PA2 Report

B07901032, Kuo Hao Wang

I. Data structure, variables

- `static unsigned short mask[16]`

  This is bitmask used in bit operation that replaced shift operation.

- `class A`

  This class is designed to deal with the shortage of `short` type. Among all test data, there is possibility of the amount of chords getting over 2^{16}-1 which is the upper bound of `short` type.

  In this class, we use an array of 17 `short` integers to store 16 17-bit integers. A 17-bit integer is divided into 1bit and a short, and we collect those 1bit and put it into `i[16]`. With this approach, j-th 17-bit integer would be `i[j]` plus 65536 if the related bit is set in `i[16]`.

  - `A()`

    This constructor is used to set all 17-bit integer to 2^{17}-1 initially.

  - `unsigned int operator() (short j)`

    This is the "get" function which combines `i[j]` and `i[16]`.

  - `void operator() (short j, int num)`

    This is the "set" function which divides num and store it into `i[j]` and `i[16]`.

  - `unsigned short i[17]`

- `vector<int> E`

  This vector stores the chord information from input file.

- `vector<int> chosen`

  This vector stores the chosen chord information and is used when outputting.

- `vector<vector<A> > M`

  This vector stores the data from the memoization of the recursion.

- `int n, i, j`

  Integer `n` stores the data size while `i`, `j` are indexes used to control data from in `E`, `M`, `chosen`.

- `fstream fin, fout`

  These `stream` objects are used in reading out the input file and writing to the output file.

- `sstream ss`

  This `stream` object is used in formatting output.

II. Programming Flow

Read in the input file and store to `n`, `E`
⟶ calculate required information and store to `M` recursively by `m(0, n-1)`
⟶ find selected chords and store to chosen recursively by `_m(0, n-1)`
⟶ write to the output file

III. Experiments

I generated my own random test cases for efficiency comparison.

Bottom-up

I tried the bottom-up method with the interval type stored first, and then I tried the bottom-up method without the interval type stored and just calculate the type while finding selected chords recursively. The first method uses more memory, whereas it has worse performance because the total number of accessing the vector is $O(n^2)$ while the total number of calculating the type while finding is O(n).

Following are some test:

- The memory is big because the size of M is n^2. I changed it to n(n-1)/2 while doing top-down.
- I listed user time because it is the real time the program is executed without being clogged

	Type stored			Without type stored		
2n	Real time (s)	User time (s)	Max mem (byte)	Real time (s)	User time (s)	Max mem (byte)
10	0	0	1,436	0	0	1,440
20	0	0	1,440	0	0	1,444
50	0	0	1,464	0	0	1,452
100	0	0	1,528	0	0	1,480
200	0	0	1,748	0	0	1,588
500	0	0	3,416	0	0	2,424
1000	0.01	0.01	9,324	0	0	5,376
2000	0.08	0.06	32,848	0.02	0.01	17,144
5000	0.5	0.41	197,168	0.25	0.19	99,316
10000	2.04	1.67	783,540	1.14	0.97	392,528
20000	8.4	7.08	3,128,156	4.97	4.32	1,564,872
50000	436.44	66.52	15,589,640	35.88	31.55	9,803,004

Top-down

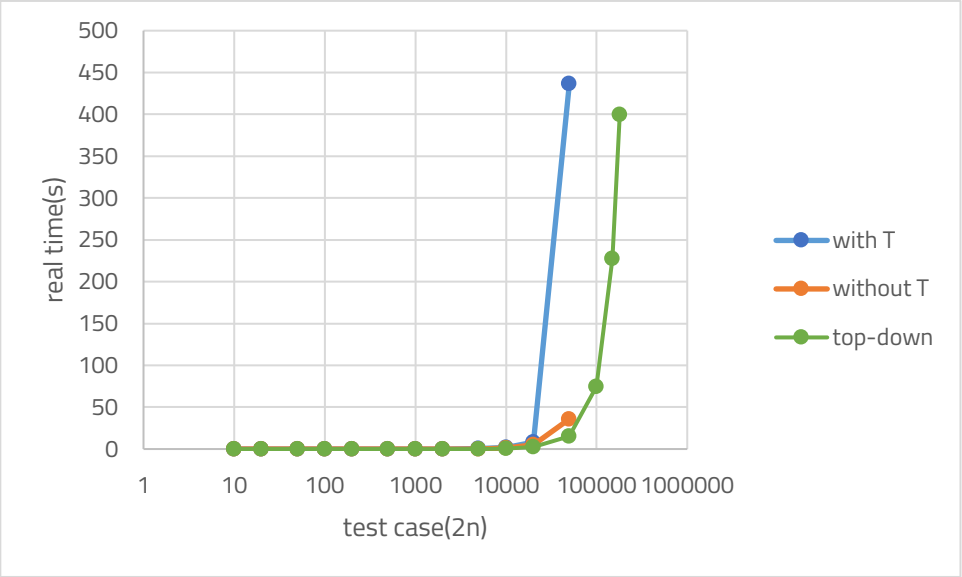
I tried and found that top-down is actually faster.

2n	Real time (s)	User time (s)	Max mem (byte)
10	0	0	3,416
20	0	0	3,416
50	0	0	3,396
100	0	0	3,428
200	0	0	3,496
500	0	0	3,756
1000	0.01	0.01	4,588
2000	0.03	0.01	7,852
5000	0.13	0.12	30,036
10000	0.54	0.48	108,404
20000	2.21	2.03	420,936
50000	15.02	13.87	2,604,208
100000	74.68	69.56	10,470,252
150000	227.56	216.04	23,501,688
180000	399.69	375.08	33,777,956

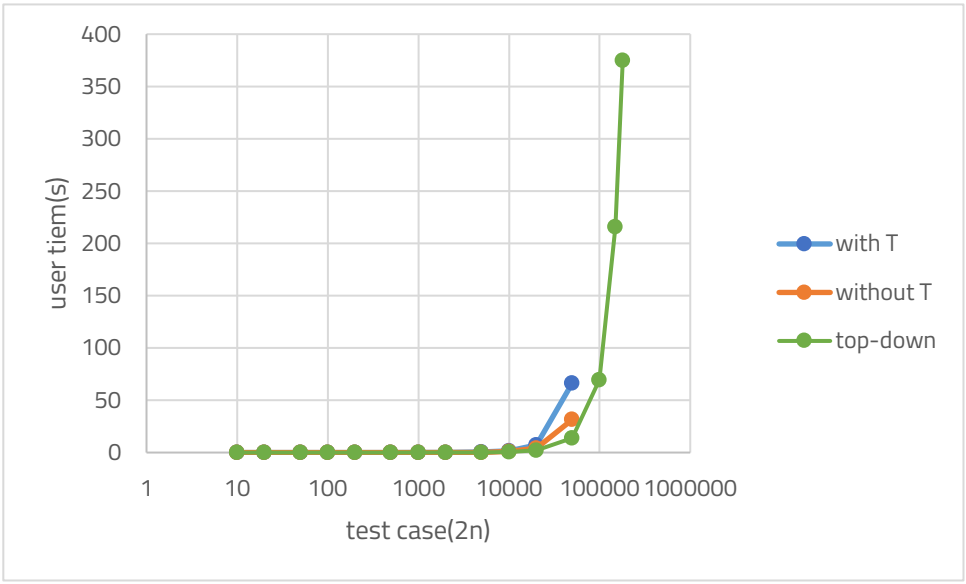
Graph and Finding

It is easily to see that these approaches are all $O(n^2)$ in time and space. One of my friends had found some algorithm that take O(n) in space and O(an) in time but I'm not confident to get myself into it.

Real time – test case



User time – test case



Maximum memory used – test case