COMP20007 Design of Algorithms
Semester 1, 2016

Assignment 2 – Hashing Analysis

Geoffrey Law (Student ID: 759218)

1.

Suppose the hash function hashes everything to the same bucket. Assume every bucket is a linked data structure. For insert, assume the item should insert to the head of the linked structure, so it is constant time $O(1)$. For search, since the hash function returns the same value, consequentially the bucket that hashed to would contain all items, so it is necessary to iterate through all node in the linked structure for searching in linear time i.e. $O(n)$.

2.

Suppose the hash function spreads the input perfectly evenly over all the buckets. Assume n < size, the hash function returns the same value and the collision resolution method is linearly search for bucket that is empty. For insert, since the assumption, we need to iterate the entire table then determine whether the bucket is empty, so it takes $O(size)$. For search, it takes $O(size)$, since the assumption, we have to iterate through each bucket to find the item;

3.

Suppose that the hash function never hashes two different inputs to the same bucket. Assume n < size and the hash function returns different value for different item. For insert and search, they both take constant time $O(1)$, since each bucket is unique for each distinct item.

4.

The bad hash function is used random integer as constant and multiplies by the first character of key to calculate the hash value. So the bad hash function would return the same hash value if the first character of two strings are the same. For example, strings “Apple” and “Australia” will hash to the same bucket, so the hash value only has 52 variations (total number of lowercase and uppercase English letter). Therefore, it is bad for the number of items is greater than 52.

5.

For the universal hash function, assume that r0, r1,… are chosen randomly and independently of the list of inputs and n < size. Theoretically, the universal hash function should end up with all items uniformly distributed in the hash table. So, in the best case, expected the universal hash function would behave perfectly which should returns a unique hash value for different items. Therefore, based on the expectation, both insert and search functions should only take constant time $\theta(1)$ in average.

6.

The collide_dumb algorithm is inspired by the permutation of a string, which allows us to permute all characters in a given string. Since we need to enumerate distinct strings as much as we can, so permutation method is a good way to achieve that. In order to make the function can generate more strings, the permutation function has modified differently such that the permutation combined with combination method to find all combinations of character for a given length. For example, we have string “abcd”, permutation should be “acbd”, “abdb”… once the function modified with the combination method, it should come up “a”, “b”, “c”, “d”, “ab”, “ac”… which can generate more strings than original permutation. The function using recursion within character swaps, so we can test the string whether hashes to zero while the function is recurring and backtracking. The string that we are going to permute is contain all 94 printable ASCII characters ranged 33 to 126, and there is a variable ‘k’ started from 1 to 94 which the length of string that we want to permute. For example, k = 1 should generate “a”, “b”, “c”, “d”, k = 2 should generate “ab”, “ac”, ”ad”…, once finished generating with some k, then increment k by 1. Therefore, the complexity is $O(n * string\_length * k!)$, in our case n is number of collisions and the string length is 94, which is not quite efficient when k is large. However in practice, k would not be greater than 4 even if n is extremely large, so number of random has chosen to be 8 just in case.

7.

Since the performance of extended Euclid algorithm (EEA), we consider only two cases; one character string and two-character string. For one-character strings, it is not necessary to solve by EEA. In our case, size is always a prime number, i.e. $r_0 * s_0 + size * (k) = 1$, so the one character s_0 must be multiple of size in order hash to 0 and has no exception for this case since every single element of r is always less than $size$. So, in one-character case, iterate i started from 0, check whether $size + size * i$ is within the range [33, 126], if it does then print it out, the loop breaks when it is out of range. So the running time for this case is $O(\frac{ASCII_{upperbound}}{size})$ where ASCII upper bound is 126, we can assume that is almost constant $O(1)$. For two characters strings, I deduced a formula to solve this case, which is choose the first character s_0 from 33 to 126, then substitute in this formula to find the second character s_1.

There exist i and j are member of positive integer

$$s_1 = [r_0 s_0 x_o - (\max\{|33 - r_0 s_0 x_o|, |126 - r_0 s_0 x_o|\} - i)] + j$$

such that

$$(\max\{|33 - r_0 s_0 x_o|, |126 - r_0 s_0 x_o|\} - i) \ mod \ size = 0 \text{ and } 33 \leq s_1 \leq 126$$

where x_0 is one solution of x from EEA $gcd(r_1, size)$ i.e. $ax + by = 1$

(the proof of this formula is on next page)

The complexity is almost constant $O(1)$ for generating one string, since this algorithm only need to call EEA once and we restricted some of the value must be within the bound [33, 126] even though i and j could be go very large but the function is designed if they are out of range then the loop will break. This algorithm cannot deal with the r value that is equal to zero. However, I made another function called “check_zero” which behaves differently when dealing with zero r. Overall, collide_clever is expected in constant running time $O(1)$ for one string, in order to find n strings, the running time is going to be $O(n)$.

Given $ax+by = \gcd(a,b) = 1 \Rightarrow (a < b) \wedge (b \text{ is a prime})$

Claim. $r_0 \cdot S_0 + r_1 \cdot S_1 = size \cdot k$, $k \in \mathbb{Z}$

choose S_0 in range $[33, 126]$

solve S_1 and k, in this case we only consider S_1

Proof. $r_0 \cdot S_0 + r_1 \cdot S_1 = size \cdot k$

$\Rightarrow r_1 \cdot S_1 - size \cdot k = -r_0 \cdot S_0$

To make it solvable,

$r_1 \cdot S_1 + size(-k) = 1 = \gcd(r_1, size)$

Than use extended euclid algorithm to solve S_1 and $-k$ and let x_0 be one solution to S_1 and y_0 be one solution to $-k$

$r_1(x_0) + size(y_0) = 1$

$\Rightarrow r_1(-r_0 \cdot S_0 \cdot X_0) + size(-r_0 \cdot S_0 \cdot X_0) = -r_0 . S_0$

$\Rightarrow r_1(r_0 S_0 X_0) + size(r_0 S_0 X_0) = r_0 S_0$

By definition of linear diophantine equation, the general solution is

$S_1 = r_0 S_0 X_0 + size \cdot c$, ~~$-k = r_0 S_0 X_0 - r_1 \cdot c$~~ , $c \in \mathbb{Z}$

No need to solve $-k$ in this case, then X_0 must be positive (take absolute)

Since the character S_1 should be a printable character, then

$$33 \leq S_1 \leq 126$$

$$\Rightarrow 33 \leq r_0 S_0 X_0 + size \cdot c \leq 126$$

$$\Rightarrow 33 - r_0 S_0 X_0 \leq size \cdot c \leq 126 - r_0 S_0 X_0$$

Consider $r_0 S_0 X_0 - P = S_1$, need to find the greatest value p within the bound.

In case of negative bound,

$$\Rightarrow \min\{|33 - r_0 S_0 X_0|, |126 - r_0 S_0 X_0|\} \leq size \cdot c \leq \max\{|33 - r_0 S_0 X_0|, |126 - r_0 S_0 X_0|\}$$

Find p, $\exists i \in \mathbb{Z}^+$, $P = \max\{|33 - r_0 S_0 X_0|, |126 - r_0 S_0 X_0|\} - i$

s.t. $P \bmod size = 0$

Once we found p, calculate $S_1' = r_0 S_0 X_0 - P$ is one solution for S_1, than $\exists j \in \mathbb{Z}^+$

$S_1 = S_1' + j$ s.t. $33 \leq S_1 \leq 126$

Hence, combined all, thus to obtain the formula

$\exists i, j \in \mathbb{Z}^+$ $S_1 = [r_0 S_0 X_0 - (\max\{|33 - r_0 S_0 X_0|, |126 - r_0 S_0 X_0|\} - i)] + j$

s.t. $(\max\{|33 - r_0 S_0 X_0|, |126 - r_0 S_0 X_0|\} - i) \bmod size = 0 \wedge 33 \leq S_1 \leq 126$.