2017-1 Computer Algorithms Homework #3

Dae-Ki Kang

March 22, 2017

(Deadline : March 31)

1. Using your favorite computer programming language (but C/C++, Java, Python, C# recommended), write programs that calculate the following.

   Mina is trying to figure out the number of distinct ways to pick marbles on the floor.

   For example, suppose that if there are three marbles and Mina can pick either one marble or two marbles at one time, then there are three different ways to pick marbles on the floor: (`one`, `one`, `one`), (`one`, `two`), and (`two`, `one`).

   If there are 4 marbles and Mina can pick upto 3 marbles at one time, then there are 7 different ways to pick marbles: (1,1,1,1), (1,1,2), (1,2,1), (2,1,1), (2,2), (1,3), and (3,1).

   As input, your program will get the total number of marbles on the floor ($1 \leq n \leq 2,000,000,000$) and the maximum number of marbles that Mina can pick at one time ($1 \leq m \leq 10$).

   As output, your program will print the remainder of the total different ways to pick marbles (r) divided by 65535 (i.e. r modulo 65535). For example, if the result is 65540, your program will print 5.

   Note that for any input, your program has to print the result and terminate in **2 seconds** on Intel i5 or similar CPU's. If your program takes more time, it will be considered failure.

   Also, please note that the number of marbles (n) can be big enough to **2,000,000,000**. (So, Mina will be very busy.)

```
Input
4
3
Output
7

Input
2000000000
2
Output
51916
```

2. Using your favorite computer programming language (but C/C++, Java, Python, C# recommended), write programs that calculate the following.

   Given one dimensional array of which the length is N, write a program that find a sum of contiguous subarray such that the sum is the maximum.

   For example, given the following array.

   ```
   33 36 -73 15 43 -17 36 -28 -1 21
   ```

   The numbers surrounded by parenthesis are contiguous subarrays, followed by their sums.

   ```
   33 36 (-73 15 43 -17) 36 -28 -1 21 --> -32
   33 36 -73 15 43 -17 36 -28 (-1 21) --> 20
   (33 36 -73 15 43 -17) 36 -28 -1 21 --> 37
   33 36 -73 15 43 -17 (36) -28 -1 21 --> 36
   ```

   Given the input $1 \leq N \leq 1,000,000$ and the numbers, the program will print the largest sum in less than **one second**.

   If you feel that the result might cause overflow, return the result number modulo 65535.

   ```
   Input
   8
   -2 -3 4 -1 -2 1 5 3
   Output
   7

   Input
   10
   33 36 -73 15 43 -17 36 -28 -1 21
   Output
   77
   ```

   Further questions:

   (a) Modify your program so that it can print the interval as well as the largest sum.

   ```
   Input
   10
   33 36 -73 15 43 -17 36 -28 -1 21
   Output
   (4,7):77
   ```

   (b) Modify your program so that it can handle two-dimensional array ($N \times N$) instead of one dimensional array.

   ```
   Input
   4
   3 -3 -2 -1
   2 6 -5 1
   -3 -5 2 4
   2 4 0 -2
   Output
   (1,1)-(2,2):8
   ```

3. Using your favorite computer programming language (but C/C++, Java, Python, C# recommended), write programs that calculate the following.

   There are $1 \leq N \leq 10^9$ candies, laid linearly on the table. You would like to give the candies to Mina, Sana, Momo, and Tzuyu.

   From left to right, you mark each candy to whom you are going to give the candy. For example, if there are 4 candies, one possible way is `(Mina, Sana, Mina, Momo)`.

   Your goal is to calculate the total number of possible ways such that Mina has even number of candies and Sana has even number of candies.

   For example, if $N = 1$, the output is 2 (`(Momo)` and `(Tzuyu)`).

   For example, if $N = 2$, the output is 6 (`(Mina,Mina)`, `(Sana,Sana)`, `(Momo,Momo)`, `(Tzuyu,Tzuyu)`, `(Momo,Tzuyu)`, and `(Tzuyu,Momo)`).

   Note that you don't have to print all the possible ways, instead you simply print the number of all the possible ways modulo 65535 (i.e. the remainder of dividing the number of all the possible ways by 65535).

   Note that for any input, your program has to print the result and terminate in **2 seconds** on Intel i5 or similar CPU's. If your program takes more time, it will be considered failure.

   Also, please note that the number of candies (N) can be big enough to **1,000,000,000**.

```
Input
1
Output
2

Input
2
Output
6
```

4. Using your favorite computer programming language (but C/C++, Java, Python, C# recommended), write programs that calculate the following.

   Your program input $2 \leq n \leq 2000000000$, and print the last five digits of $\left\lfloor \left(4+\sqrt{13}\right)^n \right\rfloor$.

   If the number to print has less than five digits, print the number as is.

   Note that for any input, your program has to print the result and terminate in **2 seconds** on Intel i5 or similar CPU's. If your program takes more time, it will be considered failure.

   Also, please note that the input (n) can be big enough to **2,000,000,000**.

```
Input
2
Output
57

Input
30
Output
42185

Input
100
Output
68625

Input
10000
Output
71873
```

Table 1: Computation table

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

5. Consider the alphabet $\Sigma = \{a, b, c\}$. The elements of Σ have the computation table such as μ.

   Thus, for example, $ab = b$, $ba = c$, $cc = c$, and $a(b((bc)a)) = a$. Note that you have different results according to the way of parenthesizing. For example, $(aa)c = a$, but $a(ac) = b$.

   For a given string $X = x_1, x_2, x_3, x_4, \ldots, x_n (x_i \in \Sigma, i = 1, 2, \ldots, n)$, find an efficient algorithm (polynomial in $|\Sigma|$ and n) to decide whether or not, it is possible to parenthesize the string X such that the value of the resulting expression is a.

   For example, your algorithm should return "yes" if the given string is "bbbba", because $(b(bb))(ba) = a$, but your algorithm should return "no" if the given string is "bbb" because both $(bb)b \neq a$ and $b(bb) \neq a$.

   What is the time complexity of your algorithm?

6. Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the top leftmost square $(1, 1)$ to the bottom rightmost square (n, n) according to the following rule. At each step, you may move the checker to the one of the three squares:

   (a) the square that is one to the right (that is $\rightarrow$, but only if the checker is not already in the rightmost column),

   (b) the square that is one down (that is $\downarrow$, but only if the checker is not already in the bottom row),

   (c) the square that is one down and one to the right (that is $\searrow$, but only if the checker is not already in the bottom rightmost square).

   Each time, you move from square x to square y, you earn $p(x, y)$ points. Give an algorithm that finds out the path (that is, the set of legal moves) from the top lefrmost square $(1, 1)$ to the bottom rightmost square (n, n) where the sum of all points in the path is the maximum.

7. Given three strings x, y and z and a text T, you want to find whether there is an occurrence of x, y and z interwoven in T. An interwoven occurrence is a substring a ($= a_1 a_2 \ldots a_{|x|+|y|+|z|}$) of T whose length is the sum of the lengths of x, y, and z, such that for some sequences $i_1 < i_2 < \cdots < i_{|x|}$, $j_1 < j_2 < \cdots < j_{|y|}$ and $k_1 < k_2 < \cdots < k_{|z|}$, $x = a_{i_1} a_{i_2} \ldots a_{i_{|x|}}$, $y = a_{j_1} a_{j_2} \ldots a_{j_{|y|}}$, $z = a_{k_1} a_{k_2} \ldots a_{k_{|z|}}$ and the intersection of the sets $\{i_1, i_2, \ldots, i_{|x|}\}$, $\{j_1, j_2, \ldots, j_{|y|}\}$ and $\{k_1, k_2, \ldots, k_{|z|}\}$ is empty. For example, the strings *abac*, *bbc*, and *abc* occur interwoven in *caabcbbabccdw*. Give an efficient algorithm for the interwoven patterns problem.

8. Consider $n \times n$ grid. Each cell in the grid has a number. Starting from the top left corner cell (1,1) of the grid, you can go in down direction or right direction. Note that you have to pass exactly as many cells as the number in your current cell. And your final goal is to see if, following this manner, you can reach the bottom right corner cell (n,n) of the grid.

   Present an algorithm for this goal. Your algorithm will print "true" if you can reach (n,n) from (1,1), and "false" otherwise.

   For example, if you have the following grid, you can go (right 2, down 1, right 1, down 2, down 1, right 3, down 2) to reach the goal 'G'. And your algorithm will print "true".

Table 2: There is a path from the start to the end.

2	5	**1**	6	1	4	1
6	1	**1**	**2**	2	9	3
7	2	3	2	1	3	1
1	1	3	**1**	7	1	2
4	1	2	**3**	4	1	**2**
3	3	1	2	3	4	1
1	5	2	9	4	7	**G**

   However, for the following grid, your algorithm should print "false".

Table 3: No paths in this grid.

2	5	1	6	1	4	1
6	1	1	2	2	9	3
7	2	3	2	1	3	1
1	1	3	1	7	1	2
4	1	2	3	4	1	**3**
3	3	1	2	3	4	1
1	5	2	9	4	7	G

   Also, your algorithm prints "false" for the extreme case as below:

Table 4: Again, no paths in this grid.

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	2
1	1	1	1	1	2	G

9. If you remove more than or equal to zero numbers in the sequence, the resulting one is a 'subsequence'. For example, if '4 3 7 6 9' is a sequence, then '4 7 6' is a subsequence. From the definition, '4 3 7 6 9' is a subsequence of itself.

   If every number in a subsequence is unique and each number in the subsequence is smaller than its next number on the right, then we call it 'increasing subsequence'. For example, if '4 3 7 6 9' is a sequence, then '3 7' is an increasing subsequence. There are many other increasing subsequences and, for example, '6 9' is one of them. Note that numbers in an increasing subsequence are sorted.

   We can see that '4 7 9' is the 'longest increasing subsequence' of '4 3 7 6 9', because it has three numbers and there is no increasing subsequence with four numbers. Note that there are other longest increasing subsequences such as '4 6 9', '3 7 9', and '3 6 9'.

   Consider one way to create a new sequence from two sequences. From two sequences A and B, you get two increasing subsequences SA and SB, respectively. Note that SA and SB are increasing subsequences, not necessarily longest increasing subsequences. Then you run 'mergesort' of SA and SB to generate a new sequence (let 'mergesort sequence'). For example, if A='1 9 4' and B='3 4 7'. '1 3 7 9' is a mergesort sequence of A and B, with SA='1 9' and SB='3 7'.

   Okay, now the goal of your algorithm is to get the length of 'longest mergesort sequence'. For example, if A='1 9 4' and B='3 4 7'. '1 3 4 7 9' is the longest mergesort sequence of A and B with SA='1 9' and SB='3 4 7'. In this case your algorithm should return 5, because '1 3 4 7 9' has five numbers.

   The followings are example runs:

```
A=1 2 4
B=3 4 7
Output=5

A=1 2 3
B=4 5 6
Output=6

A=10 20 30 1 2
B=10 20 30
Output=5
```