例 1：Name That Number 命名那个数字

在威斯康辛州牛大农场经营者之中，都习惯于请会计部门用连续数字给母牛打上烙印。但是,母牛用手机时并没感到这个系统的便利,它们更喜欢用它们喜欢的名字来呼叫它们的同伴，而不是用像这个的语句"C'mon, #4734, get along."。请写一个程序来帮助可怜的牧牛工将一只母牛的烙印编号翻译成一个可能的名字。因为母牛们现在都有手机了，使用那标准的按键的排布来把收到从数目翻译到文字:(除了 "Q" 和 "Z")

```
2: A,B,C     5: J,K,L    8: T,U,V
 3: D,E,F     6: M,N,O    9: W,X,Y
 4: G,H,I     7: P,R,S
```

可接受的名字都被放在这样一个叫作"dict.txt" 的文件中，它包含一连串的少于 5,000 个可被接受的牛的名字。 (所有的名字都是大写的且已按字典序排列) 请读入母牛的编号并返回那些能从编号翻译出来并且在字典中的名字。举例来说,编号 4734 能产生的下列各项名字: GPDG GPDH GPDI GPEG GPEH GPEI GPFG GPFH GPFI GRDG GRDH GRDI GREG GREH GREI GRFG GRFH GRFI GSDG GSDH GSDI GSEG GSEH GSEI GSFG GSFH GSFI HPDG HPDH HPDI HPEG HPEH HPEI HPFG HPFH HPFI HRDG HRDH HRDI HREG HREH HREI HRFG HRFH HRFI HSDG HSDH HSDI HSEG HSEH HSEI HSFG HSFH HSFI IPDG IPDH IPDI IPEG IPEH IPEI IPFG IPFH IPFI IRDG IRDH IRDI IREG IREH IREI IRFG IRFH IRFI ISDG ISDH ISDI ISEG ISEH ISEI ISFG ISFH ISFI

碰巧，81 个中只有一个"GREG"是有效的(在字典中)。

写一个程序来对给出的编号打印出所有的有效名字，如果没有则输出 NONE。编号可能有 12 位数字。

INPUT FORMAT

单独的一行包含一个编号(长度可能从 1 到 12)。

OUTPUT FORMAT

以字典顺序输出一个有效名字的不重复列表，一行一个名字。

SAMPLE INPUT

```
4734
```

SAMPLE OUTPUT

```
GREG
```

分析：

思路 1：一个数字对应 3 个字母，如果我们先枚举出数字所代表的所有字符串，就有 3^12 种，然后再在 5000 的字典里寻找，可以用二分查找，数据规模是 3^12*log5000=6.5e6，空间规模是 5000。 其实可以做的更好！

思路 2：一个字母只对应一个数字，从字典中读入一个单词，把它转化成唯一对应的数字，看它是否与给出的数字匹配，时间规模是 5000*12=6e4，空间规模是常数，而且编程复杂度较低。

参考代码：

```
#include <stdio.h>
int num[26]={2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,0,7,7,8,8,8,9,9,9,0};
int name[13];
char les[13];
int main()
{       int i=-1,j=0;
        while(scanf("%c",&name[++i])!=-1 && name[i]!='\n')
                name[i]-='0';
        if(name[i]=='\n')name[i]=0;
        while(scanf("%s",les)!=-1)
        {       i=-1;
                while(les[++i]!=0)
                        if(num[les[i]-'A']!=name[i])  break;
                if(les[i]==0 && name[i]==0)
                {   printf("%s\n",les);  j=1;  }
        }
        if(j==0)  printf("NONE\n");
        return 0;
}
```

例 2：Palindromic Squares 回文平方数

回文数是指从左向右念和从右向左念都一样的数。如 12321 就是一个典型的回文数。

给定一个进制 B(2<=B<=20,由十进制表示)，输出所有的大于等于 1 小于等于 300（十进制下）且它的平方用 B 进制表示时是回文数的数。用’A’,’B’……表示 10，11 等等。

INPUT FORMAT

共一行，一个单独的整数 B(B 用十进制表示)。

OUTPUT FORMAT

每行两个 B 进制的符合要求的数字，第二个数是第一个数的平方，且第二个数是回文数。

SAMPLE INPUT

```
10
```

SAMPLE OUTPUT

```
1 1
2 4
3 9
11 121
22 484
26 676
101 10201
111 12321
121 14641
202 40804
212 44944
264 69696
```

分析：枚举+进制转换。穷举 1——300 的所有平方数，转进制，回文数的判断，OK。

参考代码：

```
#include<stdio.h>
char a[20]={'0','1','2','3','4','5','6','7','8','9',
         'A','B','C','D','E','F','G','H','I','J'};

int main()
{       int b,i,j=0,k,pan,z,c[20]={0},d[20]={0};
        long p;
       scanf("%d",&b);
       for(i=1;i<300;i++)
       {       memset(c,0,sizeof(c));memset(d,0,sizeof(d));
               pan=1;j=0;  p=i*i;
               while(p!=0)
               {   c[j]=p%b;   p=p/b;j++;  }
               for(k=j/2;;k>=1;k--)
                     if(c[k-1]!=c[j-k]) pan=0;
               if(pan==0) continue;
               k=0;  z=i;
               while(z!=0)
               {   d[k]=z%b;   z=z/b;k++;  }
               if(pan==1)
               {       j--;k--;
                       for(;k>=0;k--)
                             printf("%c",a[d[k]]);
                       printf(" ");
                       for(;j>=0;j--)
                            printf("%c",a[c[j]]);
                       printf("\n");
               }
       }
       return 0;
}
```

例 3：Dual Palindromes 双重回文数

如果一个数从左往右读和从右往左读都是一样，那么这个数就叫做“回文数”。例如，12321 就是一个回文数，而 77778 就不是。当然，回文数的首和尾都应是非零的，因此 0220 就不是回文数。

事实上，有一些数（如 21），在十进制时不是回文数，但在其它进制（如二进制时为 10101）时就是回文数。

编一个程序，从文件读入两个十进制数 N (1 <= N <= 15)S (0 < S < maxlongint)然后找出前 N 个满足大于 S 且在两种或两种以上进制（二进制至十进制）上是回文数的十进制数，输出到文件上。

本问题的解决方案不需要使用大于 32 位的整型变量。

INPUT FORMAT

只有一行，用空格隔开的两个数 N 和 S。

OUTPUT FORMAT

N 行，每行一个满足上述要求的数，并按从小到大的顺序输出。

SAMPLE INPUT

```
3 25
```

SAMPLE OUTPUT

```
26
27
28
```

分析：枚举。因为数据很小，所以只需要从 s 开始枚举每个十进制数然后判断就行了。

参考代码：

```
#include<stdio.h>
int main()
{       int b,i=0,j=0,k,pan,n;
        int c[30]={0};
        long s,a;
        scanf(cin,"%d %ld",&n,&s);
        while(i<n)
        {       s++;    a=s;
                for(b=2;b<=10;b++)
                {       j=0;   pan=1;
                        memset(c,0,sizeof(c));
                        while(s!=0)
                        {    c[j]=s%b;    s=s/b;  j++;   }
                        for  (  k=j/2;k>=1;k--)
                              if(c[k-1]!=c[j-k]) pan=0;
                        if(pan==0)
                             {  s=a;  continue;  }
                        else
                            {  printf("%ld\n",a);      i++;  s=a;break;   }
                 }
        }
        return 0;
}
```

例 4：Mixing Milk 混合牛奶

牛奶包装是一个如此低利润的生意,以至于尽可能低地控制初级产品(牛奶)的价格变得十分重要。请帮助 Merry 的牛奶制造公司(Merry Milk Makers')以尽可能最廉价的方式取得他们所需的牛奶。Merry 的牛奶制造公司从一些农民那购买牛奶，每个农民卖给牛奶制造公司的价格不一定相同。而且,如一只母牛一天只能生产一定量的牛奶,农民每一天只有一定量的牛奶可以卖。每天,Merry 的牛奶制造公司从每个农民那购买一定量的牛奶,少于或等于农民所能提供的最大值。给出 Merry 牛奶制造公司的每日的牛奶需求,连同每个农民的可提供的牛奶量和每加仑的价格,请计算 Merry 的牛奶制造公司所要付出钱的最小值。

注意:每天农民生产的牛奶的总数对 Merry 的牛奶制造公司来说是足够的。

INPUT FORMAT

第 1 行:二个整数, N 和 M。

第一个数值,N,(0<= N<=2,000,000)是 Marry 的牛奶制造公司的一天需要牛奶的数量。

第二个数值,M,(0<= M<=5,000)是提供牛奶的农民个数。

第 2 到 M+1 行:每行二个整数,P_i 和 A_i。

P_i(0<= P_i<=1,000) 是农民 i 的牛奶的价格。

A_i(0 <= A_i <= 2,000,000)是农民 i 一天能卖给 Marry 的牛奶制造公司的牛奶数量。

OUTPUT FORMAT

单独的一行包含单独的一个整数，表示 Marry 的牛奶制造公司拿到所需的牛奶所要的最小费用。

SAMPLE INPUT

```
100 5
5 20
9 40
3 10
8 80
6 30
```

SAMPLE OUTPUT

```
630
```

分析：思路 1：简单的贪心算法。将所有的牛奶按价格升序排列（用快排），然后从低到高买入，直到买够 m 为止。

思路 2：最佳解题方法。因为价格的范围在 1..1000 之内，所以，我们只要在读入数据的时候把相同价格的合并即可，进行计算时，也只需要 for i:=0 to 1000 do 进行扫描如果有价格为 i 的牛奶就收购即可，所以不需要排序。

参考代码：

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

#decine MAXFARMER 5000
typedef struct Farmer Farmer;
struct Farmer {int p;  int a;    };

int farmcmp(const void *va, const void *vb)
{       return ((Farmer*)va)->p - ((Farmer*)vb)->p;}

int nfarmer;
Farmer farmer[MAXFARMER];

int  main( )
{       int i, n, a, p;
        scanf("%d %d", &n, &nfarmer);
        for(i=0; i<nfarmer; i++)
              scanf("%d %d", &farmer[i].p, &farmer[i].a);
        qsort(farmer, nfarmer, sizeof(farmer[0]), farmcmp);
        p = 0;
        for(i=0; i<nfarmer && n > 0; i++)
        {       a = farmer[i].a;
                if(a > n)  a = n;
                p += a*farmer[i].p;
                n -= a;
       }
        printf(cout, "%d\n", p);
         return 0;
}
```

例 5：Barn Repair 修理牛棚

在一个月黑风高,下着暴风雨的夜晚,农民约翰的牛棚的屋顶、门被吹飞了。 好在许多牛正在度假，所以牛棚没有住满。 剩下的牛一个紧挨着另一个被排成一行来过夜。 有些牛棚里有牛，有些没有。 所有的牛棚有相同的宽度。 自门遗失以后,农民约翰必须尽快在牛棚之前竖立起新的木板。 他的新木材供应商将会供应他任何他想要的长度,但是供应商只能提供有限数目的木板。 农民约翰想将他购买的木板总长度减到最少。

给出:可能买到的木板最大的数目 M(1<= M<=50);牛棚的总数 S(1<= S<=200); 牛棚里牛的总数 C(1 <= C <=S);和牛所在的牛棚的编号 stall_number(1 <= stall_number <= S),计算拦住所有有牛的牛棚所需木板的最小总长度。 输出所需木板的最小总长度作为答案。

INPUT FORMAT

第 1 行: M ， S 和 C(用空格分开)

第 2 到 C+1 行: 每行包含一个整数，表示牛所占的牛棚的编号。

OUTPUT FORMAT

单独的一行包含一个整数表示所需木板的最小总长度。

SAMPLE INPUT

```
4 50 18
3
4
6
8
14
15
16
17
21
25
26
27
30
31
40
41
42
43
```

SAMPLE OUTPUT

```
25
```

分析：贪心

思路一：要使木板总长度最少，就要使未盖木板的长度最大。

我们先用一块木板盖住牛棚，然后，每次从盖住的范围内选一个最大的空隙，以空隙为界将木板分成两块，重复直到分成 m 块或没有空隙。

思路二：显然，当所有木板均用上时，长度最短（证明....）。正向思维，初始状态有 c 块木板，每块木板只盖住一个牛棚。

由 c 块倒推至 m 块木板，每次寻找这样的两个牛棚：其间距在所有未连接的木板中最小。当这两个牛棚的木板连接时，总木板数减 1，总长度增加 1 。

思路三：显然，当所有木板均用上时，长度最短。用上 m 块木板时有 m-1 各间隙。现在的目标是让总间隙最大。将相邻两个有牛的牛棚之间间隔的牛棚数排序，选取最大的 m-1 个作为间隙，其余地方用木板盖住即可。

参考代码 1： c++

```
#include<iostream>
#decine max 202
using namespace std;
int a[max],b[max], now; //有牛的编号,间隙,最大长度
int m,s,c;

int cmp2(const void *va,const void *vb)
{    return   * (int *)vb - *(int *)va;   }

int main()
{   int i;
    cin>>m>>s>>c;
    for(  i = 0; i < c; i ++) cin>>a[i];
    qsort( a, c, sizeof( a[0] ), cmp2);//编号从大到小
    for( i = 0;i < c - 1; i ++)
         b[i - 1] = a[i] - a[i + 1] - 1;//所有间隔
    qsort( b, c, sizeof( b[0] ), cmp2);//所有间隔按从大到小排列
    now = a[0] - a[c - 1] + 1;
    for( i = 0 ; i < m - 1 ; i ++)
        now = now - b[i];  //减去那些间隔从大到小，直到块数等于最大购入量
    cout<<now<<endl;
    return 0;
}
```

参考代码 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#decine MAXSTALL 1000
int hascow[MAXSTALL];

int intcmp(const void *va, const void *vb)
{       return *(int*)vb - *(int*)va;  }

int main()
{   int n, m, nstall, ncow, i, j, c, lo, hi, nrun;
    int run[MAXSTALL];
    scanf("%d %d %d", &m, &nstall, &ncow);
    for(i=0; i<ncow; i++)
    {   scanf("%d", &c);        hascow[c-1] = 1;    }
    n = 0;
    for(i=0; i<nstall && !hascow[i]; i++)   n++;
    lo = i;
    for(i=nstall-1; i>=0 && !hascow[i]; i--)  n++;
    hi = i+1;    nrun = 0;    i = lo;
    while(i < hi)
    {   while(hascow[i] && i<hi)
            i++;
        for(j=i; j<hi && !hascow[j]; j++) ;
        run[nrun++] = j-i;
        i = j;
    }
    qsort(run, nrun, sizeof(run[0]), intcmp);
    for(i=0; i<nrun && i<m-1; i++)
        n += run[i];
    printf(cout, "%d\n", nstall-n);
    return 0;
}
```

参考代码 3: c++

```
#include<iostream>
#include<algorithm>
using namespace std;
const int MAXS = 202;
const int MAXM = 50;
int dis[MAXS]; int stn[MAXS];
int main()
{       int m,s,c,i,x;
        cin >> m >> s >> c;
        if(m >= c) { cout << c << endl; return 0;}
        for(i = 0; i < c; i++)
              cin >> stn[i];
        sort(stn, stn+c);
        for(i = 0; i < c-1; i++)
              dis[i] = stn[i+1] - stn[i]-1;
        sort(dis,dis+c-1);
        x = stn[c-1] - stn[0] +1;
        for(i = 0; i < m-1; i++)
             x -= dis[c-2-i];
        cout << x << endl;
        return 0;
}
```

例 6：Calfflac

据说如果你给无限只母牛和无限台巨型便携式电脑(有非常大的键盘),那么母牛们会制造出世上最棒的回文。你的工作就是去寻找这些牛制造的奇观(最棒的回文)。

在寻找回文时不用理睬那些标点符号、空格(但应该保留下来以便做为答案输出),只用考虑字母'A'-'Z'和'a'-'z'。要你寻找的最长的回文的文章是一个不超过 20,000 个字符的字符串。

我们将保证最长的回文不会超过 2,000 个字符(在除去标点符号、空格之前)。

输入格式:

输入文件不会超过 20,000 字符。这个文件可能一行或多行，但是每行都不超过 80 个字符(不包括最后的换行符)。

样例输入:

```
Confucius say:Madam,I'm Adam.
```

输出格式:

输出的第一行应该包括找到的最长的回文的长度。

下一行或几行应该包括这个回文的原文（没有除去标点符号、空格），把这个回文输出到一行或多行（如果回文中包括换行符）。

如果有多个回文长度都等于最大值，输出最前面出现的那一个。

样例输出:

```
11
Madam,I'm Adam
```

分析: 枚举。

枚举中间数（也就是认为它是回文数最中间的字母），然后左右扩展（忽略非字母）至出界和不相等。最后更新最大值。

要考虑回文是奇数和偶数两种情况。提示大家在开始扩展之前就判断（很巧妙，大家举几个例子就可以明白了）

输入中的换行符可以维持原样不变。

参考代码:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
char fulltext[21000], text[21000], *pal;  int pallen;
void cindpal(void)
{   char *p, *fwd, *bkwd, *etext;
    int len;
    etext = text+strlen(text);
    for(p=text; *p; p++)
    {   for(fwd=bkwd=p; bkwd >= text && fwd<etext && *fwd==*bkwd; fwd++, bkwd--);
        bkwd++; len = fwd - bkwd;
        if(len > pallen)
              {  pal = bkwd;       pallen = len;      }
        for(bkwd=p, fwd=p+1;bkwd>=text && fwd<etext && *fwd==*bkwd; fwd++, bkwd--);
        bkwd++; len = fwd - bkwd;
        if(len > pallen)
           {    pal = bkwd;    pallen = len;  }
    }
}
void main(void)
{   char *p, *q;    int c, i, n;
    p=fulltext;    q=text;
    while((c = getc(cin)) != EOF)
    {   if(isalpha(c))    *q++ = tolower(c);
        *p++ = c;
    }
    *p = '\0';    *q = '\0';
    cindpal();
    printf("%d\n", pallen);
    n = pal - text;
    for(i=0, p=fulltext; *p; p++)
        if(isalpha(*p))  if(i++ == n) break;
    for(i=0; i<pallen && *p; p++)
    {   putc(*p);
        if(isalpha(*p))    i++;
     }
    printf("\n");
    return 0;
}
```

例 7：Prime Cryptarithm 牛式

下面是一个乘法竖式，如果用我们给定的那 n 个数字来取代*，可以使式子成立的话，我们就叫这个式子牛式。

```
        * * *
    x     * *
     -------
       * * *
     * * *
     -------
     * * * *
```

数字只能取代*，当然第一位不能为 0。

注意一下在美国的学校中教的“部分乘积”，第一部分乘积是第二个数的个位和第一个数的积，第二部分乘积是第二个数的十位和第一个数的乘积.

写一个程序找出所有的牛式。

INPUT FORMAT

Line 1:数字的个数 n。

Line 2:N 个用空格分开的数字（每个数字都∈{1,2,3,4,5,6,7,8,9}）。

OUTPUT FORMAT

共一行，一个数字。表示牛式的总数。 下面是样例的那个牛式。

```
      2 2 2
    x   2 2
     ------
      4 4 4
    4 4 4
  ---------
    4 8 8 4
```

SAMPLE INPUT

```
5
2 3 4 6 8
```

SAMPLE OUTPUT

```
1
```

分析: 枚举

```
   S1  S4  S5
×      S2  S3
```

约束条件只有 3 个：第 3、4 行是 3 位数，第 5 行是 4 位数。按 S1 到 S5 的顺序枚举。
如果 S1×S2>10 或者 S1×S3>10，则 3、4 行肯定不是 3 位数，剪枝。
即 S1*S2+S4*S2/10>=10 || S1*S3+S4*S3/10>=10 剪枝。

参考代码: C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int isgooddigit[10];
int isgood(int n, int d)
{    if(n == 0)return 0;
     while(n)
     {    if(!isgooddigit[n%10])   return 0;
          n /= 10;        d--;    }
     if( d == 0 )  return 1;    else       return 0;
}
int isgoodprod(int n, int m)
{    if(!isgood(n,3) || !isgood(m,2) || !isgood(n*m,4)) return 0;
     while( m )
    {   if(!isgood(n*(m%10),3))   return 0;
        m /= 10;    }
     return 1;
}
int main(void)
{   int i, j, n, nfound;
    for(i=0; i<10; i++)     isgooddigit[i] = 0;
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {   scanf("%d", &j);     isgooddigit[j] = 1;    }
    nfound = 0;
    for(i=100; i<1000; i++)
        for(j=10; j<100; j++)
            if(isgoodprod(i, j))  nfound++;
    printf("%d\n", nfound);
    return 0;
}
```

参考代码 2: C++

```
#include <iostream>

using namespace std;
bool mark[10];
int n, a, b, num[10];

bool check(int m, int k)
{   if (m / k > 0) return false;
    do
    {    if (mark[m % 10] != true)        return false;
         m /= 10;
    } while (m > 0);
   return true;
}

int main(void)
{   scanf ("%d", &n);
    for (int i = 1; i <= n; i++)
    {      scanf ("%d", &num[i]);      mark[num[i]] = true;    }
    int tot(0);
    for (int a1 = 1; a1 <= n; a1++)
      for (int a2 = 1; a2 <= n; a2++)
        for (int a3 = 1; a3 <= n; a3++)
          for (int a4 = 1; a4 <= n; a4++)
            for (int a5 = 1; a5 <= n; a5++)
            {   bool flag(true);
                a = num[a1] * 100 + num[a2] * 10 + num[a3];
                if (check(a * num[a4], 1000) == false)    flag = false;
                if (check(a * num[a5], 1000) == false)    flag = false;
                b = num[a4] * 10 + num[a5];
                if (check(a * b, 10000) == false)  flag = false;
                if (flag) tot++;
            }
   printf("%d\n", tot);
}
```

例 8：Arithmetic Progressions 等差数列

一个等差数列是一个能表示成 a，a+b，a+2b,...，a+nb (n=0,1,2,3,...)的数列。

在这个问题中 a 是一个非负的整数，b 是正整数。写一个程序来找出在双平方数集合(双平方数集合是所有能表示成 p^2＋q^2 的数的集合)S 中长度为 n 的等差数列。

INPUT FORMAT

第一行：N(3<= N<=25),要找的等差数列的长度。

第二行：M(1<= M<=250),搜索双平方数的上界 0 <= p,q <= M。

OUTPUT FORMAT

如果没有找到数列,输出`NONE'。 如果找到了，输出一行或多行，每行由二个整数组成:a,b。 这些行应该先按 b 排序再按 a 排序。

所求的等差数列将不会多于 10,000 个。

SAMPLE INPUT

```
5
7
```

SAMPLE OUTPUT

```
1 4
37 4
2 8
29 8
1 12
5 12
13 12
17 12
5 20
2 24
```

分析：枚举，注意剪枝。预处理把所有的 bisquare 算出来，用一个布尔数组 bene[i]记录 i 是否是 bisquare 另外为了加速，用 list 记录所有的 bisquare（除去中间的空位置，这在对付大数据时很有用），list 中的数据要有序。

然后枚举 list 中的数，把较小的作为起点，两数的差作为公差，接下来就是用 bene 判断是否存在 n 个等差数，存在的话就存入 path 中，最后排序输出。此时缺少重要剪枝，会超时。思考程序发现，费时最多的地方是枚举 list 中的数，所以对这个地方的代码加一些小修改，情况就会不一样： 在判断是否存在 n 个等差数时，从末尾向前判断。 在枚举 list 中的数时，假设为 i,j，那么如果 list[i]+(list[j]-list[i])×(n-1)>lim（lim 是最大可能的 bisquare），那么对于之后的 j 肯定也是大于 lim 的，所以直接 break 掉。（这个非常有效）

参考代码：

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<math.h>
bool ds[125001];   long num[125001];   int n,m;
struct res  {  long a1;   long d; } ans[125001];
int compare(const void * elem1,const void * elem2)
{   res *p1,*p2; p1=(res *)elem1;    p2=(res *)elem2;
    if((p1->d)==(p2->d))return((p1->a1)-(p2->a1));
    else return((p1->d)-(p2->d));
}
int main()
{     int i,j,k,a,b,l,max,ax,whe,ln=0,none=1;;
      scanf("%d%d",&n,&m); max=2*m*m;
      for(i=1;i<=125001;i++);     ds[i]=false;
      for(i=0;i<=m;i++)
         for(j=i;j<=m;j++)
         {    k=i*i+j*j;  ds[k]=true;  }
      l=1;
      for(i=0;i<=max;i++)  if(ds[i]){num[l]=i;l++;}
      l-=1;
     for(a=1;a<=l-n+1;a++)
     {   if(!ds[num[a]])continue;
         for(b=1;b<=max/(n-1);b++)
         {   if(num[a]+b*(n-1)>max)break;
             ax=num[a];
             if(!ds[num[a]])break;
             whe=1;
             for(k=2;k<=n;k++)
             {   ax+=b;
                 if(!ds[ax]){  whe=0;break;}
             }
             if(whe){ln++;ans[ln].a1=num[a];ans[ln].d=b;none=0;}
         }
     }
     qsort(&ans[1],ln,sizeof(res),compare);
     if (none) printf(fpout,"NONE\n");
     for(i=1;i<=ln;i++)
          printf("%d %d\n",ans[i].a1,ans[i].d);
     return 0;
}
```

例 9：Prime Palindromes 回文质数

因为 151 既是一个质数又是一个回文数(从左到右和从右到左是看一样的)，所以 151 是回文质数。

写一个程序来找出范围[a,b](5 <= a < b <= 100,000,000)(一亿)间的所有回文质数。

INPUT FORMAT

第 1 行: 二个整数 a 和 b .

OUTPUT FORMAT

输出一个回文质数的列表，一行一个。

SAMPLE INPUT

```
5 500
```

SAMPLE OUTPUT

```
5
7
11
101
131
151
181
191
313
353
373
383
```

分析：两种思路。

思路 1：用筛法求出 1..1e8 范围内的素数，然后判断每个素数是否是回文数。

思路 2：生成 1..1e8 范围内的回文数，然后判断它是否是素数。

思路 1 的复杂度是 O(n)，思路 2 的复杂度是 O(√n*√n)=O(n)，从复杂度来看两种思路没有差别。但思路 1 用筛法用素数要开 O(n)的数组，在 n=1e8 是就是 90M，超出了空间限制，（而且有可能超时）,而思路 2 的空间复杂度是 O(1)的，所以我们用思路 2。

如何按照从小到大的顺序生成回文数呢？

设生成位数为 l 的回文数，若 l 是奇数，那么从小到大枚举(l+1) div 2 位的数，然后复制翻转生成一个回文数；若 l 是偶数，那么从小到大枚举 l div 2 位的数，然后复制翻转生成一个回文数。上诉两个过程交替进行就可以从小到大生成回文数了。

很有效的优化：任意偶数长度的回文数都不可能为质数（除了 11），因为它能被 11 整除，而 11 恰好只有自身和 1 两个因子。除 2 外，所有偶数均不可能是质数。

参考代码：

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<cmath>
int A,B,answer[100000],numAnswer=0;
bool prime(int a)
{     for(int i=2;i<=sqrt(a);i++)
            if (a%i==0) return false;
     return true;              }

int generate(int i,int j)
{    if (j==1)
    {   char a[30];
        printf("%d",i);     int len=strlen(a);
        for(int k=0;k<len;k++)  a[k+len]=a[len-k-1];
        a[len*2]='\0';
        char* b;      return strtol(a,&b,10);
    }
    if (j==0)
    {   char a[30];
        printf("%d",i);     int len=strlen(a);
        for(int k=0;k<len;k++)    a[k+len-1]=a[len-k-1];
        a[len*2-1]='\0';
        char* b;      return strtol(a,&b,10);
    }
}

int compareint(const void* a,const void* b)
{    return *((int *)a)-*((int *)b);  }

int main()
{   scanf("%d%d",&A,&B);     int x;
    for(int i=1;i<=9999;i++)
        for(int j=0;j<2;j++)//0:odd 1:even
           if (prime(x=generate(i,j))&&x>=A&&x<=B)  answer[numAnswer++]=x;
     qsort(answer,numAnswer,sizeof(int),compareint);
     for(int i=0;i<numAnswer;i++)      printf("%d\n",answer[i]);
     return 0;
}
```

例 10：Ordered Fractions 顺序的分数

输入一个自然数 N,对于一个最简分数 a/b（分子和分母互质的分数）,满足 1<=b<=N,0<=a/b<=1,请找出所有满足条件的分数。

这有一个例子，当 N=5 时，所有解为：

0/1 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1

给定一个自然数 N，1<=n<=160，请编程按分数值递增的顺序输出所有解。

注：①0 和任意自然数的最大公约数就是那个自然数②互质指最大公约数等于 1 的两个自然数。

INPUT FORMAT

单独的一行　一个自然数 N(1..160)

OUTPUT FORMAT:

每个分数单独占一行，按照大小次序排列

SAMPLE INPUT

```
5
```

SAMPLE OUTPUT

```
0/1
1/5
1/4
1/3
2/5
1/2
3/5
2/3
3/4
4/5
1/1
```

分析：枚举所有的分数，判断其是否是最简（分母分子最大公约数＝1），用一个数列记录所有最简分数，然后用快排排序。

两个分数比较大小，只需交叉相乘即可，如：a1/b1<a2/b2 ==>a1*b2<a2*b1。

参考代码：

```
#include <iostream>
using namespace std;
int n,a[100000],b[100000],len=0;

void swap(int &x,int &y)
{     int tmp=x;     x=y;     y=tmp;  }

int gcd(int x,int y)
{   if (x>y) swap(x,y);
    if (x==0) return y;
    return (gcd(y%x,x));
}
void sort(int l,int r)
{    int i=l,j=r;
     float mid=(float)a[(i+j)/2]/(float)b[(i+j)/2];
     for (;; )
     {   for (;(float)a[i]/(float)b[i]<mid;i++) ;
         for (;(float)a[j]/(float)b[j]>mid;j--) ;
         if (i<=j)
         {   swap(a[i],a[j]);   swap(b[i],b[j]);    i++;    j--;  }
         if (i>j) break;
     }
     if (l<j) sort(l,j);
     if (i<r) sort(i,r);
}

int main()
{   in>>n;
    len++;  a[len]=0;   b[len]=1;
    len++;    a[len]=1;    b[len]=1;
    for (int i=2;i<=n;i++)
       for (int j=1;j<=i-1;j++)
            if (gcd(i,j)==1)
            {   len++;   a[len]=j;   b[len]=i; }
    sort(1,len);
    for (int i=1;i<=len;i++)
        out<<a[i]<<"/"<<b[i]<<endl;
    return 0;
}
```

例 11：果子合并

Description:

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 n-1 次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有 3 种果子，数目依次为 1，2，9。可以先将 1、2 堆合并，新堆数目为 3，耗费体力为 3。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为 12，耗费体力为 12。所以多多总共耗费体力=3+12=15。可以证明 15 为最小的体力耗费值。

Input

输入包括两行，第一行是一个整数 n(1<＝n<=10000)，表示果子的种类数。第二行包含 n 个整数，用空格分隔，第 i 个整数 ai(1<＝ai<=20000)是第 i 种果子的数目。

Output:

输出包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^31。

Sample Input

```
3
1 2 9
```

Sample Output

```
15
```

题目分析：典型贪心

本题的思想并不难，就是将所有的果子堆数按增序排列，每次将最小的两个堆合并，直到堆数为 1 为止，这样消耗的体力肯定是最少的。这和构造赫夫曼树的算法不是一般的相似！

思路 1：可以采取以下的步骤进行合并：

（1）先将所有的数按从大到小的顺序排序，取最后两个数合并；（之所以从大到小排列，是为了方便后面的数组维护）

（2）将两个数的和插入到数组中，（插入排序）随时保持数组有序；（如果在整理数组的时候仍然调用快速排序，则速度会降慢，因为此时的数组是有序的，只需要将合并后的数放入到有序数组的适当位置，使用插入排序更合适。）

（3）合并过程中做数组维护，合并到一堆时结束。

思路 2：使用优先队列（方法与思路 1 相同）

参考代码：

```
#include <iostream>
#include <queue>
using namespace std;
typedef struct node
{   int data;
    friend bool operator <(node a , node b)
    {    return a.data > b.data;    }
}Point;
int main()
{   priority_queue<node> Q;
    int n , sum;
    Point v , a , b , x;
    while (scanf ("%d" , &n) != EOF)
    {    sum = 0;
         while (n --)
         {      scanf ("%d" , &v.data);     Q.push(v);    }
         while (!Q.empty())
         {  a = Q.top();    Q.pop();     b = Q.top();  Q.pop();
            x.data = a.data + b.data;        sum += x.data;
            if (!Q.empty())        Q.push(x);
         }
         printf ("%d\n" , sum);
         while (!Q.empty())       Q.pop();
     }
    return 1;
}
```

参考代码 2：快速排序+插入

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<string>
#include<stdlib.h>

typedef long long int64;
int64 end;
int t, i,j,k,temp, ans[2000001], a[2000001], n;

void qs(int head,int tail)
{       int i,j,temp, x;
        i=head; j=tail; x=a[(i+j)/2];
        for(;i<=j;)
        {       for(;a[i]<x;)  i++;
                for(;a[j]>x;)  j--;
                if(i<=j)
                {    temp=a[i];a[i]=a[j];     a[j]=temp;
                     i++;       j--;
                }
        }
        if(i<tail)     qs(i,tail);
        if(head<j)     qs(head,j);
}

int main ( )
{       scanf("%d",&n);
        for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
        qs(1,n);
        for(i=1;i<n;i++)
        {       ans[i]=a[i]+a[i+1];    a[i+1]+=a[i];  t=i+1;
                for(;((a[t]>a[t+1])&&(t+1<=n));)
                {  temp=a[t];  a[t]=a[t+1]; a[t+1]=temp; t++; }
        }
        for(i=1;i<=n-1;i++)    end+=ans[i];
        printf("%d",end);
        return 0;
}
```

参考代码 3：堆

```
#include<stdio.h>
#include<stdlib.h>
unsigned long p[10001]={0};
int n,i,j,t;
unsigned long sum=0;
void pile(int x,int y)
{    int m,h;     h=p[x];     m=2*x;
     while(m<=y)
     {     if(m<y&&p[m]>p[m+1]) m++;
           if(p[m]<h)
                {     p[x]=p[m];     x=m;     m*=2;   }
           else break;
     }
     p[x]=h;
}
int main()
{     scanf("%d",&n);
      for(j=1;j<=n;j++)
            scanf("%d",&p[j]);
      for(j=2;j<=n;j++)
      {     i=j;
            while(p[i]<p[i/2])
            {  t=p[i];    p[i]=p[i/2];    p[i/2]=t;    i/=2;    }
      }
      while(n>1)
      {      t=p[1];      p[1]=p[n];      p[n]=t;      n--;
             pile(1,n);      p[1]+=p[n+1];      sum+=p[1];      pile(1,n);
      }
      printf("%ld",sum);
      return 0;
}
```

参考代码 4：堆

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

int a[10001]={0};
int heapsize;

int left(int i){  return 2*i;}

int right(int i){  return 2*i+1;}

void mini_heapify(int i)
{   int l,r,smallest;
    l=left(i);  r=right(i);
    if (l<=heapsize&&a[l]<a[i])    smallest=l;  else    smallest=i;
    if (r<=heapsize&&a[r]<a[smallest])    smallest=r;
    if (smallest!=i)
   {    swap(a[i],a[smallest]);    mini_heapify(smallest);  }
}

void buildminiheap(int n)
{    heapsize=n;  int i;
     for (i=heapsize/2;i>=1;i--)    mini_heapify(i);
}

int main(int argc, char *argv[])
{  int n,i,t;  long long s=0;
   cin>>n;
   for (i=1;i<=n;i++)      cin>>a[i];
   buildminiheap(n);
   while (heapsize!=1)
   {  swap(a[heapsize],a[1]);
      t=a[heapsize];    //找出第一个最小值暂时存在变量 t 中
      heapsize--;       //减少堆的大小
      mini_heapify(1);   //重新维护
      s=s+t+a[1];      //计算消耗值
      a[1]+=t;          //将合并出来的值放在堆顶
      mini_heapify(1);    //对堆顶做维护
  }
  cout<<s<<endl;
  return 0;
}
```

例 12：Party Lamps 派对灯(IOI98)

在 IOI98 的节日宴会上，我们有 N(10<=N<=100)盏彩色灯，他们分别从 1 到 N 被标上号码。这些灯都连接到四个按钮：

按钮 1：当按下此按钮，将改变所有的灯：本来亮着的灯就熄灭，本来是关着的灯被点亮。
按钮 2：当按下此按钮，将改变所有奇数号的灯。
按钮 3：当按下此按钮，将改变所有偶数号的灯。
按钮 4：当按下此按钮，将改变所有序号是 3*K+1(K>=0)的灯。例如：1,4,7...

一个计数器 C 记录按钮被按下的次数。当宴会开始，所有的灯都亮着，此时计数器 C 为 0。

你将得到计数器 C(0<=C<=10000)上的数值和经过若干操作后某些灯的状态。写一个程序去找出所有灯最后可能的与所给出信息相符的状态，并且没有重复。

INPUT FORMAT

不会有灯会在输入中出现两次。

第一行：N。

第二行：C 最后显示的数值。

第三行：最后亮着的灯,用一个空格分开，以-1 为结束。

第四行：最后关着的灯,用一个空格分开，以-1 为结束。

OUTPUT FORMAT

每一行是所有灯可能的最后状态(没有重复)。每一行有 N 个字符，第 1 个字符表示 1 号灯，最后一个字符表示 N 号灯。0 表示关闭，1 表示亮着。这些行必须从小到大排列（看作是二进制数）。

如果没有可能的状态，则输出一行'IMPOSSIBLE'。

SAMPLE INPUT

```
10
1
-1
7 -1
```

在这个样例中，有 10 盏灯，只有 1 个按钮被按下。最后 7 号灯是关着的。

SAMPLE OUTPUT

```
0000000000
0101010101
0110110110
```

在这个样例中，有三种可能的状态： 一、所有灯都关着 二、1,4,7,10 号灯关着，2,3,5,6,8,9 亮着。 三、1,3,5,7,9 号灯关着，2，4，6，8，10 亮着。

分析：每个按钮按 2 次和没按效果是一样的。所以每个按钮或者按或者不按，一共有 2^4=16 中状态。枚举每个按钮是否按下，然后生成结果，排序输出即可（注意判重）。

参考代码：

```
#include <iostream>
using namespace std;
int main()
{   int n,c,kai[101],guan[101],lk=0,lg=0,now[101],count=0,res[17][101];
    cin>>n>>c;
    int x;    cin>>x;
    for (;x!=-1;)
    {   lk++;      kai[lk]=x;      cin>>x;    }
    cin>>x;
    for (;x!=-1;)
    {   lg++;       guan[lg]=x;        cin>>x;    }
    for (int i=1;i<=n;i++)
         now[i]=1;
    for (int i=1;i>=0;i--)
    {   for (int s=1;s<=n;s++)   now[s]=1-now[s];
        for (int j=1;j>=0;j--)
        {   for (int s=1;s<=n;s+=2)
                 now[s]=1-now[s];
            for (int k=1;k>=0;k--)
            {   for (int s=2;s<=n;s+=2)
                        now[s]=1-now[s];
                for (int t=1;t>=0;t--)
                {  for (int s=1;s<=n;s+=3)
                         now[s]=1-now[s];
                    bool okay=true;
                    int ss=i+j+k+t;
                    if ((ss>c)||((ss%2)!=(c%2))) continue;
                    for (int s=1;s<=lk;s++)
                        if (now[kai[s]]==0) {  okay=false;   break;  }
                    for (int s=1;s<=lg;s++)
                        if (now[guan[s]]==1) {okay=false; break; }
                    if (okay)
                    {    count++;
                         for (int s=1;s<=n;s++)  res[count][s]=now[s];
                    }
                }
            }
        }
    }
```

```
    for (int i=1;i<=count;i++)
    {  for (int j=i+1;j<=count;j++)
        {   bool big=false;
            for (int k=1;k<=n;k++)
               if (res[i][k]!=res[j][k])
                {   big=(res[i][k]>res[j][k])?true:false; break; }
             if (big)
            {     int tmp;
                  for (int k=1;k<=n;k++)
                  {   tmp=res[i][k];  res[i][k]=res[j][k]; res[j][k]=tmp; }
            }
        }
    }
    for (int i=1;i<=count;i++)
    {   for (int j=1;j<=n;j++) cout<<res[i][j];
        cout<<endl;
    }
    if (count==0) cout<<"IMPOSSIBLE"<<endl;
    return 0;
}
```

例 13：纪念品分组

Description

元旦快到了，校学生会让乐乐负责新年晚会的纪念品发放工作。为使得参加晚会的同学所获得的纪念品价值相对均衡，他要把购来的纪念品根据价格进行分组，但每组最多只能包括两件纪念品，并且每组纪念品的价格之和不能超过一个给定的整数。为了保证在尽量短的时间内发完所有纪念品，乐乐希望分组的数目最少。

你的任务是写一个程序，找出所有分组方案中分组数最少的一种，输出最少的分组数目。

Input

输入包含 n+2 行：

第 1 行包括一个整数 w，为每组纪念品价格之和的上限。

第 2 行为一个整数 n，表示购来的纪念品的总件数。

第 3~n+2 行每行包含一个正整数 pi (5 <= pi <= w)，表示所对应纪念品的价格。

Output

输出仅一行，包含一个整数，即最少的分组数目。

Sample Input

```
100
9
90
20
20
30
50
60
70
80
90
```

Sample Output

```
6
```

问题分析：头尾贪心。

参考代码 1：快速排序

```
#include<stdio.h>
struct th  { int price;}  num[30001],temp;
int min(int a,int b){return a<b?a:b;}
int part(int s,int e)
{   int i,j,x;
    x=num[e].price;    i=s-1;
    for(j=s;j<e;++j)
        if(num[j].price>x)
        {  ++i;    temp=num[j];   num[j]=num[i];  num[i]=temp;}
    temp=num[e];    num[e]=num[i+1];    num[i+1]=temp;
    return i+1;
}
void quick(int s,int e)
{    int q;
     if(s<e){ q=part(s,e); quick(s,q-1);   quick(q+1,e);}
}
int main()
{   int n,w,i,max,j,last;    scanf("%d %d",&w,&n);   max=0;
    for(i=1;i<=n;++i)   scanf("%d",&num[i].price);
    quick(1,n);
    for(i=1,j=n;i<=j;)
       if(num[i].price+num[j].price<=w){++max;++i;--j;}   else  {  ++i;++max;}
    printf("%d\n",max);
}
```

参考代码 2：计数排序

```
#include <iostream>
using namespace std;
int p[201]={0},s[30001];
int main()
{   int i,j,k=0,n,m,t;  cin>>m>>n;
    for (i=0; i<n; i++)  {   cin>>t;  p[t]++;  }
    for (i=200; i>=0; i--)
       for (j=0; j<p[i]; j++)    s[k++]=i;
    i=0;  j=n-1;  k=0;  t=0;
    while (k<n)
    {   if (i==j) {  t++; break;}
        if (s[i]+s[j]<=m) { t++; i++; j--;  k+=2; }  else { t++;  i++; k++; }
    }
    cout<<t<<endl;     return 0;
}
```

例 14：Feed Ratios 饲料调配

农夫约翰从来只用调配得最好的饲料来喂他的奶牛。饲料用三种原料调配成：大麦，燕麦和小麦。他知道自己的饲料精确的配比，在市场上是买不到这样的饲料的。他只好购买其他三种混合饲料（同样都由三种麦子组成），然后将它们混合，来调配他的完美饲料。

给出三组整数，表示 大麦：燕麦：小麦 的比例，找出用这三种饲料调配 x：y：z 的饲料的方法。

例如，给出目标饲料 3：4：5 和三种饲料的比例：

```
1:2:3
3:7:1
2:1:2
```

你必须编程找出使这三种饲料用量最少的方案，要是不能用这三种饲料调配目标饲料，输出“NONE”。“用量最少”意味着三种饲料的用量（整数）的和必须最小。 对于上面的例子，你可以用 8 份饲料 1，1 份饲料 2，和 5 份饲料 3，来得到 7 份目标饲料：

```
8*(1:2:3) + 1*(3:7:1) + 5*(2:1:2) = (21:28:35) = 7*(3:4:5)
```

表示饲料比例的整数，都是小于 100 的非负整数。表示各种饲料的份数的整数，都小于 100。一种混合物的比例不会由其他混合物的比例直接相加得到。

INPUT FORMAT

Line 1: 三个用空格分开的整数，表示目标饲料

Line 2..4: 每行包括三个用空格分开的整数，表示农夫约翰买进的饲料的比例

OUTPUT FORMAT

输出文件要包括一行，这一行要么有四个整数，要么是“NONE”。前三个整数表示三种饲料的份数，用这样的配比可以得到目标饲料。第四个整数表示混合三种饲料后得到的目标饲料的份数。

SAMPLE INPUT

```
3 4 5
1 2 3
3 7 1
2 1 2
```

SAMPLE OUTPUT

```
8 1 5 7
```

分析：枚举

只用简单的穷举也可以，只有 100^3=1000000。枚举 i=0..99.j=0..99,k=0..99。i,j,k 分别为三种饲料的份数。当三种饲料恰能构成目标饲料时，记录，保留最小值

参考代码 1：

```
#include <iostream>
using namespace std;
int gcd(int x,int y)
{   if (x<y) swap(x,y);
    if (x==0) return y;
    return gcd(y%x,x);
}

int main()
{    int a[4],b[4],c[4],x,y,z,d;
     for (int i=0;i<=3;i++)
          cin>>a[i]>>b[i]>>c[i];
     for (int i=0;i<100;i++)
        for (int j=0;j<100;j++)
          for (int k=0;k<100;k++)
          {   x=a[1]*i+a[2]*j+a[3]*k;
              y=b[1]*i+b[2]*j+b[3]*k;
              z=c[1]*i+c[2]*j+c[3]*k;
              if (((a[0]!=0)&&(x==0))||((b[0]!=0)&&(y==0))||((c[0]!=0)&&(z==0)))
                        continue;
              if ((x!=0)&&(x%a[0]!=0))||((y!=0)&&(y%b[0]!=0))
                    ||((z!=0)&&(z%c[0]!=0)))   continue;
              if ((x*b[0]==y*a[0])&&(y*c[0]==z*b[0]))
              {    if (x!=0) d=x/a[0];
                   else if (y!=0) d=y/b[0];
                        else if (z!=0) d=z/c[0];
                   cout<<i<<" "<<j<<" "<<k<<" "<<d<<endl;
                   return 0;
               }
          }
    out<<"NONE"<<endl;
    return 0;
}
```

例 15：Home on the Range 家的范围

农民约翰在一片边长是 N (2 <= N <= 250)英里的正方形牧场上放牧他的奶牛。(因为一些原因，他的奶牛只在正方形的牧场上吃草。)遗憾的是,他的奶牛已经毁坏一些土地。(一些 1 平方英里的正方形)

农民约翰需要统计那些可以放牧奶牛的正方形牧场(至少是 2x2 的,在这些较大的正方形中没有一个点是被破坏的，也就是说，所有的点都是“1”)。

你的工作要在被供应的数据组里面统计所有不同的正方形放牧区域(>=2x2)的个数。当然，放牧区域可能是重叠。

INPUT FORMAT

第 1 行:N,牧区的边长。

第 2 到 n+1 行:N 个没有空格分开的字符。

0 表示 "那一个区段被毁坏了";1 表示 " 准备好被吃"。

OUTPUT FORMAT

输出那些存在的正方形的边长和个数，一种一行。

SAMPLE INPUT

```
6
101111
001111
111111
001111
101101
111001
```

SAMPLE OUTPUT

```
2 10
3 4
4 1
```

解题思路：枚举左上角后二分搜索最大边长,记录最大边长后累加。

参考代码：

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

bool k[252][252];
int x[252][252],n, res[252];

int main()
{    memset(k,0,sizeof(k));
     memset(x,0,sizeof(x));
     memset(res,0,sizeof(res));
     cin>>n;    char t;
     for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
        {   cin>>t;
            if(t=='1') {  x[i][j]=1;  k[i][j]=1; }
         }
     for(int i=1;i<=n;i++)
         for(int j=1;j<=n;j++)
             x[i][j]+=x[i][j-1];
     for(int i=1;i<=n;i++)
         for(int j=1;j<=n;j++)
             x[i][j]+=x[i-1][j];
     for(int i=1;i<n;i++)
        for(int j=1;j<n;j++)
          if(k[i][j] && k[i][j+1] && k[i+1][j] && k[i+1][j+1])
         {   int l=1,r=min(n-i+1,n-j+1),m;
             while(r-l>1)
            {  m=(l+r)/2;
               if(x[i+m][j+m]-x[i+m][j-1]-x[i-1][j+m]+x[i-1][j-1] == (m+1)*(m+1))
                       l=m;
               else   r=m;
             }
             res[l+1]++;
          }
        for(int i=250;i>=2;i--)
               res[i]+=res[i+1];
        for(int i=2;i<=250;i++)
               if(res[i]>0)  cout<<i<<" "<<res[i]<<endl;
    return 0;
}
```

例 16：守望者的逃离

Description

恶魔猎手尤迪安野心勃勃，他背叛了暗夜精灵，率领深藏在海底的娜迦族企图叛变。守望者在与尤迪安的交锋中遭遇了围杀，被困在一个荒芜的大岛上。为了杀死守望者，尤迪安开始对这个荒岛施咒，这座岛很快就会沉下去。到那时，岛上的所有人都会遇难。守望者的跑步速度为 17m/s，以这样的速度是无法逃离荒岛的。庆幸的是守望者拥有闪烁法术，可在 1s 内移动 60m，不过每次使用闪烁法术都会消耗魔法值 10 点。守望者的魔法值恢复的速度为 4 点/s，只有处在原地休息状态时才能恢复。

现在已知守望者的魔法初值 M，他所在的初始位置与岛的出口之间的距离 S，岛沉没的时间 T。你的任务是写一个程序帮助守望者计算如何在最短的时间内逃离荒岛，若不能逃出，则输出守望者在剩下的时间内能走的最远距离。注意：守望者跑步、闪烁或休息活动均以秒(s)为单位，且每次活动的持续时间为整数秒。距离的单位为米(m)。

Input

输入仅一行，包括空格隔开的三个非负整数 M, S, T。

Output

输出包含两行：

第 1 行为字符串“Yes”或“No”（区分大小写），即守望者是否能逃离荒岛。

第 2 行包含一个整数。第一行为“Yes”（区分大小写）时表示守望者逃离荒岛的最短时间；第一行为“No”（区分大小写）时表示守望者能走的最远距离。

Sample Input

```
39 200 4
```

Sample Output

```
No
197
```

Hint

100%的数据满足：1 <= T <= 300000, 0 <= M <= 1000, 1 <= S <= 108.

枚举算法思想：利用有魔法就必须放魔法的关键，因为有魔法就必须要放才会快，之后拚命跑，枚举所有可能会休息的时间(刚开始的時时间全部休息，之后先用魔法再用跑的)，算出必须要走的时间。

```
#include<stdio.h>
#include<stdlib.h>
main()
{   int M,S,T;
    while(scanf("%d %d %d",&M,&S,&T)==3)
    {   int a,b,c,max=0,time=300001;
        for(a=0;a<T;a++)
        {    int walk=0,temp=0;
             if(M+4*a>=10) /*将休息的时间换成 MP*/
             {    if((M+4*a)/10*60>=S)
                  {    temp=a+((S%60==0)?(S/60):(S/60+1));
                       if(temp>T) temp=T-a;
                       walk=(temp-a)*60;
                  }
                  else
                  {    temp=a+(M+4*a)/10;
                       if(temp>T) {temp=T;walk=(T-a)*60;}
                            else walk=(M+4*a)/10*60;
                       int t=(int)(S-walk)/17.0;
                       if(temp+t>T)
                       {     walk=walk+(T-temp)*17;     temp=T;   }
                       else  {   walk=walk+(t)*17;  temp=temp+t;  }
                   }
             }
             else  /*全部走路*/
             {    temp=a+((S%17==0)?(S/17):(S/17+1));
                  if(temp>T) {temp=T-a;walk=temp*17;}
                    else    walk=walk+((S%17==0)?(S/17):(S/17+1))*17;
             }
             if(walk>=max&&temp<=time) /*将可以走的最远的距离记录*/
             {     max=walk;    time=temp;     }
             if(walk>=S&&temp<=time) {max=walk;time=temp;}
        }
        if(max>=S)    printf("Yes\n%d\n",time);
           else      printf("No\n%d\n",max);
    }
    return 0;
}
```

参考代码 2：贪心算法。通过数组记录每个时刻能够到达的最大距离。

```
#include <iostream>
using namespace std;
int t[300001];
int main()
{   int i,m,s,T;
    t[0]=0;
    cin>>m>>s>>T;
    for(i=1; i<=T;i++)
        if (m>9) {m-=10;  t[i]=t[i-1]+60;}   //闪
           else  {m+=4;t[i]=t[i-1]; }    //原地休息
     for(i=1; i<=T;i++)
     {  if (t[i-1]+17>t[i]) t[i]=t[i-1]+17;  //原地休息改为跑
        if (t[i]>=s)
        {cout<<"Yes"<<endl<<i<<endl; return 0;}
     }
     cout<<"No"<<endl<<t[T]<<endl;
     return 0;
}
```

参考代码 3：贪心算法。

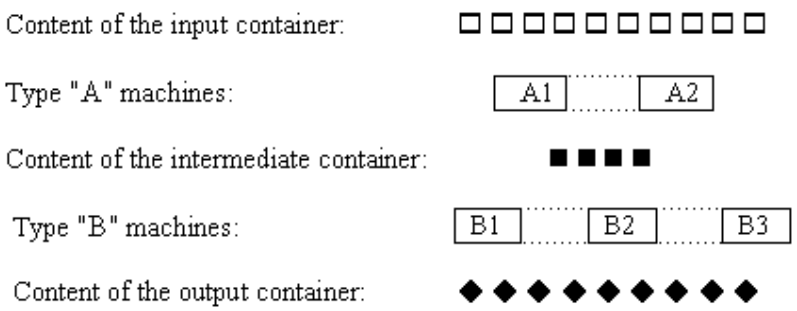
S1 是当前时间可以到达的最远距离，S2 是当前时间通过闪法可以到达的最远距离

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

int main()
{   int i,m,s,T,s1=0,s2=0;
    cin>>m>>s>>T;
    for(i=1; i<=T;i++)
    {   s1+=17;
        if (m>=10) {m-=10; s2+=60;}
          else m+=4;
        if (s2>s1) s1=s2;
        if (s1>=s)
        {cout<<"Yes"<<endl<<i<<endl; break;}
    }
    if (i>T) cout<<"No"<<endl<<s1<<endl;
    return 0;
}
```

例 17：Job Processing 工序安排

一家工厂的流水线正在生产一种产品，这需要两种操作：操作 A 和操作 B。每个操作只有一些机器能够完成。



上图显示了按照下述方式工作的流水线的组织形式。A 型机器从输入库接受工件，对其施加操作 A，得到的中间产品存放在缓冲库。B 型机器从缓冲库接受中间产品，对其施加操作 B，得到的最终产品存放在输出库。所有的机器平行并且独立地工作，每个库的容量没有限制。每台机器的工作效率可能不同，一台机器完成一次操作需要一定的时间。

给出每台机器完成一次操作的时间，计算完成 A 操作的时间总和的最小值，和完成 B 操作的时间总和的最小值。

INPUT FORMAT: (file job.in)

第一行 三个用空格分开的整数：N，工件数量 (1<=N<=1000);M1，A 型机器的数量 (1<=M1<=30);M2，B 型机器的数量 (1<=M2<=30)。

第二行…等 M1 个整数（表示 A 型机器完成一次操作的时间，1..20），接着是 M2 个整数（B 型机器完成一次操作的时间，1..20）

OUTPUT FORMAT: (file job.out)

只有一行。输出两个整数：完成所有 A 操作的时间总和的最小值，和完成所有 B 操作的时间总和的最小值（A 操作必须在 B 操作之前完成）。

SAMPLE INPUT

```
5 2 3
1 1 3 1 4
```

SAMPLE OUTPUT

```
3 5
```

问题分析:

第一问直接贪心就行了，第二问较复杂。

求解过程中是一个工件一个工件地送去加工，每次把一个工件送给用时最短的机器加工，那么显然有 cost[A][1]<=cost[A][2]...<=cost[A][n]，同样适用于 cost[B]。

参考代码:

```
#include <iostream>
using namespace std;
const long maxn=1000;
const long maxm=30;
long i,j,n=0,m1=0,m2=0,mina,minia,minb,minib;
long a[maxm],b[maxm],wa[maxm],wb[maxm],fa[maxn],fb[maxn];
int main(void)
{   cin >>n >>m1 >>m2;
    for (i=0;i<m1;i++)
          cin >>a[i];
    for (i=0;i<m2;i++)
          cin >>b[i];
    for (i=0;i<n;i++)
    {   mina=wa[0]+a[0];  minia=0;
        minb=wb[0]+b[0];  minib=0;
        for (j=1;j<m1;j++)
            if (mina>wa[j]+a[j])
            {    mina=wa[j]+a[j];   minia=j; }
        for (j=1;j<m2;j++)
            if (minb>wb[j]+b[j])
            {   minb=wb[j]+b[j];    minib=j; }
        wa[minia]=mina;   fa[i]=mina;
        wb[minib]=minb;    fb[i]=minb;
    }
    cout <<mina <<' ';
    mina=fa[0]+fb[n-1];
    for (i=1;i<n;i++)
        if (mina<fa[i]+fb[n-i-1])    mina=fa[i]+fb[n-i-1];
    cout <<mina <<endl;
    return 0;
}
```