问题 A: 熊熊与比赛

熊熊 Limak 喜欢看电视上的体育节目。他今天要去看一场比赛。这场比赛持续 90 分钟，没有休息时间。

每分钟可以是有趣的，也可以是无聊的。如果连续 15 分钟是无聊的，那么 Limak 就会立即关掉电视。

现在你知道会有 n 个有趣的分钟 t1,t2,...,tn。你的任务是计算 Limak 会看多少分钟的比赛。

输入格式
输入的第一行包含一个整数 n（1≤n≤90）--有趣的分钟数。
第二行包含 n 个整数 t1,t2,...,tn（1≤t1<t2<...tn≤90），按递增顺序给出。输出格式
输出格式
打印 Limak 将观看比赛的分钟数。

输入样例 1：
3
7 20 88
输出样例 1：
35
样例 1 解释
在第一个样例中，第 21,22,...,35 分钟都很无聊，因此 Limak 会在第 35 分钟后立即关闭电视。因此，他将看 35 分钟的比赛。

输入样例 2：
9
16 20 30 40 50 60 70 80 90
输出样例 2：
15
样例 2 解释
在第二个样例中，前 15 分钟是无聊的。

输入样例 3：
9
15 20 30 40 50 60 70 80 90
输出样例 3：
90
样例 3 解释
在第三个样例中，没有连续的 15 分钟是无聊的。所以，Limak 会看完整场比赛。

题解：

一道模拟题，给出感兴趣的时刻，若不感兴趣的时间段>=15,则关电视

注意：最后一个时刻后还可以再看 15 分钟，还有边界 90。

代码如下：

```
#include<bits/stdc++.h>
```

```
using namespace std;
  int n,a[10000];
int main()
{
       cin>>n;
       for(int i=1;i<=n;i++)
              cin>>a[i];
       if(a[1]>15)
       {
          printf("15\n");
          return 0;
       }

       for(int i=2;i<=n;i++)
       {
              if(a[i]-a[i-1]>15){//中间差 15 个，所以要>
                     cout<<a[i-1]+15;
                     return 0;
              }
       }
       cout<<"90";
       return 0;
 }
```

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
     int n,a[100];
     while(~scanf("%d",&n))
     {
           for(int i=0;i<n;i++)
                 scanf("%d",&a[i]);
           if(a[0]>15)
           {
                 printf("15\n");
                 continue;
           }
           else
           {
                 int maxn=a[0];
                 for(int i=1;i<n;i++)
                 {
                       if((a[i]-a[i-1]>15))
                       {
                             break;
                       }
                       else
                       {
```

```
                    maxn=a[i];
                }
            }
            maxn+=15;//注意咯.....
            if(maxn>=90)
                maxn=90;
            printf("%d\n",maxn);
        }
    }
    return 0;
}
```

问题 B: 二进制数

zz 非常喜欢数学。这就是为什么当他感到无聊时，他会玩一些数字来执行一些操作。zz 得到一个二进制数 x，并希望将这个数字变成 1。当 x 不等于 1 时，zz 重复以下操作：如果 x 是奇数，则他将 x 加 1 ，否则他将 x 除以 2。zz 知道，对于任何正整数，该过程都会在有限的时间内结束。zz 应该执行多少个次才能将 x 变成 1？

输入格式
第一行包含二进制系统中的正整数 x。保证 x 的第一个数字与零不同，其位数不超过 10^6。
输出格式
打印所需数量的操作。

输入样例 1：
1
输出样例 1：
0

输入样例 2：
1001001
输出样例 2：
12

输入样例 3：
101110
输出样例 3：
8

样例解释：
让我们考虑第三个样本。数字 101110 是偶数，这意味着我们应该将其除以 2。除法后，方吉得到一个奇数 10111，并加一。数字 11000 可以连续三次除以 2，得到数字 11。剩下的就是将数字增加 1（我们得到 100），然后连续两次将其除以 2。结果，我们得到 1。

题解：

对于一个二进制数，+1 是若最后一个数是 1 ，则变为 0 ，若为 0 ，则变为 1 ，直到遇到第一个 0 。而 /2 是将最后一个 0 去除。
判断一个二进制数奇偶性，若最后一位为 1 ，则为奇数，反之为偶数。
所以可以从最后一位往前枚举，发现 0 可以直接 /2 而发现 1 要先 +1 再 /2 。
可以很容易的发现，遇到第一个 1 后，后面就不存在纯粹的遇见 0 然后 /2 ，所以这是个临界点。
并且在倒数第二位时如果加上了 1 且倒数第二位是 1 ，那么会凭空多出一位，所以要特殊判断一次。

```
#include<bits/stdc++.h>
using namespace std;
string a;
int main(){
    int n,t=0,sum=0,n1=1,i;
    getline(cin,a);
    n=a.size();
    for(i=n-1;i>=1;i--){
        if(a[i]=='1'){
            a[i-1]=a[i-1]+1;
            t=t+2;//+1,/2 两次操作
        }
        else if(a[i]=='2'){
            t=t+1; //  /2
            a[i-1]=a[i-1]+1;//进一位
        }
        else if(a[i]=='0')t++;//  /2
    }
    if(a[0]=='2')t++;//特判
    cout<<t;
}
```

问题 C: zz 的游戏

题目描述
zz 和 kk 正在玩一款游戏, 初始，zz 可以设定一个长度为 n 字符串 AB 串（仅有 A 和 B），kk 可以选择某个前缀或后缀进行一次翻转（指原本的 A 变为 B，B 变为 A），当然也可以不选。最后每个玩家所获得的力量：
若字符串中第 i 个位置为 A，则给予 zz Pi 点能量；若为 B，则给予 kk Pi 点能量。求 kk 进行翻转操作后所能得到的最大能量。
输入格式
第一行包含整数 n （1≤n≤5·10^5） –那个字符。
第二行包含 n 个整数 pi （1≤pi≤10^9） – 第 i 字符的强度。
第三行包含 n 个字符 A 或 B 。
输出格式
打印唯一的整数 a – kk 可以达到的最大强度。
Examples
Input
5
1 2 3 4 5
ABABA
Output
11
Input
5
1 2 3 4 5
AAAAA
Output
15
Input
1
1
B
Output
1
Note
在第一个示例中，kk 应该翻转长度 1 的后缀。
在第二个示例中，kk 应该翻转长度为 5 的前缀或后缀（此处相同）。
在第三个示例中，kk 应该什么都不做。

先计算原本数组（未改动）可以获得的能量之和。
然后用两个不同的变量储存。
接着分别枚举前缀和后缀，
最后计算改动后的最大值并输出。

```
#include<iostream>
using namespace std;
long long a[500005],b[500005],x,y,n,ans;
char s;
```

```
int main(){
    cin>>n;//输入 n
    for(int i=0;i<n;i++)cin>>a[i];//输入 a[i]
    for(int i=0;i<n;i++){
        cin>>s;//一个一个读取字符
        if(s=='B')b[i]=1;
        else b[i]=0;//判断是否为'B' =1 表示 B
        ans+=b[i]*a[i];//读取总和
    }
    x=y=ans;//赋值
    for(int i=0;i<n;i++){//枚举前缀
        if(b[i]==1)x-=a[i];// 如果 b[i]=1，说明当前字符为 B，翻转后为 A,让 x 去减这一位
的权值
        else x+=a[i];//如果 b[i]=0，说明当前字符为 B，让 x 去加这一位的权值
        if(x>ans)ans=x;//轮换，修改最大值
    }
    for(int i=n-1;i>0;i--){//枚举后缀
        if(b[i]==1)y-=a[i];//如果 b[i]=1，说明当前字符为 B，让 y 去减这一位的权值
        else y+=a[i];//如果 b[i]=0，说明当前字符为 B，让 y 去加这一位的权值
        if(y>ans)ans=y;//轮换，修改最大值
    }
    cout<<ans<<endl;//输出进行翻转操作后所能得到的最大能量。
    return 0;
}
```

问题 D: 图像压缩

给你一个图像，它可以用二维 n×m 像素网格表示。图像的每个像素都分别由字符“0”或“1”表示。您希望压缩此图像。您需要选择一个大于 1 的整数 k（k>1），并将图像分割为 k×k 块。如果 n 和 m 不能被 k 整除，则图像的右侧和底部仅填充 0，以便它们可被 k 整除。每个块中的每个像素必须具有相同的值。

您可能需要切换一些像素的状态，使得每个块内所有像素状态一致—这样才是可压缩的。您可以任选 k，请您找到您需要切换状态的最小像素数。输出这个最小值。

更具体地说，步骤是首先选择 k，然后用零填充图像，然后，我们可以切换像素，使其可压缩。使图像在该状态下必须可压缩。

输入格式

第一行输入将包含两个整数 n，m（2≤n，m≤2500），即图像的尺寸。

接下来的 n 行输入将包含一个正好有 m 个字符的二进制字符串，代表图像。

输出格式

打印单个整数，即切换以使图像可压缩所需的最小像素数。

Example

Input

3 5

00100

10110

11001

Output

5

我们首先选择 k=2。

图像填充如下：

001000

101100

110010

000000

我们可以将图像切换为如下所示：

001100

001100

000000

000000

此时可以看到，对于 k＝2，这个图像是可压缩的。

输入样例 复制

3 5

00100

10110

11001

输出样例 复制

5

题目要求您需要选择整数 k，将图像分割为 k×k 块，每个块中的每个像素必须具有相同的值，求最小需要更改多少个点的值。

基本思路是直接枚举，枚举每一个 k 需要修改多少个像素点，即求出每个 k*k 的区域的 0/1 的个数，然后取最小值。直接枚举时间复杂度很高，可以用二维前缀和优化。首先求出二维前缀和，然后枚举 k，时间复杂度为 O(m*n*k)

```
#include<bits/stdc++.h>
using namespace std;
const int N=5010;//范围扩大两倍，因为可能再添加 k*k，防止添加后越界
int f[N][N];
//char    a[N][N];
string a[N];
int b[N][N];
int n,m;
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        cin>>a[i];//可能测试数据空格比较多，单个字符输入有问题
        for(int j=0;j<m;j++){
            b[i][j+1]=a[i][j]-'0';
        }
    }
    int ans=0;
    for(int i=1;i<N;i++)//这里枚举到 N，大于 m/n 部分
        for(int j=1;j<N;j++)//的相当于对扩充区域的初始化
            f[i][j]=f[i][j-1]+f[i-1][j]-f[i-1][j-1]+b[i][j];//前缀和
    ans=f[n][m];//初始值为所有 1 的个数


    for(int k=2;k<=max(n,m);k++){//枚举 k
        int tmp=0;
        for(int i=k;i<=n+k;i+=k){//从右下角开始枚举所有 k*k 的区域中的 1 的个
数
            for(int j=k;j<=m+k;j+=k){
                int cnt=f[i][j]-f[i-k][j]-f[i][j-k]+f[i-k][j-k];
                tmp+=min(cnt,k*k-cnt);//累积每个 k*k 需要修改的次数
            }
        }
        ans=min(ans,tmp);//求出每个 k 的最小修改值
    }
    printf("%d",ans);
```

```
        return 0;
}
```