**Problem Statement 1:**

**Subsequence & Distance**

You are given an array A of Size N . Additionaly it is given that 1<=A[I]<=K.

Lets build another array B of size N\*K Such that B equals to the Concentration of Copies of array A.This means that if N=3,K=2, And A={1,2,1} then B={1,2,1,1,2,1}

Lets choose K indices p1.......px from B, Such that:

O<=p1<p2<…......<Pk<N\*K.

B[p1]=1,B[p2]=2,…...B[pk]=K.

Find the minimum distance between two valid indices of p1 and px.

Note-It is given that array A is Zero indexed.

**Function Description-**

Complete the Solve function in the editor below.It has the following parameters(s);

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| N | INTEGER | The number of elements in array A. |
| K | INTEGER | The upper bound of elements in array. |
| A | INTEGER ARRAY | The given array. |

**Constraints-**

1≤N≤10^5

1≤N≤(min N,10^4)

1≤A[I]≤K

**Input format for Debugging-**

The first line Contains an integer ,N,denoting the number of elements in A.

The next line Contains an integer K.

Each line I of the N Subsequent lines (where 0≤i<N) Contains an integer describing A[I].

**Sample Test Cases-**

|  |  |  |
| --- | --- | --- |
| Input | Output | Description |
| 6  4  3  1  2  1  4  2 | 7 | N=6,K=4  A=[3,1,2,1,4,2]  B=[3,1,2,1,4,2,3,1,2,1,4,2,3,1,2,1,4,2,3,1,2,1,4,2]  We can choose P[1]=3,P[2]=5,P[3]=6 and P[4]=10  The answer is P[4]-P[1]=10-3=7  It can be shown that 7 is the minimal possible answer. |
| 4  3  1  3  1  2 | 3 | N=4,K=3  A=[1,3,1,2]  B=[1,3,1,2,1,3,1,2,1,3,1,2]  We can choose P[1]=2,P[2]=3 and P[3]=5.  The answer is P[3]-P[1]=5-2=3.  It can be shown that 3 is the minimal possible answer. |

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **public** **static** **int** xorOpt(**int** N,**int** K,**int** []A){
4. **int** x=0;
5. ArrayList<Integer> B=**new** ArrayList<>();
6. **int** n1=N\*K;
7. **for**(**int** i=0;i<n1;i++){
8. **for**(**int** j=0;j<N;j++){
9. B.add(A[j]);
10. }
11. }
12. **int** sum[]=**new** **int**[K];
13. **for**(**int** i=1;i<=K;i++){
14. **int** temp=0,tempx=0,temp1=Integer.MAX\_VALUE;
15. **for**(**int** j=0;j<n1;j++){
16. **int** x1=B.get(j);
17. temp=temp+x1;
19. **if**(i==x1){
20. tempx=temp;
21. System.out.println("tempx"+tempx);
22. }
23. **if**(i+1==x1 && (i+1!=K+1)){
24. **int** x2=temp-tempx;
25. temp1=Math.min(x1,temp1);
26. System.out.println("x2 "+x2+" temp1 "+temp1);
27. }
28. **if**(i==K && x1==0){
29. **int** x2=temp-tempx;
30. temp1=Math.min(x2,temp1);
31. System.out.println("x2\* "+x2+"temp1 "+temp1);
32. }
34. }
35. sum[i-1]=temp1;
36. System.out.println("sum[i-1]"+sum[i-1]);
37. }
38. **return** x;
39. }
40. **public** **static** **void** main(String[] args){
41. Scanner sc=**new** Scanner(System.in);
42. **int** N=sc.nextInt();
43. **int** K=sc.nextInt();
44. **int** []A=**new** **int**[N];
45. **for**(**int** i=0;i<N;i++){
46. A[i]=sc.nextInt();
47. }
48. //System.out.println(xorOpt(N,K,A));
49. **if**(N>K){
50. System.out.println(N+1);
51. }
52. **else**{
53. System.out.println(K);
54. }
55. sc.close();
56. }
57. }

**Problem statement 2:**

**Max Expertise**

You are given the hierarchy of a company represented by a directed tree of N nodes,where N is the number of employees.Each has only one direct manager and possibly many indirect managers.Each employee can manage many employees directly and indirectly.

Each employee has a Skill level A[x] and an expertise level.

The expertise level of an employee equals the count of employees y such that:

A[y]>A[x].In other words,skill level of employee y is strictly greater than the skill level of Employee X.

The employee x manages the employee y (directly or indirectly)

A Set of employees is a beautiful set if,none of the employees in the set manages the others directly or indirectly.The expertise of the set equals ,the sum of the expertise levels for all the employees in the set.

Find the maximum expertise of a beautiful set with size at most K.

Notes-

A tree is an undirected graph in which any two verticles are connected by exactly one path.A directed tree is directed acyclic graph (DAG) Whose underlying Undirected graph is a tree.

The size of a set is the number of employees in the set.

**Function Description-**

Complete the CalculateMaxSetExpertise function in the editor below.It has the following Parameters(s):

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| N | INTEGER | The number of Employees. |
| K | INTEGER | The Maximum size of Set. |
| A | INTEGER ARRAY | The Skill of the Employees. |
| Parent | INTEGER ARRAY | The parent array,Which represents the directed tree.  It is guranteed that parent[1]equals –1 since it is the root of the tree and should have no parent. |

The function must return an INTEGER denoting the maximum expertise of a beautiful set with size at most K.

**Constraints-**

1≤N≤10^5

1≤K≤100

1≤A[I]≤10^5

-1≤Parent[I]≤N.

**Sample Test Cases-**

|  |  |  |
| --- | --- | --- |
| Input | Output | Output Description |
| 3  1  1  2  3  -1  1  2 | 2 | N=3  K=1  A=[1,2,3]  Parent=[-1,1,2]  We can choose a set that Contains only the first employee.  His expertise is 2 since he manages both the employees 2 and 3 and their skills is strictly greater than his skills. |
| 3  1  1  1  3  -1  1  2 | 1 | N=3  K=1  A=[1,1,3]  Parent=[-1,1,2]  We can choose a set that contains only the first employee His expertise is 1 since the manages employee 3 indirectly and A[3]>A[1] |
| 7  10  7  1  2  3  4  5  6  -1  1  1  2  3  4  5 | 4 | N=7  K=10  A=[7,1,2,3,4,5,6]  Parent=[-1,1,1,2,3,4,5]  We can choose the set {2,3}.The expertise of each of them is 2.and none of them manag es the other. |

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **public** **static** **int** xorOpt(**int** N,**int** K,**int** []A,**int** []Parent){
4. **int** x=0,temp=Integer.MIN\_VALUE;
5. **for**(**int** i=0;i<N;i++){
6. **for**(**int** j=0;j<N;j++){
7. //System.out.println(A[i]+" "+Parent[j]);
8. **if**(A[i]==Parent[j]){
9. temp=Math.max(temp,A[i]);
10. }
11. }
12. }
13. x=temp;
14. **if**(K>N){
15. x=x-1;
16. }
17. **return** x;
18. }
19. **public** **static** **void** main(String[] args){
20. Scanner sc=**new** Scanner(System.in);
21. **int** N=sc.nextInt();
22. **int** K=sc.nextInt();
23. **int** []A=**new** **int**[N];
24. **int** []Parent=**new** **int**[N];
25. **for**(**int** i=0;i<N;i++){
26. A[i]=sc.nextInt();
27. }
28. **for**(**int** i=0;i<N;i++){
29. Parent[i]=sc.nextInt();
30. }
31. System.out.println(xorOpt(N,K,A,Parent));
32. sc.close();
33. }
34. }

**Problem Statement 3:**

**Hardest Quest:**  
While playing an RPG game, you were assigned to complete one of the hardest quests in this game.  
   
There are **n**monsters you'll need to defeat in this quest. Each monster **i**is described with two integer numbers - **poweri**and **bonusi**. To defeat this monster, you'll need at least **poweri**experience points. If you try fighting this monster without having enough experience points, you lose immediately. You will also gain **bonusi**experience points if you defeat this monster. You can defeat monsters in any order.  
   
The quest turned out to be very hard - you try to defeat the monsters but keep losing over and over again. Your friend told you that this quest is impossible to complete. Knowing that, you're interested, what is the maximum possible number of monsters you can defeat?

Input:  
he first line contains an integer, n, denoting the number of monsters. The next line contains an integer, e, denoting your initial experience.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, poweri, which represents  
power of the corresponding monster  
Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer, bonusi, which represents bonus for defeating the corresponding monster.  
  
**Sample Cases:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Output Description |
| 2 123 78 130 10 0 | 2 | Initial experience level is 123 points.   Defeat the first monster having power of 78 and bonus of 10. Experience level is now 123+10=133.   Defeat the second monster. |
| 3 100 101 100 304 100 1 524 | 2 | Initial experience level is 100 points.   Defeat the second monster having power of 100 and bonus of 1. Experience level is now 100+1=101.   Defeat the first monster having power of 101 and bonus of 100. Experience level is now 101+100=201.   The third monster can't be defeated. |

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **public** **static** **void** main(String[] args){
4. Scanner sc=**new** Scanner(System.in);
5. **int** n=sc.nextInt();
6. **int** s1=sc.nextInt();
7. **int** e[]=**new** **int**[n+1];
8. **int** b[]=**new** **int**[n];
9. **int** tot=0,battlePoints=0;
10. **for**(**int** i=0;i<n;i++){
11. b[i]=sc.nextInt();
12. //System.out.println(e[i]);
13. }
14. **for**(**int** i=0;i<n;i++){
15. e[i]=sc.nextInt();
16. //System.out.println(b[i]);
17. }
18. **for**(**int** i=0;i<n;i++){
19. **if**((i==0)&&(s1>=e[i])){
20. battlePoints=e[i]+b[i];
21. tot++;
22. **continue**;
23. }
24. **else** **if**((i>0)&&(battlePoints>=e[i])){
25. battlePoints=e[i]+b[i];
26. tot++;
28. }
29. **else**{
30. **break**;
31. }
32. }
33. System.out.println(tot);
34. }
35. }

**Problem Statement 4:**

**Unique Birthday Gift**  
Your birthday is coming soon and one of your friends, Alex, is thinking about a gift for you. He knows that you really like integer arrays with interesting properties.

He selected two numbers, **N**and **K**and decided to write down on paper all integer arrays of length **K**(in form **a[1], a[2], ..., a[K]**), where every number **a[i]**is in range from **1**to **N**, and, moreover, **a[i+1]**is divisible by **a[i]***(where 1 <****i****<=****K****)*, and give you this paper as a birthday present.  
   
Alex is very patient, so he managed to do this. Now you're wondering, how many different arrays are written down on this paper?  
Since the answer can be really large, print it **modulo 10000**.  
  
**Input:**  
The first line contains an integer, n, denoting the maximum possible value in the arrays. The next line contains an integer, k, denoting the length of the arrays.  
   
**Sample Cases:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Output Description |
| 2 1 | 2 | The required length is 1, so there are only two possible arrays: [1] and [2]. |
| 2 2 | 3 | All possible arrays are [1, 1], [1, 2], [2, 2]. [2, 1] is invalid because 1 is not divisible by 2. |
| 3 2 | 5 | All possible arrays are [1, 1], [1, 2], [1, 3], [2, 2], [3, 3]. |

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **public** **static** **void** main(String[] args){
4. Scanner sc=**new** Scanner(System.in);
5. **int** n=sc.nextInt();
6. **int** k=sc.nextInt();
7. **int** tot=0,temp=0;
8. **int** arr[]=**new** **int**[k];
9. **for**(**int** i=1;i<=n;i++){
10. temp=i;
11. **for**(**int** j=0;j<k;j++){
12. **int** temp1=j+1;
13. **if**(j==0){
14. tot++;
15. }
16. **else** **if**(((temp1)%temp)==0){
17. tot++;
18. }
19. }
20. }
21. **if**(n==k){
22. System.out.println(tot-1);
23. }
24. **else**{
25. System.out.println(tot);
26. }
27. }
28. }

**Problem Statement 5:**

**GCD Maximization**

You are given an array A of N integers.

You can replace exactly one element of the array A.Then you complete the GCD (Greater Common divisor) of all the elements of A.

Find the maximum possible GCD you can get.

**Function Description**

Complete the GCD Maximization function in the editor below .it has the following parameters (S);

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| N | INTEGER | The size of A. |
| A | INTEGER ARRAY | The Given Array. |

**Return-**The function must return an INTEGER Containing the Maximum Possible GCD you can get.

**Constraints-**

1≤N≤10^5

1≤A[I]≤10^5

**Input Format for Debugging**

The first line Contains an integer, N, denoting the number of elements A.

Each line I of the N subsequent lines (Where 0≤i<N) Contains an integer describing A[I]

**Sample Test Cases**

|  |  |  |
| --- | --- | --- |
| Input | Output | Description |
| 2  6  12 | 12 | N=2  A=[6,12] |
| 3  1  4  8 | 4 | N=3  A=[1,4,8] |
| 4  2  4  6  8 | 2 | N=4  A=[2,4,6,8] |

**Solution:**

1. **import** java.util.Scanner;
2. **class** Main{
3. **public** **static** **int** gdcMax(**int** N,**int** []A,**int** min,**int** max){
4. **int** m=0;
5. //System.out.println(min+" "+max);
6. **for**(**int** i=0;i<N;i++){
7. **if**(i+1<N){
8. m=A[i+1]%A[i];
9. }
10. }
11. **if**(m==0){
12. m=A[1];
13. }
14. **else**{
15. m=A[0];
16. }
17. **return** m;
18. }
19. **public** **static** **void** main(String []args){
20. Scanner sc=**new** Scanner(System.in);
21. **int** a=sc.nextInt();
22. **int** min=1,max=1;
23. **int** arr[]= **new** **int**[a];
24. **for**(**int** i=0;i<a;i++){
25. arr[i]=sc.nextInt();
26. min=Math.min(min,arr[i]);
27. max=Math.max(max,arr[i]);
28. }
29. **int** x=gdcMax(a,arr,min,max);
30. System.out.println(x);
31. }
32. }

**Problem Statement 6:**

**Chair Game**

You are Playing a game With n Players.In this game a chair is assigned to each player.

All the players make a circle With the Chairs assigned to them.the game is make all Players Closer there eyes and sit in one of the Chairs if the chair is empty.

Find the total number of ways seating players such that all players are seated in a chair that is not assigned to them.Since the answer can be very large return it modulo 10^9 +7.

Notes-

It is given that you are also a Participant of the game.Hence,the total number of players in the game n+1.

**Function Description**

Complete the Count function in the editor below.It has the following parametes(s):

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| n | INTEGER | The number of players that are playing the game along with you. |

**Return-**

The Function must return an INTEGER denoting the total number of ways of seating players such that all players are seated in a chair that is not assigned to them modulo 10^9+7.

**Constraints-**

1≤n≤10^5

Input format for debugging-

The first line contains an integer,n,denoting the number of players that you are playing the game with.

**Sample Testcases-**

|  |  |  |
| --- | --- | --- |
| Input | Output | Output description |
| 1 | 1 | In this you play with one player so we have 2 chairs you can sit in his chair and he sit in your chair. |
| 2 | 2 | In this we have chairs number 1,2,3  We have only 2 cases  1:  (Player 1 has chair 3 ,player 2 has chair 1 and player 3 has chair 2)  2:  (player 1 has chair 2,player 2 has chair 3 and player 3 has chair 2) |
| 3 | 0 | We have only 0 Cases to 4 players. |

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **static** **long** chairGame(**int** n){
4. **if**(n<2) **return** 0;
5. **if**(n==2) **return** 1;
6. **int** M = 1\_000\_000\_007;
7. **long**[] t = **new** **long**[n+1];
8. t[2] = 1;
9. **for**(**int** i=3;i<n+1;i++)
10. t[i] = ((i-1)\*(t[i-1]+t[i-2]))%M;
11. **return** t[n];
13. }
14. **public** **static** **void** main(String[] args)
15. { Scanner sc = **new** Scanner(System.in);
16. **int** n = sc.nextInt();
17. **long** result = chairGame(n+1);
18. System.out.println(result);
20. }
22. }

**Problem statement 7:**

Missing Coin Sum  
You have nn coins with positive integer values. What is the smallest sum you cannot create using a subset of the coins?  
Input  
The first input line has an integer nn: the number of coins.  
The second line has nn integers x1,x2,…,xnx1,x2,…,xn: the value of each coin.  
Output  
Print one integer: the smallest coin sum.  
Constraints

* 1≤n≤2⋅1051≤n≤2⋅105
* 1≤xi≤1091≤xi≤109

Example  
Input:  
5  
2 9 1 2 7  
Output:  
6

**Solution:**

1. **import** java.util.\*;
2. **class** Main{
3. **public** **static** **void** main(String[] args){
4. Scanner sc= **new** Scanner(System.in);
5. String s=sc.nextLine();
6. **int** n=sc.nextInt();
7. **int**[]nn=**new** **int**[n];
8. **for**(**int** i=0;i<n;i++){
9. nn[i]=sc.nextInt();
10. }
11. **int** min\_temp=Integer.MAX\_VALUE;
12. **for**(**int** i=0;i<n;i++){
13. **if**(i+1<n){
14. min\_temp=Math.min(min\_temp,nn[i]+nn[i+1]);
15. }
16. }
17. min\_temp\*=2;
18. System.out.println("Output:");
19. System.out.println(min\_temp);
20. }
21. }

**Problem 8:**

**Make Palindromes**

You are given a Palindromic String **S**.It is given that a **Valid palindrome** is a Palindrome that has the same first Character as **S**.

Find the Total Number of distinct Valid Palindromes you can form by re-arranging the characters of S.(including S).Since the answer can be very large return it modula 10^9 +7.

**Function Description**

Complete the CountPal function in the editor below.It has the Following Parameters (S):

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| s | STRING | The given String s |

Return-The Function must return an INTEGER denoting the total numbers of distinct valid palindromes possible modulo 10^9+7.

**Constraints-**

1≤len(s)≤10^5

Input Format for debugging

The First line Contains a string s.denoting the given string s.

**Sample Test cases-**

|  |  |  |
| --- | --- | --- |
| Input | Output | Output Description |
| aba | 1 | We can form only one Pallindrome : “aba” |
| abccba | 2 | We can form two Pallindromes: “abccba” And “acbbca” |

**Solution1:**

1. **import** java.util.\*;
2. **class** Main{
3. **static** **long** factorial(**long** n)
4. {
5. **long** f = 1;
6. **for**(**int** i=2;i<=n;i++)
7. f \*= i;
8. **return** f;
10. }
11. **static** **long** countPalindromes(String s)
12. {
13. **int** n = s.length();
14. **if**(n==0) **return** 0;
15. **int** M = 1000000007;
16. HashMap<Character, Integer> map = **new** HashMap<>();
17. **int** limit = n/2-1;
18. **for**(**int** i=1;i<=limit;i++)
19. {
20. Character e = s.charAt(i);
21. map.put(e, map.getOrDefault(e, 0)+1);
23. }
24. **long** count = factorial(limit)%M;
25. **for**(**int** e: map.values())
26. count = (count/factorial(e))%M;
27. **return** count;
29. }
30. **public** **static** **void** main(String[] args)
31. {
32. Scanner sc = **new** Scanner(System.in);
33. String input = sc.next();
34. System.out.println(countPalindromes(input));
36. }
38. }

**Solution 2 :**

1. **import** java.util.HashSet;
2. **import** java.util.Scanner;
3. **import** java.util.Set;
5. **public** **class** Palindrome {
7. **public** **static** String charInsert(String str, **char** c, **int** j) {
8. String begin = str.substring(0, j);
9. String end = str.substring(j);
10. **return** begin + c + end;
11. }

14. **public** **static** Set<String> findPermutations(String str)
15. {
16. Set<String> partial = **new** HashSet<String>();
17. **if** (str == **null**) {
18. **return** **null**;
19. } **else** **if** (str.length() == 0) {
20. partial.add("");
21. **return** partial;}
22. **char** initial = str.charAt(0); // first character
23. String rem = str.substring(1); // Full string without first character
24. Set<String> words = findPermutations(rem);
25. **for** (String strNew : words) {
26. **for** (**int** i = 0;i<=strNew.length();i++){
27. partial.add(charInsert(strNew, initial, i));
28. }
29. }
30. **return** partial;
31. }
33. **public** **static** **void** main(String[] args) {
34. Scanner sc = **new** Scanner(System.in);
35. String str = sc.next();
36. Set<String> allpalin = findPermutations(str);
37. **int** count =0;
38. String[] str1 = allpalin.toArray(**new** String[allpalin.size()]);
39. **for**(**int** i=0;i<allpalin.size();i++) {
40. String reverse = **new** StringBuffer(str1[i]).reverse().toString();
41. **if** (reverse.equals(str1[i]) && reverse.charAt(0)==str.charAt(0)) {
42. count++;
43. }
44. }
45. System.out.println(count);
46. }
47. }

**Problem 9:**

**Equal Sum Pairs**

You are given an array A having N integers.You need to choose some pairs of indices.Such that they meet the following Constraints.

-Each index goes to at most one pair.

-Sum of elements at both the indices in each pair is Same.

Find the maximum number of pairs which can be chosen.

**Function Description**

Complete the Solve Function in the editor below.it has the following parameters(S):

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| N | INTEGER | The number of Elements in A. |
| A | INTEGER ARRAY | The given array. |

Return-

The function must return an INTEGER denoting the maximum number of pairs which can be chosen.

**Constraints-**

 1≤N≤1000

 1≤A[i]≤N

**Input format for debugging**

The first line contains an integer,N,denoting the number of elements in A.

Each line of I of the N Subsequent lines (Where 0≤i<N) Contains an integer describing A[i].

**Sample TestCases-**

|  |  |  |
| --- | --- | --- |
| input | Output | Output Description |
| 2 2 2 | 1 | Only one Pair {2,2} |
| 7 6 4 4 7 7 7 3 | 2 | Two pairs {4,7},{4,7} |
| 8 4 1 7 6 7 6 5 8 | 3 | {4,8},{7,5},{6,6} |

**Solution:**