國立臺南大學資訊工程學系

資工三「演算法」課程
第二次作業

題目: Travel Map

班級 ： 資工三

姓名 ： 林星宇

學號 ： S10659029

老師：陳宗禧

中華民國 108 年 10 月 30 日

內容

壹、 簡介及問題描述

設計與實作 Complete Graph 判斷，理論驗證與實驗分析該問題!

1. 簡介

給定一地圖，內有 p 個興趣點(POI, Point of Interest)，例：臺南古蹟、臺南小吃等。假設目 前有一個人或一個旅行團，對於臺南市的美食或古蹟有高度興趣，並且要規劃行程，但對於 該行程需要有資訊系統輔助，給定行前的建議，底下問題為規劃者需要知道的答案：

2. 問題

(1) 哪兩個 POIs 靠最近? 距離多少? (brute-force algorithm)
(2) 這些 POIs 的範圍有多大 (Convex-Hull，它的面積以及最遠的距離)? (brute-force algorithm)
(3)

(a) 假如要到所有 p 個 POIs，則最短行程距離是多少? (exhaustive search + brute-force algorithm)
(b) 假如我們設計一個新演算法：(Convex-Hull-TSP Algorithm)

i. 求出所有點的 Convex-Hull
ii. 除 Convex-Hull 上的點外，其餘 POIs 找出離 Convex-Hull 邊最近的點投影
iii. 按照投影點依序由 Extreme Point 點開始旅遊，再繞回起始點 請問該演算法的時間複雜度(以 big O 表示)以及(行程)花多少距離?

(c) 比較 3(a)和 3(b)兩者的距離比較，以及執行時間比較?

(4) 假設我們不可能到全部的 POIs，需要縮短行程，則要找多個 POIs 間的距離不超過一固定範圍(r)形成一個 Group (Clique) g，假如 Group g 內的 POIs 個數低於 k 個，也不安排到 g 的行程。假設符合規範的 Group 共有 n 個，gi, $1 \le i \le n$，列出這 n 個 Group 的中心位置 ci, $1 \le i \le n$ (或指定該群的任一點 POI or 最靠近的停車場等)。更進一步，我們想要知道假如規劃由一點 cj 開始出發(Group 間開車)，要玩遍所有的 Group 內的 POIs(Group 內走路)，再回到出發處。請問該行程(開車與走路)要花多少距離? (exhaustive search + brute-force algorithms: Clique, TSP problems (TSP 方法用 3(a) or 3(b)亦可，兩者皆實作亦可))

貳、 理論分析

1. Brute-Force algorithm

```
for i ← 0 to n – 1 do
        for j ← i + 1 to n do
            if distance(A[ i, j]) < temp
                temp = distance(A[ i, j])
```

2. Convex-Hull brute-force algorithm

(a) 先求出兩點(x1, y1), (x2, y2)形成的線 y = ax + b
(b) 利用 Brute-force 藉由 y – ax > b 或 y – ax < b 判斷每點在線的左邊或右邊
(c) 如果每點皆在同側，則形成此線段之兩點為 Convex-Hull 邊上的點

3. Exhaustive search + brute-force algorithm

```
ShortestTravel(int a, int b, int* num) {
    if (a == b)
        for i ← 0 to n do
            temp = num[i];
            if (comparedistance > distance)
                    comparedistance = distance;}
    else
        for j ←a to n do
            swap(num[a], num[j]);
            ShortestTravel(a + 1, b, num);
            swap(num[a], num[j]);
}
```

4. Exhaustive search + brute-force algorithms

(a) 輸入至少經過n點，最常路徑r
(b) 利用組合列出所有路徑可能，並計算路徑
(c) 比較是否超過r並顯示結果

參、 演算法則

1. Brute-force algorithm

利用兩層 for 迴圈算出兩點之間的距離並存取距離最小之值

i. 演算法時間複雜度(time complexity)

O(n^2)

ii. 演算法空間複雜度(space complexity)

O(1)

2. Convex-Hull brute-force algorithm

利用兩層 for 迴圈求出兩點直線方程式，並比較所有點在此直線之同側或異側

i. 演算法時間複雜度(time complexity)

O(n^2)

ii. 演算法空間複雜度(space complexity)

O(1)

3. Exhaustive search + brute-force algorithm

利用 recursive 排列所有可能並記錄各排列總距離

iii. 演算法時間複雜度(time complexity)

O(n!)

iv. 演算法空間複雜度(space complexity)

O(1)

4. Exhaustive search + brute-force algorithms

利用 recursive 求出組合後計算重心

i. 演算法時間複雜度(time complexity)

O(n!)

ii. 演算法空間複雜度(space complexity)

O(1)

肆、 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 MS Windows

2. 程式開發工具

Visual Studio 2019

3. 電腦硬體

作業系統: Window10 家用版
系統類型: 64 位元作業系統，x64 型處理器
處理器: Intel® Core™ i7-7700HQ CPU @ 2.80GHz
顯示卡: Geforce® GTX 1050 Ti with 4GB GDDR5 Graphics

伍、 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

(a) Brute-Force algorithm:

```
//brute-force求最短距離
minDis = AdjacencyMatrix[0][1];
for (int i = 0; i < p - 1; i++)
{
    for (int j = i + 1; j < p; j++)
    {
        if (AdjacencyMatrix[i][j] < minDis)
        {
            minDis = AdjacencyMatrix[i][j];
            minPoint[0] = i;
            minPoint[1] = j;
        }
    }
}
```

(b) Convex-Hull brute-force algorithm

```
for (int i = 0; i < p - 1; i++)
{
    for (int j = i + 1; j < p; j++) {
        allsame = true;

        //求出兩點形成的線
        m = (point[i].latitude - point[j].latitude) / (point[i].longitude - point[j].longitude); //斜率
        constant = (-1 * m) * point[i].longitude + point[i].latitude;//常數比較值
        //記錄各點在線上、線的左邊或右邊
        for (int a = 0; a < p; a++)
        {
            if (a == i || a == j)
            {
                compare[a] = 0;
            }
            else
            {
                if ((point[a].latitude - point[a].longitude * m) > constant)
                    compare[a] = 1;
                else if ((point[a].latitude - point[a].longitude * m) < constant)
                    compare[a] = -1;
            }

        }
```

```
        //檢查點是否皆在同一邊
        for (int b = 0; b < p; b++)
        {
            if (compare[b] > 0) {
                allsame = false;
            }
        }
        if (allsame == false) {
            allsame = true;
            for (int c = 0; c < p; c++)
            {
                if (compare[c] < 0) {
                    allsame = false;
                }
            }
        }
```

```
        //如果點皆在同側，紀錄形成線之兩點
        if (allsame) {
            temp1.push_back(point[i]);
            temp2.push_back(point[j]);
            ConvexHull_allPoint[i] = 1;
            ConvexHull_allPoint[j] = 1;
            if (two_point)
            {
                temp_point.push_back(point[i]);
                two_point = false;
            }
            else
            {
                temp_point.push_back(point[j]);
            }

        }

    }
    two_point = true;
}
```

(c) Exhaustive search + brute-force algorithm

```
void ShortestTravel(int a, int b, int* num) {
    int temp, firstPoint, distance = 0;
    if (a == b)
    {
        for (int i = 0; i < point.size(); i++) {
            if (i == 0) {
                temp = num[i];
                firstPoint = num[i];
            }

            distance += AdjacencyMatrix[temp][num[i]];
            temp = num[i];
            if (i == point.size() - 1)
            {
                distance += AdjacencyMatrix[temp][firstPoint];
                if (comparedistance == 0)
                    comparedistance = distance;
                else if (comparedistance > distance)
                    comparedistance = distance;
            }
        }
    }
```

```
    else
        for (int j = a; j < point.size(); j++)
        {
            swap(num[a], num[j]);
            ShortestTravel(a + 1, b, num);
            swap(num[a], num[j]);
        }
}
```

(d) Exhaustive search + brute-force algorithms

i. 做組合 C(n, p)

```
void c_recur(int k, int n, int m, vector<int> list)
{
    list.push_back(k - 1);
    for (int i = k; i <= (m - n) && n > 0; ++i)
    {
        c_recur(i + 1, n - 1, m, list);
    }
    if (n == 0)
    {
        for (int i = 0; i < list.size(); ++i)
        {
            temp.Path.push_back(list[i]);
        }
        path.push_back(temp);
        temp.Path.clear();
    }
}
```

ii. 求出路徑之重心

```
Point polygon_centroid(vector <int> input_path)
{
    Point G;
    float cx = 0, cy = 0, w = 0;
    for (int i = input_path.size() - 1, j = 0; j < input_path.size(); i = j++)
    {
        float a = cross(point[input_path[i]], point[input_path[j]]);
        cx += (point[input_path[i]].longitude + point[input_path[j]].longitude) * a;
        cy += (point[input_path[i]].latitude + point[input_path[j]].latitude) * a;
        w += a;
    }

    G.name = "停車場";
    G.longitude = cx / 3 / w;
    G.latitude = cy / 3 / w;
    return G;
}

double cross(Point v1, Point v2)
{
    // 沒有除法，儘量避免誤差。
    return v1.longitude * v2.latitude - v1.latitude * v2.longitude;
}
```

iii. 計算路徑和比較

```
//組合結果
for (int i = 0; i < path.size(); i++)
{
    distance = 0;

    //先求此組合之重心G
    for (int j = 0; j < path[i].Path.size(); j++)
    {
        G.push_back(polygon_centroid(path[i].Path));
    }

    //計算各路徑總距離
    for (int j = 0; j < path[i].Path.size(); j++)
    {
        if (j == 0)
            distance += Distance(G[i].longitude, point[path[i].Path[j]].longitude, G[i].latitude, point[path[i].Path[j]].latitude);
        else if(j != path[i].Path.size() - 1)
            distance += Distance(point[path[i].Path[j - 1]].longitude, point[path[i].Path[j]].longitude, point[path[i].Path[j - 1]].latitude,
        else if (j == path[i].Path.size() - 1)
            distance += Distance(G[i].longitude, point[path[i].Path[j]].longitude, G[i].latitude, point[path[i].Path[j]].latitude);
    }
```

```
    //顯示距離小於使用者輸入的結果
    if (distance < input_km)
    {
        cout << endl << "此路徑中心點為: ";
        cout << G[i].name << "(" << G[i].longitude << ", " << G[i].latitude << ")" << endl;
        cout << "路徑為:" << endl;
        cout << G[i].name << "-->";
        for (int j = 0; j < path[i].Path.size(); j++)
        {
            cout << point[path[i].Path[j]].name << "-->";
        }
        cout << G[i].name << " ";
        cout << "總距離: " << distance << "公尺" << endl << endl;
        amount++;
    }

}
cout << "總共有" << amount << "個方法" << endl;
```

2. Input Code Format

Three of examples for input use are in below….

(1) input1.txt : 7 個點
(2) input2.txt : 10 個點
(3) input3.txt : 11 個點

3. Output Code Format

(1) 景點、緯度、經度
(2) TSP(以 Adjacency Matrix 表示)
(3) 第一題
(4) 第二題
    i. Convex Hull 圍成的景點
    ii. Convex Hull 圍成最大面積
    iii. Convex Hull 最遠兩點
(5) 第三題
    i. Brute-Force 最短距離
    ii. Convex Hull TSP
    iii. 比較兩者距離差距
(6) 第四題
    i. 路徑
    ii. 方法個數

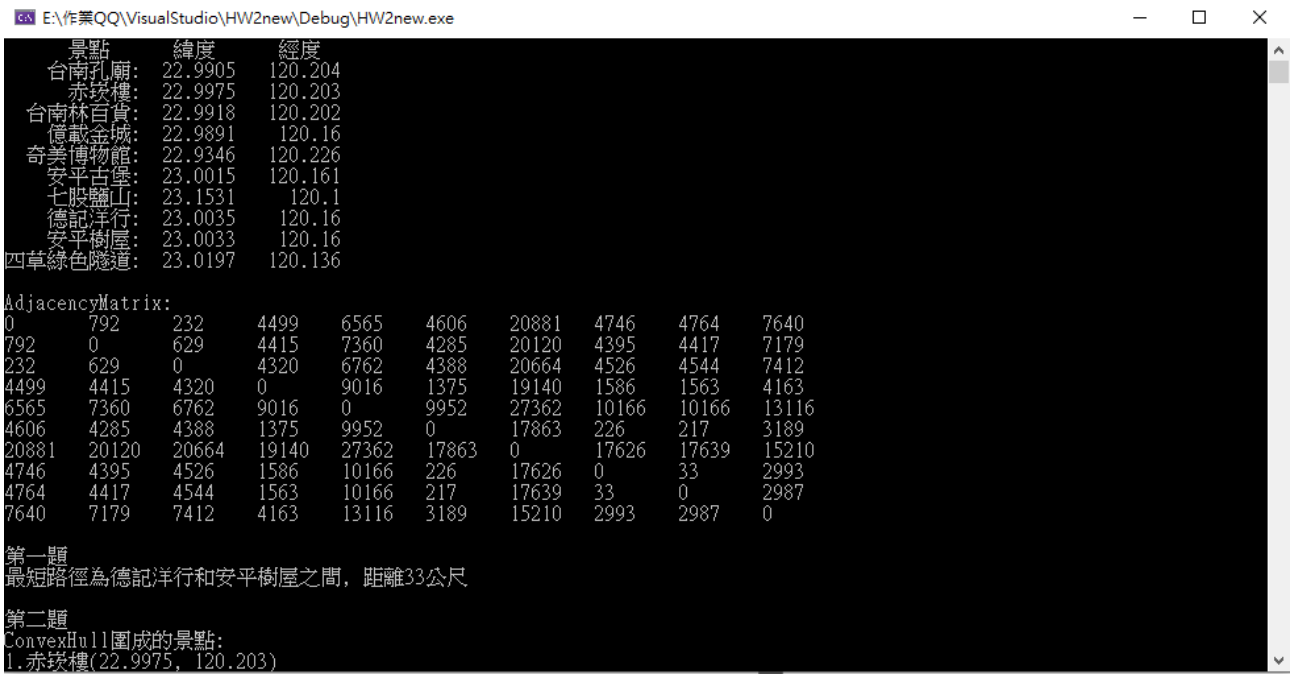
陸、 執行結果、討論與心得

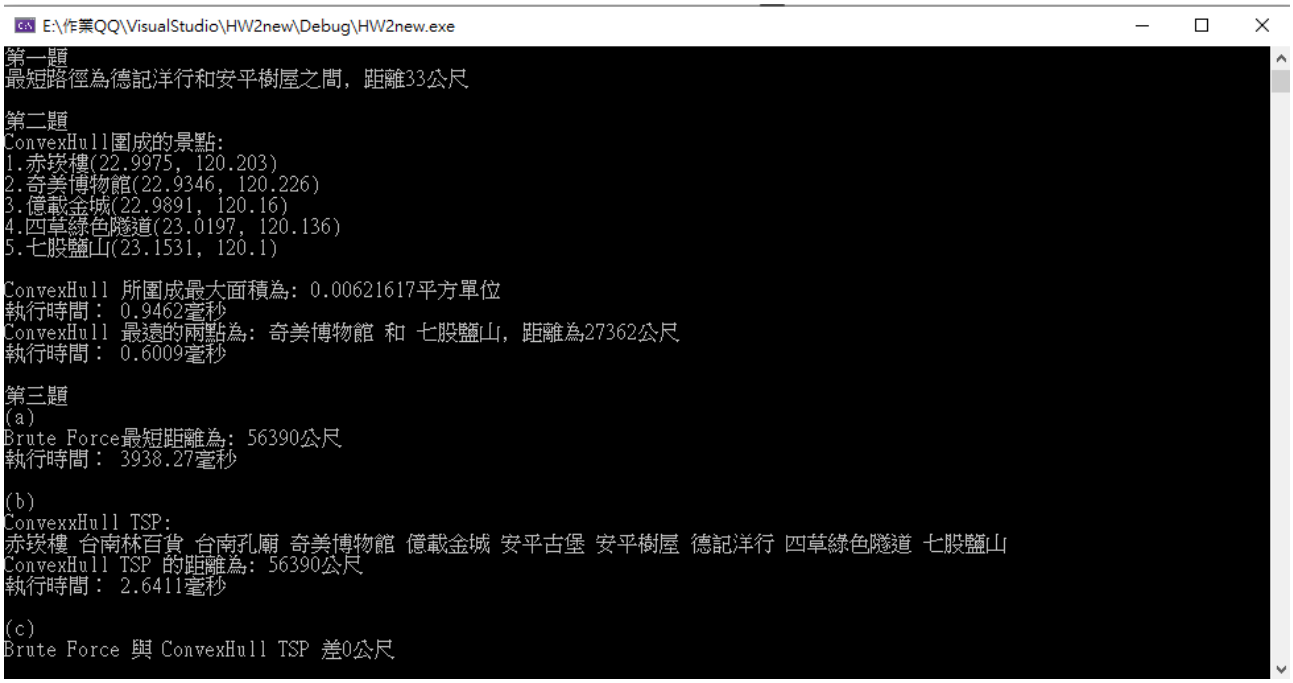
執行結果與討論 (執行時間、problem *n* 的大小等問題討論)等…

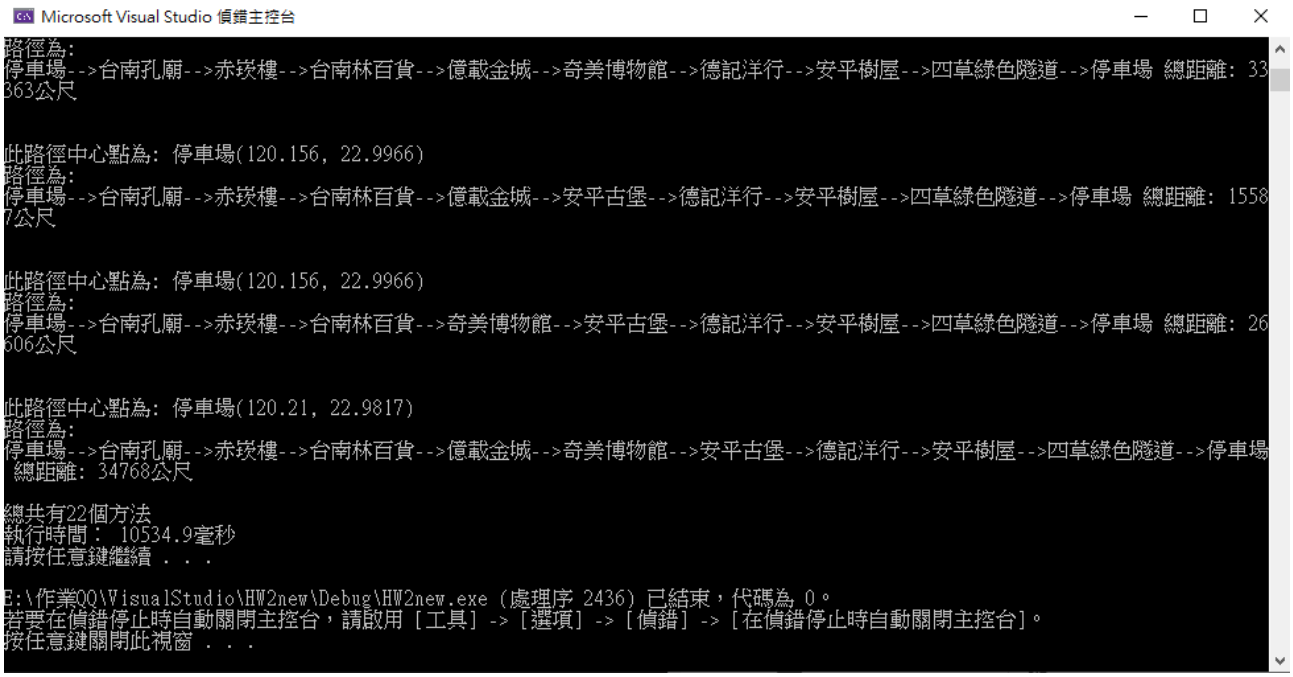
1. 執行結果

Output of program:

以 10 個點為例:





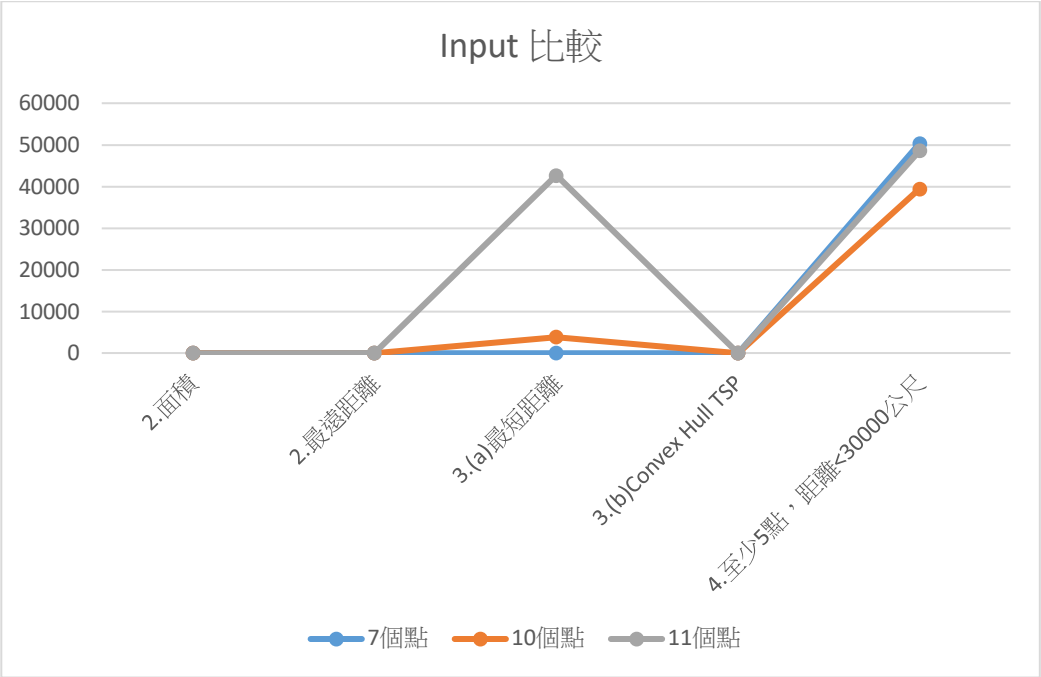


2. 討論

執行時間、問題大小等問題討論! 利用 MS Excel 畫出問題大小與執行時間的關係!

Running Time (單位:毫秒)

	2.面積	2.最遠距離	3.(a)最短距離	3.(b)Convex Hull TSP	4.至少 5 點，距離 <30000 公尺
7 個點	0.4596	1.0188	6.8152	2.8256	50378.4
10 個點	0.6832	0.7581	3837.05	6.3134	39472.9
11 個點	1.5438	0.778	42652.4	3.5204	48586.3



3. 心得

從圖表結果可以看出，最短距離因為使用 Brute-Force，O(n!)，此演算法非常耗時，當點少變多時，所需時間的上升速度非常快，因此 3(b)採用的方法雖然未必是最佳解，但以空間換取時間有時是很重要的。

柒、 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.