Alg22 PA2

電機四 B08901123 何景盛

Code:

Input data 為多組成對的 endpoints 組成，我用 vector<int> nodePairs 的方式來存，假設一組的 endpoints 有兩點 j, k，用 index 的來代表其中一個 endpoint 來存進 vector 裡(nodePairs[j] = k 和 nodePairs[k] = j)。

整體程式分為四個 function 和一個 main function:

- Help_message(): 當發生 input arguments 不符會時，會 print 出正確格式。
- findAnotherEndpoint(int j, vector<int> nodePairs): 收到一個 endpoint 時會 return 弦的另一個 endpoint。
- mps(int n, vector<int> nodePairs): 收到 endpoint 的數目和弦的 data 找出最大弦的數目並把它 return。
- findChord(int i, int j, vector<vector<int>> &m, vector<int> &nodePairs, vector<int> &kArray): 收到 i, j 範圍、由 DP 所寫的 mps table、弦的 data 和用來記錄最大不相交弦的 endpoint array，獲得以上 input 再用 recursion 把 endpoints 寫入 kArray。

```
 9 void help_message()
10 {
11         cout << "Usage: mps <input_file_name> <output_file_name>" << endl;
12 }
```

Line11: print 出正確的 input argument。

```
13
14 int findAnotherEndpoint(int j, vector<int> nodePairs)
15 {
16         return nodePairs[j];
17 }
```

Line16: 根據 nodePairs array 去 return endpoint j 的另一個 endpoint。

```
19 vector<vector<int>> mps(int n, vector<int> nodePairs)
20 {
21         vector<vector<int>> m(n, vector<int>(n, 0));
22         for (int j = 0; j < n; j++)
23         {
24                 int k = findAnotherEndpoint(j, nodePairs);
25                 for (int i = 0; i <= j - 1; i++)
26                 {
27                         if (i <= k && k <= j - 1 && m[i][k - 1] + 1 + m[k + 1][j - 1] > m[i][j - 1])
28                         {
29                                 m[i][j] = m[i][k - 1] + 1 + m[k + 1][j - 1];
30                         }
31                         else
32                         {
33                                 m[i][j] = m[i][j - 1];
34                         }
35                 }
36         }
37         return m;
38 }
```

Line21: 建立和初始化 DP table m。

Line22-36: 根據上課給的論文裡的 pseudo-code 的操作來填 m。

$MAXIMUM\text{-}PLANAR\text{-}SUBSET(C, N)$
for $j \leftarrow 0$ **to** $2N - 1$ **do**
  $k \leftarrow$ the number such that $kj \in C$ or $jk \in C$
  for $i \leftarrow 0$ **to** $j - 1$ **do**
    if $i \leq k \leq j - 1$ and $|M(i, k - 1)| + 1 + |M(k + 1, j - 1)| > M(i, j - 1)$
      $M(i, j) \leftarrow M(i, k - 1) \cup +1 \cup M(k + 1, j - 1)$
    else $M(i, j) \leftarrow M(i, j - 1)$

```
40 void findChord(int i, int j, vector<vector<int>> &m, vector<int> &nodePairs, vector<int> &kArray)
41 {
42         int k = 0;
43         k = findAnotherEndpoint(j, nodePairs);
44         if (j > i)
45         {
46                 if (k > j || k < i)
47                 {
48                         findChord(i, j - 1, m, nodePairs, kArray);
49                 }
50                 else if (k > i && k < j)
51                 {
52                         if (m[i][k - 1] + m[k][j] > m[i][j - 1])
53                         {
54                                 findChord(i, k - 1, m, nodePairs, kArray);
55                                 findChord(k, j, m, nodePairs, kArray);
56                         }
57                         else
58                         {
59                                 findChord(i, j - 1, m, nodePairs, kArray);
60                         }
61                 }
62                 else
63                 {
64                         kArray.push_back(k);
65                         findChord(i + 1, j - 1, m, nodePairs, kArray);
66                 }
67         }
68 }
```

用 recursion 的方法來找 chord，並分成三種 case:

Line46-49: case1(k>j||k<i)，k 在 i, j 外面不會把它記在 kArray 內。

Line50-61: case2(i<k<j)，k, j 把 i, j 分開兩部，所以要在這兩個區間做 recursion 繼續找 jk。

Line62-66: case3(k=i)，因為 k=i，所以直接記在 kArray 內。

```
70 int main(int argc, char *argv[])
71 {
72
73         if (argc != 3)
74         {
75                 help_message();
76                 return 0;
77         }
78
79         // read input and build node vector
80         fstream fin(argv[1]);
81         fstream fout;
82         fout.open(argv[2], ios::out);
83         int n;
84         int a, b;
85         fin >> n;
86         vector<int> nodePairs(n, 0);
87         vector<int> kArray;
88         while (fin >> a >> b)
89         {
90                 nodePairs[a] = b;
91                 nodePairs[b] = a;
92         }
93
94         vector<vector<int>> m = mps(n, nodePairs); // mps number
95         findChord(0, n - 1, m, nodePairs, kArray); // find chord array
96
97         // write file
98         fout << m[0][n - 1] << endl;
99         for (int i = 0; i < kArray.size(); i++)
100        {
101                fout << kArray[i] << " " << findAnotherEndpoint(kArray[i], nodePairs) << endl;
102        }
103        fin.close();
104        fout.close();
105
106        return 0;
107 }
```

Line73-77: 處理不合法的 input argument 並執行 help_message()。

Line80-82: 建構 fin 和 fout 來讀和寫檔案。

Line83-92: 建立一些變數，把 input file 的第一行 endpoint 數存在 n，其他的 data 讀出來並存在 nodePairs 內。

Line94: 用 m 矩陣來存 mps() return 出來的矩陣。

Line95: 執行 findChord()來把 mps 弦的 k endpoint 存在 kArray 內。

Line98: 寫入弦最大值。

Line99-102: 把所找到的每條弦的兩個 endpoints 寫入 output file。

Line103-104: 關閉 fin 和 fout。

Makefile:

```
all : src/mps.cpp
        g++ -02 src/mps.cpp -o bin/mps

clean:
        rm *.o
```

Analysis:

Time complexity:

- Help_message(): O(1)
- findAnotherEndpoint(): O(1)
- mps function(): 用兩層 for loop 來完成， O(n^2)
- findChord(): 只是在 DP table 裡遞迴出弦，只會走 endpoint 數目的次數，O(n)

Space complexity:

主要是建一個 n*n 大小的 DP table 來存最大弦數目， O(n^2)。

Discussion:

在這次作業我認為最主要的是 data structure，一開始我是用 vector<pair<int, int>>的類型來存 input data，pair 是(j, k)，但在給定 j 找 k 的時候，我用 for loop 來找，這會用 linear time，在理論上兩個 O(n)是跟一個 O(n)一樣，所以我也沒想很多，直到在跑程式的時候，我發現在小 case 可以跑，但 case 一大就跑不動，所以我之後就用現在的方法用 index 當一個 endpoint 來儲存 input data，time complexity 為 O(1)，然後就可以跑到 100000 的 case。另外的一個發現是 function 的 input argument 的寫法也是很重要，在 findChord function 裡原本 input 那些 nodePairs, m 是沒有用 reference，然後就跑不動，但之後加了 reference 就可以，所以 input argument 也是重點。