CS 170 HW 13

Daniel Deng, SID 3034543526

1 Study Group

(a) Auston, SID 3034554056

(b) Yes

2 One-Sided Error and Las Vegas Algorithms

(a) Since the randomized algorithm $R(x)$ runs in polynomial time in the size of the input, the number of coin-flips must also be at most polynomial in the size of the input. Therefore, we can modify each instance of the $R(x)$ algorithm as a deterministic $A(x, r)$ where r is a poly-length sequence of coin flips. Now, consider a nondeterministic Turing machine that runs $A(x, r)$ with all possible sequences of coin flips. If the correct answer to the RP instance is "No", then all instances of $A(x, r)$ will return "No". If the correct answer is "Yes", at least half of the instances of $A(x, r)$ will return "Yes". Therefore, since at all times some computation path of the nondeterministic returns an accepting state for RP, $RP \subseteq NP$.

(b) Given that the problem has a ZPP algorithm, we construct a new algorithm that runs the ZPP algorithm on a given input, but terminates after running for a fixed amount of time and return "No" as a default. Under this scheme, if the true answer is "No", then the algorithm will always return "No" correctly; if the true answer is "Yes", then the algorithm will return "Yes" correctly as long as the algorithm completes running in the allotted time. We can find the termination time that guarantees that the algorithm return "Yes" correctly with probability greater than 1/2 using Markov's inequality.

Let X = the runtime of the ZPP algorithm (non-negative), and $\mathbb{E}[X]$ is polynomial in the size of the input. Using Markov's inequality, we have

$$P(X > \lambda) < \frac{\mathbb{E}[X]}{\lambda}$$

Substitute $\lambda = 2\mathbb{E}[X]$, we get

$$P(X > 2\mathbb{E}[X]) < \frac{1}{2}$$

This inequality reveals that as long as we set the termination time to be double the expected runtime, the probability of the algorithm terminated without outputting a "Yes" is less than 1/2, which means that the probability the algorithm returns "Yes" correctly is greater than 1/2.

Therefore, we can always construct a RP algorithm from a ZPP algorithm, and that if a problem has a ZPP algorithm, then it has an RP algorithm.

3 Quick Select

(a) QuickSelect finds the kth smallest element in A. Since X_{ij} is an indicator R.V.

$$\mathbb{E}[X_{ij}] = Pr(X_{ij} = 1)$$

Case 1: $k \leq i < j$. If $p < k$ or $p > j$, k, i, j will be placed on the same side for the next step, and i, j might still be compared. If $k \leq p < i$, i, j will both be placed in the right side and never looked at again. This leaves $i \leq p \leq j$, and we know that i and j are compared only if the pivot is i or j. Therefore, i and j must be selected as pivots in the range $[k, j]$ for them to be ever compared.

$$Pr(X_{ij} = 1 \mid k \leq i < j) = \frac{2}{j - k + 1}$$

Case 2: $i < k < j$: Picking pivot outside $[i, j]$ will keep i, j, k on the same side and continue to be evaluated. Within the range $[i, j]$, i, j will only be compared if either i or j is selected as the pivot.

$$Pr(X_{ij} = 1 \mid i \leq k \leq j) = \frac{2}{j - i + 1}$$

Case 3: $i < j \leq k$. Similar logic to Case 1.

$$Pr(X_{ij} = 1 \mid i < j \leq k) = \frac{2}{k - i + 1}$$

(b)

$$\begin{aligned}
\mathbb{E}[runtime \mid k \le i < j] &= \sum_{i=k}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-k+1} \\
&= 2\left((1)\frac{1}{2} + (2)\frac{1}{3} + \cdots + (n-k)\frac{1}{n-k+1}\right) \\
&= 2\sum_{a=1}^{n-k} \frac{a}{a+1} < 2(n-k) \implies O(n) \\
\mathbb{E}[runtime \mid i < k < j] &= \sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \frac{2}{j-i+1} \\
&= 2\sum_{i=1}^{k-1} \left(\frac{1}{k-i+2} + \frac{1}{k-i+3} + \cdots + \frac{1}{n-i+1}\right) \\
&\quad \text{(know from lecture that } \frac{1}{2} + \cdots + \frac{1}{n} < \ln n) \\
&< 2\sum_{i=1}^{k-1} (\ln(n-i+1) - ln(k-i+1)) \\
&= 2\sum_{i=1}^{k-1} \ln\left(\frac{n-i+1}{k-i+1}\right) \\
&= 2\ln\left(\frac{(n)(n-1)\ldots(n-k+2)}{(k)(k-1)\ldots(2)}\right) \\
&= 2\ln\left(\frac{n!}{k!(n-k+1)!}\right) \\
&< 2\ln\binom{n}{k} \le 2\ln\binom{n}{\frac{n}{2}} \\
&\le 2\ln\left(\frac{en}{\frac{n}{2}}\right)^{\frac{n}{2}} = n\ln(2e) \implies O(n) \\
\mathbb{E}[runtime \mid i < j \le k] &= \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{2}{k-i+1} \\
&= 2\sum_{i=1}^{k-1} \frac{k-i}{k-i+1} < 2(k-1) \implies O(n)
\end{aligned}$$

Therefore, the expected runtime is $O(n)$.

4 Pairwise Independent Hashing

(a) By the pigeonhole principle, if $\mathcal{H}$ has strictly less than m^2 functions, it is impossible to have at least one function correspond to each of the m^2 combinations of values. Therefore, the probability of picking any pair of values is not uniform, and $\mathcal{H}$ cannot be pairwise independent.

(b) If $\mathcal{H}$ is pairwise independent, then the probability of a pair of values being any of the m^2 combinations is uniform. In other words, the probability of two values colliding is exactly $\frac{1}{m^2}$. Since there are m ways to collide, the total probability of two values colliding is $\frac{1}{m}$, indicating that $\mathcal{H}$ is also universal.

(c) (i) Yes. Take the example where $\mathcal{H}$ is an universal hash family that contains exacly 2 hash functions h and h' with domain $\{x_0, x_1, x_2\}$ and range $\{0, 1\}$. Let the mapping for h be $\{1, 0, 1\}$ and the mapping for h' be $\{0, 0, 1\}$. In this case, if the friend gives me x_0, he/she will be able to tell which hash function I was using since the values for x_0 are unique for both. If I was using h, the friend would give x_1, which is guaranteed to collide; if I was using h', the friend would give x_2 and that would guarantee to collide.

(ii) No. Even though the friend knows $\mathcal{H}$, knowing any $h(x)$ only prunes the functions that does not have that $h(x)$ value as a mapping. Since all m^2 pairs are sampled uniformly, knowing one value in any pair will still leave behind m funciton to be chosen from uniformly since H is pairwise independent. Therefore, the probability of finding a collision is strictly $\frac{1}{m}$, and the friend will not be able to obtain a higher probability of collision no matter what variable he/she gives next.

5 Two-level Hashing

(a) Let

$$X_i = \sum_{j=1}^{n} I_{i,x_j}$$

where I_{i,x_j} is the indicator R.V. for $h(x_j) = i$. Then,

$$\begin{aligned}
X_i^2 = \left(\sum_{j=1}^{n} I_{i,x_j}\right)^2 &= \sum_{j=1}^{n} I_{i,x_j}^2 + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} I_{i,x_j} I_{i,x_k} \\
\implies \mathbb{E}[X_i^2] &= \sum_{j=1}^{n} \mathbb{E}[I_{i,x_j}^2] + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \mathbb{E}[I_{i,x_j} I_{i,x_k}] \\
&= \sum_{j=1}^{n} Pr(I_{i,x_j} = 1) + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} Pr(I_{i,x_j} = 1, I_{i,x_k} = 1) \\
&= \sum_{j=1}^{n} \frac{1}{m} + 2\sum_{j=1}^{n-1}\sum_{k=j+1}^{n} \frac{1}{m^2} \qquad \text{(2-wise independent)} \\
&= \frac{n}{m} + 2\sum_{j=1}^{n-1} \frac{n-x}{m^2} = \frac{n}{m} + \frac{n(n-1)}{m^2} \\
\implies \mathbb{E}\left[\sum_{i=0}^{m-1} X_i^2\right] &= \sum_{i=0}^{m-1} \mathbb{E}[X_i^2] = n + \frac{(n)(n-1)}{m}
\end{aligned}$$

(b) **Description:**

1. Construct a level-1 hash table T_1 of size n
2. Randomly choose a hash function $h : [U] \to [n]$ from a 2-wise independent hash family and hash the n inputs and store the hashed values in linked lists (size X_i) in the corresponding bucket i of T_1.
3. If $\sum_{i=1}^{n} X_i > 4n$, clear T_1 and repeat step 2. Otherwise, for each bucket i, construct a level-2 hash table T_2^i of size X_i^2.
4. For each bucket i, randomly choose a hash function $h_i : [U] \to [X_i]$ from an universal hash family and hash the values stored in the linked list. If any values collide, choose another hash function and repeat hashing.

Analysis:

- Memory: T_1 has size n, and all the T_2^i tables combined have size $\sum_{i=0}^{n-1} X_i \leq 4n$. Overall, the 2-level data structure has memory complexity $O(n)$.
- Worst case query time: At worst, a query needs to hash into T_1 and then into T_2^i if there is collision. So there are at most 2 constant time hashing operations and 2 constant time access. Therefore, the worst case query time is $O(1)$.
- Expected preprocessing time: Constructing T_1 takes $O(n)$. Using Markov's Inequality, we know that the probability of the size of the T_2 tables (which relates to the amount of collisions due to hashing with h) is at most $\frac{1}{2}$:

$$Pr\left(\sum_{i=0}^{n-1} X_i^2 > 4n\right) < \frac{\mathbb{E}\left[\sum_{i=0}^{n-1} X_i^2\right]}{4n} = \frac{n + \frac{(n)(n-1)}{n}}{4n} = \frac{1}{2} - \frac{1}{4n} < \frac{1}{2}$$

  Therefore, the expected number of times we need to resample the level-1 hash function is 2. Choosing each hash function is $O(1)$, evaluating all inputs is $O(n)$, and assigning all variables to linked lists is $O(n)$, for an expected constant number of times.

  After that, creating the level-2 hash tables take $O(n)$ since their total size is proportional to n. From lecture and from the inverse birthday paradox, we know that the probability of picking a perfect hashing function for each level-2 hash table is $\frac{1}{2}$ given the size of X_i^2 with respect to the input size of X_i, which means that the hash functions for each sub-table also has an expected number of times to be sampled to be 2. Evaluating n inputs for an expected constant number of times is $O(n)$.

  Therefore, the overall expected preprocessing time is $O(n)$.