Algorithms PA2 Report

B08901207 EE3 呂俐君

Since the deadline was changed, I edited a “New version” based on my old ones.
Below are two versions of my report, first is new with old one follows.

Note: running on EDA Union port 40051.

（A）Code Design Concept (New)

My code can be divided into 6 parts, which are basic settings, inputs, error checking, calculating, outputs, and deletion. Here we will briefly go through each but focus on the forth: calculating.

(a) Basic Settings
By #include <fstream> and int main(int argc, char* argv[]), we are able to read data from the specified file and create another file to store the outputs.

(b) Inputs
In the beginning, we f-in the first line getting number “n2” to decide the data size.

second, we create an array called pairs with size of n2, to store the information of each pair of nodes using function “**GetInPuts**”.
For example, if the current input chord is “0 4”, we set pairs[4]=0 and pairs[0]=4. This array will later help us looking for the existing chord when given one end node.

(c) Error Checking
Since the last input is a single zero, we simply read it as a checking message to see if the data input properly using function “**CheckErr**”.

(d) Calculating

(1) Define a class “**MaxPlanner**” which contains public functions and private variables.
(2) private variables introduction:
・M[i][j]- records the number of Max Planner Subset from node i to node j.
・vector pairDetail- records the chords of Max Planner Subset. For each chord, only one of the node are recorded, and these nodes are stored in sequence.
(3) We will first calculate M[i][j] by function “**Compute_Num**”.
(4) Next we then find the pairDetail based on M[i][j] by topdown recurrence.

“**Compute_Chord**”.
This is the most importance advance compared to my old version. This progress saves lots of time looking for useless MPS chords of subproblems.

Compute_Num
The Max Planner Subset between node 0 and node (n2-1). The relation of M[i][j] and its subproblems can be classified as 4 categories. Let “K” be the pair node of node j, which means chord Kj exists.
(1) [i=j] there won’t be any chord between one single node, so M[i][j]=0.
(2) [K=i] chord ij exists means the M[i][j]= M[i][j-1]+1, the added “1” represents chord ij itself.
(3) [i<K<j] this case is relatively complicated since the chord Kj might intersect with some Max Planner Subset chords of M[i][j-1]. We have to compare 2 cases:
(3-1) We should take M[i][j-1] and give up chord Kj, or
(3-2) We take chord Kj, M[i][K], and M[K][j], giving up some chords of M[i][j-1] intersect Kj.
(4) [K>j] If chord Kj does not belong to interval [i,j], it makes no difference between M[i][j] and M[i][j-1]. —> M[i][j]= M[i][j-1].

Compute_Chord
(1) [i=j] do nothing.
(2) [K=i] add K into the pairDetail and compute **Compute_Chord(i+1, j-1)**.
(3) [i<K<j]:
(3-1) **Compute_Chord(i, j-1)**
(3-2) add K into the pairDetail and compute both **Compute_Chord(i, K-1) & Compute_Chord(K+1, j)**. In order to maintain pairDetail sequentially, we adjust the order of implementations.
(4) [K>j] add nothing and compute **Compute_Chord(i+1, j-1)**.

(e) Outputs

After calculating, we are going to put the result into the created file. First line is the number of Max Planner Subset, which is M[0][n2-1]. The following are pairs of nodes that belong to the chords of this Max Planner Subset.
Here we can easily use the “**PrintDetails**” and array “pairs” built in part(b) to get the corresponding node when one end node is given.

(f) Deletion

Those dynamic arrays we created previously need to be delete. “**~MaxPlanner()**” and “delete[] pairs; ” are used. This part will not affect any implementation.

（B）Time complexity Analysis (New)

Steps(a) = constant time C1.

Steps(b) = linear n time C2*n.

Steps(c) = constant time C3.

Steps(d) = $(2n-1)^2$ of table M
and O(n) of pairDetail value pushing $C4n^2$.

Steps(e) = linear O(n) time C5*n.

Steps(f) = n time C6*n.

Total = C1+ C2*n + C3 + $C4n^2$ + C5*n + C6*n = $\underline{\Theta(n^2).\ \#}$

（C）Reflection

————New version————
In this case, topdown method is quite useful. My basic concept was not changed but I adjust the way it runs, and it turns out to be great. In new version, I use “class” to record some inner variables so that some troubles (e.g. returning more than one type of data) are solved.
This is a challenging PA, but I am happy that I finally concur it.
————Old version————
I finished this PA according to the 9th problem of HW2, the solution of which I totally thought by myself. When the correct result came out, I felt super excited. Though there are still many technical problems such as makefile, fstream, etc that I am not familiar with, I’ve learnt a lot by gradually concur them during the process.

The time complexity of this algorithm is not ideal, especially when giving those chords is required. Instead of using 3D array, I am wondering if there is any good way to categorize the chords.

——————————Old version of my report——————————

（A）Code Design Concept

My code can be divided into 6 parts, which are basic settings, inputs, error checking, calculating, outputs, and deletion. Here we will briefly go through each but focus on the forth: calculating.

(g) Basic Settings
By #include <fstream> and int main(int argc, char* argv[]), we are able to read data from the specified file and create another file to store the outputs.

(h) Inputs
In the beginning, we f-in the first line getting number “n2” to decide the data size. second, we create an array called pairs with size of n2, to store the information of each pair of nodes.
For example, if the current input chord is “0 4”, we set pairs[4]=0 and pairs[0]=4. This array will later help us looking for the existing chord when given one end node.

(i) Error Checking
Since the last input is a single zero, we simply read it as a checking message to see if the data input properly.

(j) Calculating
Two of the most important arrays are M[i][j] and pairDetail[i][j][k].
M[i][j]- records the number of Max Planner Subset from node i to node j.
pairDetail[i][j][k]- records the chords of Max Planner Subset between node i to node j. For each chord, only one of the node are recorded, and these nodes are stored in sequence.

In dynamic programming, we define M[i][j] and pairDetail[i][j][k] as the subproblem of our goal:
The Max Planner Subset between node 0 and node (n2-1). The relation of M[i][j] and its subproblems can be classified as 4 categories. Let “K” be the pair node of node j, which means chord Kj exists.
(1) [i=j] there won’t be any chord between one single node, so M[i][j]=0.
(2) [K=i] chord ij exists means the M[i][j]= M[i][j-1]+1, the added “1” represents chord ij itself.
(3) [i<K<j] this case is relatively complicated since the chord Kj might intersect with some Max Planner Subset chords of M[i][j-1]. We have to compare 2 cases:
  (3-1) We should take M[i][j-1] and give up chord Kj, or
  (3-2) We take chord Kj, M[i][K], and M[K][j], giving up some chords of M[i][j-1] intersect Kj.
(4) [K>j] If chord Kj does not belong to interval [i,j], it makes no difference between M[i][j] and M[i][j-1]. —> M[i][j]= M[i][j-1].

Let’s start with the double “for loop”.
Suppose i always <=j, we start from j=i and gradually extend the interval until j=last node, then we can can ensure M[i][j-1] and boundary value M[i][i]=0 are always finished before M[i][j].
We start from i=last node and decrease i, then M[i][K] and M[K][j] (i<K<j) are always

finished before M[i][j].
The order of the for loops ensure no invalid access will occur.

As for array “pairDetail”, we record the Max Planner Subset chords of interval [i,j] based on interval [i,j-1] or interval [i,K] and interval [K,j], and add the last chord Kj if needed. This implementation automatically sorts the desired nodes from small to large.

(k) Outputs
After calculating, we are going to put the result into the created file. First line is the number of Max Planner Subset, which is M[0][n2-1]. The following are pairs of nodes that belong to the chords of this Max Planner Subset.
Here we can easily use the array “pairs” built in part(b) to get the corresponding node when one end node is given.

(l) Deletion
Those dynamic arrays we created previously need to be delete. This part will not affect any implementation.

（B）Time complexity Analysis

Steps(g) = constant time C1.

Steps(h) = linear n time C2*n.

Steps(i) = constant time C3.

Steps(j) = $(2n-1)^2$ of table and O(n) of pairDetail value passing C4n^3.

Steps(k) = linear O(n) time C5*n.

Steps(l) = n^2 time C6*n^2.

Total = C1+ C2*n + C3 + C4n^3 + C5*n + C6*n^2 = $\underline{O(n^3)}$. #