一种双核 CPU 的两个核能够同时的处理任务，现在有 n 个已知数据量的任务需要交给 CPU 处理，假设已知 CPU 的每个核 1 秒可以处理 1kb，每个核同时只能处理一项任务。n 个任务可以按照任意顺序放入 CPU 进行处理，现在需要设计一个方案让 CPU 处理完这批任务所需的时间最少，求这个最小的时间。

```
importjava.util.Scanner;
publicclassMain{
publicstaticvoidmain(String[]args){
Scannersc=newScanner(System.in);
while(sc.hasNext()){
intn=sc.nextInt();
int[]arr=newint[n];
intsum=0;
for(inti=0;i<arr.length;i++){
arr[i]=sc.nextInt()>>10;
sum+=arr[i];
}
//dp[j]表示在容量为 j 的情况下可存放的重量
//如果不放 arr[i]重量为 dp[j],如果放 arr[i]重量为 dp[j-arr[i]]+arr[i];
int[]dp=newint[sum/2+1];
for(inti=0;i<n;i++){
for(intj=sum/2;j>=arr[i];j--){
dp[j]=Math.max(dp[j],dp[j-arr[i]]+arr[i]);
}
}
System.out.println(Math.max(dp[sum/2],sum-dp[sum/2])<<10);
}
}
}
```

终于到周末啦！小易走在市区的街道上准备找朋友聚会，突然服务器发来警报,小易需要立即回公司修复这个紧急 bug。假设市区是一个无限大的区域，每条街道假设坐标是(X，Y)，小易当前在(0，0)街道，办公室在(gx,gy)街道上。小易周围有多个出租车打车点，小易赶去办公室有两种选择，一种就是走路去公司，另外一种就是走到一个出租车打车点，然后从打车点的位置坐出租车去公司。每次移动到相邻的街道(横向或者纵向)走路将会花费 walkTime 时间，打车将花费 taxiTime 时间。小易需要尽快赶到公司去，现在小易想知道他最快需要花费多少时间去公司。

```
importjava.util.*;
publicclassMain{
publicstaticvoidmain(Stringargs[]){
Scannersc=newScanner(System.in);
intn=sc.nextInt();
inttx[]=newint[n];
```

```
intty[]=newint[n];
for(inti=0;i<n;i++){
tx[i]=sc.nextInt();//打车点 x 坐标
}
for(inti=0;i<n;i++){
ty[i]=sc.nextInt();//打车点 y 坐标
}
intgx=sc.nextInt();//公司 x 坐标
intgy=sc.nextInt();//公司 y 坐标

intwt=sc.nextInt();//走路速度
intdt=sc.nextInt();//打车速度

intwalkTime=(Math.abs(gx)+Math.abs(gy))*wt;//如果全部走路
intdriveTime=Integer.MAX_VALUE;
for(inti=0;i<n;i++){//如果打车
driveTime=Math.min(driveTime,
(Math.abs(ty[i])+Math.abs(tx[i]))*wt+(Math.abs(ty[i]-gy)+Math.abs(tx[i]-gx))*dt);
}
System.out.println(Math.min(driveTime,walkTime));
}
}
```

在幼儿园有 n 个小朋友排列为一个队伍，从左到右一个挨着一个编号为(0~n-1)。其中有一些是男生，有一些是女生，男生用'B'表示，女生用'G'表示。小朋友们都很顽皮，当一个男生挨着的是女生的时候就会发生矛盾。作为幼儿园的老师，你需要让男生挨着女生或者女生挨着男生的情况最少。你只能在原队形上进行调整，每次调整只能让相邻的两个小朋友交换位置，现在需要尽快完成队伍调整，你需要计算出最少需要调整多少次可以让上述情况最少。
例如：
GGBBG->GGBGB->GGGBB
这样就使之前的两处男女相邻变为一处相邻，需要调整队形 2 次

```
importjava.util.Scanner;



publicclassshs{



publicstaticvoidmain(String[]args){

//TODOAuto-generatedmethodstub
```

```
Scannerscan=newScanner(System.in);

Strings=scan.nextLine();

intb=0;

intg=0;

intbsum=0;

intgsum=0;

for(inti=0;i<s.length();i++){

if(s.charAt(i)=='G'){

gsum+=(i-g);

g++;

}elseif(s.charAt(i)=='B'){

bsum+=(i-b);

b++;

}

}



System.out.println(Math.min(bsum,gsum));

}



}
```

小易有一个长度为 n 序列，小易想移除掉里面的重复元素，但是小易想是对于每种元素保留最后出现的那个。小易遇到了困难,希望你来帮助他。

```
importjava.util.*;
```

```
/**
*此题巧妙地用到了 List 去重的方法。
*比如说有一组数，为 155161
*从[0,n-1]使用 List 去重之后，保存地其实就是每种元素最开始出现的那个元素。
*但是我们换个角度去想，如果我们从后向前遍历，保存地就是每种元素最后出现的那个元素，只不过相对从前到后来说，数倒过来了。
*我们最终将数反转过来即可。
*时间复杂度为 O(n)
*@author 何嘉龙
*
*/
publicclassMain{
publicstaticvoidmain(String[]args){
Scannerin=newScanner(System.in);
while(in.hasNext()){
intn=in.nextInt();
int[]arr=newint[n];
List<Integer>lists=newArrayList<>();
for(inti=0;i<n;i++){
arr[i]=in.nextInt();
}
for(inti=n-1;i>=0;--i){
if(!lists.contains(arr[i])){
lists.add(arr[i]);
}
}
for(inti=lists.size()-1;i>=0;--i){
System.out.print(lists.get(i));
if(i!=0){
System.out.print("");
}
}
}
in.close();
}
}
```

小易拥有一个拥有魔力的手环上面有 n 个数字(构成一个环),当这个魔力手环每次使用魔力的时候就会发生一种奇特的变化：每个数字会变成自己跟后面一个数字的和(最后一个数字的后面一个数字是第一个),一旦某个位置的数字大于等于 100 就马上对 100 取模(比如某个位置变为 103,就会自动变为 3).现在给出这个魔力手环的构成，请你计算出使用 k 次魔力之后魔力手环的状态。

```
importjava.util.Scanner;
```

```
//请见 http://blog.csdn.net/zheng548/article/details/71075163

publicclassMain{
/**
*利用矩阵快速幂
参考：http://www.cnblogs.com/vongang/archive/2012/04/01/2429015.html
http://www.lai18.com/content/1108003.html?from=cancel
*/
publicstaticvoidmain(String[]args){
Scannersc=newScanner(System.in);
intn=sc.nextInt();
intk=sc.nextInt();
//用一个二维数组存储
int[][]arr=newint[1][n];
for(inti=0;i<n;i++){
arr[0][i]=sc.nextInt();
}
//初始化单元矩阵
int[][]mul=newint[n][n];
for(inti=0;i<n;i++){
if(i<n-1){
mul[i][i]=1;
mul[i+1][i]=1;
}else{
mul[i][i]=1;
mul[0][i]=1;
}
}

for(;k>0;k>>=1){
if((k&1)==1){
arr=Core(arr,mul);
}
mul=Core(mul,mul);
}
inti;
for(i=0;i<n-1;i++){
System.out.print(arr[0][i]+"");
}
System.out.println(arr[0][i]);
}

privatestaticint[][]Core(int[][]a,int[][]b){
introwRes=a.length;
```

```
intcolumnRes=b[0].length;//TODO:
intcolumnA=a[0].length;//==b.length;

int[][]c=newint[rowRes][columnRes];
for(inti=0;i<rowRes;i++){
for(intj=0;j<columnRes;j++){

for(intk=0;k<columnA;k++){
if(a[i][k]==0||b[k][j]==0){
continue;//剪枝
}

c[i][j]+=a[i][k]*b[k][j];
}
//100 取余运算
if(c[i][j]>=100){
c[i][j]%=100;
}
}
}
returnc;
}

}
```

现在有 n 位工程师和 6 项工作(编号为 0 至 5)，现在给出每个人能够胜任的工作序号表(用一个字符串表示，比如：045，表示某位工程师能够胜任 0 号，4 号，5 号工作)。现在需要进行工作安排，每位工程师只能被安排到自己能够胜任的工作当中去，两位工程师不能安排到同一项工作当中去。如果两种工作安排中有一个人被安排在的工作序号不一样就被视为不同的工作安排，现在需要计算出有多少种不同工作安排计划。

//利用回溯法求解.

//题意有两个地方没说清楚：1、一个人只能做一项工程，而不能分饰两角；

//2、有几个工程师，每个工程师有一个工作即满足题意，不用 6 项工作全部都要有人做。

//前一个人选了哪项工作，直接影响后一个人的选择余地。

//所以需要用一个 set 记录在当前这个人选择之前，前面那些人已经选了的工作，进而他只能选除 set 中已有剩下的工作。

//当考察完最后一个人，情况数+1（递归出口），证明是满足题意的方案。下面是代码

```
importjava.util.HashSet;
importjava.util.Scanner;

//网易：工程师与项目的匹配

publicclassWangyi_WorkAndWorker{

privatestaticintcases=0;

publicstaticvoidmain(String[]args)
{
//读取输入
Scannerin=newScanner(System.in);
intn=in.nextInt();
String[]ables=newString[n];
for(inti=0;i<n;i++)
ables[i]=in.next();

//计算
backtracing(ables,0,newHashSet<Integer>());

System.out.println(cases);

in.close();
}


publicstaticvoidbacktracing(String[]ables,intindex,HashSet<Integer>set){

if(index>=ables.length){
cases++;
return;
}

Stringable=ables[index];
for(inti=0;i<able.length();i++){

intwork=able.charAt(i)-'0';
if(!set.contains(work)){
set.add(work);
backtracing(ables,index+1,set);
set.remove(work);
```

```
}
}
}
}
```

小易最近在数学课上学习到了集合的概念,集合有三个特征：1.确定性 2.互异性 3.无序性.
小易的老师给了小易这样一个集合：
S={p/q|w≤p≤x,y≤q≤z}
需要根据给定的 w，x，y，z,求出集合中一共有多少个元素。小易才学习了集合还解决不了这个复杂的问题,需要你来帮助他。

```
#include<stdio.h>
#defineN10000
intfind(floatarr[],intn,floatx);
intmain(){
intw,x,y,z,i,j,len=0;
floattemp,arr[N];
scanf("%d%d%d%d",&w,&x,&y,&z);
for(i=w;i<=x;i++){
for(j=y;j<=z;j++){
temp=(float)i/j;
if(!find(arr,len,temp)){
arr[len++]=temp;
}
}
}
printf("%d",len);
return0;
}
/*判断元素是否在数组中，存在则返回 1，否则为 0*/
intfind(floatarr[],intn,floatx){
inti;
for(i=0;i<n;i++){
if(arr[i]==x)return1;
}
return0;
}
```

常规的表达式求值，我们都会根据计算的优先级来计算。比如*/的优先级就高于+-。但是小易所生活的世界的表达式规则很简单，从左往右依次计算即可，而且小易所在的世界没有除法，意味着表达式中没有/，只有(+,-和*)。现在给出一个表达式，需要你帮忙计算出小易所在的世界这个表达式的值为多少

```
#include<stdio.h>
```

```
#include<string.h>
#defineN50
intmain(){
charstr[N];
gets(str);
intresult=str[0]-'0';//得到第 0 个字符，并减去字符'0'使其转换成数字
inti,temp;
for(i=1;i<strlen(str)-1;i+=2){//从第二个字符开始算
temp=str[i+1]-'0';//要操作的数
if(str[i]=='+'){
result+=temp;//结果
}elseif(str[i]=='-'){
result-=temp;
}elseif(str[i]=='*'){
result*=temp;
}
}
printf("%d\n",result);
return0;
}
```

小易有一块 n*n 的棋盘，棋盘的每一个格子都为黑色或者白色，小易现在要用他喜欢的红色去涂画棋盘。小易会找出棋盘中某一列中拥有相同颜色的最大的区域去涂画，帮助小易算算他会涂画多少个棋格。

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>

intmain()
{
usingnamespacestd;
intn;
while(cin>>n){
vector<vector<char>>matrix(n,vector<char>(n));
for(inti=0;i<n;i++){
for(intj=0;j<n;j++){
cin>>matrix[i][j];
}
}
intmax=0;
for(inti=0;i<n;i++){
intret=1;
```

```
for(intj=0;j<n-1;j++){
if(matrix[j][i]==matrix[j+1][i]){
ret++;
if(ret>max){
max=ret;
}
}
else{
ret=1;
}
}
}
cout<<max<<endl;
}
return0;
}
```

小易参与了一个记单词的小游戏。游戏开始系统提供了 m 个不同的单词，小易记忆一段时间之后需要在纸上写出他记住的单词。小易一共写出了 n 个他能记住的单词，如果小易写出的单词是在系统提供的，将获得这个单词长度的平方的分数。注意小易写出的单词可能重复，但是对于每个正确的单词只能计分一次。

```
packageenglish;

importjava.util.*;
publicclassMain{
publicstaticvoidmain(String[]args){
Scannerin=newScanner(System.in);
while(in.hasNext()){
intn=in.nextInt();
intm=in.nextInt();
List<String>systemWords=newArrayList<>();
Set<String>writeWordsSet=newHashSet<>();
List<String>writeWordsList=newArrayList<String>();
for(inti=0;i<n;i++){
systemWords.add(in.next());
}
for(inti=0;i<m;i++){
Stringstr=in.next();
if(systemWords.contains(str)){
writeWordsSet.add(str);
}
}
writeWordsList.addAll(writeWordsSet);
```

```
intsum=0;
for(inti=0;i<writeWordsList.size();i++){
intlen=writeWordsList.get(i).length();
sum+=Math.pow(len,2);
}
System.out.println(sum);
}
in.close();
}
}
```

小易有 n 块砖块，每一块砖块有一个高度。小易希望利用这些砖块堆砌两座相同高度的塔。为了让问题简单，砖块堆砌就是简单的高度相加，某一块砖只能使用在一座塔中一次。小易现在让能够堆砌出来的两座塔的高度尽量高，小易能否完成呢。

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n = 0;
    cin >> n;
    vector<int> ts(n);
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        cin >> ts[i];
        sum += ts[i];
    }
    vector<vector<int>> dp(2, vector<int>(2*sum+1, -1));
    dp[0][sum] = 0;
    vector<int>* dp0 = &(dp[0]);
    vector<int>* dp1 = &(dp[1]);
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= 2*sum; ++j) {
            int h = ts[i-1];
            (*dp1)[j] = (*dp0)[j];
            if (j - h >= 0 && (*dp0)[j-h] != -1) {
                (*dp1)[j] = max((*dp1)[j], (*dp0)[j-h]);
            }
            if (j + h <= 2*sum && (*dp0)[j+h] != -1) {
                (*dp1)[j] = max((*dp1)[j], (*dp0)[j+h] + h);
            }
        }
        auto tmp = dp0;
        dp0 = dp1;
        dp1 = tmp;
```

```
        }
        if ((*dp0)[sum] == 0)
            cout << -1 << endl;
        else
            cout << (*dp0)[sum] << endl;
        return 0;
}
/*假设砖块分为 A，B 两堆，dp[i][j]中的 j 表示 B-A 的长度。
因为 B-A 有可能是负的，所以 j 整体偏移 sum 个长度，即 2*sum+1。
而 dp[i][j]的值则表示当 A-B 的值为 j 时，B 的最大长度是多少。
dp[i][j] = dp[i-1][j]表示我不用第 i 块砖
dp[i][j] = max(dp[i-1][j-h], dp[i-1][j])表示我把砖放在 A 堆。
dp[i][j] = max(dp[i-1][j+h]+h, dp[i-1][j])表示我把砖放在 B 堆。
然后会爆内存，所以用了滚动数组。*/
```

易老师购买了一盒饼干，盒子中一共有 k 块饼干，但是数字 k 有些数位变得模糊了，看不清楚数字具体是多少了。易老师需要你帮忙把这 k 块饼干平分给 n 个小朋友，易老师保证这盒饼干能平分给 n 个小朋友。现在你需要计算出 k 有多少种可能的数值

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        int n = sc.nextInt();
        long[][] dp = new long[str.length()+1][]; //不用 long 的话通过率只能为 90%
        for(int i = 0;i<=str.length();i++){
            dp[i] = new long[n];
        }
        dp[0][0] = 1;
        for(int i = 1;i<=str.length();i++){
            char c = str.charAt(i-1);
            for(int j = 0;j<n;j++){
                for(int k = 0;k<10;k++){
                    if(c=='X'||c=='0'+k){
                        dp[i][(j*10+k)%n]+=dp[i-1][j];
                    }
                }
            }
        }
        System.out.println(dp[str.length()][0]);
    }
}
//以上发现第 i 行的值只会访问到第 i-1 行的值，所以实际只需要两个数组即可，优化空间之
```

后的代码：

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        int n = sc.nextInt();
        long[] dp = new long[n];
        dp[0] = 1;
        for(int i = 1;i<=str.length();i++){
            char c = str.charAt(i-1);
            long[] tmp = new long[n];
            for(int j = 0;j<n;j++){
                for(int k = 0;k<10;k++){
                    if(c=='X'||c=='0'+k){
                        tmp[(j*10+k)%n]+=dp[j];
                    }
                }
            }
            dp = tmp;
        }
        System.out.println(dp[0]);
    }
}
```