計算機概論 搜尋法 AI 實作

電機一甲 11428109 潘宇綸

一、前言

此題目是探討二分搜尋法(Binary Search)與偏向切分搜尋法(Biased Search)之間的差異，與其運行效率，並使用 AI 協助生成程式碼進行驗證，讓學生了解基本演算法及搜尋法概念。

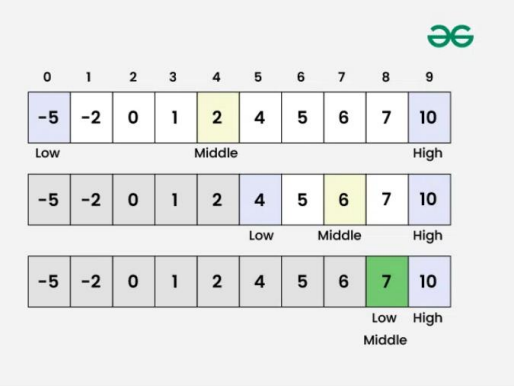
二、研究方法

1. 使用 AI 協助進行程式碼生成，參考網路資源了解數學意義上的搜尋演算法，學習時間複雜度等問題。製作一個程式可以調控 N 之大小，並顯示其時間複雜度及平均測試步驟次數，探討兩搜尋法之差異。
2. 研究工具
   - i. Google Gemini AI – 3 Pro
   - ii. C++ 17 in Visual Studio Code
   - iii. Copilot GPT-5 mini in Visual Studio Code
   - iv. 網路資源

三、研究步驟

1. 二分搜尋法的粗略介紹

所謂的二分搜尋法，對大部分有寫過程式的人一定不陌生，因為這個演算法是幾乎大部分的新手會遇到的課題，常常出現在中學的電腦課程中當作考試題目來應用。二分搜就粗略的敘述，就是將一筆排列後的資料，透過比較目標(target)與中間值的大小，來斷定目標位在資料的前二分之一或是後二分之一。



在上圖的 case 當中，我們設定一個 low 與 High 的區間，例如第一排的 low 對應的 index 是 0，值是-5，而 High 的 index 是 9，值是 10，總數字量 N=10。我們透過將 low 與 high 的兩個 index 相加除以二後，取整數的值是 4，代表 index=4 的值是我們要去切分的目標 Middle。已知 Middle 之值為 2，去驗證此值是否為我們的目標(target)，若否，則去判斷 Middle 與 target 的大小關係，去判斷目標值在此陣列中位於左區(<Middle)或是右區(>Middle)，改變取樣的 low 與 high 位置。依照這樣的算法持續執行，直到找到 target。

2. 二分搜尋法的數學及時間複雜度

二分搜尋法在數學上可以用一串遞迴關係式表達

$$T(N) = 1 + T\left(\frac{N}{2}\right)$$

$$N = \textit{資料數量}, T(N)\textit{代表平均猜測次數}$$

上面的方程式，表示當具有 N 筆資料，每做一次猜測都會將次數加一，並且資料量減半。範圍變成 N/2。如果我們需要討論此演算法的效率，就得要計算從 N 開始做幾步驟，才能使整體的 low 與 high 範圍變成 1，也就是 target=low=high。

$$N \times \left(\frac{1}{2}\right)^{x} = 1$$

上面的方程式，顯示了當我具有 N 筆資料，總共做了 x 個步驟，才會將資料數量降低至 1。看起來很合理，可以把它寫成下式。

$$N = 2^{x}$$

$$x = log_2 N$$

可以知道，當我有 N 筆資料，其步驟就是以二為底的數對 N 產生。可以將它寫成時間複雜度。

$$O(log_2 N)$$

抑或是使用遞迴的數學方式也可以達成。

$$T(N) = 1 + T\left(\frac{N}{2}\right)$$

$$\textit{一次展開}: T(N) = 1 + \left[1 + T\left(\frac{N}{4}\right)\right] = 2 + T\left(\frac{N}{4}\right)$$

$$\textit{二次展開}: T(N) = 2 + \left[1 + T\left(\frac{N}{8}\right)\right] = 3 + T\left(\frac{N}{8}\right)$$

$$\textit{K次展開}: T(N) = K + T\left(\frac{N}{2^k}\right)$$

$$\textit{當問題解決}: N = 1 \ \textit{，故} T(N) = T(1)$$

$$\left(\frac{N}{2^k}\right) = 1$$

$$N = 2^k$$

$$K = log_2 N$$

當然可以再去比較二元樹(binary tree)。但礙於篇幅不詳細探討。

3. 二分搜尋法的程式設計

```
        int solveHua(int n, int target) {
            int low = 1;
            int high = n;
            int steps = 0;

            while (low <= high) {
                steps++;
                // 猜中間的數
                int mid = low + (high - low) / 2;

                if (mid == target) {
                    return steps;
                } else if (mid < target) {
                    low = mid + 1; // 答案在右邊
                } else {
                    high = mid - 1; // 答案在左邊
                }
            }
            return steps;
}
```

4. 三分之一搜尋法與二分搜尋法

此搜尋法目標與二分搜相同，同樣是有一個 low 與 high 的極值，但是對於取一個比大小的標準有些不同，將取 1/3 的地方作為標準點，此時會出現兩種情況，一、target 的值位在 1/3 以下，屬於較為幸運的情況，因為此時節省的資料量最大，而情況二、target 的值位於 1/3 右側，也就是在剩下的 2/3 中，這樣只節省了 1/3 的資料量，故在這裡稱為不幸運的情況。

5. 三分之一搜尋法的數學及時間複雜度

我們依樣將他化為數學遞迴的形式

$$T(N) = 1 + \frac{1}{3}T\left(\frac{N}{3}\right) + \frac{2}{3}T(\frac{2N}{3})$$

求解時間複雜度

$$T(N) = 1 + \frac{1}{3}T\left(\frac{N}{3}\right) + \frac{2}{3}T\left(\frac{2N}{3}\right)$$

$$let\ T(N) = k\log_2 N$$

$$k \log_2 N = 1 + \frac{1}{3} T\left(\frac{N}{3}\right) + \frac{2}{3} T\left(\frac{2N}{3}\right)$$

$$k \log_2 N = 1 + \frac{1}{3} k \log_2\left(\frac{N}{3}\right) + \frac{2}{3} k log_2\left(\frac{2N}{3}\right)$$

$$k = \frac{1}{log_2 3 - \frac{2}{3}} \approx 1.089$$

上面的 k，代表在時間複雜度 O(k $\log_2$N)的係數，已知二分搜的時間複雜度為$\log_2 N$則可以知道，三分之一搜應該是他的複雜度的 1.089 倍，這都是理論值，我們可以實作論證。

6. 三分之一搜尋法的程式設計

```
int solveYing(int n, int target) {
    int low = 1;
    int high = n;
    int steps = 0;

    while (low <= high) {
        steps++;

        // 計算範圍長度，並猜在 1/3 的位置
        // 注意：這裡的實作是取相對位置的 1/3 處
        int range = high - low;
        int mid = low + range / 3;

        if (mid == target) {
            return steps;
        } else if (mid < target) {
            // "不幸運"的情況：
            // 猜的數比目標小，表示目標在右邊剩下的 2/3 區域
            // 範圍縮小比較慢 (剩下約 2/3)
            low = mid + 1;
        } else {
            // "幸運"的情況：
            // 猜的數比目標大，表示目標在左邊剛切出來的 1/3 區域
            // 範圍縮小很快 (剩下約 1/3)
            high = mid - 1;
        }
    }
    return steps;}
```

四、研究結果

1. 我們使用 N=10000，target 從 1 到 10000 逐一測試，並創建兩個函式，分別輸入兩個引數，一個是 N 的大小(資料量)，以及 target 位置，並且讓他自動記錄總搜索數。並在 terminal 上輸出平均猜測次數。
2. 以下是測試結果

N	100	1000	10000	100000	1000000
Binary Sch	5.8000	8.9870	12.3631	15.6895	18.9514
1/3 Sch	6.2800	9.8360	13.4399	17.0454	20.6611
單位	次				

N	100	1000	10000	100000	1000000
$\frac{\frac{1}{3}Sch}{BinarySch}$	1.08275	1.09446	1.08709	1.08642	1.09021
Average	1.088186				
Standard Deviation	0.00439				
Variance	0.00001				
Mean Error	1.088186-1.089 = -0.000814				
% Error	-0.075%				

五、研究結論

透過程式的實作，我們計算出 N 從 100 到 1000000 的各種範圍，二分搜與三分之一蒐所產生的平均次數，我們求出平均、變異數、標準差，並與理論值討論，得到誤差為 -8.14×10^{-4}，百分誤差為-0.075%，在實驗中，應該可以算是誤差非常非常小，也就是驗證以上的理論是正確的:「三分之一搜是二分搜之步驟數的 1.089 倍」。

所以我們終於可以回答以下問題了。

1. 理論上、小華的方法平均要猜幾次?小英的方法平均要猜幾次?
2. 使用程式模擬的方式，驗證理論的結果。

根據我們的實驗及理論，小華(二分搜)要猜測的次數大約為

$$T1 = O(log_2 N)$$

而另一方面，小英(三分之一搜)猜測的次數應該為

$$1.089T1$$

六、研究心得

很感謝老師給予這次機會去嘗試更好的了解時間複雜度，及重新觸碰演算法等。在這個 AI 的時代，許多的實驗及數據探討不再需要上網大量爬文，只需要詢問 AI 並不停的交換意見即可，當然，實作上的驗證也需要人工來審核，所以還是花了些時間撰寫此篇作業。

七、附錄

1. 參考資料
   i. https://medium.com/@john40066/%E6%90%9C%E5%B0%8B%E6%BC%94%E7%AE%97%E6%B3%95-searching-algorithm-d6869b1afe6a
   ii. https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E5%BE%9E%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6%E8%AA%8D%E8%AD%98%E5%B8%B8%E8%A6%8B%E6%BC%94%E7%AE%97%E6%B3%95-%E4%B8%80-b46fece65ba5
   iii. https://medium.com/appworks-school/binary-search-%E9%82%A3%E4%BA%9B%E8%97%8F%E5%9C%A8%E7%B4%B0%E7%AF%80%E8%A3%A1%E7%9A%84%E9%AD%94%E9%AC%BC-%E4%B8%80-%E5%9F%BA%E7%A4%8E%E4%BB%8B%E7%B4%B9-dd2cd804aee1

2. 程式碼

```
#include <iostream>
#include <vector>
#include <numeric>
#include <iomanip>

using namespace std;

// 設定總數字範圍 N (可以設定為 1000 或 10000 來獲得精確平均值)
const int N = 1000000;

// 小華的策略：二分搜尋法 (Binary Search)
// 每次猜中間，範圍縮小一半
int solveHua(int n, int target) {
    int low = 1;
    int high = n;
    int steps = 0;

    while (low <= high) {
        steps++;
        // 猜中間的數
        int mid = low + (high - low) / 2;

        if (mid == target) {
            return steps;
        } else if (mid < target) {
            low = mid + 1; // 答案在右邊
        } else {
            high = mid - 1; // 答案在左邊
        }
    }
    return steps;
}

// 小英的策略：三分之一搜尋法 (1/3 Split)
// 每次猜 1/3 處。幸運範圍剩 1/3，不幸運範圍剩 2/3
int solveYing(int n, int target) {
    int low = 1;
    int high = n;
    int steps = 0;
```

```
        while (low <= high) {
            steps++;

            // 計算範圍長度，並猜在 1/3 的位置
            // 注意：這裡的實作是取相對位置的 1/3 處
            int range = high - low;
            int mid = low + range / 3;

            if (mid == target) {
                return steps;
            } else if (mid < target) {
                // "不幸運"的情況：
                // 猜的數比目標小，表示目標在右邊剩下的 2/3 區域
                // 範圍縮小比較慢 (剩下約 2/3)
                low = mid + 1;
            } else {
                // "幸運"的情況：
                // 猜的數比目標大，表示目標在左邊剛切出來的 1/3 區域
                // 範圍縮小很快 (剩下約 1/3)
                high = mid - 1;
            }
        }
        return steps;
    }

    int main() {
        long long totalStepsHua = 0;
        long long totalStepsYing = 0;

        cout << "開始模擬 N = " << N << " 的情況..." << endl;

        // 模擬每一個數字當作目標答案的情況
        for (int target = 1; target <= N; ++target) {
            totalStepsHua += solveHua(N, target);
            totalStepsYing += solveYing(N, target);
        }

        // 計算平均值
        double avgHua = (double)totalStepsHua / N;
        double avgYing = (double)totalStepsYing / N;
```

```
    cout << fixed << setprecision(4);
    cout << "-----------------------------------------" << endl;
    cout << "模擬結果 (平均猜測次數):" << endl;
    cout << "小華 (二分法): " << avgHua << " 次" << endl;
    cout << "小英 (1/3 法):  " << avgYing << " 次" << endl;
    cout << "-----------------------------------------" << endl;

    // 結論判定
    if (avgHua < avgYing) {
        cout << "結論: 小華的策略較優 (平均次數較少)" << endl;
    } else {
        cout << "結論: 小英的策略較優" << endl;
    }

    return 0;
}
```

CYEE 中原電機一甲 潘宇綸 11428109

附錄完