Alice and Bob are bored, so they decide to play a game with their wallets. Alice has a� coins in her wallet, while Bob has b� coins in his wallet.

Both players take turns playing, with Alice making the first move. In each turn, the player will perform the following steps **in order**:

1. Choose to exchange wallets with their opponent, or to keep their current wallets.
2. Remove 11 coin from the player's current wallet. The current wallet cannot have 00 coins before performing this step.

The player who cannot make a valid move on their turn loses. If both Alice and Bob play optimally, determine who will win the game.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** n,a,b;
5. cin>>n;
6. **while**(n--)
7. {
8. cin>>a>>b;
9. (a+b)%2==0?
10. cout<<**"Bob\n"**:cout<<**"Alice\n"**;
11. }
12. }

You are given a string s� of length n� consisting of characters "+" and "-". s� represents an array a� of length n� defined by ai=1��=1 if si=��= "+" and ai=−1��=−1 if si=��= "-".

You will do the following process to calculate your penalty:

1. Split a� into non-empty arrays b1,b2,…,bk�1,�2,…,�� such that b1+b2+…+bk=a†�1+�2+…+��=�†, where ++ denotes array concatenation.
2. The *penalty* of a single array is the absolute value of its sum multiplied by its length. In other words, for some array c� of length m�, its penalty is calculated as p(c)=|c1+c2+…+cm|⋅m�(�)=|�1+�2+…+��|⋅�.
3. The total penalty that you will receive is p(b1)+p(b2)+…+p(bk)�(�1)+�(�2)+…+�(��).

If you perform the above process optimally, find the minimum possible penalty you will receive.

†† Some valid ways to split a=[3,1,4,1,5]�=[3,1,4,1,5] into (b1,b2,…,bk)(�1,�2,…,��) are ([3],[1],[4],[1],[5])([3],[1],[4],[1],[5]), ([3,1],[4,1,5])([3,1],[4,1,5]) and ([3,1,4,1,5])([3,1,4,1,5]) while some invalid ways to split a� are ([3,1],[1,5])([3,1],[1,5]), ([3],[],[1,4],[1,5])([3],[],[1,4],[1,5]) and ([3,4],[5,1,1])([3,4],[5,1,1]).

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
4. **int** main()
5. {
6. **int** t; cin>>t;
7. **while**(t--)
8. {
9. **int** n; cin>>n;
10. string c; cin>>c;
11. **int** p=0,m=0;
12. **for**(**int** i=0 ; i<n ; i++)
13. {
14. **if**(c[i] == **'+'**) p++;
15. **else** m++;
16. }
17. cout<<n-2\*min(p,m)<<endl;
18. }
19. }

Alex is solving a problem. He has n� constraints on what the integer k� can be. There are three types of constraints:

1. k� must be **greater than or equal to** some integer x�;
2. k� must be **less than or equal to** some integer x�;
3. k� must be **not equal to** some integer x�.

Help Alex find the number of integers k� that satisfy all n� constraints. It is guaranteed that the **answer is finite** (there exists at least one constraint of type 11 and at least one constraint of type 22). Also, it is guaranteed that **no two constraints are the exact same**.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main() {
5. **int** t; cin>>t; **while**(t--){
6. **int** n; cin>>n;
7. **int** a,x;
8. **vector<int>**aa;
9. **int** mn = 1, mx = 1e9;
10. **while**(n--){
11. cin>>a>>x;
12. **if**(a==1) mn = max(mn,x);
13. **else** **if**(a==2) mx = min(mx,x);
14. **else** aa.push\_back(x);
15. }
16. **int** ans= mx-mn+1;
17. **for**(**int** xx : aa) **if**(xx <=mx && xx>=mn) ans--;
18. cout<<max(ans,0)<<endl;
19. }
20. }

Alice and Bob are playing a game. They have an array a1,a2,…,an�1,�2,…,��. The game consists of two steps:

* First, Alice will remove **at most** k� elements from the array.
* Second, Bob will multiply **at most** x� elements of the array by −1−1.

Alice wants to maximize the sum of elements of the array while Bob wants to minimize it. Find the sum of elements of the array after the game if both players play optimally.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** t,n,x,y,a[200005];
4. **int** main(){
5. scanf(**"%d"**,&t);
6. **while**(t--){
7. scanf(**"%d%d%d"**,&n,&x,&y);
8. **for**(**int** i=1;i<=n;i++)scanf(**"%d"**,&a[i]);
9. sort(a+1,a+n+1);
10. **for**(**int** i=1;i<=n;i++)a[i]+=a[i-1];
11. **int** ans=-1e9;
12. **for**(**int** i=max(0,n-x);i<=n;i++)ans=max(ans,a[max(0,i-y)]-(a[i]-a[max(0,i-y)]));
13. printf(**"%d\n"**,ans);
14. }
15. **return** 0;
16. }

A square of positive (strictly greater than 00) area is located on the coordinate plane, with sides parallel to the coordinate axes. You are given the coordinates of its corners, in random order. Your task is to find the area of the square.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** (t);cin>>t;
5. **while**(t--){
6. **int** a,b,c,d;
7. cin>>a>>d>>b>>d>>c>>d>>d>>d;
8. cout<<(b!=a ? (a-b)\*(a-b) : (a-c)\*(a-c))<<endl;}
9. **return** 0;
10. }

*In order to test the hypothesis about the cats, the scientists must arrange the cats in the boxes in a specific way. Of course, they would like to test the hypothesis and publish a sensational article as quickly as possible, because they are too engrossed in the next hypothesis about the phone's battery charge.*

Scientists have n� boxes in which cats may or may not sit. Let the current state of the boxes be denoted by the sequence b1,…,bn�1,…,��: bi=1��=1 if there is a cat in box number i�, and bi=0��=0 otherwise.

Fortunately, the unlimited production of cats has already been established, so in one day, the scientists can perform one of the following operations:

* Take a new cat and place it in a box (for some i� such that bi=0��=0, assign bi=1��=1).
* Remove a cat from a box and send it into retirement (for some i� such that bi=1��=1, assign bi=0��=0).
* Move a cat from one box to another (for some i,j�,� such that bi=1,bj=0��=1,��=0, assign bi=0,bj=1��=0,��=1).

It has also been found that some boxes were immediately filled with cats. Therefore, the scientists know the initial position of the cats in the boxes s1,…,sn�1,…,�� and the desired position f1,…,fn�1,…,��.

Due to the large amount of paperwork, the scientists do not have time to solve this problem. Help them for the sake of science and indicate the minimum number of days required to test the hypothesis.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std ;
3. **int** main(){
4. **int** t; cin>>t;
5. **while**(t--){
6. string s,d;
7. **int** n,a=0,f=0;
8. cin>>n>>s>>d;
9. **for**(**int** i=0;i<n;i++){
10. **if**(s[i]==**'1'**&&d[i]==**'0'**)a++;
11. **if**(s[i]==**'0'**&&d[i]==**'1'**)f++;
12. }
13. cout<<max(a,f)<<endl;
14. }**return** 0;
15. }

You are given an integer n� and three strings a,b,c�,�,�, each consisting of n� lowercase Latin letters.

Let a template be a string t� consisting of n� lowercase and/or uppercase Latin letters. The string s� matches the template t� if the following conditions hold for all i� from 11 to n�:

* if the i�-th letter of the template is **lowercase**, then si�� must be **the same** as ti��;
* if the i�-th letter of the template is **uppercase**, then si�� must be **different** from the **lowercase version** of ti��. For example, if there is a letter 'A' in the template, you cannot use the letter 'a' in the corresponding position of the string.

Accordingly, the string doesn't match the template if the condition doesn't hold for at least one i�.

Determine whether there exists a template t� such that the strings a� and b� match it, while the string c� does not.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** T,n;
4. string a,b,c;
5. **signed** main(){
6. cin>>T;
7. **while**(T--){
8. cin>>n>>a>>b>>c;
9. **int** f=0;
10. **for**(**int** i=0;i<n;i++)**if**(a[i]!=c[i]&&b[i]!=c[i])f=1;
11. **if**(f)cout<<**"YES\n"**;
12. **else** cout<<**"NO\n"**;
13. }
14. }

You are given two positive integers n� and k�.

Your task is to find a string s� such that all possible strings of length n� that can be formed using the first k� lowercase English alphabets occur as a subsequence of s�.

If there are multiple answers, print the one with the smallest length. If there are still multiple answers, you may print any of them.

**Note:** A string a� is called a subsequence of another string b� if a� can be obtained by deleting some (possibly zero) characters from b� without changing the order of the remaining characters.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;
5. cin>>t;
6. **while**(t--){
7. **int** n,x;
8. cin>>n>>x;
9. **while**(n--){
10. **for**(**char** i=**'a'**;i<**'a'**+x;i++)
11. cout<<i;
12. }
13. cout<<endl;
14. }
15. }

Jay managed to create a problem of difficulty x� and decided to make it the second problem for Codeforces Round #921.

But Yash fears that this problem will make the contest highly unbalanced, and the coordinator will reject it. So, he decided to break it up into a problemset of n� sub-problems such that the difficulties of all the sub-problems are a positive integer and their sum is equal to x�.

The coordinator, Aleksey, defines the balance of a problemset as the [GCD](https://en.wikipedia.org/wiki/Greatest_common_divisor) of the difficulties of all sub-problems in the problemset.

Find the maximum balance that Yash can achieve if he chooses the difficulties of the sub-problems optimally.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;
5. cin>>t;
6. **while**(t--){
7. **int** n,x;
8. cin>>n>>x;
9. **while**(n%x!=0)x=(n-1)/(n/x)+1;
10. cout<<n/x<<endl;
11. }
12. }

A brick is a strip of size 1×k1×�, placed horizontally or vertically, where k� can be an arbitrary number that is at least 22 (k≥2�≥2).

A brick wall of size n×m�×� is such a way to place several bricks inside a rectangle n×m�×�, that all bricks lie either horizontally or vertically in the cells, do not cross the border of the rectangle, and that each cell of the n×m�×� rectangle belongs to exactly one brick. Here n� is the height of the rectangle n×m�×� and m� is the width. **Note** that there can be bricks with different values of k in the same brick wall.

The wall stability is the difference between the number of horizontal bricks and the number of vertical bricks. **Note** that if you used 00 horizontal bricks and 22 vertical ones, then the stability will be −2−2**, not**22.

What is the maximal possible stability of a wall of size n×m�×�?

It is guaranteed that under restrictions in the statement at least one n×m�×� wall exists.

Solve:

#include<bits/stdc++.h>

using namespace std;

int main()

{

int t;cin>>t;

while(t--)

{

int a,b;

cin>>a>>b;

cout<<a\*(b/2)<<endl;

}

}

You are given two permutations a� and b� of length n�. A permutation is an array of n� elements from 11 to n� where all elements are distinct. For example, an array [2,1,32,1,3] is a permutation, but [0,10,1] and [1,3,11,3,1] aren't.

You can (as many times as you want) choose two indices i� and j�, then swap ai�� with aj�� and bi�� with bj�� simultaneously.

You hate inversions, so you want to minimize the total number of inversions in both permutations.

An inversion in a permutation p� is a pair of indices (i,j)(�,�) such that i<j�<� and pi>pj��>��. For example, if p=[3,1,4,2,5]�=[3,1,4,2,5] then there are 33 inversions in it (the pairs of indices are (1,2)(1,2), (1,4)(1,4) and (3,4)(3,4)).

Solve:

1. **#include** **<iostream>**
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;
5. cin>>t;
6. **while**(t--){
7. **int** n;
8. cin>>n;
9. **int** a[n];
10. **int** b[n];
11. **for**(**int** i=0;i<n;i++){
12. cin>>a[i];
13. }
14. **for**(**int** i=0;i<n;i++){
15. cin>>b[a[i]-1];
16. }
17. **for**(**int** i=0;i<n;i++){
18. cout<<i+1<<**" "**;
19. }
20. cout<<**"\n"**;
21. **for**(**int** i=0;i<n;i++){
22. cout<<b[i]<<**" "**;
23. }
24. }
25. **return** 0;
26. }

You have a horizontal strip of n� cells. Each cell is either white or black.

You can choose a **continuous** segment of cells once and paint them all white. After this action, all the black cells in this segment will become white, and the white ones will remain white.

What is the minimum length of the segment that needs to be painted white in order for all n� cells to become white?

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **int** ll;
4. **int** main(){
5. **int** t;
6. cin>>t;
7. **while**(t--){
8. **int** n;
9. string s;
10. cin>>n>>s;
11. **int** a = s.find(**'B'**);
12. **int** b = s.rfind(**'B'**);
13. cout<<b-a+1<<endl;
14. }
15. }

Polycarp lost the string s� of length n� consisting of lowercase Latin letters, but he still has its *trace*.

The *trace* of the string s� is an array a� of n� integers, where ai�� is the number of such indices j� (j<i�<�) that si=sj��=��. For example, the *trace* of the string abracadabra is the array [0,0,0,1,0,2,0,3,1,1,40,0,0,1,0,2,0,3,1,1,4].

Given a *trace* of a string, find **any** string s� from which it could have been obtained. The string s� should consist only of lowercase Latin letters a-z.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** a,c;
6. cin>>a;
7. **while**(a--)
8. {
9. **int** b;
10. cin>>b;
11. **int** arr[200001]={0};
12. **while**(b--)
13. {
14. cin>>c;
15. cout<<**char**(**'a'**+arr[c]);
16. arr[c]++;
17. }
18. cout<<endl;

}

}

Given an array a� of n� integers, an array b� of m� integers, and an even number k�.

Your task is to determine whether it is possible to choose **exactly** k2�2 elements from both arrays in such a way that among the chosen elements, every integer from 11 to k� is included.

For example:

* If a=[2,3,8,5,6,5]�=[2,3,8,5,6,5], b=[1,3,4,10,5]�=[1,3,4,10,5], k=6�=6, then it is possible to choose elements with values 2,3,62,3,6 from array a� and elements with values 1,4,51,4,5 from array b�. In this case, all numbers from 11 to k=6�=6 will be included among the chosen elements.
* If a=[2,3,4,5,6,5]�=[2,3,4,5,6,5], b=[1,3,8,10,3]�=[1,3,8,10,3], k=6�=6, then it is not possible to choose elements in the required way.

Note that you are not required to find a way to choose the elements — your program should only check whether it is possible to choose the elements in the required way.

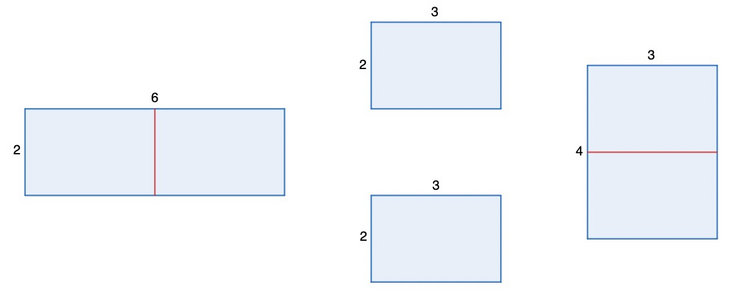
Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** t,n,m,k,x; **set<int>**a,b,c;
4. **int** main(){
5. cin>>t; **while**(t--){
6. cin>>n>>m>>k;
7. **for**(**int** i=0;i<n;++i){cin>>x; **if**(x<=k){a.insert(x);c.insert(x);}}
8. **for**(**int** i=0;i<m;++i){cin>>x; **if**(x<=k){b.insert(x);c.insert(x);}}
9. **if**(c.size()==k&&2\*a.size()>=k&&2\*b.size()>=k)cout<<**"yes"**<<endl; **else** cout<<**"no"**<<endl;
10. a.clear(); b.clear(); c.clear();
11. }
12. **return** 0;
13. }

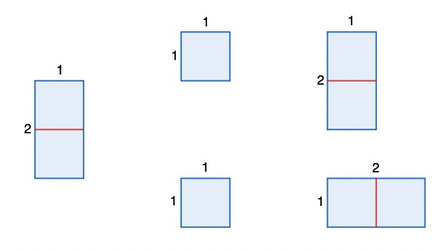
Bob has a rectangle of size a×b�×�. He tries to cut this rectangle into two rectangles with integer sides by making a cut parallel to one of the sides of the original rectangle. Then Bob tries to form some **other** rectangle from the two resulting rectangles, and he can rotate and move these two rectangles as he wishes.

Note that if two rectangles differ only by a 90∘90∘ rotation, they are considered **the same**. For example, the rectangles 6×46×4 and 4×64×6 are considered the same.

Thus, from the 2×62×6 rectangle, another rectangle can be formed, because it can be cut into two 2×32×3 rectangles, and then these two rectangles can be used to form the 4×34×3 rectangle, which is different from the 2×62×6 rectangle.



However, from the 2×12×1 rectangle, another rectangle cannot be formed, because it can only be cut into two rectangles of 1×11×1, and from these, only the 1×21×2 and 2×12×1 rectangles can be formed, which are considered the same.



Help Bob determine if he can obtain some other rectangle, or if he is just wasting his time.

Solve:

1. **#include** **<iostream>**
2. **int** main()
3. {
4. **int** t;std::cin>>t;
5. **while**(t--)
6. {
7. **int** a,b;
8. std::cin>>a>>b;
9. std::cout<<((a%2==0&&2\*b!=a)||(b%2==0&&2\*a!=b)?**"YES\n"**:**"NO\n"**);
10. }
11. }

Vasya has two hobbies — adding permutations†† to arrays and finding the most frequently occurring element. Recently, he found an array a� and decided to find out the maximum number of elements equal to the same number in the array a� that he can obtain after adding some permutation to the array a�.

More formally, Vasya must choose exactly one permutation p1,p2,p3,…,pn�1,�2,�3,…,�� of length n�, and then change the elements of the array a� according to the rule ai:=ai+pi��:=��+��. After that, Vasya counts how many times each number occurs in the array a� and takes the maximum of these values. You need to determine the maximum value he can obtain.

††A permutation of length n� is an array consisting of n� distinct integers from 11 to n� in arbitrary order. For example, [2,3,1,5,4][2,3,1,5,4] is a permutation, but [1,2,2][1,2,2] is not a permutation (22 appears twice in the array), and [1,3,4][1,3,4] is also not a permutation (n=3�=3 but there is 44 in the array).

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** n,a[200005];
4. **int** main(){
5. **int** t;
6. cin>>t;
7. **while**(t--){
8. cin>>n;
9. **for**(**int** i=0;i<n;i++)cin>>a[i];
10. sort(a,a+n);
11. **int** m=unique(a,a+n)-a;
12. **int** ans=0;
13. **for**(**int** i=0;i<m;i++){
14. **int** pos=lower\_bound(a,a+m,a[i]+n)-a;
15. ans=max(ans,pos-i);
16. }
17. cout<<ans<<endl;
18. }
19. }

output

standard output

Nikita had a word consisting of exactly 33 lowercase Latin letters. The letters in the Latin alphabet are numbered from 11 to 2626, where the letter "a" has the index 11, and the letter "z" has the index 2626.

He encoded this word as the sum of the positions of all the characters in the alphabet. For example, the word "cat" he would encode as the integer 3+1+20=243+1+20=24, because the letter "c" has the index 33 in the alphabet, the letter "a" has the index 11, and the letter "t" has the index 2020.

However, this encoding turned out to be ambiguous! For example, when encoding the word "ava", the integer 1+22+1=241+22+1=24 is also obtained.

Determine the lexicographically smallest word of 33 letters that could have been encoded.

A string a� is lexicographically smaller than a string b� if and only if one of the following holds:

* a� is a prefix of b�, but a≠b�≠�;
* in the first position where a� and b� differ, the string a� has a letter that appears earlier in the alphabet than the corresponding letter in b�.

solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
4. main(){
5. **int** t,n;scanf(**"%d"**,&t);**while**(t--){
6. scanf(**"%d"**,&n);**if**(n>52){printf(**"%czz\n"**,n+44);}**else** **if**(n>27){printf(**"a%cz\n"**,n+69);}
7. **else** **if**(n<29){printf(**"aa%c\n"**,n+94);}
8. }}

There are n� containers of water lined up, numbered from left to right from 11 to n�. Each container can hold any amount of water; initially, the i�-th container contains ai�� units of water. The sum of ai�� is divisible by n�.

You can apply the following operation any (possibly zero) number of times: pour any amount of water from the i�-th container to the j�-th container, where i� must be **less** than j� (i.e. i<j�<�). Any index can be chosen as i� or j� any number of times.

Determine whether it is possible to make the amount of water in all containers the same using this operation.

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;
5. cin>>t;
6. **while**(t--)
7. {
8. **int** n,p=0,q=0;
9. cin>>n;
10. **vector**<**int** >a(n);
11. **for**(**auto** &v:a)
12. {
13. cin>>v;
14. p+=v;
16. }
17. **for**(**auto** v:a)
18. {
19. **if**((q+=v-p/n)<0)
20. **break**;
21. }
22. cout<<(q<0 ?**"NO"**:**"YES"**)<<endl;
23. }
24. }

Sasha decided to give his girlfriend an array a1,a2,…,an�1,�2,…,��. He found out that his girlfriend evaluates the *beauty* of the array as the sum of the values (ai−ai−1)(��−��−1) for all integers i� from 22 to n�.

Help Sasha and tell him the maximum beauty of the array a� that he can obtain, if he can rearrange its elements in any way.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;cin>>t;
5. **while**(t--){
6. **int** n; cin>>n; **int** a[n];
7. **for**(**int** i=0;i<n;i++) cin>>a[i];
8. sort(a,a+n);
9. cout<<a[n-1]-a[0]<<endl;
10. }
11. }

Even in kindergarten, Sasha liked a girl. Therefore, he wanted to give her a drawing and attract her attention.

As a drawing, he decided to draw a square grid of size n×n�×�, in which some cells are colored. But coloring the cells is difficult, so he wants to color as few cells as possible. But at the same time, he wants **at least** k� diagonals to have at least one colored cell. Note that the square grid of size n×n�×� has a total of 4n−24�−2 diagonals.

Help little Sasha to make the girl fall in love with him and tell him the minimum number of cells he needs to color.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** n,k;
4. **int** T;
5. **int** main(){
6. cin>>T;
7. **while**(T--){
8. cin>>n>>k;
9. **if**(k==4\*n-2||k%2)cout<<k/2+1<<**'\n'**;
10. **else** cout<<k/2<<**'\n'**;
11. }
12. }

There are 2n2� positive integers written on a whiteboard. Being bored, you decided to play a one-player game with the numbers on the whiteboard.

You start with a score of 00. You will increase your score by performing the following move **exactly** n� times:

* Choose two integers x� and y� that are written on the whiteboard.
* Add min(x,y)min(�,�) to your score.
* Erase x� and y� from the whiteboard.

Note that after performing the move n� times, there will be no more integers written on the whiteboard.

Find the maximum final score you can achieve if you optimally perform the n� moves.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
3. **int** t,n,a[105],ans;
4. **int** main() {
5. cin>>t;
6. **while** (t--){
7. cin>>n; ans=0;
8. **for** (**int** i=1; i<=2\*n; i++) cin>>a[i];
9. sort(a+1,a+2\*n+1);
10. **for** (**int** i=1; i<=2\*n; i+=2) ans+=a[i];
11. cout<<ans<<**'\n'**;
12. }
13. }

You are given a positive integer n�.

Find a permutation†† p� of length n� such that there do **not** exist two **distinct** indices i� and j� (1≤i,j<n1≤�,�<�; i≠j�≠�) such that pi�� divides pj�� and pi+1��+1 divides pj+1��+1.

Refer to the Notes section for some examples.

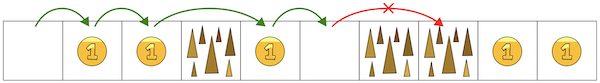
Under the constraints of this problem, it can be proven that at least one p� exists.

†† A permutation of length n� is an array consisting of n� distinct integers from 11 to n� in arbitrary order. For example, [2,3,1,5,4][2,3,1,5,4] is a permutation, but [1,2,2][1,2,2] is not a permutation (22 appears twice in the array), and [1,3,4][1,3,4] is also not a permutation (n=3�=3 but there is 44 in the array).

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
4. **int** T;
5. **int** N;
6. **int** v[1000000];
8. **int** main() {
9. cin >> T;
10. **while** (T--) {
11. cin >> N;
12. **for** ( **int** i=0; i<N; i++ ) {
13. cout << ( (i%2) ? (N - i/2) : (i/2 + 1) ) << **" "**;
14. }
15. cout << endl;
16. }
17. }

During your journey through computer universes, you stumbled upon a very interesting world. It is a path with n� consecutive cells, each of which can either be empty, contain thorns, or a coin. In one move, you can move one or two cells along the path, provided that the destination cell does not contain thorns (and belongs to the path). If you move to the cell with a coin, you pick it up.

Here, green arrows correspond to legal moves, and the red arrow corresponds to an illegal move.

You want to collect as many coins as possible. Find the maximum number of coins you can collect in the discovered world if you start in the leftmost cell of the path.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. string s;
4. **int** t,n,a;
5. **int** main()
6. {cin>>t;
7. **while**(t--)
8. {cin>>n>>s;
9. a=0;
10. **for**(**int** i=0;i<n;i++)
11. {**if**(s[i]==**'\*'**&&s[i+1]==**'\*'**)**break**;
12. **else** **if**(s[i]==**'@'**)a++;}
13. cout<<a<<endl;}}

The Chaya tribe believes that there are n� signs of the apocalypse. Over time, it has been found out that the i�-th sign occurs every ai�� years (in years ai��, 2⋅ai2⋅��, 3⋅ai3⋅��, ……).

According to the legends, for the apocalypse to happen, the signs must occur sequentially. That is, first they wait for the first sign to occur, then strictly after it, the second sign will occur, and so on. That is, if the i�-th sign occurred in the year x�, the tribe starts waiting for the occurrence of the (i+1)(�+1)-th sign, starting from the year x+1�+1.

In which year will the n�-th sign occur and the apocalypse will happen?

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. **int** main(){
4. **int** t,p,x,n;
5. cin>>t;
6. **while** (t--){
7. cin>>n;
8. cin>>p;
9. n--;
10. **while** (n--){
11. cin>>x;
12. p=p-p%x+x;
13. }
14. cout<<p<<endl;
15. }
16. }

Vladislav has a string of length 55, whose characters are each either AA or BB.

Which letter appears most frequently: AA or BB?

Solve:

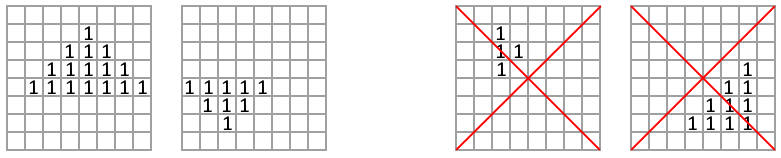
1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** t; cin>>t;
5. string s;
6. **while**(t--){
7. cin>>s;
8. cout<< (count(s.begin(),s.end(),**'A'**) >= 3? **'A'**:**'B'**)<<**'\n'**;
9. }
10. }

output

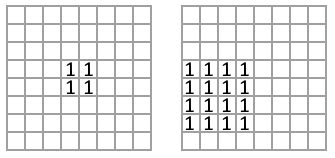
standard output

Vladislav has a binary square grid of n×n�×� cells. A triangle or a square is drawn on the grid with symbols 11. As he is too busy being cool, he asks you to tell him which shape is drawn on the grid.

* A *triangle* is a shape consisting of k� (k>1�>1) consecutive rows, where the i�-th row has 2⋅i−12⋅�−1 consecutive characters 11, and the central 1s are located in one column. An upside down triangle is also considered a valid triangle (but not rotated by 90 degrees).

Two left pictures contain examples of triangles: k=4�=4, k=3�=3. The two right pictures don't contain triangles.

* A *square* is a shape consisting of k� (k>1�>1) consecutive rows, where the i�-th row has k� consecutive characters 11, which are positioned at an equal distance from the left edge of the grid.

Examples of two squares: k=2�=2, k=4�=4.

For the given grid, determine the type of shape that is drawn on it.

Solve:

1. **#include** **<iostream>**
2. **#include** **<algorithm>**
3. **using** **namespace** std;
4. **int** main()
5. {
6. **int** t, n;
7. cin >> t;
8. **while** (t--)
9. {
10. cin >> n;
11. **long** **long** a[n];
12. **for** (**size\_t** i = 0; i < n; i++)
13. {
14. cin >> a[i];
15. }
16. sort(a, a + n);
17. cout << ((a[n - 2] == a[n - 1]) ? **"SQUARE"** : **"TRIANGLE"**) << endl;
18. }
19. **return** 0;
20. }

There is a ribbon divided into n� cells, numbered from 11 to n� from left to right. Each cell either contains a chip or is free.

You can perform the following operation any number of times (possibly zero): choose a chip and move it to the **closest free cell to the left**. You can choose any chip that you want, provided that there is at least one free cell to the left of it. When you move the chip, the cell where it was before the operation becomes free.

Your goal is to move the chips in such a way that **they form a single block, without any free cells between them**. What is the minimum number of operations you have to perform?

Solve:

1. **#include<cstdio>**
2. **#define** maxn 55
3. **using** **namespace** std;
4. **int** A[maxn];
5. **int** main(){
6. **int** T,n,p,cnt,q,a;
7. scanf(**"%d"**,&T);
8. **while**(T--){
9. cnt=0;p=q=0;
10. scanf(**"%d"**,&n);
11. **for**(**int** i=0;i<n;i++){
12. scanf(**"%d"**,&a);
13. **if**(a==1){
14. cnt++;q=i+1;**if**(!p)p=i+1;
15. }
16. }printf(**"%d\n"**,q-p-cnt+1);
17. }**return** 0;
18. }

You are playing a computer game. The current level of this game can be modeled as a straight line. Your character is in point 00 of this line. There are n� monsters trying to kill your character; the i�-th monster has health equal to ai�� and is initially in the point xi��.

Every second, the following happens:

* first, you fire up to k� bullets at monsters. Each bullet targets exactly one monster and decreases its health by 11. For each bullet, you choose its target arbitrary (for example, you can fire all bullets at one monster, fire all bullets at different monsters, or choose any other combination). Any monster can be targeted by a bullet, regardless of its position and any other factors;
* then, all alive monsters with health 00 or less die;
* then, all alive monsters move 11 point closer to you (monsters to the left of you increase their coordinates by 11, monsters to the right of you decrease their coordinates by 11). If any monster reaches your character (moves to the point 00), you lose.

Can you survive and kill all n� monsters without letting any of them reach your character?

Solve:

1. **#include**<bits/stdc++.h>
2. **#define** **int** **long** **long**
3. **using** **namespace** std;
4. **int** a[300001],s[300001];
5. **int** t,n,k,cnt,res;
6. **signed** main()
7. {
8. cin>>t;
9. **while**(t--)
10. {
11. cin>>n>>k;
12. **for**(**int** i=1;i<=n;++i) cin>>a[i],s[i]=0;
13. **for**(**int** i=1,x;i<=n;++i)
14. {
15. cin>>x;
16. x=abs(x);
17. s[x]+=a[i];
18. }
19. **int** now=0,res=1;
20. **for**(**int** i=1;i<=n;++i)
21. {
22. now+=s[i];
23. **if**(now>i\*k) res=0;
24. }
25. cout << (res?**"YES"**:**"NO"**) << endl;
26. }
27. }

You are given an array a� of n� integers. You must perform the following two operations on the array (the first, then the second):

1. Arbitrarily rearrange the elements of the array or leave the order of its elements unchanged.
2. Choose at most one contiguous segment of elements and replace the signs of all elements in this segment with their opposites. Formally, you can choose a pair of indices l,r�,� such that 1≤l≤r≤n1≤�≤�≤� and assign ai=−ai��=−�� for all l≤i≤r�≤�≤� (negate elements). Note that you may choose not to select a pair of indices and leave all the signs of the elements unchanged.

What is the **maximum sum of the array elements** after performing these two operations (the first, then the second)?

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. { **int** k,w,i,n,s,d,p3,p,a,t;
5. cin>>t;
6. **while**(t) {
7. s=0;
8. cin>>n;
9. **for**(i=0; i<n; i++) {cin>>a; s=s+abs(a);
10. }
11. cout<<s<<endl;
12. t--;}
13. }

You are given an array a1,a2,…,an�1,�2,…,��.

In one move, you can perform either of the following two operations:

* Choose an element from the array and remove it from the array. As a result, the length of the array decreases by 11;
* Choose an element from the array and increase its value by 11.

You can perform any number of moves. If the current array becomes empty, then no more moves can be made.

Your task is to find the **minimum** number of moves required to make the sum of the elements of the array a� divisible by 33. It is possible that you may need 00 moves.

Note that the sum of the elements of an empty array (an array of length 00) is equal to 00

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** t;
6. cin>>t;
7. **while**(t--)
8. {
9. **int** n,a,sum=0,count=0;
10. cin>>n;
11. **for**(**int** i=0;i<n;i++)
12. {
13. cin>>a;
14. sum+=a;
15. **if**(a%3==1) count=1;
16. }**if**(sum%3==0 )
17. cout<<**"0\n"**;
18. **else** **if**(sum%3==2)
19. cout<<**"1\n"**;
20. **else** **if**(count)
21. cout<<**"1\n"**;
22. **else**
23. cout<<**"2\n"**;
24. }
25. }

You are given three **positive** integers a�, b� and l� (a,b,l>0�,�,�>0).

It can be shown that there always exists a way to choose **non-negative** (i.e. ≥0≥0) integers k�, x�, and y� such that l=k⋅ax⋅by�=�⋅��⋅��.

Your task is to find the number of distinct possible values of k� across all such ways.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main(){
6. **set<int>**s;
7. **int** t,l,a,b;
8. cin>>t;
9. **while**(t--){
10. cin>>a>>b>>l;
11. s.clear();
12. **for**(**int** i=1;l%i==0;i\*=a)
13. **for**(**int** j=1;l%(i\*j)==0;j\*=b)
14. s.insert(l/i/j);
16. cout<<s.size()<<**'\n'**;
17. }
18. }

You are given an array a1,a2,…,an�1,�2,…,��. Initially, ai=i��=� for each 1≤i≤n1≤�≤�.

The operation swap(k)swap(�) for an integer k≥2�≥2 is defined as follows:

* Let d� be the largest divisor†† of k� which is not equal to k� itself. Then swap the elements ad�� and ak��.

Suppose you perform swap(i)swap(�) for each i=2,3,…,n�=2,3,…,� in this exact order. Find the position of 11 in the resulting array. In other words, find such j� that aj=1��=1 after performing these operations.

†† An integer x� is a divisor of y� if there exists an integer z� such that y=x⋅z�=�⋅�.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. **int** t,n,x;
4. **int** main()
5. {cin>>t;
6. **while**(t--)
7. {cin>>n;
8. x=1;
9. **while**(x<n)
10. {x\*=2;}
11. **if**(x==n)cout<<x<<endl;
12. **else** cout<<x/2<<endl;}}

You are given a 2×n2×� grid filled with zeros and ones. Let the number at the intersection of the i�-th row and the j�-th column be aij���.

There is a grasshopper at the top-left cell (1,1)(1,1) that can only jump one cell right or downwards. It wants to reach the bottom-right cell (2,n)(2,�). Consider the binary string of length n+1�+1 consisting of numbers written in cells of the path without changing their order.

Your goal is to:

1. Find the lexicographically smallest†† string you can attain by choosing any available path;
2. Find the number of paths that yield this lexicographically smallest string.

†† If two strings s� and t� have the same length, then s� is lexicographically smaller than t� if and only if in the first position where s� and t� differ, the string s� has a smaller element than the corresponding element in t�.

Solve:

1. **#include** **<iostream>**
2. **using** **namespace** std;
3. **#include**<bits/stdc++.h>
4. **int** main() {
5. **int** t;
6. cin>>t;
7. **while**(t--){
8. **int** n;
9. cin>>n;
10. string a,b;
11. cin>>a>>b;
12. string ans=**""**;
13. ans=a[0]+b;
14. **int** res=1;
15. **for**(**int** i=1;i<n;i++){
16. **if**(a[i]<b[i-1]){
17. ans[i]=a[i];
18. res=1;
19. }
20. **if**(b[i-1]<a[i]){
21. **break**;
22. }
23. **if**(a[i]==b[i-1]){
24. res++;
25. }
26. }
27. cout<<ans<<endl;
28. cout<<res<<endl;
29. }
30. **return** 0;
31. }

Given an array a� of n� elements, find the maximum value of the expression:

|ai−aj|+|aj−ak|+|ak−al|+|al−ai||��−��|+|��−��|+|��−��|+|��−��|

where i�, j�, k�, and l� are four **distinct** indices of the array a�, with 1≤i,j,k,l≤n1≤�,�,�,�≤�.

Here |x||�| denotes the absolute value of x�.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** t,n,a[1005];
4. **int** main(){
5. cin>>t;
6. **for**(**int** i=0;i<t;i++){
7. cin>>n;
8. **for**(**int** j=0;j<n;j++){
9. cin>>a[j];
10. }
11. sort(a,a+n);
12. cout<<2\*a[n-1]+2\*a[n-2]-2\*a[0]-2\*a[1]<<endl;
13. }
14. **return** 0;
15. }

You have 55 different types of coins, each with a value equal to one of the first 55 triangular numbers: 11, 33, 66, 1010, and 1515. These coin types are available in abundance. Your goal is to find the minimum number of these coins required such that their total value sums up to exactly n�.

We can show that the answer always exists.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** t,n,a[16] = {0,1,2,1,2,3,1,2,3,2,1,2,2,2,3};
5. cin >> t;
6. **while**(t--){
7. **int** n;
8. cin >> n;
9. **if**(n > 15)
10. a[5] = 1,a[8] = 2;
11. cout<< n/15 + a[n%15] << endl;
12. a[5] = 3,a[8] = 3;
13. }
14. }

Congratulations, you have been accepted to the Master's Assistance Center! However, you were extremely bored in class and got tired of doing nothing, so you came up with a game for yourself.

You are given a string s� and an **even** integer n�. There are two types of operations that you can apply to it:

1. Add the reversed string s� to the end of the string s� (for example, if s=�= cpm, then after applying the operation s=�= cpmmpc).
2. Reverse the current string s� (for example, if s=�= cpm, then after applying the operation s=�= mpc).

It is required to determine the lexicographically smallest†† string that can be obtained after applying **exactly** n� operations. Note that you can apply operations of different types in any order, but you must apply exactly n� operations in total.

††A string a� is lexicographically smaller than a string b� if and only if one of the following holds:

* a� is a prefix of b�, but a≠b�≠�;
* in the first position where a� and b� differ, the string a� has a letter that appears earlier in the alphabet than the corresponding letter in b�

solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** t;cin>>t;
6. **while**(t--)
7. {
8. **int** n;
9. string s;
10. cin>>n>>s;
11. s=min(s,string(s.rbegin(),s.rend())+s);
12. cout<<s<<endl;
13. }
14. **return** 0;
15. }

\

In the Master's Assistance Center, Nyam-Nyam was given a homework assignment in informatics.

There is an array a� of length n�, and you want to divide it into k>1�>1 subsegments†† in such a way that the MEX‡MEX‡ on each subsegment is equal to the same integer.

Help Nyam-Nyam find any suitable division, or determine that it does not exist.

††A division of an array into k� subsegments is defined as k� pairs of integers (l1,r1),(l2,r2),…,(lk,rk)(�1,�1),(�2,�2),…,(��,��) such that li≤ri��≤�� and for each 1≤j≤k−11≤�≤�−1, lj+1=rj+1��+1=��+1, and also l1=1�1=1 and rk=n��=�. These pairs represent the subsegments themselves.

‡MEX‡MEX of an array is the smallest non-negative integer that does not belong to the array.

For example:

* MEXMEX of the array [2,2,1][2,2,1] is 00, because 00 does not belong to the array.
* MEXMEX of the array [3,1,0,1][3,1,0,1] is 22, because 00 and 11 belong to the array, but 22 does not.
* MEXMEX of the array [0,3,1,2][0,3,1,2] is 44, because 00, 11, 22, and 33 belong to the array, but 44 does not.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main() {
6. **int** t;
7. cin>>t;
9. **while**(t--){
10. **int** n;
12. cin>>n;
13. **vector<int>**a(n),b(n);
14. **for**(**int** i=0;i<n;i++){
15. cin>>a[i];
16. b[a[i]]++;
17. }
18. **int** x=-1;
20. **for**(**int** i=0;i<n;i++){
21. **if**(b[i]==0){x=i;**break**;}
22. }
24. **if**(x==-1){
25. cout<<-1<<endl;**continue**;
26. }
28. **set<int>**s,ss;
29. **int** in=0,r=-1;
30. **for**(**int** i=0;i<n;i++){
31. **if**(a[i]<x)s.insert(a[i]);
32. **if**(s.size()==x){
33. in++;s=ss;
34. **if**(in==1)r=i;
35. }
36. }
38. **if**(in>1){
39. cout<<2<<endl;
40. cout<<1<<**" "**<<r+1<<endl;
41. cout<<r+2<<**" "**<<n<<endl;
42. }
43. **else**{
44. cout<<-1<<endl;
45. }
47. }
49. **return** 0;
50. }

Rudolf is going to visit Bernard, and he decided to take the metro to get to him. The ticket can be purchased at a machine that accepts exactly two coins, the sum of which does not exceed k�.

Rudolf has two pockets with coins. In the left pocket, there are n� coins with denominations b1,b2,…,bn�1,�2,…,��. In the right pocket, there are m� coins with denominations c1,c2,…,cm�1,�2,…,��. He wants to choose exactly one coin from the left pocket and exactly one coin from the right pocket (two coins in total).

Help Rudolf determine how many ways there are to select indices f� and s� such that bf+cs≤k��+��≤�.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** t;
6. cin>>t;
7. **while**(t--)
8. {
9. **int** n,i,m,k;
10. cin>>n>>m>>k;
11. **vector<int>**v(n);
12. **for**(i=0;i<n;i++)
13. {
14. cin>>v[i];
15. }
16. **int** o;
17. **long** **long** ans=0;
18. **for**(**int** j=0;j<m;j++)
19. {
20. cin>>o;
21. **for**(i=0;i<n;i++)
22. {
23. **if**((o+v[i])<=k) ans++;
24. }
25. }
26. cout<<ans<<endl;
27. }
29. }

Rudolf has an array a� of n� integers, the elements are numbered from 11 to n�.

In one operation, he can choose an index i� (2≤i≤n−12≤�≤�−1) and assign:

* ai−1=ai−1−1��−1=��−1−1
* ai=ai−2��=��−2
* ai+1=ai+1−1��+1=��+1−1

Rudolf can apply this operation any number of times. Any index i� can be used zero or more times.

Can he make all the elements of the array equal to zero using this operation?

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** t;
6. cin>>t;
7. **while**(t--)
8. {
9. **int** n;
10. cin>>n;
11. **int** a[n];
12. **for**(**int** i=0;i<n;i++)cin>>a[i];
13. **int** l=0,r=n-1;
14. **while**(a[l]>=0&&l<=r-2)
15. {
16. a[l+1]-=2\*a[l];a[l+2]-=a[l];a[l]=0;
17. l++;
18. }
19. **if**(l==r-1&&a[l]==0&&a[r]==0)cout<<**"YES"**<<endl;
20. **else** cout<<**"No"**<<endl;
21. }
23. }

udolf has a string s� of length n�. Rudolf considers the string s� to be ugly if it contains the substring†† "pie" or the substring "map", otherwise the string s� will be considered beautiful.

For example, "ppiee", "mmap", "dfpiefghmap" are ugly strings, while "mathp", "ppiiee" are beautiful strings.

Rudolf wants to shorten the string s� by removing some characters to make it beautiful.

The main character doesn't like to strain, so he asks you to make the string beautiful by removing the minimum number of characters. He can remove characters from **any** positions in the string (not just from the beginning or end of the string).

†† String a� is a substring of b� if there exists a **consecutive** segment of characters in string b� equal to a�.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main(){
5. **int** t; cin>>t;
6. **while**(t--){
7. **int** n; string s;
8. cin>>n>>s;
9. **int** cnt=0;
10. **for**(**int** i=0; i<n; i++){
11. **if**(s.substr(i,3) == **"map"** || s.substr(i,3)==**"pie"**) cnt++;
12. **if**(s.substr(i,5) == **"mapie"**) cnt--;
13. }
14. cout<<cnt<<endl;
15. }
16. }

You are given an integer n�.

Your task is to build a string of uppercase Latin letters. There must be exactly n� special characters in this string. Let's call a character *special* if it is equal to exactly one of its neighbors.

For example, there are 66 special characters in the AAABAACC string (at positions: 11, 33, 55, 66, 77 and 88).

Print any suitable string or report that there is no such string.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** n,T;
4. **int** main(){
5. cin>>T;
6. **while**(T--){
7. cin>>n;
8. **if**(n%2==1)cout<<**"NO"**;
9. **else**{
10. cout<<**"YES"**<<endl;
11. **for**(**int** i=1;i<=n/2;i++)cout<<**"AAB"**;
12. }
13. cout<<endl;
14. }
15. }

You are given an integer array a� of length n�.

You can perform the following operation any number of times (possibly zero): take any element of the array a�, which is at least 1010, delete it, and instead insert the digits that element consisted of in the same position, in order they appear in that element.

For example:

* if we apply this operation to the 33-rd element of the array [12,3,45,67][12,3,45,67], then the array becomes [12,3,4,5,67][12,3,4,5,67].
* if we apply this operation to the 22-nd element of the array [2,10][2,10], then the array becomes [2,1,0][2,1,0].

Your task is to determine whether it is possible to make a� sorted in non-descending order using the aforementioned operation **any number of times (possibly zero)**. In other words, you have to determine if it is possible to transform the array a� in such a way that a1≤a2≤⋯≤ak�1≤�2≤⋯≤��, where k� is the current length of the array a�.

Solve:

1. **#include** **<iostream>**
2. **using** **namespace** std;
4. **int** t,n,a[100010];
5. **bool** f;
7. **int** main()
8. {
9. cin>>t;
10. **while**(t--)
11. {
12. cin>>n;
13. f=1;
14. **for**(**int** i=1;i<=n;i++) cin>>a[i];
15. **for**(**int** i=n-1;i>0&&f;i--) **if**(a[i]>a[i+1]) **if**(a[i]/10>a[i]%10||a[i]%10>a[i+1]) f=0; **else** a[i]/=10;
16. **if**(f) cout<<**"YES\n"**; **else** cout<<**"NO\n"**;
17. }
18. **return** 0;
19. }

There are n� islands, numbered 1,2,…,n1,2,…,�. Initially, every pair of islands is connected by a bridge. Hence, there are a total of n(n−1)2�(�−1)2 bridges.

Everule lives on island 11 and enjoys visiting the other islands using bridges. Dominater has the power to destroy at most k� bridges to minimize the number of islands that Everule can reach using (possibly multiple) bridges.

Find the minimum number of islands (including island 11) that Everule can visit if Dominater destroys bridges optimally.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
4. **int** main(){
5. **int** t;
6. cin>>t;
7. **while**(t--){
8. **int** n,k;
9. cin>>n>>k;
10. cout<<(k>=n-1?1:n)<<endl<<endl;
11. }
12. }

You are given an integer n� and three strings a,b,c�,�,�, each consisting of n� lowercase Latin letters.

Let a template be a string t� consisting of n� lowercase and/or uppercase Latin letters. The string s� matches the template t� if the following conditions hold for all i� from 11 to n�:

* if the i�-th letter of the template is **lowercase**, then si�� must be **the same** as ti��;
* if the i�-th letter of the template is **uppercase**, then si�� must be **different** from the **lowercase version** of ti��. For example, if there is a letter 'A' in the template, you cannot use the letter 'a' in the corresponding position of the string.

Accordingly, the string doesn't match the template if the condition doesn't hold for at least one i�.

Determine whether there exists a template t� such that the strings a� and b� match it, while the string c� does not.

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** T,n;
4. string a,b,c;
5. **signed** main(){
6. cin>>T;
7. **while**(T--){
8. cin>>n>>a>>b>>c;
9. **int** f=0;
10. **for**(**int** i=0;i<n;i++)**if**(a[i]!=c[i]&&b[i]!=c[i])f=1;
11. **if**(f)cout<<**"YES\n"**;
12. **else** cout<<**"NO\n"**;
13. }
14. }

Stepan is a very busy person. Today he needs to send n� messages at moments m1,m2,…mn�1,�2,…�� (mi<mi+1��<��+1). Unfortunately, by the moment 00, his phone only has f� units of charge left. At the moment 00, the phone is turned on.

The phone loses a� units of charge for each unit of time it is on. Also, at any moment, Stepan can turn off the phone and turn it on later. This action consumes b� units of energy each time. Consider turning on and off to be instantaneous, so you can turn it on at moment x� and send a message at the same moment, and vice versa, send a message at moment x� and turn off the phone at the same moment.

If at any point the charge level drops to 00 (becomes ≤0≤0), it is impossible to send a message at that moment.

Since all messages are very important to Stepan, he wants to know if he can send all the messages without the possibility of charging the phone.

Solve:

1. **#include** **<iostream>**
2. **using** **namespace** std;
4. **int** main()
5. {
6. **int** t;
7. cin>>t;
8. **while**(t--)
9. {
10. **long** **long** n,f,a,b,x,m=0;
11. cin>>n>>f>>a>>b;
12. **for**(**int** i=0;i<n;i++)
13. {
14. cin>>x;
15. f-=min(b,(x-m)\*a);
16. m=x;
17. }
18. **if**(f<=0)
19. cout<<**"NO"**<<endl;
20. **else**
21. cout<<**"YES"**<<endl;
22. }
23. **return** 0;
24. }

Lura was bored and decided to make a simple language using the five letters aa, bb, cc, dd, ee. There are two types of letters:

* *vowels* — the letters aa and ee. They are represented by VV.
* *consonants* — the letters bb, cc, and dd. They are represented by CC.

There are two types of *syllables* in the language: CVCV (consonant followed by vowel) or CVCCVC (vowel with consonant before and after). For example, baba, cedced, babbab are syllables, but aaaa, edaeda, babababa are not.

A *word* in the language is a sequence of syllables. Lura has written a word in the language, but she doesn't know how to split it into syllables. Help her break the word into syllables.

For example, given the word bacedbabbacedbab, it would be split into syllables as ba.ced.babba.ced.bab (the dot .. represents a syllable boundary).

**Input**

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main(){
5. **int** t,n;
6. cin>>t;
7. **while**(t--){
8. string s;
9. cin>>n>>s;
10. cout<<s[0];
11. **for**(**int** i=1;i<n;i++){
12. **if**((s[i+1]==**'a'**||s[i+1]==**'e'**))
13. cout<<**"."**;
14. cout<<s[i];
15. }
16. cout<<endl;
17. }
18. }

Let's define a *permutation* of length n� as an array p� of length n�, which contains every number from 11 to n� exactly once.

You are given a permutation p1,p2,…,pn�1,�2,…,�� and a number k�. You need to sort this permutation in the ascending order. In order to do it, you can repeat the following operation any number of times (possibly, zero):

* pick two elements of the permutation pi�� and pj�� such that |i−j|=k|�−�|=�, and swap them.

Unfortunately, some permutations can't be sorted with some fixed numbers k�. For example, it's impossible to sort [2,4,3,1][2,4,3,1] with k=2�=2.

That's why, before starting the sorting, you can make at most one *preliminary exchange*:

* choose any pair pi�� and pj�� and swap them.

Your task is to:

1. check whether is it possible to sort the permutation **without** any preliminary exchanges,
2. if it's not, check, whether is it possible to sort the permutation using exactly **one** preliminary exchange.

For example, if k=2�=2 and permutation is [2,4,3,1][2,4,3,1], then you can make a preliminary exchange of p1�1 and p4�4, which will produce permutation [1,4,3,2][1,4,3,2], which is possible to sort with given k�.

Solve:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
3. **int** main(){
4. **int** t;
5. cin>>t;
7. **while**(t--){
9. **int** n,k,c=0,x;
10. cin>>n>>k;
11. **for**(**int** i=1;i<n+1;i++){
12. cin>>x;
13. c+=i%k!=x%k;
15. }
16. cout<<((c>2)?-1:c/2)<<endl;
17. }
18. }

Spring has come, and the management of the AvtoBus bus fleet has given the order to replace winter tires with summer tires on all buses.

You own a small bus service business and you have just received an order to replace n� tires. You know that the bus fleet owns two types of buses: with two axles (these buses have 44 wheels) and with three axles (these buses have 66 wheels).

You don't know how many buses of which type the AvtoBus bus fleet owns, so you wonder how many buses the fleet might have. You have to determine the minimum and the maximum number of buses that can be in the fleet if you know that the total number of wheels for all buses is n�.

Solve:

1. **#include<iostream>**
2. **using** **namespace** std;
3. **long** **long** a,t;
4. **int** main(){
5. cin>>t;
6. **while**(t--){
7. cin>>a;
8. **if**(a<4||a%2==1) puts(**"-1"**);
9. **else**{
10. cout<<(a+4)/6<<**' '**<<a/4<<endl;
11. }
12. }
13. }

Given a positive integer n�, find the maximum size of an interval [l,r][�,�] of positive integers such that, for every i� in the interval (i.e., l≤i≤r�≤�≤�), n� is a multiple of i�.

Given two integers l≤r�≤�, the size of the interval [l,r][�,�] is r−l+1�−�+1 (i.e., it coincides with the number of integers belonging to the interval).

Solve:

1. **#include**<bits/stdc++.h>
2. **using** **namespace** std;
4. **int** main(){
5. **long** **long** t,n;
6. cin>>t;
7. **while**(t--){
8. cin>>n;
9. **int** i=1;
10. **while**(n%i==0)i++;
11. cout<<i-1<<endl;
12. }
13. }

Ian and Mary are frogs living on lattice points of the Cartesian coordinate plane, with Ian living on (0,0)(0,0) and Mary living on (a,b)(�,�).

Ian would like to visit Mary by jumping around the Cartesian coordinate plane. Every second, he jumps from his current position (xp,yp)(��,��) to another lattice point (xq,yq)(��,��), such that no lattice point other than (xp,yp)(��,��) and (xq,yq)(��,��) lies on the segment between point (xp,yp)(��,��) and point (xq,yq)(��,��).

As Ian wants to meet Mary as soon as possible, he wants to jump towards point (a,b)(�,�) using **at most**22**jumps**. Unfortunately, Ian is not good at maths. Can you help him?

A lattice point is defined as a point with both the x�-coordinate and y�-coordinate being integers

Solve:

1. **#include**<bits/stdc++.h>
3. **int** t,a,b;
5. **int** main(){
6. scanf(**"%d"**,&t);
7. **while**(t--){
8. scanf(**"%d%d"**,&a,&b);
9. printf(**"2\n%d %d\n%d %d\n"**,a-1,1,a,b);
10. }
11. **return** 0;
12. }