UM–SJTU Joint Institute
VE477 Intro to Algorithms

Homework 1

Wang, Tianze, 515370910202

Question 1 Hash Tables

(1)

The condition *exactly k keys hash to a same slot* means *exactly k keys hash to a same slot and the rest* $(n-k)$ *keys hash to the other slots*. Then the probability is calculated as

$$P_k = \frac{\text{Combs satisfying the condition}}{\text{All Combs}} = \frac{\binom{n}{k} \cdot 1^{n-k} \cdot (n-1)^{n-k}}{n^n} = \frac{1}{n}^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}$$

The numerator means *choosing k numbers from n which only belongs to one slot, and the rest has totally* $(n-1)$ *spaces to go*. And the denominator means *All combinations for n numbers*.

(2)

The probability of exactly k keys hash to a same slot is P_k. And the probability of exactly one slot has k hash keys is nP_k.

If $k \geq \frac{1}{2}n$, there will be only one slot, so $P'_k = nP_k$. For other cases, namely $k < \frac{1}{2}n$, have the probability $P'_k < nP_k$.

(3)

The Stirling formula is given as

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Then $\binom{m}{n}$ could be represented as

$$\binom{m}{n} = \frac{m!}{n!(m-n)!} = \frac{\sqrt{2\pi m}\left(\frac{m}{e}\right)^m}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi(m-n)}\left(\frac{m-n}{e}\right)^{(m-n)}} = \frac{m^m}{n^n \cdot (m-n)^{m-n}} \cdot \frac{\sqrt{m}}{\sqrt{2\pi n}\sqrt{m-n}}$$

For $\frac{\sqrt{m}}{\sqrt{2\pi n}\sqrt{m-n}}$, we have

$$\frac{\sqrt{m}}{\sqrt{2\pi n(m-n)}}$$

When $m = \frac{1}{2}$, $n = 1$, it exists the biggest value, but it's still smaller than 1. So we could conclude that

$$\frac{1}{\sqrt{2\pi n(m-n)/m}} = \frac{1}{\sqrt{2\pi(n - \frac{n^2}{m})}} < 1$$

Then

$$\binom{m}{n} < \frac{m^m}{n^n \cdot (m-n)^{m-n}}$$

We put this formula back into the definition of P_k,

$$\begin{aligned}
\frac{1}{n}^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k} &< \frac{1}{n}^k \left(1 - \frac{1}{n}\right)^{n-k} \frac{n^n}{k^k \cdot (n-k)^{n-k}} \\
&= \frac{(n-1)^{n-k}}{k^k \cdot (n-k)^{n-k}} \\
&< \frac{n^{n-k}}{k^k (n-k)^{n-k}}
\end{aligned}$$

As $e = (1 + \frac{1}{x})^x$,

$$\begin{aligned}
\frac{n^{n-k}}{k^k(n-k)^{n-k}} &= \frac{1}{k^k} \cdot \frac{n}{n-k}^{n-k} \\
&= \frac{1}{k^k} \cdot \left(1 + \frac{k}{n-k}\right)^{n-k} \\
&= \frac{1}{k^k}((1 + \frac{1}{t})^t)^k, \quad \text{let } t = \frac{n-k}{k} \\
&= \frac{1}{k^k} e^k
\end{aligned}$$

□

(4)

Not done yet.

(5)

We need to construct the k using the result from 1.4. According to the definition of $E[x]$,

$$E[M] = \sum_{i=0}^{n} i \cdot P(M = i)$$

So

$$\begin{aligned}
E[M] &= P(M = 1) + 2P(M = 2) + \cdots + nP(M = n) \\
&= (P(M = 1) + 2P(M = 2) + \cdots + (k-1)P(M = k-1)) + (kP(M = k) + \cdots + nP(M = n)) \\
&\leq (k-1)(P(M = 1) + P(M = 2) + \cdots + P(M = k-1)) + n(P(M = k) + \cdots + P(M = n))
\end{aligned}$$

We now choose k as: $k > \dfrac{c \log n}{\log \log n}$ and $k - 1 \leq \dfrac{c \log n}{\log \log n}$

Then $E[M]$ becomes

$$E[M] \leq \frac{c \log n}{\log \log n} P(M \leq \frac{c \log n}{\log \log n}) + nP(M > \frac{c \log n}{\log \log n})$$

And from the conclusion in 1.4, $P(M > \dfrac{c \log n}{\log \log n}) < \dfrac{1}{n^2}$, thus $E[M]$ becomes

$$\begin{aligned}
E[M] &\leq \frac{c \log n}{\log \log n} P(M \leq \frac{c \log n}{\log \log n}) + n \sum_{i=k}^{n} \frac{1}{n^2} \\
&\leq \frac{c \log n}{\log \log n} \cdot 1 + 1
\end{aligned}$$

Which means

$$E[M] = \mathcal{O}(\frac{\log n}{\log \log n})$$

Question 2 MST

Algorithm 1: Ex. 2, Pseudo code

```
Input  : Graph: $G$, Tree: $T$, The weight-decreased edge that not in $T$: $E$
Output : New Tree T′
Function TreeWeight(T):
    sum ← 0;
    for All edges in T do
        sum ← sum + edge.weight;
    end for
    return sum
end
Function DFS (n):
    mark n as visited;
    for each node u connected to v do
        if u is not visited then
            DFS (u)
        end if
    end for
end
Function Is_Connected (G):
    choose any one of the node in G as n;
    DFS (n);
    if All the nodes u in G are visited then
        return True
    else
        return False
    end if
end
Function MST_New(T, G, E):
    src ← E.begin;
    dst ← E.end;
    T′ ← T;
    for All edges in T connected to src and dst, denoted as e do
        T_temp ← T′ remove e add E;
        /* Here I directly use the function 'Find()' from course slides c1.38 */
        if e.length > E.length and Is_Connected(T_temp) and Find(e.src) $\neq$ Find(e.dst) then
            if TreeWeight(T_temp)<TreeWeight(T′) then
                T′ ← T_temp;
                e_before ← e;
            end if
        end if
    end for
    return T′
end
```

Question 3 Simple Algorithms

(1)

Not so hard, leave blank first.

(2)

1.

Algorithm 2: Ex. 3, Pseudo code

```
Input  : two integers x,y
Output : Their multiplication
Function mult(x,y):
    if x = 0 or y = 0 then
        return 0
    else
        return mult(2x, $\lfloor y/2 \rfloor$) + x· (y mod 2)
    end if
end
```

2.

The essence of this algorithm is to carry out the multiplication algorithm in **binary** form.

For two integers x and y, we transform into two arrays A and B, each element of the array corresponds to each digit of the binary number.

Then we see array A as a keep shifting one and we use it as a whole, and array B as an indicator to tell when to multiple, we use it bit by bit. The mult(*2x,* $\lfloor y/2 \rfloor$) operation means array A shift left one digit and array B shift left one digit. The $i-th$ calculation means calculating the former sum plus the value at $i-th$ digit of B. Then we need to read the current digit of array B, which is *y mod 2*, if it's 1, it means their will be a multiplication, otherwise there will be none. And multiply by the shifted array A, which is $x \cdot ($y mod $2)$.

Question 4 Horse Race Problem

(1) Algorithm

The minimum number of races is 7. It is described as follows.

First separate 25 horses into 5 groups A, B, C, D, E, and race inside these groups, this will count to **5** races. This will result into an sorted array $A[1]$ to $A[5]$, etc. The smaller index means the faster.

Then choose the fastest horses from each group, namely $A[1], B[1], C[1], D[1], E[1]$, and let them race, count to **1** race. (We denote this group as R, smaller index means the faster)

Finally, We choose $\underline{X[2]}$ and $\underline{X[3]}$, where $R[1] \in X, X = \{A, B, C, D, E\}$,
and we choose $\underline{R[2]}$ and $\underline{Y[2]}$, where $R[2] \in Y, Y = \{A, B, C, D, E\}$,
and $\underline{R[3]}$.
Let them race. (We denote this group as S, smaller index means the faster)

Then we take $R[1]$ and $S[1]$ and $S[2]$. These are the fastest horses. □

Algorithm 3: Ex. 4, Pseudo code

```
Input   : 5 horse
Output  : An array X[1:5], smaller index means faster horse
Function Race(5 horses):
    Five horse races;
    return X
end
Input   : 25 horses
Output  : The fastest among them
Function Three_Fastest_Horses(25 horses):
    Randomly separate them into 5 groups $A, B, C, D, E$, each with 5 horses;
    for i in $\{A, B, C, D, E\}$ do
        i ← Race(i);
    end for
    R ← Race(A[1], B[1], C[1], D[1], E[1]) ;
    X ← $X \in \{A, B, C, D, E\} \cap R[2] \in X$;
    Y ← $Y \in \{A, B, C, D, E\} \cap R[3] \in Y$;
    S ← Race(X[2], X[3], R[2], Y[2], R[3]) ;
    return R[1], S[1], S[2]
end
```

(2) Correctness

The correctness of this method: $R[1]$ is obviously the fastest, so the hard point is to get the second and third fastest. So we need to use one race to tell the second and third fastest. In the X group as defined before, at most 3 can be the top-3 fastest, in the Y group as defined before, at most 2 can be the top-3 fastest, since $R[1]$ is faster than any one of Y. In the group $R[3]$ belongs to, only one can be chosen, since $R[1]$ and $R[2]$ are faster than any one from them.

Question 5 Critical Thinking

(1) Knapsack problem

Fit the knapsack with the largest items first. This is because putting totally same weight of small items need more operations than big items. Which means if we put small items first, we will need more time on our trial till we get an error.

(2) Hash number

The choice of hash number is to avoid hash collision, or to decrease the possibility of the chance it happens.

(3) Greedy non globally optimal

A easy example is shown as making changes. Suppose we have 4 kinds of coin, \$1, \$5, \$10, \$12, and we wish to make a change of \$15. According to greedy algorithm, we will try to go from the most to the least, so we will use

1 \$12 and **3** \$1 coins. However, this use more than **1** \$10 and **1** \$5. In this case, we reach locally optimal via the greedy method, however, it's not globally optimal.