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 | ： | 曾俊杰 |
| 學號 | ： | S11059003 |

老師：陳宗禧

中華民國 112年10月13日

**目錄**

[(一) 簡介及問題描述 4](#_Toc149480117)

[**1. 簡介 4**](#_Toc149480118)

[**2. 問題 4**](#_Toc149480119)

[(二) 理論分析 5](#_Toc149480120)

[**1. Brute-force理論 5**](#_Toc149480121)

[**2. Exhaustive search + Brute-force理論 5**](#_Toc149480122)

[**3. Convex-Hull-TSP理論 5**](#_Toc149480123)

[(三) 演算法則 6](#_Toc149480124)

[**1. Brute-force Algorithm 6**](#_Toc149480125)

[**(1) Closest(POIs) 6**](#_Toc149480126)

[**(2) Max\_ConvexHull(POIs) 7**](#_Toc149480127)

[**2. Brute-force Algorithm and Exhaustive search Algorithm 9**](#_Toc149480128)

[**3. Convex-Hull-TSP Algorithm 11**](#_Toc149480129)

[(四) 程式設計環境架構 17](#_Toc149480130)

[**1. 程式語言 17**](#_Toc149480131)

[**2. 程式開發工具 17**](#_Toc149480132)

[**3. 電腦硬體 17**](#_Toc149480133)

[**4. 作業系統 17**](#_Toc149480134)

[(五) 程式 (含source code, input code, and output code) 18](#_Toc149480135)

[**1. 主程式 18**](#_Toc149480136)

[**(1) Cloest(POIs) 18**](#_Toc149480137)

[**(2) Max\_ConvexHull(POIs) 20**](#_Toc149480138)

[**(3) Shortest\_Path(POIs) 23**](#_Toc149480139)

[**(4) Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull) 26**](#_Toc149480140)

[**2. Input Code Format 34**](#_Toc149480141)

[**3. Output Code Format 35**](#_Toc149480142)

[**(1) Closet(POIs) 35**](#_Toc149480143)

[**(2) Max\_ConvexHull(POIs) 35**](#_Toc149480144)

[**(3) Shortest\_Path(POIs) 35**](#_Toc149480145)

[**(4) Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull) 36**](#_Toc149480146)

[(六) 執行結果、討論與心得 37](#_Toc149480147)

[**1. 執行結果 37**](#_Toc149480148)

[**(1) Closet(POIs) 38**](#_Toc149480149)

[**(2) Max\_ConvexHull(POIs) 38**](#_Toc149480150)

[**(3) Shortest\_Path(POIs) 38**](#_Toc149480151)

[**(4) Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull) 38**](#_Toc149480152)

[**2. 討論 39**](#_Toc149480153)

[**(1) Running Time 39**](#_Toc149480154)

[**(2) Distance Comparison 39**](#_Toc149480155)

[**(3) Error of reflection points 39**](#_Toc149480156)

[**3. 心得 43**](#_Toc149480157)

[參考文獻 45](#_Toc149480158)

# (一) 簡介及問題描述

## **1. 簡介**

給定一個有n個興趣點(POI, Point of Interest)的地圖，以Brute-force algorithm求任最近之兩點與其距離，以及最大Convex-Hell的面積和周長。利用Exhaustive search和Brute-force algorithm求最短串連n個興趣點的路徑(TSP problem)；利用Convex-Hell-TSP algorithm再求一次，計算該演算法的時間複雜度，比較這兩個演算法的最終的路徑長度和執行時間。

## **2. 問題**

建立一個具有n個興趣點的二維vector(vector<vector<double>>)來表示該地圖。

1. Closest(POIs)問題定義：使用Brute-force algorithm求最短的POI1和POI2距離，並記錄該兩點。
2. Max\_ConvexHull(POIs)問題定義：使用Brute-force algorithm求最大的POIs convex-hull面積和周長
3. Shortest\_Path(POIs)問題定義：求通過n個POIs的最短路徑，最終需回到起點形成環狀多邊形。
   * 1. 使用Exhaustive search和Brute-force algorithm求最短路徑距離、時間複雜度。
     2. 使用Convex-Hull-TSP Algorithm求最短路徑距離、時間複雜度。

比較a和b的路徑距離比例與執行時間。

# (二) 理論分析

## **1. Brute-force理論**

1. 先固定一點，依序計算已固定點以外的點的距離，直到最後一點，接續固定下一點，以此反覆進行，直到圖中任意兩點都已經比較完成，即可求出最短兩點和距離。
2. 先固定一點，依序對已故定點以外的點做ax + by = c直線方程式，其中a, b, c為常數，比較構成直線外的n-2個點是否位於同一側(如果有一點大於c，則不可有其他點小於c，反之亦然)，如果n-2個點都在同一側，代表該直線位於Convex-Hull的邊上，可以紀錄該直線的長度。接續固定下一點，以此反覆進行直到圖中任意兩點都已經連成過直線，並完成比較，即可求出Convex-Hull的面積和周長。

## **2. Exhaustive search + Brute-force理論**

先固定一點，依序計算已固定點以外的點的距離，直到最後一點，找到距離當前最短的點，紀錄下來，接續以此為新固定點，進行排除已固定過的點的搜尋，直到連回最一開始的固定點，即可求出最短通過n個POIs的路徑。

## **3. Convex-Hull-TSP理論**

求出所有點的Convex-Hull，除Convex-Hull 上的點外，其餘POIs 找出離Convex-Hull 邊最近的點投影，首先求出所有Convex-Hull邊的直線方程式dx + ey + f = 0，對於該點座標

(，加上直線方程式的t倍法向量，令其為(，此時該點就能位於直線上，求出該點x值和y值後，檢查該點是否位於該Convex-Hull的範圍之內(兩端點之間的矩形)，如果有，則接續比較該點與符合條件的Convex-Hull的距離，即可求出最短相距且符合條件的Convex-Hull，接續將剩下不在Convex-Hull邊上的點的投影求出，按照投影點由第一個輸入的點開始繪圖，即可求出最短通過n個POIs的路徑。

# (三) 演算法則

## **1. Brute-force Algorithm**

* 1. **Closest(POIs)**

一張含有 圖表, 文字, 行, 寫生 的圖片

自動產生的描述

func get\_distance**(**x1**,** y1**,** x2**,** y2**):** result**,** result is double

**return** sqrt**((**x1 **-** x2**)** **^** 2 **+** **(**y1 **-** y2**)** **^** 2**)**

func closest**(**location**[**n**][**2**])**

min\_distance **<-** get\_distance**(**location**[**0**][**0**],** location**[**0**][**1**],** location**[**1**][**0**],** location**[**1**][**1**]**

min\_index **<-** **[**0**,** 1**]**

**for** i **(**0 to n**-**2**)** **do**

**for** j **(**0 to n-1**)** **do**

current\_distance **<-** get\_distance**(**location**[**i**][**0**],** location**[**i**][**1**],**

location**[**j**][**0**],** location**[**j**][**1**]**

**if** min\_distance **>** current\_distance then

min\_distance **<-** current\_distance

min\_index **<-** **[**i **,**j**]**

**else** **if** min\_distance **=** current distance then

min\_index**.**append**([**i**,** j**])**

end **if**

end **for**

end **for**

Print pair**(**s**)** of closest POIs **and** their distance.

1. **演算法時間複雜度(Time Complexity): O()**

For the most-cost-time part is the double nest loop, it executes (n) \* (n) times, so the time complexity is O().

1. **演算法空間複雜度(Space Complexity): O(*n*)**

Because the array for storing closest POIs indexs is from 1 up to n-1, so the space complexity may be O(n).

* 1. **Max\_ConvexHull(POIs)**

一張含有 圖表, 寫生, 圖畫, 文字 的圖片

自動產生的描述

func get\_distance**(**x1**,** y1**,** x2**,** y2**):** result**,** result is double

**return** sqrt**((**x1 **-** x2**)** **^** 2 **+** **(**y1 **-** y2**)** **^** 2**)**

func check\_same\_side**(**standard**,** compare**):** result**,** result is bool

**if** standard **>** compare then

**return** 1

end **if**

**return** 2

func max\_convex\_hull**(**location**[**n**][**2**]):** result**,** result is int**[]**

perimeter **<-** 0

area **<-** 0

**for** i **(**0 to n**-**2**)** **do**

**for** j **(**0 to n-1**)** **do**

**if** location**[**j**][**1**]** **-** locaion then

**continue**

end **if**

coefficient **<-** location**[**j**][**1**]** **-** location**[**i**][**1**])** **/** **(**location**[**j**][**0**]** **-** location**[**i**][**0**]**

divide **<-** location**[**i**][**1**]** **-** coefficient **\*** location**[**i**][**0**]**

k **<-** 0

flag1 **<-** 0

**while** flag1 **=** 0

**if** k **!=** i **and** k **!=** j then

flag1 **<-** check\_same\_side**(**divide**,** location**[**k**][**1**]** **-**

coefficient **\*** location**[**k**][**0**])**

end **if**

k **+=** 1

end **while**

flag2 **<-** flag1

**for** k **(**k to n**-**2**)** **do**

**if** k **!=** i **and** k **!=** j then

flag2 **<-** check\_same\_side**(**divide**,** location**[**k**][**1**]** **-** coefficient **\***

location**[**k**][**0**]**

**if** flag1 **!=** flag2 then

**break**

end **if**

end **if**

end **for**

**if** flag1 **=** flag2 then

result**.**append**(**i**)**

result**.**append**(**j**)**

perimeter **+=** get\_distance**(**location**[**i**][**0**],** location**[**i**][**1**],**

location**[**j**][**0**],** location**[**j**][**1**])**

area **+=** **(**location**[**i**][**0**]** **\*** location**[**j**][**1**]** **-** location**[**j**][**0**]** **\*** location**[**i**][**1**])** **/** 2

end **if**

end **for**

end **for**

**return** result

Print pairs of convex**-**hull POIs **and** the perimeter **and** area of the convex**-**hull**.**

1. **演算法時間複雜度(Time Complexity): O()**

For the most-cost-time part is the triple nest loop, it executes (n) \* (n+1) \* (n-1) times, so the time complexity is O().

1. **演算法空間複雜度(Space Complexity): O(*n*)**

Because the array for storing POIs indexes which combine to the convex-hull is up to n pairs, so the time complexity is O(n).

## **2. Brute-force Algorithm and Exhaustive search Algorithm**

一張含有 圖表, 文字, 寫生, 方案 的圖片

自動產生的描述

func get\_distance**(**x1**,** y1**,** x2**,** y2**):** result**,** result is double

**return** sqrt**((**x1 **-** x2**)** **^** 2 **+** **(**y1 **-** y2**)** **^** 2**)**

func compare\_distance**(**min\_distance**,** current\_distance**,** i**,** j**,** index1**,** index2**)** **:** result**,** result is double

**if** i **!=** j then

**if** min\_distance **>=** current\_distance then

index1 **<-** i

index2 **<-** j

**return** current\_distance

end **if**

end **if**

**return** min\_distance

func binary\_search**(**array**,** target**,** left**,** right**):** result**,** result is int

**while** left **<=** right **do**

result **<-** **(**left **+** right**)** **/** 2

**if** array**[**result**]** **=** target then

**return** result

**else** **if** array**[**result**]** **>** target then

right **<-** result **-** 1**;**

**else**

left **<-** result **+** 1**;**

end **if**

end **while**

func shortest\_Path**(**location**[**n**][**2**])**

k **<-** 0

next\_index **<-** 0

current\_index **<-** 0

total\_distance **<-** 0

used\_POI <- []

min\_POI <- []

**for** i **(**0 to n**-**1**)** **do**

used\_POI**.**append**(**i**)**

end **for**

**for** i **(**0 to n**-**2**)** **do**

**if** used\_POI**[**0**]** **==** current\_index then

min\_distance **=** get\_distance**(**location**[**used\_POI**[**1**]][**0**],** location**[**used\_POI**[**1**]][**1**],**

location**[**current\_index**][**0**],** location**[**current\_index**][**1**])**

**else**

min\_distance **=** get\_distance**(**location**[**used\_POI**[**0**]][**0**],** location**[**used\_POI**[**0**]][**1**],**

location**[**current\_index**][**0**],** location**[**current\_index**][**1**])**

**for** j **(**0 to n**-**1**)** **do**

current\_distance **<-** get\_distance**(**location**[**used\_POI**[**j**]][**0**],** location**[**used\_POI**[**j**]][**1**],**

location**[**current\_index**][**0**],** location**[**current\_index**][**1**])**

min\_distance **=** compare\_distance**(**min\_distance**,** current\_distance**,**

current\_index**,** used\_POI**[**j**],** used\_index**,** next\_index**)**

end **for**

current\_index **<-** next\_index

total\_distance **+=** min\_distance

min\_POI**.**append**(**used\_index**)**

min\_POI**.**append**(**next\_index**)**

used\_index **<-** binary\_search**(**used\_POI**,** used\_index**,** 0**,** used\_POI**.**size**()** **-** 1**)**

used\_POI**.**erase**(**used\_index**)**

print**(**"(" **+** location**[**min\_POI**[**k**]][**0**]** **+** ", " **+** location**[**min\_POI**[**k**]][**1**]** **+** ") and

**(**" + location[min\_POI[k + 1]][0] + "**,** " + location[min\_POI[k + 1]][1] + "**)**\n")

k **+=** 2

end **for**

print**(**"(" **+** location**[**used\_POI**[**0**]][**0**]** **+** ", " **+** location**[**used\_POI**[**0**]][**1**]** **+** ") and

**(**" + location[0][0] + "**,** " + location[0][1] + "**)**\n")

total\_distance **+=** get\_distance**(**location**[**used\_POI**[**0**]][**0**],** location**[**used\_POI**[**0**]][**1**],**

location**[**0**][**0**],** location**[**0**][**1**])**

print**(**total\_distance **+** "\n"**)**

1. **演算法時間複雜度(Time Complexity): O()**

Because in the outer for-loop, we compare between binary search and the inner for-loop. Obviously, when n > 2, then  **>** n. Thus (n) \* () is O().

1. **演算法空間複雜度(Space Complexity): O(*n*)**

Because using used\_POI for n times connection and min\_POI for storing n POIs for each twice, so the space complexity is 3n.

## **3. Convex-Hull-TSP Algorithm**

一張含有 文字, 圖表, 方案, 字型 的圖片

自動產生的描述

func get\_distance**(**x1**,** y1**,** x2**,** y2**):** result**,** result is double

**return** sqrt**((**x1 **-** x2**)** **^** 2 **+** **(**y1 **-** y2**)** **^** 2**)**

func is\_able\_to\_in\_line**(**location**[**n**][**2**],** convex\_hull**[],** side\_number**,** x**,** y**,** able\_in\_line**[][**4**])**

**if** location**[**convex\_hull**[**2 **\*** side\_number**]][**0**]** **>** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**0**]** then

left\_x **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**0**]**

right\_x **<** **-**location**[**convex\_hull**[**2 **\*** side\_number**]][**0**]**

**if** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]** **>** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]** then

left\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]**

right\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]**

**else**

left\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]**

right\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]**

end **if**

end **if**

**else**

left\_x **<-** location**[**convex\_hull**[**2 **\*** side\_number**]][**0**]**

right\_x **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**0**]**

**if** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]** **>** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]** then

left\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]**

right\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]**

**else**

left\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number**]][**1**]**

right\_y **<-** location**[**convex\_hull**[**2 **\*** side\_number **+** 1**]][**1**]**

end **if**

end **if**

**if** left\_x **<=** x **and** x **<=** right\_x **and** left\_y **<=** y **and** y **<=** right\_y then

able\_in\_line**.**append**([**side\_number**,** x**,** y**])**

end **if**

func link\_points\_in\_the\_side**(**location**[**n**][**2**],** convex\_hull\_rest\_POI**[],** convex\_hull**[],** index**,** count**,** total\_distance**)**

k **<-** 0

next\_index **<-** 0

print**(**"(" **+** location**[**convex\_hull**[**2 **\*** index**]][**0**]** **+** ", " **+** location**[**convex\_hull**[**2 **\*** index**]][**1**]** **+** ") to "**)**

previous\_index **<-** convex\_hull**[**2 **\*** index**]**

**while** k **<** convex\_hull\_rest\_POI**.**size**()** **and** convex\_hull\_rest\_POI**[**k**][**1**]** **!=** index **do**

k**++**

end **while**

**for** j **(**0 to count**)** **do**

**for** l **(**k to convex\_hull\_rest\_POI**.**size**())** **do**

next\_index **<-** k**;**

**if** location**[**convex\_hull**[**2 **\*** index**]][**0**]** **>** location**[**convex\_hull**[**2 **\*** index **+** 1**]][**0**]** **and** convex\_hull\_rest\_POI**[**l**][**1**]** **=** index then

**if** convex\_hull\_rest\_POI**[**l**][**2**]** **>** convex\_hull\_rest\_POI**[**next\_index**][**2**]** then

next\_index **<-** l

end **if**

**else** **if** location**[**convex\_hull**[**2 **\*** index**]][**0**]** **<** location**[**convex\_hull**[**2 **\*** index **+** 1**]][**0**]** **and** convex\_hull\_rest\_POI**[**l**][**1**]** **=** index then

**if** convex\_hull\_rest\_POI**[**l**][**2**]** **<** convex\_hull\_rest\_POI**[**next\_index**][**2**]** then

next\_index **<-** l

end **if**

end **if**

end **for**

total\_distance **+=** get\_distance**(**location**[**previous\_index**][**0**],** location**[**previous\_index**][**1**],**

location**[**convex\_hull\_rest\_POI**[**next\_index**][**0**]][**0**],** location**[**convex\_hull\_rest\_POI**[**next\_index**][**0**]][**1**])**

previous\_index **<-** convex\_hull\_rest\_POI**[**next\_index**][**0**]**

print**(**"(" **+** location**[**convex\_hull\_rest\_POI**[**next\_index**][**0**]][**0**]** **+** ", " **<<** location**[**convex\_hull\_rest\_POI**[**next\_index**][**0**]][**1**]** **+** ")\n

**(**" + location[convex\_hull\_rest\_POI[next\_index][0]][0] + "**,** " + location[convex\_hull\_rest\_POI[next\_index][0]][1] + "**)** to ")

convex\_hull\_rest\_POI**.**erase**(**convex\_hull\_rest\_POI**.**begin**()** **+** next\_index**)**

k **<-** 0

**while** k **<** convex\_hull\_rest\_POI**.**size**()** **and** convex\_hull\_rest\_POI**[**k**][**1**]** **!=** index **do**

k **+=** 1

end **while**

end **for**

total\_distance **+=** get\_distance**(**location**[**previous\_index**][**0**],** location**[**previous\_index**][**1**],**

location**[**convex\_hull**[**2 **\*** index **+** 1**]][**0**],** location**[**convex\_hull**[**2 **\*** index **+** 1**]][**1**])**

print**(**"(" **+** location**[**convex\_hull**[**2 **\*** index **+** 1**]][**0**]** **+** ", " **+** location**[**convex\_hull**[**2 **\*** index **+** 1**]][**1**]** **+** ")\n"**)**

func Convex\_Hull\_TSP\_Algorithm**(**location**[**n**][**2**],** convex\_hull**[])**

minimum\_index **<-** 0

total\_distance **<-** 0

side\_point\_count

rest\_POI **<-** **[]**

side\_point\_count **<-** **[]**

traveller **<-** **[]**

line\_coefficient **<-** **[]**

able\_reflection\_side **<-** **[]**

convex\_hull\_rest\_POI **<-** **[]**

line\_coefficient**.**resize**(**convex\_hull**.**size**()** **/** 2**)**

side\_point\_count**.**resize**(**convex\_hull**.**size**()** **/** 2**)**

side\_point\_count**.**fill**(**0**)**

**for** i **(**0 to n**-**1**)** **do**

flag **<-** **true**

**for** j **(**0 to convex\_hull**.**size**()** **-** 1**)** **do**

**if** i **=** convex\_hull**[**j**]** then

flag **<-** **false**

end **if**

end **for**

**if** flag then

rest\_POI**.**append**(**i**)**

end **if**

end **for**

**for** i **(**0 to convex\_hull**.**size**()** **-** 1**)** **do**

coefficient **<-** **(**location**[**convex\_hull**[**i**]][**1**]** **-** location**[**convex\_hull**[**i **+** 1**]][**1**])** **/** **(**location**[**convex\_hull**[**i**]][**0**]** **-** location**[**convex\_hull**[**i **+** 1**]][**0**])**

constant **<-** location**[**convex\_hull**[**i**]][**1**]** **-** coefficient **\*** location**[**convex\_hull**[**i**]][**0**]**

line\_coefficient**[**i **/** 2**].**append**(**location**[**convex\_hull**[**i**]][**1**]** **-** location**[**convex\_hull**[**i **+** 1**]][**1**])**

line\_coefficient**[**i **/** 2**].**append**(**location**[**convex\_hull**[**i**]][**0**]** **-** location**[**convex\_hull**[**i **+** 1**]][**0**])**

line\_coefficient**[**i **/** 2**].**append**((**location**[**convex\_hull**[**i**]][**0**]** **-** location**[**convex\_hull**[**i **+** 1**]][**0**])** **\*** constant**)**

end **for**

**for** i **(**0 to rest\_POI**.**size**()** **-** 1**)** **do**

able\_reflection\_side**.**clear**()**

minimum\_index **<-** 0

**for** j **(**0 to line\_coefficient**.**size**()** **-** 1**)** **do**

reflection\_coefficient **<-** **-**1 **\*** **(**line\_coefficient**[**j**][**0**]** **\*** location**[**rest\_POI**[**i**]][**0**]** **-** line\_coefficient**[**j**][**1**]** **\*** location**[**rest\_POI**[**i**]][**1**]** **+** line\_coefficient**[**j**][**2**])** **/** **(**pow**(**line\_coefficient**[**j**][**0**],** 2**)** **-** pow**(**line\_coefficient**[**j**][**1**],** 2**))**

rest\_POI\_x **<-** location**[**rest\_POI**[**i**]][**0**]** **+** reflection\_coefficient **\*** line\_coefficient**[**j**][**0**]**

rest\_POI\_y **<-** location**[**rest\_POI**[**i**]][**1**]** **+** reflection\_coefficient **\*** line\_coefficient**[**j**][**1**]**

**if** j **=** 0 then

minimum\_distance **<-** get\_distance**(**location**[**rest\_POI**[**i**]][**0**],** location**[**rest\_POI**[**i**]][**1**],** rest\_POI\_x**,** rest\_POI\_y**)**

**else**

current\_distance **<-** get\_distance**(**location**[**rest\_POI**[**i**]][**0**],** location**[**rest\_POI**[**i**]][**1**],** rest\_POI\_x**,** rest\_POI\_y

**if** minimum\_distance **>** current\_distance then

minimum\_distance **<-** current\_distance

minimum\_index **<-** j

end **if**

end **if**

is\_able\_to\_in\_line**(**location**,** convex\_hull**,** j **\*** 1.0**,** rest\_POI\_x**,** rest\_POI\_y**,** able\_reflection\_side**)**

end **for**

**if** able\_reflection\_side**.**empty**()** then

convex\_hull\_rest\_POI**.**append**([**rest\_POI**[**i**]** **\*** 1.0**,** minimum\_index **\*** 1.0**,** rest\_POI\_x**,** rest\_POI\_y**])**

**else**

minimum\_distance **<-** get\_distance**(**location**[**rest\_POI**[**i**]][**0**],** location**[**rest\_POI**[**i**]][**1**],** able\_reflection\_side**[**0**][**1**],** able\_reflection\_side**[**0**][**2**])**

**for** j **(**1 to able\_reflection\_side**.**size**()** **-** 1**)** **do**

minimum\_index **<-** 0

current\_distance **<-** get\_distance**(**location**[**rest\_POI**[**i**]][**0**],** location**[**rest\_POI**[**i**]][**1**],** able\_reflection\_side**[**j**][**1**],** able\_reflection\_side**[**j**][**2**])**

**if** minimum\_distance **>** current\_distance then

minimum\_distance **<-** current\_distance

minimum\_index **<-** j

end **if**

end **for**

convex\_hull\_rest\_POI**.**append**([**rest\_POI**[**i**]** **\*** 1.0**,** minimum\_index **\*** 1.0**,** able\_reflection\_side**[**minimum\_index**][**1**],** able\_reflection\_side**[**minimum\_index**][**2**]])**

end **if**

side\_point\_count**[**minimum\_index**]** **+=** 1

end **for**

**for** i **(**0 to side\_point\_count**.**size**()** **-** 1**)**

**if** side\_point\_count**[**i**]** **=** 0 then

print**(**"(" **+** location**[**convex\_hull**[**2 **\*** i**]][**0**]** **+** ", " **+** location**[**convex\_hull**[**2 **\*** i**]][**1**]** **+** ") to (" **+** location**[**convex\_hull**[**2 **\*** i **+** 1**]][**0**]** **+** ", " **+** location**[**convex\_hull**[**2 **\*** i **+** 1**]][**1**]** **+** ")\n"**)**

total\_distance **+=** get\_distance**(**location**[**convex\_hull**[**2 **\*** i**]][**0**],** location**[**convex\_hull**[**2 **\*** i**]][**1**],** location**[**convex\_hull**[**2 **\*** i **+** 1**]][**0**],** location**[**convex\_hull**[**2 **\*** i **+** 1**]][**1**])**

**else**

link\_points\_in\_the\_side**(**location**,** convex\_hull\_rest\_POI**,** convex\_hull**,** i**,** side\_point\_count**[**i**],** total\_distance**)**

end **if**

end **for**

print**(**"The Convex-Hull-TSP Algorithm max distance is " **+** total\_distance**)**

1. **演算法時間複雜度: O()**

Because the nest double loop for checking each reflection point on each convex-hull side is up to (n-3+1) \* (n) = -2n, so the time complexity is O().

1. **演算法空間複雜度****: O(*n*)**

I use several 1-D or 2-D(n by 2) vector, so the space complexity is O(n).

# (四) 程式設計環境架構

## **1. 程式語言**

C++ in Microsoft Windows10 64-bits operating system

## **2. 程式開發工具**

Microsoft Visual Studio 2019

## **3. 電腦硬體**

CPU: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

Main Memory: 16.0 GB

顯示卡1: Intel® UHD Graphics

顯示卡2: NVIDIA GeForce RTX 3050 Ti Laptop GPU

## **4. 作業系統**

Windows 10 家用版

版本: 22H2

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

1. 主程式

1. **Cloest(POIs)**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

void get\_POIs\_location(vector<vector<double>>&);

void closest(vector<vector<double>>);

double get\_distance(double, double, double, double);

double found\_smaller\_distance(double, double, vector<int>&, int, int);

int main(void) {

vector<vector<double>> location;

get\_POIs\_location(location);

closest(location);

return 0;

}

void get\_POIs\_location(vector<vector<double>>& location) {

ifstream in\_file;

string file\_name;

char longitude, latitude;

double longitude\_value, latitude\_value;

cout << "Enter the map file name in (.txt) format: " ;

cin >> file\_name;

in\_file.open(file\_name+".txt");

while (!in\_file.eof()) {

in\_file >> longitude >> longitude\_value >> latitude >> latitude\_value;

if (longitude == 'W') {

longitude\_value \*= -1;

}

if (latitude\_value == 'S') {

latitude\_value \*= -1;

}

location.push\_back({ longitude\_value, latitude\_value });

}

in\_file.close();

}

void closest(vector<vector<double>> location) {

double min\_distance, current\_distance;

int i, j;

vector<int> closest\_POI;

min\_distance = get\_distance(location[0][0], location[0][1], location[1][0], location[1][1]);

for (i = 0; i < location.size() - 1; i++) {

for (j = i + 1; j < location.size(); j++) {

current\_distance = get\_distance(location[i][0], location[i][1], location[j][0], location[j][1]);

min\_distance = found\_smaller\_distance(current\_distance, min\_distance, closest\_POI, i, j);

}

}

for (i = 0; i < closest\_POI.size(); i += 2) {

cout << "Found a pair of the closest POIs: (" << location[closest\_POI[i]][0] << ", " << location[closest\_POI[i]][1] << ") and (" << location[closest\_POI[i + 1]][0] << ", " << location[closest\_POI[i + 1]][1] << ").\n";

}

cout << "The distance of the closest POIs is " << min\_distance << "\n";

}

double get\_distance(double x1, double y1, double x2, double y2) {

return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

double found\_smaller\_distance(double current\_distance, double min\_distance, vector<int>& closest\_POI, int i, int j) {

if (current\_distance <= min\_distance) {

if (current\_distance < min\_distance && !closest\_POI.empty()) {

closest\_POI.clear();

}

closest\_POI.push\_back(i);

closest\_POI.push\_back(j);

return current\_distance;

}

return min\_distance;

}

1. **Max\_ConvexHull(POIs)**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

void get\_POIs\_location(vector<vector<double>>&);

vector<int> max\_convex\_hull(vector<vector<double>>);

double get\_distance(double, double, double, double);

int check\_same\_side(double, double);

int main(void) {

vector<vector<double>> location;

get\_POIs\_location(location);

max\_convex\_hull(location);

return 0;

}

void get\_POIs\_location(vector<vector<double>>& location) {

ifstream in\_file;

string file\_name;

char longitude, latitude;

double longitude\_value, latitude\_value;

cout << "Enter the map file name in (.txt) format: ";

cin >> file\_name;

in\_file.open(file\_name + ".txt");

if (!in\_file.is\_open()) {

cout << "Failed to open the file.\n";

exit(1);

}

while (!in\_file.eof()) {

in\_file >> longitude >> longitude\_value >> latitude >> latitude\_value;

if (longitude == 'W') {

longitude\_value \*= -1;

}

if (latitude\_value == 'S') {

latitude\_value \*= -1;

}

location.push\_back({ longitude\_value, latitude\_value });

}

in\_file.close();

}

vector<int> max\_convex\_hull(vector<vector<double>> location)

{

int i, j, k, flag1, flag2;

double divide, coefficient;

double perimeter = 0, area = 0;

vector<int> convex\_hull;

for (i = 0; i < location.size() - 1; i++) {

for (j = i + 1; j < location.size(); j++) {

if (location[j][0] - location[i][0] == 0) {

continue;

}

coefficient = (location[j][1] - location[i][1]) / (location[j][0] - location[i][0]);

divide = location[i][1] - coefficient \* location[i][0];

k = 0;

flag1 = 0;

do {

if (k != i && k != j) {

flag1 = check\_same\_side(divide, location[k][1] - coefficient \* location[k][0]);

}

k++;

} while (flag1 == 0);

flag2 = flag1; //avoid n = 3

for (; k < location.size() - 1; k++) {

if (k != i && k != j) {

flag2 = check\_same\_side(divide, location[k][1] - coefficient \* location[k][0]);

if (flag1 != flag2) {

break;

}

}

}

if (flag1 == flag2) {

convex\_hull.push\_back(i);

convex\_hull.push\_back(j);

perimeter += get\_distance(location[i][0], location[i][1], location[j][0], location[j][1]);

area += (location[i][0] \* location[j][1] - location[j][0] \* location[i][1]) / 2;

}

}

}

cout << "The convex-hull is finished as following:\n";

for (i = 0; i < convex\_hull.size(); i += 2) {

cout << "(" << location[convex\_hull[i]][0] << ", " << location[convex\_hull[i]][1] << ") and (" << location[convex\_hull[i + 1]][0] << ", " << location[convex\_hull[i + 1]][1] << ")\n";

}

cout << "The max perimeter of the convex-hull is " << perimeter << "\nThe max area of the convex-hull is " << abs(area) << "\n";

return convex\_hull;

}

double get\_distance(double x1, double y1, double x2, double y2) {

return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

int check\_same\_side(double standard, double compare) {

if (standard > compare) {

return 1;

}

return 2;

}

1. **Shortest\_Path(POIs)**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

void get\_POIs\_location(vector<vector<double>>&);

void shortest\_Path(vector<vector<double>>);

double get\_distance(double, double, double, double);

double compare\_distance(double, double, int, int, int\*, int\*);

int binary\_search(vector<int>, int, int, int);

int main(void) {

vector<vector<double>> location;

get\_POIs\_location(location);

shortest\_Path(location);

return 0;

}

void get\_POIs\_location(vector<vector<double>>& location) {

ifstream in\_file;

string file\_name;

char longitude, latitude;

double longitude\_value, latitude\_value;

cout << "Enter the map file name in (.txt) format: ";

cin >> file\_name;

in\_file.open(file\_name + ".txt");

if (!in\_file.is\_open()) {

cout << "Failed to open the file.\n";

exit(1);

}

while (!in\_file.eof()) {

in\_file >> longitude >> longitude\_value >> latitude >> latitude\_value;

if (longitude == 'W') {

longitude\_value \*= -1;

}

if (latitude\_value == 'S') {

latitude\_value \*= -1;

}

location.push\_back({ longitude\_value, latitude\_value });

}

in\_file.close();

}

void shortest\_Path(vector<vector<double>> location) {

int i, j, used\_index;

int k = 0, next\_index = 0, current\_index = 0;

vector<int> used\_POI, min\_POI; //space: n+2n

double min\_distance, current\_distance;

double total\_distance = 0;

cout << "The minimum n POIs traversal is finished as following:\n";

for (i = 0; i < location.size(); i++) {

used\_POI.push\_back(i);

}

for (i = 0; i < location.size() - 1; i++) {

if (used\_POI[0] == current\_index) {

min\_distance = get\_distance(location[used\_POI[1]][0], location[used\_POI[1]][1], location[current\_index][0], location[current\_index][1]);

}

else {

min\_distance = get\_distance(location[used\_POI[0]][0], location[used\_POI[0]][1], location[current\_index][0], location[current\_index][1]);

}

for (j = 0; j < used\_POI.size(); j++) {

current\_distance = get\_distance(location[used\_POI[j]][0], location[used\_POI[j]][1], location[current\_index][0], location[current\_index][1]);

min\_distance = compare\_distance(min\_distance, current\_distance, current\_index, used\_POI[j], &used\_index, &next\_index);

}

current\_index = next\_index;

total\_distance += min\_distance;

min\_POI.push\_back(used\_index);

min\_POI.push\_back(next\_index);

used\_index = binary\_search(used\_POI, used\_index, 0, used\_POI.size() - 1);

used\_POI.erase(used\_POI.begin() + used\_index);

cout << "(" << location[min\_POI[k]][0] << ", " << location[min\_POI[k]][1] << ") and (" << location[min\_POI[k + 1]][0] << ", " << location[min\_POI[k + 1]][1] << ")\n";

k += 2;

}

cout << "(" << location[used\_POI[0]][0] << ", " << location[used\_POI[0]][1] << ") and (" << location[0][0] << ", " << location[0][1] << ")\n";

total\_distance += get\_distance(location[used\_POI[0]][0], location[used\_POI[0]][1], location[0][0], location[0][1]);

cout << "The minimum distance is " << total\_distance << "\n";

}

double get\_distance(double x1, double y1, double x2, double y2) {

return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

double compare\_distance(double min\_distance, double current\_distance, int i, int j, int\* index1, int\* index2) {

if (i != j) {

if (min\_distance >= current\_distance) {

\*index1 = i;

\*index2 = j;

return current\_distance;

}

}

return min\_distance;

}

int binary\_search(vector<int> array, int target, int left, int right) {

int middle;

do {

middle = (left + right) / 2;

if (array[middle] == target) {

return middle;

}

else if (array[middle] > target) {

right = middle - 1;

}

else {

left = middle + 1;

}

} while (left <= right);

return -1;

}

1. **Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull)**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

void get\_POIs\_location(vector<vector<double>>&);

void Convex\_Hull\_TSP\_Algorithm(vector<vector<double>>, vector<int>);

double get\_distance(double, double, double, double);

vector<int> max\_convex\_hull(vector<vector<double>>);

int check\_same\_side(double, double);

void is\_able\_to\_in\_line(vector<vector<double>>, vector<int>, double, double, double, vector<vector<double>>&);

void link\_points\_in\_the\_side(vector<vector<double>>, vector<vector<double>>, vector<int>, int, int, double&);

int main(void) {

vector<vector<double>> location;

get\_POIs\_location(location);

vector<int> convex\_hull = max\_convex\_hull(location);

Convex\_Hull\_TSP\_Algorithm(location, convex\_hull);

return 0;

}

void get\_POIs\_location(vector<vector<double>>& location) {

ifstream in\_file;

string file\_name;

char longitude, latitude;

double longitude\_value, latitude\_value;

cout << "Enter the map file name in (.txt) format: ";

cin >> file\_name;

in\_file.open(file\_name + ".txt");

if (!in\_file.is\_open()) {

cout << "Failed to open the file.\n";

exit(1);

}

while (!in\_file.eof()) {

in\_file >> longitude >> longitude\_value >> latitude >> latitude\_value;

if (longitude == 'W') {

longitude\_value \*= -1;

}

if (latitude\_value == 'S') {

latitude\_value \*= -1;

}

location.push\_back({ longitude\_value, latitude\_value });

}

in\_file.close();

}

double get\_distance(double x1, double y1, double x2, double y2) {

return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

vector<int> max\_convex\_hull(vector<vector<double>> location)

{

int i, j, flag2;

int k = 0, flag1 = 0;

double divide, coefficient;

double perimeter = 0, area = 0;

vector<int> convex\_hull;

for (i = 0; i < location.size() - 1; i++) {

for (j = i + 1; j < location.size(); j++) {

if (location[j][0] - location[i][0] == 0) {

continue;

}

coefficient = (location[j][1] - location[i][1]) / (location[j][0] - location[i][0]);

divide = location[i][1] - coefficient \* location[i][0];

k = 0;

do {

if (k != i && k != j) {

flag1 = check\_same\_side(divide, location[k][1] - coefficient \* location[k][0]);

}

k++;

} while (flag1 == 0);

flag2 = flag1; //avoid n = 3

for (; k < location.size() - 1; k++) {

if (k != i && k != j) {

flag2 = check\_same\_side(divide, location[k][1] - coefficient \* location[k][0]);

if (flag1 != flag2) {

break;

}

}

}

if (flag1 == flag2) {

convex\_hull.push\_back(i);

convex\_hull.push\_back(j);

perimeter += get\_distance(location[i][0], location[i][1], location[j][0], location[j][1]);

area += (location[i][0] \* location[j][1] - location[j][0] \* location[i][1]) / 2;

}

}

}

return convex\_hull;

}

int check\_same\_side(double standard, double compare) {

if (standard > compare) {

return 1;

}

return 2;

}

void Convex\_Hull\_TSP\_Algorithm(vector<vector<double>> location, vector<int> convex\_hull) {

int i, j;

int minimum\_index = 0;

vector<int> rest\_POI, side\_point\_count, traveller;

double coefficient, constant, reflection\_coefficient, rest\_POI\_x, rest\_POI\_y, minimum\_distance, current\_distance;

double total\_distance = 0;

vector<vector<double>> line\_coefficient, able\_reflection\_side, convex\_hull\_rest\_POI;

bool flag;

line\_coefficient.resize(convex\_hull.size() / 2);

side\_point\_count.resize(convex\_hull.size() / 2);

fill(side\_point\_count.begin(), side\_point\_count.end(), 0);

for (i = 0; i < location.size(); i++) {

flag = true;

for (j = 0; j < convex\_hull.size(); j++) {

if (i == convex\_hull[j]) {

flag = false;

}

}

if (flag) {

rest\_POI.push\_back(i);

}

}

for (i = 0; i < convex\_hull.size(); i += 2) {

coefficient = (location[convex\_hull[i]][1] - location[convex\_hull[i + 1]][1]) / (location[convex\_hull[i]][0] - location[convex\_hull[i + 1]][0]);

constant = location[convex\_hull[i]][1] - coefficient \* location[convex\_hull[i]][0];

line\_coefficient[i / 2].push\_back(location[convex\_hull[i]][1] - location[convex\_hull[i + 1]][1]);

line\_coefficient[i / 2].push\_back(location[convex\_hull[i]][0] - location[convex\_hull[i + 1]][0]);

line\_coefficient[i / 2].push\_back((location[convex\_hull[i]][0] - location[convex\_hull[i + 1]][0]) \* constant);

}

for (i = 0; i < rest\_POI.size(); i++) {

able\_reflection\_side.clear();

minimum\_index = 0;

for (j = 0; j < line\_coefficient.size(); j++) {

reflection\_coefficient = -1 \* (line\_coefficient[j][0] \* location[rest\_POI[i]][0] - line\_coefficient[j][1] \* location[rest\_POI[i]][1] + line\_coefficient[j][2]) / (pow(line\_coefficient[j][0], 2) - pow(line\_coefficient[j][1], 2));

rest\_POI\_x = location[rest\_POI[i]][0] + reflection\_coefficient \* line\_coefficient[j][0];

rest\_POI\_y = location[rest\_POI[i]][1] + reflection\_coefficient \* line\_coefficient[j][1];

if (j == 0) {

minimum\_distance = get\_distance(location[rest\_POI[i]][0], location[rest\_POI[i]][1], rest\_POI\_x, rest\_POI\_y);

}

else {

current\_distance = get\_distance(location[rest\_POI[i]][0], location[rest\_POI[i]][1], rest\_POI\_x, rest\_POI\_y);

if (minimum\_distance > current\_distance) {

minimum\_distance = current\_distance;

minimum\_index = j;

}

}

is\_able\_to\_in\_line(location, convex\_hull, j \* 1.0, rest\_POI\_x, rest\_POI\_y, able\_reflection\_side);

}

if (able\_reflection\_side.empty()) {

convex\_hull\_rest\_POI.push\_back({ rest\_POI[i] \* 1.0, minimum\_index \* 1.0, rest\_POI\_x, rest\_POI\_y });

}

else {

minimum\_distance = get\_distance(location[rest\_POI[i]][0], location[rest\_POI[i]][1], able\_reflection\_side[0][1], able\_reflection\_side[0][2]);

minimum\_index = 0;

for (j = 1; j < able\_reflection\_side.size(); j++) {

current\_distance = get\_distance(location[rest\_POI[i]][0], location[rest\_POI[i]][1], able\_reflection\_side[j][1], able\_reflection\_side[j][2]);

if (minimum\_distance > current\_distance) {

minimum\_distance = current\_distance;

minimum\_index = j;

}

}

convex\_hull\_rest\_POI.push\_back({ rest\_POI[i] \* 1.0, minimum\_index \* 1.0, able\_reflection\_side[minimum\_index][1], able\_reflection\_side[minimum\_index][2] });

}

side\_point\_count[minimum\_index]++;

}

cout << "The Convex-Hull-TSP Algorithm is finished as following:\n";

for (i = 0; i < side\_point\_count.size(); i++) {

if (side\_point\_count[i] == 0) {

cout << "(" << location[convex\_hull[2 \* i]][0] << ", " << location[convex\_hull[2 \* i]][1] << ") to (" << location[convex\_hull[2 \* i + 1]][0] << ", " << location[convex\_hull[2 \* i + 1]][1] << ")\n";

total\_distance += get\_distance(location[convex\_hull[2 \* i]][0], location[convex\_hull[2 \* i]][1], location[convex\_hull[2 \* i + 1]][0], location[convex\_hull[2 \* i + 1]][1]);

}

else {

link\_points\_in\_the\_side(location, convex\_hull\_rest\_POI, convex\_hull, i, side\_point\_count[i], total\_distance);

}

}

cout << "The Convex-Hull-TSP Algorithm distance is " << total\_distance;

}

void is\_able\_to\_in\_line(vector<vector<double>> location, vector<int> convex\_hull, double side\_number, double x, double y, vector<vector<double>>& able\_in\_line)

{

double left\_x, left\_y, right\_x, right\_y;

if (location[convex\_hull[2 \* side\_number]][0] > location[convex\_hull[2 \* side\_number + 1]][0]) {

left\_x = location[convex\_hull[2 \* side\_number + 1]][0];

right\_x = location[convex\_hull[2 \* side\_number]][0];

if (location[convex\_hull[2 \* side\_number]][1] > location[convex\_hull[2 \* side\_number + 1]][1]) {

left\_y = location[convex\_hull[2 \* side\_number + 1]][1];

right\_y = location[convex\_hull[2 \* side\_number]][1];

}

else {

left\_y = location[convex\_hull[2 \* side\_number]][1];

right\_y = location[convex\_hull[2 \* side\_number + 1]][1];

}

}

else {

left\_x = location[convex\_hull[2 \* side\_number]][0];

right\_x = location[convex\_hull[2 \* side\_number + 1]][0];

if (location[convex\_hull[2 \* side\_number]][1] > location[convex\_hull[2 \* side\_number + 1]][1]) {

left\_y = location[convex\_hull[2 \* side\_number + 1]][1];

right\_y = location[convex\_hull[2 \* side\_number]][1];

}

else {

left\_y = location[convex\_hull[2 \* side\_number]][1];

right\_y = location[convex\_hull[2 \* side\_number + 1]][1];

}

}

if (left\_x <= x && x <= right\_x && left\_y <= y && y <= right\_y) {

able\_in\_line.push\_back({ side\_number, x, y });

}

}

void link\_points\_in\_the\_side(vector<vector<double>> location, vector<vector<double>> convex\_hull\_rest\_POI, vector<int> convex\_hull, int index, int count, double& total\_distance)

{

int j, l;

int k = 0, next\_index = 0, previous\_index;

cout << "(" << location[convex\_hull[2 \* index]][0] << ", " << location[convex\_hull[2 \* index]][1] << ") to ";

previous\_index = convex\_hull[2 \* index];

while (k < convex\_hull\_rest\_POI.size() && convex\_hull\_rest\_POI[k][1] != index) {

k++;

}

for (j = 0; j < count; j++) {

for (l = k; l < convex\_hull\_rest\_POI.size(); l++) {

next\_index = k;

if (location[convex\_hull[2 \* index]][0] > location[convex\_hull[2 \* index + 1]][0] && convex\_hull\_rest\_POI[l][1] == index)

{

if (convex\_hull\_rest\_POI[l][2] > convex\_hull\_rest\_POI[next\_index][2]) {

next\_index = l;

}

}

else if (location[convex\_hull[2 \* index]][0] < location[convex\_hull[2 \* index + 1]][0] && convex\_hull\_rest\_POI[l][1] == index)

{

if (convex\_hull\_rest\_POI[l][2] < convex\_hull\_rest\_POI[next\_index][2]) {

next\_index = l;

}

}

}

total\_distance += get\_distance(location[previous\_index][0], location[previous\_index][1], location[convex\_hull\_rest\_POI[next\_index][0]][0], location[convex\_hull\_rest\_POI[next\_index][0]][1]);

previous\_index = convex\_hull\_rest\_POI[next\_index][0];

cout << "(" << location[convex\_hull\_rest\_POI[next\_index][0]][0] << ", " << location[convex\_hull\_rest\_POI[next\_index][0]][1] << ")\n(" << location[convex\_hull\_rest\_POI[next\_index][0]][0] << ", " << location[convex\_hull\_rest\_POI[next\_index][0]][1] << ") to ";

convex\_hull\_rest\_POI.erase(convex\_hull\_rest\_POI.begin() + next\_index);

k = 0;

while (k < convex\_hull\_rest\_POI.size() && convex\_hull\_rest\_POI[k][1] != index) {

k++;

}

}

total\_distance += get\_distance(location[previous\_index][0], location[previous\_index][1], location[convex\_hull[2 \* index + 1]][0], location[convex\_hull[2 \* index + 1]][1]);

cout << "(" << location[convex\_hull[2 \* index + 1]][0] << ", " << location[convex\_hull[2 \* index + 1]][1] << ")\n";

}

2. Input Code Format

By using longitude and latitude decimal dimension format.

(1) Map1

E 1.00000 N 1.00000

E 1.50000 N 2.00000

E 1.50000 N 3.50000

E 0.50000 N 3.90000

E 1.50000 N 5.00000

E 5.00000 N 5.70000

E 5.30000 N 4.60000

E 6.00000 N 4.30000

E 4.70000 N 0.30000

E 3.44000 N 1.00000

3. Output Code Format

* 1. **Closet(POIs)**

Found a pair opf the closest POIs: (5.3, 4.6) and (6, 4.3).

The distance of the closest POIs is 0.761577

1. **Max\_ConvexHull(POIs)**

The convex-hull is finished as following:

(1, 1) and (0.5, 3.9)

(1, 1) and (4.7, 0.3)

(0.5, 3.9) and (1.5, 5)

(1.5, 5) and (5, 5.7)

(5, 5.7) and (6, 4.3)

(6, 4.3) and (4.7, 0.3)

The max perimeter of the convex-hull is 17.6908

The max area of the convex-hull is 25.955

1. **Shortest\_Path(POIs)**

The minimum n POIs traversal is finished as following:

(1, 1) and (1.5, 2)

(1.5, 2) and (1.5, 3.5)

(1.5, 3.5) and (0.5, 3.9)

(0.5, 3.9) and (1.5, 5)

(1.5, 5) and (5, 5.7)

(5, 5.7) and (5.3, 4.6)

(5.3, 4.6) and (6, 4.3)

(6, 4.3) and (3.44, 1)

(3.44, 1) and (4.7, 0.3)

(4.7, 0.3) and (1, 1)

The minimum distance is 20.0363

1. **Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull)**

The Convex-Hull-TSP Algorithm is finished as following:

(1, 1) to (1.5, 2)

(1.5, 2) to (1.5, 3.5)

(1.5, 3.5) to (0.5, 3.9)

(1, 1) to (3.44, 1)

(3.44, 1) to (4.7, 0.3)

(0.5, 3.9) to (1.5, 5)

(1.5, 5) to (5, 5.7)

(5, 5.7) to (6, 4.3)

(6, 4.3) to (5.3, 4.6)

(5.3, 4.6) to (4.7, 0.3)

The Convex-Hull-TSP Algorithm distance is 19.4561

**(六) 執行結果、討論與心得**

1. 執行結果

To test the (3) and (4) execution time, so I change the main function respectively as follows.

(3)

int main(void) {

vector<vector<double>> location;

double start, end;

get\_POIs\_location(location);

start = clock();

shortest\_Path(location);

end = clock();

cout << "The program run " << (end - start) << "ms";

return 0;

}

(4)

int main(void) {

vector<vector<double>> location;

double start, end;

get\_POIs\_location(location);

vector<int> convex\_hull = max\_convex\_hull(location);

start = clock();

Convex\_Hull\_TSP\_Algorithm(location, convex\_hull);

end = clock();

cout << "\nThe program run " << (end - start) << "ms";

return 0;

}

* 1. **Closet(POIs)**

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

1. **Max\_ConvexHull(POIs)**

一張含有 文字, 軟體, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

1. **Shortest\_Path(POIs)**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

1. **Convex\_Hull\_TSP\_Algorithm(POIs, ConvexHull)**

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. 討論

Discussing for Brute-Force Algorithm + Exhaustive search Algorithm and Convex-Hull-TSP-Algorithm for traveling all n POIs to get the shortest path.

1. **Running Time**

由於Convex\_Hull\_TSP\_Algorithm有用到Max\_ConvexHull函式來求出Convex-hull，所以將這個過程視為「參數輸入」，忽略不計。同時，也不計算Map的輸入時間，僅計算主要函式的運算時間。Shortest\_Path約為3ms，而Convex\_Hull\_TSP\_Algorithm則是4ms，該結果不太符合我所計算出來的時間複雜度。

1. **Distance Comparison**

根據結果，Shortest\_Path約為20.0363單位長，而Convex\_Hull\_TSP\_Algorithm則是19.4561單位長，所以 1.0298，前者多後者約3%，沒有超出合理的誤差範圍，因此Convex\_Hull\_TSP\_Algorithm在此情況中，算是一個好的演算法。

1. **Error of reflection points**

在Convex\_Hull\_TSP\_Algorithm實作中，我在對每個點算每個convex-hull的映射點時，發現它們幾乎無法和原本的點連線和直線法向量垂直，並且有些點甚至是跑到不可理喻的地方，形成嚴重的偏差，不過有鑑於在我仔細檢查公式確認無誤後，結果是，它們仍然在直線上(至少在desmos看起來是)，所以我也只能照著這個結果來作映射，以下是Map1中，(1.5, 2), (1.5, 3.5), (5.3, 4.6), (3.44, 1)依照順序的desmos各convex-hull直線的映射點圖。

組成Convex-hull的點示意圖

一張含有 文字, 螢幕擷取畫面, 繪圖, 數字 的圖片

自動產生的描述

Convex-hull示意圖

一張含有 文字, 行, 螢幕擷取畫面, 繪圖 的圖片

自動產生的描述

(1.5, 2)與6個相對應的映射點圖(可映射於1、2、4、6，最終選擇1)

一張含有 文字, 行, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述

(1.5, 3.5)與6個相對應的映射點圖(可映射於1、2、4，最終選擇1)

一張含有 行, 圖表, 文字, 繪圖 的圖片

自動產生的描述

(5.3, 4.6)與6個相對應的映射點圖(沒有可映射點，最終選擇6)

一張含有 文字, 螢幕擷取畫面, 行, 繪圖 的圖片

自動產生的描述

(3.44, 1)與6個相對應的映射點圖(可映射於1、2、4、6，最終選擇2)

一張含有 文字, 行, 螢幕擷取畫面, 繪圖 的圖片

自動產生的描述

一張含有 文字, 行, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述

3. 心得

本實作確實提供我一個很好的圖形式題目練習，不過我對於所建議的資料結構有所疑慮，也就是adjacency list和adjacency matrix，因為在本題來說，POIs並不會有所關連而有連線，我無從得知每個POIs有什麼關係，所以我判定兩者是不適用於本題，而直接以一個n by 2的vector來儲存每個點的經緯度，並將二維座標軸用於表示每個點的位置，其中以x = 0是右為東、左為西的經度分界；以y = 0下為南、上為北的緯度分界，可惜礙於時間不足，只有做一張地圖。另外，我的解法可能並不是真正切合題目，因為Convex-Hull-TSP Algorithm似乎應該注意順序方向，而我的輸出則是無向，包括convex-hull也並非順時針或逆時針輸出，而是跟POIs輸入的順序有關。我更了解到Brust-Force Algorithm並不是一種令人避之唯恐不及的做法，正好相反，它是我們對於題目第一種可能想到的直觀解法，並且在某些條件下更是optimal的演算法。我在本題最大的收穫，莫過於重新溫習如何求線外一點投影到直線，以及它們之間的距離公式，假設一直線為ax + by + c = 0，則線外一點(, )到該直線的投影點為( + ta, + tb)，其中t需要將投影點代入直線方程式中求得；還有用來算幾何多邊形的面積Shoelace formula。

**參考文獻**

1. <https://www.geeksforgeeks.org/2d-vector-in-cpp-with-user-defined-size/>
2. <https://stackoverflow.com/questions/936687/how-do-i-declare-a-2d-array-in-c-using-new>
3. <https://en.wikipedia.org/wiki/Shoelace_formula>
4. [https://www.youtube.com/watch?v=ooqGnzfqAE0&ab\_channel=均一教育平台JunyiAcademy](https://www.youtube.com/watch?v=ooqGnzfqAE0&ab_channel=%E5%9D%87%E4%B8%80%E6%95%99%E8%82%B2%E5%B9%B3%E5%8F%B0JunyiAcademy)
5. <https://shengyu7697.github.io/std-vector/>
6. [如何在Windows或Mac上使用命令提示符运行exe文件 (wikihow.com)](https://zh.wikihow.com/%E5%9C%A8Windows%E6%88%96Mac%E4%B8%8A%E4%BD%BF%E7%94%A8%E5%91%BD%E4%BB%A4%E6%8F%90%E7%A4%BA%E7%AC%A6%E8%BF%90%E8%A1%8Cexe%E6%96%87%E4%BB%B6)