HW2 Hashing

B12508026 戴偉璿

May 8, 2025

Part A

我使用的 hash 方法是 middle square method，這個方法的原理是將數字平方後取中間的四位數作為 hash 值。這可以避免資料連續時有過多的碰撞。當碰撞發生後，我使用的 probing 方法是混合 quadratic probing 以及 linear probing，在使用 quadratic probing 發生兩次碰撞（secondary clustering）之後就改為使用 linear probing，這樣可以避免產生過大的 secondary clustering，但如果在相同的部位發生太多次的碰撞，仍然可能會有 primary clustering 的問題。

發生 secondary clustering 的位置在 id=18（資料 11508011、12508729 會有相同的 hash 值）、id=40（資料 11508863、11508599 會有相同的 hash 值）。在 table_size 為 59 的時候僅發生這兩次碰撞。

計算方法為 C++ 程式

Part B

這個方法在所有的資料中僅發生了兩次的碰撞，已經算是極低，難以再想出新的優化方法。

程式碼

```cpp
#include<bits/stdc++.h>
#define int long long
#define eb emplace_back
using namespace std;

const int N=1000000, table_size=59, max_probing_times=2;

int arr[25]={10102109, 10106402, 10106918, 12508729, 12508629, 12508765,
             12508068, 12508705, 12508842, 11508011, 11508817, 11508388,
             11508189, 11508331, 11508675, 11508521, 11508287, 11508863,
             11508979, 11508532, 11508035, 11508599, 10613285, 13945978,
                12945157};

int hash_table[table_size];

//i:原始資料的位置, id:hash table中的位置, x:探查的次數
inline void quandratic_probing(int i, int id, int x){
    while(x>max_probing_times){
        if(id>=table_size)id%=table_size;
        if(!hash_table[id]){
            hash_table[id]=arr[i];
            return;
```

```
        }
        ++id;
    }

    if(!hash_table[id]){
        hash_table[id]=arr[i];
        return;
    }
    id=(id+x*x)%table_size;
    quandratic_probing(i, id, x+1);
}

int32_t main(){
    for(int i=0;i<25;++i){
        int id=((arr[i]*arr[i]/N)%N)%table_size;
        if(hash_table[id])quandratic_probing(i, id, 1);
        else hash_table[id]=arr[i];
    }

    for(int i=0;i<table_size;++i){
        cout<<i<<":"<<hash_table[i]<<'\n';
    }

}
/*
作者：戴偉璿
日期：2025/05/01
說明：

使用的方法為mid-square，處理collision的方法為quandratic

為了避免一直找不到空位導致遞迴次數過多而stack overflow，因此設置一個最大探查次數(max_probing_times)，一旦超過這個數值則進行linear probing，由於table size大於資料數量，因此每個資料都能找到位置
*/
```