陳守仁
B09901116
EE3

PA2 Report

Input Size	Maximum Planar Subset	
	CPU time (ms)	Memory (KB)
12.in	0.143	5896
1000.in	8.943	9856
10000.in	834.829	396832
100000.in	175470	39160380

Note 1: The code ran on EDA Union Lab Machines(might be affected by amount of people using)

<u>**Data Structure used in the Program:**</u>

<u>**main.cpp:**</u>

Imports compute.h for class Computation that contains functions: "ReadFile(const char *)", "WriteFile(const char *)", "calcMPS()", "getResult(int i, int j)". What my "main" does is to call the functions in the order of Read, Calculate MPS, Record Result, and Write.

<u>**compute.h**</u>

Compute.h file contains class Computation that contains functions defined in compute.cpp. Public contains: Computation() constructor, ~Computation() destructor, void calcMPS(), void recordChord(int i, int j), void ReadFile(const char *), void WriteFile(const char *). Private contains a few variables that will be used in our compute.cpp file, there are int sampleSize, int **mps, int *recordChord and int *chords.

<u>Each function functions:</u>

Destructor : delete the pointer to an object allocated with new.

ReadFile : Take in the file and record the inputs in arrays and a variable.

calcMPS : The idea is to use a bottom up approach by iteratively constructing solutions from the smallest to the biggest subproblem, and recording the max number of chords in each of the array[i][j], and the array[0][n-1] will contain the maximum number of chords that we want.

getResult : uses a top down recursive function to record down the chords(vertices) that form a maximum planar subset in an array.

WriteFile : using array recorded in getResult and the maximum number of chords from calcMPS to output the output file.

Each Variable's Purpose:

int sampleSize : record the total number of vertices given in the input file. Used in all the functions.

*int **mps :* initialized to 0, and +1 if there's a chord exist in i to j, finally, it will record the maximum number of non-intersecting chords possible in the interval of i to j in each row and column.

*int *recordChord :* initialized to 0, using mps array and chords array to find out the chord that forms an MPS and those chords are set to 1.

*int *chords :* initialize during ReadFile where it stores the pairs that make up a chord.

compute.cpp

Detail Explanation of my Code:

ReadFile : Take in the file and then save the first line onto sampleSize to record the total amount of vertices in the circle. To pair the chord, since it contains vertices of 0 to n-1, we can record the chord in an array of size "sampleSize" and input two numbers of each line into it, pairing them like chords[num1] = num2 and chords[num2] = num1. Initialize recordChord[sampleSize] to 0 here since we used a recursive function to record at getResult function.

calcMPS : I open up a 2d array of size nxn, where n = number of vertices, using new and initialize all of them to 0. Then I created a double for loop from j=0 to n-1 and from i = j-1 down to 0 to iterate through all the i and j combinations once in a bottom-up form. Then I set an int k = chords[j] so j and k are connected chords. Using the hints given by the TA in the videos, I write down the three cases.

1) When ij are chords(k==i), then mps[i][j]=mps[i+1][j-1]+1.
2) When k is within the interval of i to j, then mps[i]][j]=mps[i][k-1]+1+mps[k+1][j-1]. Also in this case, we have to make sure mps[i][k-1]+1+mps[k+1][j-1] is bigger than mps[i][j-1] since we don't want our previous case(mps[i][j-1]) bigger than out current case(bottom up case).
3) When k is not in i and j, then mps[i][j] = mps[i][j-1].
Case 1 means ij is a chord, so add one then check if in ij, there are other possible chords(mps[i][j]=mps[i+1][j-1]).
Case 2 means jk is a chord so add one, then check if between i and k-1 and k+1 and j-1 there are any possible chords.
Case 3 means that k is not with ij, and ij is not a chord, so we can iterate to close up the checking interval to i to j-1 to check if there are any possible chords.

At the end of this bottom up iterative process, we will get the maximum planar subset at the top right corner of the table or at mps[0][sampleSize-1], or maximum planar subset of each respective row and column.

getResult: The input will be the interval i and j, and we want to record all the chords that are MPS in i and j. Using a top down method to record only the chords needed. First check mps[i][j] ==mps[i][j-1], since if it is the same, then it means that the number of chords did not change, and it is not the vertices of the chord. Until the recursive function finds a k within i and j or i and j is a chord, then it must be a vertex. Then recursively call the interval between to check if there are inner chords for both the second and third case.

WriteFile: The write files run getResult(0, sampleSize-1). First print out mps[0][sampleSize-1], then iterate in order through recordChord and output each vertices and its chords.

Others and Findings

When I was writing the code, I didn't focus on the memory space used. When I successfully ran the 12, 1000, and 10000 cases, I started to run the 100,000 cases on my computer, it was running really slow. I notice that creating a 2d array of 100,000 x 100,000 will take up 40GB of memory space and that is a consideration that must be accounted for when writing a code. I tried to create a vector with only j from 0 to 2N-1 and i from 0 to j, but there were a lot of segmentation errors when I did it, so I decided not to use it. The space complexity is O(nxn). The time complexity calcMPS is O(nxn).