Algorithm HW2 PS1

Chien-I, Tseng

B11901072

1 Problem Statement

This program solves the Maximum Planar Subset problem by finding the maximum number of non-intersecting chords in a circle and listing the chords in the solution.

2 Algorithm Design

2.1 Dynamic Programming Approach

2.1.1 States Definition

1. dp[i][j] represents the maximum number of the non-intersecting chords in the region between i and j.
2. path[i][j] represents the endpoints of the chords chosen for the solution.

2.1.2 Base Cases

1. when i $\geq$ j, dp[i][j] = 0
2. if dp[i][j] has calculated, return the value

2.1.3 Recursion

case1: if there is a chord starting at i and ending at "end", dp[i][j] = 1+dp[i+1][end-1]+dp[end+1][j]
case2: skip i, dp[i][j] = dp[i+1][j]
chose the maximum value above these two cases.

2.2 Space Complexity and Time Complexity

We have O(n^2) dp table and O(n^2) path table to fill, so the space complexity is O(n^2).
For each space, there are O(n) possible chords to consider

3 Data Structure

3.1 Chord

Written in the file ./src/maxPlanarSubset.h

```cpp
struct Chord {
    int start, end;
    Chord(int s, int e) : start(s), end(e) {}
};
```

3.2 maxPlanarSubset

```cpp
class MaxPlanarSubset {
private:
    std::vector<Chord> chords;
    std::vector<std::vector<int>> dp;
    std::vector<std::vector<int>> path;
    int n;  // number of chords

    // Helper functions
    bool intersect(const Chord& a, const Chord& b);
    void reconstructSolution(int i, int j, std::vector<Chord>& result);

public:
    // Constructor
    explicit MaxPlanarSubset(int size);

    // Public member functions
    void addChord(int start, int end);
    int solve(int i, int j);
    std::vector<Chord> getSolution();
};
```

4 Function

4.1 Solve

Written in the file ./src/maxPlanarSubset.cpp
This function contains the main logic of dynamic programming and store the endpoints in path[i][j] for the solution reconstruction.

```cpp
int MaxPlanarSubset::solve(int i, int j) {
    if (i >= j) return 0;
    if (dp[i][j] != -1) return dp[i][j];

    // Don't include chord starting at i
    int val = solve(i + 1, j);

    // Try to include chords starting at i
    for (const Chord& chord : chords) {
        if (chord.start == i && chord.end <= j) {
            int current = 1 + solve(i + 1, chord.end - 1) + solve(chord.end + 1, j);
            if (current > val) {
                val = current;
                path[i][j] = chord.end;
            }
        }
    }

    dp[i][j] = val;
    return val;
}
```

4.2 Solution Reconstruction

According to path[i][j], we can reconstruct the solution chord set.

```cpp
void MaxPlanarSubset::reconstructSolution(int i, int j, std::vector<Chord>& result) {
    if (i >= j) return;

    if (path[i][j] == -1) {
        reconstructSolution(i + 1, j, result);
    } else {
        result.push_back(Chord(i, path[i][j]));
        reconstructSolution(i + 1, path[i][j] - 1, result);
        reconstructSolution(path[i][j] + 1, j, result);
    }
}
```