Programming Assignment 2 Report: Maximum Planar Subset

Data Structures

1. Chord Structure

```
struct Chord {
    int start, end;  // Represents the endpoints of a chord
};
```

This simple structure represents a chord in the circle, where:

- start: The starting point of the chord
- end: The ending point of the chord

2. Dynamic Programming Tables

```
// DP table for storing maximum counts
std::vector<std::vector<int>> dp;

// Table for reconstructing the solution
std::vector<std::vector<int>> solution;
```

- `dp[i][j]`: Stores the maximum number of non-crossing chords in the range [i,j]
- `solution[i][j]`: Stores the endpoint of the chord chosen at position i that gives the optimal solution

Algorithm Implementation

1. Dynamic Programming Approach

The solution uses a bottom-up dynamic programming approach where:

- Base case: Single points cannot form chords (`dp[i][i]` = 0)
- For each subproblem [i,j], we consider:
    - Not using any chord starting at i
    - Using a chord starting at i and ending at some $k \leq j$

2. Solution Reconstruction

The solution is reconstructed using backtracking through the solution table:

- Start from the full range [0, 2n-1]
- For each position, check if a chord was used in the optimal solution
- Recursively process the remaining subranges

Key Findings and Challenges

1. Input/Output Handling

- Proper file I/O handling is crucial for large test cases
- Error checking for file operations is important

2. Memory Management

- Using vectors for dynamic allocation simplifies memory management
- The DP tables require $O(n^2)$ space complexity

3. Debug Support

- Added comprehensive debug output for development
- Used preprocessor directives for conditional debugging

4. Performance Considerations

- Bottom-up DP approach avoids recursion overhead
- Solution reconstruction is efficient with the solution table

Optimization Techniques

1. Space Optimization

Used two separate tables for clarity:

- One for counting maximum chords
- One for reconstructing the solution

2. Time Optimization

- Avoided redundant calculations in DP
- Used reference parameters to prevent copying

Testing Strategy

1. Input Validation

- Tested with various input sizes
- Verified edge cases (empty input, single chord)

2. Output Verification

- Checked solution validity
- Verified non-crossing property of selected chords

Conclusions

The Maximum Planar Subset problem demonstrates the power of dynamic programming in solving complex geometric problems. The implementation successfully handles:

- Efficient computation of maximum non-crossing chords
- Correct solution reconstruction
- Proper memory management
- Robust input/output handling

The use of appropriate data structures and careful implementation of the DP algorithm resulted in a solution that effectively solves the problem while maintaining good performance characteristics.