演算法概論 HW1

109550030 李寀菱

Environment

DEV-C++ 5.11

Results

1. Method

假幣問題要求使用一個只能比較兩側重量之天平來找出混雜在真幣之中的假幣，由於硬幣總數為 n 個未知數，因此考慮使用 Divide-and-Conquer 的演算法設計解題方法。

首先考慮將所有硬幣分做兩堆，若偶數則平分、若為奇數則將多餘的一個加入其中一堆再比較重量，按照公式、這類演算法之時間複雜度可以表示為

$$T(n) = \begin{cases} T(n/2) + a, & n \geq c \\ b, & n < c \end{cases}$$

而 T(n) = 2T(n/2) +a = [T(n/4) +a] +a = [T(n/8) +a] +2a = ⋯ = T(1) + a*$\log_2 n$ = b + a*$\log n$ = O($\log n$)。

（或者使用 Master Method 可得）T(n) = T(n/2) +k，此時令 a=1，b=2，f(n)=1，則 $n^{\log_2 1}$=1，f(n)=Θ($n^{\log_2 1}$)=Θ(1)，因此可知 T(n)= O($\log n$)。

此時考慮使 log 之底數加大為 3、再重新計算演算法，此次將所有硬幣分為個數分別為$\lceil n/3 \rceil$、$\lceil n/3 \rceil$及 n-2*$\lceil n/3 \rceil$的三堆分組，此時演算法之時間複雜度可以表示為

$$T(n) = \begin{cases} T(n/3) + a, & n \geq c \\ b, & n < c \end{cases}$$

而 T(n) = T(n/3) +a = [T(n/9) +a] +a = [T(n/27) +a] +2a = ⋯ = T(1) + a*$\log_3 n$ = b + a*$\log_3 n$ = O($\log n$)。

2. Solution

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>


/* run this program using the console pauser or add your own getch, system("pause") or input loop */

using namespace std;

ifstream inFile;
ofstream out;
void getData();
int getSum(vector <int>sum, int l, int r);
int FindFakeCoin(vector <int>data, int l, int r);


void getData(){
    inFile.open("input.txt");/*open input file*/
    out.open("output.txt");

    if(!inFile){/*if the file broken*/
        cout<<"Open file failed."<<endl;
        exit(1);
    }

    int num;
    while(inFile >> num){/*how many coins*/
        vector <int> weight;/*create a vector and empty it*/
        weight.clear();
```

首先使用 getData 這個函式來開啟文字檔案、讀取硬幣重量，且注意要求為直到檔案結尾為止不斷迴圈讀入；為了測試方便，若檔案損毀或其他原因導致文字檔案沒有開啟，將會顯示提示訊息。

接著讀入每一次共有多少硬幣，為了後續操作方便，將資料記入 weight 這個 vector 之中，並在使用前清空整個 vector。

```cpp
        int times = num;
        while(times--){
            int coin;
            inFile >> coin;
            weight.push_back(coin);/*get coins into vector*/
        }

        //for(auto i: weight){/*print the vector*/
        //  cout << i << " ";
        //}
        //cout << endl;

        int fake_pos = FindFakeCoin(weight, 0, num-1);/*find fake coin*/
        cout << fake_pos << endl;
        out << fake_pos << endl;/*output the file*/
    }

    inFile.close();
    out.close();
}
int getSum(vector <int>sum, int l, int r){/*get sum*/
    int i, weightsum = 0;
    for (i = l;i <= r;i++){
        weightsum += sum[i];
    }
    return weightsum;
}

int FindFakeCoin(vector <int>data, int l, int r){
```

依次讀入每個硬幣的重量，並呼叫 FindFakeCoin 這個函式、找出假幣位置。

在 FindFakeCoin 中，傳入的為包含假幣的一堆硬幣之 vector，以及兩邊末端的位置，首先利用此兩個參數計算出此 vector 中有多少元素，接著處理個數為 1 和個數為 4 之特殊情況。

若為 1 則直接回傳位置，若硬幣個數為 4 個，將會由於後段使用的 ceiling 函數導致其分堆為 2、2、0 而無法計算平均值，因此如果傳入的硬幣個數只有 4 個則會直接找出假幣。

處理完特殊情況後，使用 ceiling 函數來決定每堆硬幣中有多少的個數，分別為 $\lceil n/3 \rceil$、$\lceil n/3 \rceil$及 n-2*$\lceil n/3 \rceil$個，並將三堆硬幣各自加總起來求取平均值。

得到平均值後互相比較，其數值與另外兩組不同的一堆中必定包含假幣，將包含假幣的這一堆硬幣呼叫 recursive function，並且要注意額外處理當一堆硬幣中只剩下 2 個回傳的狀況。

由於以 3 為底的處理方式，使得硬幣最終必定剩下最少 1 個、最多 3 個，這兩種情況已經在先前的步驟裡處理妥當，此時只須注意 2 個硬幣的情形。由於在比較硬幣重量時，假幣究竟是較重或者較輕這點不得而知，因此沒有辦法只憑 2 個硬幣就判斷哪個是假幣，此時其中一個解決方法是多取一個鄰近的硬幣、使整堆硬幣的個數變為 3 個。因為所有硬幣中僅存在一個假幣，因此任選的另一個硬幣必為真幣，此時依 3 個硬幣的處理方式便能找出假幣。

3. Running time

為了計算方便，將整個程式當作只有一個 case 來計算 running time，但實際上檔案中可能會有多於一個以上的測試 case。

(1)void getData：O(n)+running time of FindFakeCoin function

(2)int getSum：O(n)

(3)int FindFakeCoin：O(1)+3O(n)+O(1)+6T(n/3)

經過分析可以得知 running time 總計為 O(n)+ O(1)+3O(n)+O(1)+6T(n/3)= 4O(n)+6T(n/3)= O(n)+O($\log n$)=O(n)

4. 心得

第一次的演算法作業應該還算得上是簡單，由於是藉由天平來尋找硬幣這樣的問題，有了明顯的提示因此很快便知道了該使用 Divide-and-Conquer 的思路來設計演算法，不過因為自己一開始過於先入為主的觀念，所以把整個程式寫成了尋找較輕的假幣，等到寫完準備完成報告的時候才發現題目沒有說明假幣的重量，於是只好把判斷假幣的部分全部打掉重練了。

好在其實更改的部分也並不太難，只是判斷一堆只剩下兩個硬幣的狀況讓我稍微花費了一點時間，之後開始計算時間複雜度後，我在網路上搜尋的資料也提到能夠將硬幣分做三堆來加快計算速度，雖然我查找的資料並沒有明說，不過我想繼續放大底數雖然也能加快運算速度，但也相對地會製造更多麻煩吧，因此也就沒有嘗試實作了，不過底數為 3 的 Divide-and-Conquer 比底數為 2 加速了大約 1.6 倍、若是做成底數為 5 則可以加快 2.3 倍左右。

總而言之，這份作業讓我也對時間複雜度有了更深的認識，也讓我覺得自己對演算法有了更進一步的了解。