Programming Assignment #2

B11901029 李致頡

Code Explanation

1. Storing chords set C

```cpp
// Read in number of vertices
inFile >> N;
vector<int> C(N);
// Read in N/2 pairs of vertices
for (int i = 0; i < N/2; ++i) {
    inFile >> v1 >> v2;
    C[v1] = v2;
    C[v2] = v1;
}
```

We first read in the number of vertices $N = 2n$, then build a list `C` of size N to save the chords: if there's a chord $\overline{35}$, we assign `C[3]=5`, `C[5]=3`, so that we can access the chord from either endpoint.

2. Solve MPS problem: bottom-up dynamic programming with solution construction
For the number k such that $\overline{kj} \in C$,

$$M(i,j) = \begin{cases} 0\ , & i \geq j \\ M(i,j-1)\ , & k \notin [i,j] \\ \max\{M(i,j-1), M(i,k-1)+1+M(k+1,j-1)\}, k \in (i,j) \\ M(i+1,j-1)+1\ , & k = i \end{cases}$$

Since almost all $M(i,j)$ need to be calculated, top-down implementation would need more memory due to recursive function call, so we choose bottom-up implementation. First, construct a $(2n) \times (2n)$ `MPS` table, initialize with all 0.

```cpp
// MPS Table: default all 0
vector<vector<unsigned short>> MPS(N, vector<unsigned short>(N, 0));
```

And use bottom-up iteration to implement the $M(i,j)$ recursion relationship.
For later solution construction, it's also needed to memorize the recursion path.
To save the memory required, notice that $M(i,j)$ only use half of `MPS`:
$M(i,i) = 0$, and if $i > j$, $M(i,j)$ is meaningless. Rather than building another table, we use the unused half of `MPS` to save another half table `ARROW`, where `ARROW[i][j]` $(i < j)$ is saved at `MPS[N-1-i][N-1-j]`. (diagonal place)

```cpp
// Make use of the space unused in MPS to store Arrows
#define Arrows(i,j) MPS[N-1-i][N-1-j]
#define KOUT 1
#define KIN 2
#define KBOUNDARY 3
```

```cpp
unsigned short ComputeMPS(vector<int>& C, vector<vector<unsigned short>>& MPS) {
    // Bottom-up Iterative Implementation
    int N = C.size();
    for (int i = N-2; i >= 0; --i) {
        for (int j = i+1; j < N; ++j) {
            int k = C[j];
            // k not in [i,j]
            if (k < i || k > j) {
                MPS[i][j] = MPS[i][j-1];
                Arrows(i,j) = KOUT;
            }
            // k in (i,j)
            else if (i < k && k < j) {
                if (k+1 <= j-1) {
                    MPS[i][j] = MPS[i][k-1] + 1 + MPS[k+1][j-1];
                } else {
                    MPS[i][j] = MPS[i][k-1] + 1;
                }
                // max{MPS[i][j-1], MPS[i][k-1]+1+MPS[k+1][j-1]}
                if (MPS[i][j-1] >= MPS[i][j]) {
                    MPS[i][j] = MPS[i][j-1];
                    Arrows(i,j) = KOUT;
                } else {
                    Arrows(i,j) = KIN;
                }
            }
            // k == i
            else {
                if (i+1 <= j-1) {
                    MPS[i][j] = MPS[i+1][j-1] + 1;
                } else {
                    MPS[i][j] = 1;
                }
                Arrows(i,j) = KBOUNDARY;
            }
        }
    }
    return MPS[0][N-1];
}
```

A nested for-loop is used, so the time complexity is $O(N^2)$.

3. Print the resulting chords in MPS

   Like what is taught in class to print LCS, we follow the recursion path recorded when computing MPS to print the resulting chords. Also, make sure that for each chord, the vertex with smaller order is printed out first.

```
1  void PrintMPS(ofstream& outFile, vector<int>& C, vector
   <vector<unsigned short>>& MPS, int i, int j) {
2      if (i >= j) return;
3      int N = C.size();
4      if (Arrows(i,j) == KOUT) {
5          PrintMPS(outFile, C, MPS, i, j-1);
6      } else if (Arrows(i,j) == KIN) {
7          PrintMPS(outFile, C, MPS, i, C[j]-1);
8          outFile << C[j] << " " << j << '\n';
9          PrintMPS(outFile, C, MPS, C[j]+1, j-1);
10     } else {
11         outFile << i << " " << j << '\n';
12         PrintMPS(outFile, C, MPS, i+1, j-1);
13     }
14 }
```

   Time complexity of calling `PrintMPS(outFile, C, MPS, 0, N-1)`:
   We first analyze the most complex case where the recursion path split: `ARROW[i][j] == KIN`. Denote $m = j - i + 1$, then we can formulate the always-splitting worst case by

$$T(m) = \begin{cases} \Theta(1), & m = 1 \\ T(c\alpha m) + T(c(1-\alpha)m) + \Theta(1), & m \geq 1 \end{cases}$$

   where $c = \frac{(k-1-i+1)+(j-1-k-1+1)}{j-i+1} = \frac{m-2}{m} < 1$ is the problem size ratio after split and before split, and $\alpha = \frac{k-i}{m-2} < 1$ stands for the splitting proportion each time.

   By Recursion Tree analysis, we can see that $T(N) \in O(N)$.

   For the other 2 cases, $T(m) = T(m-1)$ or $T(m-2)$ also in $O(N)$.

   Mix of 3 cases means less splitting, so it will not be worse than the always-splitting case. Therefore, `PrintMPS(outFile, C, MPS, 0, N-1)` $\in O(N)$.

Result

Run the above program on EDA Union server edaU13, all runtime < 10 minutes.

Input File	Approximate Runtime (sec)	Number of MPS chords
12.in	< 1	3
1000.in	< 1	52
10000.in	< 1	176
100000.in	58	566