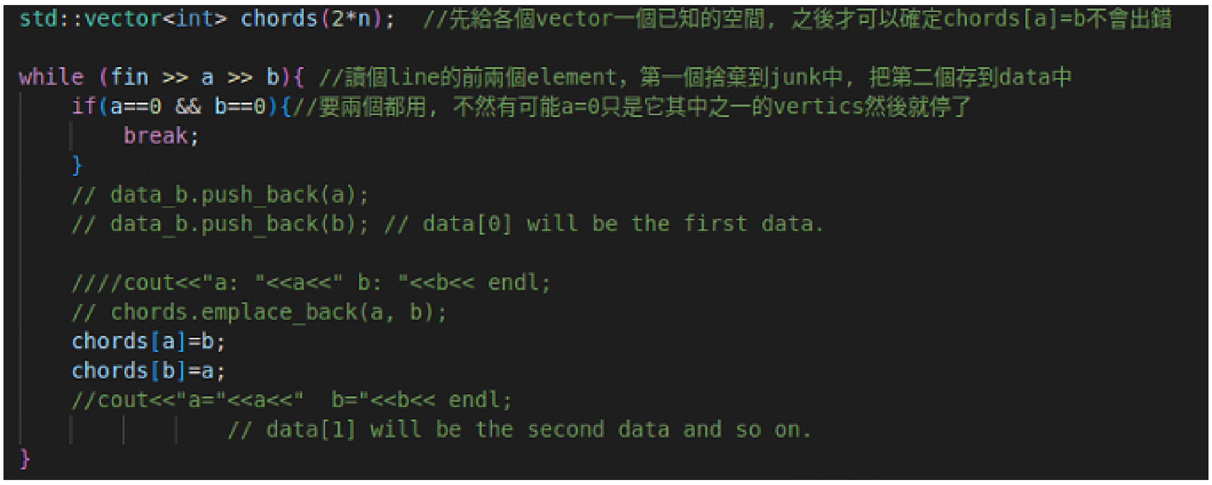
Programming Assignment #2 Maximum Planar Subset (Dynamic Programming)

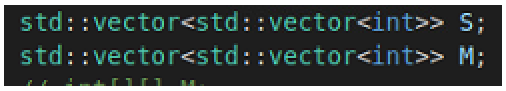
(a) Data structure (2%) How do you store the data of chords and/or other supporting information?

1. data of chord: 用一個一維vector去記錄每個vertics 上chord的另一端的paired vertics



```
std::vector<int> chords(2*n);  //先給各個vector一個已知的空間，之後才可以確定chords[a]=b不會出錯

while (fin >> a >> b){ //讀個line的前兩個element，第一個捨棄到junk中，把第二個存到data中
    if(a==0 && b==0){//要兩個都用，不然有可能a=0只是它其中之一的vertics然後就停了
        break;
    }
    // data_b.push_back(a);
    // data_b.push_back(b); // data[0] will be the first data.

    ////cout<<"a: "<<a<<" b: "<<b<< endl;
    // chords.emplace_back(a, b);
    chords[a]=b;
    chords[b]=a;
    //cout<<"a="<<a<<"  b="<<b<< endl;
                // data[1] will be the second data and so on.
}
```

2. other supporting information: 利用二維vector M 跟 S 去分別記錄maximum planar subset number 跟 the case of each M[i,j] is calculated from



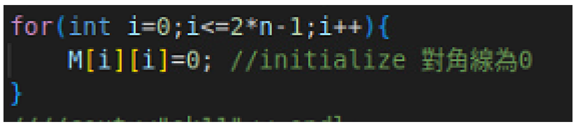
```
std::vector<std::vector<int>> S;
std::vector<std::vector<int>> M;
```

(b) Algorithm (3%)

- Describe the algorithm of your program, including how your program finds the number of chords and the chords themselves of the maximum planar subset
- E.g. if you use dynamic programming, you should mention the recurrence equation and how to utilize your data structure to do calculation

1. Use Dynamic Programming with recurrence equation:

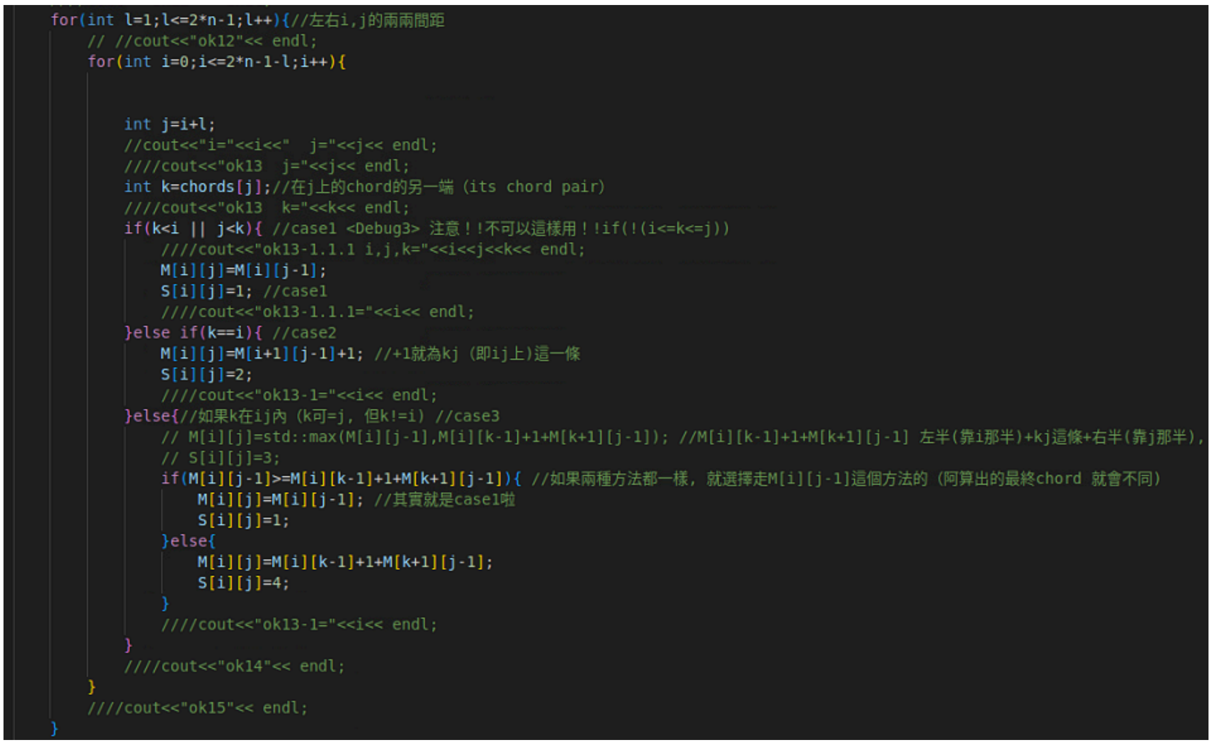
2. Construct table M[i,j] and S[i,j] with both size of 2n x 2n
   - M[i,j]: 用來存放某i,j區間下的 Maximum Planar Subset chord 數(即此區間最大互不交跨的chord的數量)
   - S[i,j]: 用來存放各M[i,j] 是由recurrence equation中的哪個case計算出來的，以便在recurrence trace回去找各chord的時候有辦法回朔找subset
3. 設定base case: (M[i],[j]對角線皆為0)



```
for(int i=0;i<=2*n-1;i++){
    M[i][i]=0; //initialize 對角線為0
}
```

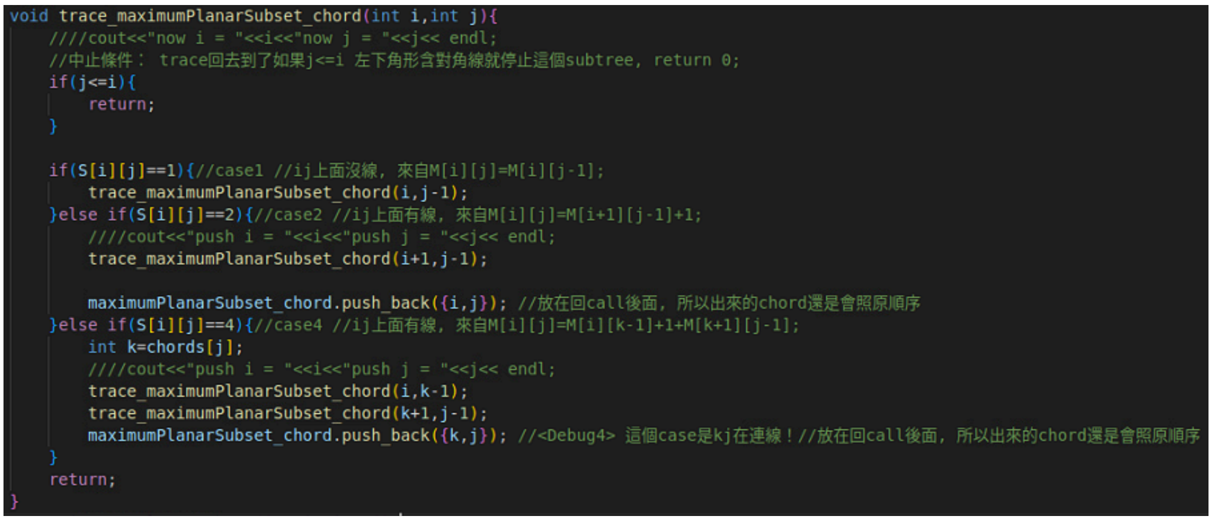
4. 填滿M[i,j] 與 S[i,j]

```
從l=1 to l-2n (l: i, j間間格)
    從i=0 to i=2n-1 (i: 範圍的左界)
        j=i+l (j: 範圍的右界)
        k=chords[j] (k 為j的chord 另一端的paired_vertics)
        利用k 的位置去判斷現在i,j為case 幾, M[i,j]要從哪個subset來
        紀錄M[i,j] 為 recurrence equation case1 or 2 or 3
        紀錄S[i,j] 為 計算M[i,j]時的case
```



```
for(int l=1;l<=2*n-1;l++){//左右i,j的兩兩間距
    // //cout<<"ok12"<< endl;
    for(int i=0;i<=2*n-1-l;i++){


        int j=i+l;
        //cout<<"i="<<i<<"  j="<<j<< endl;
        ////cout<<"ok13  j="<<j<< endl;
        int k=chords[j];//在j上的chord的另一端 (its chord pair)
        ////cout<<"ok13  k="<<k<< endl;
        if(k<i || j<k){ //case1 <Debug3> 注意！!不可以這樣用！!if(!(i<=k<=j))
            ////cout<<"ok13-1.1.1 i,j,k="<<i<<j<<k<< endl;
            M[i][j]=M[i][j-1];
            S[i][j]=1; //case1
            ////cout<<"ok13-1.1.1="<<i<< endl;
        }else if(k==i){ //case2
            M[i][j]=M[i+1][j-1]+1; //+1就為kj (即ij上)這一條
            S[i][j]=2;
            ////cout<<"ok13-1="<<i<< endl;
        }else{//如果k在ij內 (k可=j, 但k!=i) //case3
            // M[i][j]=std::max(M[i][j-1],M[i][k-1]+1+M[k+1][j-1]); //M[i][k-1]+1+M[k+1][j-1] 左半(靠i那半)+kj這條+右半(靠j那半),
            // S[i][j]=3;
            if(M[i][j-1]>=M[i][k-1]+1+M[k+1][j-1]){ //如果兩種方法都一樣, 就選擇走M[i][j-1]這個方法的 (阿算出的最終chord 就會不同)
                M[i][j]=M[i][j-1]; //其實就是case1啦
                S[i][j]=1;
            }else{
                M[i][j]=M[i][k-1]+1+M[k+1][j-1];
                S[i][j]=4;
            }
            ////cout<<"ok13-1="<<i<< endl;
        }
        ////cout<<"ok14"<< endl;
    }
    ////cout<<"ok15"<< endl;
}
```

5. 利用recursive 從M[0,11] 一路跟著S[i][j] 紀錄的case, 根據各case 的 recurrence equation 往回回溯, 若為case2 與 case4(case3的第二個情形), 則i,j為其中之一選擇的chord, 並存入maximumPlanarSubset_chord中 直到 j<=i 中止



```
void trace_maximumPlanarSubset_chord(int i,int j){
    ////cout<<"now i = "<<i<<"now j = "<<j<< endl;
    //中止條件： trace回去到了如果j<=i 左下角形含對角線就停止這個subtree, return 0;
    if(j<=i){
        return;
    }

    if(S[i][j]==1){//case1 //ij上面沒線, 來自M[i][j]=M[i][j-1];
        trace_maximumPlanarSubset_chord(i,j-1);
    }else if(S[i][j]==2){//case2 //ij上面有線, 來自M[i][j]=M[i+1][j-1]+1;
        ////cout<<"push i = "<<i<<"push j = "<<j<< endl;
        trace_maximumPlanarSubset_chord(i+1,j-1);

        maximumPlanarSubset_chord.push_back({i,j}); //放在回call後面, 所以出來的chord還是會照原順序
    }else if(S[i][j]==4){//case4 //ij上面有線, 來自M[i][j]=M[i][k-1]+1+M[k+1][j-1];
        int k=chords[j];
        ////cout<<"push i = "<<i<<"push j = "<<j<< endl;
        trace_maximumPlanarSubset_chord(i,k-1);
        trace_maximumPlanarSubset_chord(k+1,j-1);
        maximumPlanarSubset_chord.push_back({k,j}); //<Debug4> 這個case是kj在連線！//放在回call後面, 所以出來的chord還是會照原順序
    }
    return;
}
```

(c) Time Complexity Analysis (3%)

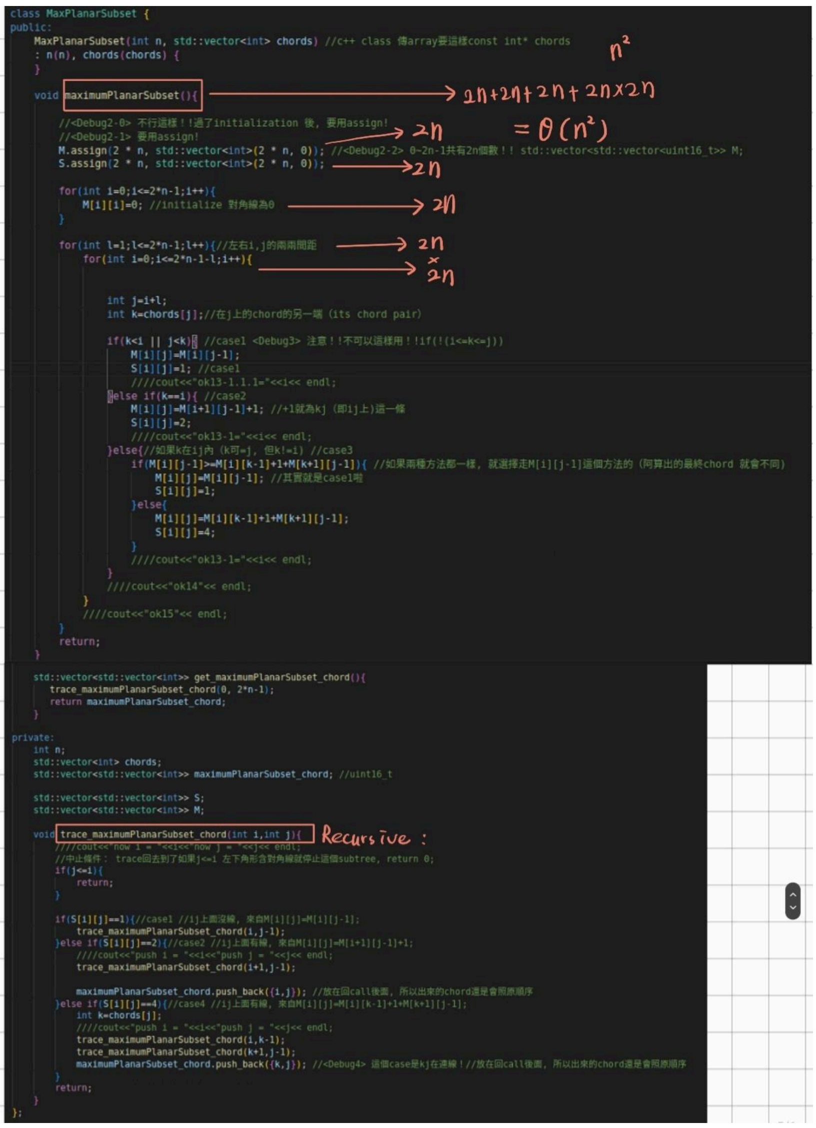
- In terms of number of chords n
- Analysis based on your implementation code

Ans:

因

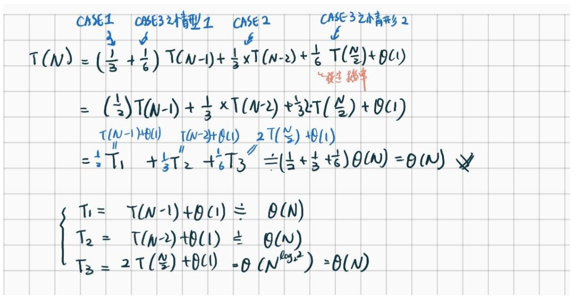
$$T(maximumPlanarSubset(\,)) = \theta(n^2)\ \&\ T(trace\_maximumPlanarSubset\_chord(\,)) = \theta(n^2)$$

$$\Rightarrow T(n) = \theta(n^2)$$



```cpp
class MaxPlanarSubset {
public:
    MaxPlanarSubset(int n, std::vector<int> chords) //c++ class 傳array要這樣const int* chords
    : n(n), chords(chords) {
    }

    void maximumPlanarSubset(){
        //<Debug2-0> 不行這樣！!過了initialization 後，要用assign!
        //<Debug2-1> 要用assign!
        M.assign(2 * n, std::vector<int>(2 * n, 0)); //<Debug2-2> 0~2n-1共有2n個數！! std::vector<std::vector<uint16_t>> M;
        S.assign(2 * n, std::vector<int>(2 * n, 0));

        for(int i=0;i<=2*n-1;i++){
            M[i][i]=0; //initialize 對角線為0
        }

        for(int l=1;l<=2*n-1;l++){//左右i,j的兩兩間距
            for(int i=0;i<=2*n-1-l;i++){

                int j=i+l;
                int k=chords[j];//在j上的chord的另一端 (its chord pair)

                if(k<i || j<k){ //case1 <Debug3> 注意！!不可以這樣用！!if(!(i<=k<=j))
                    M[i][j]=M[i][j-1];
                    S[i][j]=1; //case1
                    ////cout<<"ok13-1.1.1="<<i<< endl;
                }else if(k==i){ //case2
                    M[i][j]=M[i+1][j-1]+1; //+1就為kj (即ij上)這一條
                    S[i][j]=2;
                    ////cout<<"ok13-1="<<i<< endl;
                }else{//如果k在ij內 (k可=j, 但k!=i) //case3
                    if(M[i][j-1]>=M[i][k-1]+1+M[k+1][j-1]){ //如果兩種方法都一樣，就選擇走M[i][j-1]這個方法的 (阿算出的最終chord 就會不同)
                        M[i][j]=M[i][j-1]; //其實就是case1啦
                        S[i][j]=1;
                    }else{
                        M[i][j]=M[i][k-1]+1+M[k+1][j-1];
                        S[i][j]=4;
                    }
                    ////cout<<"ok13-1="<<i<< endl;
                }
                ////cout<<"ok14"<< endl;
            }
            ////cout<<"ok15"<< endl;
        }
        return;
    }

    std::vector<std::vector<int>> get_maximumPlanarSubset_chord(){
        trace_maximumPlanarSubset_chord(0, 2*n-1);
        return maximumPlanarSubset_chord;
    }

private:
    int n;
    std::vector<int> chords;
    std::vector<std::vector<int>> maximumPlanarSubset_chord; //uint16_t

    std::vector<std::vector<int>> S;
    std::vector<std::vector<int>> M;

    void trace_maximumPlanarSubset_chord(int i,int j){
        ////cout<<"now i = "<<i<<"now j = "<<j<< endl;
        //中止條件： trace回去到了如果j<=i 左下角形含對角線就停止這個subtree, return 0;
        if(j<=i){
            return;
        }

        if(S[i][j]==1){//case1 //ij上面沒線, 來自M[i][j]=M[i][j-1];
            trace_maximumPlanarSubset_chord(i,j-1);
        }else if(S[i][j]==2){//case2 //ij上面有線, 來自M[i][j]=M[i+1][j-1]+1;
            ////cout<<"push i = "<<i<<"push j = "<<j<< endl;
            trace_maximumPlanarSubset_chord(i+1,j-1);

            maximumPlanarSubset_chord.push_back({i,j}); //放在回call後面，所以出來的chord還是會照原順序
        }else if(S[i][j]==4){//case4 //ij上面有線, 來自M[i][j]=M[i][k-1]+1+M[k+1][j-1];
            int k=chords[j];
            ////cout<<"push i = "<<i<<"push j = "<<j<< endl;
            trace_maximumPlanarSubset_chord(i,k-1);
            trace_maximumPlanarSubset_chord(k+1,j-1);
            maximumPlanarSubset_chord.push_back({k,j}); //<Debug4> 這個case是kj在連線！//放在回call後面，所以出來的chord還是會照原順序
        }
        return;
    }
};
```

n^2
$2n+2n+2n+2n\times2n$
$2n$
$=\theta(n^2)$
$2n$
$2n$
$2n$
$\times$
$2n$
Recursive :



$$T(N) = \left(\underset{\text{CASE1}}{\tfrac{1}{3}} + \underset{\text{CASE3之情型1}}{\tfrac{1}{6}}\right) T(N-1) + \underset{\text{CASE2}}{\tfrac{1}{3}} \times T(N-2) + \underset{\text{CASE3之小情形2}}{\tfrac{1}{6}} \underset{\text{發生機率}}{T\left(\tfrac{N}{2}\right)} + \theta(1)$$

$$= \left(\tfrac{1}{2}\right)T(N-1) + \tfrac{1}{3} \times T(N-2) + \tfrac{1}{3}2T\left(\tfrac{N}{2}\right) + \theta(1)$$

$$= \tfrac{1}{2}\overset{T(N-1)+\theta(1)}{T_1} + \tfrac{1}{3}\overset{T(N-2)+\theta(1)}{T_2} + \tfrac{1}{6}\overset{2T(\frac{N}{2})+\theta(1)}{T_3} \doteqdot \left(\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{6}\right)\theta(N) = \theta(N) \cancel{\ }$$

$$\begin{cases} T_1 = T(N-1)+\theta(1) \doteqdot \theta(N) \\ T_2 = T(N-2)+\theta(1) \doteqdot \theta(N) \\ T_3 = 2T\left(\tfrac{N}{2}\right)+\theta(1) = \theta(N^{\log_2 2}) = \theta(N) \end{cases}$$

(d) README (2%)

- Clear instruction of how to compile and execute your program
- You may refer to PA#1 README

```
This is README file for Algorithm PA#2
Author: <余佳融 B10611038>
Date: 2024/11/3
=====
SYNOPSIS:

./bin/mps <input_file_name> <output_file_name>
ex: ./bin/mps inputs/12.in outputs/12.out
This program supports finding the maximum planar subset of a 0~2n-1 vertices
=====
DIRECTORY:

bin/       executable binary
doc/       reports
inputs/   input data (all chords)
outputs/  output result (maximum planar subset chords)
src/       source C++ codes
utility/  checker
======
HOW TO COMPILE:

        Under the root directory of this PA, simply type
        make
======
HOW TO RUN:

        cd PA2/
        ./bin/mps <input_file_name> <output_file_name>
        For example,
        under <student_id>_pa1
        ./bin/mps inputs/12.in outputs/12.out
======
OTHER NOTICE:
```