410921208
楊右宇

Question1

(1)

ACO :

蝴蟻找食物時，起初都是沒有目標的，他從螞蟻洞中走出，隨機的爬向各個方向，在這期間他會向外界播撒一種化學物質，也就是費洛蒙，所以這裡就可以得到的一個前提，越多螞蟻走過的路徑，費洛蒙濃度就會越高，那麼某條路徑資訊素濃度高了，自然就會有越多的螞蟻感覺到了，就會聚集過來了。

用演算法去模擬螞蟻的上述行為時，由於我們希望走得路徑長越短越好，所以結果越好的路徑，費洛蒙就應該要越多，往後將會有更高的概率被螞蟻所選擇到。以此答案就會往好處收斂，而選擇時，費洛蒙只會使被選中的機率增加，還是有可能選其他解，以此來跳脫局部最佳解。

DP :

首先定義 DP 陣列的兩個狀態，一個狀態為目前走過的所有點，我們使用狀態壓縮的方式來表示一個 set，第二個狀態為目前走到哪個點。

在求解的過程中，假定我們要球的狀態為走過集合 SET(ACDE)，且最後走到 C 點，這條旅程的最短路徑必經過點 ADE，再到 C 點，此處符合最佳化原則。

更新的方程式為 dp[set(k)][A] = min(dp[set(k)-A][B] + dis[B][A])，其中 A 為任意點，set(k)必包含點 A，B 屬於集合 set(k)-A 中的任一點。

B & B :

在 Branch and Bound 方法中，對於樹中的當前節點，我們計算如果我們關閉該節點，我們可以獲得的最佳可能解決方案的界限。如果最佳可能解決方案本身的邊界比當前最佳（迄今為止計算的最佳）更差，那麼我們忽略以該節點為根的子樹。

再狀態樹的設計，因為路徑為環，所以從哪出發其時沒差，我們任取一個點出發，往下的每一層中，走過的點會越來越多。由於某些點已經走過，我們在計算 bound value 時能選的數字就會減少，在能選的數字中全選最小者，就是此狀態的 bound value，以 priority queue 紀錄未計算的節點，直到清空 queue。

(2)

ACO :

(a) 初始化信息素(費洛蒙)，以及启发式信息(跟兩城市間的距離成反比)，

並生成螞蟻。

(b) 為每隻螞蟻選出一條道路，其到下一個點的機率受到此點的初始化信息素以及启发式信息的影響，直到選過所有點，成為一路徑。
(c) 計算每隻螞蟻此次迭代的路線距離，紀錄最短的。
(d) 更新信息素，針對每隻螞蟻，其再自己走的路徑留下的信息素根據走過的距離增加而減少；再來是計算信息素的蒸發。
(e) 清空螞蟻的記憶，將所在點都設為沒去過。
(f) 重複執行(b)到(e)直到算法執行到預先指定的最大迭代次數。

DP :

(a) 求出距離矩陣 dis[][]。
(b) 初始化 dp[][]，將所有只有一個一個點的集合，以及到該點的狀態設為零，其他狀態設為極大值。
(c) 枚舉每個狀態(SET)以及每個點，判斷此點是否存在 SET 中，如果沒有就直接往下一個點或狀態；如果為合理狀態，以公式求此狀態的解 dp[set(k)][A] = min(dp[set(k)-A][B] + dis[B][A])，其中 A 為任意點，set(k) 必包含點 A，B 屬於集合 set(k)-A 中的任一點。同時記錄 DP 表中其上一個狀態，以便最後印出完整路徑以及找出 Head Point。
(d) 求最後解，狀態(SET)為全都有，此處會有以不同點為結尾，找出最小者，比較時公式為 ans=min(ans,dp[SET(all)][P]+dis[P][H])，其中 P 為測資中的任一點，H 為此狀態的開頭，因為是環，要走回出發點。

B & B :

(a) 求出距離矩陣 dis[][]。
(b) 將跟節點設為經過第零個點的 list，計算 bound value 並將此點加入 priority queue。Bound value 的計算規則為如果此點已確定，則查表選擇該路徑，list 最後一個點的選擇範圍不包含所有在 list 中的點，其餘不在 list 中的點選擇範圍為所有不包含在 list 中的點，但可以選 list 的開頭。
(c) 如果 queue 不為空，則將 queue 中最上層元素拿出；如果為空就結束。
(d) 如果此元素之 bound value 大於目前最小值，結束程式，因為後面的 bound value 必大於當前元素。
(e) 如果此元素為一解，更新目前最佳解。
(f) 計算此元素的下層元素，計算其 bound value，比較 best valule 決定是否加入 queue 中。
(g) 回到 (c) ，直到 queue 中為空。

(3)

ACO :

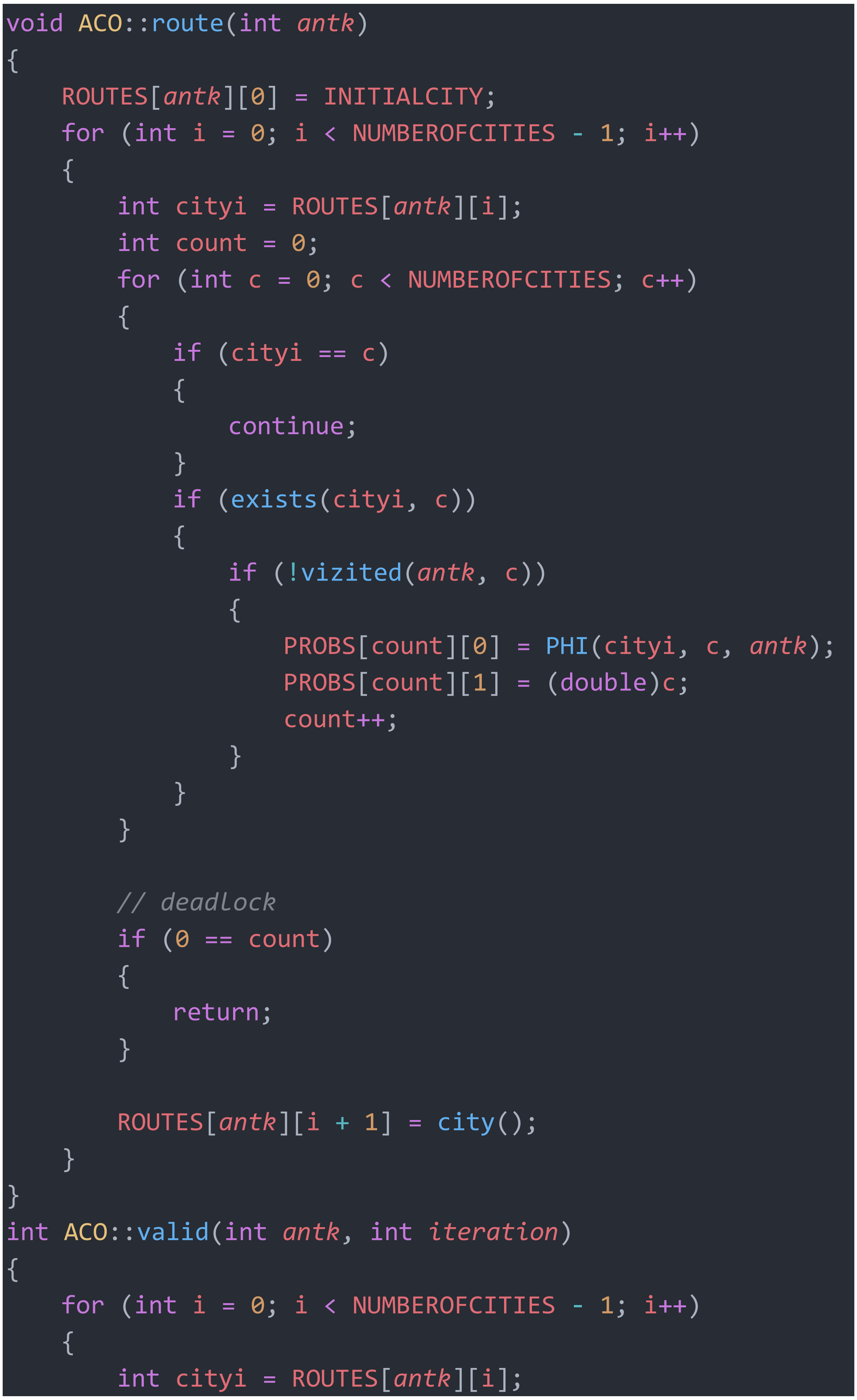
```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <limits>
#include <climits>
#include <ctime>
#include <vector>

using namespace std;

class Randoms
{

private:
    long xpto;

public:
    Randoms(long x) { xpto = -x; }

    double Normal(double avg, double sigma)
    {
        return (avg + sigma * gaussdev(&xpto));
    }

    double Uniforme()
    {
        return ran1(&xpto);
    }

    double sorte(int m)
    {
        return (1.0 * rand()) / (1.0 * RAND_MAX) * 2.0 * m
- m;
    }
```

```
#define IA 16807
#define IM 2147483647
#define AM (1.0 / IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1 + (IM - 1) / NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

    float ran1(long *idum)
    {
        int j;
        long k;
        static long iy = 0;
        static long iv[NTAB];
        float temp;
        if (*idum <= 0 || !iy)
        {
            if (-(*idum) < 1)
                *idum = 1;
            else
                *idum = -(*idum);
            for (j = NTAB + 7; j >= 0; j--)
            {
                k = (*idum) / IQ;
                *idum = IA * (*idum - k * IQ) - IR * k;
                if (*idum < 0)
                    *idum += IM;
                if (j < NTAB)
                    iv[j] = *idum;
            }
            iy = iv[0];
        }
        k = (*idum) / IQ;
        *idum = IA * (*idum - k * IQ) - IR * k;
        if (*idum < 0)
            *idum += IM;
```

```c
    j = iy / NDIV;
    iy = iv[j];
    iv[j] = *idum;
    if ((temp = AM * iy) > RNMX)
        return RNMX;
    else
        return temp;
}

float gaussdev(long *idum)
{
    //    float ran1(long *idum);

    static int iset = 0;
    static float gset;
    float fac, rsq, v1, v2;
    if (*idum < 0)
        iset = 0;
    if (iset == 0)
    {
        do
        {
            v1 = 2.0 * ran1(idum) - 1.0;
            v2 = 2.0 * ran1(idum) - 1.0;
            rsq = v1 * v1 + v2 * v2;

        } while (rsq >= 1.0 || rsq == 0.0);
        fac = sqrt(-2.0 * log(rsq) / rsq);

        gset = v1 * fac;
        iset = 1;
        return v2 * fac;
    }
    else
    {
        iset = 0;
        return gset;
    }
    }
};

class ACO
{
public:
    ACO(int nAnts, int nCities,
        double alpha, double beta, double q, double ro,
double taumax,
        int initCity);
    virtual ~ACO();

    void init();

    void connectCITIES(int cityi, int cityj);
    void setCITYPOSITION(int city, double x, double y);

    void printPHEROMONES();
    void printGRAPH();
    void printRESULTS();

    void optimize(int ITERATIONS);

private:
    double distance(int cityi, int cityj);
    bool exists(int cityi, int cityc);
    bool vizited(int antk, int c);
    double PHI(int cityi, int cityj, int antk);

    double length(int antk);

    int city();
    void route(int antk);
    int valid(int antk, int iteration);

    void updatePHEROMONES();

    int NUMBEROFANTS, NUMBEROFCITIES, INITIALCITY;
    double ALPHA, BETA, Q, RO, TAUMAX;

    double BESTLENGTH;
    int *BESTROUTE;

    int **GRAPH, **ROUTES;
    double **CITIES, **PHEROMONES, **DELTAPHEROMONES,
**PROBS;

    Randoms *randoms;
};

ACO::ACO(int nAnts, int nCities,
         double alpha, double beta, double q, double ro,
double taumax,
         int initCity)
{
    NUMBEROFANTS = nAnts;
    NUMBEROFCITIES = nCities;
    ALPHA = alpha;
    BETA = beta;
    Q = q;
    RO = ro;
    TAUMAX = taumax;
    INITIALCITY = initCity;

    randoms = new Randoms(21);
}
ACO::~ACO()
{
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        delete[] GRAPH[i];
        delete[] CITIES[i];
        delete[] PHEROMONES[i];
        delete[] DELTAPHEROMONES[i];
        if (i < NUMBEROFCITIES - 1)
        {
            delete[] PROBS[i];
        }
    }
    delete[] GRAPH;
    delete[] CITIES;
    delete[] PHEROMONES;
    delete[] DELTAPHEROMONES;
    delete[] PROBS;
}

void ACO::init()
{
    GRAPH = new int *[NUMBEROFCITIES];
    CITIES = new double *[NUMBEROFCITIES];
    PHEROMONES = new double *[NUMBEROFCITIES];
    DELTAPHEROMONES = new double *[NUMBEROFCITIES];
    PROBS = new double *[NUMBEROFCITIES - 1];
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        GRAPH[i] = new int[NUMBEROFCITIES];
        CITIES[i] = new double[2];
        PHEROMONES[i] = new double[NUMBEROFCITIES];
        DELTAPHEROMONES[i] = new double[NUMBEROFCITIES];
        PROBS[i] = new double[2];
        for (int j = 0; j < 2; j++)
        {
            CITIES[i][j] = -1.0;
            PROBS[i][j] = -1.0;
        }
        for (int j = 0; j < NUMBEROFCITIES; j++)
        {
            GRAPH[i][j] = 0;
            PHEROMONES[i][j] = 0.0;
            DELTAPHEROMONES[i][j] = 0.0;
        }
    }

    ROUTES = new int *[NUMBEROFANTS];
    for (int i = 0; i < NUMBEROFANTS; i++)
    {
        ROUTES[i] = new int[NUMBEROFCITIES];
        for (int j = 0; j < NUMBEROFCITIES; j++)
        {
            ROUTES[i][j] = -1;
        }
    }

    BESTLENGTH = (double)INT_MAX;
    BESTROUTE = new int[NUMBEROFCITIES];
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        BESTROUTE[i] = -1;
    }
}

void ACO::connectCITIES(int cityi, int cityj)
{
    GRAPH[cityi][cityj] = 1;
    PHEROMONES[cityi][cityj] = randoms->Uniforme() *
TAUMAX;
    GRAPH[cityj][cityi] = 1;
    PHEROMONES[cityj][cityi] = PHEROMONES[cityi][cityj];
}
void ACO::setCITYPOSITION(int city, double x, double y)
{
    CITIES[city][0] = x;
    CITIES[city][1] = y;
}
void ACO::printPHEROMONES()
{
    cout << " PHEROMONES: " << endl;
    cout << "   | ";
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        printf("%5d   ", i);
    }
```

```cpp
    cout << endl
         << "- | ";
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        cout << "--------";
    }
    cout << endl;
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        cout << i << " | ";
        for (int j = 0; j < NUMBEROFCITIES; j++)
        {
            if (i == j)
            {
                printf("%5s   ", "x");
                continue;
            }
            if (exists(i, j))
            {
                printf("%7.3f ", PHEROMONES[i][j]);
            }
            else
            {
                if (PHEROMONES[i][j] == 0.0)
                {
                    printf("%5.0f   ", PHEROMONES[i][j]);
                }
                else
                {
                    printf("%7.3f ", PHEROMONES[i][j]);
                }
            }
        }
        cout << endl;
    }
    cout << endl;
}
```

```
double ACO::distance(int cityi, int cityj)
{
    return (double)
        sqrt(pow(CITIES[cityi][0] - CITIES[cityj][0], 2) +
             pow(CITIES[cityi][1] - CITIES[cityj][1], 2));
}

bool ACO::exists(int cityi, int cityc)
{
    return (GRAPH[cityi][cityc] == 1);
}

bool ACO::vizited(int antk, int c)
{
    for (int l = 0; l < NUMBEROFCITIES; l++)
    {
        if (ROUTES[antk][l] == -1)
        {
            break;
        }
        if (ROUTES[antk][l] == c)
        {
            return true;
        }
    }
    return false;
}
double ACO::PHI(int cityi, int cityj, int antk)
{
    double ETAij = (double)pow(1 / distance(cityi, cityj),
BETA);
    double TAUij = (double)pow(PHEROMONES[cityi][cityj],
ALPHA);

    double sum = 0.0;
    for (int c = 0; c < NUMBEROFCITIES; c++)
    {
        if (exists(cityi, c))
        {
            if (!vizited(antk, c))
            {
                double ETA = (double)pow(1 /
distance(cityi, c), BETA);
                double TAU =
(double)pow(PHEROMONES[cityi][c], ALPHA);
                sum += ETA * TAU;
            }
        }
    }
    return (ETAij * TAUij) / sum;
}

double ACO::length(int antk)
{
    double sum = 0.0;
    for (int j = 0; j < NUMBEROFCITIES - 1; j++)
    {
        sum += distance(ROUTES[antk][j], ROUTES[antk][j +
1]);
    }
    return sum;
}

int ACO::city()
{
    double xi = randoms->Uniforme();
    int i = 0;
    double sum = PROBS[i][0];
    while (sum < xi)
    {
        i++;
        sum += PROBS[i][0];
    }
    return (int)PROBS[i][1];
}
```

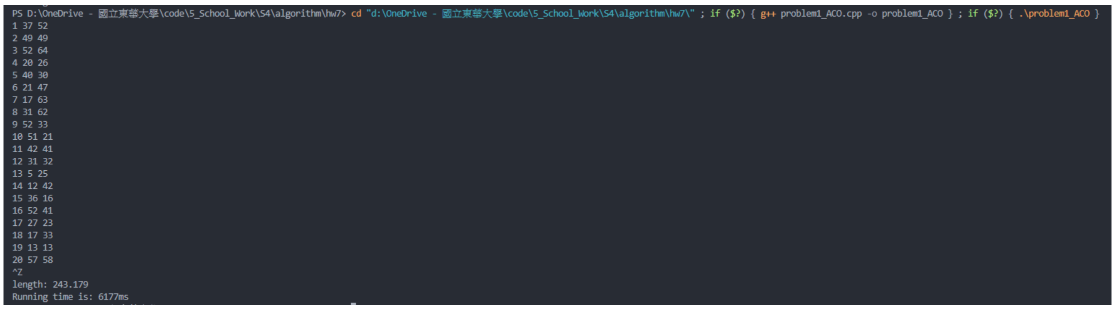


```cpp
void ACO::route(int antk)
{
    ROUTES[antk][0] = INITIALCITY;
    for (int i = 0; i < NUMBEROFCITIES - 1; i++)
    {
        int cityi = ROUTES[antk][i];
        int count = 0;
        for (int c = 0; c < NUMBEROFCITIES; c++)
        {
            if (cityi == c)
            {
                continue;
            }
            if (exists(cityi, c))
            {
                if (!vizited(antk, c))
                {
                    PROBS[count][0] = PHI(cityi, c, antk);
                    PROBS[count][1] = (double)c;
                    count++;
                }
            }
        }

        // deadlock
        if (0 == count)
        {
            return;
        }

        ROUTES[antk][i + 1] = city();
    }
}
int ACO::valid(int antk, int iteration)
{
    for (int i = 0; i < NUMBEROFCITIES - 1; i++)
    {
        int cityi = ROUTES[antk][i];
        int cityj = ROUTES[antk][i + 1];
        if (cityi < 0 || cityj < 0)
        {
            return -1;
        }
        if (!exists(cityi, cityj))
        {
            return -2;
        }
        for (int j = 0; j < i - 1; j++)
        {
            if (ROUTES[antk][i] == ROUTES[antk][j])
            {
                return -3;
            }
        }
    }

    if (!exists(INITIALCITY, ROUTES[antk][NUMBEROFCITIES -
1]))
    {
        return -4;
    }

    return 0;
}

void ACO::printGRAPH()
{
    cout << " GRAPH: " << endl;
    cout << "  | ";
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        cout << i << " ";
    }
    cout << endl
         << "- | ";
    for (int i = 0; i < NUMBEROFCITIES; i++)
```

```cpp
    {
        cout << "- ";
    }
    cout << endl;
    int count = 0;
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        cout << i << " | ";
        for (int j = 0; j < NUMBEROFCITIES; j++)
        {
            if (i == j)
            {
                cout << "x ";
            }
            else
            {
                cout << GRAPH[i][j] << " ";
            }
            if (GRAPH[i][j] == 1)
            {
                count++;
            }
        }
        cout << endl;
    }
    cout << endl;
    cout << "Number of connections: " << count << endl
         << endl;
}
void ACO::printRESULTS()
{
    BESTLENGTH += distance(BESTROUTE[NUMBEROFCITIES - 1],
INITIALCITY);
    // cout << "BEST ROUTE:" << endl;
    // for (int i = 0; i < NUMBEROFCITIES; i++)
    // {
    //     cout << BESTROUTE[i] << " ";
    // }
    // cout << endl;
    cout << "length: " << BESTLENGTH << endl;

    // cout << endl
    //      << " IDEAL ROUTE:" << endl;
    // cout << "0 7 6 2 4 5 1 3" << endl;
    // cout << "length: 127.509" << endl;
}

void ACO::updatePHEROMONES()
{
    for (int k = 0; k < NUMBEROFANTS; k++)
    {
        double rlength = length(k);
        for (int r = 0; r < NUMBEROFCITIES - 1; r++)
        {
            int cityi = ROUTES[k][r];
            int cityj = ROUTES[k][r + 1];
            DELTAPHEROMONES[cityi][cityj] += Q / rlength;
            DELTAPHEROMONES[cityj][cityi] += Q / rlength;
        }
    }
    for (int i = 0; i < NUMBEROFCITIES; i++)
    {
        for (int j = 0; j < NUMBEROFCITIES; j++)
        {
            PHEROMONES[i][j] = (1 - RO) * PHEROMONES[i][j]
+ DELTAPHEROMONES[i][j];
            DELTAPHEROMONES[i][j] = 0.0;
        }
    }
}

void ACO::optimize(int ITERATIONS)
{
    for (int iterations = 1; iterations <= ITERATIONS;
iterations++)
    {
        // cout << flush;
        // cout << "ITERATION " << iterations << " HAS
STARTED!" << endl
        //      << endl;

        for (int k = 0; k < NUMBEROFANTS; k++)
        {
            // cout << " : ant " << k << " has been
released!" << endl;
            while (0 != valid(k, iterations))
            {
                // cout << "  :: releasing ant " << k << "
again!" << endl;
                for (int i = 0; i < NUMBEROFCITIES; i++)
                {
                    ROUTES[k][i] = -1;
                }
                route(k);
            }

            // for (int i = 0; i < NUMBEROFCITIES; i++)
            // {
            //     cout << ROUTES[k][i] << " ";
            // }
            // cout << endl;

            // cout << "  :: route done" << endl;
            double rlength = length(k);

            if (rlength < BESTLENGTH)
            {
                BESTLENGTH = rlength;
                for (int i = 0; i < NUMBEROFCITIES; i++)
                {
                    BESTROUTE[i] = ROUTES[k][i];
                }
            }
            // cout << " : ant " << k << " has ended!" <<
endl;
        }

        // cout << endl << "updating PHEROMONES . . .";
        updatePHEROMONES();
        // cout << " done!" << endl
        //      << endl;
        // printPHEROMONES();

        for (int i = 0; i < NUMBEROFANTS; i++)
        {
            for (int j = 0; j < NUMBEROFCITIES; j++)
            {
                ROUTES[i][j] = -1;
            }
        }

        // cout << endl
        //      << "ITERATION " << iterations << " HAS
ENDED!" << endl
        //      << endl;
    }
}

#define ITERATIONS (int)200
#define NUMBEROFANTS (int)45
#define NUMBEROFCITIES (int)20

// if (ALPHA == 0) { stochastic search & sub-optimal
route }
#define ALPHA (double)0.4
// if (BETA  == 0) { sub-optimal route }
#define BETA (double)2.5
// Estimation of the suspected best route.
#define Q (double)80
// Pheromones evaporation.
#define RO (double)0.2
```

```cpp
// Maximum pheromone random number.
#define TAUMAX (int)1

#define INITIALCITY (int)0

int main()
{
    vector<vector<double>> data;
    double a, b, c;
    while (cin >> a >> b >> c)
    {
        vector<double> t;
        t.push_back(a);
        t.push_back(b);
        t.push_back(c);
        data.push_back(t);
    }
    // cout << data.size() << endl;
    // CITY_NUM = data.size();

    ACO *ANTS = new ACO(NUMBEROFANTS, data.size(),
                        ALPHA, BETA, Q, RO, TAUMAX,
                        INITIALCITY);

    ANTS->init();

    for (int i = 0; i < data.size(); ++i)
    {
        for (int j = i + 1; j < data.size(); ++j)
        {
            ANTS->connectCITIES(i, j);
        }
    }

    for (int i = 0; i < data.size(); ++i)
    {
        ANTS->setCITYPOSITION(i, data[i][1], data[i][2]);
    }
    clock_t start_time = clock();
    // ANTS->printGRAPH();

    // ANTS->printPHEROMONES();

    ANTS->optimize(ITERATIONS);

    ANTS->printRESULTS();

    clock_t end_time = clock();
    cout << "Running time is: " <<
static_cast<double>(end_time - start_time) /
CLOCKS_PER_SEC * 1000 << "ms" << endl;
    return 0;
}
```

# ACO 結果



```
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem1_ACO.cpp -o problem1_ACO } ; if ($?) { .\problem1_ACO }
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
8 31 62
9 52 33
10 51 21
11 42 41
12 31 32
13 5 25
14 12 42
15 36 16
16 52 41
17 27 23
18 17 33
19 13 13
20 57 58
^Z
length: 243.179
Running time is: 6177ms
```

DP :

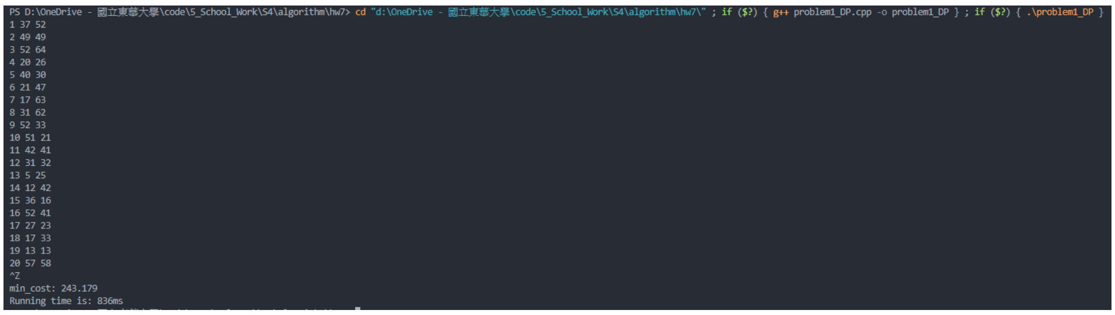
```cpp
#include <bits/stdc++.h>
// #include <vecotr>
using namespace std;
const double INF = 100000.0;
const int MAX_N = 22;
// const int start = 1; // vertax 1 為起點
// int N = 20;
// int m = 1 << N;
// double dp[1048576][MAX_N];
// int parents[1048576][MAX_N];

struct point
```

```
{
    double x;
    double y;
};

class Solution
{
    int n;
    int m;
    int start = 1;

    point city[MAX_N];
    double adj[MAX_N][MAX_N];
    double **dp;
    int **parents;

public:
    Solution(vector<vector<double>> &data)
    {
        n = data.size();
        m = 1 << data.size();

        dp = new double *[m];
        parents = new int *[m];
        for (int i = 0; i < m; ++i)
        {
            dp[i] = new double[n];
            parents[i] = new int[n];
        }
        for (int i = 0; i < n; i++)
        {
            city[i + 1].x = data[i][1];
            city[i + 1].y = data[i][2];
        }

        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
```

```cpp
            {
                if (i == j)
                    adj[i][j] = INF;
                else
                {
                    adj[i][j] = abs(city_dis(i, j));
                }
            }
        }
    }
    ~Solution()
    {
        for (int i = 0; i < m; i++)
        {
            delete[] parents[i];
            delete[] dp[i];
        }
        delete[] parents;
        delete[] dp;
    }

    double city_dis(int a, int b)
    {
        double temp_x, temp_y;
        temp_x = pow(city[a].x - city[b].x, 2);
        temp_y = pow(city[a].y - city[b].y, 2);
        return sqrt(temp_x + temp_y);
    }
    double min_col(int i)
    {
        double temp = 0x7f * 1.0;
        for (int j = 1; j <= n; j++)
            if (i != j)
                temp = min(temp, adj[i][j]);

        return temp;
    }
    void print()
    {
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
                printf("%11.2f", adj[i][j]);
            cout << endl;
        }
    }
    void solve()
    {

        for (int i = 0; i < m; i++) ///初始化
            for (int j = 0; j < n; j++)
            {
                dp[i][j] = INT_MAX / 2;
                parents[i][j] = -1;
            }
        for (int i = 0; i < n; i++)
            dp[1 << i][i] = 0.0; //代表以哪個起點開始走，並設
為0

        for (int Set = 0; Set < m; Set++) /// row（Set 代表
我現在已走過哪些city)
        {

            for (int last = 0; last < n; last++) // col。
（last 代表在所有走過的city 中，最後一個city，所以last 必定是
Set 裡其中一位)
            {
                if (last == start - 1 || ((Set & (1 <<
(start - 1))) == 0) || ((Set & (1 << last)) == 0))
                    continue; //判斷此last 是否在Set 中，方
法：將1 左移到那個城市的位元並與set 做AND,0 代表不存在

                int set_prev = Set - (1 << last); //假設
Set={BCD},且B 為最後一個，那可能為：CDB、DCB，所以要先求
Set{CD}的時候，用DP 表求min
```

```
                if (set_prev == 0) //假设今天只有一个城
市:001、010、100， 所以当我减掉最后一个城市时，會變成000，因為
他前面沒有城市，所以不需要判斷他前面是誰，只要回傳自己的字串的
長度
                {
                    continue;
                }
                for (int last_prev = 0; last_prev < n;
last_prev++) // 暴力搜（這裡就是找，B 前面的是 C 還是 D）
                {
                    if ((set_prev & (1 << last_prev)) == 0)
                        continue;


// ast_prev 必屬於 set_prev
                    if ((dp[set_prev][last_prev] != INT_MAX
/ 2 && adj[last_prev + 1][last + 1] != INF) &&
dp[set_prev][last_prev] + adj[last_prev + 1][last + 1] <=
dp[Set][last]) // Set{CD}中，以 C 結尾再加 C->B 的結尾與以 D 結
尾再加 D->B 求 min
                    {
                        dp[Set][last] =
dp[set_prev][last_prev] + adj[last_prev + 1][last + 1];
                        parents[Set][last] = last_prev;
                    }
                }
            }
        }

        double ret = INT_MAX * 1.0;
        int tail = 0;
        for (int last = 0; last < n; last++) /// search 最
後一個(dp[m-1][last]不一定是最短的要搞清楚！！！！！)
        {
            if (dp[m - 1][last] < ret)
            {
                ret = dp[m - 1][last];
                tail = last;
            }
        }
        cout << "min_cost: " << ret + adj[tail + 1][start] << endl;

        /////////backtracking 找路徑
        vector<int> path;
        path.push_back(start); //要回到起點
        path.push_back(tail + 1);
        int Set = m - 1;
        int last = tail;
        while (parents[Set][last] != -1)
        {
            int last_prev = parents[Set][last];
            path.push_back(last_prev + 1);
            Set = Set - (1 << last); ///假設 ABCD 且 C 為最後一個，那我將 C 放入 stack，再跳到 ABD  ，以此類推直到 parents[Set][last] == -1
            last = last_prev;
        }
        reverse(path.begin(), path.end()); ///要反轉(因為是回推)
        // cout << "path: ";

        // for (unsigned int i = 0; i < path.size(); i++)
        // {
        //     cout << path[i] << " ";
        //     if (i != path.size() - 1)
        //         cout << "-> ";
        // }
    }
};

int main()
{
    vector<vector<double>> data;
    double a, b, c;
    while (cin >> a >> b >> c)
    {
        vector<double> t;
        t.push_back(a);
        t.push_back(b);
        t.push_back(c);
        data.push_back(t);
    }

    clock_t start_time = clock();

    Solution *sol = new Solution(data);
    sol->solve();

    clock_t end_time = clock();
    cout << "Running time is: " <<
static_cast<double>(end_time - start_time) /
CLOCKS_PER_SEC * 1000 << "ms" << endl;
}
```

# DP 結果

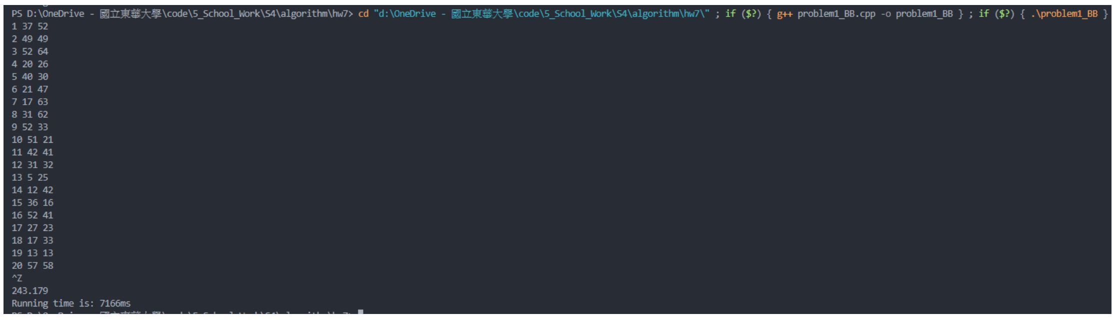


```
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem1_DP.cpp -o problem1_DP } ; if ($?) { .\problem1_DP }
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
8 31 62
9 52 33
10 51 21
11 42 41
12 31 32
13 5 25
14 12 42
15 36 16
16 52 41
17 27 23
18 17 33
19 13 13
20 57 58
^Z
min_cost: 243.179
Running time is: 836ms
```

B&B :

```
#include <bits/stdc++.h>
#define MAX 9999
using namespace std;
vector<double> minRow;
double minL = MAX; // for length
int CITIES;
class City
{
public:
    int id;
    int x;
    int y;
    City() {}
};

void Make_adj(vector<City> arr, vector<vector<double>>
&edges)
{
    for (int c = 0; c < CITIES; c++)
        for (int cc = c + 1; cc < CITIES; cc++)
            edges[c][cc] = edges[cc][c] = sqrt((arr[c].x -
arr[cc].x) * (arr[c].x - arr[cc].x) + (arr[c].y -
arr[cc].y) * (arr[c].y - arr[cc].y));
}

double TSP(vector<vector<double>> &edges, vector<bool>
visited, int current, int visit, double bound, double len)
{
    double minBound = MAX;
    if (visit == CITIES)
    {
        len = len + edges[0][current];
        if (len < minL)
            minL = len;
        return minL;
    }
    int minBound_id;
    double bounds[CITIES];
    double lens[CITIES];
    for (int i = 0; i < CITIES; i++) //建立 bound 表
    {
        if (i == current || visited[i]) //如果為已經過或是為
當前點則 bound=MAX，避免 branch
            bounds[i] = lens[i] = MAX;
        else
        {
            bounds[i] = bound - minRow[current] +
edges[current][i];
```

```
            lens[i] = len + edges[current][i];
            if (bounds[i] < minBound)
            {
                minBound = bounds[i]; //找到最小 bound
                minBound_id = i;
            }
        }
    }
    if (minBound > minL) //當前的都很爛的話則退回
        return minL;
    vector<bool> localVisited = visited;
    while (1)
    {
        bool is_Found_min = false;
        double Min = minL;
        if (minL == MAX)
            Min = minBound;
        if (minBound_id != current && bounds[minBound_id]
<= Min && !localVisited[minBound_id])
        {
            vector<bool> tempVisited = visited;
            tempVisited[minBound_id] = true;
            double new_minL = TSP(edges, tempVisited,
minBound_id, visit + 1, bounds[minBound_id],
lens[minBound_id]);
            if (new_minL <= minL)
                is_Found_min = true;
        }
        localVisited[minBound_id] = true;
        if (!is_Found_min) //沒有找到更好的
            break;
        bounds[minBound_id] = MAX;
        lens[minBound_id] = MAX;
        minBound = MAX;
        for (int i = 0; i < CITIES; ++i) //找其他更好的
            if (bounds[i] < minBound)
            {
                minBound = bounds[i];
                minBound_id = i;
            }
        if (minBound == MAX) //都不能走
            break;
    }
    return minL;
}

int main()
{
    vector<City> cities;
    City temp;
    while (cin >> temp.id)
    {
        cin >> temp.x;
        cin >> temp.y;
        cities.push_back(temp);
    }

    clock_t start_time = clock();

    CITIES = cities.size();
    vector<vector<double>> edges(CITIES,
vector<double>(CITIES));
    Make_adj(cities, edges);
    double minTotal = 0;
    for (int i = 0; i < CITIES; ++i) //為了確定界線，因此需
要找最小值
    {
        double mindis = MAX;
        for (int j = 0; j < CITIES; ++j)
        {
            if (j == i)
                continue;
            if (edges[i][j] < mindis)
                mindis = edges[i][j];
        }
        minRow.push_back(mindis);
        minTotal += mindis; //最小值總和
    }

    vector<bool> visited; //行進路程檢查
    for (int i = 0; i < CITIES; ++i)
    {
        visited.push_back(false);
    }
    visited[0] = true; //設定第一個點

    double distance = TSP(edges, visited, 0, 1, minTotal,
0);
    cout << distance << endl;

    clock_t end_time = clock();
    cout << "Running time is: " <<
static_cast<double>(end_time - start_time) /
CLOCKS_PER_SEC * 1000 << "ms" << endl;
}
```
# B&B 結果



```
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem1_BB.cpp -o problem1_BB } ; if ($?) { .\problem1_BB }
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
8 31 62
9 52 33
10 51 21
11 42 41
12 31 32
13 5 25
14 12 42
15 36 16
16 52 41
17 27 23
18 17 33
19 13 13
20 57 58
^Z
243.179
Running time is: 7166ms
```

(4)
ACO :

Time complexity :
```
迭代次數->O(l)*
{
蝸蟻數量->O(m)*
{
決定訪問城市 Build_Trip  中
{
  for (int i=0; i<N; i++)//將可以走的路初始->O(n) +
  while (tour_count < N)//決定下一個 City->O(n)
  {
    Next_City()中
    {
      for (int i=0; i<N; i++)//計算走到可行走的各點權重->O(n) +
      for (int i=0; i<N; i++) //計算到各點的機率->O(n)+
      for (int j=0; j<N; j++)//隨機移動->O(n)
    }
  }
} +
計算訪問總距離 UpdateTourLength->O(n))
  } +
  更新費洛蒙的 UpdatePheromones
  {
for (int i = 0; i<AntCount; i++)for (int j = 0; j<N-1; j++)//計算賀爾蒙變化->O(mn) +
for (int i = 0; i<N; i++)for (int j = 0; j<N; j++) //更新賀爾蒙->O(n^2)
  }
}
```
因此 O(l)* {O(m)*{{O(n)+O(n)*O(3n)}+ O(n)}+ O(mn)+ O(n^2)}
得 O(3m^2 nl+2ml+nml+n^2 l)-> O(m^2 nl+n^2 l)

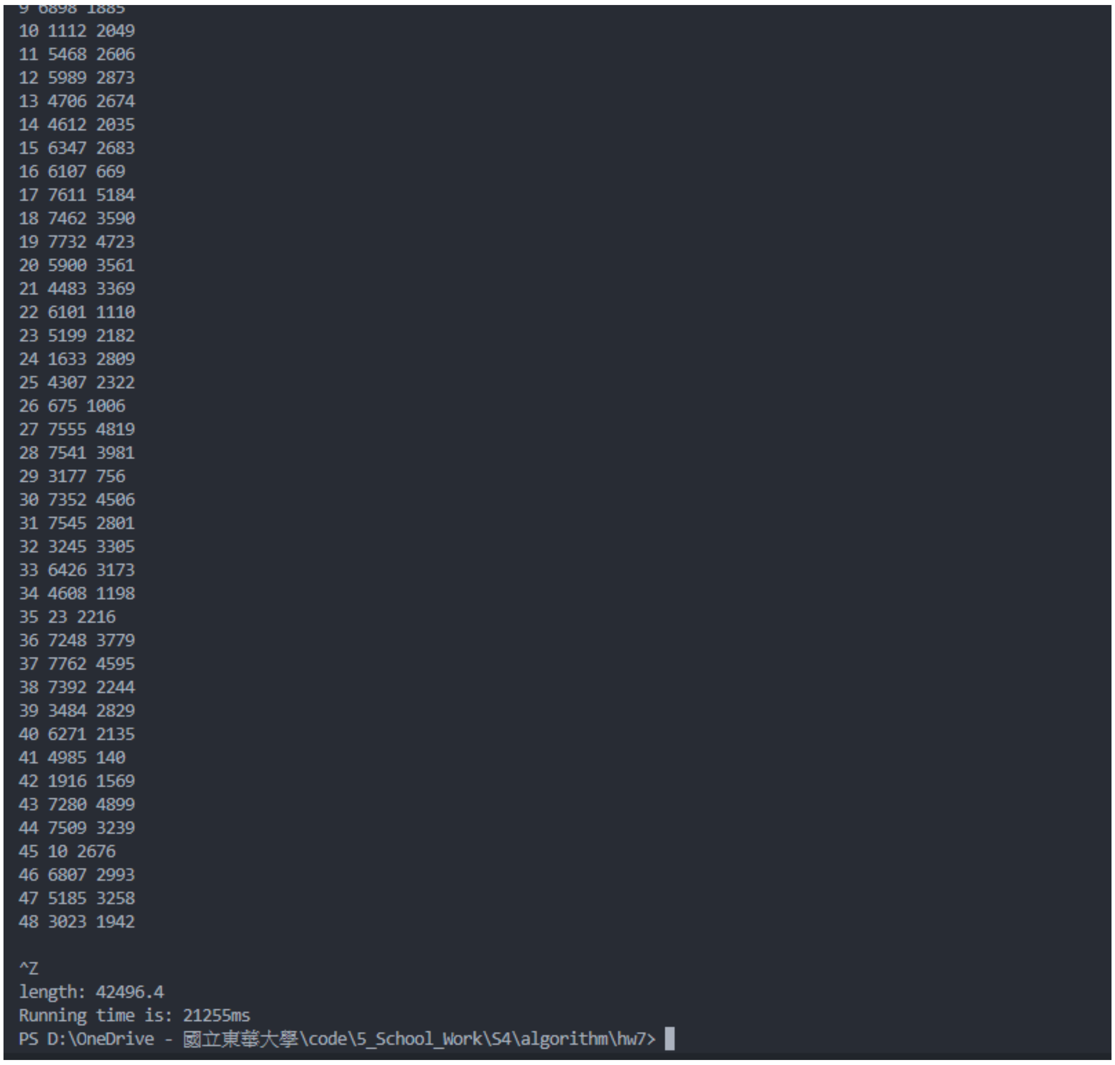
Space complexity:
用來儲存賀爾蒙的陣列 Tau[][]->O(n^2)
用來暫存儲存賀爾蒙變化的陣列 DeltaTau[][]->O(n^2)
用來儲存目前最段路經的陣列 temptour[N]->O(n)
用來儲存每個 City 的陣列 citylist[N]->O(n)
用來分別每隻蝸蟻的陣列 ant[AntCount]->O(m),每隻麻蟻都有記錄他的訪問路徑
陣列 tour[N + 1]、訪問檢查陣列 can_visit[N]、訪問機率陣列

select_probability[N]，因此總和為 O(3mn)
因此 Space Complexity->O(n^2+mn)

DP :
Time complexity :
　　每個集合 A 包含 k 個頂點，我們需要考慮 n－1－k 個頂點，針對每個頂點，基本運算必須被執行 k 次。包含 k 個頂點的 V－｛v｝子集合共有 C（n－1，k）個，所以基本運算被執行總次數為
$T(n) = \sum_{k=1}^{n-2} (n-1-k)k\binom{n-1}{k} \epsilon\theta(n^2 2^n)$ (every case)。
Space complexity :
　　在計算時，dp[][]、parents[][]，期兩個狀態的大小分別為狀態壓縮，為2^n，另一個狀態為目前早到的點，為 n，總複雜度為 O($n2^n$) (every case)。
B&B :
Time complexity :
　　每個節點在算 bound value 時的複雜度為 n^2，而我們樹往下長時每往下一層，能做的選擇會變少，因為每個節點只走一輪，為(n-1)!，總複雜度為 O(n!n^2) (worst case)。
　　如果我們直直往下找到一條路徑後，其他都被我們已 bound value 刪去了，複雜度為 O(n^3) (best case)。
Space complexity :
　　用於儲存每個 City 的左標位置 vector<City> cities->O(n)
用於存取鄰接長的 vector<vector<double>> edges(CITIES , vector<double> (CITIES))->O(n^2)
初始訪問記錄 vector<bool> visited->O(n)
用於存取每個 City 到各個 City 的最小距離 vector<double >minRow->O(n)
用於紀錄當前 City 到其他 City 的 bound　bounds[CITIES]->O(n^2)
用於紀錄當前 City 到其他 City 的長度　lens[CITIES]->O(n^2)
用於紀錄當前 City 是否經過其他 City　vector<bool> localVisited->O(n^2)
用於作為進入下一個 City 的訪問紀錄 vector<bool> tempVisited->O(n^2)
因此 Space Complexity->**O(n^2)**

(5)

從最後的運行時間可以看出 DP > ACO > B&B，ACO 會如此慢的原因是我刻意挑選參數，並多跑了幾次，所以可以算出最佳解，但我在測試途中有成功花一秒內得到兩百五十幾的成績，所以我認為啟發式的演算法跟一般演算法式不能比較的，因為啟發式演算法可以根據不同組測資，用不同參數的道不同結果，而在有限時間中，一般演算法得不出答案，而啟發式演算法可以給你一個接近的值。再來是如下圖資料，有 48 個點，以下是我用 20 隻螞蟻跑 100 次的結果。秒數雖然多了一些但有成功跑完，而 DP 的部分 2^48 次方，程式爆炸了，完全跑不出結果，啟發式在數量發常大的問題至少能給出一個較好的參考解。



```
9 6898 1885
10 1112 2049
11 5468 2606
12 5989 2873
13 4706 2674
14 4612 2035
15 6347 2683
16 6107 669
17 7611 5184
18 7462 3590
19 7732 4723
20 5900 3561
21 4483 3369
22 6101 1110
23 5199 2182
24 1633 2809
25 4307 2322
26 675 1006
27 7555 4819
28 7541 3981
29 3177 756
30 7352 4506
31 7545 2801
32 3245 3305
33 6426 3173
34 4608 1198
35 23 2216
36 7248 3779
37 7762 4595
38 7392 2244
39 3484 2829
40 6271 2135
41 4985 140
42 1916 1569
43 7280 4899
44 7509 3239
45 10 2676
46 6807 2993
47 5185 3258
48 3023 1942

^Z
length: 42496.4
Running time is: 21255ms
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7>
```

Question2
GA :
```
#include <iostream>
#include <vector>
#include <fstream>
#include <cmath>
#include <string>
#include <cstring>
#include <ctime>
#include <stdlib.h>
#include <unistd.h>            //哈希函数
#define CITY_N 40              //城市个数
#define POPSIZE 300            //种群个体数
#define MAXVALUE 0x7fffffff //路径最大值上限
#define N 100000               //需要根据实际求得的路径值修正
#define MAX_GEN 50000          //最大进化代数
#define CROSS 0.5              //交叉算子
#define MUT 0.05               //变异算子
using namespace std;

int CITY_NUM = 20;

unsigned seed = (unsigned)time(0); //每次产生不同的随机序列
double Hash[CITY_N + 1];

typedef struct CityPosition
{
    double x;
    double y;
} CityPosition; //城市位置

CityPosition CityPos[38] = {};

double CityDistance[CITY_N][CITY_N]; //城市距离

typedef struct
{
    int colony[POPSIZE][CITY_N + 1]; //城市种群,默认出发城市编号为0，则城市编号的最后一个城市还应该为0
    double fitness[POPSIZE];         // 每个个体的适应度，即1/Distance[POPSIZE]
    double Distance[POPSIZE];        //每个个体的总路径
    int BestRooting[CITY_N + 1];     //最优城市路径序列
    double BestFitness;              //最优路径适应值
    double BestValue;                //最优路径长度
    int BestNum;                     //最优路径城市数目
} TSP, *PTSP;

//计算城市距离CityDistance[i][j]
double CalculatDist(CityPosition CityPos[])
{
    for (int i = 0; i < CITY_NUM; i++)
    {
        for (int j = 0; j < CITY_NUM; j++)
        { //最后一个城市还应该返回到出发节点
            if (i != j)
                CityDistance[i][j] = sqrt(pow(CityPos[j].x - CityPos[i].x, 2) + pow(CityPos[j].y - CityPos[i].y, 2));
            else
                CityDistance[i][i] = 0;
        }
    }
}

//数组复制
void copy(int a[], int b[])
{
    for (int i = 0; i < CITY_NUM + 1; i++)
    {
        a[i] = b[i];
    }
}

//用来检查新生成的节点是否在当前群体中，0 号节点是默认出发节点和终止节点
```

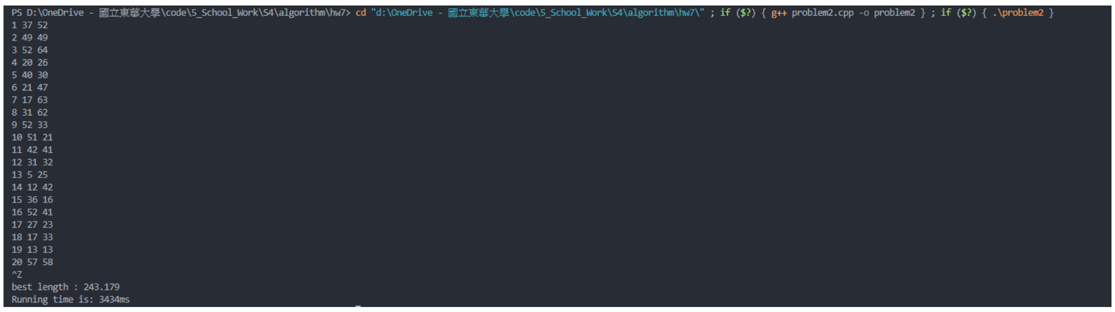
```
bool check(TSP &city, int pop, int num, int k)
{
    for (int i = 0; i <= num; i++)
    {
        if (k == city.colony[pop][i])
            return true; //新生成节点存在于已经生成的路径中
    }
    return false; //新生成节点没有存在于已经生成的路径中
}

//种群初始化，即为 city.colony[i][j]赋值
void InitColony(TSP &city)
{
    int r;
    for (int i = 0; i < POPSIZE; i++)
    {
        city.colony[i][0] = 0;
        city.colony[i][CITY_NUM] = 0;
        city.BestValue = MAXVALUE;
        city.BestFitness = 0; //适应值越大越好
    }
    for (int i = 0; i < POPSIZE; i++)
    {
        for (int j = 1; j < CITY_NUM; j++)
        {
            r = rand() % (CITY_NUM - 1) + 1; //产生 1～
CITY_NUM-1 之间的随机数
            while (check(city, i, j, r))     //随机产生城市
序号，即为 city.colony[i][j]赋值
            {
                r = rand() % (CITY_NUM - 1) + 1;
            }

            city.colony[i][j] = r;
        }
    }
}
```

```
//计算适应度,同时选出最优的
void CalFitness(TSP &city)
{
    int start, end;
    int Best = 0;
    for (int i = 0; i < POPSIZE; i++)
    { //求每个个体的总路径，适应度
        city.Distance[i] = 0;
        for (int j = 1; j <= CITY_NUM; j++)
        {
            start = city.colony[i][j - 1];
            end = city.colony[i][j];
            city.Distance[i] = city.Distance[i] +
CityDistance[start][end]; // city.Distance[i]每个个体的总路
径
        }
        city.fitness[i] = N / city.Distance[i];
        if (city.fitness[i] > city.fitness[Best]) //选出最
大的适应度，即选出所有个体中的最短路径
            Best = i;
    }
    copy(city.BestRooting, city.colony[Best]); //将最优个体
拷贝给city.BestRooting
    city.BestFitness = city.fitness[Best];
    city.BestValue = city.Distance[Best];
    city.BestNum = Best;
}

//适应度
double GetFittness(int a[])
{
    int i, start, end;
    double Distance = 0;
    for (i = 0; i < CITY_NUM; i++)
    {
        start = a[i];
        end = a[i + 1];
        Distance += CityDistance[start][end];
    }
    return N / Distance;
}

//选择算子：轮盘赌法
void Select(TSP &city)
{
    int TempColony[POPSIZE][CITY_NUM + 1];
    int i, j, t;
    double s;
    double GaiLv[POPSIZE];
    double SelectP[POPSIZE + 1];
    double avg;
    double sum = 0;
    for (i = 0; i < POPSIZE; i++)
        sum += city.fitness[i];
    for (i = 0; i < POPSIZE; i++)
        GaiLv[i] = city.fitness[i] / sum;
    SelectP[0] = 0;
    for (i = 0; i < POPSIZE; i++)
        SelectP[i + 1] = SelectP[i] + GaiLv[i] * RAND_MAX;
    memcpy(TempColony[0], city.colony[city.BestNum],
sizeof(TempColony[0]));
    for (t = 1; t < POPSIZE; t++)
    {
        double ran = rand() % RAND_MAX + 1;
        s = (double)ran / 100.0;
        for (i = 1; i < POPSIZE; i++)
            if (SelectP[i] >= s)
                break;
        memcpy(TempColony[t], city.colony[i - 1],
sizeof(TempColony[t]));
    }
    for (i = 0; i < POPSIZE; i++)
        memcpy(city.colony[i], TempColony[i],
sizeof(TempColony[i]));
}
```

```
//交叉：头尾不变，中间打乱顺序交叉
void Cross(TSP &city, double pc) //交叉概率是pc
{
    int i, j, t, l;
    int a, b, ca, cb;
    int Temp1[CITY_NUM + 1], Temp2[CITY_NUM + 1];
    for (i = 0; i < POPSIZE; i++)
    {
        double s = ((double)(rand() % RAND_MAX)) /
RAND_MAX;
        if (s < pc)
        {
            cb = rand() % POPSIZE;
            ca = cb;
            if (ca == city.BestNum || cb == city.BestNum)
//如果遇到最优则直接进行下次循环
                continue;

            l = rand() % (CITY_NUM / 2) + 1; // 1-一半的位
置
            a = rand() % (CITY_NUM - l) + 1; //全部

            memset(Hash, 0, sizeof(Hash)); //将s中当前位置
后面的n个字节 用ch替换并返回s 。
            Temp1[0] = Temp1[CITY_NUM] = 0;
            for (j = 1; j <= l; j++) //打乱顺序即随机，选出
来的通过Hash标记为1
            {
                Temp1[j] = city.colony[cb][a + j - 1]; //
a+l=2~48 25~38
                Hash[Temp1[j]] = 1;
            }
            for (t = 1; t < CITY_NUM; t++)
            {
                if (Hash[city.colony[ca][t]] == 0)
                {
                    Temp1[j++] = city.colony[ca][t];
                    Hash[city.colony[ca][t]] = 1;
                }
            }
            memcpy(city.colony[ca], Temp1, sizeof(Temp1));
        }
    }
}

//对换变异
void Mutation(TSP &city, double pm) //变异概率是 pm
{
    int i, m;
    int Temp[CITY_NUM + 1];
    for (int k = 0; k < POPSIZE; k++)
    {
        double s = ((double)(rand() % RAND_MAX)) /
RAND_MAX; //随机产生概率 0~1 间
        i = rand() %
POPSIZE;                                        //随机产生 0~POPSIZE
之间的数
        if (s < pm && i !=
city.BestNum)                       // i!=city.BestNum，即保
证最优的个体不变异
        {
            int a, b, t;
            a = (rand() % (CITY_NUM - 1)) + 1;
            b = (rand() % (CITY_NUM - 1)) + 1;
            copy(Temp, city.colony[i]);
            if (a > b) //保证让 b>=a
            {
                t = a;
                a = b;
                b = t;
            }
            for (m = a; m < (a + b) / 2; m++)
            {
                t = Temp[m];
                Temp[m] = Temp[a + b - m];
                Temp[a + b - m] = t;
            }

            if (GetFittness(Temp) <
GetFittness(city.colony[i]))
            {
                a = (rand() % (CITY_NUM - 1)) + 1;
                b = (rand() % (CITY_NUM - 1)) + 1;
                memcpy(Temp, city.colony[i],
sizeof(Temp));
                if (a > b)
                {
                    t = a;
                    a = b;
                    b = t;
                }
                for (m = a; m < (a + b) / 2; m++)
                {
                    t = Temp[m];
                    Temp[m] = Temp[a + b - m];
                    Temp[a + b - m] = t;
                }

                if (GetFittness(Temp) <
GetFittness(city.colony[i]))
                {
                    a = (rand() % (CITY_NUM - 1)) + 1;
                    b = (rand() % (CITY_NUM - 1)) + 1;
                    memcpy(Temp, city.colony[i],
sizeof(Temp));
                    if (a > b)
                    {
                        t = a;
                        a = b;
                        b = t;
                    }
                    for (m = a; m < (a + b) / 2; m++)
                    {
                        t = Temp[m];
```

```cpp
                        Temp[m] = Temp[a + b - m];
                        Temp[a + b - m] = t;
                    }
                }
            }
            memcpy(city.colony[i], Temp, sizeof(Temp));
        }
    }
}

void OutPut(TSP &city)
{
    // cout << "best route : " << endl;
    // for (int i = 0; i <= CITY_NUM; i++)
    // {
    //     cout << city.BestRooting[i] + 1;
    //     if (i != CITY_NUM)
    //         cout << " ";
    // }
    // cout << endl;
    cout << "best length : " << city.BestValue << endl;
}

int main()
{
    TSP city;
    srand(seed);

    // 讀取數據
    int m = 0, n = 0;
    double d = 0;
    int a, b, c;
    vector<vector<double>> data;
    while (cin >> a >> b >> c)
    {
        vector<double> t;
        t.push_back(a);
        t.push_back(b);
        t.push_back(c);
        data.push_back(t);
    }
    // cout << data.size() << endl;
    CITY_NUM = data.size();

    // double num[data.size()] = {0}, lont[data.size()] =
{0}, lati[data.size()] = {0};
    // for (int i = 0; i < data.size(); ++i)
    // {
    //     num[i] = data[i][0];
    //     lont[i] = data[i][1];
    //     lati[i] = data[i][2];
    // }

    CityPosition CityPos[data.size()];
    for (int i = 0; i < data.size(); i++)
    {
        CityPos[i].x = data[i][1];
        CityPos[i].y = data[i][2];
    }

    clock_t start_time = clock();

    CalculatDist(CityPos); //计算城市距离
    InitColony(city);      //生成初始种群
    CalFitness(city);      //计算适应度,同时选出最优的

    for (int i = 0; i < MAX_GEN; i++)
    {
        // cout << i << endl;
        Select(city);         //选择：轮盘赌法
        Cross(city, CROSS);   //交叉
        Mutation(city, MUT); //变异
        CalFitness(city);    //计算新的适应值
    }

    OutPut(city); //输出
    clock_t end_time = clock();
    cout << "Running time is: " <<
static_cast<double>(end_time - start_time) /
CLOCKS_PER_SEC * 1000 << "ms" << endl;
    return 0;
}
```

結果



```
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem2.cpp -o problem2 } ; if ($?) { .\problem2 }
1 37 52
2 49 49
3 52 64
4 20 26
5 40 30
6 21 47
7 17 63
8 31 62
9 52 33
10 51 21
11 42 41
12 31 32
13 5 25
14 12 42
15 36 16
16 52 41
17 27 23
18 17 33
19 13 13
20 57 58
^Z
best length : 243.179
Running time is: 3434ms
```

Question3

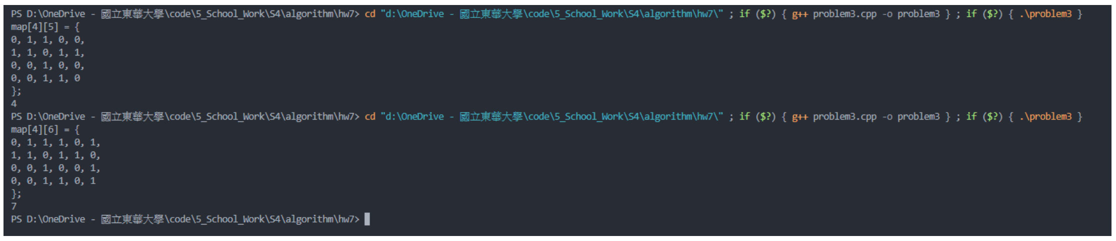
(1)

```
Int max <- 0.
For (each row in map)
    For (each item in row)
        If ( item is minefield) {
            clear adjacent mines(up、down、left、right) recursively, and return the
            value of mines we cleared this time as t.
            If ( t > max)
                t <- max.
        }
Print the max value.
```

(2)

```cpp
#include <cstdio>
#include <queue>
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <map>
#include <climits>
#include <sstream>

using namespace std;

bool make_map(string str, vector<vector<int>> &m)
{
    int row = 0, col = 0;

    str.erase(0, 4);
    // cout << str << endl;
    string matches("0123456789");
    size_t offset = str.find_first_of(matches);
    string next(str.substr(offset));

    stringstream intStream(next);
    intStream >> row;
    str.erase(0, str.find("]") + 2);

    offset = str.find_first_of(matches);
    next = str.substr(offset);

    stringstream intStream2(next);
    intStream2 >> col;
    str.erase(0, str.find("]") + 2);
    // cout << row << ":" << col << endl;

    int k = 0;
    vector<int> temp;
    for (int i = 0; i < row; ++i)
    {
        for (int j = 0; j < col; ++j)
        {
            while (str[k] != '1' && str[k] != '0')
                ++k;
            // cout << k << endl;
            temp.push_back(str[k] - '0');
            ++k;
        }
        m.push_back(temp);
        temp.clear();
    }
    return true;
}

int remove_minefield(int r, int c, vector<vector<int>> &m,
int n)
{
    if (r < 0 || c < 0 || r >= m.size() || c >=
m[r].size())
```

```
        return n;
    if (!m[r][c])
        return n;
    m[r][c] = 0;
    int dir_i[4] = {-1, 1, 0, 0};
    int dir_j[4] = {0, 0, -1, 1};
    for (int i = 0; i < 4; ++i)
    {
        n = remove_minefield(r + dir_i[i], c + dir_j[i],
m, n);
    }
    return n + 1;
}

int main()
{
    // cout << "Input: ";
    string input;
    getline(cin, input);
    while (input.substr(input.size() - 2, input.size()) !=
"};")
    {
        string t;
        getline(cin, t);
        input = input + t;
        input.erase(remove(input.begin(), input.end(), '
'), input.end());
    }
    vector<vector<int>> m;
    // cout << input << endl;
    // cout << input.size() << endl;

    make_map(input, m);

    // for (auto row : m)
    // {
    //     int j = 0;
    //     for (auto val : row)
    //     {
    //         cout << val << " ";
    //     }
    //     cout << endl;
    // }
    int res = 0;
    int max = 0;
    int i = 0;
    for (auto row : m)
    {
        int j = 0;
        for (auto val : row)
        {
            if (m[i][j])
            {
                int t = remove_minefield(i, j, m, 0);
                res++;
                if (t > max)
                    max = t;
                // cout << t << endl;
            }
            ++j;
        }
        ++i;
    }
    cout << max;
}

// map[4][5] = {
// 0, 1, 1, 0, 0,
// 1, 1, 0, 1, 1,
// 0, 0, 1, 0, 0,
// 0, 0, 1, 1, 0
// };

// map[4][6] = {
// 0, 1, 1, 1, 0, 1,
// 1, 1, 0, 1, 1, 0,
// 0, 0, 1, 0, 0, 1,
// 0, 0, 1, 1, 0, 1
// };
```
結果



```
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem3.cpp -o problem3 } ; if ($?) { .\problem3 }
map[4][5] = {
0, 1, 1, 0, 0,
1, 1, 0, 1, 1,
0, 0, 1, 0, 0,
0, 0, 1, 1, 0
};
4
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> cd "d:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7\" ; if ($?) { g++ problem3.cpp -o problem3 } ; if ($?) { .\problem3 }
map[4][6] = {
0, 1, 1, 1, 0, 1,
1, 1, 0, 1, 1, 0,
0, 0, 1, 0, 0, 1,
0, 0, 1, 1, 0, 1
};
7
PS D:\OneDrive - 國立東華大學\code\5_School_Work\S4\algorithm\hw7> █
```

(3)

Time complexity :

令 N 為輸入的字串總長度，我們的輸入可能為多行，預處理會將其整合為一條字串，其中不存在任何空格、換行，為 O(N)。再來會將其轉為二為陣列，掃瞄一輪字串就能轉換了，為 O(N)。最後計算時會掃描整個 map，為 O(N)，掃描時，遇到 minefield 會做多餘處裡，但只有 minefield 有被處理的機會，且 minefield 不可能被重複處理，為 O(N)，總複雜度為 O(N)(every case)。

Space complexity :

首先我們只是將多餘的字元去除，多餘的部分應為常數，所以 map 的大小為 O(N)。其他變數 r、c、i 為 O(1)。在輸入時的暫存 string t 跟轉為陣列時用到的 temp 皆不會大於 O(N)，總複雜度為 O(N)(every case)。