CS4050 Design And Analysis of Algorithms

Assignment Problem Set

2_1) Problem Statement

Cat Snuke and wolf Sothe are playing the Path Game.

The Path Game is played on a rectangular grid of square cells. The grid has 2 rows and some positive number of columns. Each cell is either black or white.A left-to-right path in the grid is a sequence of white cells such that the first cell in the sequence is in the leftmost column, the last cell in the sequence is in the rightmost column, and each pair of consecutive cells shares a common side.The initial coloring of the grid is such that there is at least one left-to-right path. You are given this initial coloring as a String[] **board** with two elements. For each i and j, **board**[i][j] is either '#' (representing a black cell) or '.' (representing a white cell).

The players take alternating turns. In each turn, the current player has to choose and color one white cell black. Snuke goes first. The game ends when there is no longer a left-to-right path on the board. The last player who colored a cell loses the game. In other words, the loser is the player who was forced to block the last left-to-right path.

Assume that both players play the game optimally. Return "Snuke" (quotes for clarity) if Snuke wins the game, and "Sothe" otherwise.

Definition

Class:	PathGame
Method:	judge
Parameters:	String[]
Returns:	String

Method signature: String judge(String[] board)
(be sure your method is public)

Constraints

- **board** will contain 2 elements.
- Each element in **board** will contain between 1 and 1000 characters, inclusive.
- All elements in **board** will have the same length.
- Each character in **board** will be '#' or '.'.
- The grid described by **board** will contain a left-to-right path.

Examples

1)

{"#.."

```
,"..."}
```

Returns: "Snuke"

Snuke has exactly one winning move: he must color the lower right cell. After this move the resulting grid will still contain a left-to-right path. Sothe will then have four possible moves, but each of those loses the game immediately.

2)

```
{"#"
,"."}
```

Returns: "Sothe"

Snuke has to color the only white cell black, and he immediately loses the game.

3)

```
{"....."
,"..#.."}
```

Returns: "Sothe"

4)

```
{".#..."
,"....."}
```

Returns: "Snuke"

5)

```
{".....#..#........##......."
,"..........#..........#...."}
```

Returns: "Snuke"

2_2) Problem Statement

Alice likes eating candies. Her favorite type of candy are the Surprise Candies. Surprise Candies come in N different flavors and in N different shapes. You know the following facts about the shapes and flavors of Surprise Candies:

- The shapes are numbered 0 through N-1.
- The flavors are numbered 0 through N-1.
- You can tell the shape of a candy before buying it. (Thus, you can do stuff like "buy exactly 47 candies of shape 3".)
- You can only tell the flavor of a candy when eating it. (Thus, you *do not* know the flavor when you are buying the candy.)
- For each shape of Surprise Candies, there are exactly two flavors that can have that shape.
- For each flavor of Surprise Candies, there are exactly two shapes that can have that flavor.

In Alice's street there is a store that sells Surprise Candies. Alice knows the exact inventory of this store. You are given this information in int[]s **Type1**, **Number1**, **Type2**, and **Number2**. Each of these int[]s has exactly N elements. For each i, their elements at index i correspond to the shape number i, as follows:

- The store contains exactly **Number1**[i] candies with shape i and flavor **Type1**[i].
- The store contains exactly **Number2**[i] candies with shape i and flavor **Type2**[i].

Alice wants to eat candies of all N flavors today. However, she is lazy to go to the store, so she sent Kirito to do the shopping for her. Kirito must buy a set of candies that is guaranteed to contain all N flavors. Return the smallest number of candies Kirito may buy.

Definition

Class:	CandyCollection
Method:	solve
Parameters:	int[], int[], int[], int[]
Returns:	int
Method signature:	int solve(int[] Type1, int[] Number1, int[] Type2, int[] Number2)

(be sure your method is public)

Constraints

- N will between 1 and 1000, inclusive.
- **Type1**, **Number1**, **Type2**, **Number2** will each contain exactly N elements.
- For each i, **Type1**[i] and **Types2**[i] will be different.

- Each element of **Type1** and **Type2** will be between 0 and N-1, inclusive.
- Each element of **Number1** and **Number2** will be between 1 and 1000, inclusive.
- Each of the values 0 through N-1 will appear exactly twice in **Type1** and **Type2** together.

Examples

1)

{0,0}

{1,1}

{1,1}

{1,1}

Returns: 2

In this test case we have N=2. Thus, there are two shapes and two flavors. The store has exactly one candy for each combination (shape,flavor). Kirito can simply buy two candies with the same shape, their flavors must be different.

2)

{0,0}

{2,5}

{1,1}

{2,5}

Returns: 3

In this test case we have N=2 again, but now the supply of candies in the store is larger. There are 2+2 = 4 candies of shape 0, and 5+5 = 10 candies of shape 1. The optimal strategy for Kirito is to buy 3 candies of shape 0. Both flavors have to be present in those three candies.

3)

{0,0,2,3}

{1,1,2,2}

{1,1,3,2}

{1,1,2,2}

Returns: 5

One optimal solution is to buy two candies of shape 0 and three candies of shape 2.

4)

{0,1,2,3}

{5,5,10,13}

{1,2,3,0}

{8,8,10,15}

Returns: 20

5)

{12,9,0,16,9,18,12,3,6,0,8,2,10,6,5,2,14,10,5,13}

{895,704,812,323,334,674,665,142,712,254,869,548,645,663,758,38,860,724,742,530}

{1,4,7,11,15,8,18,13,17,17,19,14,7,11,4,1,15,19,3,16}

{779,317,36,191,843,289,107,41,943,265,649,447,806,891,730,371,351,7,102,394}

Returns: 5101

2_3) Problem Statement

Jeip drew N black points onto the plane. The points were in a special configuration: for each pair of points, their distance was either at most **D**, or at least 1.99***D**.

You are given the coordinates of the points as int[]s **x** and **y**. You are also given the int **D**.

Ouju wants to paint some (possibly none or all) of the N points red. The distance between any two red points must be at least 1.99***D**. Return the number of ways to do this, modulo 1,000,000,007.

Definition

Class:	OnePointNineNine
Method:	countSubsets
Parameters:	int[], int[], int
Returns:	int
Method signature:	int countSubsets(int[] x, int[] y, int D)

(be sure your method is public)

Constraints

- **x** and **y** will contain between 1 and 1000 elements, inclusive.
- **x** and **y** will contain the same number of elements.
- Each element of **x** and **y** will be between 0 and 1,000,000,000, inclusive.
- **D** will be between 1 and 1,000,000,000, inclusive.
- No two points are exactly at the same position.
- The points are in a special configuration described in the statement.

Examples

1)

```
{0,0,10,10}
{0,10,0,10}
47
```

Returns: 5

At most one of these four points can be red.

2)

```
{0,0,10,10}
{0,10,0,10}
4
```

Returns: 16

Any subset of these four points can be red.

3)

{0,4,8}

{0,3,6}

5

Returns: 5

In one of the valid solutions the points (0,0) and (8,6) are both red.

4)

{0, 4, 8, 20, 25, 30, 35, 40}

{0, 3, 6, 20, 20, 20, 20, 20}

5

Returns: 65

5)

{4637, 7770, 9911, 3887, 310, 8546, 104, 9820, 6710, 4128, 8224, 2492, 8956, 6162, 3392, 9736, 1540, 7744, 3783, 5451, 3756, 6153, 4846, 9852,
2678, 6500, 4117, 3994, 9126, 8950, 4913, 8598, 5692, 3400, 133, 4284, 656, 4742, 8727, 4904, 338, 7144, 7447, 8807, 1985, 6591, 40, 9614, 1839,2724, 391, 1419, 2404, 9268, 1490, 3121, 654, 1337, 7787, 9269, 9413, 4515, 7787, 8622, 6718, 839, 238, 2490, 253, 1029, 9286, 5226, 180, 6451,7826, 1707, 5119, 7238, 3393, 8980, 7234, 879, 5481, 703, 3991, 35, 3205, 2697, 9462, 4489, 2074, 7880, 1909, 150, 2378, 1555, 5232, 5959, 8755, 7679}

{4026, 2791, 3044, 4049, 6759, 6606, 3440, 8858, 6954, 2544, 4778, 2367, 5113, 8588, 3772, 4741, 3693, 5140, 8822, 8853, 9934, 6277, 5097, 285,1031, 9872, 1012, 5883, 8992, 7257, 8889, 6558, 9997, 3868, 7731, 7508, 3729, 6398, 4102, 2054, 4835, 5707, 4271, 1676, 9487, 6336, 9829, 1058,9965, 4998, 1042, 6320, 7669, 1893, 6021, 4211, 8496, 7585, 6882, 8410, 5155, 5869, 3376, 7173, 5726, 1574, 8911, 4192, 8324, 963, 8867, 7292,7127, 4238, 6796, 6225, 4143, 7775, 4312, 965, 5933, 558, 8642, 268, 7208, 5688, 267, 4338, 4023, 7982, 4535, 545, 7228, 1884, 1660, 3241, 6388, 6572,6515, 5912}

1

Returns: 976371285

2_4) Problem Statement

There are N airports numbered 0 through N-1. Airport 0 is called the central airport. The coordinates of the airport i is (**x**[i], **y**[i]). You are given a String[] **flight**. If the j-th (0-based) character of the i-th element of **flight** is '1', there is a direct flight between the airport i and the airport j. Otherwise there is no direct flight between the airport i and the airport j. Note that direct flights are bidirectional (in other words, **flight** is symmetric). There are no flights that start and end at the same airport.

We say that airport Y is reachable from airport X if there is a sequence of zero or more consecutive flights that gets us from X to Y. Notably, any airport is always reachable from itself. Currently, some airports may be unreachable from the central airport. Cat Snuke decided to fix this by performing the following process:

1. If all airports are reachable from the central airport, end the process.
2. Choose an airport that is reachable from the central airport uniformly at random. Let's call this airport A.
3. Choose an airport that is not reachable from the central airport uniformly at random. Let's call this airport B.
4. Add a bidirectional direct flight between A and B, and go to step 1.

After the process ends, Cat Snuke computes the length of the shortest path between airports 0 and 1. The length of a path is defined as the sum of lengths of all direct flights used in the path. The length of a direct flight is defined as the Euclidean distance between the two airports it connects. Return the expected length of the shortest path between airports 0 and 1.

Definition

Class:	RandomFlights
Method:	expectedDistance
Parameters:	int[], int[], String[]
Returns:	double
Method signature:	double expectedDistance(int[] x, int[] y, String[] flight)

(be sure your method is public)

Constraints

- N will be between 2 and 18, inclusive.
- **x** and **y** will contain exactly N elements each.
- Each element of **x** and **y** will be between 0 and 1000, inclusive.
- No two airports are at exactly the same position.
- **flight** will contain exactly N elements.
- Each element of **flight** will contain exactly N characters.
- Each character in **flight** will be either '0' or '1'.
- **flight** will be symmetric according to the main diagonal.
- For each valid i, the i-th character of the i-th element of **flight** will be '0'.

Examples

1)

```
{7, 10, 9}
{1, 3, 3}
{"000",
 "001",
 "010"}
```

Returns: 3.7169892001050897

Initially there is only one direct flight: between airports 1 and 2. If Snuke adds a direct flight between 0 and 1, the length of the shortest path will be sqrt(13). If Snuke adds a direct flight between 0 and 2, the length of the shortest path will be sqrt(8) + 1. Both cases are equally likely, so the expected length is (sqrt(13) + sqrt(8) + 1) / 2 = 3.716...

2)

```
{4, 1, 5, 6, 6}
{1, 5, 10, 3, 4}
{"00110",
 "00101",
 "11000",
 "10001",
 "01010"}
```

Returns: 8.927446638338974

All airpots are already reachable from the central airport, so no new direct flights will be added. There are two paths from airport 0 to airport 1: 0 -> 2 -> 1 (length: 15.458...) and 0 -> 3 -> 4 -> 1 (length: 8.927...). You should return the length of the shortest path.

3)

```
{7, 10, 9, 7, 7}
{1, 3, 3, 6, 4}
{"00001",
 "00000",
 "00010",
 "00100",
 "10000"}
```

Returns: 6.2360162308285005

4)

```
{97, 27, 20, 34, 30, 37, 65, 21, 74, 27, 84, 79, 15, 78, 16, 7, 11, 24}
{1, 72, 20, 73, 58, 55, 45, 19, 48, 4, 33, 22, 25, 95, 100, 85, 65, 53}
{"000000000000000100",
 "001010000001000000",
 "010010000000000000",
 "000000000000100001",
 "011000000001000000",
 "000000001000000010",
 "000000000000010000",
 "000000000000000100",
 "000001000100001010",
 "000000001000000010",
 "000000000000100000",
 "010010000000000000",
 "000100000010000001",
 "000000100000000000",
 "000000001000000010",
 "100000010000000000",
 "000001001100001000",
 "000100000000100000"}
```

Returns: 255.25627726422454

5)

```
{35, 8, 71, 81, 43, 76, 55, 15, 72, 39, 99, 23, 14, 77, 47, 43, 60, 67}
{68, 96, 98, 16, 7, 74, 52, 63, 98, 77, 52, 93, 52, 4, 56, 11, 75, 63}
{"000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000",
 "000000000000000000"}
```

Returns: 138.0804889521365

2_5) Problem Statement

Nancy has a directed graph with **N** vertices and E edges. The vertices are numbered 1 through **N**. Each edge of the graph has a positive integer weight. This graph is described by three int[]s with E elements each: **from**, **to**, and **weight**. For each valid i, the graph contains an edge from **from**[i] to **to**[i], and its weight is **weight**[i]. Note that Nancy's graph may contain multiple edges with the same start and end. It may also contain self-loops.

Nancy is currently standing in the vertex 1. She can reach other vertices by moving along the edges. The cost of using an edge is equal to its weight. Nancy's goal is to reach the vertex **N** and to minimize the total cost of doing so.

Nancy has a special power she can use to make her travels cheaper. Whenever she traverses an edge, she can use that special power to make the weight of that edge temporarily negative. You are given an int **charges**: the number of times Nancy can use her special power. Each use of the special power only works for one traversal of an edge. Nancy can traverse each edge arbitrarily many times. Each time she traverses an edge, she may use her special power, if she still has some charges left.

Compute and return the minimal total cost of Nancy's journey.

Definition

Class:	NegativeGraphDiv1
Method:	findMin
Parameters:	int, int[], int[], int[], int
Returns:	long
Method signature:	long findMin(int N, int[] from, int[] to, int[] weight, int charges)

(be sure your method is public)

Constraints

- **N** will be between 1 and 50, inclusive.
- E will be between 1 and 2500, inclusive.
- **from, to, weight** will each contain exactly E elements.
- **from** and **to** will only contain numbers between 1 and **N**, inclusive.
- There will be a path from node 1 to node **N**.
- **weight** will contain numbers between 0 and 100,000, inclusive.
- **charges** will be between 0 and 1,000,000,000, inclusive.

Examples

1)

```
3
{1,1,2,2,3,3}
{2,3,1,3,1,2}
{1,5,1,10,1,1}
1
Returns: -9
```

The optimal path for Nancy is 1->2->3, and using her single charge on the last edge.

2)

```
1
{1}
{1}
{100}
100000
Returns: -10000000
```

The graph may contain self-loops. Here, the optimal solution is that Nancy uses the self-loop 100,000 times, each time using her special power to change its cost from 100 to -100.

3)

```
2
{1,1,2}
{2,2,1}
{6,1,4}
2
Returns: -9
```

There can be multiple edges between vertices.

4)

```
2
{1}
{2}
{98765}
1000000000
Returns: -98765
```

Nancy may not be able to use all her charges.

5)

```
40
{21,2,36,21,32,1,34,1,40,38,19,10,39,40,31,29,22,18,24,8,25,1,12,31,1,34,16,13,39,39,26,30,4,28,8,9
,27,13,6,16,7,11,7,38,30,20,22,29,19,5,22,13,12,7,33,5,10,31,10,39,18,18,3,19,17,17,34,9,7,17,21,13,
12,16,36,39,9,7,3,5,26,16,32,4,26,12,27,24,19,1,19,17,9,22,16,12,31,37,32,9,31,8,2,39,18,26,32,12,2
8,11,32,34,2,12,12,33,27,24,5,5,40,34,4,8,10,17,39,30,26,24,10,37,23,40,38,17,4,28,33,31,28,19,36,5
,24,17,11,19,4,40,20,16,11,10,9,22,23,23,8,30,10,23,16,21,10,18,8,28,15,20,38,5,22,4,29,32,13,13,15
,24,28,27,11,24,24,23,40,34,20,28,18,26,34,21,13,11,33,28,8,5,9,31,1,32,7,22,12,8,12,8,12,31,35,33,
27,18,6,22,38,9,40,35,15,16,30,4,3,29,2,34,40,3,12,20,29,14,2,3,8,37,12,28,25,7,22,33,4,15,5,14,26,2
2,16,33,12,11,14,11,5,25,30,21,20,30,25,30,28,37,23,31,30,3,15,5,25,14,8,13,12,4,18,9,20,17,11,21,5
```

,25,15,9,40,26,28,36,1,10,33,34,5,3,21,32,15,30,33,32,31,19,12,2,16,13,15,4,33,33,26,6,7,36,20,14,7,
39,17,33,4,5,22,21,13,29,38,34,6,24,18,29,4,20,33,16,14,30,20,20,7,21,13,5,20,1,8,18,9,12,24,10,22,
33,40,28,30,23,7,36,27,38,36,15,3,36,8,20,27,12,5,33,40,7,25,20,13,36,30,13,9,3,15,38,33,27,36,4,9,
18,39,7,12,30,17,2,21,17,11,26,14,29,26,31,15,13,12,19,35,11,25,19,15,34,9,12,17,37,22,22,16,10,13
,17,12,12,32,1,40,10,34,29,39,7,17,3}

{27,37,32,14,19,25,4,14,40,9,36,23,21,25,39,13,4,30,11,32,22,12,29,40,35,32,4,15,25,8,17,18,17,19,
34,1,16,16,26,28,2,28,21,16,6,3,12,39,24,31,3,25,26,30,9,35,29,20,4,21,25,15,32,27,24,13,3,10,21,40
,12,39,4,10,2,39,28,23,21,19,21,11,32,20,36,7,11,21,16,29,17,24,18,32,17,38,23,35,1,19,28,20,31,37,
16,20,7,31,3,7,36,15,36,10,6,37,26,39,39,26,31,8,36,26,10,21,24,33,13,29,31,10,12,20,31,39,35,29,3
9,30,34,26,15,30,4,36,16,38,2,31,22,18,17,12,21,4,11,32,3,27,14,22,16,37,40,18,25,28,20,27,27,30,1
1,19,16,18,25,4,21,14,14,13,22,34,25,9,31,27,14,32,23,21,3,1,24,15,30,35,1,24,37,7,28,8,21,40,6,29,
36,37,21,8,17,20,20,28,22,22,13,29,34,20,26,11,10,8,33,10,37,11,16,5,9,18,39,1,8,4,5,2,20,30,4,23,3
0,23,19,25,30,37,40,31,31,33,33,8,36,40,15,11,31,37,5,40,2,11,19,32,1,32,27,7,23,9,13,18,12,36,11,3
2,34,38,35,15,6,36,32,28,6,40,2,33,2,6,7,27,40,30,40,12,39,13,1,22,17,36,1,4,34,25,25,6,20,25,30,4,2
4,35,7,5,29,20,28,37,1,30,6,5,5,1,10,40,13,14,21,32,19,25,37,21,28,34,28,29,27,24,33,35,12,25,3,4,1
2,31,12,7,28,27,10,35,35,18,6,5,16,24,39,26,15,29,13,5,14,36,35,14,2,39,22,21,2,2,19,38,15,25,10,13
,2,39,31,13,5,39,11,33,4,18,18,38,19,39,39,32,23,34,30,21,30,16,36,22,25,14,13,16,10,8,27,29,40,1,2
9,4,20,25,32,21,40,1,10,2,17,5,8,5,31,18,18,25,13,3}

{93486,55539,34680,56705,10415,99106,26365,51838,20794,93178,33943,80260,60780,2405,7834
6,71929,24723,37614,62649,83486,32073,83866,88345,83213,28266,12730,27623,25353,89948,550
02,36720,87151,39759,66091,3690,83881,82635,69084,54138,65876,54205,10236,90478,37230,223
37,8209,14258,18375,5268,21824,76819,4094,63971,1454,21318,44848,92540,26354,18611,2394,6
8363,64345,60985,725,33692,21195,60992,10243,62006,74884,7822,66830,16114,14663,60701,381
74,30493,26360,7745,7165,47668,83884,89463,46615,7729,22187,78818,10817,31141,79159,24280
,67585,48039,3000,98743,39455,12030,27595,55470,14617,93275,21233,72418,26911,74250,13704
,68327,89871,72035,3437,97910,2325,83391,39732,79248,22431,93095,65547,74205,33341,2112,9
3431,32198,96772,20187,71348,19102,10840,27306,76027,86368,55373,40192,49370,90597,21685,
23630,68099,50880,84474,61907,8446,30127,80977,53007,36125,47411,16950,7445,17166,71035,2
0293,24724,7404,6579,11282,83421,37466,80598,53398,15720,66117,28965,90631,44004,27582,29
003,65915,35683,48237,66625,50977,82957,54602,3610,431,36560,38034,5948,72313,16772,81862
,11871,33534,91097,93908,57721,91890,46009,21400,41257,66253,90004,56285,85267,23486,6953
,66722,69568,74398,32371,15307,26503,31676,50683,46103,43963,90273,8938,41054,90794,95115
,11689,61662,72089,12258,2816,4029,36441,17784,53116,4444,93653,63726,57293,38647,59684,7
2212,770,60745,25817,47807,85312,27451,51063,83640,2978,49906,744,69457,20244,8799,7,8911
8,78421,58827,38689,21327,32558,35756,53505,30526,39049,63417,24967,54923,62001,38920,597
19,65486,174,52231,84176,83531,39896,70918,52397,5169,14876,92511,70020,88390,1168,35311,
56953,12966,94805,3885,14103,24615,10710,27635,24064,23022,4506,29939,2044,48330,53688,41
320,21646,62598,41367,22910,70284,11512,64411,16215,41072,87263,38301,81731,41930,96029,9
3359,32457,98487,62899,51282,8656,45002,83345,36991,77420,37086,53484,18823,74711,81453,8
1394,79933,38466,61198,98665,5528,9196,40298,55089,24162,99257,58066,90088,66809,53472,73
237,31964,4792,42336,88141,80467,96893,44276,76106,79430,17072,85727,49790,84394,47215,16
792,44544,75123,4203,6283,49250,54852,27312,42709,97204,16834,80593,27523,31700,44435,253
38,43513,84894,43514,90145,74675,54302,85249,10281,44231,28994,63813,24198,24686,7082,464
71,3025,57872,57510,13666,48726,3907,12500,53578,28346,15804,32122,89396,15406,11207,5957
1,78192,84950,84280,84597,27811,56165,69239,53400,32562,95679,32377,5909,29805,85810,3159
3,85985,27637,24974,41557,56442,48912,62362,98580,59154,78739,75721,72097,90622,91334,796
85,19371,15487,67804,2582,26623,14347,98894,92041,83228,58089,96181,3913,75974,18569,1142
8,95165,27563}

160743953

Returns: -15328623718914

2_6) Problem Statement

One of the modes in the game "osu!" is called "catch the beat". In this mode, you control a character that catches falling fruit. The game is played in the vertical plane. For simplicity, we will assume that both your character and all pieces of fruit are points in that plane. You start the game at the coordinates (0, 0). Your character can only move along the x-axis. The maximum speed of your character is 1 unit of distance per second. For example, you need at least 3 seconds to move from (-2, 0) to (1, 0).

There are **n** pieces of fruit. The pieces are numbered 0 through **n**-1. For each i, fruit i starts at (x[i], y[i]). All pieces of fruit fall down with constant speed of 1 unit of distance per second. That is, a fruit currently located at (xf, yf) will move to (xf, yf-t) in t seconds. You will catch a fruit if the character is located at the same point as the fruit at some moment in time.

The initial coordinates x[] and y[] are generated using the following pseudocode:

```
x[0] = x0
for i = 1 to n-1:
   x[i] = (x[i-1] * a + b) % mod1

for i = 0 to n-1:
   x[i] = x[i] - offset

y[0] = y0
for i = 1 to n-1:
   y[i] = (y[i-1] * c + d) % mod2
```

(In the pseudocode, '%' represents the 'modulo' operator.)

You are given all the ints used in the above pseudocode. Return the maximum number of pieces of fruit you can catch.

Definition

Class:	CatchTheBeat
Method:	maxCatched
Parameters:	int, int, int, int, int, int, int, int, int, int
Returns:	int
Method signature:	int maxCatched(int n, int x0, int y0, int a, int b, int c, int d, int mod1, int mod2, int offset)

(be sure your method is public)

Constraints

- **n** will be between 1 and 500,000, inclusive.
- **mod1** and **mod2** will be between 1 and 1,000,000,000, inclusive.
- **x0**, **a** and **b** will be between 0 and (**mod1** - 1), inclusive.

- **y0**, **c** and **d** will be between 0 and (**mod2** - 1), inclusive.
- **offset** will be between 0 and 1,000,000,000, inclusive.

Examples

1)

```
3
0
0
1
1
1
1
100
100
1
```

Returns: 2

There are 3 pieces of fruit. Their initial coordinates are (-1, 0), (0, 1), and (1, 2). Clearly you cannot catch fruit 0. You can catch the other two. One way of doing so:

1. Wait at (0, 0) for 1 second.
2. Catch fruit 1.
3. Move to (1, 0) in 1 second.
4. Immediately catch fruit 2.

2)

```
1
0
1234
0
0
0
0
1000000000
1000000000
1000
```

Returns: 1

The only fruit is located at (-1000, 1234). We can go to (-1000, 0) and then wait for 234 seconds to catch it.

3)

```
1
0
999
0
0
```

```
0
0
1000000000
1000000000
1000
```

Returns: 0

Now the only fruit is located at (-1000, 999). We can't catch it.

4)

```
100
0
0
1
1
1
1
3
58585858
1
```

Returns: 66

5)

```
500000
123456
0
1
0
1
1
1000000000
1000000000
0
```

Returns: 376544

The fruits are located in (123456, 0), (123456, 1), ..., (123456, 499999).

6)

```
500000
0
0
0
0
0
0
1
1
0
```

Returns: 500000

In this case all the fruits start at (0, 0). Note that there can be more than one fruit at any position. We can catch all such fruit at the same time.

7)

```
10
999999957
79
993948167
24597383
212151897
999940854
999999986
999940855
3404
```

Returns: 3

Watch out for integer overflow when generating the coordinates.

2_7) Problem Statement

You have N computers numbered 0 through N-1. They are connected into a single network. The topology of the network is a tree. You are given its description as int[]s **parent** and **dist**. Each of the int[]s contains exactly N-1 elements. For each i between 0 and N-2, inclusive, there is a cable connecting computers i+1 and **parent**[i], and the length of that cable is **dist**[i]. You are also given an int **maxDist** with the following meaning: The distance between any two computers in the same network must not exceed **maxDist**. (The distance between two computers is defined as the total length of cable between them.) If this is currently not the case for your network, you have to divide it into several smaller networks.

Formally, it means that you need to choose the number K of smaller networks you will have. Then you need to assign each of your computers into exactly one of the K networks. The following properties must be satisfied:

- Each of the K new networks must form a connected subtree of the original tree.
- The diameter of each new network must be at most **maxDist**.

Return the smallest value of K for which it is possible to divide the original network into K new networks with the above properties.

Definition

Class:	Ethernet
Method:	connect
Parameters:	int[], int[], int
Returns:	int

Method signature: int connect(int[] parent, int[] dist, int maxDist)
(be sure your method is public)

Constraints

- **parent** will contain between 1 and 50 elements, inclusive.
- **dist** will contain the same number of elements as **parent**.
- For each valid i, the i-th element of **parent** will be between 0 and i, inclusive.
- Each element of **dist** will be between 1 and 500, inclusive.
- **maxDist** will be between 1 and 500, inclusive.

Examples

1)

```
{0,0,0}
{1,1,1}
2
Returns: 1
```

The diameter of this network is 2, which is small enough.

2)

```
{0,0,0,0,0,0,0}
{1,2,3,4,5,6,7}
8
Returns: 4
```

One optimal solution: the new networks will be formed by computers {4}, {6}, {7}, and {0,1,2,3,5}.

3)

```
{0,1,2,3,4,5}
{1,2,3,4,5,6}
6
Returns: 3
```

One optimal solution is to put computers {0,1,2,3} into the first new network, {4,5} into the second one, and {6} will be the third network.

4)

```
{0,0,0,1,1}
{1,1,1,1,1}
2
Returns: 2
```

The two new networks can be {0,2,3} and {1,4,5}.

5)

```
{0,1,0,3,0,5,0,6,0,6,0,6,4,6,9,4,5,5,2,5,2}
{93,42,104,105,59,73,161,130,30,81,62,93,131,133,139,5,13,34,25,111,4}
162
Returns: 11
```

2_8) Problem Statement

Vasa has two undirected trees. Each of the trees has n vertices. In each tree, the vertices are labeled 0 through n-1, inclusive, in no particular order. The shapes of the trees may be different. You are given the description of the two trees as int[]s **tree1** and **tree2** with n-1 elements each. For each valid i, the vertices i and **tree1**[i] are connected by an edge in the first tree. Similarly, for each valid i, the vertices i and **tree2**[i] are connected by an edge in the second tree.

Vasa tried to measure how similar the two trees are. He did so in two steps. First, he defined the value S(e1,e2): the similarity between the edge e1 in the first tree and the edge e2 in the second tree. Then, he defined the similarity between his two trees as the sum of sqr(S(e1,e2)) over all pairs (e1,e2), where sqr(x)=x*x. The definition of S(e1,e2) follows.

We will now define the similarity S(e1,e2). Consider the following process:

- If we were to remove the edge e1 from the first tree, it would split into two components. Choose one of them and call it A.
- If we were to remove the edge e2 from the second tree, it would split into two components. Choose one of them and call it B.
- Let X be the set of labels that are present both in A and in B.

Then, the similarity S(e1,e2) is defined to be the largest possible size of the set X. (The maximum is taken over both choices of A and both choices of B.)

Compute and return the similarity between the two given trees.

Definition

Class:	TreesAnalysis
Method:	treeSimilarity
Parameters:	int[], int[]
Returns:	long

Method signature: long treeSimilarity(int[] tree1, int[] tree2)
(be sure your method is public)

Constraints

- n will be between 2 and 4,000, inclusive.
- **tree1** will contain n - 1 elements.
- **tree1** and **tree2** will contain the same number of elements.
- Each element of **tree1** will be between 0 and n - 1, inclusive.
- Each element of **tree2** will be between 0 and n - 1, inclusive.
- **tree1** and **tree2** will represent trees as described in the problem statement.

Examples

1)

{1}
{1}

Returns: 1

The similarity between the edge 0-1 in the first tree and the edge 0-1 in the second tree is 1. One corresponding set X is the set X = {0}.

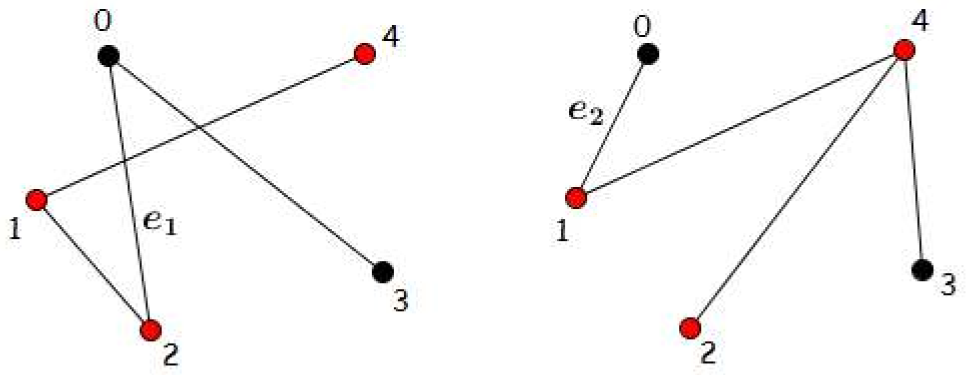
2)

{2, 4, 1, 0}
{1, 4, 4, 4}

Returns: 111

Here the first tree is a path (edges 0-2, 1-4, 2-1, and 3-0) and the second tree is not a path (edges 0-1, 1-4, 2-4, and 3-4). As each tree has four edges, their similarity is the sum of 4*4 = 16 terms.

For example, let's compute one of those 16 terms. The similarity between the edge e1 = 0-2 in the first tree and the edge e2 = 0-1 in the second tree is 3, as depicted in the figure below. Hence, this pair of edges contributes sqr(3) = 3*3 = 9 towards the total similarity between these two trees.



3)

{1, 2, 3, 4}
{1, 2, 3, 4}

Returns: 128

4)

{2, 3, 4, 4, 5, 8, 5, 6, 10, 8}
{9, 0, 1, 0, 3, 0, 5, 0, 7, 10}

Returns: 6306

5)

{222, 261, 167, 133, 174, 150, 165, 311, 208, 268, 111, 222, 154, 277, 282, 201, 46, 124, 194, 331, 4, 216, 111, 275, 72, 322, 137, 216, 241, 48, 72, 101, 232, 165, 151, 263, 139, 16, 122, 140, 84, 135, 314, 106, 309, 126, 102, 151, 208, 27, 242, 93, 83, 314, 136, 77, 82, 215, 16, 232, 286, 156, 294, 38, 67, 204, 206, 137, 174, 282, 188, 143, 84, 279, 236, 136, 158, 10, 65, 332, 122, 44, 329, 62, 174, 67, 102, 216, 245, 296, 287, 307, 93, 197, 169, 268, 266, 294, 157, 277, 95, 68, 214, 135, 211, 127, 82, 108, 212, 161, 243, 212, 207, 119, 119, 158, 97, 290, 21, 217, 230, 85, 171, 13, 138, 294, 304, 204, 318, 115, 70, 210, 195, 223, 37, 164, 149, 3, 164, 328, 316, 108, 330, 48, 38, 324, 222, 193, 50,41, 184, 93, 148, 41, 151, 139, 106, 301, 264, 80, 249, 110, 244, 109, 212, 223, 279, 330, 67, 27, 301, 165, 236, 194, 3, 157, 1, 217, 311, 87,105, 4, 286, 37, 6, 31, 111, 66, 230, 227, 244, 322, 196, 65, 69, 305, 112, 133, 231, 68, 153, 206, 309, 248, 329, 58, 69, 69, 328, 0,29, 233, 243, 305, 167, 80, 65, 194, 190, 179, 142, 196, 324, 206, 134, 50, 272, 261, 10, 147, 329, 322, 14, 142, 169, 21, 296, 284, 241, 55, 304, 150, 166, 230, 167, 304, 87, 156, 156, 97, 274, 324, 196, 101, 82, 106, 260, 242, 233, 207, 305, 10, 166, 53, 18, 154, 233, 217, 296, 263, 168, 138, 30, 115, 135, 188, 98, 309, 292, 204, 150, 210, 332, 330,166, 96, 70, 24, 229, 215, 201, 285, 40, 287, 142, 68, 133, 208, 268, 161, 310, 41, 203, 73, 275, 184, 163, 227, 89, 110, 328, 108, 112, 125, 164, 127, 179, 267, 221, 49, 139, 1, 84, 136, 38, 6, 70, 246, 243, 3, 188, 297}

{174, 262, 195, 288, 157, 278, 36, 133, 230, 273, 222, 138, 97, 23, 189, 141, 296, 55, 45, 301, 81, 199, 188, 289, 187, 164, 113, 58, 138, 300, 289, 282, 329, 91, 130, 178, 92, 143, 48, 81, 311, 133, 151, 286, 171, 196, 199, 80, 83, 121, 65, 151, 277, 136, 49, 111, 58, 36, 259,14, 31, 9, 136, 181, 122, 324, 249, 114, 9, 37, 259, 242, 165, 174, 34, 36, 298, 92, 301, 237, 178, 82, 65, 295, 110, 311, 274, 235, 68, 56, 259, 180, 195, 52, 110, 68, 140, 71, 52, 296, 290, 115, 213, 82, 209, 209, 74, 178, 302, 131, 99, 205, 296, 309, 288, 180, 329, 71, 143, 58, 152, 273, 196, 7, 169, 88, 231, 331, 213, 181, 80, 249, 170, 246, 16, 127, 75, 276, 332, 174, 21, 180, 163, 78, 242, 312, 295, 199, 89, 142, 85, 195, 115, 119, 95, 94, 279, 290, 3, 33, 93, 284, 20, 47, 47, 78, 331, 271, 113, 179, 249, 331, 92, 324, 9, 71, 232, 46, 28, 289, 80, 28, 80, 134, 20, 280, 277, 48, 205, 107, 52, 320, 4, 191, 160, 182, 189, 227, 295, 115, 54, 195, 78, 292, 189, 273, 301, 69, 305, 36, 222, 167, 326, 106, 48, 45, 74, 61, 181, 311, 292, 270, 201, 34, 314, 218, 214, 92, 269, 18, 37, 151, 142, 209, 11, 227, 327, 198, 28, 272, 152, 22, 47, 143, 332, 253, 273, 35, 78, 130, 295, 223, 181, 329, 18, 238, 300, 186, 274, 99, 300, 322, 41, 185, 311, 288, 198, 2, 37, 83, 238, 133, 122, 178, 107, 106, 66, 238, 69, 90, 38, 109, 246, 278, 288, 250, 321, 269, 130, 28, 115, 122, 33, 185, 275, 99, 130, 99, 152, 268, 133, 249, 180, 30, 210, 201, 324, 29, 290, 143, 3, 269, 68, 106, 230, 1, 269, 29, 120, 259, 324, 328, 23, 243, 9, 61, 14, 118, 199, 146, 237, 14}

Returns: 11478648052

2_9) Problem Statement

Shiny has a company. There are N employees in her company. The employees are numbered 0 through N-1 in order in which they joined the company. Employee 0 is the only employee with no boss. Every other employee has precisely one direct boss in the company. You are given a int[] **superior** with N elements. Element 0 of **superior** will be -1 to denote that employee 0 has no boss. For each i between 1 and N-1, inclusive, element i of **superior** will be the number of the boss of employee i.

For each employee, their boss joined the company before them. Formally, for each i between 1 and N-1, inclusive, **superior**[i] will be between 0 and i-1, inclusive.

At the moment, the employees of Shiny's company cannot do anything useful. Shiny would like to change this. She decided that she will pay for the employees' training. More precisely, each employee will be trained to do two different types of work. (The two types of work may be different for different employees.) There are K types of work for which training is available. You are given a int[] **training** with K elements. For each i,**training**[i] is the cost of training any single employee to do work of type i. If multiple employees are trained to do the same work type, Shiny must pay for each of them separately.

Each employee of the company has their own *department*. The department of employee x is formed by employee x and all the employees such that x is their boss. Formally, for any y different from x, employee y belongs into the department of employee x if and only if **superior**[y]=x. Note that if **superior**[z]=y and **superior**[y]=x, employee z *does not* belong into the department of employee x.

Shiny likes *diverse* departments. A department is *diverse* if:

- Each employee in the department is doing something, and
- no two employees in the department are doing the same type of work.

When Shiny comes to inspect a department, the employees in the department try to choose their work so that the department will be diverse. If they can do that, Shiny says that the department is *good*. Shiny considers her company *good* if all N departments are good. (Note that the departments are not required to be diverse at the same time. A company is good as soon as each of its departments can be diverse at some point in time.)

Shiny now wants to choose, for each employee, the two work types they will be trained to do. Shiny wants to have a good company, and also to spend as little money as possible.

If it is possible for Shiny to have a good company, return the smallest possible total amount of money spent on training the employees. If it is impossible, return -1 instead.

Definition

Class: GoodCompanyDivOne
Method: minimumCost
Parameters: int[], int[]
Returns: int
Method signature: int minimumCost(int[] superior, int[] training)
(be sure your method is public)

Notes

- Each employee must learn to perform exactly two different work types (even though they might never need to do one of those two types of work).

Constraints

- **superior** will contain between 1 and 30 elements, inclusive.
- **superior**[0] will be -1.
- For each valid i>0, **superior**[i] will be between 0 and i-1, inclusive.
- **training** will contain between 2 and 30 elements, inclusive.
- Each element of **training** will be between 1 and 100, inclusive.

Examples

1)

```
{-1}
{1, 2}
Returns: 3
```

There is only one employee (employee 0) and two work types. Employee 0 has to be trained to do both work types. This costs 1+2 = 3. After the training, the company is clearly good.

2)

```
{-1, 0, 0}
{1, 2, 3}
Returns: 10
```

One optimal solution:

- Employee 0 learns work types 0 and 1 which costs 1+2 = 3.
- Employee 1 learns work types 0 and 1 which costs 1+2 = 3.
- Employee 2 learns work types 0 and 2 which costs 1+3 = 4.

The total cost is 3+3+4 = 10. The company is now good because:

- The department of employee 0 is formed by employees 0, 1, and 2. This department is good because they can choose work types 0, 1, and 2, respectively.
- The department of employee 1 is formed only by employee 1 and it is clearly

good.

- The department of employee 2 is formed only by employee 2 and it is clearly good.

3)

```
{-1, 0, 0, 0}
{1, 2, 3}
```

Returns: -1

There are only three work types, but there are four employees in employee 0's department. Therefore, this department can never be good.

4)

```
{-1, 0, 0, 2, 2, 2, 1, 1, 6, 0, 5, 4, 11, 10, 3, 6, 11, 7, 0, 2, 13, 14, 2, 10, 9, 11, 22, 10, 3}
{4, 2, 6, 6, 8, 3, 3, 1, 1, 5, 8, 6, 8, 2, 4}
```

Returns: 71

2_10) Problem Statement

Fox Ciel uses an alphabet that has **n** letters. She likes all the words that have the following properties:

1. Equal letters are never consecutive.
2. There is no subsequence of the form xyxy, where x and y are (not necessarily distinct) letters. Note that a subsequence doesn't have to be contiguous.
3. There is no longer word with properties 1 and 2.

Examples:

- Ciel does not like "ABBA" because there are two consecutive 'B's.
- Ciel does not like "THETOPCODER" because it contains the subsequence "TETE".
- Ciel does not like "ABACADA" because it contains the subsequence "AAAA". (Note that here x=y='A'.)
- Ciel does not like "ABCA" because "ABCBA" is longer.
- If **n**=1 and the one letter Ciel uses is 'A', then she likes the word "A".
- If **n**=2 and the two letters Ciel uses are 'A' and 'B', then she likes the words "ABA" and "BAB".

Given the int **n**, compute and return the number of words Ciel likes, modulo 1,000,000,007.

Definition

Class:	LongWordsDiv1
Method:	count
Parameters:	int
Returns:	int
Method signature:	int count(int n)
(be sure your method is public)	

Constraints

- **n** will be between 1 and 5000, inclusive.

Examples

1)

1

Returns: 1

The only word Ciel likes is "A" (assuming 'A' is the only letter in the alphabet).

2)

2

Returns: 2

The words Ciel likes are "ABA" and "BAB".

3)

5

Returns: 1080

4)

100

Returns: 486425238

Don't forget to compute the answer modulo 1,000,000,007.

2_11)Problem Statement

Elly is playing Scrabble with her family. The exact rules of the game are not important for this problem. You only need to know that Elly has a holder that contains a row of N tiles, and that there is a single letter on each of those tiles. (Tiles are small square pieces of wood. A holder is a tiny wooden shelf with room for precisely N tiles placed in a row.)

While Elly waits for the other players, she entertains herself in the following way. She slightly taps the table, causing the tiles on her holder jump a little and some of them switch places. Formally, suppose that Elly has N tiles. There are N positions on the holder, we will label them 0 through N-1 from left to right. When Elly taps the table, the tiles on her holder will form some permutation of their original order. You are given an int **maxDistance** with the following meaning: in the permutation that Elly produces by tapping the table, no tile will be more than **maxDistance** positions away from its original position (in either direction).

For example, suppose that before a tap the letters in Elly's holder formed the string "TOPCODER" when read from left to right. If **maxDistance** is 3, one possible string after Elly taps the table is "CODTEPOR". This can happen in the following way:

1. The letter 'T' at position 0 moves three positions to the right (to position 3).
2. The letter 'O' at position 1 remains on its initial position (position 1).
3. The letter 'P' at position 2 moves three positions to the right (to position 5).
4. The letter 'C' at position 3 moves three positions to the left (to position 0).
5. The letter 'O' at position 4 moves two positions to the right (to position 6).
6. The letter 'D' at position 5 moves three positions to the left (to position 2).
7. The letter 'E' at position 6 moves two positions to the left (to position 4).
8. The letter 'R' at position 7 remains on its initial position (position 7).

Note that the letter 'D' (at position 5) cannot move to position 1, because this would require it to move more than the maximal distance 3.

It turns out that the string "CODTEPOR" is the lexicographically smallest one Elly can get from "TOPCODER" with a single tap and **maxDistance** = 3. Now you want to write a program that, given the String **letters** and the int **maxDistance**, returns the lexicographically smallest string the girl can get after a single tap.

Definition

Class:	EllysScrabble
Method:	getMin
Parameters:	String, int
Returns:	String
Method signature:	String getMin(String letters, int maxDistance)

(be sure your method is public)

Notes

- Given two different strings A and B of equal length, the lexicographically smaller one is the one that contains a smaller character at the first position where they differ.

Constraints

- **letters** will contain between 1 and 50 characters, inclusive.
- **letters** will contain only uppercase letters from the English alphabet ('A'-'Z').
- **maxDistance** will be between 1 and 9, inclusive.

Examples

1)

"TOPCODER"

3

Returns: "CODTEPOR"

The example from the problem statement.

2)

"ESPRIT"

3

Returns: "EIPRST"

In this example the letters 'E', 'P', 'R', and 'T' stay on their initial places, and the letters 'S' and 'I' swap. Since the distance between them is exactly 3 (which also happens to be the maximal distance they can move), this is a valid final configuration.

3)

"BAZINGA"

8

Returns: "AABGINZ"

Note that the maximal distance may be greater than the number of letters Elly has. In such cases, the lexicographically smallest result is, obviously, the sorted sequence of letters.

4)

"ABCDEFGHIJKLMNOPQRSTUVWXYZ"

9

Returns: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

As the input letters are already sorted, we can assume they'll just fall back to their initial places.

6)

"GOODLUCKANDHAVEFUN"

7

Returns: "CADDGAHEOOFLUKNNUV"

7)

"AAAWDIUAOIWDESBEAIWODJAWDBPOAWDUISAWDOOPAWD"

6

Returns:
"AAAADDEIBWAEUIODWADSBIAJWODIAWDOPOAWDUOSPWW"

8)

"ABRACADABRA"

2

Returns: "AABARACBDAR"

1. The letter 'A' at position 0 remains on its initial place (position 0).
2. The letter 'B' at position 1 moves one position to the right (to position 2).
3. The letter 'R' at position 2 moves two positions to the right (to position 4).
4. The letter 'A' at position 3 moves two positions to the left (to position 1).
5. The letter 'C' at position 4 moves two positions to the right (to position 6).
6. The letter 'A' at position 5 moves two positions to the left (to position 3).
7. The letter 'D' at position 6 moves two positions to the right (to position 8).
8. The letter 'A' at position 7 moves two positions to the left (to position 5).
9. The letter 'B' at position 8 moves one position to the left (to position 7).
10. The letter 'R' at position 9 moves one position to the right (to position 10).
11. The letter 'A' at position 10 moves one position to the left (to position 9).

2_12) Problem Statement

Today, Bob is trying to count the colorings of a strange graph. It looks like a cycle of cycles.The graph has two types of edges -- *solid* edges and *dashed* edges.The edges between the vertices of the original cycles are all *solid* edges.The edges that connect cycles together are all *dashed* edges.

More precisely, the graph consists of N cycles. The cycles are labelled, in order, as C_0, C_1, ..., C_{N-1}. You are given a int[] **vertexCount** with N elements. Cycle C_i has **vertexCount**[i] vertices. The vertices of cycle C_i are labelled, in order, as $v_{i, 0}$, $v_{i, 1}$, ..., $v_{i, \mathbf{vertexCount}[i]-1}$. Note that the last vertex in this order is also connected to the first one. All of the edges that form these cycles are *solid* edges.

You are also given two int[]s **fromVertex** and **toVertex**, with N elements each. These int[]s describe the connection between cycle C_i and C_{i+1} as follows: The vertex in cycle C_i with label $v_{i, \mathbf{fromVertex}[i]}$ and the vertex in cycle C_{i+1} with label $v_{i+1, \mathbf{toVertex}[i]}$ will be connected with a *dashed* edge.

Bob has **K** distinct colors. He wants to know how many different ways there are to color the vertices of the graph under the following two rules:

- Vertices connected with a *solid* edge must be colored using a different color.
- Vertices connected with a *dashed* edge must be colored using the same color.

Two colorings are considered different if there is a vertex $v_{i, j}$ which has a different color in each coloring. Let C be the number of colorings that correspond to the given rules. As C can be extremely large, you should compute and return the value (C modulo 1,000,000,007).

Definition

Class:	CycleColoring
Method:	countColorings
Parameters:	int[], int[], int[], int
Returns:	int
Method signature:	int countColorings(int[] vertexCount, int[] fromVertex, int[] toVertex, int K)

(be sure your method is public)

Notes

- i+1 is considered modulo N - hence, references to cycle C_N are referring to cycle C_0.

Constraints

- **vertexCount** will contain between 1 and 50 elements, inclusive.
- **vertexCount**, **fromVertex**, and **toVertex** will all contain the same number of elements.
- Each element of **vertexCount** will be between 3 and 1,000,000, inclusive.
- Element **fromVertex**[i] will be between 0 and **vertexCount**[i] - 1, inclusive.
- Element **toVertex**[i] will be between 0 and **vertexCount**[i+1] - 1, inclusive.
- K will be between 2 and 1,000,000,000, inclusive.

Examples

1)

{3, 3}
{0, 0}
{0, 0}
3

Returns: 12

This graph consists of two cycles of length 3. There are two dashed edges, each connecting vertex 0 of one cycle to vertex 0 of the other cycle. These two vertices must share the same color. We have 3 possibilities for that color. Once we fix it, we have 2 possibilities how to color the rest of each cycle. Hence, the answer is 3*2*2 = 12.

2)

{6}
{4}
{1}
3

Returns: 12

This graph only has one cycle. Note that for N=1 there is still one dashed edge and it connects two vertices on the same cycle. The 12 valid colorings of the resulting graph correspond to the 12 valid colorings of the graph from Example 0.

3)

{3, 3}
{0, 1}
{1, 2}
3

Returns: 0

Vertices 0 and 2 on cycle 0 must both have the same color as vertex 1 on cycle 1. However, vertices 0 and 2 on cycle 0 are adjacent and therefore must have different colors. This is a contradiction. Therefore, there are no valid colorings of this graph.

4)

{9, 5}
{8, 3}
{0, 2}
8

Returns: 589124602

5)

{14, 15, 16, 17}
{5, 10, 4, 6}
{10, 3, 14, 10}
614

Returns: 818050159

6)

{482373, 283712, 883, 12834, 5, 5, 482734, 99912, 35881, 277590}
{59283, 49782, 0, 0, 3, 2, 84791, 40017, 2263, 461}
{150173, 709, 11108, 0, 4, 7, 5902, 712, 190232, 390331}
479360784

Returns: 763634309

2_13) Problem Statement

Consider the integers between low and high, inclusive. We are going to select a sequence of N integers from this range. The sequence is allowed to contain repeated elements, hence there are (high-low+1)^N possible sequences (where '^' denotes exponentiation).

Out of those sequences, we are only interested in the ones that have one additional property: the greatest common divisor (GCD) of their elements must be exactly K.

You are given the ints N, K, low, and high. Let X be the number of N-tuples described above. Because X can be very large, compute and return the value (X modulo 1,000,000,007).

Definition

Class: RandomGCD
Method: countTuples
Parameters: int, int, int, int
Returns: int
Method signature: int countTuples(int N, int K, int low, int high)
(be sure your method is public)

Constraints

- N, K and low will each be between 1 and 1,000,000,000, inclusive.
- high will be between low and 1,000,000,000, inclusive.
- The difference high - low will be less than or equal to 100,000.

Examples

1)

2

2

2

4

Returns: 3

There are 9 possible sequences: {(2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4), (4, 2), (4, 3), (4, 4)}. Out of these, 3 of them have the requested gcd of 2: {(2, 2), (2, 4), (4, 2)}. Hence, the answer is 3.

2)

2

100

2

4

Returns: 0

Sometimes no combinations yield the requested GCD.
3)

1

5

5

5

Returns: 1

Sometimes you select only one number.
4)

73824

17347

9293482

9313482

Returns: 0

5)

222

222

222

22222

Returns: 339886855

2_14) Problem Statement

Alberto is an aviation pioneer. He pilots an airplane called "14-bis". Initially, there are **F** units of fuel in the fuel tank of his airplane.

There are some flight missions Alberto may take. The missions all start and end in the same location, and he may do them in any order. However, he can only do each mission at most once. You are given two int[]s of the same length: **duration** and **refuel**. For each valid i:

- **duration**[i] is the amount of fuel consumed while running mission i
- After Alberto completes mission i and gets paid, he will buy **refuel**[i] units of fuel. This amount will always be strictly smaller than the amount consumed during the mission.

Alberto can only choose a mission if he has enough fuel for it. That is, at the beginning of the mission his fuel tank must have at least **duration**[i] units of fuel.

Compute and return the maximum number of missions Alberto can take.

Definition

Class:	AlbertoTheAviator
Method:	MaximumFlights
Parameters:	int, int[], int[]
Returns:	int

Method signature: int MaximumFlights(int F, int[] duration, int[] refuel)
(be sure your method is public)

Constraints

- **F** will be between 1 and 5,000 inclusive.
- **duration** and **refuel** will have between 1 and 50 elements, inclusive.
- Each element of **duration** will be between 1 and 5,000, inclusive.
- Each element of **refuel** will be between 0 and 5,000, inclusive.
- For each i, **refuel**[i] will be strictly smaller than **duration**[i].
- **duration** and **refuel** will contain the same number of elements.

Examples

1)

```
10
{10}
{0}
```

Returns: 1

There is only one mission. Alberto has enough fuel to take it, so the optimal solution is to take it.

2)

```
10
{8, 4}
{0, 2}
```

Returns: 2

3)

```
12
{4, 8, 2, 1}
{2, 0, 0, 0}
```

Returns: 3

4)

```
9
{4, 6}
{0, 1}
```

Returns: 2

5)

```
100
{101}
{100}
```

Returns: 0

There is only one mission. Alberto does not have enough fuel to take it. The answer is 0.

6)

```
1947
{2407, 2979, 1269, 2401, 3227, 2230, 3991, 2133, 3338, 356, 2535, 3859, 3267, 365}
{2406, 793, 905, 2400, 1789, 2229, 1378, 2132, 1815, 355, 72, 3858, 3266, 364}
```

Returns: 3

2_15) Problem Statement

Cat Snuke received a directed graph as a present.

You are given a String[] **graph**. The number of elements in **graph** is the number of vertices of the graph he received. If there is an edge from the vertex i (0-based) to the vertex j, the j-th character of the i-th element of **graph** is 'Y'. Otherwise the j-th character of the i-th element of **graph** is 'N'. Cat Snuke wonders how many walks of length L are there when L is very big. Return the minimal nonnegative integer K such that the number of walks of length L is O(L^K). If there is no such K, return -1 instead.

See notes for formal definitions of walks and big-O notation.

Definition

Class:	BigO
Method:	minK
Parameters:	String[]
Returns:	int
Method signature:	int minK(String[] graph)

(be sure your method is public)

Notes

- A sequence of vertices v_0, v_1, ..., v_L is called a *walk* of length L if for each i between 0 and L-1, inclusive, there exists an edge from vertex v_i to v_(i+1). Note that a walk may use each edge of the graph arbitrarily many times.
- The number of walks of length L is O(L^K) iff the following condition holds: There exists a constant C such that for any positive integer L the number of walks of length L is at most C * L^K.

Constraints

- **graph** will contain between 2 and 50 elements, inclusive.
- Each element of **graph** will contain exactly V characters, where V is the number of elements of **graph**.
- Each character in **graph** will be either 'Y' or 'N'.
- For each i, the i-th character of the i-th element of **graph** will be 'N'.

Examples

1)

```
{"NYY",
 "YNY",
 "YYN"}
```

Returns: -1

For this graph there are exactly 3*2^L walks of length L. There is no K such that 3*2^L is O(L^K).

2)

```
{"NYNNN",
 "NNYNN",
 "NNNYN",
 "NNNNY",
 "NNNNN"}
```

Returns: 0

The number of walks of length L is zero when L is big enough.

3)

```
{"NYNNN",
 "YNNNN",
 "NNNYN",
 "NNNNY",
 "NNYNN"}
```

Returns: 0

The number of walks of length L is 5 for any L.

4)

```
{"NYYYNYYYNNYYYYYYNYNN",
 "NNNNYNYYNNYYYNYYNYYN",
 "NYNNYYYNNNYYYYNYNYNN",
 "NYYNNYYYYNNNYYNNYNYY",
 "NYNYNNNNNNYYYYYNYYYN",
 "YNNNNNNYNNYNNYYYYYYY",
 "NNYYNNNNNYNYNYNNYNYY",
 "NNYNYYNNNNYNYNYYYYNN",
 "NYYNYYNNNYNNYYYNYNYN",
 "YYNNYNNYYNYNNNNNYNNN",
 "YYNYYNNYYYNYYNYNYYYY",
 "YYNNYYNYNYNNNNYNNNNY",
 "NNYYNYYYNNNNNYYYYYNY",
 "YNNNYNNNNYNNNNNYNNNY",
 "YYYYNYYNNYNNNNNYNNNN",
 "NYYYYNYNYYNNYNNNYNNY",
 "YYYYYNNNYYYYNYYYNNYN",
 "NNYNNYNYNYNNNNNNYNYN",
 "YYNYYNNNNNYNNYNYNNNY",
 "YYYYNYNYYNNYNYNYNNNN"}
```

Returns: -1

5)

```
{"NYNYYYNYYYNYYNYYNYYNYYNYNNYYYNNNYYNNNYNYYNYNNNYNY",
 "NNNNNNNNNNNNNNNNYNNNYNNNNNYNYNYNNNNNNNYNNNNNNNNNNNN",
 "NYNYYYYNYNYYNNYYYYYYYYYNYYNYYYYYNNYYYYYYYNNYYYY",
 "NYNNYYNNNNNNNNNNNYNNNYNYNNYNYNYYNYYNNYYYNNNNNNNNYYY",
 "NNNNNNNNNNNNYNNYYNNNYNNNNYNNYNNYNNYNNYNYNNNNNNNNYYN",
 "NNNNNNNNNNNNNNNNYNNNYNNNNNNNNNYNNNNNNNYNNNNNNNNNNNN",
 "YYNYNNNYNYYYYNYYYYYYYYYNYYYYYYYNYYNNYNYYYYYNNYNYY",
 "NYNYYNNNYYNNYNNYYYNNYYNYNYYNYYYNNNYNNYYYNNNNNNNNYYY",
 "NYNYYYNYNYNNNNNNYYYNNNNNNNNYYNYYYYNYYNNYYYNNNNNNNNYYY",
 "NYNNNYNNNNNNNNNYYNNNYNNNNNYNYNYNNYYNNYNYNNNYNNNNYN",
 "NYNYYYNNYYNYYNYNYYNYYYNYNYYNYYYYNYNNNNYYYYNYNNNNYNY",
 "NYNNYYNNNYNNYNNYYNNYYNNNYYNYYNNNYYNNNNYNNNYNNNYYY",
 "NNNNNNNNNNNNNNNYYNNNYNNNNNNNYNNYNYNNNYYNNNNNNNNNYNN",
 "YYYYNYYYYYYYNYYYYNYYYYYYYYYYYNYYNYYYNYYYYYNNYYNYN",
 "NYNYYNNYYYNYYNNYYYNYNYNYNYYNYYYYNYYNNYYYYYNYNNNNNY",
 "NYNNNNNNNNNNNNNNYNNNYNNNNNYNYNYYNYNNNNNNNNNNNNNNNN",
 "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNN",
 "NYNNNYNNNNNNNNNYYNNNNNNNNNNNNYNYYNYYNNYNNNNNNNNNYYN",
 "NYNYYYNNYYNYNNYYYYNYYYNYNYYYYNNYNNYNNYNYYYYNNNYYYY",
 "NNNYYYNNNNYNYYNNYYYNNYYNNNYYNYYYYNYYNNNYYNYNYNNNYYY",
 "NNNNNNNNNNNNNNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNNNNNN",
 "NYNNNYNNNNNNNNNYNNNNNNNNNNYYNYYYYNYYNNYNNNNNNNNNYYN",
 "YYNYYYNNYYNYYNYNYYNYNYNYNYYYYYYYNYNNNYYYYYNYNNYYYY",
 "NYNYYYNNNNNNYNNNYYNNYYNNNYYNYYYYNYYNNYYYNNNNNNNNYNY",
 "YNNNYYYYYYYYYNYYYYYYYYYNNYYNYNYNYNNYYYYYYYYNYNNYYYY",
 "NYNNNYNNNNNNNNNYYNNNYNNNNNYNYNYYNYYNNNNYNNNNNNNNYNN",
 "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNNNN",
 "NYNYYYNYYYNYNNNNYYNYYNNYNYYNYYYYNYYNNNNNYYNYNNYYNN",
 "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNYNNNNNNNNNNNN",
 "NYNNNYNNNNNNNNNYYNNNYNNNNNNNNYYNYYNNNNYNNNNNNNNYYY",
 "NNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN",
 "NNNNNNNNNNNNNNNNYNNNYNNNNNYNNNYNNNNNNNNNNNNNNNNNNNN",
 "NYNYYNNYYYNNNNNYYYNYYYNNNYYYYYYNYYNNYYYYNNYNNNNYY",
 "NNNNNNNNNNNNNNNYYNNNYNNNNNYNYNYYNNNNNYNNNNNNNNNNNN",
 "NYNNNNNNNNNNNNNNYNNNYNNNNNYNNNYYNNNNNNNNNNNNNNNNYNN",
 "NYNYYYYYNYYYYNYYYYYYNYYYNNNYYYYYNYYNYNYYYYYYYNYYYYNY",
 "YYNYYYNYYYNNNNNYNYYYYNYNNNYYYYYYNYYNNYYNYNNYNNYNNY",
 "NNNNNNNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNNNNNNNNNNNNN",
 "NYNNNYNNNNNNNNNYYNNNNYNNNNYNYNNYNNNNNYNYNNNNNNNNYNN",
 "NYNNNYNNNNNNNNNYYNNNNNNNNNYYNYNYNNYYNNNNNNNNNNNNYNN",
 "NYNYYYNYYYNYNNNNNYNNYYNYNNYNNYYYNYYNNYYYNNNNNNNNYYY",
 "NYNYYNNNNYNYYNNYYYNYYYNNNYYNYYYNNYYNNYYYNNNYNNNNYY",
 "NYNYNYNYYYNYYNYYNYNYNYYYNNYYYYYNNNNNYYYNYNYNNNYNYY",
 "NYNNNYNNNYNNNNNYYYNNYNNNNNYNYNNNNYYNNYNYNNNNNNNNYYN",
 "YYYNNYNYYYNYNNYYYYYNYYNYNYYYYYNYYNYNYYYYYNNYNNYNYN",
 "YYNNYYYYYYYNYNYYNYYYYYYYYYYYNYYYYNYYYNNYYYYNYYNNYYYY",
 "NYNNYYNYYYNNYNNYYYNYNYNYNYYNNYYNNYNNNYYYNYNYNNNNYY",
 "NNNNNNNNNNNNNNNNYNNNNNNNNNNNNNNYNNNNNNNNNNNNNNNNNNN",
 "NNNNNNNNNNNNNNNNNNNNNYNNNNNYNYNYNNNNNNNYNNNNNNNNNYNN",
 "NYNNYYNNNNNNNNNYYNNNNNNNNNYYNYNNYNYYNNYNYNNNNNNNNYNN"}
```

Returns: 7

2_16) Problem Statement

Andrew has a combination lock. The lock consists of multiple dials that are placed next to each other. Each dial contains the digits 0 through 9, in order. At any moment, exactly one of the digits on each dial is visible. The string formed by the currently visible digits is called the current combination.

The visible digit on a dial can be changed by rotating the dial up or down. Rotating the dial up changes 0 to 1, 1 to 2, and so on. Note that the digits on a dial wrap around: if we rotate up a dial that shows a 9, it will show a 0 again. Naturally, rotating the dial down changes the digit in the other direction.

We are able to rotate multiple dials at the same time, as long as they are next to each other. More precisely, in a single turn we can take an arbitrarily long segment of *consecutive* dials, and rotate all of them *one step* in the same direction (i.e., either all of them up, or all of them down).

For example, suppose that the current combination is "123". In one step, we can change it to many different combinations, including "012" (all three dials down), "234" (all three dials up), "133" (middle dial up), and "013" (first two dials down). Note that we cannot change "123" to "224" in a single step.

You are given two String[]s: **P** and **Q**. Concatenate the elements of **P** to get S. S is the current combination. Concatenate the elements of **Q** to get T. T is the secret combination that unlocks the lock. That is, to open the lock we need to change S into T by rotating some of the dials. Return the smallest number of steps needed.

Definition

Class:	CombinationLockDiv1
Method:	minimumMoves
Parameters:	String[], String[]
Returns:	int

Method signature: int minimumMoves(String[] P, String[] Q)
(be sure your method is public)

Constraints

- **P** and **Q** will each contain no more than 50 elements.
- Each element of **P** and **Q** will contain no more than 50 characters.
- S will contain at least 1 character.
- S will contain the same number of characters as T.
- Each character in S and T will be a digit ('0'-'9').

Examples

1)

{"123"}
{"112"}

Returns: 1

Rotate the last two dials down.

2)

{"1"}
{"7"}

Returns: 4

Rotate the dial down 4 times: from 1 to 0, from 0 to 9, from 9 to 8, and from 8 to 7.

3)

{"6","07"}
{"","60","7"}

Returns: 0

4)

{"1234"}
{"4567"}

Returns: 3

5)

{"020"}
{"909"}

Returns: 2

6)

{"4423232218340"}
{"6290421476245"}

Returns: 18

2_17) Problem Statement

Alien Fred wants to destroy the Earth. But before he does that, he wants to solve the following problem.

He has the set {1, 2, 3, ..., 2**N**}. He wants to split this set into two new sets A and B. The following conditions must all be satisfied:

- Each element of the original set must belong to exactly one of the sets A and B.
- The two new sets must have the same size. (I.e., each of them must contain exactly **N** numbers.)
- For each i from 1 to **N**, inclusive: Let A[i] be the i-th smallest element of A, and let B[i] be the i-th smallest element of B. The difference |A[i] - B[i]| must be greater than or equal to **K**.

You are given the two ints **N** and **K**. Let X be the total number of ways in which Fred can choose the sets A and B. Return the value (X modulo 1,000,000,007).

Definition

Class:	AlienAndSetDiv1
Method:	getNumber
Parameters:	int, int
Returns:	int
Method signature:	int getNumber(int N, int K)

(be sure your method is public)

Constraints

- **N** will be between 1 and 50, inclusive.
- **K** will be between 1 and 10, inclusive.

Examples

1)

2
2

Returns: 2

The initial set is {1, 2, 3, 4}. The following 6 pairs of subsets are possible in this case:

- A={1, 2} and B={3, 4}
- A={1, 3} and B={2, 4}
- A={1, 4} and B={2, 3}
- A={2, 3} and B={1, 4}
- A={2, 4} and B={1, 3}

- A={3, 4} and B={1, 2}

The first option and the last option are both valid. The other 4 options are invalid.
Note that order of the two sets matters: the option A={1,2} and B={3,4} differs from the option A={3,4} and B={1,2}.

2)

```
3
1
```

Returns: 20

3)

```
4
2
```

Returns: 14

4)

```
10
7
```

Returns: 40

2_18) Problem Statement

There are N cities in Treeland. The cities are numbered 1 through N. The roads in Treeland have the topology of a tree. That is, there are exactly N-1 bidirectional roads in Treeland, each connecting a pair of cities, and it is possible to travel between any two cities along the roads. For the purpose of this problem, all roads have the same length, and this length is our unit of distance.You are given two int[]s **A** and **B** that describe the tree. Each of these int[]s has N-1 elements. For each valid i, there is a road that connects the cities **A**[i] and **B**[i].

There are some foxes in Treeland. Currently, each of the foxes lives in a different city. You are given a String **haveFox** with N characters. For each i, character i of **haveFox** is 'Y' if there is a fox in city i+1, or 'N' otherwise.
The foxes would like to live closer to each other. To achieve that, some foxes (possibly all of them or none at all) will move to different cities. There are three constraints for the move:

1. After the foxes move, there must again be at most one fox in each city. (There are no restrictions on how the foxes travel while they are moving.)
2. After the foxes move, the set of cities inhabited by the foxes must be connected. That is, for any two different cities i and j that both contain a fox, all the cities on the (only) path between i and j must also contain a fox.
3. The total distance traveled by the foxes during the move must be as small as possible.

Return the smallest possible sum of distances traveled by the foxes.

Definition

Class:	FoxConnection
Method:	minimalDistance
Parameters:	int[], int[], String
Returns:	int
Method signature:	int minimalDistance(int[] A, int[] B, String haveFox)

(be sure your method is public)

Constraints

- N will be between 2 and 50, inclusive.
- **A** will contain exactly N-1 elements.
- Each element of **A** will be between 1 and N, inclusive.
- **B** will contain exactly N-1 elements.
- Each element of **B** will be between 1 and N, inclusive.
- The graph described by **A** and **B** will be a tree.
- **haveFox** will contain exactly N characters.

- Each character in **haveFox** will be either 'Y' or 'N'.

Examples

1)

{1,2,3}
{2,3,4}
"YNNY"

Returns: 2

Treeland looks as follows: 1-2-3-4. Two foxes are located in city 1 and city 4. One optimal solution is:

- The fox located in city 1 moves to city 2.
- The fox located in city 4 moves to city 3.

2)

{1,1,1,1}
{2,3,4,5}
"NYYYY"

Returns: 1

We can move any one of the foxes to city 1. After that the cities with foxes will form a connected set.

3)

{1,3,4,5,4}
{2,2,2,4,6}
"YNYNYY"

Returns: 2

4)

{1,2,3,4,5,6,7,8,9}
{2,3,4,5,6,7,8,9,10}
"YNNNYNYNNY"

Returns: 7

5)

{1,2,3,4,3,6,8,7}
{2,3,4,5,6,8,9,6}
"YNNYYNYYY"

Returns: 3

6)

{1}
{2}
"NY"

Returns: 0

There can be only 1 fox.

7)

```
{1}
{2}
"NN"
```

Returns: 0

And there can be no foxes.

2_19) Problem Statement

We are interested in pairs of strings (A, B) that satisfy the following conditions:

- A and B consist of exactly **n** characters each.
- Each character in A and B is one of the first **k** lowercase letters of the English alphabet.
- There exists a string C such that A + C = C + B. Here, + denotes string concatenation.

For example, if **n** = 3 and **k** = 4 then one valid pair of strings is ("aad", "daa"): both strings have length 3, only the first 4 letters are used in each of them, and C = "aa" shows that the third condition is satisfied as well.

You are given the ints **n** and **k**. Find the number of such pairs of strings, and return the number modulo 1,000,000,007.

Definition

Class:	PairsOfStrings
Method:	getNumber
Parameters:	int, int
Returns:	int
Method signature:	int getNumber(int n, int k)

(be sure your method is public)

Constraints

- **n** will be between 1 and 1,000,000,000, inclusive.
- **k** will be between 1 and 26, inclusive.

Examples

1)

2
2

Returns: 6

The following 6 pairs satisfy the conditions: ("aa", "aa"), ("ab", "ab"), ("ab", "ba"), ("ba", "ab"), ("ba", "ba"), ("bb", "bb").

2)

3
2

Returns: 20

3)

3
4

Returns: 184

4)

6
2

Returns: 348

5)

100
26

Returns: 46519912

2_20) Problem Statement

It's winter time! Time for snowmen to play some games.

Two snowmen are playing a game. In this game, the first snowman must choose a subset of the set {1, 2, ..., N}, and the second one must choose a subset of the set {1, 2, ..., M}. The following two conditions must be fulfilled:

- The two sets have an empty intersection.
- The XOR of all elements in the first set is less than the XOR of all elements in the second set.

You are given two ints **N** and **M**. Let X be the total number of different ways to choose the pair of sets. Return the value (X modulo 1,000,000,007).

Definition

Class:	WinterAndSnowmen
Method:	getNumber
Parameters:	int, int
Returns:	int
Method signature:	int getNumber(int N, int M)

(be sure your method is public)

Constraints

- **N** will be between 1 and 2000, inclusive.
- **M** will be between 1 and 2000, inclusive.

Examples

1)

```
2
2
Returns: 4
```

The following 4 pairs of subsets are possible in this case:

- {} and {1}
- {} and {2}
- {} and {1, 2}
- {1} and {2}

2)

```
1
1
```

Returns: 1

The only pair possible in this case is {} and {1}.

3)

3

5

Returns: 74

4)

7

4

Returns: 216

5)

47

74

Returns: 962557390

2_21) Problem Statement

Note that the memory limit for all tasks in this is 256 MB.

Fox Ciel has a matrix A that consists of N rows by M columns. Both N and M are even. Each element of the matrix is either 0 or 1. The rows of the matrix are numbered 0 through N-1 from top to bottom, the columns are numbered 0 through M-1 from left to right. The element in row i, column j is denoted A(i, j). You are given a String[] A that describes the matrix A. The character A[i][j] is '1' if A(i, j)=1 and it is '0' otherwise.

A palindrome is a string that reads the same forwards and backwards. For example, "1001" and "0111001110" are palindromes while "1101" and "000001" are not.

Some rows and some columns in Ciel's matrix may be palindromes. For example, in the matrix below both row 0 and column 3 are palindromes. (Row 0 is the palindrome "0000", column 3 is the palindrome "0110".)

```
0000
0011
0111
1110
```

You are also given two ints: rowCount and columnCount. Ciel wants her matrix A to have at least rowCount rows that are palindromes, and at the same time at least columnCount columns that are palindromes. If this is currently not the case, she can change A by changing some of the elements (from '0' to '1' or vice versa). Compute and return the smallest possible number of elements she needs to change in order to reach her goal.

Definition

Class: PalindromeMatrix
Method: minChange
Parameters: String[], int, int
Returns: int
Method signature: int minChange(String[] A, int rowCount, int columnCount)
(be sure your method is public)

Constraints
- N and M will be between 2 and 14, inclusive.
- N and M will be even.
- A will contain N elements.
- Each element of A will contain M characters.
- Each character of A will be either '0' or '1'.
- rowCount will be between 0 and N.
- columnCount will be between 0 and M.

Examples
1)

{"0000"
,"1000"
,"1100"
,"1110"}

2
2
Returns: 1
An optimal solution is to change A(3, 0) to 0. Then we will have palindromes in two rows (0 and 3), and in two columns (0 and 3).

2)
{"0000"
,"1000"
,"1100"
,"1110"}
3
2
Returns: 3
This is similar to the previous example, but in this case we must have three row palindromes. An optimal solution is to change A(1, 0), A(2, 0) and A(3, 0) to 0.

3)
{"01"
,"10"}
1
1
Returns: 1

4)
{"1110"
,"0001"}
0
0
Returns: 0
Here, we do not have to change A at all.
5)

{"01010101"
,"01010101"
,"01010101"
,"01010101"
,"01010101"
,"01010101"
,"01010101"
,"01010101"}

2

3

Returns: 8

6)

```
{"000000000000"
,"011101110111"
,"010001010101"
,"010001010101"
,"011101010101"
,"010101010101"
,"010101010101"
,"011101110111"
,"000000000000"
,"000000000000"}
```

5

9

Returns: 14

7)

```
{"11111101001110"
,"11000111111111"
,"00010101111001"
,"10110000111111"
,"10000011010010"
,"10001101101101"
,"00101010000001"
,"10111010100100"
,"11010011110111"
,"11100010110110"
,"00100101010100"
,"01001011001000"
,"01010001111010"
,"10100000010011"}
```

6

8

Returns: 31

2_22) Problem Statement

Little Petya likes computer games a lot. Recently he has received one as a gift from his mother. He has reached the final level of this game which can be represented as a rectangular grid described by String[] **grid**. Every cell of this grid is described by one of the following characters:

- '#' - represents a cell that contains a wall.
- '.'(period) - represents an empty cell.
- '1'..'9', 'a'..'z', 'A'..'Z' - represent a cell that contains a charging station with a given power. Digits '1'..'9' represent numbers from 1 to 9, characters 'a'..'z' represent numbers from 10 to 35, characters 'A'..'Z' represent numbers from 36 to 61.
- '*' - represents the starting cell.
- '$' - represents the target cell.

There is exactly one starting cell and exactly one target cell. Both of these cells are empty.

Petya's character is initially located in the starting cell. Every second he can either perform a move or stay in the same cell. If he decides to make a move, he chooses one of the four directions (up, right, down, left) and a positive integer *k*. After that the character jumps by *k* cells in the chosen direction. Note that he can't jump over the walls and the destination cell must not contain a wall as well. This action costs *k^2* units of energy. For each second Petya's character spends on a cell with a charging station without movement, his energy increases by the power of this charging station.

You are also given ints **E** and **T**. At the beginning Petya's character has **E** units of energy. If the level of energy becomes strictly less than 0 the character dies. Petya's goal is to reach the target cell in not more than **T** seconds. Return the maximal amount of energy Petya can have at the end, or -1 if he can't reach the destination within **T** seconds while keeping his character alive. Note that the game ends after exactly **T** seconds from the beginning, not when the character steps on the target cell.

Definition

Class:	JumpingOnTheGrid
Method:	maxEnergy
Parameters:	String[], int, int
Returns:	long

Method signature: long maxEnergy(String[] grid, int E, int T)
(be sure your method is public)

Constraints

- **grid** will contain between 1 and 25 elements, inclusive.
- All elements of **grid** will contain the same number of characters.
- Each element of **grid** will contain between 1 and 25 characters, inclusive.

- **grid** will contain only characters '.', '#', '*', '$', '1'..'9', 'a'..'z', 'A'..'Z'.
- There will be exactly one character '*' and exactly one character '$' in the grid.
- **T** will be between 1 and 1 000 000 000, inclusive.
- **E** will be between 1 and 1 000 000 000, inclusive.

Examples

1)

```
{"*$.",
 "#1.",
 "..."}
1
1
```

Returns: 0

Petya moves his character directly to the target cell. Note that with power 0 the character is still alive.

2)

```
{"*$.",
 "#.1",
 "..."}
1
1000000000
```

Returns: 0

Even though **T** is large now, after the first step Petya has no more power, so he cannot reach the charging station.

3)

```
{"*$.",
 "#2.",
 "..."}
2
10
```

Returns: 13

The optimal strategy is to move one step to the right, then one step down, then to wait for 7 seconds on the charging station, and finally to move one step up.

4)

```
{"*$.",
 "#aA",
 "..."}
2
10
```

Returns: 151

5)

```
{"*..Z",
 "##..",
 "#...",
 "...$"}
8
4
```

Returns: 55

One of the possible optimal strategies is to jump by 2 cells to the right, then jump 1 more cell to the right to reach the charging station. After that we can wait for 1 second there and then jump by 3 cells directly to the target cell.

6)

```
{"#*#..",
 "####.",
 "ZZZZZ",
 "...$."}
1000000000
1000000000
```

Returns: -1

7)

```
{"#*#..",
 "#.##.",
 "ZZZZZ",
 "...$."}
4
1000000000
```

Returns: 60999999812

2_23) Problem Statement

Little Elephant from the Zoo of Lviv likes strings that consist of characters 'R', 'G' and 'B'. You are given a String[] **list**. Concatenate all elements of **list** to get the string S of length N. The characters in S are numbered from 0 to N-1, inclusive.

You are also given int **minGreen**. Little Elephant thinks that string is *nice* if and only if it contains a substring of at least **minGreen** consecutive characters 'G'. For example, if **minGreen** = 2, then strings "GG", "GGRGBB" and "RRRGRBGGG" are nice, but "G", "GRG", "BBRRGRGB" are not.

Little Elephant wants to know the number of quadruples of integers (a,b,c,d) such that:

- Each of a, b, c, d is between 0 and N-1, inclusive.
- a <= b and c <= d. (Both a..b and c..d are valid ranges of values.)
- b < c. (The entire range a..b lies before the range c..d.)
- The string T = S[a..b] + S[c..d] is nice.

Compute and return the number of such quadruples (a,b,c,d).

Definition

Class:	LittleElephantAndRGB
Method:	getNumber
Parameters:	String[], int
Returns:	long
Method signature:	long getNumber(String[] list, int minGreen)

(be sure your method is public)

Constraints

- **list** will contain between 1 and 50 elements, inclusive.
- Each element of **list** will contain between 1 and 50 characters, inclusive.
- Each element of **list** will consist only of characters 'R', 'G' and 'B'.
- **minGreen** will be between 1 and 2500, inclusive.

Examples

1)

```
{"GRG"}
2
Returns: 1
```

The only valid quadruple is (0,0,2,2). For this quadruple we have S[a..b]="G" and S[c..d]="G", thus T = "GG".

2)

{"GG", "GG"}
3

Returns: 9

There are 3 valid quadruples such that T="GGGG" and 6 quadruples such that T="GGG".

3)

{"GRBGRBBRG"}
2

Returns: 11

One of the valid quadruples is (0,0,3,5). This quadruple corresponds to the nice string T="GGRB".

4)

{"RRBRBBRRR", "R", "B"}
1

Returns: 0

5)

{"GRGGGRBRGG", "GGGGGGGG", "BRGRBRB"}
4

Returns: 12430

2_24) Problem Statement

The pony Rainbow Dash wants to choose her pet. There are N animals who want to be her pet. Rainbow Dash numbered them 0 through N-1.To help her make the decision, Rainbow Dash decided to organize a relay race for the animals. The race track is already known, and for each animal we know how fast it is. More precisely, you are given int[]s **A** and **B** with the following meaning: For each i, the animal number i will take between **A**[i] and **B**[i] seconds (inclusive) to complete the track.

For the race the animals will be divided into two competing teams. This is a relay race, so the team members of each team will all run the same track, one after another -- when the first team member finishes, the second one may start, and so on. Thus the total time in which a team completes the race is the sum of the times of all team members. Note that we can use the estimates given by **A** and **B** to estimate the total time for any team of animals.

Given two teams S and T, the value maxdiff(S,T) is defined as the largest possible difference in seconds between the time in which team S finishes the course and the time in which team T finishes the course.

Rainbow Dash now needs to assign each of the animals to one of the two competing teams. She wants to see a close competition, so she wants the teams to finish as close to each other as possible. Formally, she wants to divide all animals into teams S and T in a way that minimizes maxdiff(S,T). Return the smallest possible value of maxdiff(S,T).

Definition

Class:	MayTheBestPetWin
Method:	calc
Parameters:	int[], int[]
Returns:	int
Method signature:	int calc(int[] A, int[] B)

(be sure your method is public)

Notes

- The teams are not required to contain the same number of animals.

Constraints

- **A** will contain between 2 and 50 elements, inclusive.
- **A** and **B** will contain the same number of elements.
- Each element of **A** will be between 1 and 10,000, inclusive.
- Each element of **B** will be between 1 and 10,000, inclusive.
- For each i, **B**[i] will be greater than or equal to **A**[i].

Examples

1) {3,4,4,7}
   {3,4,4,7}

Returns: 2

In this test case we know the exact time in which each of the animals completes the track. An optimal solution is to choose teams S={0,3} and T={1,2}. Then team S will certainly complete the track in 3+7 = 10 seconds, and team T in 4+4 = 8 seconds. Thus, maxdiff(S,T)=2.

2)

{1,3,5,4,5}

{2,5,6,8,7}

Returns: 5

Here one of the optimal solutions is S={2,3} and T={0,1,4}. For these two teams we have maxdiff(S,T)=5. For example, it is possible that S will complete the track in 6+8 = 14 seconds, and T will complete it in 1+3+5 = 9 seconds. It is also possible that S will complete the track up to 5 seconds before T does.

3)

{2907,949,1674,6092,8608,5186,2630,970,1050,2415,1923,2697,5571,6941,8065,4710,716,756,5185,134
1,993,5092,248,1895,4223,1783,3844,3531,2431,1755,2837,4015}

{7296,6954,4407,9724,8645,8065,9323,8433,1352,9618,6487,7309,9297,8999,9960,5653,4721,7623,6017
,7320,3513,6642,6359,3145,7233,5077,6457,3605,2911,4679,5381,6574}

Returns: 52873

2_25)Problem Statement

Little Elephant from the Zoo of Lviv likes permutations. A permutation of size **N** is a sequence (a_1, ..., a_N) that contains each of the numbers from 1 to **N** exactly once. For example, (3,1,4,5,2) is a permutation of size 5.

Given two permutations A = (a_1, ..., a_N) and B = (b_1, ..., b_N), the value magic(A,B) is defined as follows: magic(A,B) = max(a_1,b_1) + max(a_2,b_2) + ... + max(a_N,b_N).

You are given the int **N**. You are also given another int **K**. Let X be the number of pairs (A,B) such that both A and B are permutations of size **N**, and magic(A,B) is greater than or equal to **K**. (Note that A and B are not required to be distinct.) Return the value (X modulo 1,000,000,007).

Definition

Class:	LittleElephantAndPermutationDiv1
Method:	getNumber
Parameters:	int, int
Returns:	int
Method signature:	int getNumber(int N, int K)

(be sure your method is public)

Constraints

- **N** will be between 1 and 50, inclusive.
- **K** will be between 1 and 2500, inclusive.

Examples

1)

```
1
1
Returns: 1
```

For **N**=1 the only pair of permutations is ((1), (1)). The magic of this pair of permutations is 1, so we count it.

2)

```
2
1
Returns: 4
```

Now there are four possible pairs of permutations. They are shown below, along with their magic value.

- magic((1,2), (1,2)) = 1+2 = 3

- magic((1,2), (2,1)) = 2+2 = 4
- magic((2,1), (1,2)) = 2+2 = 4
- magic((2,1), (2,1)) = 2+1 = 3

In all four cases the magic value is greater than or equal to **K**.

3)

```
3
8
```

Returns: 18

4)

```
10
74
```

Returns: 484682624

5)

```
50
1000
```

Returns: 539792695

2_26)Problem Statement

You are exploring a dungeon. In the dungeon you found some locked doors. Each locked door has some red and some green keyholes (zero or more of each kind). In order to open a door, you must insert fitting keys into all its keyholes simultaneously. All the keys used to open a door break in the process of opening it and you have to throw them away. However, each door hides a small chamber that contains some new keys for you. Once you open the door, you may take all of those keys and possibly use them to open new doors. (Of course, it only makes sense to open each door at most once. If you open the same door again, there will be no new keys for you.)

There are three kinds of keys: red, green, and white ones. Each red key fits into any red keyhole. Each green key fits into any green keyhole. Each white key fits into any keyhole (both red and green ones). You are given int[]s **doorR**, **doorG**, **roomR**, **roomG**, and **roomW**. These five int[]s all have the same length. For each valid i, the values at index i describe one of the doors you found: the door has **doorR**[i] red and **doorG**[i] green keyholes, and upon opening it you gain new keys: **roomR**[i] red ones, **roomG**[i] green ones, and **roomW**[i] white ones.

You are also given the int[] **keys** with three elements: **keys**[0] is the number of red keys, **keys**[1] the number of green keys, and **keys**[2] the number of white keys you have at the beginning. Your goal is to have as many keys as possible at the moment when you decide to stop opening doors. (The colors of the keys do not matter.) You are allowed to open the doors in any order you like, and to choose the keys used to open each of the doors. You are also allowed to stop opening doors whenever you are satisfied with your current number of keys. Compute and return the maximal total number of keys you can have at the end.

Definition

Class:	KeyDungeonDiv1
Method:	maxKeys
Parameters:	int[], int[], int[], int[], int[], int[]
Returns:	int
Method signature:	int maxKeys(int[] doorR, int[] doorG, int[] roomR, int[] roomG, int[] roomW, int[] keys)

(be sure your method is public)

Constraints

- **doorR**, **doorG**, **roomR**, **roomG** and **roomW** will each contain between 1 and 12 elements, inclusive.
- **doorR**, **doorG**, **roomR**, **roomG** and **roomW** will contain the same number of elements.
- Each element of **doorR**, **doorG**, **roomR**, **roomG** and **roomW** will be between 0 and 10, inclusive.
- **keys** will contain exactly 3 elements.

- Each element of **keys** will be between 0 and 10, inclusive.

Examples

1)

```
{1, 2, 3}
{0, 4, 9}
{0, 0, 10}
{0, 8, 9}
{1, 0, 8}
{3, 1, 2}
```

Returns: 8

First you have 3 red keys, 1 green key, 2 white keys. You can end with 8 keys as follows:

- First, you open door 0 using 1 red key. From the opened chamber you gain 1 white key, so currently you have 2 red keys, 1 green key, and 3 white keys.
- Second, you open door 1 using 2 red keys, 1 green key, and 3 white keys (all of them into green locks). Immediately after opening the door you have no keys: all the ones you had were just used and thus they broke. However, the chamber you just opened contains 8 green keys.

You can't end with more than 8 keys, so you should return 8.

2)

```
{1, 1, 1, 2}
{0, 2, 3, 1}
{2, 1, 0, 4}
{1, 3, 3, 1}
{1, 0, 2, 1}
{0, 4, 0}
```

Returns: 4

You have only green keys, while each door has at least 1 red keyhole. So you cannot open any of the doors.

3)

```
{2, 0, 4}
{3, 0, 4}
{0, 0, 9}
{0, 0, 9}
{8, 5, 9}
{0, 0, 0}
```

Returns: 27

Initially you have no key at all, but door 1 also has no key hole. Therefore, you can start by opening door 1.

4)

```
{5, 3, 0, 0}
{0, 1, 2, 1}
{0, 9, 2, 4}
{2, 9, 2, 0}
{0, 9, 1, 1}
{1, 1, 0}
```

Returns: 32

5)

```
{9,5,10,8,4,3,0,8,4,1,3,9}
{9,10,0,8,9,4,3,8,1,8,10,4}
{1,2,0,2,3,3,5,3,1,3,0,5}
{5,2,5,0,5,2,3,4,0,0,5,2}
{1,5,1,2,0,4,4,0,3,3,1,3}
{5,0,1}
```

Returns: 16

2_27) Problem Statement

Let A be a sequence of integers. The *LISNumber* of A is the smallest positive integer L such that A can be obtained by concatenating L strictly increasing sequences. For example, the LISNumber of A = {1, 4, 4, 2, 6, 3} is 4, since we can obtain A as {1, 4} + {4} + {2, 6} + {3}, and there is no way to create A by concatenating 3 (or fewer) strictly increasing sequences. The LISNumber of a strictly increasing sequence is 1.

You have N types of cards. For each i, 0 <= i < N, you have **cardsnum**[i] cards of the i-th type. Each card of the i-th type contains the number i.

You are given the int[] **cardsnum** and an int **K**. You want to arrange all the cards you have into a row in such a way that the resulting sequence of integers has LISNumber **K**. Note that you must use all the cards you have, you can only choose their order.

Let X be the number of different valid sequences you can produce. Compute and return the number X, modulo 1,000,000,007.

Definition

Class:	LISNumber
Method:	count
Parameters:	int[], int
Returns:	int

Method signature: int count(int[] cardsnum, int K)
(be sure your method is public)

Constraints

- **cardsnum** will contain between 1 and 36 elements, inclusive.
- Each element of **cardsnum** will be between 1 and 36, inclusive.
- **K** will be between 1 and 1296, inclusive.

Examples

1)

```
{1, 1, 1}
2
Returns: 4
```

In this case, there are 3 types of cards and you have one of each. Among the 6 sequences you can make, the following 4 have LISNumber 2:

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

2)

```
{2}
1
```

Returns: 0

The only sequence you can make is {0, 0} and its LISNumber is 2.

3)

```
{36, 36, 36, 36, 36}
36
```

Returns: 1

Only the sequence {0, 1, 2, 3, 4, 0, 1, 2, 3, 4, ... (36 times) ... } has LISNumber 36.

4)

```
{3, 2, 11, 5, 7}
20
```

Returns: 474640725

5)

```
{31, 4, 15, 9, 26, 5, 35, 8, 9, 7, 9, 32, 3, 8, 4, 6, 26}
58
```

Returns: 12133719

6)

```
{27, 18, 28, 18, 28, 4, 5, 9, 4, 5, 23, 5,
 36, 28, 7, 4, 7, 13, 5, 26, 6, 24, 9, 7,
 7, 5, 7, 24, 7, 9, 36, 9, 9, 9, 5, 9}
116
```

Returns: 516440918

2_28) Problem Statement

Once upon a time, there was a civilization called Ruritania. It had *n* building sites numbered from 0 to *n*-1. There were various types of buildings such as libraries, markets, and palaces. Each building type was assigned an integer from 1 to 50. The building at site *i* (0-based index) was of type **kind**[i].

With the passing of millennia, Ruritania declined and its building sites were covered in sand, concealing all the buildings. Due to wind and terrain, the depth of the sand varied. The building at site *i* (0-based index) was buried **depth**[i] meters below the surface. Recently, an intrepid archeologist excavated **K** building sites using a machine that could dig to a maximum depth of *D* meters. Thus, he only discovered buildings that had been buried at most *D* meters below the surface.

You are given int[]s **kind**, **depth**, and **found** as well as the int **K**. The types of buildings discovered by the excavation are given by the int[] **found**, which contains at most one value for each building type even if several buildings of a type were excavated.

Return the number of **K**-tuples of sites that could have been excavated to arrive at the given values. If the given information is not consistent with any configuration of building sites, return 0.

Definition

Class:	Excavations
Method:	count
Parameters:	int[], int[], int[], int
Returns:	long
Method signature:	long count(int[] kind, int[] depth, int[] found, int K)

(be sure your method is public)

Constraints

- **kind** will contain N elements, where N is between 1 and 50, inclusive.
- Each element of **kind** will be between 1 and 50, inclusive.
- **depth** will contain N elements.
- Each element of **depth** will be between 1 and 100,000, inclusive.
- **found** will contain between 1 and 50 elements, inclusive.
- Each element of **found** will occur in **kind** at least once.
- The elements of **found** will be distinct.
- **K** will be between the number of elements in **found** and N, inclusive.

Examples

1)

{1, 1, 2, 2}
{10, 15, 10, 20}
{1}
2

Returns: 3

There are four building sites. Two have buildings of type 1 and two have buildings of type 2. The type 1 buildings are at depths 10 and 15. The type 2 buildings are at depths 10 and 20. The archeologist has excavated two sites and discovered only type 1 buildings. He must have excavated one of three possible pairs of sites:

- Sites 0 and 1. The archeologist's machine excavates to a maximum depth *D* of at least 10.
- Sites 0 and 3. The machine excavates to a maximum depth *D* that falls in the interval [10, 20).
- Sites 1 and 3. The machine excavates to a maximum depth that falls in the interval [15, 20).

The other pairs of sites could not have been excavated. For example, the archeologist could not have excavated sites 0 and 2, because he would have found either none or both of the buildings.

2)

{1, 1, 2, 2}
{10, 15, 10, 20}
{1, 2}
2

Returns: 4

The archeologist could have chosen any pair of sites containing a type 1 and a type 2 building. With a large enough value of *D*, he could have excavated both.

3)

{1, 2, 3, 4}
{10, 10, 10, 10}
{1, 2}
3

Returns: 0

The archeologist cannot have excavated three sites, or else he would have found three types of buildings.

4)

{1, 2, 2, 3, 1, 3, 2, 1, 2}
{12512, 12859, 125, 1000, 99, 114, 125, 125, 114}
{1, 2, 3}
7

Returns: 35

5)

{50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50}

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3}

{50}

18

Returns: 9075135300

2_29) Problem Statement

There are some deer in the zoo. Each deer has two antlers. You are given int[]s **antler1** and **antler2**. These two int[]s will contain the same number of elements. For each index i, **antler1**[i] and **antler2**[i] are the weights of the two antlers of one of the deer.

You are also given an int **capacity**. A deer is unbalanced if the weight difference between his antlers is strictly more than **capacity**. You decided to perform some operations on the deer. Your goal is to make all deer balanced. In each operation, you can choose some two antlers (each on a different deer) and swap them. Return the minimal number of operations required to make all deer balanced. If this is impossible, return -1 instead.

Definition

Class:	AntlerSwapping
Method:	getmin
Parameters:	int[], int[], int
Returns:	int
Method signature:	int getmin(int[] antler1, int[] antler2, int capacity)

(be sure your method is public)

Constraints

- **antler1** and **antler2** will contain the same number of elements.
- **antler1** and **antler2** will contain between 1 and 16 elements, inclusive.
- Each element of **antler1** and **antler2** will be between 1 and 1,000,000, inclusive.
- **capacity** will be between 0 and 1,000,000, inclusive.

Examples

1)

```
{3, 2, 2}
{3, 5, 5}
0
Returns: 1
```

There are three deer in the zoo. As **capacity**=0, a deer is only balanced if he has two antlers of exactly equal weight. Currently, deer 0 is balanced and the other two are not. We can fix that in a single operation. For example, we can swap deer 1's antler 1 and deer 2's antler 2. After this operation, deer 1 will have two antlers that weigh 5 each, and deer 2 will have two antlers that weigh 2 each.

2)

```
{4, 2, 6, 4, 8, 5, 2, 3}
{3, 4, 5, 2, 8, 5, 7, 6}
1
```

Returns: 2

One of the optimal ways is as follows:

- Swap deer 1's antler with weight 2 and deer 3's antler with weight 4.
- Swap deer 6's antler with weight 7 and deer 7's antler with weight 3.

3)

```
{12, 34, 56, 78}
{1234, 2345, 3456, 4567}
100
```

Returns: -1

If it is impossible to achieve the goal, return -1.

4)

```
{47, 58, 2013}
{49, 55, 2013}
3
```

Returns: 0

5)

```
{4, 1, 7, 5, 7, 8, 2, 1, 3, 1, 7, 5, 9, 4, 9, 1}
{10, 6, 5, 3, 1, 8, 4, 4, 4, 7, 1, 4, 6, 5, 10, 10}
1
```

Returns: 7

2_30) Problem Statement

There are some deer in the zoo. Each deer has two antlers. You are given int[]s **antler1** and **antler2**. These two int[]s will contain the same number of elements. For each index i, **antler1**[i] and **antler2**[i] are the weights of the two antlers of one of the deer. You are also given an int **capacity**. A deer is unbalanced if the weight difference between his antlers is strictly more than **capacity**. You decided to perform some operations on the deer. Your goal is to make all deer balanced. In each operation, you can choose some two antlers (each on a different deer) and swap them. Return the minimal number of operations required to make all deer balanced. If this is impossible, return -1 instead.

Definition

Class:	AntlerSwapping
Method:	getmin
Parameters:	int[], int[], int
Returns:	int
Method signature:	int getmin(int[] antler1, int[] antler2, int capacity)

(be sure your method is public)

Constraints

- **antler1** and **antler2** will contain the same number of elements.
- **antler1** and **antler2** will contain between 1 and 16 elements, inclusive.
- Each element of **antler1** and **antler2** will be between 1 and 1,000,000, inclusive.
- **capacity** will be between 0 and 1,000,000, inclusive.

Examples

1)

```
{3, 2, 2}
{3, 5, 5}
0
Returns: 1
```

There are three deer in the zoo. As **capacity**=0, a deer is only balanced if he has two antlers of exactly equal weight. Currently, deer 0 is balanced and the other two are not. We can fix that in a single operation. For example, we can swap deer 1's antler 1 and deer 2's antler 2. After this operation, deer 1 will have two antlers that weigh 5 each, and deer 2 will have two antlers that weigh 2 each.

2)

```
{4, 2, 6, 4, 8, 5, 2, 3}
{3, 4, 5, 2, 8, 5, 7, 6}
1
```

Returns: 2

One of the optimal ways is as follows:

- Swap deer 1's antler with weight 2 and deer 3's antler with weight 4.
- Swap deer 6's antler with weight 7 and deer 7's antler with weight 3.

3)

```
{12, 34, 56, 78}
{1234, 2345, 3456, 4567}
100
```

Returns: -1

If it is impossible to achieve the goal, return -1.

4)

```
{47, 58, 2013}
{49, 55, 2013}
3
```

Returns: 0

5)

```
{4, 1, 7, 5, 7, 8, 2, 1, 3, 1, 7, 5, 9, 4, 9, 1}
{10, 6, 5, 3, 1, 8, 4, 4, 4, 7, 1, 4, 6, 5, 10, 10}
1
```

Returns: 7

2_31) Problem Statement

Gobble City has a tree topology: There are N intersections, numbered 0 through N-1. The intersections are connected by N-1 bidirectional roads in such a way that it is possible to travel between any two intersections. More precisely, for each pair of intersections there is a unique path (a sequence of roads; see Notes for a formal definition) that connects them. You are given a int[] **roads** that describes the roads: for each i (0 <= i <= N-2), we have 0 <= **roads**[i] <= i and there is a road between intersections numbered **roads**[i] and i+1.

On each road there is a single lamp. The lamps are numbered 0 through N-2. For each i, lamp i is on the road that connects **roads**[i] and i+1. You are given a String **initState** that describes the initial states of all lamps: **initState**[i]='1' means that lamp i is initially on, and **initState**[i]='0' means that it is off.

Now you are at the Control Center for these lamps. The only way in which you can operate the lamps looks as follows: You enter the numbers of two intersections (X and Y) into the Control Center computer, and the computer toggles the state of all lamps on the path between X and Y. (Toggling the state of a lamp means that if the lamp was off it is now on, and vice versa.) You can perform arbitrarily many such operations, one after another.Some of the lamps are important to you. You are given this information in the String **isImportant**: **isImportant**[i]='1' means that lamp i is important, and **isImportant**[i]='0' means that it is not.Your goal is to turn on all important lamps at the same time. (We do not care about the state of the remaining lamps.) Return the minimum number of operations needed to achieve the goal.

Definition

Class:	TurnOnLamps
Method:	minimize
Parameters:	int[], String, String
Returns:	int
Method signature:	int minimize(int[] roads, String initState, String isImportant)
	(be sure your method is public)

Notes

- In your solution some lamps may be toggled multiple times.
- A path between two intersection a and b is a sequence of intersections (v_0=a, v_1, v_2, ..., v_k=b) such that all v_0, v_1, ..., v_k are pairwise distinct and for each i=0..k-1, there exists a road between v_i and v_{i+1}.

Constraints

- **roads** will contain N-1 elements, where N is between 2 and 50, inclusive.
- For each i, the i-th (0-based index) element of **roads** will be between 0 and i, inclusive.

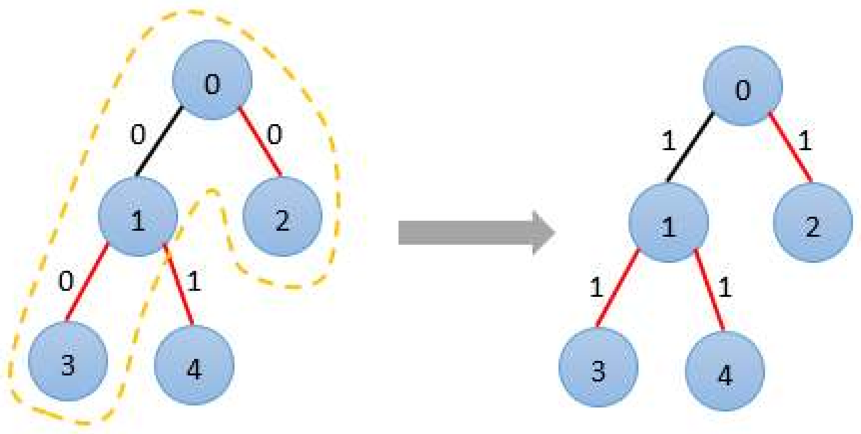
- **initState** will be N-1 characters long.
- **initState** will only contain the characters '0' and '1'.
- **isImportant** will be N-1 characters long.
- **isImportant** will only contain the characters '0' and '1'.

Examples

1){0,0,1,1}
"0001"
"0111"

Returns: 1

The figure below illustrates this test case and its optimal solution.

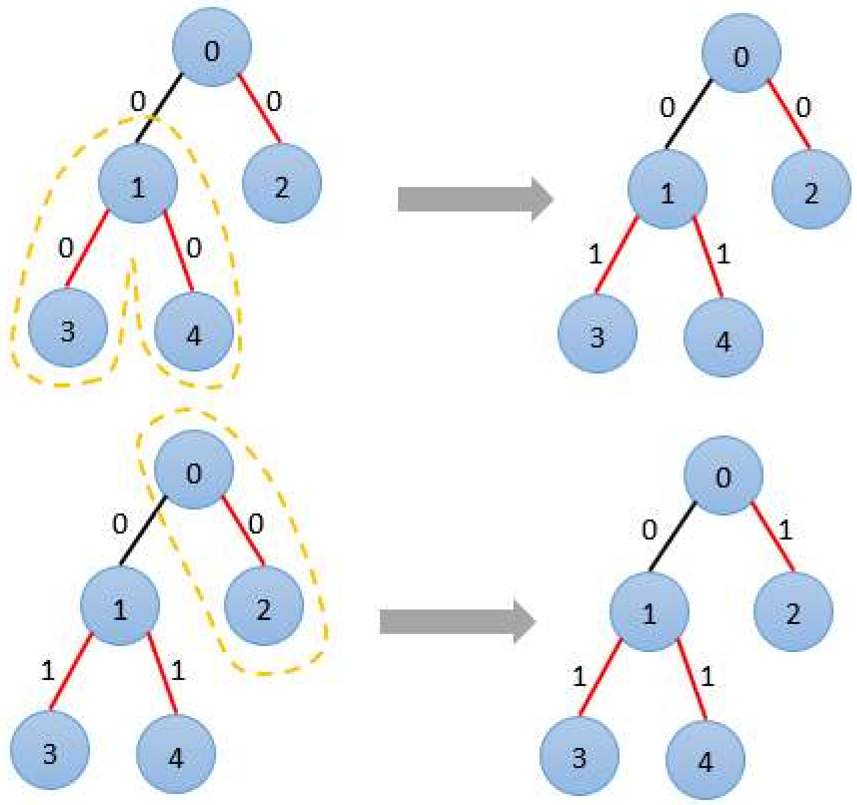


Red roads are the important ones. The optimal solution chooses the intersections 2 and 3. The path that corresponds to this choice is enclosed by the dashed line. All lamps on this path are toggled from 0 to 1.

2) {0,0,1,1}
"0000"
"0111"

Returns: 2

This is almost the same test case as Example 0, but now the initial state of the lamp 3 (i.e., the lamp on the road that connects intersections 1 and 4) is now 0. As this is an important lamp, we now need two operations in order to turn all important lamps to 1.

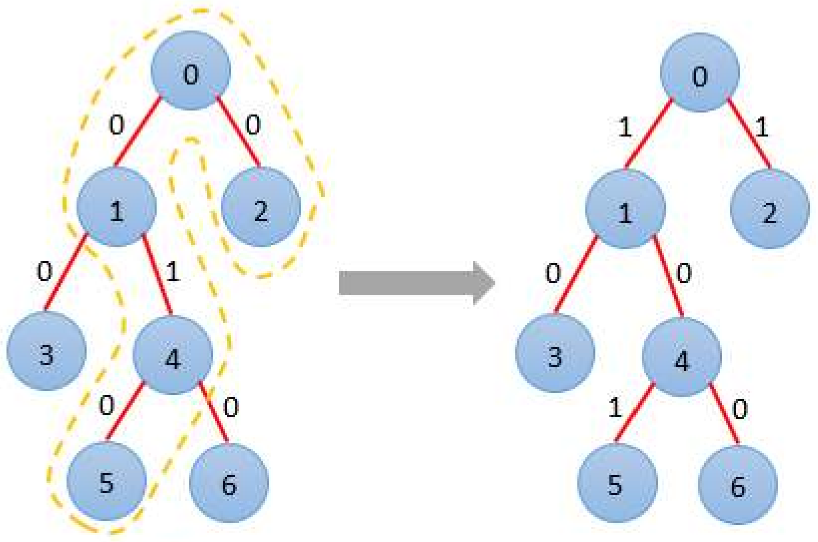


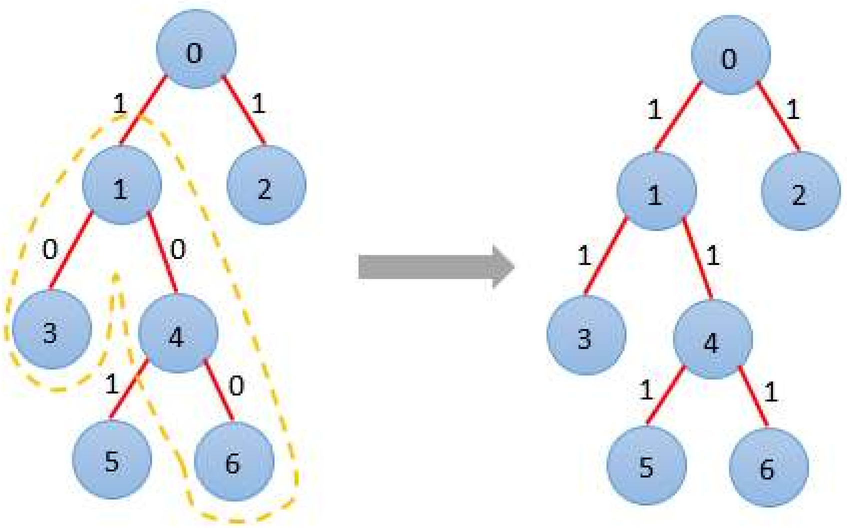
3){0,0,1,1,4,4}

"000100"

"111111"

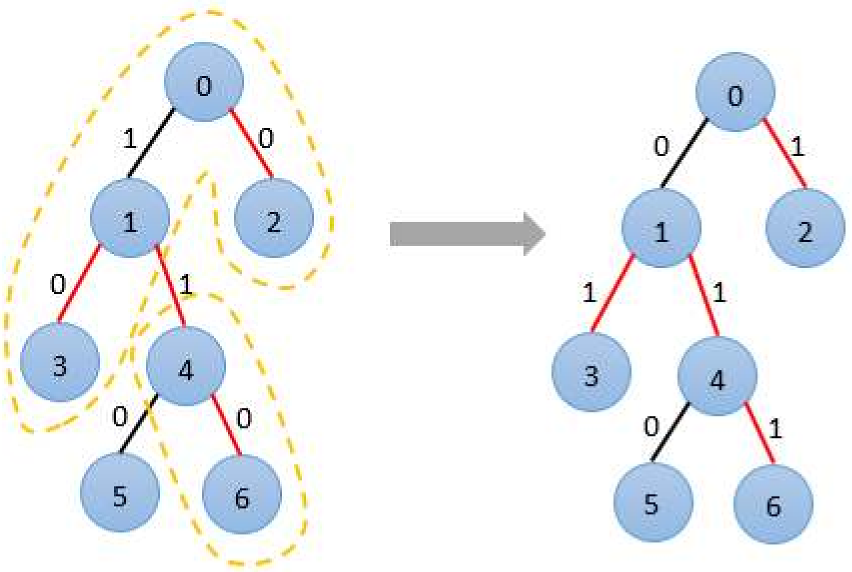
Returns: 2





Note that lamp on the road between intersections 1 and 4 was toggled twice: first from 1 to 0, and then from 0 back to 1.

4) {0,0,1,1,4,4}
   "100100"
   "011101"
  Returns: 2



5)
{0,0,2,2,3,1,6,3,1}

"010001110"

"000110100"

Returns: 1

6)
{0,0,1,2,4,4,6,1,2,5,2,8,8,3,6,4,14,7,18,14,11,7,1,12,7,5,18,23,0,14,11,10,2,2,6,1,30,11,9,12,5,35,25,
11,23,17,14,45,15}

"0000000000010000000000000010000010100000000000000"

"1010111111111011011111000110111111111111111110111"

Returns: 14

2_32) Problem Statement

There are N skyscrapers standing in a line from left to right. The buildings are numbered 1 through N, in some order (not necessarily from left to right). You do not know this order. For each i, the height of building i is i. Thus, the heights of the N buildings form a permutation of 1 through N. Each of the buildings is painted using a single color. Multiple buildings may share the same color. For each i, you know the color of building i. (The technical details are specified below.) You are standing far to the left of all the buildings. From your location, you can only see parts of some buildings, because smaller buildings farther from you are obscured by taller ones that are closer. Formally, you can see (some top part of) building X if and only if all of the buildings in front of X (that is, to the left of X in the row) are lower than building X.

Additionally, from where you are standing you cannot distinguish between two buildings that have the same color. More precisely, whenever you see two buildings with heights H1 and H2 (H1 < H2) such that they both have the same color and you can see no building that stands between them, they seem to you as one building. This also naturally extends to more than two buildings of the same color. You think you now see **L** buildings. In other words, as you look at the buildings from where you are standing, you see exactly **L**-1 places where the color changes.

You are given String[]s **color1** and **color2**, and the int **L**. The String[]s **color1** and **color2** can be used to determine the colors of buildings in the following way: Concatenate all elements of **color1** to obtain a String C1. Concatenate all elements of **color2** to obtain a String C2. The strings C1 and C2 will each have exactly N characters. Now, for each i and j: buildings i and j have the same color if and only if both C1[i-1]=C1[j-1] and C2[i-1]=C2[j-1]. In other words, for each i the color of building i is described by the two-character string C1[i-1] + C2[i-1].

Given this information, compute and return the number of possible orders of the N buildings, modulo 1,000,000,009.

Definition

Class:	ColorfulBuilding
Method:	count
Parameters:	String[], String[], int
Returns:	int
Method signature:	int count(String[] color1, String[] color2, int L)

(be sure your method is public)

Constraints

- **color1** and **color2** will each contain between 1 and 36 elements, inclusive.
- **color1** and **color2** will contain the same number of elements.
- Each element of **color1** and **color2** will contain between 1 and 36 characters, inclusive.
- For each i, the i-th elements of **color1** and **color2** will contain the same number of

characters.

- Each character of each element of **color1** and **color2** will be a lowercase or an uppercase letter ('a'-'z' or 'A'-'Z').
- **L** will be between 1 and N, where N is the total number of characters in **color1**.

Examples

1)

```
{"aaba"}
{"aaaa"}
3
```

Returns: 6

There are 4 buildings. Buildings 1, 2, and 4 share the same color, building 3 is different. There are 6 valid orders of buildings (from left to right):

- 1, 2, 3, 4
- 1, 3, 2, 4
- 1, 3, 4, 2
- 2, 1, 3, 4
- 2, 3, 1, 4
- 2, 3, 4, 1

Note that if the buildings stand in the order 1, 2, 3, 4, you can see all four buildings. However, as buildings 1 and 2 have the same color, you regard them as one building. Therefore, you think you see 3 buildings.

2)

```
{"aaba"}
{"aaba"}
4
```

Returns: 0

For each possible order of these buildings you will think you see at most 3 buildings.

3)

```
{"ab", "ba", "a", "aab"}
{"bb", "ba", "a", "aba"}
5
```

Returns: 432

Make sure to concatenate all the elements of **color1** and **color2**.

4)

```
{"xxxxxxxxxxxxxxxxxxxx",
 "xxxxxxxxxxxxxxxxxxxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOxxxxxxxxxxxxxxxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxxxxxxxxxxxxxxxOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxxxxxxxxxxxxxxxxxxx",
 "xxxxxxxxxxxxxxxxxxxx"}
{"xxxxxxxxxxxxxxxxxxxx",
 "xxxxxxxxxxxxxxxxxxxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOxxxxxxxxxxxxOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxOOOOOOOOOOOOOOOOxx",
 "xxxxxxxxxxxxxxxxxxxx",
 "xxxxxxxxxxxxxxxxxxxx"}
58
Returns: 619787617
```

5)

```
{"SJXcabKTWeUXhwxGixFepQaQlFxrAedNAtVP",
 "gorBIkcTlOFbLDbFeElYAyAqszQdMbpxSRdE",
 "SQVHPehlNtesbJDflyGxMqvgzjXisRnqDcQO",
 "pIAEBwbmpPWCooQAujbGXFGLvXxTmExLARkf",
 "AFnWyWKVObfxDGCjTxdZaObgwdxlPKtIxaAd",
 "uznMpJVNjAofbHJjOrZeSHgSagOCUMGbvkVR",
 "LBRrDbTAyKfVnedKiRfzgRzECpcsziqaTwdo",
 "JrJHvsEVVGDkNVGqLbpxyLDPloBuNDQKnReI",
 "SSYpbjKHSCnQhuyYrVauWDHDyhAoGyecrZMv",
 "UdetQfWEUWHHuAxRSdkJOOJSixKpQXpCFZHO",
 "KXVsQbuQtIgsULOMsTvPFNUqkBldMTLCipYK",
 "hoXConjnDWQkZVtyZlwSedvdVrNWqQhGUClQ",
"TpsvvyoXsXmQpBAGGVDrXCkodoRHQZmpoQDW",
"csiJspzTqeFBRmPgeEtTAzfrfCGlTZqcPuyO",
"vsPDVBJVaJmUAtDdcsKoUkPEbDmAwtZKwjjP",
"MOfoMhMiKIvGQoeIJXHzyClWRtRuKXMqxUAF",
"KyyUCkRBjsYvmPFFEGBqqVhIUdtvIyyLacfu",
"BfuwfSajSlcuTzhMufHSQLudvGEGlyHsEmBD",
"PLpIXZkdyXveTMfSvqnDGKWOZrTBMUIlZrqF",
"dzVMCqrSLbanRJTYpDJNHAOLPSzmvSEPQJYm",
"rAjRkrSjouJyFaCSPPLYSzqDmMoADyWAbobJ",
"eOCBrJNoyFnGpXpxiExXcoOHnVsaEPXhPfLe",
"XMjRksnxWssPEINhdhbiLBSYpCLtwNshFjXS",
"HnnDeUAbuswsgsYQuAaXySLkFYUwFXwYTreM",
"uqLnwOEGbwZZDgAAWEdLRZxFlogDmlhjhgHM",
"NcfaQsgPQfirkYDRFrLQpySmBGfRHumKULZf",
"mOpmmgclsxRzXskEywfryqCRyATNoJwnlHiD",
"AnoKoKAjrasttjNlHCROnvTJMhEHlVPVoVMo",
"yegLyIuRkkENFAjwzDoPLKjgUHHEkfzYDIpo",
"EcxRGGfuaBXUFXkSxctJWOLmmVbvoMkWtTvV",
"nBMkOBHgaltEVzfyGxseGhmBscfGIbxFbqRn",
"GvkEcLtqdOofGtsbWDafVMbNdJxsffKDzSiR",
"jhZQUVzTzalrZcebvyqPWtOUUyniBKTWdyLi",
"ODJLQPMdjDdTlxrfGsNmBfeGYjzeXApqKDhJ",
"vlJkWMzdVQujKdLViQClOrJXMTBkuZEasFgg",
"FAsbuzrLVIgaryqXBfuBRAbReleXSSgEKSvt"}
{"XAhlUHtfjOpOlQlowWppQcodMGUiqoLobxnD",
"IDzGlKrHiVGdBjZxIcoxjwagbWieKvUwZrjg",
"SkpkTVeIFeoNQzUUgmYAYskKONNZdgXxSiWd",
"rXBGhTmqnvxkmqiutoDzPusDiOUggEFXGCzm",
"fzrxVuTbFXunGbeEavjshmrIRlDorTuISCxn",
"LsvIaxTkOBjcskiekmrKNNFEXqnfMNXLWqqu",
"ekzxGIlbJPVEQPYAbTuMtZKsCiubDXDiBVzU",
"hZuOCJxvBOYENGSFcUiKKAKfCdebutVmnyvB",
"UveNkKRQmHnedrROQOpHJfrHjwcNLUShlDbn",
"ptUkvbaGDryHRkYfHKLkSgVpjWCEcgazyxGK",
"nVPYEqTTJsRPjzjfdOiULhnZPFeNcnbXaQlk",
"IPXBXRhMQIkxpygsgbcRfMuvIcuzUPPHGOWX",
"yWWlNyEyqZSIOXBFAybIuFpVqpvmKRaRFrAE",
"EvBJVtHvKhjrFcmtpdBbFTdTVtXXZQKAglKT",
"bCVjHzUvyINFkxXageZQMzCyNhcifACdJVDh",
"lZITYcDSeIbLweyFtoMAfOQyBNupKlhcNpgo",
"BduslNrJdWOUukYFFidEkMFaghfofpxVgvJd",
"YrJpDZKqdjEPzdLsOQEdkcrBfNHPemXHokCW",
"GjeUKSGjDlgKTyUGNrMQbBLxRUcgrWpkAwOD",
"wgxTcswqdJHaDugNIRMvrhBsdDaJAssVbSRW",
"qmVmqFEpvgGioMXDSFqEoQcDOAaUoGPEovSO",
"KrukPlpfOhawaovCfteTSCIdLMrtImVtiMyQ",
"ykwmxHsKMFzFHwcbyyedLvhZPnaNGqJMMCxd",
"HUNYCXjNLQIFCLLGpCXHBCHLTxLynBxnHFbx",
"uwjzbNbJepVFgMPUXVirxYHzExquBEtPmKju",
"xXAxAbJePyUsVHeLytDvAxBGMRtnvCEiZZqe",
"xMkQoIVxWPXPgaOYmDjTOXiMImVdzojERNxS",
"dwICFwMAmdOIUxyAdXdshasnzwyhfnVWVqZJ",
"etypXNVvSTEQvriGBZdSGmDEHhvpSqkFklCS",
"YkxpFBCRYUueRcKaJUXVdaMoYMYEooPQVMOr",
"DTrexDqclZNKdPuTRFHualJSFziCLPCZjpxo",
"TfEijcAsSJPikkmBSbXMqYHAhPTcpcKVSkIX",
"xKXHYPYMJxFpSbxltDKYuRiTkOLxpQKnXZPs",
"YFYuvuYHfpFJcrLNIdlNfBxRnWdppsdalBkx",
"NFTysBvNFjejdnlhRTclbcfGipNCxpFEOriY",
 "thkgVflJYmbUYbIlafNUMGePQWiZyYzYXvUR"}
 1200
Returns: 396065426
```

2_33) Problem Statement

Manao is studying graph theory and simple cycles in particular. A simple cycle of length $L \geq 3$ in graph G is a sequence of vertices $(v_0, v_1, ..., v_{L-1})$ such that all $v_0, v_1, ..., v_{L-1}$ are pairwise distinct and for each i=0..L-1, an edge between v_i and $v_{(i+1) \bmod L}$ exists in G. Manao is interested in graphs formed by connecting two trees. The connection process is as follows. Manao takes two trees composed of N vertices each. The vertices in each tree are labeled from 0 to N - 1. Then, he generates a permutation P of numbers from 0 to N - 1 uniformly at random. Finally, the graph is formed by connecting vertex i of the first tree to vertex P[i] of the second tree, for each i from 0 to N - 1. To remove ambiguity, the vertices of the first tree within the graph are referred to as $A_0, A_1, ..., A_{N-1}$ and the vertices of the second graph are referred to as $B_0, B_1, ..., B_{N-1}$. Manao wants to know the expected number of simple cycles of length **K** in the resulting graph.

You are given two String[]s, **tree1** and **tree2**. Merge the elements of **tree1** to obtain a single string formatted as a space-separated list of N - 1 integers describing the first tree. Let x[i] denote the i-th integer (0-based index) in the list. Then, for each i, we have $0 \leq x[i] \leq i$ and in our tree the vertices x[i] and i+1 are connected by an edge. **tree2** describes the second tree in the same fashion.

Compute and return the expected number of simple cycles of length **K** in the graph formed by connecting the two given trees as described above. Two simple cycles are equal if one of them can be cyclically shifted, or reversed and cyclically shifted, to coincide with the second. According to this definition, (1, 2, 3, 4), (2, 3, 4, 1) and (3, 2, 1, 4) are all equal.

Definition

Class:	TreeUnion
Method:	expectedCycles
Parameters:	String[], String[], int
Returns:	double
Method signature:	double expectedCycles(String[] tree1, String[] tree2, int K)

(be sure your method is public)

Notes

- The returned value must have an absolute or relative error less than 1e-9.

Constraints

- The concatenation of elements of **tree1** will be formatted as a space-separated list of N - 1 integers, where N is between 2 and 300, inclusive.
- **tree1** will contain between 1 and 50 elements, inclusive.
- Each element of **tree1** will be between 1 and 50 characters long, inclusive.
- For each i, the i-th integer (0-based index) in the concatenation of elements of **tree1** will

be between 0 and i, inclusive, and will have no extra leading zeros.

- The concatenation of elements of **tree2** will be formatted as a space-separated list of N - 1 integers, where N is between 2 and 300, inclusive.
- **tree2** will contain between 1 and 50 elements, inclusive.
- Each element of **tree2** will be between 1 and 50 characters long, inclusive.
- For each i, the i-th integer (0-based index) in the concatenation of elements of **tree2** will be between 0 and i, inclusive, and will have no extra leading zeros.
- **K** will be between 3 and 7, inclusive.

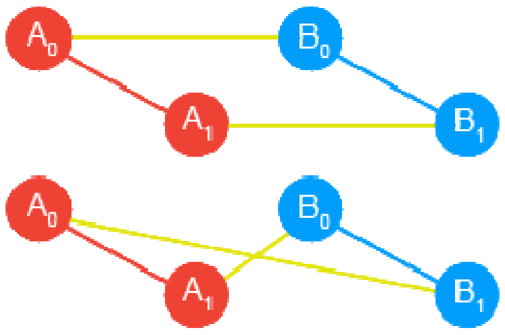
Examples

1)

```
{"0"}
{"0"}
4
```

Returns: 1.0

Manao has two trees with two vertices each. He can connect them in two ways:



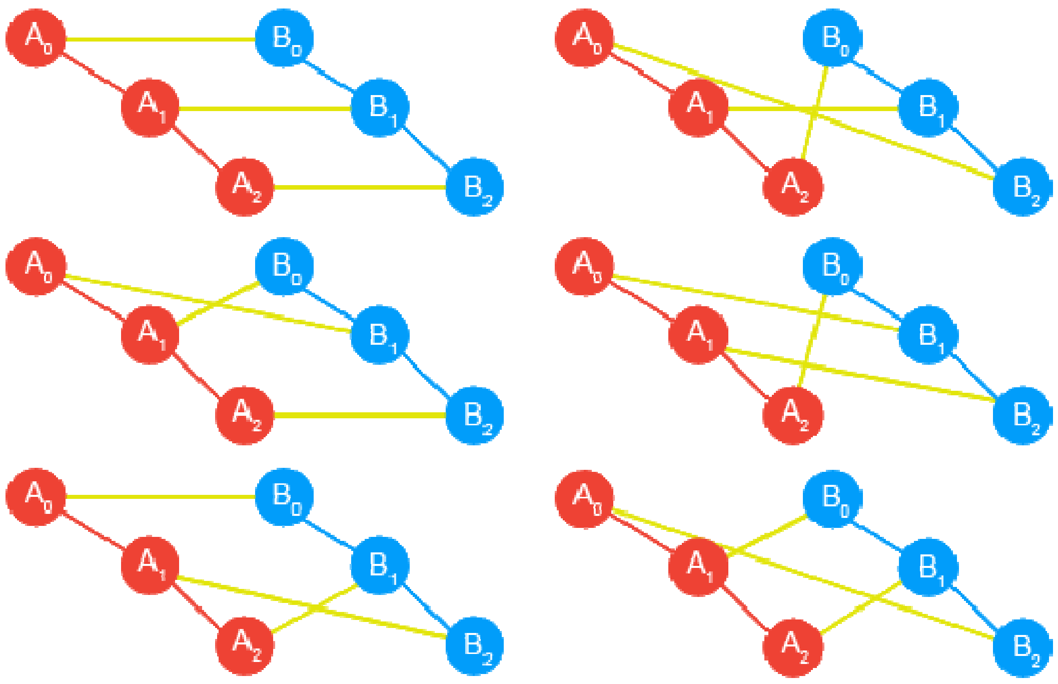
Either way, the resulting graph is a single cycle of length 4.

2)

```
{"0 1"}
{"0 1"}
4
```

Returns: 1.3333333333333333

Manao has two chains composed of three vertices each. There are 6 possible permutations which result in the following graphs:



Each of the two graphs shown in the topmost row contains two cycles of length 4. (Note that in each case the two cycles share the edge A_1B_1.) Each of the other four graphs only contains one cycle of length 4. Thus the expected number of cycles of length 4 is (2+2+1+1+1+1)/6 = 8/6 = 1.3333333333.

3)

```
{"0 1"}
{"0 1"}
6
```

Returns: 0.3333333333333333

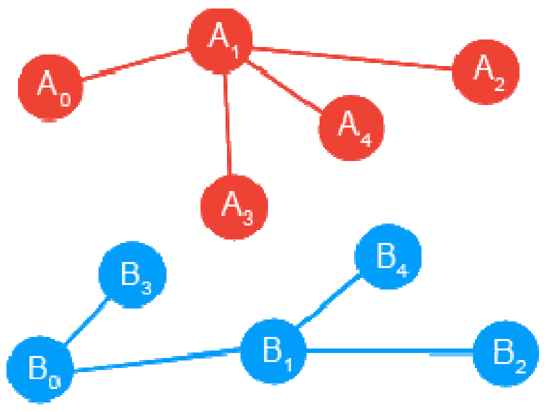
These are the same trees as in the previous example, but this time Manao is looking for simple cycles with 6 vertices. Only the topmost two graphs contain a cycle of length 6, thus the expected number of such cycles for a random permutation P is 1/3.

4)

```
{"0 ", "1 1 1"}
{"0 1 0 ", "1"}
5
```

Returns: 4.0

The corresponding trees are these:



5)

```
{"0 1 2 0 1 2 0 1 2 5 6 1", "0 11", " 4"}
{"0 1 1 0 2 3 4 3 4 6 6", " 10 12 12"}
7
```

Returns: 13.314285714285713

2_34) Problem Statement

A group of freshman rabbits has recently joined the Eel club. No two of the rabbits knew each other. Yesterday, each of the rabbits went to the club for the first time. For each i, rabbit number i entered the club at the time s[i] and left the club at the time t[i]. Each pair of rabbits that was in the club at the same time got to know each other, and they became friends on the social network service Shoutter. This is also the case for rabbits who just met for a single moment (i.e., one of them entered the club exactly at the time when the other one was leaving). In Shoutter, each user can post a short message at any time. The message can be read by the user's friends. The friends can also repost the message, making it visible to their friends that are not friends with the original poster. In turn, those friends can then repost the message again, and so on. Each message can be reposted in this way arbitrarily many times. If a rabbit wants to repost multiple messages, he must repost each of them separately.

Today, each of the rabbits posted a self-introduction to Shoutter. Each rabbit would now like to read the self-introductions of all other rabbits (including those that are currently not his friends). Compute and return the minimal number of reposts necessary to reach this goal. If it is impossible to reach the goal, return -1 instead. As the number of rabbits can be greater than what the TopCoder arena supports, you are given the times s[i] and t[i] encoded in the following form: You are given String[]s **s1000**, **s100**, **s10**, and **s1**. Concatenate all elements of **s1000** to obtain a string S1000. In the same way obtain the strings S100, S10, and S1. Character i of each of these strings corresponds to the rabbit number i. More precisely, these characters are the digits of s[i]: we obtain s[i] by converting the string S1000[i]+S100[i]+S10[i]+S1[i] to an integer. For example, if S1000[4]='0', S100[4]='1', S10[4]='4', and S1[4]='7', then s[4]=to_integer("0147")=147. You are also given String[]s **t1000**, **t100**, **t10**, and **t1**. These encode the times t[i] in the same way.

Definition

Class:	ShoutterDiv1
Method:	count
Parameters:	String[], String[], String[], String[], String[], String[], String[], String[]
Returns:	int
Method signature:	int count(String[] s1000, String[] s100, String[] s10, String[] s1, String[] t1000, String[] t100, String[] t10, String[] t1)

(be sure your method is public)

Constraints

- **s1000, s100, s10, s1, t1000, t100, t10** and **t1** will each contain between 1 and 50 elements, inclusive.
- **s1000, s100, s10, s1, t1000, t100, t10** and **t1** will contain the same number of elements.

- Each element of **s1000, s100, s10, s1, t1000, t100, t10** and **t1** will contain between 1 and 50 characters, inclusive.
- For each i, the i-th elements of all input variables will all contain the same number of characters.
- Each character in the input variables will be a digit ('0'-'9').
- For each i, t[i] will be greater than or equal to s[i].

Examples

1)

```
{"22", "2"}
{"00", "0"}
{"11", "1"}
{"21", "4"}
{"22", "2"}
{"00", "0"}
{"11", "1"}
{"43", "6"}
```

Returns: 2

After parsing the input, you will get the following information: Rabbit 0 will enter the room at 2012 and leave the room at 2014. Rabbit 1 will enter the room at 2011 and leave the room at 2013. Rabbit 2 will enter the room at 2014 and leave the room at 2016. Therefore, Rabbit 0 and Rabbit 1 will be friends, and Rabbit 0 and Rabbit 2 will be friends too, but Rabbit 1 and Rabbit 2 won't be friends.

Rabbit 0 can already see the self-introductions of all rabbits, but rabbits 1 and 2 cannot see each other's self-introduction. Two actions are needed: First, Rabbit 0 reposts the self-introduction of Rabbit 1, and then Rabbit 0 reposts the self-introduction of Rabbit 2. Now everybody can read everything.

2)

```
{"00"}
{"00"}
{"00"}
{"13"}
{"00"}
{"00"}
{"00"}
{"24"}
```

Returns: -1

If it is impossible to achieve the goal, return -1.

3)

```
{"0000"}
{"0000"}
{"0000"}
{"1234"}
```

```
{"0000"}
{"0000"}
{"0000"}
{"2345"}
```

Returns: 6

The following pairs will be friends: Rabbit 0 and 1, 1 and 2, and 2 and 3. One of the optimal strategies is as follows:

- Rabbit 1 shares introductions of Rabbit 0 and 2.
- Rabbit 2 shares introductions of Rabbit 1 and 3.
- Rabbit 1 shares introduction of Rabbit 3 (this is possible because now Rabbit 3's introduction is shared by Rabbit 2, who is a Rabbit 1's friend).
- Rabbit 2 shares introduction of Rabbit 0 (this is possible because now Rabbit 0's introduction is shared by Rabbit 1, who is a Rabbit 2's friend).

4)

```
{"0000000000"}
{"0000000000"}
{"0000000000"}
{"7626463146"}
{"0000000000"}
{"0000000000"}
{"0000000000"}
{"9927686479"}
```

Returns: 18

5)

```
{"00000000000000000000000000000000000000000000000000"}
{"00000000000000000000000000000000000000000000000000"}
{"50353624751857130208544645495168271486083954769538"}
{"85748487990028258641117783760944852941545064635928"}
{"00000000000000000000000000000000000000000000000000"}
{"00000000000000000000000000000000000000000000000000"}
{"61465744851859252308555855596388482696094965779649"}
{"37620749792666153778227385275518278477865684777411"}
```

Returns: 333

2_35) Problem Statement

Farmer Bessie and Cow John are walking along a straight road. They are cautious, because they have heard that there may be some wolves on the road. The road consists of **N** sections. The sections are numbered 0 through **N**-1, in order. Each section of the road contains at most one wolf. You have M additional pieces of information about the positions of the wolves. Each piece of information is an interval of the road that contains at most two wolves. More precisely, for each i between 0 and M-1, inclusive, you are given two integers left[i] and right[i] such that the sections with numbers in the range from left[i] to right[i], inclusive, contain at most two wolves in total. You are given two String[]s **L** and **R**. The concatenation of all elements of **L** will be a single space separated list containing the integers left[0] through left[M-1]. **R** contains all the integers right[i] in the same format. Return the number of ways in which wolves can be distributed in the sections of the road, modulo 1,000,000,007.

Definition

Class:	WolfInZooDivOne
Method:	count
Parameters:	int, String[], String[]
Returns:	int
Method signature:	int count(int N, String[] L, String[] R)

(be sure your method is public)

Constraints

- **N** will be between 1 and 300, inclusive.
- **L** and **R** will contain between 1 and 50 elements, inclusive.
- Each element of **L** and **R** will contain between 1 and 50 characters, inclusive.
- Each character in **L** and **R** will be a digit ('0'-'9') or a space (' ').
- M will be between 1 and 300, inclusive.
- The concatenation of all elements of **L** will be a single space separated list of M integers. The integers will be between 0 and N-1, inclusive, and they will be given without unnecessary leading zeroes.
- The concatenation of all elements of **R** will be a single space separated list of M integers. The integers will be between 0 and N-1, inclusive, and they will be given without unnecessary leading zeroes.
- For each i, the i-th integer in **L** will be smaller than or equal to the i-th integer in **R**.

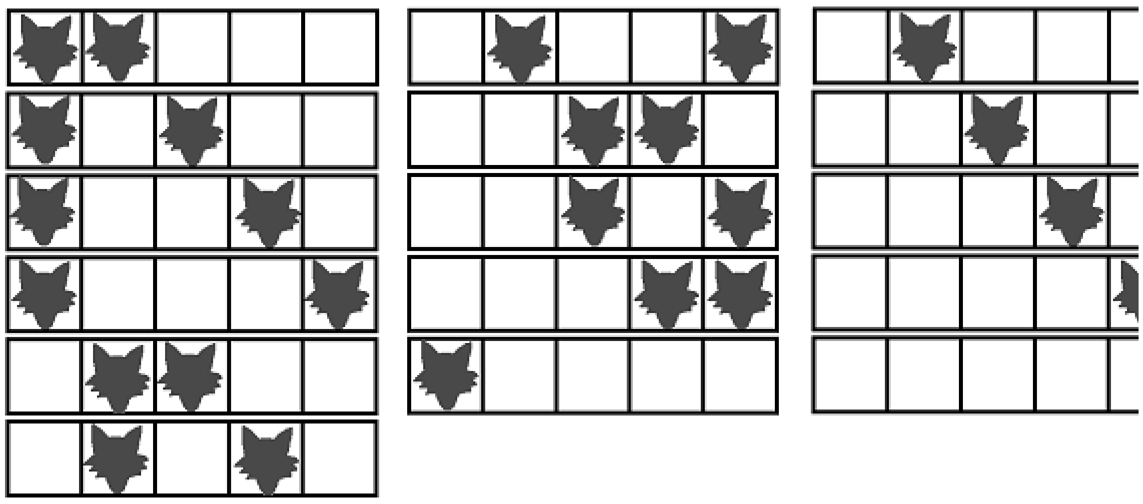
Examples

1)

```
5
{"0"}
{"4"}
```

Returns: 16

There are at most two wolves in the segments 0 through 4, i.e., in the entire road. The following picture shows all possible ways how the wolves can be distributed. Note that there can also be only one wolf, or no wolves at all.



2)

5

{"0 1"}

{"2 4"}

Returns: 21

3)

10

{"0 4 2 7"}

{"3 9 5 9"}

Returns: 225

4)

100

{"0 2 2 7 10 1","3 16 22 30 33 38"," 42 44 49 51 57 60 62"," 65 69 72 74 77 7","8 81 84 88 91 93 96"}

{"41 5 13 22 12 13 ","33 41 80 47 40 ","4","8 96 57 66 ","80 60 71 79"," 70 77 ","99"," 83 85 93 88 89 97 97 98"}

Returns: 6419882

You must first concatenate the elements of **L** and only then split it into integers. The same holds for **R**.

2_36) Problem Statement

Elly has a standard chessboard, divided into 8 by 8 unit square cells. She wants to place pebbles onto some of the cells. You are given a String[] **board**. The j-th character of the i-th element of **board** is '#' if she wants to put a pebble onto the cell (i, j), and it is '.' otherwise. Initially the chessboard doesn't contain any pebbles. Elly places the pebbles one by one. The cost of adding a pebble is defined as follows. If this is the first pebble to be placed (i.e., the board is empty), it can be placed for free. Otherwise, the cost is the Manhattan distance (see Notes for the definition) to the most distant pebble that has already been placed on the board. Return the minimal total cost of placing a pebble onto each chosen cell.

Definition

Class:	EllysChessboard
Method:	minCost
Parameters:	String[]
Returns:	int

Method signature: int minCost(String[] board)
(be sure your method is public)

Notes

- The Manhattan distance between the cell (x1, y1) and the cell (x2, y2) is defined as |x1-x2| + |y1-y2|, where || denotes absolute value.

Constraints

- **board** will contain exactly 8 elements.
- Each element of **board** will contain exactly 8 characters.
- Each character in **board** will be either '#' or '.'.

Examples

1)

```
{"........",
 "........",
 "...#....",
 ".#......",
 ".......#",
 "........",
 "........",
 "........"}
```

Returns: 10

Elly wants to put pebbles on three cells: (2, 3), (3, 1), and (4, 7). One of the optimal ways to do this is as follows:

- First, put a pebble to (2, 3). It costs nothing.
- Next, put a pebble to (3, 1). It costs |2-3| + |3-1| = 3.
- Next, put a pebble to (4, 7). The Manhattan distance between (4, 7) and (2, 3) is 6, and the Manhattan distance between (4, 7) and (3, 1) is 7, so the cost is max(6, 7) = 7.

The total cost is 0 + 3 + 7 = 10.

2)

```
{"........",
 "........",
 "........",
 "........",
 "........",
 "........",
 "........",
 "........"}
```

Returns: 0

3)

```
{".#......",
 "........",
 "..#..#.#",
 "...#..#.",
 "........",
 "...#...#",
 "...#...#",
 "........"}
```

Returns: 58

4)

```
{"##..####",
 "#####..#",
 "..#.#...",
 "#..##.##",
 ".#.###.#",
 "####.###",
 "#.#...#.",
 "##....#."}
```

Returns: 275

5)

```
{"########",
 "########",
 "########",
 "########",
 "########",
 "########",
 "########",
 "########"}
```

Returns: 476

2_37) Problem Statement

Manao had a sheet of paper. He drew **N** points on it, which corresponded to vertices of a regular **N**-gon. He numbered the vertices from 1 to **N** in clockwise order. After that, Manao connected several pairs of points with straight line segments. Namely, he connected points **points**[i] and **points**[i+1] for each i between 0 and M-2, where M is the number of elements in **points**. Note that all numbers in **points** are distinct. Manao took a look at what he had drawn and decided to continue his traversal by adding every remaining point of the polygon to it and then returning to point **points**[0]. In other words, Manao is going to connect point **points**[M-1] with some point *tail*[0] which is not in **points**, then connect *tail*[0] with some point *tail*[1] which is neither in **points** nor in *tail*, and so on. In the end, he will connect point *tail*[**N**-M-1] with point **points**[0], thus completing the traversal. Manao is really fond of intersections, so he wants to continue the traversal in such a way that every new line segment he draws intersects with at least one of the previously drawn line segments. (Note that the set of previously drawn segments includes not only the original set of segments, but also the new segments drawn before the current one.) Count and return the number of ways in which he can complete the traversal.

Definition

Class:	PolygonTraversal
Method:	count
Parameters:	int, int[]
Returns:	long
Method signature:	long count(int N, int[] points)

(be sure your method is public)

Constraints

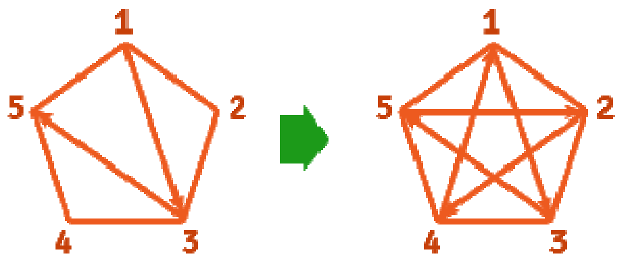
- **N** will be between 4 and 18, inclusive.
- **points** will contain between 2 and **N**-1 elements, inclusive.
- Each element of **points** will be between 1 and **N**, inclusive.
- The elements of **points** will be distinct.

Examples

1)

```
5
{1, 3, 5}
Returns: 1
```

The only way for Manao to complete the traversal is:



2)

6
{1, 4, 2}

Returns: 1

The only way to complete the traversal is to visit vertices {6, 3, 5, 1}, in order. Note that the segment 5-1 does not intersect the original two segments (1-4 and 4-2), but it does intersect segments 2-6 and 6-3 which were both added before 5-1.

3)

7
{2, 4, 7}

Returns: 2

The two possible tails are:

- 3-5-1-6-2
- 3-6-1-5-2

4)

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

Returns: 0

Manao needs to connect points 5 and 7 and then connect points 7 and 1. Obviously, segment 1-7 will not intersect with any other segment.

5)

18
{1, 7, 18}

Returns: 4374612736

2_38) Problem Statement

Manao likes to play the Division Game with his friends. This two-player game is played with some collection of natural numbers S. Manao plays first and the players alternate in making a move. A move is choosing some number X from S and a natural number Y > 1 such that Y divides X. Then, X is replaced by X / Y in the collection. Note that at any moment the collection may contain multiple copies of the same number. The game proceeds until no more moves can be made. The player who managed to make the last move is declared the winner. Since hot debates arise on what numbers should be in S, the friends decided to regularize their choice. They always choose a contiguous interval of numbers [A, B] to be the initial collection S. That is, at the beginning of the game, the collection S contains each of the integers A through B, inclusive, exactly once. Manao knows that A and B will satisfy the condition **L** &le A &le B &le **R**. You are given the ints **L** and **R**. Count the number of such intervals for which Manao will win the Division Game given that both players play optimally.

Definition

Class:	TheDivisionGame
Method:	countWinningIntervals
Parameters:	int, int
Returns:	long
Method signature:	long countWinningIntervals(int L, int R)

(be sure your method is public)

Notes

- Only one number from the collection changes in each move. For example, if the collection contains three copies of the number 8, and the player chooses X=8 and Y=4, only one of the 8s in the collection will change to a 2.

Constraints

- **L** will be between 2 and 1,000,000,000, inclusive.
- **R** will be between **L** and **L** + 1,000,000, inclusive.

Examples

1)

```
9
10
```

Returns: 2

If the chosen interval is [9,9] or [10,10], the collection S contains only one number. In these two situations Manao can win the game in a single move. On the other hand, if the chosen interval is [9,10], Manao will lose to an optimally playing opponent.

2)

```
2
5
```

Returns: 9

The only case where Manao loses is if the game starts with the interval [2,3]. Note that if the starting interval is [2,5], Manao can choose X=4 and Y=2 in his first move. After that move, the collection will contain the values 2, 2, 3, and 5.

3)

```
2
6
```

Returns: 13

Manao will also lose the game if the starting interval is [3,6].

4)

```
2
100
```

Returns: 4345

5)

```
12566125
12567777
```

Returns: 1313432

2_39) Problem Statement

You are playing a card game. In the card game, each card holds a magic spell with two properties: its level and its damage. During the game, you will play some of the cards (possibly none or all of them) to attack your enemy. Initially, there are n cards. The cards are placed in a row and they are labeled from 0 to n-1, in order. You are given two int[]s: **level** and **damage**. For each i, the level of card i is **level**[i], and its damage is **damage**[i].

In each turn of the game, you can do one of two possible actions:

1. Let L be the level and D the damage of the card that is currently the leftmost card in the row. If there are at least L cards in the row, you may play the leftmost card. Playing it deals D damage to your enemy. After you play this card, the first L cards in the row (including the played one) are discarded. That is, the cards currently labeled 0 through (L-1), inclusive, are discarded. The order of the remaining cards does not change.
2. If you have at least one card, you can take the last card in the row and move it to the beginning. For example, if the row initially contained cards A,B,C,D,E, in this order, after this operation it will contain E,A,B,C,D.

After each turn, the cards are relabeled 0 through x-1, where x is their current count.

Return the maximal total damage you can deal to your opponent.

Definition

Class:	SpellCards
Method:	maxDamage
Parameters:	int[], int[]
Returns:	int
Method signature:	int maxDamage(int[] level, int[] damage)

(be sure your method is public)

Constraints

- **level** will contain between 1 and 50 elements, inclusive.
- **level** and **damage** will contain the same number of elements.
- Each element in **level** will be between 1 and 50, inclusive.
- Each element in **damage** will be between 1 and 10,000, inclusive.

Examples

1)

```
{1,1,1}
```

{10,20,30}

Returns: 60

You can play card 0 three times in a row, dealing 10+20+30 = 60 damage.

2)

{3,3,3}

{10,20,30}

Returns: 30

Here, it is optimal to start by moving the last card to the beginning of the row. In the second turn we then use the card and deal 30 damage. Afterwards, all three cards are discarded.

3)

{4,4,4}

{10,20,30}

Returns: 0

This time you can't use any spell card.

4)

{50,1,50,1,50}

{10,20,30,40,50}

Returns: 60

You can use 2 cards with damage 20 and 40.

5)

{2,1,1}

{40,40,10}

Returns: 80

6)

{1,2,1,1,3,2,1}

{10,40,10,10,90,40,10}

Returns: 170

7)

{1,2,2,3,1,4,2}

{113,253,523,941,250,534,454}

Returns: 1918

2_40) Problem Statement

You are playing a solitaire game called Left-Right Digits Game. This game uses a deck of N cards. Each card has a single digit written on it. These digits are given as characters in the String **digits**. More precisely, the i-th character of **digits** is the digit written on the i-th card from the top of the deck (both indices are 0-based). The game is played as follows. First, you place the topmost card (whose digit is the 0-th character of **digits**) on the table. Then, you pick the cards one-by-one from the top of the deck. For each card, you have to place it either to the left or to the right of all cards that are already on the table. After all of the cards have been placed on the table, they now form an N-digit number. You are given a String **lowerBound** that represents an N-digit number. The primary goal of the game is to arrange the cards in such a way that the number X shown on the cards will be greater than or equal to **lowerBound**. If there are multiple ways to satisfy the primary goal, you want to make the number X as small as possible. Return the smallest possible value of X you can achieve, as a String containing N digits. If it is impossible to achieve a number which is greater than or equal to **lowerBound**, return an empty String instead.

Definition

Class:	LeftRightDigitsGame2
Method:	minNumber
Parameters:	String, String
Returns:	String
Method signature:	String minNumber(String digits, String lowerBound)

(be sure your method is public)

Notes

- **lowerBound** has no leading zeros. This means that any valid number X should also have no leading zeros (since otherwise it will be smaller than **lowerBound**).

Constraints

- **digits** will contain between 1 and 50 characters, inclusive.
- Each character of **digits** will be between '0' and '9', inclusive.
- **lowerBound** will contain the same number of characters as **digits**.
- Each character of **lowerBound** will be between '0' and '9', inclusive.
- The first character of **lowerBound** will not be '0'.

Examples

1)

```
"565"
"556"
```

Returns: "556"

You can achieve exactly 556. The solution is as follows:

- Place the first card on the table.
- Place the second card to the right of all cards on the table.
- Place the last card to the left of all cards on the table.

2)

"565"
"566"

Returns: "655"

3)

"565"
"656"

Returns: ""

The largest number you can achieve is 655, but it is still less than 656.

4)

"9876543210"
"5565565565"

Returns: "5678943210"

5)

"8016352"
"1000000"

Returns: "1086352"

2_41) Problem Statement

A platypus has been given the mission to paint the cells on a grid either black or white according to the following two conditions:

- For each color, all cells of that color must be connected. Formally, a pair of cells of color X is connected if there is a path of adjacent cells of color X between them. (Two cells are adjacent if they share a common edge.) We require that for each color, each pair of cells of that color must be connected.
- All the cells of each color must form a convex shape. A group of cells of a given color is convex if in each row and each column the cells of that color form a connected segment (possibly taking the whole row or column). In other words, whenever two cells of the same color share the same row or the same column, all cells between them must also have that particular color.

The platypus is also allowed to paint the grid completely white or black.The platypus may have already painted some of the cells. The current state of the grid is given as a String[] **grid**. The i-th character of the j-th element of **grid** that represents the cell at row j, column i is 'W' if it has been painted white, 'B' if it has been painted black, and '?' if it does not have a color yet. Let X be the number of different ways how to color the rest of the grid according to the above conditions. Return the value X modulo 1000000007 (10^9 + 7). Two ways to color a grid are different if the color of at least one cell differs.

Definition

Class:	TwoConvexShapes
Method:	countWays
Parameters:	String[]
Returns:	int
Method signature:	int countWays(String[] grid)

(be sure your method is public)

Constraints

- **grid** will contain between 1 and 50 elements, inclusive.
- Element 0 of **grid** will contain between 1 and 50 characters, inclusive.
- The remaining elements of **grid** will contain the same number of characters as element 0.
- Each character in each element of **grid** will be one of 'B', 'W', and '?' (quotes for clarity).

Examples

1)

```
{"??",
 "??"}
```

Returns: 14

Of all the 16 different ways to color the grid, only the following 2 are *not* valid.

BW WB
WB BW

This is because cells of the same color are not connected.

2)

```
{"B?",
 "??"}
```

Returns: 7

The following seven ways to color the grid are correct:

```
BB BW BB BW BB BB BW
BB BW WW WW WB BW BB
```

3)

```
{"WWB",
 "WWW",
 "WWW",
 "WWW"}
```

Returns: 1

All colors have already been picked. The only possible coloring is already valid.

4)

```
{"BBBBBB",
 "WWBBBB",
 "WBBBBB"}
```

Returns: 0

This coloring of the grid is not valid, the black cells do not form a convex shape.

5)

```
{"?BB?",
 "BBBB",
 "?BB?"}
```

Returns: 5

6)

```
{"?WWWWWWWWWWWWWWWWWWWWWWWWWWWWWW",
 "B?WWWWWWWWWWWWWWWWWWWWWWWWWWWWW",
 "BB?WWWWWWWWWWWWWWWWWWWWWWWWWWWW",
 "BBB?WWWWWWWWWWWWWWWWWWWWWWWWWWW",
 "BBBB?WWWWWWWWWWWWWWWWWWWWWWWWWW",
 "BBBBB?WWWWWWWWWWWWWWWWWWWWWWWWW",
 "BBBBBB?WWWWWWWWWWWWWWWWWWWWWWWW",
 "BBBBBBB?WWWWWWWWWWWWWWWWWWWWWWW",
 "BBBBBBBB?WWWWWWWWWWWWWWWWWWWWWW",
 "BBBBBBBBB?WWWWWWWWWWWWWWWWWWWWW",
 "BBBBBBBBBB?WWWWWWWWWWWWWWWWWWWW",
 "BBBBBBBBBBB?WWWWWWWWWWWWWWWWWWW",
 "BBBBBBBBBBBB?WWWWWWWWWWWWWWWWWW",
 "BBBBBBBBBBBBB?WWWWWWWWWWWWWWWWW",
 "BBBBBBBBBBBBBB?WWWWWWWWWWWWWWWW",
 "BBBBBBBBBBBBBBB?WWWWWWWWWWWWWWW",
 "BBBBBBBBBBBBBBBB?WWWWWWWWWWWWWW",
 "BBBBBBBBBBBBBBBBB?WWWWWWWWWWWWW",
 "BBBBBBBBBBBBBBBBBB?WWWWWWWWWWWW",
 "BBBBBBBBBBBBBBBBBBB?WWWWWWWWWWW",
 "BBBBBBBBBBBBBBBBBBBB?WWWWWWWWWW",
 "BBBBBBBBBBBBBBBBBBBBB?WWWWWWWWW",
 "BBBBBBBBBBBBBBBBBBBBBB?WWWWWWWW",
 "BBBBBBBBBBBBBBBBBBBBBBB?WWWWWWW",
 "BBBBBBBBBBBBBBBBBBBBBBBB?WWWWWW",
 "BBBBBBBBBBBBBBBBBBBBBBBBB?WWWWW",
 "BBBBBBBBBBBBBBBBBBBBBBBBBB?WWWW",
 "BBBBBBBBBBBBBBBBBBBBBBBBBBB?WWW",
 "BBBBBBBBBBBBBBBBBBBBBBBBBBBB?WW",
 "BBBBBBBBBBBBBBBBBBBBBBBBBBBBB?W"}
```

Returns: 73741817

Each of the 2^30 ways to color the remaining cells in the grid is valid.

2_42) Problem Statement

A magician has challenged you to a game of wits. First he shows you some coins. Different coins may have different values. Next he takes some hats and hides all the coins inside the hats, in such a way that no two coins are hidden in the same hat. Finally, he places each of the hats with their respective coin onto some cell of a checkerboard. Now he has given you some guesses. In each guess you may ask the magician to reveal the contents of one of the hats. After you make your guess, but before he reveals the contents of a hat, the magician may magically reshuffle all coins that are still hidden. That is, he can use a magic spell to redistribute the coins among all hats that still were not revealed, including the hat you just selected. After reshuffling, each hat must again contain at most one coin. After you make a guess and the magician reshuffles the hidden coins, the hat you selected is flipped upside down (and remains in this state until the end of the game). If it contained a coin, the coin remains in the hat, but it is now visible and the magician cannot move it in the future. If it did not contain a coin, the magician can't ever put a coin into this hat anymore. Furthermore, the magician has given you one more set of guarantees. At any moment in the game, the following constraints will all be satisfied:

- For each row, the number of hats in the row plus the number of coins in the row will be an even number.
- For each column, the number of hats in the column plus the number of coins in the column will be an even number.

You are given a String[] **board** representing the checkerboard. The j-th character of the i-th element of **board** is 'H' when there is a hat at row i column j of the checkerboard and '.' otherwise. You are also given an int[] **coins** representing the different coins that are hidden under hats. You are also given an int **numGuesses** representing the number of guesses that you get.At the end of the game you get to keep all the coins that were revealed while playing. Your goal is to maximize the total value of the coins you get. The magician's goal is to minimize the total value of the coins you get. If it is not possible to hide all the coins in **coins** under the hats on the given board while meeting all the constraints above, return -1. Otherwise, return the total value of coins you'll get, assuming both you and the magician play optimally.

Definition

Class:	MagicalHats
Method:	findMaximumReward
Parameters:	String[], int[], int
Returns:	int
Method signature:	int findMaximumReward(String[] board, int[] coins, int numGuesses)

(be sure your method is public)

Constraints

- **board** will contain between 1 and 13 elements, inclusive.
- Each element of **board** will contain between 1 and 13 characters, inclusive.
- Each element of **board** will contain the same number of characters.
- Each character of each element of **board** will be either 'H' or '.'.
- **board** will contain at most 13 occurrences of the character 'H'.
- **coins** will contains between 1 and 13 elements, inclusive.
- Each element of **coins** will be between 1 and 10,000, inclusive.
- There will always be at least as many 'H' characters in **board** as elements in **coins**.
- **numGuesses** will be between 1 and the number of 'H' characters in **board**, inclusive.

Examples

1)

```
{"H"}
{1}
1
Returns: 1
```

One guess for one hat. The reward is just the contents of the hat.

2)

```
{"HHH",
 "H.H",
 "HH."}
{9}
1
Returns: 9
```

The only position the 9 coin can possibly be in is the top left corner.

3)

```
{"HH",
 "HH"}
{1,2,3,4}
3
Returns: 6
```

The magician manages to always give you the worst possible reward regardless of how you guess.

4)

```
{"HHH",
 "HHH",
 "H.H"}
{13,13,13,13}
2
```

Returns: 13

5)

```
{"HHH",
 "HHH",
 "H.H"}
{13,13,13,13}
3
```

Returns: 26

6)

```
{"H.H.",
 ".H.H",
 "H.H."}
{17}
6
```

Returns: -1

7)

```
{"HHH",
 "H.H",
 "HHH",
 "H.H",
 "HHH"}
{33,337,1007,2403,5601,6003,9999}
5
```

Returns: 1377

8)

```
{".............",
 ".............",
 ".............",
 ".............",
 ".............",
 ".............",
 ".....H.H.....",
 "......H......",
 ".....H.H.....",
 ".............",
 ".............",
 ".............",
 "............."}
{22}
3
```

Returns: 22

2_43) Problem Statement

John and Brus think 4 and 7 are lucky numbers. They think 11, the sum of 4 and 7, is a very lucky number. They decided to make a multiple of 11. They went to a shop and bought N pieces of strings of digits. You are given String[] **pieces** containing N elements. The i-th element of **pieces** is the string of digits written on the i-th piece they bought. They want to make a multiple of 11 by concatenating all N pieces. The pieces are painted with different colors, so it's possible to distinguish two pieces even if they contain the same strings of digits. There are exactly N! ways to concatenate the pieces. How many of them are divisible by 11? Return this number modulo 1,000,000,007.

Definition

Class:	ElevenMultiples
Method:	countMultiples
Parameters:	String[]
Returns:	int
Method signature:	int countMultiples(String[] pieces)

(be sure your method is public)

Constraints

- **pieces** will contain between 1 and 50 elements, inclusive.
- Each element of **pieces** will contain between 1 and 50 characters, inclusive.
- Each character in **pieces** will be a digit ('0'-'9').
- No element of **pieces** will start with zero ('0').

Examples

1)

{"58", "2012", "123"}

Returns: 2

There are 6 ways to concatenate the pieces: "582012123", "581232012", "201258123", "201212358", "123582012", and "123201258". Only "582012123" and "201258123" are divisible by 11.

2)

{"1", "1111", "1", "11"}

Returns: 24

There are 24 ways to concatenate the pieces. Even though all of them result in the same number "11111111", they are considered distinct. This number is divisible by 11.

3)

{"4392548694373865979538938749895327 4"}

Returns: 1

This big number is divisible by 11.

3)

{"983", "4654", "98", "3269", "861", "30981"}

Returns: 96

4)

{"193", "8819", "40676", "97625892", "5719", "45515667", "32598836", "70559374", "38756", "724","93391", "942068", "506", "901150", "874", "895567", "7560480", "7427691", "799450", "85127"}

Returns: 537147821

5)

{"687045939630", "997856158148599044", "2014910234712225061", "9658113323175370226", "1584118137","67925153345598920", "6960366756", "863413844386808834", "799302243562410012", "44481835751","8004606814733183", "19623906615609", "238599983260581 62", "461385591582", "9261878982390119", "1569373294276", "318106951168934", "65389049931", "12791173342", "507877942026","3947173045690", "472425746178910", "524552931853595", "40771812249667850232", "563988469071932","28147819070", "797007158858587", "5716177008624223", "387565700495309324", "4716621063133318"}

Returns: 814880650

2_44) Problem Statement

King Dengklek has two N-sided dice. The dice are not necessarily equal. Each of the dice is fair: when rolled, each side will come up with probability 1/N. Some time ago, each side of each die was labeled by a positive integer between 1 and **X**, inclusive. The labels were all unique. In other words, exactly 2*N distinct integers were used to label the sides of the two dice. King Dengklek has been playing with the first die for a long time. Therefore, some of its labels were scratched off. The corresponding sides of the die are now empty. The second die still has all of its labels. The current labels on the first die are given in the int[] **firstDie**: if the i-th side has no label, **firstDie**[i] is 0, otherwise **firstDie**[i] is the label. The current labels on the second die are given in **secondDie**: the i-th side of the second die has the label **secondDie**[i]. In King Dengklek's favorite game, he takes one of the dice, his opponent takes the other, and they each roll the die they have. The one who throws a larger number is the winner. King Dengklek wants to fill in the missing labels on the first die. His goal is to fill them in such a way that his favorite game becomes as fair as possible. When filling in the missing labels, King Dengklek wants to preserve the two properties mentioned above: first, each integer between 1 and **X**, inclusive, may only occur at most once on the two dice. Second, no other labels may be used. However, there is an exception to the second rule: King Dengklek is also allowed to use the label 0. Moreover, he may even use this label multiple times.

You are given the int[]s **firstDie** and **secondDie**, and the int **X**. For a particular way to fill in the missing labels, let P be the probability that the player with the first die wins in the king's game. Find the labeling that minimizes the value |P - 0.5| and return the corresponding value of P. If there are two possible solutions, return the smaller one.

Definition

Class:	KingdomAndDice
Method:	newFairness
Parameters:	int[], int[], int
Returns:	double
Method signature:	double newFairness(int[] firstDie, int[] secondDie, int X)

(be sure your method is public)

Notes

- Your return value must have a relative or an absolute error of less than 1e-9.
- |x| denotes the the absolute value of x. For example, |3| = |-3| = 3.

Constraints

- **firstDie** and **secondDie** will contain the same number of elements, between 2 and 50, inclusive.
- **X** will be between 2*N and 1,000,000,000, inclusive, where N is the number of elements in **firstDie**.
- Each element of **firstDie** will be between 0 and **X**, inclusive.

- Each element of **secondDie** will be between 1 and **X**, inclusive.
- Each integer between 1 and **X**, inclusive, will occur at most once in **firstDie** and **secondDie** together.

Examples

1)

```
{0, 2, 7, 0}
{6, 3, 8, 10}
12
```

Returns: 0.4375

One possible solution is to relabel the first die into {4, 2, 7, 11}. The probability of winning against the second die will be 7/16.

2)

```
{0, 2, 7, 0}
{6, 3, 8, 10}
10
```

Returns: 0.375

One possible solution is to relabel the first die into {9, 2, 7, 5}. The probability of winning against the second die will be 3/8.

3)

```
{0, 0}
{5, 8}
47
```

Returns: 0.5

One possible solution is to relabel the first die into {10, 0}.

4)

```
{19, 50, 4}
{26, 100, 37}
1000
```

Returns: 0.2222222222222222

The first die does not have any missing labels.

5)

```
{6371, 0, 6256, 1852, 0, 0, 6317, 3004, 5218, 9012}
{1557, 6318, 1560, 4519, 2012, 6316, 6315, 1559, 8215, 1561}
10000
```

Returns: 0.49