National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2025

Handout #17
October 11, 2025
Yao-Wen Chang & Cheng-Yu Chiang

Programming Assignment #2 (due online at 12:00 noon, November 2, 2025)

Online Resources with five public input cases will be provided at the NTU COOL. (Notice that more hidden test cases will be used for the final test.):
https://cool.ntu.edu.tw/courses/53535/assignments/326023 (Prof. Yao-Wen Chang's class)
https://cool.ntu.edu.tw/courses/52206/assignments/343205 (Prof. James Chien-Mo Li's class)

Problem: Maximum Planar Subset

Given is a set C of n chords of a circle (see Figure 1 (a)). We assume that no two chords of C share an endpoint. Number the endpoints of these chords from 0 to $2n-1$, clockwise around the circle (see Figure 1 (c)). Let $M(i,j), i \leq j$, denote the number of chords in the maximum *planar subset* (*i.e.*, no two chords overlap each other in the subset) in the region formed by the chord $\overline{ij}$ and the arc between the endpoints i and j (see Figure 1 (d)). As the example shown in Figure 1 (a), $M(2,7) = 1$, M(3, 3) = 0, and $M(0,11) = 3$. You are asked to write a program that computes the number of chords in the maximum planar subset in a circle of n chords, *i.e.*, compute $M(0,2n-1)$, and reports the details of each chords, as shown in Figure 1 (b).

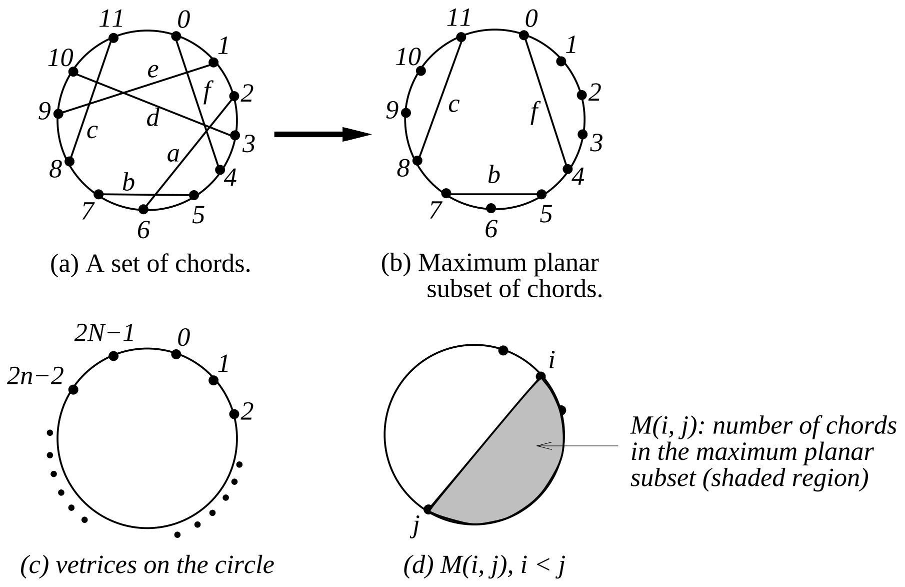
11
0
1
2
3
4
5
6
7
8
9
10
a
b
c
d
e
f
(a) A set of chords.
(b) Maximum planar subset of chords.
2N−1
2n−2
(c) vetrices on the circle
i
j
M(i, j): number of chords in the maximum planar subset (shaded region)
(d) M(i, j), i < j

Figure 1: Maximum planar subset.

Input

The input consists of an integer $2n$, $1 \leq n \leq 90,000$, denoting the number of vertices on a circle, followed by n lines, each containing two integers a and b ($0 \leq a, b \leq 2n-1$), denoting two endpoints of a chord. A single "0" (zero) in the input line signifies the end of input.

Output

The output file reports the number of chords in the maximum planar subset in the input circle of n chords, followed by a list of the two endpoints for each resulting chord in the maximum planar subset (sorted by the first endpoint in the increasing order).

Here is an input/output example (see Table 1):

Sample Input	Sample Output
12	3
0 4	0 4
1 9	5 7
2 6	8 11
3 10	
5 7	
8 11	
0	

Table 1: Input/output example.

Command-line Parameter:
The executable binary must be named as **"mps"** and use the following command format.

```
./mps <input_file_name> <output_file_name>
```

For example, if you would like to run your binary for the input file `12.in` and generate a solution named `12.out`, the command is as follows:

```
./mps 12.in 12.out
```

Compilation & Execution:
We expect that your code can be compiled and run as follows. Type the following commands under <**student_id**>**_pa2/** directory,

```
make
./bin/mps inputs/<input_file_name> outputs/<output_file_name>
```

Optional Method Check (for report only):
If the optional flag "`--method=td|bu`" is provided, the program must run the requested DP strategy and print the first three and the last unique first-visited subproblems (i, j) in the order they were first visited, using the format:

$$(i_1, j_1), (i_2, j_2), (i_3, j_3), \ldots, (i_{last}, j_{last})$$

Definition of "visit":

- Top-down (td): when (i, j) is computed for the first time (memo miss), with $i \leq j$.
- Bottom-up (bu): when $M(i, j)$ is first assigned in the DP table (typically $i < j$).

Examples (on the sample input):

`./mps --method=bu 12.in 12.out`

`Print` $\rightarrow$ $(0,1)$, $(1,2)$, $(2,3)$, $\ldots$, $(0,11)$

`./mps --method=td 12.in 12.out`

`Print` $\rightarrow$ $(0,11)$, $(0,7)$, $(0,4)$, $\ldots$, $(0,8)$

Required Files:
You need to create a directory named <**student_id**>**_pa2/** (*e.g.* b07901000_pa2/) **(the student ID should start with a lowercase letter)** which must contain the following documents:

- A directory named **src/** containing your source codes (*e.g.* `maxPlanarSubset.cpp`): only `*.h`, `*.hpp`, `*.c`, `*.cpp` are allowed in src/, and no directories are allowed in src/;
- A directory named **bin/** containing your executable binary named **mps**;
- A directory named **doc/** containing your report;

- A makefile named **makefile** that produces an executable binary from your source codes by simply typing "make": the binary should be generated under the directory <student_id>_pa2/bin/;
- A text readme file named **README** describing how to compile and run your program.
- A report named **report.pdf** on the data structures used in your program and your findings in this programming assignment. Please implement both DP methods—Top-down and Bottom-up, and include a concise discussion of the two.

We will use our own test cases, so you do NOT need to submit the input files. Nevertheless, you should have at least the following items in your *.tgz file.

```
src/<all your source code>
bin/mps
doc/report.pdf
makefile
README
```

Submission Requirements:

1. Your program must be compilable and executable on EDA union servers.
2. The runtime limit for each test case is **10 minutes**.
3. Please use the following command to compress your directory into a *.tgz* file:

   ```
   tar zcvf <filename>.tgz <your directory>
   ```

   The submission file should be named as <**student_id**>**_pa2.tgz** (*e.g.* b07901000_pa2.tgz). For example, if your student ID is *b07901000*, then you should use the command below.

   ```
   tar zcvf b07901000_pa2.tgz b07901000_pa2/
   ```

4. Please submit your *.tgz* file to the NTU COOL system before **1pm, November 5, 2023 (Sunday)**.
5. You are required to run the `checkSubmitPA2.sh` script to check if your *.tgz* submission file is correct. Suppose you are in the same level as the PA1 directory,

   ```
   bash checkSubmitPA2.sh <your submission>
   ```

   For example,

   ```
   bash checkSubmitPA2.sh b07901000_pa2.tgz
   ```

Language/Platform:

1. Language: C or C++.
2. Platform: Unix/Linux

Evaluation:
The individual score per test case is determined by the correctness of the output result as well as the file format. Bonus will be given to most efficient programs. (Note that there are more hidden test cases for evaluating your program.)