# Sample Solution for Problem Set 6

Data Structures and Algorithms, Fall 2020

December 11, 2020

## Contents

# 1 Problem 1

Choose one of the $m$ spots in the hash table at random. Let $n_k$ denote the number of elements stored at $T[k]$. Next pick a number $x$ from 1 to $L$ uniformly at random. If $x < n_j$, then return the $x$th element on the list. Otherwise, repeat this process. Any element in the hash table will be selected with probability $1/mL$, so we return any key with equal probability. Let $X$ be the random variable which counts the number of times we must repeat this process before we stop and $p$ be the probability that we return on a given attempt. Then $E[X] = p(1 + \alpha) + (1 - p)(1 + E[X])$ since we'd expect to take $1 + \alpha$ steps to reach an element on the list, and since we know how many elements are on each list, if the element doesn't exist we'll know right away. Then we have $E[X] = \alpha + 1/p$. The probability of picking a particular element is $n/mL = \alpha/L$. Therefore, we have

$$\begin{aligned} E[X] &= \alpha + L/\alpha \\ &= L(\alpha/L + 1/\alpha) \\ &= O(L(1 + 1/\alpha)) \end{aligned}$$

since $\alpha \leq L$.

## 2 Problem 2

**(a)** We will show that each string hashes to the sum of it's digits $\mod 2^p - 1$. We will do this by induction on the length of the string. As a base case, suppose the string is a single character, then the value of that character is the value of $k$ which is then taken $\mod m$. For an inductive step, let $w = w_1 w_2$ where $|w_1| \geq 1$ and $|w_2| = 1$. Suppose $h(w_1) = k_1$. Then, $h(w) = h(w_1)2^p + h(w_2) \mod 2^p - 1 = h(w_1) + h(w_2) \mod 2^p - 1$. So, since $h(w_1)$ was the sum of all but the last digit $\mod m$, and we are adding the last digit $\mod m$, we have the desired conclusion.

**(b)** $0$ for empty slots.

- Linear probing: [22, 88, 0, 0, 4, 15, 28, 17, 59, 31, 10]
- Quadratic probing: [22, 0, 88, 17, 4, 0, 28, 59, 15, 31, 10]
- Double hashing: [22, 0, 59, 17, 4, 15, 28, 88, 0, 31, 10]

# 3 Problem 3

Proof by contradiction:

**(1)**

Suppose that

$$\epsilon < \frac{1}{|B|} - \frac{1}{|U|}.$$

This means that for all pairs $k, l$ in $U$, we have that the number $n_{k,l}$ of hash functions in $\mathcal{H}$ that have a collision on those two elements satisfies

$$n_{k,l} \leq \epsilon|\mathcal{H}| < \frac{|\mathcal{H}|}{|B|} - \frac{|\mathcal{H}|}{|U|}.$$

So, summing over all pairs of elements in $U$, we have that the total number $N$ satisfies

$$N = \sum_{l,l\neq k}^{U}\sum_{k}^{U} n_{k,l} < \frac{|\mathcal{H}||U|^2}{2|B|} - \frac{|\mathcal{H}||U|}{2}.$$

**(2)**

For any hash function $h$, the number of conflicting pairs, $n_h$, is minimized if and only if each slot contains an equal number of keys. (Proved in the appendix) In other words, $n_h$ satisfies

$$n_h \geq |B|\binom{|U|/|B|}{2} = |B|\frac{|U|^2 - |U||B|}{2|B|^2} = \frac{|U|^2}{2|B|} - \frac{|U|}{2}$$

Summing over all hash functions, we get that the total number $N$ satisfies

$$N = \sum_{h}^{\mathcal{H}} n_h \geq |\mathcal{H}|\left(\frac{|U|^2}{2|B|} - \frac{|U|}{2}\right)$$

which is contrary to the conclusion of **(1)**. Therefore $\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|}$.

**(appendix)**

Let $b \equiv |B|$, suppose that the number of keys contained in each slot is $n_1, n_2, \ldots, n_b$. There is $\sum_{i=1}^{b} n_i = |U|$. The total number of conficting pairs is

$$\begin{aligned}
\binom{n_1}{2} + \cdots + \binom{n_b}{2} &= \frac{1}{2}\left(n_1^2 - n_1 + \cdots + n_b^2 - n_b\right) \\
&= \frac{1}{2}\left[\left(n_1^2 + \cdots + n_b^2\right) - (n_1 + \cdots + n_b)\right] \\
&= \frac{1}{2}\left[\left(n_1^2 + \cdots + n_b^2\right)\left(\frac{1}{\sqrt{b}}^2 + \ldots \frac{1}{\sqrt{b}}^2\right) - (n_1 + \cdots + n_b)\right] \\
&\geq \frac{1}{2}\left[\left(\frac{1}{\sqrt{b}}n_1 + \cdots + \frac{1}{\sqrt{b}}n_b\right)^2 - (n_1 + \cdots + n_b)\right] \\
&= \frac{1}{2}\left(\frac{1}{|B|}|U|^2 - |U|\right) = |B|\binom{|U|/|B|}{2}
\end{aligned}$$

By cauchy's inequality, the equal sign is true if and only if $n_1 = n_2 = \cdots = n_b$.

# 4 Problem 4

## (implement)

$S$ is an array. Initially $S$ is empty and $S.size = 0$.

**Algorithm 1:** INSERT$(S, x)$

```
1 S[S.size] ← x;
2 S.size ← S.size + 1;
```

**Algorithm 2:** DELLARGEHALF$(S)$

```
1 QUICKSELECT(S, ⌊S.size/2⌋);
2 S.size ← ⌊S.size/2⌋;
```

## (complexity)

- Charge 2 dollar for INSERT$(S, x)$
- Cost $S.size$ dollar for DELLARGEHALF$(S)$
- There are at least $2 \times S.size$ dollar deposits. **(*)**
  - (*) is true when $S.size = 0$.
  - After INSERT$(S, x)$, $S.size$ increase 1, deposits increase 2. (*) is true.
  - After DELLARGEHALF$(S)$,
    deposits: at least $2 \times S.size_{old} - S.size_{old} = S.size_{old}$;
    $S.size_{new}$: less than $\frac{1}{2} \times S.size_{old}$.
    (*) is true.

# 5 Problem 5

Suppose the binary string is $B$, with the lowest digit $B[0]$. We use the following algorithm to perform a single INC operation:

1 Initialize $i = 0$.

2 **While** $B[i] == 1$ **do**

- $B[i] \leftarrow 0$.
- $i \leftarrow i + 1$.

3 $B[i] \leftarrow 1$

Define the potential function $\Phi_i$ as the number of 1 in $B$ before the $i$-th INC operation. Suppose in the $i$-th INC operation, the **While** loop in step 2 is executed for $n_i$ times, obviously we have $\Phi_i - \Phi_{i+1} = n_i - 1$. The running time for the $i$-th INC operation is $c_0 n_i + c_1$ for some constant $c_0, c_1$. Thus, the running time for $n$ INC operations is

$$\sum_{1 \le i \le n} (c_0 n_i + c_1) = c_0 \left( \sum_{1 \le i \le n} (\Phi_i - \Phi_{i+1}) + n \right) + c_1 n \le c_0(\Phi_1 + n) + c_1 n$$

Recall that $\Phi_1 = b$ and $n = \Omega(b)$, we have $c_0(\Phi_1 + n) + c_1 n = O(n)$.

# 6 Problem 6

Suppose the stack is empty and has size 1 initially. Strictly speaking, we will analysis the running time for any execution combinition of $n$ POP and PUSH operations, and justify that the cost is $O(n)$.

**(a)** No. For any $i$ and let $n = 2^i$, Consider the following execution: $n$ times of PUSH, followed by $n$ times of PUSH, POP pairs. One can varify that after $n$ times of PUSH the size of the stack is exactly $n$, and each PUSH, POP pair will cost $\Omega(n)$ time since the stack will double for each PUSH and shrink for each POP. Thus, $3n$ times of operations cost $\Omega(n^2)$ times.

**(b)** Yes. Suppose store 3 coins for each $POP$ and $PUSH$ operation, we will prove we always own non-negetive number of coins at any time if we use one coin for each time cost. We prove it by induction. Initially we have zero coin. If before the $i$-th operation, we always have non-negetive number, now consider the $i$-th operation: if it do not cause an expansion or shrinking, the number of coins will increase by 2. Otherwise:

It is a $PUSH$ operation and cause a expansion from $n$ to $2n$. If the last change of the stack is from $\frac{n}{2}$ to $n$, then $\frac{n}{2} * 2$ coins must be saved, thus still non-negetive. If the last change of the stack is shrink from $4n$ to $n$, consider the former change of the stack, if it is from $16n$ to $4n$, then there should be $3n * 2$ coins saved, which means it is non-negetive. Otherwise it is from $2n$ to $4n$, then there must be $n * 2$ coins save, which means it is still non-negetive.

It is a $POP$ operation and cause a shrinking from $4n$ to $n$, consider the last change of the stack, if it is from $2n$ to $4n$, then there are at least $2n$ elements and $n * 2$ coins must be saved. Otherwise it is from $\frac{n}{2}$ to $n$, in which case there are $\frac{n}{2} * 2$ coins save. In both case the coins are non-negetive.

**(c)** Yes. Similar to (b).

# 7 Problem 7

## (a)

We will prove that the amortized time complexity for this case is $O(1)$.

Note that $c_i = k+4$, where $i = 2^k * m$ for some odd integer $m$. Let $\Phi(x) =$ number of 1s in the array after $x$-th operat

Therefore, $\hat{c_i} = c_i + \Phi(i) - \Phi(i-1) = 4 + 2 *$ number of bits changes from 0 to $1 \leq 6$.

## (b)

We will prove that the amortized time complexity for this case is $O(\log n)$.

Note that $c_i = k + 2^k$, where $i = 2^k * m$ for some odd integer $m$. WLOG, we may assume that $c_i = 2^k$ to simplify our analysis. Let $\Phi(x) = x\lceil \log n \rceil - \sum_{i=1}^{x} c_i$, and the following observations concludes our proof.

- $\Phi(x) \geq \Phi(0) = 0$. It follows from the fact that

$$\sum_{i=1}^{x} c_i \leq \sum_{k=0}^{+\infty} 2^k \left\lfloor \frac{x}{2^k} \right\rfloor \leq x\lceil \log n \rceil$$

- $\hat{c_i} = c_i + \Phi(i) - \Phi(i-1) = \lceil \log n \rceil$

## (c)

We will prove that the amortized time complexity for this case is $O(n)$.

WLOG, we will assume that $c_i = 4^k$, where $i = 4^k * m$ for some odd integer $m$. Let $\Phi(x) = 4xn - \sum_{i=1}^{x} c_i$, let's prove the following

- $\Phi(x) \geq \Phi(0) = 0$. It follows from the fact that

$$\sum_{i=1}^{x} c_i \leq \sum_{k=0}^{+\infty} 4^k \left\lfloor \frac{x}{2^k} \right\rfloor \leq x \sum_{k=0}^{\lceil \log n \rceil} 2^k \leq 4xn$$

- $\hat{c_i} = c_i + \Phi(i) - \Phi(i-1) = 4n$