# 第五次上机解题报告

13211055 罗仪

**摊还分析**

# 题目描述

针对一个k位二进制计数器问题，我们用一个位数组A[0..k-1]作为计数器，计数器采用小端表示法。为了准确分析该 计数器的计数效率，我们采取“势能法”摊还分析。定义计数器在进行某次递增（加1）操作后的势为该次操作后计数器中1的个数。例如，k=2,A=[0 0]时，执行一次递增操作，A变为A‘=[0 1]，这次操作的势是A’中1的个数，即为1。本问题要求你求出从某个给定的计数器状态（即计数器的初始值I）之后连续N个递增操作的摊还代价。

# 输入

多组测试数据。  
每组测试数据共两行：  
第一行两个正整数，分别为二进制计数器的位数K和初始时计数器的值I(0<k<20,0<=I<2^k)；  
第二行一个正整数，为题目要求你求出的递增操作的个数N（0<N<30）。

# 输出

每组测试数据输出N行，第i行输出初始状态后第i个操作的摊还代价（1<=i<=N）。

# 输入样例

3 0  
2

# 输出样例

2  
2

分析：

摊还代价=ci+Φ(Di)-Φ(Di-1),其中Φ(Di)是当前的势，Φ(Di-1)是之前的势。势能法定义势为i次操作后计数器中1的个数。代价ci是ti或ti+1，因为除了复位ti位，还至多置位1位。

具体过程可参考书２６２页。

代码：

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

int a[50]={0};

int k,l;

int increment()

{

int i=0;

int c=0;//c是递增1次的代价ci。

while(i<k&&a[i]==1)

{

a[i]=0;c++;

i++;

}

if(i<k)

{

a[i]=1;c++;

}

return c;

}

int main()

{

while(cin>>k>>l)

{

memset(a,0,sizeof(a));

int i=0;

//初始计数器是l，因此需要把计数器初始化为l的2进制表示。

while(l)

{

a[i]=l%2;

l/=2;

i++;

}

int n;

cin>>n;

int pre=0,d=0;

for(int j=0;j<k;j++)

if(a[j]) pre++;//pre是上一步的势Φ(Di-1)

for(int i=1;i<=n;i++)

{

int c=increment();

d=0;

for(int j=0;j<k;j++)

if(a[j]) d++;// d是这一步的势Φ(Di)

cout<<c+d-pre<<endl;

pre=d;//pre更新

}

}

}

# 哈弗曼编码

# 题目描述

一个文件中存在N种字符，给定这N种字符分别出现的频数，请求出采取哈弗曼编码压缩后该文件的大小（单位为bit）。

# 输入

多组测试数据。  
第一行为字符的种类数N（2<N<500）;  
第二行N个整数，分别为N种字符在文件中出现的频数F[i]（1<=i<=N,0<F[i]<=100）。

# 输出

每组测试数据输出一行，即采取哈弗曼编码压缩后文件的大小。

# 输入样例

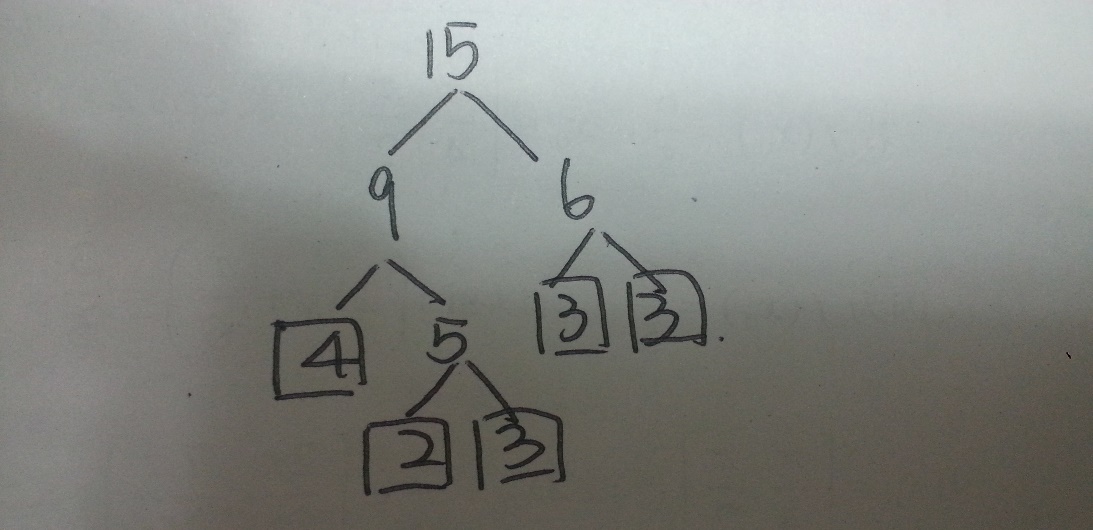
5   
3 4 3 3 2

# 输出样例

35

分析：

STL中的优先队列 priority\_queue,即在每次插入后都会保证队列中的数按优先级有序. 关于priority\_queue<int,vecto<int>r,greater<int> >, 第一个参数为队列中的数据类型,第二个参数为容器类型,第三个参数为比较方法,常用的有greater,less等,也可以根据实际情况自己去写.当参数为greater<int>时,则可以保证每次插入后，队头的数都是最小的数. 使用优先队列每次选择队头的两个数并将其出列,相加后将结果放入队列中,直到队列为空为止.

压缩后文件的大小是编码的二进制位，即

是每个字符c的频数，是字符c的码字的长度，也就是在二叉树中的深度。

对于样例：ans=2\*3+3\*3+4\*2+3\*2+3\*2=35

样例的结果：ans=5+6+9+15=35

（左图是样例最终的编码树。）

代码：

#include <cstdio>

#include <queue>

using namespace std;

typedef long long LL;

priority\_queue<LL ,vector<LL>,greater<LL> >q;//建立小顶堆；

long long n,ans;

int main()

{

while(scanf("%lld",&n)!=EOF)

{

while(!q.empty())

q.pop();

for(int i=1;i<=n;i++)

{

long long x;

scanf("%lld",&x);

q.push(x);

}

ans=0;

while(q.size()>1)

{

LL a=q.top();

q.pop();

LL b=q.top();

q.pop();

ans+=(a+b); // 因为编码长度和其在树中的层数相关

q.push(a+b);

}

printf("%lld\n",ans);

}

return 0;

}

# 高精度乘法

# 题目描述

给定两个8进制正整数A和B（A和B均小于10000位），请利用离散傅里叶变换计算A与B的乘积。

# 输入

多组测试数据。  
每组测试数据只有一行，即给定的正整数A和B。

# 输出

每组测试数据输出一行，即A与B的乘积。

# 输入样例

1 7

# 输出样例

7  
Proposed by LHC

分析：

对于任何一个N位的整数都可以看作是An\*10^(n-1) + An-1\*10^(n-2) + ... + A2\*10^2 + A1\*10 + A0。如果把10看作是一个自变量，那么任何一个整数就可以视作为一个多项式，两个整数相乘也便可以看作是两个多项式相乘。

  A(x)=a0+a1x+a2x2+a3x3+……+an-1xn-1

 A[0](x)=a0+a2x+a4x2+……+an-2xn/2-1

 A[1](x)=a1+a3x+a5x2+……+an-1xn/2-1

 A[0](x2)+x\*A[1](x2)=A(x)

以上是二进制平摊反转置换跟求和的主要式子。

两个次数界为n的多项式A（x）和B（x）相乘，输入输出均采用系数表示法。（假定n为2的幂）

1）使次数界增加一倍：A（x）和B（x）扩充为次数界为2n的多项式，并构造起系数表示

2）求值：两次应用2n阶FFT，计算出A（x）和B（x）的长度为2n的点值表示

3）点乘：计算多项式C(x)=A（x）\*B（x）的点值表示

4）插值：对2n个点值对应用一次FFT计算出其逆DFT，就可以构造出多项式C（x）的系数表示

计算叶子DTF时采用的二进制平摊反转置换，其作用是为了避免算法的递归而实现自底向上的计算方式。回顾一下在计算原串DFT的时候，假设离散点数为0-7，那么有以下过程：

**(0 1 2 3 4 5 7) = (0 2 4 6) + (1 3 5 7) 1**

**(0 2 4 6) = (0 4) + (2 6) 2**

**(1 3 5 7) = (1 3) + (5 7) 2**

**(0 4) = (0) + (4) 3**

**(2 6) = (2) + (6) 3**

**(1 3) = (1) + (3) 3**

**(5 7) = (5) + (7) 3**

分析这些分组的二进制位会发现，第1次分组是根据第0位是否为1来划分的，即奇偶性；第2次分组是根据第1位是否为1来划分的；第三次分组是根据第 2位是否为1来划分的。这个特性与与一般的按照大小划分的数很类似（首先按照最高为是否为1划分，然后是次高位...），因此就可以通过一个算法来使得 00...00 - 11...11这样递增的序列中的每一个数实现高位和低位的翻转，二进制平摊反转置换就是用于达到这个目的。

算法从1开始到N-2（FFT算法要求N必须是2的幂，保证每次折半之后不会出现奇数），因此0和N-1翻转后还是本身，接着维护好一个下标 j ，这个数就是与递增中的第 i 个数翻转之后对应的数，初始化 j 的下标示 N/2，这个数要是从后往前来定义二进制数的话，就会是1，例如若N=8，那么4的二进制位为100，假定只有3位2进制位组成2进制数，从右往左看，其值为1。接下来就是要找 j 的下一个数了，这个数从右往左看应该是2才能够满足要求，于是从右往左寻找，遇到1变为0,遇到0就跳出，并且将该位赋值为1，这个伟大的过程的作用仅仅只是给在右往左定义的二进制数 j 加了一个1。

(如果还不明白的话可以拿一组数模拟一下。)

代码：

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <math.h>

using namespace std;

const double PI = acos(-1.0);

//复数结构体

struct complex

{

double r,i;

complex(double \_r = 0.0,double \_i = 0.0)

{

r = \_r; i = \_i;

}

//三种虚数运算

complex operator +(const complex &b)

{

return complex(r+b.r,i+b.i);

}

complex operator -(const complex &b)

{

return complex(r-b.r,i-b.i);

}

complex operator \*(const complex &b)

{

return complex(r\*b.r-i\*b.i,r\*b.i+i\*b.r);

}

};

/\*

\* 进行FFT和IFFT前的反转变换。

\* 位置i和 （i二进制反转后位置）互换

\* len必须为2的幂

\*/

void change(complex y[],int len)// 二进制平摊反转置换

{

int i,j,k;

for(i = 1, j = len/2;i < len-1; i++)

{

if(i < j)swap(y[i],y[j]);

//交换互为下标反转的元素，i<j保证交换一次

k = len/2;

while( j >= k) // 由最高位检索，遇1变0，遇0变1，跳出

{

j -= k;

k /= 2;

}

if(j < k) j += k;

}

}

/\*

\* 做FFT

\* len必须为2^k形式，

\* on==1时是DFT，on==-1时是IDFT

\*/

void fft(complex y[],int len,int on)

{

change(y,len);  // 调用反转置换

for(int h = 2; h <= len; h <<= 1) // 控制层数

{

complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h)); //初始化单位复根

for(int j = 0;j < len;j+=h)

{

complex w(1,0); // 初始化螺旋因子

for(int k = j;k < j+h/2;k++) //蝴蝶操作

{

complex u = y[k];

complex t = w\*y[k+h/2];

y[k] = u+t;

y[k+h/2] = u-t;

w = w\*wn; // 更新螺旋因子

}

}

}

if(on == -1)

for(int i = 0;i < len;i++)

y[i].r /= len; // IDFT

}

const int MAXN = 200010;

complex x1[MAXN],x2[MAXN];

char str1[MAXN/2],str2[MAXN/2];

int sum[MAXN];

int main()

{

while(scanf("%s%s",str1,str2)==2)

{

int len1 = strlen(str1);

int len2 = strlen(str2);

int len = 1;

while(len < len1\*2 || len < len2\*2)

len<<=1;// 将次数界变成2^n，配合二分与反转

for(int i = 0;i < len1;i++)//倒置存入

x1[i] = complex(str1[len1-1-i]-'0',0);

for(int i = len1;i < len;i++)

x1[i] = complex(0,0);//多余的初始化为0

for(int i = 0;i < len2;i++)

x2[i] = complex(str2[len2-1-i]-'0',0);

for(int i = len2;i < len;i++)

x2[i] = complex(0,0);

//求DFT

fft(x1,len,1);

fft(x2,len,1);

for(int i = 0;i < len;i++)

x1[i] = x1[i]\*x2[i];//计算点乘结果

fft(x1,len,-1);//求逆FFT

for(int i = 0;i < len;i++)

sum[i] = (int)(x1[i].r+0.5);//四舍五入

for(int i = 0;i < len;i++)

{

sum[i+1]+=sum[i]/8;

sum[i]%=8;

}//进位，注意题目是八进制。

len = len1+len2-1;

while(sum[len] <= 0 && len > 0)

len--;

//检索最高位

for(int i = len;i >= 0;i--)

printf("%c",sum[i]+'0');//倒序输出

printf("\n");

}

return 0;

}

# 加分

# 题目描述

某次考试，N名同学在满分为N分的试卷中分别得到了a[i]分  
由于助教们比较仁慈，所以决定给大家的分数适当提高一些  
为了掩人耳目，助教A决定将每个同学提高0或者K的正整数倍的分数  
这时他想：如果每个同学提高的分数可以不同的话，能否最后将同学们的分数变成一个1~N的排列？  
ps：一个1~N的排列是指该数组升序排序之后恰好第一项为1，第二项为2……第N项为N

# 输入

多组测试数据。  
测试数据为2行：  
第一行为两个空格隔开的整数N，K ( 1 <= N <= 500，1 <= K <= N )，表示有N个同学，满分为N，助教加分的系数为K  
第二行为用空格隔开的N个整数，表示每个同学的得分( 1<=得分<=N )

# 输出

输出一行  
该方案可行输出“zheyekeyi”不包含引号  
该方案不可行输出“bienaole”不包含引号

# 输入样例

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

# 输出样例

zheyekeyi  
bienaole

# hint

第一组只要第一个同学的分数加上K=1的1倍，其他同学分数不变即可  
第二组不可

分析：

每个人的成绩输入成一个数组a[i]，再设一个标记数组b，如果已经处理成功（收纳到1~N的成绩中），则标记b[a[i]]=1。对每个a[i]处理时，如果未处理过，则用循环每次加k，再看更新过的a[i]是不是在1~N的成绩中，即看b[a[i]]是否得1。如果超过了n则flag=0，失败，输出结果。否则，输出成功。

代码：

#include<iostream>

#include<cstdio>

using namespace std;

int main()

{

int n,k;

while(cin>>n>>k)

{

int a[600]={0},b[600]={0};

for(int i=0;i<n;i++)

cin>>a[i];

int flag=0;

for(int i=0;i<n;i++)

{

if(b[a[i]]==0)

{b[a[i]]=1;flag=1;}

//注意flag=1，不写的话会WA，因为这种情况是处理成功的。

else {

flag=0;

while(a[i]<=n)

{

a[i]+=k;

if(a[i]<=n&&!b[a[i]])

{

b[a[i]]=1;

flag=1;break;

}

}

if(flag==0)

{cout<<"bienaole"<<endl; break;}

}

}

if(flag)

cout<<"zheyekeyi"<<endl;

}

}

以下由沛沛助教赞助提供

# 穿越雷池

# 题目描述

现有一片雷池区域m\*n，m行n列  
一名战士要从雷区的上端穿越到下端  
每次战士可以选择下一行与他所占位置挨着的3个点走  
直到走到最下一行为止  
每个地雷的伤害不同，所以不同的走法最终受到的总伤害也不同  
求受到的最小伤害时的走法

# 输入

多组测试数据。  
测试数据为m+1行：  
第一行为两个空格隔开的整数m,n ( 1 <= m,n <= 100 )，表示有m行n列  
接下来m行，每行n个用空格隔开的整数，为地雷造成的伤害，1<=伤害<=100

# 输出

每组输出一行  
按顺序输出受到伤害最小的走法  
从第一行到最后一行，输出各个列的坐标(1~n)，中间用空格隔开  
若有多种走法，输出最右的一组

# 输入样例

4 3  
55 32 75  
17 69 73  
54 81 63  
47 5 45  
6 6  
51 57 49 65 50 74  
33 16 62 68 48 61  
2 49 76 33 32 78  
23 68 62 37 69 39  
68 59 77 77 96 59  
31 88 63 79 32 34

# 输出样例

2 1 1 2  
3 2 1 1 2 1

# hint

55 32 75  
17 69 73  
54 81 63  
47 5  45  
2 1 1 2

6 6  
51 57 49 65 50 74  
33 16 62 68 48 61  
2  49 76 33 32 78  
23 68 62 37 69 39  
68 59 77 77 96 59  
31 88 63 79 32 34  
3 2 1 1 2 1

分析：

此题目是一道显而易见的动态规划题目

其基本题意就是从最上面一行，每次可以向其左下、下、右下方向走

问怎样得到最小的权值和，只不过问的是走路的路径

要求只输出路径的列坐标，并且多路权值和相同时输出靠右的答案

首先做动态规划

设区域为 1~n 行 1~m 列

a[i][j]为数组值，dp[i][j]为动态规划数值

将第0列与第m+1列设为很大的值，防止i-1和i+1越界

dp[1][j]=a[1][j];1<=j<=m//第一行值=数组值

dp[i][j]=min(dp[i-1][j-1],dp[i-1][j],dp[i-1][j+1])+a[i][j];

2<=i<=n,1<=j<=m//下面每行=上面3个的最小值+该位置数组值

由于需要存储路径，开一个ans[][]数组，存储到达该位置的前一个位置的列坐标

在每一次dp[i][j]计算时，算出最小值后，选择最小且靠右的一项的列坐标放入该数组内

最终输出答案时，先访问最后一行最小且靠右的dp值，为dp[n][k]，访问该dp值上一个的列坐标，即ans[n][k]，上一个的位置即是dp[n-1][ans[n][k]]

以此类推求出所有路径并输出

附：

有同学不理解靠右，原因可能是怀疑有交叉的结果

其实并不会出现交叉的结果

例如 ：

a1 a2 a3 a4

b1 b2 b3 b4

c1 c2 c3 c4

假设发生交叉了，如：a1+b2+c3==a4+b3+c2;

若a4+b3+c3<=它们，则a4->b3->c3是最靠右最小的路径

若a4+b3+c3>它们，因为6个值a1,a4,b2,b3,c2,c3和是一定的

则a1+b2+c2必然<它们，则a1->b2->c2为最小值，所以不会出现交叉

代码如下：

#include<cstdio>

#include<iostream>

#include<fstream>

using namespace std;

int T,a[105][105],x[105][105][2],m,n,res[105],resj;

int main()

{

while(scanf("%d%d",&m,&n)!=EOF)

{

for(int i=0;i<=m+1;i++)

for(int j=0;j<=n+1;j++)

x[i][j][0]=99999999;//此为dp[i][j]数组

for(int i=1;i<=m;i++)

for(int j=1;j<=n;j++)

scanf("%d",&a[i][j]);//此为a[i][j]存值数组

for(int j=1;j<=n;j++)

x[1][j][0]=a[1][j];

for(int i=2;i<=m;i++)

for(int j=1;j<=n;j++)

{

x[i][j][0]=x[i-1][j+1][0]+a[i][j];

x[i][j][1]=j+1;//先查找最右边一项，下面判断都没带等号，所以最终结果一定靠右

if(x[i][j][0]>x[i-1][j][0]+a[i][j])

{

x[i][j][0]=x[i-1][j][0]+a[i][j];

x[i][j][1]=j;

}

if(x[i][j][0]>x[i-1][j-1][0]+a[i][j])

{

x[i][j][0]=x[i-1][j-1][0]+a[i][j];

x[i][j][1]=j-1;

}

}

int resn=99999999;

for(int j=1;j<=n;j++)

{

if(resn>=x[m][j][0])

{

resn=x[m][j][0];

resj=j;

}

}

for(int i=m;i>=1;i--)

{

if(i!=1)

{

res[i]=resj;//此为开了一个数组存储输出信息，因为刚才的查找的倒叙的，一会正序输出。

resj=x[i][resj][1];

}

else

res[i]=resj;

}

for(int i=1;i<=m;i++)

{

if(i!=m)

printf("%d ",res[i]);

else

printf("%d\n",res[i]);

}

}

}

# 战士科高校的劣等生

# 题目描述

我是一名战士科高校的劣等生，  
我只有一个技能那就是瞬间把任何一个敌人直接杀掉！  
在某次执行任务时，有一排敌人挡在我的面前；  
正当我准备将敌人逐个击破时发现敌人的反击很强烈；  
每当我瞬秒一个敌人的时候，它旁边的敌人总会对我造成即时的反击伤害；  
杀掉一个敌人之后，其他敌人会补上位置！！！  
这导致我不可以无伤地干掉他们这一群军队了；  
但我可以感知到每个敌人的攻击力；  
可是我脑子有限，知道了攻击力也不知道如何选择进攻顺序才能使自己受到的伤害最小  
所以，大家帮帮我这个劣等生吧~

# 输入

多组测试数据。  
测试数据为2行：  
第一行为一个整数N ( 1 <= N <= 200 )，表示有N个敌人站成一排  
第二行为用空格隔开的N个整数，表示每个敌人的攻击力( 1<=攻击力<=1000 )

# 输出

输出一行，为战士受到的最小伤害

# 输入样例

2  
1 100

# 输出样例

1

分析：

此题目是一道区间dp题目，思路和代码不是很难，但是没有接触过的同学可能不太容易想到

题目大意是n个怪，我砍死一个敌人时，旁边的敌人会对我造成伤害，问最小总伤害，怪是会聚拢的，也可以叫补（向）充（前）位（看）置（齐）

思路是这样的

我杀掉连续a[i]~a[j]这么多怪物一共花费的最小值=

先杀a[i]~a[k-1]，再杀a[k+1]~a[j]，最后再杀a[k]的和

设dp[i][j]为杀掉连续a[i]~a[j]的最小值

当i=j时，dp[i][i]=a[i-1]+a[i+1];

当i!=j时，dp[i][j]=min(dp[i][k-1]+dp[k+1][j]+a[i-1]+a[j+1]);

i+1<=k<=j-1;即杀a[i]~a[k-1]最少费dp[i][k-1]，杀a[k+1]~a[j] 最少费dp[k+1][j]，最后杀a[k]，这时队伍合拢，旁边为i-1和j+1两项。

Ps：我（张沛助教）代码不是很好，上面分析理解了就好

代码：

#include<cstdio>

#include<iostream>

#include<fstream>

using namespace std;

int f(int a,int b)//f函数为a和b的最小值

{

if(a<b)return a;

return b;

}

int b[205],x[205][205],N;

int main()

{

while(scanf("%d",&N)!=EOF)

{

for(int i=1;i<=N;i++)

scanf("%d",&b[i]);

b[0]=0;b[N+1]=0;

for(int i=0;i<=N+1;i++)

for(int j=0;j<=N+1;j++)

{

if(i<=j)

x[i][j]=99999999;//此为dp[i][j]

else

x[i][j]=0;

}

for(int i=0;i<N;i++)

for(int j=1;j+i<=N;j++)

{

if(i==0)

x[j][j]=b[j-1]+b[j+1];

else if(i==1)

x[j][j+1]=f(x[j][j],x[j+1][j+1])+b[j-1]+b[j+2];

else

{

for(int k=j;k<=j+i;k++)

x[j][j+i]=f(x[j][k-1]+x[k+1][j+i]+b[j-1]+b[j+i+1],x[j][j+i]);

}

}

printf("%d\n",x[1][N]);

}

}