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 $P'_k \leq nP_k$, 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

```
Function findCircle(T, v):
    Array ← [];
    origin ← v;
    q ← queue;
    push v into q;
    while q is not empty do
        cur ← q.pop();
        cur.state ← visited;
        if the successor of cur contains origin and the predecessor is not origin then
            push all the nodes on the path of origin to cur to Array;
            return Array;
        end
        push all the adjacent nodes of cur into q;
    end
    return Array;
end
Function updateMST(G, T, e):
    T′ ← T + {e};
    Nodes ← findCircle(T′, v);
    Max ← the edge of the highest weight in Nodes;
    T′ ← T′ - {Max};
    return T′
end
```