CS4050 Design and Analysis of Algorithms

Assignment Problem Set

3_1) Problem Statement

There are **N** boxes, numbered 1 through **N**. For each i, box i contains exactly i candies and one pebble. Thus, there are exactly i+1 objects in box i. We are now going to create a collection of **N** objects using the following simple procedure: from each box, we will draw one object uniformly at random. Let p be the probability that our collection will contain exactly **K** candies. You are given the ints **N**, **K**, and **MOD**. Return the value (p * (**N**+1)!) modulo **MOD**.

Definition

Class:	CandyDrawing
Method:	findProbability
Parameters:	int, int, int
Returns:	int

Method signature: int findProbability(int N, int K, int MOD)

(be sure your method is public)

Constraints

- N will be bewteen 1 and 1,000,000,000 (inclusive).
- K will be between 0 and 2,000 (inclusive).
- MOD will be between 1,000,000,000 and 2,000,000,000 (inclusive).
- MOD will be a prime.

Examples

1)

2
1
1000000007

Returns: 3

We have two boxes: box 1 with a candy and a pebble, and box 2 with two candies and a pebble. We are looking for the probability of drawing exactly one candy. This can happen in two different ways: either we draw the candy from box 1 and the pebble from box 2, or vice versa. The probability of the first way is (1/2)*(1/3) = 1/6. The probability of the second way is (1/2)*(2/3) = 1/3. Thus, the total probability is p = 1/6 + 1/3 = 1/2. We have p * (N+1)! = p*6 = 3, therefore the answer is (3 mod 1,000,000,007) = 3.

2)

3
2
1000000007

Returns: 11

3)

10
4
1000000007

Returns: 157773

4)

1000000000
1000
1000000009

Returns: 629516825

3_2) Problem Statement

Cat Snuke has a tree with N vertices. The vertices are labeled 0 through N-1. Snuke is able to modify the tree by performing a special operation. The operation consists of two steps. In the first step, he selects and removes an arbitrary edge of the tree. This divides the tree into two connected components. In the second step, he adds an arbitrary edge that connects those two components back into one tree.
You are given a int[] **p** that describes a tree, and an int **K**. The int[] **p** contains N-1 elements. For each i, the vertex **p**[i] and the vertex (i+1) are connected by an edge. Note that the initial labeling of the tree is such that for each i, **p**[i] is less than (i+1).

Snuke currently has the tree described by **p**. He may now modify it by repeatedly performing the operation described above. The number of operations Snuke will perform will be between 0 and **K**, inclusive. Two trees are considered different if there are vertices p and q such that one of the trees contains the edge p-q but the other one does not. Return the number of different trees Snuke can produce, modulo 1,000,000,007.

Definition

Class:	TreeDistance
Method:	countTrees
Parameters:	int[], int
Returns:	int
Method signature:	int countTrees(int[] p, int K)
(be sure your method is public)	

Constraints

- **p** will contain between 1 and 49 elements, inclusive. (This means that N will be between 2 and 50, inclusive.)
- For each valid i, **p**[i] will be between 0 and i, inclusive.
- **K** will be between 0 and 50, inclusive.

Examples

1)

{0, 0}
1
Returns: 3

There are three different trees for N=3. The one described by this **p** contains the edges 0-1 and 0-2. Snuke can turn it into either of the other two trees in a single operation.

2)

{0, 1, 2, 2, 0}
1

Returns: 28

Some of the trees for N=6 cannot be obtained from the given one in a single operation.

3)

{0, 1, 2, 2, 0}
2

Returns: 222

In two operations Snuke can produce some more trees, but not all of them. For example, Snuke would need three operations to produce the tree with the edges 0-1, 0-2, 0-3, 0-4, and 0-5.

4)

{0, 1, 2, 2, 0}
50

Returns: 1296

In 50 operations Snuke can produce all trees.

5)

{0, 1, 2, 2, 0}
0

Returns: 1

As no modifications are allowed, there is only one tree Snuke can produce in this test case.

6)

{0, 1, 0, 3, 3, 4, 4, 5, 6, 8, 3, 1, 12, 12, 13, 10, 4, 8, 13, 17, 2, 10, 12, 20, 2, 14, 17, 19, 15, 0, 22, 15, 3, 8, 3, 17, 27, 2, 12, 38, 37, 4, 40, 29, 9, 22, 43, 32, 37}
1

Returns: 7124

7)

{0, 0, 0, 0, 2, 3, 1, 2, 3, 7, 3, 10, 8, 8, 9, 1, 2, 0, 7, 17, 19, 2, 17, 2, 0,
6, 4, 9, 12, 14, 8, 12, 10, 30, 20, 30, 8, 36, 28, 22, 8, 2, 2, 13, 26, 14, 46, 6, 25}
10

Returns: 310259667

3_3)Problem Statement

Manao has a rectangular board divided into **n** times **m** cells. The rows of the board are numbered from 1 to **n** and the columns are numbered from 1 to **m**. The cell in row i, column j is referred to as (i, j). Each cell contains an uppercase letter of the English alphabet.

Having such a board, Manao likes to traverse it. The traversal always starts in the cell (1, 1). In each step of the traversal Manao moves either one cell down or one cell to the right. That is, from any cell (x, y) Manao will move either to (x+1, y) or to (x, y+1). The traversal always ends in the cell (**n**, **m**). During the traversal Manao records the letters in the visited cells (including the first and the last cell). The obtained string is called a string path for the given board.

You are given the ints **n** and **m**, and two distinct Strings **A** and **B**. Manao claims that he performed two different traversals of his **n** x **m** board and obtained the string paths **A** and **B**. Compute the number of different boards for which this is possible. Return this number modulo 1,000,000,009.

Definition

Class:	StringPath
Method:	countBoards
Parameters:	int, int, String, String
Returns:	int
Method signature:	int countBoards(int n, int m, String A, String B)
(be sure your method is public)	

Constraints

- **n** will be between 1 and 8, inclusive.
- **m** will be between 1 and 8, inclusive.
- **A** and **B** will be exactly **n**+**m**-1 characters long.
- **A** and **B** will consist of uppercase letters ('A'-'Z') only.
- **A** and **B** will be distinct.

Examples

1)

```
2
2
"ABC"
"ADC"
```

Returns: 2

The two possible boards are:

```
AB  AD
```

DC BC

2)

2
2
"ABC"
"ABD"
Returns: 0

It is impossible for two string paths to have a different last letter.

3)

3
4
"ABCDDE"
"ACCBDE"
Returns: 1899302

4)

8
8
"ZZZZZZZZZZZZZZZ"
"ZABCDEFGHIJKLMZ"
Returns: 120390576

3_4) Problem Statement

The Fibonacci sequence is defined as follows:

- F[0] = 0
- F[1] = 1
- for each i >= 2: F[i] = F[i-1] + F[i-2]

Thus, the Fibonacci sequence starts as follows: 0, 1, 1, 2, 3, 5, 8, 13, ... The elements of the Fibonacci sequence are called Fibonacci numbers.

Fibonacci base is a positional numeral system. The only two allowed digits are 0 and 1. The weights assigned to positions in the number are all *distinct positive* Fibonacci numbers, in order. For example, in Fibonacci base the sequence of digits 1101 represents the number 1*5 + 1*3 + 0*2 + 1*1 = 9. Some numbers have more than one representation in Fibonacci base. For example, we can also represent 9 as 10001, because 1*8 + 0*5 + 0*3 + 0*2 + 1*1 = 9.

Consider the following greedy algorithm:

```
decompose(n):
  L = an empty list
  while n > 0:
    find the largest Fibonacci number f <= n
    append f to L
    n = n-f
  return L
```

It can easily be shown that the above algorithm will write any positive integer n as a sum of *distinct* Fibonacci numbers. In other words, the above algorithm chooses one particular representation of n in Fibonacci base. For example, for n=9 we get the representation 10001 (i.e., 8+1), and for n=30 we get 1010001 (i.e., 21+8+1).

We can now define a new function g. The value g(n) is computed as follows:

1. Use the above algorithm to find a representation of n in Fibonacci base.
2. Take the sequence of digits obtained in step 1, and interpret it as a binary number (i.e., a number in base 2).
3. Return the value of that binary number.

For example, suppose that n=30. First, we compute that 30 in Fibonacci base is 1010001. Next, we compute that 1010001 in base 2 is 64+16+1=81. Hence, g(30)=81.

You are given longs **A** and **B**. Compute and return the following value: (g(**A**) xor g(**A**+1) xor g(**A**+2) xor ... xor g(**B**-2) xor g(**B**-1) xor g(**B**)) modulo 1,000,000,007.

Definition

Class:	FibonacciXor
Method:	find
Parameters:	long, long
Returns:	int

Method signature: int find(long A, long B)
(be sure your method is public)

Constraints

- **B** will be between 1 and 10^15, inclusive.
- **A** will be between 1 and **B**, inclusive.

Examples

1)

1
2
Returns: 3

We have g(1)=1, g(2)=2, and (1 xor 2)=3.

2)

3
10
Returns: 25

Our greedy algorithm chooses the following Fibonacci base representations for the numbers 3 through 10:

00100

00101

01000

01001

01010

10000

10001

10010

(Note that for clarity we included some leading zeros in some of the representations.) If we consider these as base-2 numbers, their values are 4, 5, 8, 9, 10, 16, 17, and 18. Thus, the answer is (4 xor 5 xor 8 xor ... xor 18) = 25.

3)

1
1000000000000000
Returns: 780431495

Don't forget the modulo 1,000,000,007.

3_5) Problem Statement

Farmer John and Eel Brus are studying string theory at the university. One day John found a very interesting sequence of strings s_1, s_2, ..., s_K and told Brus about it. Brus only remembers the following information:

- Each string in the sequence consists of lowercase letters only.
- The sequence starts with **A**. In other words, s_1 = **A**.
- The sequence ends with **B**. In other words, s_K = **B**.
- For each i between 1 and K-1, inclusive, s_{i+1} can be obtained by inserting one lowercase letter to s_i. For example, if s_1 is "tco", valid options for s_2 include "qtco", "trco", "tcso", and "tcot", but not "xco", "txoc", or "srm".

Return the number of sequences that match Brus's information, modulo 1,000,000,007. Brus's memory is always correct, so it is guaranteed that at least one such sequence exists.

Definition

Class:	StringSequences
Method:	countSequences
Parameters:	String, String
Returns:	int

Method signature: int countSequences(String A, String B)

(be sure your method is public)

Constraints

- **A** will contain between 1 and 49 characters, inclusive.
- Each character in **A** will be a lowercase letter ('a'-'z').
- **B** will contain between N+1 and 50 characters, inclusive, where N is the number of characters in **A**.
- Each character in **B** will be a lowercase letter ('a'-'z').
- There will be at least one sequence that matches Brus's information in the problem statement.

Examples

1)

"oxoxox"
"foxfoxfox"

Returns: 6

The following six sequences match Brus's information:

- "oxoxox", "foxoxox", "foxfoxox", "foxfoxfox"
- "oxoxox", "foxoxox", "foxoxfox", "foxfoxfox"
- "oxoxox", "oxfoxox", "foxfoxox", "foxfoxfox"
- "oxoxox", "oxfoxox", "oxfoxfox", "foxfoxfox"
- "oxoxox", "oxoxfox", "foxoxfox", "foxfoxfox"
- "oxoxox", "oxoxfox", "oxfoxfox", "foxfoxfox"

2)

"aaaaa"
"aaaaaaaa"
Returns: 1

Only the sequence "aaaaa", "aaaaaa", "aaaaaaa", "aaaaaaaa" matches Brus's information.

3)

"tco"
"tcotco"
Returns: 18

4)

"a"
"alnfrlrealjnsliejsraijneroav"
Returns: 135925750

5)

"ppmtmtttppmtmpppmtmtmp"
"ppmmmpmtmpttmttptpmttptmtmppmppttpmtpppppmmtmmmptp"
Returns: 856841145

3_6) Problem Statement

In this problem, all strings consist of uppercase English letters only. That is, there are 26 distinct letters. The weight of a string S can be computed as follows: for each letter that occurs at least once in S, its leftmost and rightmost occurrences L and R are found and the weight is increased by R-L. For example, if S="ABCACAZ", the weight of S is (5-0) + (1-1) + (4-2) + (6-6) = 7. (The leftmost occurrence of 'A' is at the index L=0, the rightmost one is at R=5, so 'A' contributes 5-0 = 5 to the weight of S. The only 'B' contributes 0, the pair of 'C's adds 2, and the only 'Z' adds 0.)

A string is S called *light* if no other string of the same length has a smaller weight.

You are given a int[] **L**. Manao is going to choose some *light* strings. The elements of **L** specify the lengths of these strings. For example, if **L** = { 3, 42, 1 }, Manao will first choose a light string of length 3, then a light string of length 42, and finally a light string of length 1. Then, Manao is going to concatenate all of the chosen strings, in the given order. Compute and return the smallest possible weight of a string Manao may obtain at the end.

Definition

Class:	StringWeight
Method:	getMinimum
Parameters:	int[]
Returns:	int
Method signature:	int getMinimum(int[] L)
(be sure your method is public)	

Constraints

- **L** will contain between 1 and 50 elements, inclusive.
- Each element of **L** will be between 1 and 100, inclusive.

Examples

1)

{1}

Returns: 0

Every string of length 1 has weight 0.

2)

{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}

Returns: 1

Manao is going to concatenate 27 strings of length 1. If Manao takes 25 distinct strings and 2 equal strings and places the equal strings side by side, the weight of the resulting string will be 1.

3)

{26, 2, 2}

Returns: 8

One possible concatenation of minimum weight is "ABC...XYZ"+"YZ"+"YZ".

4)

{25, 25, 25, 25}

Returns: 1826

5)

{14, 6, 30, 2, 5, 61}

Returns: 1229

3_7) Problem Statement

Little Petya likes rooted trees a lot. Recently he has received one as a gift from his mother. The only thing Petya likes more than rooted trees is playing with little Masha. The children painted each node of Petya's new tree either black or white. The tree is represented by the int[] **parent**. Let N denote the number of nodes in the tree. The nodes are numbered 0 through N-1. Node 0 is considered to be the root of the tree. Then for each i between 0 and N-2, inclusive, the tree contains an edge between nodes (i+1) and **parent**[i]. (Note that **parent**[i] may sometimes be greater than i+1.) The colors of nodes are given in the String **color** that consists of characters 'W' and 'B'. If the i-th character (0-based index) of **color** is 'W', then the i-th node is colored white, otherwise it's colored black.

The children decided to play a game with this tree. In the game Petya and Masha take alternating turns, Masha plays first. On his or her turn, the current player selects a white node, along with any subset of its descendants. (The subset can be arbitrary, possibly disconnected or even empty.) The player then toggles the color of all selected vertices: black nodes become white and vice versa. A player who can't make a turn loses the game. You goal is to determine who will be the winner assuming that both kids play optimally. Return "Masha" (without quotes) if Masha wins, otherwise return "Petya".

Definition

Class:	GameWithTree
Method:	winner
Parameters:	int[], String
Returns:	String
Method signature:	String winner(int[] parent, String color)
(be sure your method is public)	

Notes

- Node A is called a descendant of the node B if either B is a parent of A or a parent of A is descendant of the node B.

Constraints

- **parent** will contain N-1 elements.
- Each element of **parent** will be between 0 and N-1, inclusive.
- **color** will contain N characters.
- N will be between 2 and 50, inclusive.
- Each character of **color** will be either 'B' or 'W'.

- It's guaranteed that the graph described by **parent** is a rooted tree with root 0.

Examples

1)

{0}
"WW"

Returns: "Masha"

As the root is white, Masha may select the root together with any subset of other vertices in the tree. The optimal strategy for her is to select and toggle both vertices.

2)

{0,0}
"BWW"

Returns: "Petya"

Here the root is black. Masha must select and toggle exactly one of the leaves on her first turn. Then, Petya will select the other leaf and win the game.

3)

{0,1,2,3}
"BBBBB"

Returns: "Petya"

In this test case there are no white nodes, so Masha can't even make the first move.

4)

{5,5,6,6,0,0}
"BBWWBWW"

Returns: "Petya"

Here we have a black root that has two children with identical subtrees. Petya can mirror Masha's moves. This will guarantee him a victory.

5)

{5,5,6,6,0,0}
"BWWBBBW"

Returns: "Masha"

One optimal strategy for Masha: In the first turn, Masha will select and toggle only the node 6. Thus, only two white nodes will remain, both of them will be leaves. After Petya toggles one of them, Masha will toggle the other one and win. (Note that there are also other winning strategies for Masha in this situation.

3_8)Problem Statement

Alien Fred wants to destroy the Earth. But before he does that, he wants to play with a permutation.

Fred has a permutation of all the integers between 1 and N, inclusive. You are given a int[] **P** of size N. For each i, the i-th element (0-based index) of **P** represents the i-th element (again, 0-based index) of Fred's permutation.

Fred will now make a sequence of changes to **P**. Each change will look as follows: Fred will select a contiguous non-empty segment of **P**, and change all of its elements to the largest one among them. (Note that after such a change **P** will most likely stop being a permutation. This is allowed.)

You are also given an int **K**. Fred is allowed to make at most **K** consecutive changes to **P**. He is allowed to make fewer than **K** changes if he wants to, including the possibility to make no changes at all.

Let X be the total number of different sequences he can produce at the end of the above process. Return the value (X modulo 1,000,000,007).

Definition

Class:	AlienAndPermutation
Method:	getNumber
Parameters:	int[], int
Returns:	int

Method signature: int getNumber(int[] P, int K)

(be sure your method is public)

Constraints

- **P** will contain between 1 and 200 elements, inclusive.
- **P** will represent a permutation of integers between 1 and N, inclusive, where N is the number of elements in **P**.
- **K** will be between 0 and 200, inclusive.

Examples

1)

{1, 2}
1

Returns: 2

The following two sequences are possible in this case: (1, 2) and (2, 2).

2)

{3, 1, 2}
2

Returns: 4

Four sequences are possible in this case:

- (3, 1, 2)
- (3, 2, 2)
- (3, 3, 2)
- (3, 3, 3)

3)

{4, 3, 2, 1}
2

Returns: 13

4)

{1, 7, 2, 3, 6, 4, 5}
3

Returns: 77

5)

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
12

Returns: 379796836

3_9) Problem Statement

Two sequences A and B are called similar if they have the same length and have the following property: we can obtain two identical sequences by deleting at most two elements of A and at most two elements of B. (The elements deleted from A don't have to have the same indices as the elements deleted from B. The order of the remaining elements in each sequence has to be preserved.)

You are given ints N and bound. Consider all the sequences with length N such that each element is an integer between 1 and bound, inclusive. Compute the number of ordered pairs of such sequences that are similar. Return this count modulo 1,000,000,009.

Definition

Class: SimilarSequencesAnother
Method: getCount
Parameters: int, int
Returns: int
Method signature: int getCount(int N, int bound)
(be sure your method is public)

Constraints
- N will be between 1 and 100, inclusive.
- bound will be between 1 and 1,000,000,000, inclusive.

Examples
1)

2

10

Returns: 10000

Any two 2-element sequences are similar. There are (10^2)^2 = 10,000 pairs of such sequences.
2)

3

3

Returns: 687

In this case, two sequences are similar if they have at least one element in common. In other words, the two sequences are similar if there is a value that occurs in each of them. Let's count the pairs of sequences that are not similar. There are two possibilities:

Each sequence contains the same value three times. For example, one sequence could be {1,1,1} and the other {3,3,3}. There are 6 such pairs.

One sequence contains the same value three times, and the other contains the other two values, each of them at least once. For example, one sequence could be {2,3,2} and the other {1,1,1}. There are 2 * 3 * (2^3 - 2) = 36 such pairs.

Thus, the total number of pairs of similar sequences is 3^6 - 6 - 36 = 687.
3)

8

1

Returns: 1

4)

100

123456789

Returns: 439681851

5)

1

1000000000

Returns: 81

3_10) Problem Statement

Dimas is very fond of trees, he really enjoys solving problems on trees. Recently, his professor Rohit gave him a very difficult task to solve. This is it:

You're given a tree on **N** vertices. (A tree is an undirected connected graph with no cycles.) Different edges of the tree may have different lengths. Initially, all vertices are white. You now have to process **Q** queries. There are two types of queries:

- type 1: Given a node x, paint it blue.
- type 2: Given a node x, compute the sum of all distances between x and a blue node.

You are given the ints **N**, **Q**, **startSeed**, **threshold**, and **maxDist**. Use the algorithm given below as pseudocode to generate both the tree and the sequence of queries you should process.

```
int curValue = startSeed;

int genNextRandom() {
   curValue = (curValue * 1999 + 17) % 1000003;
   return curValue;
}

void generateInput() {
   for (int i = 0; i < N-1; i++) {
      distance[i] = genNextRandom() % maxDist;
            parent[i] = genNextRandom();
      if (parent[i] < threshold) {
         parent[i] = i;
      } else {
         parent[i] = parent[i] % (i + 1);
      }
   }

   for (int i = 0; i < Q; i++) {
      queryType[i] = genNextRandom() % 2 + 1;
      queryNode[i] = genNextRandom() % N;
   }
}
```

The output of the above pseudocode are four arrays: parent, distance, queryType, and queryNode.

The arrays parent and distance have **N**-1 elements each. For each valid i, our tree contains an edge between the vertices (i+1) and parent[i]. The length of that edge is distance[i]. Note that parent[i] will always be between 0 and i, inclusive.

The arrays queryType and queryNode have **Q** elements each. For each valid i, the i-th query (0-based index) you should process has the type queryType[i], and should be

applied to the vertex queryNode[i]. The queries must be processed in the given order.

Return the bitwise XOR of the answers to all type 2 queries.

Definition

Class:	TreeColoring
Method:	color
Parameters:	int, int, int, int, int
Returns:	long
Method signature:	long color(int N, int Q, int startSeed, int threshold, int maxDist)

(be sure your method is public)

Notes

- The intended solution does not rely on any properties of the tree and queries generator provided in the problem statement. It can process 100,000 queries on any tree containing up to 100,000 vertices. It is also able to calculate individual answers to each type 2 query (not just bitwise XOR of all answers).

Constraints

- **N** will be between 2 and 100,000, inclusive.
- **Q** will be between 1 and 100,000, inclusive.
- **startSeed** will be between 0 and 1,000,002, inclusive.
- **threshold** will be between 0 and 1,000,003, inclusive.
- **maxDist** will be between 1 and 1,000,003, inclusive.

Examples

1)

4
6
15
2
5

Returns: 7

- parent = {0,1,2}
- distance = {2,1,3}
- queryType = {2,1,2,2,2,1}
- queryNode = {2,3,2,3,1,3}

Here are our responses to the 6 queries:

- There are no blue nodes so the answer is clearly zero.
- We paint the node #3 blue.
- The distance between node #2 and node #3 is 3.

- The distance between node #3 and itself is 0.
- The distance between node #1 and node #3 is 4.
- As the node #3 is already blue, we just ignore this query.

2)

4
5
2
9
10

Returns: 30

Here are the edges of the tree you should generate: 0-1 (length 5), 0-3 (length 4), and 1-2 (length 6).

- For query 0 we return 0 because there are no blue nodes yet.
- Queries 1 and 2 instruct us to color vertices 0 and 3 blue.
- Then, query 3 asks us to compute the sum of distances between the vertex 2 and each of the blue nodes. The distance between 2 and 0 is 11, and the distance between 2 and 3 is 15. Hence the sum of all distances is 11+15 = 26.
- Similarly we can compute that the answer to the last query is 4+0 = 4.

3)

8
8
3
5
7

Returns: 6

4)

14750
50
29750
1157
21610

Returns: 2537640

5)

100000
100000
123456
500000
474747

Returns: 726915029831

6)

100000
100000
654321
1000003
1000003

Returns: 562600687570528