國立臺南大學資訊工程學系

資工三「演算法」課程

第一次作業

**題目: Minimum Distance**

|  |  |  |
| --- | --- | --- |
| 班級 | ： | 資工三 |
| 姓名 | ： | 黃詩豪 |
| 學號 | ： | S10259002 |

老師：陳宗禧

中華民國 104年10月07日

# 目錄

1. **簡介及問題描述……….……………..…………………………………………1**
   1. **簡介…………….…………………………………………………………………………2**
   2. **問題………………………….……………………………………………………………4**
2. **理論分析….………………………..………………………………………8**
3. **演算法則….…………………………..……………………………………10**
4. **程式設計環境架構.………………………..…………………………………12**
5. **程式.…………………………………………..………………………………14**
6. **執行結果、討論與心得……………………..………………………………18**

參考文獻……………………………………………………………………………22

**(一) 簡介及問題描述**

1. 簡介

在正整數的陣列中，找出兩數最小之差，亦即兩數最短距離，討論暴力法（MinDistance）與改進的方法（MyMinDistance）執行之效能。

2. 問題

Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.

**ALGORITHM** *MinDistance*(A[0..n-1])

//Input: Array A[0..n-1] of numbers

//Output: Minimum distance between two of its elements

*dmin* **← ∞**

**for** *i* ← 0 **to** *n* – 1 **do**

**for** *j* ← *i* + 1 **to** *n* – 1 **do**

**if** *i* ≠ *j* **and** |A[*i*] – A[*j*]| < *dmin*

*dmin* ←|A[*i*] – A[*j*]|

**return** *dmin*

Make as many improvements as you can in this algorithmic solution to the problem. If you need to, you may change the algorithm altogether; if not, improve the implementation given.

寫出兩個演算法的時間複雜度，並比較其執行時間。比較數量*n*需動態調整(如10,000~100,000等)，並比較之。

**(二) 理論分析**

1. Minimum Distance

陣列共有n個元素，使用兩個迴圈，讀取陣列中的元素，然後相減取絕對值，每次的計算都更新現在已知的最小距離。外層迴圈從i=0至i=n-1，共執行n次；內層迴圈從i+1至n-1，共執行（n-1-i）次，程式總共執行

故時間複雜度（Big O）為：O(n2)

2. Improved Minimum Distance

陣列共有n個元素，為了找出最小距離之兩數，可以先將數值排序，再計算相鄰之兩數差，並記錄最小距離，直到全部元素皆已計算過。利用快速排序法（QuickSort）預先將陣列排序，所需時間複雜度為O(nlogn)，然後利用迴圈從陣列第一個元素開始計算，將第i+1個與第i個元素相減，直到第n-2個元素為止，便可得知最小距離，此程序需執行n-1次，故時間複雜度為O(n)。最後，因為O(nlogn) + O(n)屬於O(nlogn)，所以總共的時間複雜度為O(nlogn)。

**(三) 演算法則**

Minimum distance系統實作的演算法以及流程圖…

1. 第一個演算法(Algorithm)

**ALGORITHM** *MinDistance*(A[0..n-1])

//Input: Array A[0..n-1] of numbers

//Output: Minimum distance between two of its elements

*dmin* **← ∞**

**for** *i* ← 0 **to** *n* – 1 **do**

**for** *j* ← *i* + 1 **to** *n* – 1 **do**

**if** *i* ≠ *j* **and** |A[*i*] – A[*j*]| < *dmin*

*dmin* ←|A[*i*] – A[*j*]|

**return** *dmin*

1. 演算法時間複雜度(time complexity)

外迴圈從0執行至n-1，共執行n次，內迴圈從i+1執行至n-1次，共執 行(n-1-i)次，主程式總共執行n(n-1)/2次，故時間複雜度為O(n2)。

1. 演算法空間複雜度(space complexity)
2. Fixed space requirements:輸入A[0..n-1]陣列，故為n個記憶體空間大小
3. Variable space requirements:程式中無動態配置記憶體變數，以及遞迴程式，故無變動空間

因此空間複雜度為O(n)，n是輸入陣列的大小。

2. 第二個演算法(Algorithm)

**ALGORITHM** *MyMinDistance*(A[0..n-1], ArraySize)

//Input: Array A[0..n-1] of numbers

// Integer ArraySize of A[0..n-1]

//Output: Minimum distance between two of its elements

*dmin* **← ∞,** *diff* ← 0

***quick\_sort***(A[0..n-1], 0, ArraySize)

**for** *i* **to** n – 1 **do**

*diff* ← A[*i* + 1] – A[*i*]

**if** *dmin > diff*

*dmin* ← diff

**end for**

**return** *dmin*

**ALGORITHM** *quick\_sort*(A[0..n-1], left, right)

//Input: Array A[0..n-1] of numbers

// Integer left is the first element index of A[0..n-1]

// Integer right is the last element index of A[0..n-1]

//Output: Sorted Array of A[0..n-1] in increasing order

**if** left < right

*pivot* ← A[(left + right) / 2]

*i* ← left – 1

*j* ← right + 1

**while** *i* < *j*

**while** A[++*i*] < *pivot;*

**while** A[--*j*] > *pivot;*

**if** *i* < *j*

*SWAP*(A[*i*], A[*j*])

**end while**

*quick\_sort*(A[0..n-1], left, *i*-1)

*quick\_sort*(A[0..n-1], *j*+1, right)

**end if**

1. 演算法時間複雜度(time complexity)

利用快速排序法（QuickSort）將陣列排序，所需時間為O(nlogn)，然後計算最 小距離，使用單一迴圈執行n-1次，所需時間為O(n)，因此總共所需時間複雜 度為O(nlogn)。

1. 演算法空間複雜度(space complexity)
2. Fixed space requirement:輸入n個元素大小的陣列，所以固定的空間複雜度為O(n)
3. Variable space requirement:因為使用QuickSort排序且QuickSort使用遞迴的方式，故會使用到堆疊的空間，在平均的情況下，如果選擇的支點剛好是偶數，則空間複雜度為O(logn)；在最糟的情況下，如果選擇的支點剛好在第一個元素，則空間複雜度為O(n)，因此變動的空間複雜度為O(n)

演算法總共的空間複雜度為O(n)。

**(四) 程式設計環境架構**

程式設計語言、工具、環境與電腦硬體等規格說明…

1. 程式語言

C in Linux GCC

OS: Linux Mint 17.2 "Rafaela" - Cinnamon (64-bit)

2. 程式開發工具

Eclipse，Version: Mars Release (4.5.0)

Eclipse C/C++ Plugin: CDT 8.8.0 for Eclipse Mars

Compiler: Linux GCC

3. 電腦硬體

CPU: Intel Xeon E3-1231 v3

Main Memory: 32GB

**(五) 程式 (含source code, input code, and output code)**

程式含source code, input code, and output code等…

1. 主程式

見資料夾MD\_Prog\_S10259002\_SourceCode

2. Input Code Format

使用亂數產生n個正整數，儲存至Test$i.txt檔案（$i為1, 5, 10），檔案中第一列表示有n個正整數，第二列為亂數產生的1至n個正整數數值

(1)Test1.txt, with size n = 10000

(2)Test5.txt, with size n = 50000

(3)Test10.txt, with size n = 100000

3. Output Code Format

程式輸出結果是以直接顯示於Terminal上方式，印出在n個正整數的陣列中，找到的最小距離

(1) 0

(2) 0

(3) 0

**(五) 執行結果、討論與心得**

執行結果與討論 (執行時間、problem *n*的大小等問題討論)等…

1. 執行結果

Output of program:

可以正確判斷在n個正整數的陣列當中，最小距離為多少。

2. 討論

執行時間、問題大小等問題討論

1. Running Time

上圖為兩個方法分別跑n=10000, 20000, 30000,... ,100000，所得到的執行時間。從圖中可以發現當n值逐漸增加時，時間複雜度為O(n2)的MinDistance方法所花費的時間遠比使用O(nlogn)的MyMinDistance方法多，導致第二個方法在視覺上不管n值的大小都無增加比對的時間，因此將分別列出兩個方法個別的執行時間折線圖，以便觀察是否如理論上得到的曲線相似。

i.在MinDistance方法中，使用的時間複雜度為O(n2)，下圖為此方法實際執行時間

ii.在MyMinDistance方法中，時間複雜度為O(nlogn)，下圖為此方法實際執行時間

(2) Problem size *n*

i.MinDistance方法使用巢狀迴圈，主要是輸入的陣列所使用到記憶體空間，因此只隨著輸入的陣列大小而成長。

ii.MyMinDistance方法先使用Quick Sort排序輸入之陣列，此排序方法在最糟情況下需額外n個空間，在加上原本輸入陣列n個空間，因此會以2n的方式隨著輸入的陣列大小成長。

3. 心得

設計MinDistance演算法的過程中，先理解最笨的方法以後，想到可以先將數值排序，如此一來，只需要比較兩兩相鄰的數之距離即可。但排序有很多種方法，例如：Bubble Sort、Insertion Sort、Quick Sort等，所以我直接找出理論上每個排序的時間複雜度，發現Heap Sort、Quick Sort的時間複雜度還不錯，因此使用此三種排序方法跑實驗。但在測試一千萬筆數值的陣列時，發現Quick Sort的執行時間比較短，所以最後選擇使用Quick Sort，雖然Heap Sort的時間複雜度與Quick Sort相同，但或許是Heap Sort在建立Heap的時候花費的時間較多。

這次設計的過程中，雖然想過不用排序的方法，能不能找到比O(n2)還快的方法，但想了很久還是想不到合適的方法，最後還是使用Quick Sort降低時間複雜度。透過這次演算法作業，對於計算時間複雜度有更進一步的理解，以及空間複雜度的計算，也在實驗中，體會好的演算法所帶來的執行效率是比暴力法還要來的快。

**參考文獻**

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms," Third Edition, The MIT Press, 2009.
2. R.C.T. Lee, S.S. Tseng, R.C. Chang, and Y.T.Tsai, "Introduction to the Design and Analysis of Algorithms," McGraw-Hill, 2005.
3. Anany V. Levitin, "Introduction to the Design and Analysis of Algorithms," 2nd Edition, Addison Wesley, 2007.
4. Richard Neapolitan and Kumarss Naimipour, "Foundations of Algorithms," Fourth Edition, Jones and Bartlett Publishers, 2010.
5. http://www.csie.ntnu.edu.tw/~u91029/SequenceManipulation.html