



Problem A

He asks you to count how many $s_i = s_{i+1}$ in range $[L, R]$, we can use prefix sum to minimize time for each query.

We can use prefix in this way:

$$pref(i) = \begin{cases} arr[i] + pref(i-1), & arr[i] == arr[i-1] \\ pref(i-1), & arr[i] \neq arr[i-1] \end{cases}$$

When current index equal previous index we increment the value of prefix, otherwise we use previous value.

. . # #

0	0	0	1	1	2	3
---	---	---	---	---	---	---

For each query answer is: (How many $s_i = s_{i+1}$ in range $[0, R]$ - How many $s_i = s_{i+1}$ in range $[0, L - 1]$).

```
1  #include <iostream>
2  #include <bits/stdc++.h>
3  using namespace std;
4  int main ()
5  {
6      string s; cin>>s;  int q, l , r, oo=s.size();
7      int arr [oo-1];
8      for (int i = 0; i < oo-1; ++i) {
9          if(s[i]==s[i+1]){arr[i+1]=arr[i]+1;}
10         else
11         {
12             arr[i+1]=arr[i];
13         }
14     }
15     cin>>q;
16     while (q-->0)
17     {
18         cin>>l>>r;
19         cout<<(arr[r-1]-arr[l-1])<<endl;
20     }
21 }
```



Problem C

We can use prefix sum as **problem A**, for each number realize the condition we will increment the value of prefix:

$$\begin{aligned} &pref(i) \\ &= \begin{cases} pref(i-1) + 1, & (n \% 10 == 2) \vee (n \% 10 == 3) \vee (n \% 10 == 9) \\ pref(i-1), & otherwise \end{cases} \end{aligned}$$

```
1  #include <iostream>
2  #include <bits/stdc++.h>
3  using namespace std;
4  int n;
5  long long arr[N];
6  long long pref[N];
7  int check(int num)
8  {
9      if(num%10==2||num%10==3||num%10==9){return 1;}
10     return 0;
11 }
12 int main ()
13 {
14     cin>>n;
15     for(int i=1;i<N;i++)
16     {
17         pref[i]=pref[i-1]+ check(i);
18     }
19     while (n--)
20     {
21         int l,r;cin>>l>>r;
22         cout<<pref[r]-pref[l-1]<<endl;
23     }
24 }
```



Problem E

In this problem we need to choose x, y optimally. that when we divide $(x + y)$ **the loss is as small as possible**, so we will use smallest values in array first:

200 300 500



250

Sum	Division	loss
$(200 + 300) = 500$	250	250
$(300 + 500) = 800$	400	400
$(200 + 500) = 700$	350	350

To get first 2 smallest values every time, we will sort.

```
1  #include<iostream>
2  #include <algorithm>
3  using namespace std;
4  int main()
5  {
6      int x;
7      cin>>x;
8      float arr[x];
9      for (int i=0;i<x;i++)
10     {
11         cin>>arr[i];
12     }
13     sort (arr,arr+x);
14     for (int i=1;i<x;i++)
15     {
16         arr[i]=(arr[i]+arr[i-1])/2;
17     }
18     cout<<arr[x-1];
19     return 0;}
```

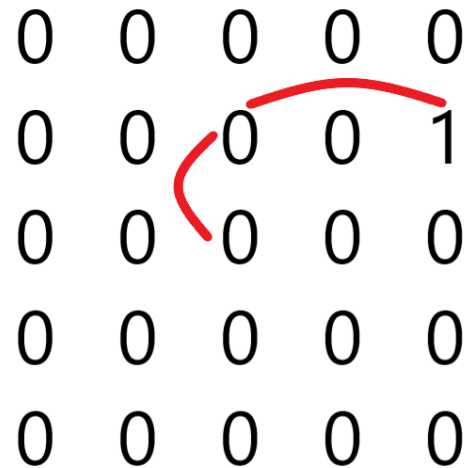


Problem F

we need to move 1 from index $[4][1]$ to index $[2][2]$, so we need to move **2 horizontal**, and **1 vertical**.

If 1 in index $[x][y]$ so answer is:

$$|x - 2| + |y - 2|$$



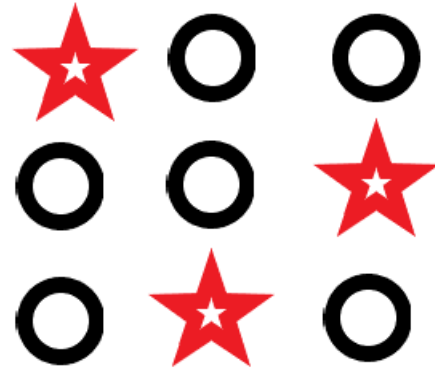
```
1  #include <iostream>
2  using namespace std;
3  int butifule_matrix(int arr [5][5])
4  {
5
6      for ( int i =0 ; i< 5 ; i++)
7      {
8          for ( int j =0 ; j<5 ; j++)
9          {
10             if (arr[i][j]==1){ return abs(i-2)+abs(j-2);}
11          }
12      }
13  }
14  int main ()
15  {
16      int arr[5][5];
17      for ( int i =0 ; i< 5 ; i++)
18      {
19          for ( int j =0 ; j<5 ; j++)
20          {
21              cin>>arr[i][j];
22          }
23      }
24      cout<<butifule_matrix(arr);
25  }
```



Problem H

In this example we have two cases:

- Two neighbors **O**: we have two options
 - Take separately so cost will be $2 * X$
 - Take both O so cost will be Y
- Only one O:
 - Take this one with cost X



So, we will take minimum when we have two neighbors.

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int tc ; cin>>tc;
5      while (tc-->0)
6      {
7          int n , m , x , y ;
8          cin>>n>>m>>x>>y;
9          char arr[n][m];
10         for (int i = 0; i < n; ++i) {
11             for (int j = 0; j < m ; ++j) {
12                 cin>>arr[i][j];
13             }
14         }
15         int ans = 0 ;
16         for (int i = 0; i < n; ++i) {
17             for (int j = 0; j < m ; ++j) {
18                 if(j+1 < m && arr[i][j]=='.' && arr[i][j+1]=='.')
19                 {
20                     ans += min(2*x , y) ;
21                     j++;
22                 }
23                 else if(arr[i][j]=='.')
24                 {
25                     ans+= x;
26                 }
27             }
28         }
29         cout<<ans<<endl;
30     }
31 }
```



Problem I

We will use prefix sum to answer each query in $O(1)$, so we need to know the index of character in alphabet.

We can do this using this formula:

$$pref[i + 1] = pref[i] + (s[i] - 'a' + 1)$$

Because $int('a') = 49$, and $int('b') = 50$

'a' - 'a'	0
'b' - 'a'	1
'c' - 'a'	2

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int n , q ;
5      cin>>n>>q ;
6      int pref[n+1];
7      string s ; cin>>s;
8      pref[0] = 0;
9      for (int i = 0; i < s.size(); ++i) {
10         pref[i+1] = pref[i] + (s[i] - 'a'+1) ;
11     }
12     while (q-->0)
13     {
14         int l , r ;
15         cin>>l>>r ;
16         cout<<pref[r] - pref[l-1]<<endl;
17     }
18 }
```



Problem J

We search for continuous K element in array their sum as small as possible.

I can use prefix sum to get sum of $\sum_i^{i-k} Arr[i]$.

1 2 6 1 1 7 1

I will minimize value $pref[i] - pref[i - k - 1]$.

```
1  #include <iostream>
2  using namespace std;
3  int val(int l , int r , int arr [])
4  {
5      if(l==0){return arr[r];}
6      else{return arr[r]-arr[l-1];}
7  }
8  int main(){
9      int n , k ,ind=0;cin>>n>>k;
10     int arr[n];
11     cin>>arr[0];
12     int minn= 1000000000;
13     for(int i = 1 ; i< n ;i++)
14     {
15         int x; cin>>x;
16         arr[i]=x+arr[i-1];
17     }
18     for(int i = 0 ; i< n-k+1 ; i++)
19     {
20         if(val(i,i+k-1,arr) <minn)
21         {
22             minn=val(i,i+k-1,arr);
23             ind=i;
24         }
25     }
26     cout<<ind+1;
27 }
```



Problem K

In this problem we need to minimize the cost of string, first let's define cost.

If we take any index I and any index L so the cost increase when:

$$S[I] = S[L] \text{ and } S[I + 1] = S[L + 1]$$



So if we have **A B** in string try to not have another **A B**.

By use this sequence:

A AB AC AD **B** BC BD C CD

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4  int n , k;
5  cin>>n>>k;
6  while (n>0) {
7      for (int j = 0; j < k and n>0; ++j)
8      {
9          cout<<(char)(j + 'a'); n--;
10         for (int l = j+1; l < k and n>0; ++l)
11         {
12             cout<<(char)(j + 'a'); n--;
13             if(n>0)
14             {
15                 cout<<(char)(l + 'a'); n--;
16             }
17         }
18     }
19 }
```




Problem L

At first:

$$a^2 - b = c$$

$$a^2 + b^2 = c^2$$

Subtract two equations:

$$b^2 + b = c^2 - c$$

$$b(b + 1) = c(c - 1)$$

B never equal C because if $b = c$, $(b + 1) \neq (b - 1)$.

So:

$$b = c - 1$$

$$c = b + 1$$

Now in first equation:

$$a^2 - b = b + 1$$

$$a^2 = 2b + 1$$

So a^2 always will be odd so a is **Odd**.

$$a \leq b \leq c \leq N$$

if $C = N$ (max value)

$$a^2 = 2c - 1$$

So a will be any **odd** value between $[3, \sqrt{2c - 1}]$.

Count odd values in this range in $O(1)$ by using $\frac{\sqrt{2n - 1} + 1}{2} - 1$.

Or iterate loop on odd values in $O(\frac{N}{2})$.