EE4033 Algorithms

Programming Assignment #2

B11901110 陳璿吉

I. README

Make sure you are at the root directory of PA2, type "`make`" to compile the C++ code. The binary file `mps` will be stored in the `bin` folder. Then, type the following:

```
./bin/mps inputs/xx.in outputs/xx.out
```

You can check the output file by navigating to the `outputs` folder.

II. Data structure

In this programming assignment, a vector target of $2n$ elements is used to store the chord information. For example, if (a, b) is a chord, then target$[a] = b$ and target$[b] = a$. (See Figure below)

```
int n, chords_number; //number of points and chords

fin >> buffer;

n = buffer;

chords_number = n/2;

vector<int> target(n, 0);

for(int i = 0; i < chords_number ; i++)
{
    fin >> buffer;
    int start = buffer;

    fin >> buffer;

    int end = buffer;
    target[start] = end;
    target[end] = start;
}
```

III. Algorithm

We employ bottom-up approach of dynamic programming to solve this problem. Let $M[i,j]$ be the maximum number of non-intersecting chords in (i,j). Let k be the point connected to j, we have following recurrence:

$$M[i,j] = \begin{cases} 0, & \text{if } i = j. \text{ (base case)} \\ \max(M[i,j-1], M[i+1,j]), & \text{if } k < i \text{ or } k > j. \\ \max(M[i,k-1] + M[k,j], M[i,j-1], M[i+1,j]), & \text{if } i < k < j \\ M[i,j-1] + 1, & \text{if } k = j \end{cases}$$

In addition to finding $M[0, 2n-1]$, we are also required to find all the non-intersecting chords covered in $(0, 2n-1)$. To implement this in my program, I created two tables called `M` and `table`. The former stores the number of chords, while the latter stores each starting point of the chord.

The algorithm uses two loops, the first one loops through length 1 to $2n-1$, and the second one loops through starting point i from 0 to $2n-1-\text{length}$. We define $j = i + \text{length}$, hence the program finds the solution from $(0,1), (1,2), \ldots (2n-2, 2n-1), (0,2), (1,3), \ldots, (0, 2n-1)$. I simply copied the solution of previous entries to new entries if they are different. Otherwise, I used pointers to point the new entry to the old one to save some memory.

IV. Time Complexity Analysis

Since this solution uses two for loops, the running time of this program is $O(n^2)$, where n is the number of chords. When $n = 12$ and $n = 1000$, the program takes less than 1 second to execute. When $n = 10000$, the program takes around 45 seconds to execute. As copying existing solution to new entries takes a lot of space, the program slows down dramatically as n increases.