VE477 Introduction to Algorithms

Homework 7

Taoyue Xia, 518370910087

2021/09/23

Ex1 — Hash tables

1. First we know that the combination of choosing k keys from total n keys is $\binom{n}{k}$. Also, we know that the probability of a key falling into any slot is equal, so that the probability of choosing one slot for k keys is $(\frac{1}{n})^k$. Moreover, since it follows a binomial distribution, we should take the probability of other keys not falling into the specific slot into account, which is $(1-\frac{1}{n})^{n-k}$. Finally, we can combine them together, and finally get the probability P_k that exactly k keys hashed into a same slot is:

$$P_k = (\frac{1}{n})^k(1-\frac{1}{n})^{n-k}\binom{n}{k}$$

2. We know that each slot will have an equal probability of having k keys, so that the probability of one slot having k keys is nP_k. Since P'_k denotes for the probability of the slot with most keys having k keys, which have some extra restrictions on the former case. Therefore, $P'_k \leq nP_k$.

3. We have Stirling Formula as $n! = \sqrt{2\pi n}(\frac{n}{e})^n$. Thus we will have:

$$\begin{aligned}
P_k &= (\frac{1}{n})^k(1-\frac{1}{n})^{n-k}\binom{n}{k} \\
&= (\frac{1}{n})^k(1-\frac{1}{n})^{n-k}\frac{n!}{(n-k)!k!} \\
&\approx (\frac{1}{n})^k(1-\frac{1}{n})^{n-k}\frac{\sqrt{2\pi n}(\frac{n}{e})^n}{\sqrt{2\pi(n-k)}(\frac{n-k}{e})^{n-k}k!} \\
&= \frac{(n-1)^{n-k}}{n^n}\frac{\sqrt{2\pi n}(\frac{n}{e})^n}{\sqrt{2\pi(n-k)}(\frac{n-k}{e})^{n-k}k!} \\
&= (n-1)^{n-k}\frac{\sqrt{2\pi n}}{\sqrt{2\pi(n-k)}(n-k)^{n-k}e^k k!} \\
&\approx (\frac{n-1}{n-k})^{n-k}\cdot\sqrt{\frac{n}{2\pi k(n-k)}}\cdot\frac{1}{k^k} \\
&< (1+\frac{k-1}{n-k})^{n-k}\frac{1}{k^k} \\
&< \frac{e^k}{k^k}, \text{using the squeeze theorem}
\end{aligned}$$

Proof done.

4. From problem 3, we can simply take the logarithm of both sides and get:

$$\log P_k < k - k \log k$$

Since $k - k \log k$ is strict decreasing when k is increasing, and $k \geq \frac{c \log n}{\log \log n}$, so we can take the least value of k into account, along with $c > 1$, which gives us:

$$\begin{aligned}
\log P_k &< \frac{c \log n}{\log \log n} - \frac{c \log n}{\log \log n} \log(\frac{c \log n}{\log \log n}) \\
&= \frac{c \log n}{\log \log n} - \frac{c \log n}{\log \log n}(\log c + \log \log n - \log \log \log n) \\
&< c \log n [\frac{1}{\log \log n}(1 - \log \log n + \log \log \log n)]
\end{aligned}$$

Set $t = \log \log n$, we can have $\log P_k < c \log n \frac{1 + \log t - t}{t}$, now we can take the derivative of $\frac{1+\log t - t}{t}$. Which is:

$$\frac{d(\frac{1+\log t - t}{t})}{dt} = \frac{t(\frac{1}{t} - 1) - 1 - \log t + t}{t^2} < 0$$

When $t > 1$, which is $n > e^2$, thus we can find that:

$$-1 < \frac{1}{\log \log n}(1 - \log \log n + \log \log \log n) < 0$$

$$\log P_k < -c \log n \quad \Rightarrow \quad P_k < \frac{1}{n^c}$$

Since we have proved in problem 2 that $P'_k \leq n P_k$, therefore,

$$P'_k < \frac{n}{n^c} = \frac{1}{n^{c-1}}$$

Finally, we can find that with $c = 3$, $P'_k < 1/n^2$, proof done.

5. By defining k as $\lfloor \frac{c \log n}{\log \log n} \rfloor$ We can calculate $E(M)$ as:

$$\begin{aligned} E(M) &= \sum_{i=1}^{n} i \cdot Pr(M = i) \\ &= \sum_{i=1}^{k} i \cdot Pr(M = i) + \sum_{i=k+1}^{n} i \cdot Pr(M = i) \\ &\leq \frac{c \log n}{\log \log n} \sum_{i=1}^{k} Pr(M = i) + n \sum_{i=k+1}^{n} Pr(M = i) \\ &= \frac{c \log n}{\log \log n} Pr(M \leq \frac{c \log n}{\log \log n}) + nPr(M > \frac{c \log n}{\log \log n}) \end{aligned}$$

According to the previous problem, when $k \geq \frac{c \log n}{\log \log n}$, the probability of P_k is less than $1/n^{c-1}$, so we can conclude that:

$$E(M) \leq \frac{c \log n}{\log \log n} \cdot 1 + n \cdot \frac{1}{n^{c-1}}$$

Therefore, $E(M) = \mathcal{O}(\frac{c \log n}{\log \log n})$. Proof done.

Ex2 — Minimum spanning tree

Algorithm 1: Update MST

Input : $G = \langle V, E \rangle$ an undirected graph, T the original MST, $e = \langle v, w \rangle$ the edge whose weight decreased

Output: T' the updated MST

```
1  Function findCircle(T, v):
2      Array ← [];
3      origin ← v;
4      q ← queue;
5      push v into q;
6      while q is not empty do
7          cur ← q.pop();
8          cur.state ← visited;
9          if the successor of cur contains origin and the predecessor is not origin then
10             push all the nodes on the path of origin to cur to Array;
11             return Array;
12         end
13         push all the adjacent nodes of cur into q;
14     end
15     return Array;
16 end
17 Function updateMST(G, T, e):
18     T' ← T + {e};
19     Nodes ← findCircle(T', v);
20     Max ← the edge of the highest weight in Nodes;
21     T' ← T' - {Max};
22     return T'
23 end
```

Ex3 — Simple Algorithms

1. Take the number as if it is defined in decimal bits (since in binary bits it is too easy), the pseudocode is shown below:

Algorithm 2: n-bits Integers Addition

Input : Two arrays a and b of two n-bits integers

Output: An array of an n+1-bit integer

```
/* Assume that the bits with low significance is of small index.      */
```

```
1  A ← an n+1-bit array;
2  carry ← 0;
3  i ← 0;
4  for i ← 0 to n − 1 do
5  |   sum ← a[i] + b[i] + carry;
6  |   A[i] ← sum  mod  10;
7  |   carry ← sum/10;
8  end
9  if carry equals 1 then
10 |   A[n] ← carry;
11 end
12 return A;
```

2. a) The pseudocode is shown below:

Algorithm 3: Multiplication by addition

Input : two integers x and y

Output: The multiplication of x and y

```
1 Function mult(x, y):
2     if x = 0 or y = 0 then
3         return 0;
4     else
5         return x · (y mod 2)+ mult(2x, ⌊y/2⌋)
6     end
7 end
```

b) First we can show the whole process by the following formula:

$$\begin{aligned} x \cdot y &= x \cdot (y \bmod 2) + 2x \cdot (\lfloor y/2 \rfloor \bmod 2) + \cdots + 2^n x \cdot (\lfloor y/2^n \rfloor \bmod 2) + 0 \\ &= x \cdot ((y\%2) + 2(\lfloor y/2 \rfloor \%2) + \cdots + 2^n(\lfloor y/2^n \rfloor \%2)) \end{aligned}$$

By converting this expression into binary bits representation, it is very easy to understand. First, by calculating $y \bmod 2$, we can get the least significant bit of y, and multiply it by x. Then, calculating $\lfloor y/2 \rfloor$ means right move y for one bit. Then we get the second bit of y by calculating $\lfloor y/2 \rfloor \%2$. To preserve the correctness, x is multiplied by 2 to meet the requirement of multiplying with y's second bit. Using the previous procedure, We can calculate until the last bit of y is reached. Therefore, the correctness of this algorithm is proved.

Ex4 — Critical thinking

1. I think both algorithms can solve the Knapsack problem, with different usage conditions. But it is optimal to choose the largest item first, so that we need fewer moves to get the correct answer, or raise an error, which saves time.

2. If we choose $m = 2^n$, then $H(K) = K \bmod m$, it's hashed value will be the least significant n bits of K. If two numbers has the same n least significant bits, it is true that they will

create the same hash, which introduce a lot of collision. By introducing a prime which is not close ot power of 2, it will create a lot of divergence in the process of hashing, thus making collisions less probable.

3. Using the knapsack problem in problem 1. Suppose that we have a set of numbers $S = \{1, 5, 7, 9\}$, and the sum $n = 12$. With using the greedy algorithm, it will choose from the largest element smaller than the sum, which will give the locally optimal solution $\{9, 1, 1, 1\}$.
   However, the globally optimal solution is $\{5, 7\}$.

4. If we can determine each horse's racing time in a race, then we just need 5 races to find the fastest three horses, because all the 25 horses' racing time is recorded and obviously observed.

   However, if we cannot know the exact time of racing of each horse, but only knowing the relative speed of each horse, we will need 7 races to determine the three fastest horses.

   Firstly, we need 5 races to test all the 25 horses, namely group A, B, C, D, E. Then, we can take the 5 fastest horses A_1, B_1, C_1, D_1, E_1in each race to compete in the 6-th race. The fastest will be the fastest horse in all the 25 horses, suppose it is A_1. Then the second fastest horse will be chosen between A_2, the second fastest horse in group A and B_1. The third fastest horse will be chosen between A_3, B_2 and C_1. Then take the five horses A_2, B_1, A_3, B_2 and C_1 to compete in race 7. Finally, the two fastest horses in race 7 will be the second and third fastest horses in the total 25 horses.