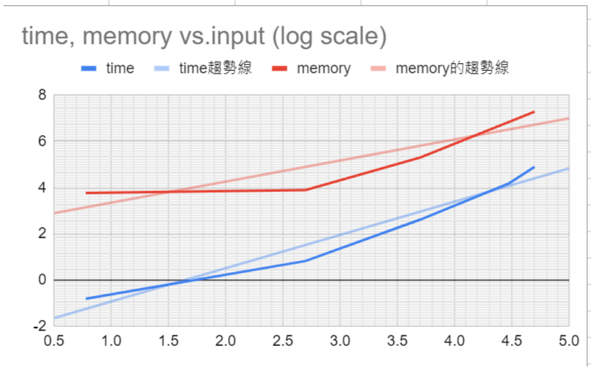
1. **Data Structure:**
    1) **array (1D,2D)** : chord (records the input chords)
       calculated (records the max number of chords in max planar(i,j) subset)
    2) **int** : total_num (the input 2n, the total number of vertices on the circle)
       mps_num (number of chords in the maximum planar subset in the input circle of n chords)
       first_node, second node (the two end points of a node)
       trash (a single '0' in the input line signifies the end of input, which will not be used)
    3) **bool** : opt (tells the program if we want to record this chord to the final answer)
    4) **set and pair** : max_chord_pair (records the chords that made the mps)
    5) **int function**: MAX_CHORD (i,j,opt) (returns the number of chords in mps(i,j))
2. **Algorithm: dp**
    1) **find the number of chords**: use top-down recursion to calculate the number of chords. start from MAX_CHORD(0,total_num-1,true), I will elaborate the 'true' (opt) later. If M(i,j) has been calculated before, then it will just return the number. This will largely reduce the run time since it doesn't need to run the recursion too many times.

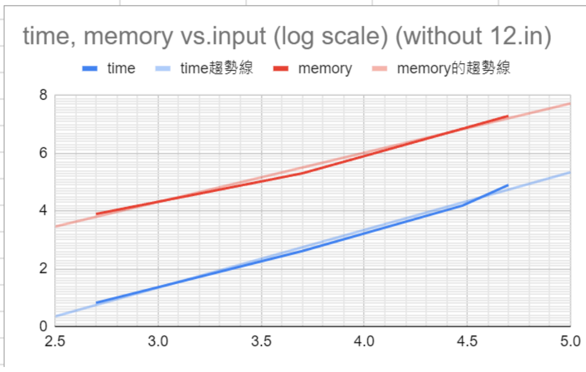
$$M(i,j)=\begin{cases} M(i,j-1) & , \text{chord } kj \in C,\ k \notin [i,j] \\ \max(M(i,k-1)+M(k+1,j-1)+1,\ M(i,j-1)) & , \text{chord } kj \in C,\ k \in [i,j] \\ M(i+1,j-1)+1 & , \text{chord } ij \in C \end{cases}$$

    2) **find the chords**: First, max_chord_pair will record the chords as pairs of nodes and save them in the set, and using set can avoid saving the same chords again. As for the bool 'opt', it decides whether we want to record this chord or not. The correct output only wants us to save the chords forming the mps, so when doing the recursion, the chords will be record when it's started from MAX_CHORD (0,total_num-1,true), those MAX_CHORD (i,j,false) will only be used for returning calculated[i][j].
    3) **for the data structures used**: As mentioned in the '1.' part. Additional: max_chord_pair will only contain the chords of mps(0,total_num-1).
3. **Time complexity analysis:**
    1) **theoretically**: both time and memory should be approximately O(n^2)
    2) **reality** :all cases: time slope= 1.44, memory slope=0.913
       without 12.in: time slope=2, memory slope=1.71

(these are the result from eda union)

case	12.in	1000.in	10000.in	60000.in	100000.in
time(ms)	0.152	6.655	410.231	15471.9	79694.1
memory(kb)	5904	7884	201580	7098552	19672200
number of chords in mps	3	52	176	1	566
input size (log scale)	0.7781512504	2.698970004	3.698970004	4.477121255	4.698970004
time (log scale)	-0.8181564121	0.8231480598	2.613028476	4.18954365	4.90142617
memory (log scale)	3.771146349	3.896746616	5.304447441	6.851169768	7.293852931
	all cases		without 12.in		
slope of log(time) vs. log(input)	1.44		2		
slope of log(memory) vs. log(input	0.913		1.71		

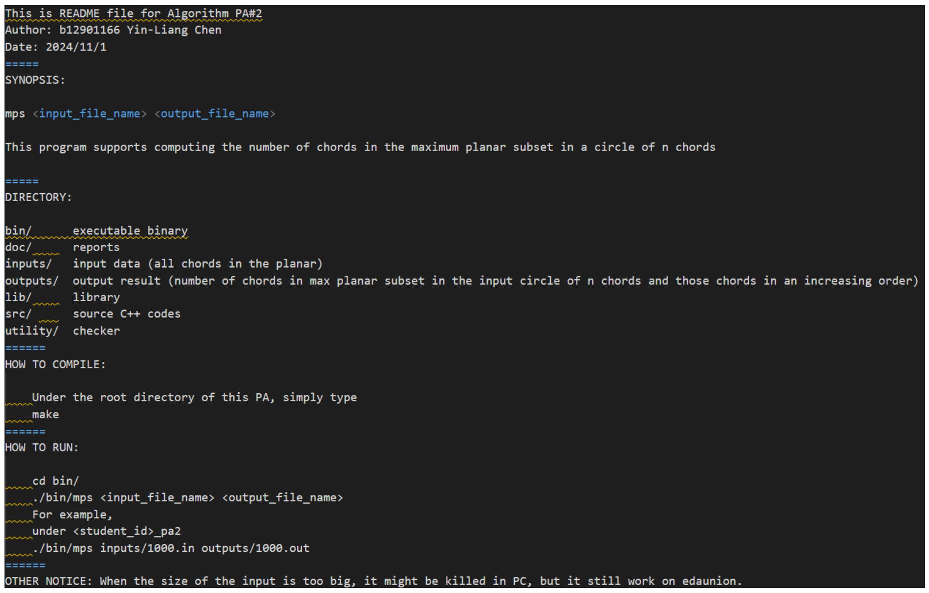




3) **possible reason**:
For the memory part: For calculated[][], I only assigned its size as (total_num^2)/2; and for chord[], its size=total_num.
For the time part: When doing dp, the runtime might differ from the input chords,which will change the number of recurrences and the runtime of each recurrence. Also, when doing fout, it also depends on the size of max_chord_pair, and of course, it will be different depending on the input chords. (Although it might only cause a minor difference.)
Others: Reasons why with and without 12.case have different time slope and memory slope might because 12 is too small! when doing most of the operation in other cases, n^2 is way more bigger than O(1)&O(n) operation both in the memory and time part. However, 12^2=144, which is quite close to O(1)&O(n), hence, those O(1)&O(n) operations might have a big influence on both the runtime and memory, causing the slope to change from 2 to 1.44 and 1.71 to 0.913 in time and memory respectively.

4) **time analysis**: read the input file: O(n)
construct 2D array calculated[][]: O(n^2)
write the output: O(n)
do recursion(dp with recording): O(n^2)
total runtime:O(n^2)

4. README



```
This is README file for Algorithm PA#2
Author: b12901166 Yin-Liang Chen
Date: 2024/11/1
=====
SYNOPSIS:

mps <input_file_name> <output_file_name>

This program supports computing the number of chords in the maximum planar subset in a circle of n chords

=====
DIRECTORY:

bin/      executable binary
doc/      reports
inputs/   input data (all chords in the planar)
outputs/  output result (number of chords in max planar subset in the input circle of n chords and those chords in an increasing order)
lib/      library
src/      source C++ codes
utility/  checker
======
HOW TO COMPILE:

    Under the root directory of this PA, simply type
    make
======
HOW TO RUN:

    cd bin/
    ./bin/mps <input_file_name> <output_file_name>
    For example,
    under <student_id>_pa2
    ./bin/mps inputs/1000.in outputs/1000.out
======
OTHER NOTICE: When the size of the input is too big, it might be killed in PC, but it still work on edaunion.
```

5. Other findings : I have put a /backup code in the file because when I'm coding, my PC always show Killed in 100000.in, so I consider if it's because my calculated[][] is too big (size of 100000*100000), so I write some other ways to try to avoid this thing happening. For example, backup.cpp is my first draft, with calculated[][] size=total_num^2, but it runs too fast and be killed when 100000; BACKUP2.cpp is for calculated using map, but it will not operate normally when input size is 60000. As for test.cpp, I use it to find out if my MAKEFILE operates normally. Finally, I turn the calculated[][] into size of (total_num^2)/2, which can reduce its size and shorten the time for initializing. It still can be used since for chord(i,j), i must be smaller than j, so when storing the value, the table of calculated[i][j] with i>j will obviously not be used.

6. Others : Since it's quite hard to find out whether the output chords are correct, I wrote a program called check_answer in ./utilities to find out if there are any two chords that will intersect.