in [\frac{n}{2}]} k$. In either case we have that $2|E| \geq \frac{kn}{2}$ or namely that $|E| \geq \frac{kn}{4}$ and we are done.

(c) Use a randomized algorithm to prove that there are $O(n^4)$ 3-way cuts of minimum size.
We will show that the probability of finding a specific 3-way min cut is at least $\frac{1}{p(n)}$ where $p(n)$ is some polynomial of degree 4. This can be done using a randomized algorithm very similar to the one covered in class. Namely, we use the same procedure of contracting a random edge until we have exactly 4 vertices remaining. Then, to figure out which edge to contract to get to our final stage in which we have three vertices, we actually try out all 6 (4 choose 2) possible contractions to see which leaves us with the smallest cut. Provided that we have not destroyed our min cut yet, the probability that it is preserved in this last step is 1, since we actually try all possibilities. The multi-edges among the 3 vertices at the end will be our supposed min 3-way cut. The probability that this algorithm finds a specific 3-way min cut is $\prod_{i=1}^{n-4}(1 - \frac{k}{|E(G_{i-1})|})$ where $G_0 := G$ and G_i is the graph obtained by contracting the edge chosen at step i in G_{i-1}. By part (b), we know that $|E(G_i)| \geq \frac{kn}{4}$ where $n := |V(G_i)|$ for all $i \in \{0, 1, \ldots, n-2\}$. So, we see that P(we find a specific 3-way min cut) $\geq \prod_{i=1}^{n-4}(1 - \frac{k}{\frac{k(n-(i-1))}{4}}) = \prod_{i=1}^{n-4}(1 - \frac{4}{n-(i-1)}) = \prod_{i=1}^{n-4}(\frac{n-i-3}{n-i+1}) = \frac{4*3*2}{n(n-1)(n-2)(n-3)} = \frac{1}{\frac{1}{24}n(n-1)(n-2)(n-3)}$. This means that if G has r distinct 3-way min cuts, then the probability of finding any one of them is P(we find any 3-way min cut) $\geq \frac{r}{\frac{1}{24}n(n-1)(n-2)(n-3)}$. Since any probability is always bounded by 1, this means that $1 \geq \frac{r}{\frac{1}{24}n(n-1)(n-2)(n-3)}$ or equivalently $\frac{1}{24}n(n-1)(n-2)(n-3) \geq r$. So, $r = O(n^4)$.

(Bonus) Problem 4 Median-finding algorithm

Suppose in the median-finding algorithm we set the size of the subset R to be n^c for some constant $c > 0$ (the algorithm presented in class has $c = 3/4$). Modify the algorithm for general c (if necessary) and analyze the failure probability and the running time.

We modify the algorithm as follows. First, pick n^c elements from S uniformly at random. Call the set (possibly a multiset) of these elements R. Then, sort R. Let $l := \frac{n^c}{2} - n^{\frac{1}{3}c}$. (Note that I modified the term $n^{\frac{1}{2}}$ given in the original algorithm because if $c < \frac{1}{2}$ such a term would cause problems. Thus, I scaled it in terms of c. A factor of $\frac{1}{3}$ seemed to give appropriate l and r for the examples of c I considered). Then, let $u := \frac{n^c}{2} + n^{\frac{1}{3}c}$. These are indices of certain elements in r we wish to single out. Define $r_l =$ the lth element of R when R is sorted in ascending order. $r_u =$ the uth element of R when R is sorted in ascending order. Now, iterate through S which is likely not sorted, and partition it into $S_{<r_l} := \{s \in S | s < r_l\}$, $S_{\geq l, \leq u} := \{s \in S | r_l \leq s \leq r_u\}$, and $S_{>u} := \{s \in S | s > r_u\}$. Now, we wish to calculate the probability that the median, m, of S falls in the set $S_{\geq l, \leq u}$. How can this fail to happen? This can fail to happen if any of the following events happen. The first event is that m falls in $S_{<l}$. The second event is that m falls in $S_{>u}$. These cases are symmetric, meaning that the probability of either occurring is the same. Event 2 happens when $|\{r \in R | r > m\}| < \frac{n^c}{2} - n^{\frac{1}{3}c}$. We construct a set of variables to count the number of elements in such a set. We set $X_i = 1$ if and only if $r_i \geq m$ and set $X_i = 0$ otherwise. Then, let the variable $Y := \sum_{i=1}^{n^c} X_i$ so that, in particular, $Y = |\{r \in R | r > m\}|$. We wish to calculate $P(Y < l) = P(\sum_{i=1}^{n^c} X_i < l)$. We first calculate the expected value of Y. Namely, we get $E[Y] = \sum_{i=1}^{n^c} E[X_i] = \sum_{i=1}^{n^c} P(X_i = 1) = \sum_{i=1}^{n^c} (\frac{\frac{n-1}{2}+1}{n}) = (n^c)(\frac{\frac{n-1}{2}+1}{n}) = (n^{c-1})(\frac{n+1}{2}) = \frac{n^c + n^{c-1}}{2}$. We now wish to calculate $P(Y < l)$. We notice that $Y < l$ exactly when $-Y > -l$ exactly when $-Y > -l = n^{\frac{1}{3}c} - \frac{n^c}{2} = \alpha(n,c) - E[Y] = \alpha(n,c) - \frac{n^c + n^{c-1}}{2}$ for some function $\alpha(n,c)$ to be determined. We get namely $\alpha(n,c) = n^{\frac{1}{3}c} + \frac{n^{c-1}}{2}$. So, $Y < l$ exactly when $|Y - E[Y]| = E[Y] - Y > \alpha(n,c) = n^{\frac{1}{3}c} + \frac{n^{c-1}}{2}$. So, we apply Chebyshev's Inequality to get $P(Y < l) = P(|Y - E[Y]| = E[Y] - Y > n^{\frac{1}{3}c} + \frac{n^{c-1}}{2}) \leq \frac{Var[Y]}{(n^{\frac{1}{3}c} + \frac{n^{c-1}}{2})^2}$. We calculate $Var[Y] = n^c(\frac{\frac{n-1}{2}+1}{n})(1 - \frac{\frac{n-1}{2}+1}{n}) = \frac{1}{4}n^{c-2}(n^2 - 1)$. So, $P(\text{Event 1}) = P(\text{Event 2}) \leq \frac{n^{c-2}(n^2-1)}{(4n^{\frac{1}{3}c} + 4\frac{n^{c-1}}{2})^2}$. Event 3 is actually defined to be the case in which the size of C where $C := \{s \in S | r_0 \leq s \leq r_{n^c}\}$ is $\geq 4n^c$.