Algorithms Programming Assignment #2

B10901006 電機三 鄒昀樺

Data Structure

- A 2n*2n 2D int array **M** saves the number of chords in the maximum planar subset of (i, j), where i and j represent points on the circle with i<=j, in M(i, j).
- A 2n*2n 2D char array **D** saves the fill-in condition of M(i, j).
- A 1D array **chords** records the chords from the input.
- A 1D array **Doc** documents the chords chosen in the final mps output.

Algorithms

I use a recursive function 'Mfill' to fill in the array M. Each space (i, j) in M is filled in according to three circumstances: (let k be the opposite endpoint the chords with one endpoint j)

1. k is not between i and j
   ➔ M(i, j) = M(i, j-1)
2. k is between i and j
   ➔ M(i, j) = MAX(M(i, j-1), M(i, k-1)+M(k+1, j-1)+1)
3. k is i
   ➔ M(i, j) = M(i+1, j-1)+1

The fill-in circumstance is recorded as a char in D(i, j). After filling in M and D, I use another recursive function 'trace' to identify chords chosen in each M(i, j). The function uses D to trace back the chords added in each recursion of Mfill starting from M(0, 2n-1). The chosen chords are saved in Doc. No sorting is needed at the end of the algorithm because the chosen chords are already sorted automatically when they are saved as Doc[i] = j.

Time and Space Complexity Analysis

The time and space complexity in this algorithm are both $O(n^2)$, where n is the input number of points on the circle (which actually means '2n' in the problem description, but no influence). The complexity results from filling in 2D arrays. In the worst case of the top-down algorithm, every space in M and D needed to be filled. Therefore, the total cost in time and space are both $O(n^2)$.

Top-down and Bottom-up Comparison

At first I used the bottom-up measure. In the 100000.in case, it spent about 6 to 7 minutes to finish the whole program. However, after I rewrite the program in the top-down measure, the time consumption has shrunk to less than 2 minutes, which is much faster. The reason is that the top-down measure in most case does not have to go through every space in the 2D array, which bottom-up does. Top-down measure only

computes the entries that matters. In this problem, the top-down measure outerperforms the bottom-up measure.