算法分析与设计教程

习题解答

第 1 章　算法引论

1. 解：算法是一组有穷的规则，它规定了解决某一特定类型问题的一系列计算方法。

   频率计数是指计算机执行程序中的某一条语句的执行次数。

   多项式时间算法是指可用多项式函数对某算法进行计算时间限界的算法。

   指数时间算法是指某算法的计算时间只能使用指数函数限界的算法。

2. 解：算法分析的目的是使算法设计者知道为完成一项任务所设计的算法的优劣，进而促使人们想方设法地设计出一些效率更高效的算法，以便达到少花钱、多办事、办好事的经济效果。

3. 解：事前分析是指求出某个算法的一个时间限界函数（它是一些有关参数的函数）　　；事后测试指收集计算机对于某个算法的执行时间和占用空间的统计资料。

4. 解：评价一个算法应从事前分析和事后测试这两个阶段进行，事前分析主要应从时间复杂度和空间复杂度这两个维度进行分析；　　事后测试主要应对所评价的算法作时空性能分布图。

5. 解：① n=11；　②n=12；　③n=982；　④n=39。

第 2 章　递归算法与分治算法

1. 解：递归算法是将归纳法的思想应用于算法设计之中，　　递归算法充分地利用了计算机系统内部机能，　自动实现调用过程中对于相关且必要的信息的保存与恢复；　　分治算法是把一个问题划分为一个或多个子问题，　　每个子问题与原问题具有完全相同的解决思路，　　进而可以按照递归的思路进行求解。

2. 解：通过分治算法的一般设计步骤进行说明。

3. 解：
```
int fibonacci(int n)
{
    if(n<=1)
        return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

4. 解：
```
void hanoi(int n,int a,int b,int c)
{
    if(n>0)
     {
        hanoi(n-1,a,c,b);
        move(a,b);
        hanoi(n-1,c,b,a);
    }
}
```

5. 解：①　$f(n)=2*2^{n}-n-2$

   ②　$f(n)=O(n*\log n)$

6. 解：算法略。时间复杂度分析：设时间复杂度为 T（n），则

$$T(n)=\begin{cases} C & n\le 0 \\ T(n-1)+T(n-2) & n>0 \end{cases}$$

因此，$T(n)\le 2T(n-1)\le 2^2T(n-2)\le\cdots\le 2^{n-1}T(1)=O(2^n)$。

7. 解：后序遍历二叉树的非递归算法如下：

```
typedef struct{
    BiTree link;
    int flag;
}Stacktype;
void NRpostOrder(BiTree b){
Stacktype Stack[MAX_TREE_SIZE];
BiTree   p;
int top,sign;
if(b==NULL) return;
top=-1;                                  /* 栈顶位置初始化   */
p=b;
while(!(p==NULL&&top==-1)){
   if(p!=NULL){                          /* 结点第一次进栈   */
          Stack[++top].link=p;
          Stack[top].flag=1;
          p=p->lchild;                   /*找该结点的左孩子    */
}
else{
     p=Stack[top].link;
     sign=Stack[top--].flag;
     if(sign==1){                        /* 结点第二次进栈   */
        Stack[++top].link=p;
        Stack[top].flag=2;                /* 标记第二次出栈   */
        p=p->lchild;
     }
     else{
        Visit(p->data) ；                 /* 访问该结点数据域值    */
        p=NULL;
          }
       }
   }
}
```

8. 解：第一趟排序后数组中的元素排列为 35、37、38、36、47、65、53、73；
第二趟排序后数组中的元素排列为 35、37、38、36、47、53、65、73；
第三趟排序后数组中的元素排列为 35、36、37、38、47、53、65、73；
第四趟排序后数组中的元素排列为 35、36、37、38、47、53、65、73。

9. 解：①原数组是有序的；

②n*(n-1)/2 。

10. 解：（略）思路同第 8 题。

11. 解： void maxmin(int A[],int &e_max,int low,int high)

```
{
  int mid,x1,y1,x2,y2;
  if((high-low)<=1){
    if(A[high]>A[low]){
      e_max=A[high];
      e_min=A[low];
    }
    else{
      e_max=A[low];
      e_min=A[high];
    }
  }
  else{
    mid=(low+high)/2;
    maxmin(A,x1,y1,low,mid);
    maxmin(A,x2,y2,mid+1,high);
    e_max=max(x1,x2);
    e_min=min(x1,x2);
  }
}
```

12. 解： void poly_product(float p[],float q[],float r0[],int n)

```
{
  int k,i;
  float *r1,*r2,*r3;
  r1=new float[2*n-1];
  r2=new float[2*n-1];
  r3=new float[2*n-1];
  for(i=0;i<2*n-1;i++)
    r0[i]= r1[i]=r2[i]=r3[i]=0;
  if(n==2)
    product(p,q,c);
  else{
  k=n/2;
  poly_product(p,q,r0,k);
  poly_product(p+k,q+k,r1+2*k,k);
  plus(p,p+k,r2+k,k);
  plus(q,q+k,r3,k);
  poly_product(r2+k,r3,r2+k,k);
  mins(r2+k,r0,2*k-1);
  mins(r2+k,r1+2*k,2*k-1);
```

```
      plus(r0+k,r2+k,r0+k,2*k-1);
      plus(r0+2*k,r1+2*k,r0+2*k,2*k-1);
    }
    delete r1;
    delete r2;
    delete r3;
  }
void product(float p[],float q[],float c[])
{
    c[0]=p[0]*q[0];
    c[2]=p[1]*q[1];
    c[1]=(p[0]+p[1])*(q[0]+q[1])-c[0]-c[2];
}
void plus(float p[],float q[],float c[],int n)
{
   int i;
   for(i=0;i<n;i++)
     c[i]=p[i]+q[i];
}
void mins(float p[],float q[],int n)
{
   int i;
     for(i=0;i<n;i++)
       p[i]=p[i]-q[i];
}
```

第 3 章　贪心算法

1. 解：（1）最优解的序列为（　1,2/3,1,0,1,1,1　　）；

   FO　　（I ）=10+2/3*5+15+6+18+3=166/3　；

   按 Pi 的非增次序输入时的序列为（　1,0,1,4/7,0,1,0　　）；

   FG　　（I ）=10+15+4/7*7+18=47 ；

   因此，FO（I ）/FG（I ）=166/141 ；

   （2）按 wi 的非降次序列输入时的序列为（　1,1,4/5,0,1,1,1　　）；

   此时的背包问题解为　FG^{*}（I ）=10+5+4/5*15+6+18+3=54 ；

   因此，FO（I ）/FG^{*}（I ）=83/81 。

2. 解：举一个反例如下：

   设背包容量为　13，有　7 件物品，它们的容量组成的向量为（　1,2,4,5,3,3,7　），它们的效益值组成的向量为（　6,10,18,14,8,7,7　）。

   按照贪心算法计算求解总效益值为 6+10+18+14=48，此时物品的总容量为 1+2+4+5=12≤13；可是按照　6+10+18+8+7=49，此时物品的总容量为　1+2+4+3+3=13 亦没有超过背包容量　13，由于总效益值　48＜49，因此该策略在这种情况下得不到最优解。

3. 解：

```
#include<iostream>
```

```
using namespace std;
const int N=12;
void OutputResult(int Select[N])
{
   cout<<        ” {0 ” ;
   for(int i=1;i<N;i++)
     if(Select[i]==1)
       cout<<          ” ,  ” <<i;
   cout<<        ’ }  ’ <<endl;
 }
int main()
{
   int Begin[N]={1,3,0,3,2,5,6,4,10,8,15,15};
   int End[N]={3,4,7,8,9,10,12,14,15,18,19,20};
   int Select[N]={0,0,0,0,0,0,0,0,0,0,0,0};
   int i=0;
   int TimeStart=0;
   while(i<N)
   {
      if(Begin[i]>=TimeStart)
      {
          Select[i]=1;
          TimeStart=End[i];
       }
      i++;
    }
    OutputResult(Select);
    result 0;
  }
```

4. 解题思路提示如下：

   连接法： 将间距较小的相邻区间连接成一个大的新区间, 然后将所得的几个新区间的距离相加继而求得最小长度。

5. 解：

```
class JobNode{
    friend void Greedy(JobNode *,int,int);
    friend void main(void);
    public:
      operator int()const{return time;}
    private:
      int ID,time;
};
class MachineNode{
    friend void Greedy(JobNode *,int,int);
          public:
```

```
    operator int()const{return avail;}
  private:
    int ID,avail;
};
template<class Type>
void Greedy(Type a[],int n,int m)
{  if(n<=m){
    cout<<          “为每个作业分配一台机器。    ”<<endl;
    return;
    }
    Sort(a,n);
    MinHeap<MachineNode>H(m);
    MachineNode x;
    for(int i=1;i<=m;i++){
      x.avail=0;
      x.ID=i;
      H.Insert(x);
    }
    for(int i=n;i>=1;i--){
      H.DeleteMin(x);
      cout<<          “将机器”  <<x.ID<<  “从”  <<x.avail<<    “到”
        <<(x.avail+a[i].time)<<                    “的时间段分配给作业”    <<a[i].ID<<endl;
          x.avail+=a[i].time;
          H.Insert(x);
       }
     }
```

6. 解：

```
  #include <iostream>
    #include <cstring>
  using namespace std;
    void deleteS(char *p)
    {
          while(*p)
          {
                *p = *(p+1);
                p++;
          }
    }
  int main()
  {
    char S[240];
    int i,m;
    cout <<"       请输入数字串  :"<<endl;
    cin >> S;
```

```
    cout <<"       请输入须删掉多少个数    :N"<<endl;
    do {
          cout <<"0 < N <        数字串的长度   " <<endl;
          cin >> m;
    } while( m<0 || m >= (int)strlen(S) );
while(m>0)
    {
           i = 0;
           while( i<(int)strlen(S) &&(S[i]<=S[i+1]) )
                  i++;
           deleteS(&S[i]);
              m--;
      }
   /*        删除  S 串中高位的   0*/;
    while(S[0]=='0') deleteS(&S[0]);
    if ( strlen(S)!=0 )
          cout << S <<endl;
    else
              cout <<   ‘ 0 ’ <<endl;
    return 0;
   }
```

7. 解：

```
#include<iostream>
#include<cmath>
using namespace std;
#define maxn 1005
struct POINT
{
  double l,r;
 };
 double x[maxn],y[maxn];
 POINT points[maxn];
 int can,ans,n,d,mk[maxn];
 int comp(const void *a,const void *b)  /*                线段排序比较函数    */
 {
    POINT *aa,*bb;
    aa=(POINT *)a; bb=(POINT *)b;      /*                强制类型转换   */
    if(aa->r-bb->r<1e-8&&bb->r-aa->r<1e-8)/*aa                的右端点与   bb 的右端点相同   */
      return 0;
    if(aa->r<bb->r)         /*aa                的右端点在   bb 的右端点右边  */
      return -1;
    return 1;               /*aa                的右端点在   bb 的右端点左边  */
}
void init()                /*                输入与初始化  */
```

```
{
    int i;
    for(can=1,i=0;i<n;i++)
    {
        cin>>x[i]>>y[i];        /*                  输入岛屿  i  的坐标 */
        if        （d+1e-8>y[i]  ）       /*       如果岛屿  i  在雷达监控范围内，计算海岸线上
                                                   对应的雷达安装区域   */
        {
            points[i].l=x[i]-sqrt(double(d)*d-y[i]*y[i]);
            points[i].r=x[i]+sqrt(double(d)*d-y[i]*y[i]);
        }
        else can=0;
    }
}
void solve()
{
    int i,j;
    qsort(points,n,sizeof(POINT),comp);              /*  将各岛屿按在海岸线的可安装雷达区
                                                         域右端大小排序   */
    for(i=0;i<n;i++) mk[i]=0;
    for(ans=0,i=0;i<n;i++)
      if(!mk[i])
    {
        mk[i]=1;                /*                   进行贪心策略选择    */
        ans++;
        for(j=i+1;j<n;j++)
          if((!mk[j])&&(points[j].l<=points[i].r+1e-8))
              mk[j]=1;
    }
}
int main()
{
    int caseno=0;
    while(cin>>n>>d,(n>0||d>0))
    {
        cout<<"Case"<<++caseno<<":";
        init();
        if(!can)
           cout<<-1<<endl;
        else
        {
            solve();
            cout<<ans<<endl;
         }
```

```
        }
        return 0;
    }
```

第 **4** 章　动态规划算法

1. 解：最短路径为　0->1->6->7->9　，路径长度为　17。
2. 解：

x	0	1	2	3	4
$f_1(x)$	7	13	16	17	19
$d_1(x)$	0	1	2	3	4

x	0	1	2	3	4
$f_2(x)$	13	19	25	28	30
$d_2(x)$	0	0 or 1	1	1	2

x	0	1	2	3	4
$f_3(x)$	18	31	37	43	46
$d_3(x)$	0	1	1	1	1

最后的决策结果：分配给第一、第二、第三工程的资源份额依次是　　2 份、 1 份、 1 份。

3. 解： cbaab。求解过程略。
4. 解：应选择中间　4 个物品，即重量分别为　3,7,2,3　；效益值分别为　6,5,4,3　的物品。求解过程略。
5. 解：

```
#include<iostream>
using namespace std;
const int MAXN=100;
int n,r[MAXN][MAXN], m[MAXN][MAXN],s[MAXN][MAXN];
int money(int i,int j)
{
int k,tmp=0,u;
if(i==j) return 0;
if(i+1==j)
{
   s[i][j]=i;
   return m[i][j];
 }
 u=m[i][j];
 s[i][j]=i;
```

```
 for(k=i+1;k<j;k++)
 {
    tmp=money(i,k)+money(k,j);
    if(tmp<u)
    {
       u=tmp;
       s[i][j]=k;
    }
}
   return u;
}
int main()
{
   int i,j,num=0,a;
   while(cin>>n)
   {
      if(n==0) break;
      else
      {
         num++;
         cout<<"Case"<<num<<":"<<endl;
         for(i=0;i<n;i++)
          for(j=i+1;j<=n;j++)
          {
             cin>>a;
             r[i][j]=a;
             if(a==-1)
                 r[i][j]=65535;
             m[i][j]=r[i][j];
          };
        money(0,n);
        cout<< money(0,n)<<endl;
        }
   }
   return 0;
}
```

6. 求解思路：设所给的两个字符串分别为 A=A[1..m] 和 B=B[1..n] 。

状态表示：考虑从字符子串 A=A[1..i] （按序）变换到字符子串 B[1..j] 的最少字符操作问题，有 d(i,j)= δ(A[1..i],B[1..j]) 。显然， 2 个单字符 a、 b 之间的编辑距离：当 a≠b 时，为 δ(a,b)=1 ；当 a=b 时，为 δ(a,b)=0 。问题的解为 d(m,n) 。

根据最优子结构性质，建立下面的状态转移方程：

d(i,j)=min{d(i-1,j-1)+ δ(A[i],B[j]),d(i-1,j)+1,d(i,j-1)+1}

初始条件为： d(i,0)=i,i=0 ～m;d(0,j)=j,j=0 ～n。

问题的解为 d(m,n) 。

7.　解：

```
#include "cstdio"
#include "iostream"
#include "string"
using namespace std;
int Min(int a,int b)
{
	return a>b?b:a;
}
char pp(char c1,char c2)
{
	if((c1=='['&&c2==']')||(c1=='('&&c2==')'))
		return c2;
	else
		return 0;
}
char* change(char c)
{
	if(c=='[' || c==']')
		return "[]";
	if(c=='('|| c==')')
		return "()";
	return 0;
}
char* scat(const char *s1,char *s2)
{
	char *s=(char*)calloc(strlen(s1)+strlen(s2)+1,sizeof(char));
	strcpy(s,s1);
	strcat(s,s2);
	return s;
}
char* ctos(char c)
{
	char* s=(char*)calloc(2,sizeof(char));
	s[0]=c;
	s[1]='\0';
	return s;
}
int main(void)
{
	int n,i,j,l,k;
	int **m;
	char *str,***regs;
	printf("	请输入括号个数	: ");
```

```
scanf("%d",&n);
m=(int**)calloc(n+1,sizeof(int*));
regs=(char***)calloc(n+1,sizeof(char**));
for(i=0;i<=n;++i)
{
        m[i]=(int*)calloc(n+1,sizeof(int));
        regs[i]=(char**)calloc(n+1,sizeof(char*));
        for(j=0;j<=n;++j)
        {
                regs[i][j]=(char*)calloc(2*n,sizeof(char));
        }
}
fflush(stdin);
str=(char*)calloc(n+1,sizeof(char));
printf("     请输入以（）和[ ] 组合的括号串 : ");
for(i=0;i<n;++i)
        scanf("%c",&str[i]);
str[n]='\0';
printf("     原串：%s\n",str);
for(l=1;l<=n;++l)
{
        for(i=0;i<=n-l;++i)
                {
                        j=i+l-1;
                        int min=l,w=-1;
                        for(k=i+1;k<=j;++k)
                        {
                                if((pp(str[i],str[k])!=0) && (min>m[i+1][k-1]+m[k+1][j]))
//i   与哪个  k 正真匹配
                                {
                                        min=m[i+1][k-1]+m[k+1][j];
                                        w=k;
                                }
                        }
                        if(w!=-1)
                        {
                                m[i][j]=min;

        strcpy(regs[i][j],scat(scat(scat(ctos(str[i]),regs[i+1][w-1]),ctos(pp(str[i
],str[w]))),regs[w+1][j]));
                        }
                        else
                        {
                                m[i][j]=Min(min,1+m[i+1][j]);  //i                没有匹配的
```

```
                    strcpy(regs[i][j],scat(change(str[i]),regs[i+1][j]));
                }
            }
        }
        printf("     最少添加数 : %d\n   正则括号序列 : %s\n",m[0][n-1],regs[0][n-1]);
        return 0;
}
```

8. 求解思路：输入表示成两个长度为 L 的数组 color 和 len
   - L 表示有多少“段”不同的颜色方块
   - color[i] 表示第 i 段的颜色
   - len[i] 表示第 i 段的方块长度

   题目的例子成 color={1,2,3,1}, len={1,4,3,1}

   用(i,j,k) 表示在第 i~j 段方块的右边添加 k 个颜色为 color[j] 的方块后得到的方块序列，相当于考虑第 i~j 段，但让 len[j] 的值增加 k。

   d [i , j , k] 表示把序列 (i,j,k) 消除的最大得分

   考虑最后一段，有两类决策

   ①如果马上消掉，就是 d[i , j -1,0]+(len[j]+ k)2 ；

   ②如果和前面的若干段一起消，设这“若干段”中最后面的那一段是 p(i <=p<j)，得分为 d[i , p, k+len[j]]+d[p+1, j -1,0], 其中 color[p]=color[j]

   边界条件是 f [i , i -1,0]=0 。

第 5 章 回溯算法

1. 解：

```
class Flowshop{
  friend Flow(int * *,int,int[])
  private:
    void Backtrack(int i);
 int * * M          ,          /*          各作业所需的处理时间    */
    *x,             /*          当前作业调度  */
    *bestx,         /*          当前最优作业调度   */
    *f2,            /*          机器 2 完成处理的时间  */
    f1,             /*          机器 1 完成处理的时间  */
    f,              /*          完成时间和  */
    bestf,          /*          当前最优值  */
    n;              /*          作业数 */
}       ;
void Flowshop::Backtrack(int i)
{
  if(i>n){
    for(int j=1;j<=n;j++)
      bestx[j]=x[j];
    bestf=f;
  }
  else
```

```
for(int j=i;j<=n                    ; j++){
   f1+=M[x[j]][1];
   f2[i]=((f2[i-1]>f1)?f2[i-1]:f1)+M[x[j]][2];
   f+=f2[i];
   if(f<bestf){
     Swap(x[i],x[j]);
     Backtrack(i+1);
     Swap(x[i],x[j]);
   }
     f1-=M[x[j]][1];
     f-=f2[i];
    }
}
int Flow(int * * M,int n,int bestx[])
{
   int ub=INT_MAX;
   Flowshop X;
   X.x=new int[n+1]              ;
   X.f2=new int[n+1]               ;
   X.M=M;
   X.n=n;
   X.bestx=bestx;
     X.bestf=ub;
     X.f1=0;
     X.f=0;
     for(int i=0;i<=n;i++)
       X.f2[i]=0,X.x[i]=i;
     X.Backtrack(1);
     delete[]X.x;
     delete[]X.f2;
     return X.bestf;
 }
```

2. 解：

```
class Color{
   friend int mColoring(int,int,int * *);
   private:
   bool Ok(int k);
   void Backtrack(int t);
   int n,                /*                图的顶点数  */
      m,                 /*                可用颜色数  */
      * * a;             /*                图的邻接矩阵  */
      * x,               /*                当前解 */
      long sum;          /*                当前已找到的可    m着色方案数  */
} ;
```

```
bool Color::Ok(int k)              /*                    检查颜色可用性   */
{
   for(int j=1;j<=n;j++)
      if((a[k][j]==1)&&(x[j]==x[k])
          return false;
      return true;
}
void Color::Backtrack(int t)
{
  if(t>n){
    sum++;
    for(int i=1;i<=n;i++)
       cout<<x[i]<<            ’ ’ ;
    cout<<endl;
  }
  else
   for(int i=1;i<=m;i++){
      x[t]=i;
      if(Ok(t))
         Backtrack(t+1);
      x[t]=0;
   }
}
int mColoring(int n,int m,int * *a)
{
    Color X;
    X.n=n;                         /*                    初始化  X*/
    X.m=m;
    X.a=a;
    X.sum=0;
    int *p=new int[n+1];
    for(int i=0;i<=n;i++)
      p[i]=0;
    X.x=p;
    X.Backtrack(1);
    delete[]p;
       return X.sum;
   }
```

3. 解：

```
class Stamp{
    friend int MaxStamp(int,int,int[]);
    private:
      void Backtrack(int i,int r);
      int n,                       /*                    邮票面值数   */
```

```
        m,                  /*          每张信封最多允许贴的邮票数      */
        maxvalue,               /*      当前最优值   */
        maxint,             /*          大整数 */
        maxl,               /*          邮资上界  */
        *x,                 /*          当前解 */
        *y,                 /*          贴出各种邮资所需最少邮票数      */
        *bestx;             /*          当前最优解   */
} ;
void Stamp::Backtrack(int i,int r)
 {
   for(int j=0;j<=x[i-2]*(m-1);j++)
     if(y[j]<m)
       for(int k=1;k<=my[j];k++)
         if          （y[j]+k<y[j+x[i-1]*k]       ）
            y[j+x[i-1]*k]=y[j]+k;
   while(y[r]<maxint)
         r++          ;
     if(i>n){
         if(r-1>maxvalue){
           maxvalue=r-1;
           for(int j=1;j<=n;j++)
              bestx[j]=x[j];
            }
        int *z=new int[maxl+1];
        for(int k=1;k<=maxl;k++)
          z[k]=y[k];
        for(int j=x[i-1]+1;j<=r;j++){
          x[i]=j;
          Backtrack(i+1,r)
          for(int k=1;k<=maxl;k++)
          y[k]=z[k];
        }
   delete []z;
   }
}
int MaxStamp(int n,int m,int bestx[])
{
   Stamp X;
   int maxint=32767;
   int maxl=1500;
     X.n=n;
   X.m=m;
   X.maxvalue=0;
```

```
X.maxint=maxint;
X.maxl=maxl;
X.bestx=bestx;
X.x=new int[n+1];
X.y=new int[maxl+1];
for(int i=0;i<=n;i++)
   X.x[i]=0;
for(int i=1;i<=maxl;i++)
      X.y[i]=maxint;
    X.x[1]=1;
    X.y[0]=0;
    X.Backtrack(2,1);
    delete[]X.x;
    delete[]X.y;
 return X.maxvalue;
}
```

4. 解：（八皇后问题解法）

```
#include<iostream>
using namespace std;
const int Normalize=9;
int Num;
  int q[9];
  bool C[9];
  bool L[17];
  bool R[17];
  void Try(int i)
{
   int k;
   for(int j=1;j<=8;j++)
  {
     if((C[j]==true)&&(L[i-j+Normalize]==true)&&(R[i+j])==true)
    {
        q[i]=j;
        C[j]=false;
        L[i-j+Normalize]=false;
 R[i+j]=false;
 if(i<8)
 {
    Try(i+1);
  }
 else
{
   Num++;
```

```
      cout<<"              方案 "<<Num<<":"<<;
      for(k=1;k<=8;k++)
          cout<<q[k]<<" ";
      cout<<endl;
    }
    C[j]=true;
    L[i-j+Normalize]=true;
    R[i+j]=true;
}
  }
}
int main()
 {
     int i;
     Num=0;
     for(i=0;i<9;i++)
     C[i]=true;
     for(i=0;i<17;i++)
        L[i]=R[i]=true;
     Try(1);
     return 0;
 }
```

5.　解：

```
#include<iostream>
using namespace std;
#define MAX 200
#define MAXN 20
int b[MAXN];
int n,m,k,cm,cn;                    /*cn                  是当前剩余个数,     cm是当前剩余数值   */
int comp(const void *a,const void *b) /*                  快速排序比较函数    */
{
   return *(int *)b-*(int *)a;
}
void NonRecursComput()              /*                    迭代回溯算法主要程序    */
{
  int lev=0,t,a[MAX],x[MAXN];
 bool flag=0;
 x[0]=-1;
 while(lev>=0)
{
   x[lev]=x[lev]+1;
   if(x[lev]==k)                    /*                    下标到最后，回溯    */
   {
      lev--;
```

```
    cm+=a[lev];
    cn++;
 }
 else
{
   if((cm>=b[x[lev]])&&(cn>0)) /*                  如(cm>=b[x[lev]]    ≥0，则尝试再用    b[x[lev]]
                                                   作为一个和项   */
  {
     a[lev]=b[x[lev]];
     cm-=a[lev];
     cn--;
     if((cm==0)&&(cn>=0))          /*                  已满足要求  */
    {
       cout<<"Possible!"<<endl;
       return;
    }
    t=x[lev];
    lev++;
    if(cm-b[t]<0) x[lev]=t;         /*                  重新设置起点位置    */
    else x[lev]=t-1               ;
}
if((cn<0)&&(cm>0))
        {
           lev=-1;
           break;
        }
      }
    }
    if(lev==-1)                   /*                  回溯到根部又无路可走，则无解      */
             cout<<"Impossible!"<<endl;
    return    ;
}
int main()
{
   int i,cnt=1,mincn;
   while(cin>>m>>n>>k)
{
   if(n==0||m==0)
       break;
   if((n>0)&&(m>0))
   {
            cout<<"Case"<<cnt++<<":"<<endl;
            for(i=0;i<k;i++)
              cin>>b[i];
```

```
        qsort(b,k,sizeof(int),comp);    /*              对 k 个数进行从大到小的排序     */
        cm=m,cn=n;
        mincn=int(cm/b[0]);             /*              确定所需数的个数下界     */
        if((mincn>cn)||(cn*b[0]<cm))
           cout<<"Impossible!"<<endl;
        else NonRecursComput();
  }
}
return 0;
}
```

6. 解：

```
  #include<iostream>
  using namespace std;
  #define MAXN 65535
    struct node                     /*                  结点结构 */
  {
      int      left,right;  /*left          、right    分别记录当前结点通过添      0、添 1 而得到的序列   */
      int visited          ；/* 标记是否被访问到,      -1 表示未访问过,     0 表示左支,    1 表示右支 */
    } ；
    typedef node NODE;
    NODE p[MAXN];      /*       表示结点数组   */
    long maxb,a[MAXN]; /*       数组 a 存放最优解  */
int m   ;
void init(int m)   /*            初始化，构建一个有向图     */
{
  long i,k;
  maxb=(1<<m)-1   /*       记有向图的顶点最大数     -1*/
  for(i=0;i<=maxb;i++)
  {
    p[i].left=-1;       /*              设定无左右孩子、无访问标记      */
    p[i].right=-1;
    p[i].visited=-1;
    k=i;                /*              考察当前十进制数是     i  的结点的左右孩子生成情况      */
    k=(k<<1)&maxb;      /*              结点的二进制数尾部添      0，用位操作取后     m位 */
    if       （k!=i ）
      p[i].left=k;      /*                去除重复结点   - 左支 */
      k=k+1;            /*                尝试右支  */
      if       （k!=i ）
        p[i].right=k;   /*                去除重复结点   - 右支 */
    }
  }
  bool NotEqual(int k,int b)  /*          判定数   b  与当前部分解     a 前 k 个是否有相同的    */
{
  bool flag * true;
```

```
int i;
for(i=0;i<=k;i++)
{
   if(a[i]==b)
   {
      flag=false;
      break;
   }
}
   return flag;
}
void Compt()                    /*                          求最优解   a，采用迭代回溯算法    */
{
   long i=1,j;
   bool flag=false;
   a[0]=0;
   p[0].visited=0;
}
while(true)                     /*                          搜索子树  */
{
   while(i<=maxb&&p[a[i-1]].left!=-1&&p[a[i-1]].left!=0
        &&NotEqual(i-1,p[a[i-1]].left)) /*                          如左子树一直可行，沿左子树下去       */
{
   a[i]=p[a[i-1]].left;         /*                          取左孩子的值   */
   p[a[i-1]].visited=0;         /*                          设访问左孩子标志    */
   i++;
   flag=false;                  /*                          设定可能进入右子树的标志      */
}
if(i>maxb)                      /*                          如个数已够，直接输出结果      */
{
   for(j=0;j<=maxb;j++)
     cout<<((a[j]&(1<<(m-1)))>>(m-1));   /*                          按字符串输出   */
   cout<<endl             ;
   return             ;
}
else                            /*                          如个数未满足要求    */
{
   if(p[a[i-1]].right!=-1&&p[a[i-1]].right!=0
      &&NotEqual(i-1,p[a[i-1]].right)) /*                          如右子树可行，进入右子树      */
   {
      a[i]=p[a[i-1]].right;         /*                          取右孩子的值   */
      p[a[i-1]].visited=0;          /*                          设访问右孩子标志    */
      i++;
      flag=true;                    /*                          设定进入左子树的标志     */
```

```
  }
}
while(!flag)                    /*                          剪枝回溯  */
{
  i--             ;
  while(i>0&&p[a[i-1]].visited==1)    /*                    沿右子树返回   */
  {
    a[i]=-1;
    p[a[i-1]].visited=-1;          /*                       取消访问标志   */
    i--             ;
   }
   if(p[a[i-1]].right!=-1&&p[a[i-1]].right!=0
    &&NotEqual(i-1,p[a[i-1]].right))                  /*  如右子树可行,    再进入上一层右子树    */
   {
    a[i]=p[a[i-1]].right;       /*                      取右孩子的值   */
    p[a[i-1]].visited=0;        /*                      设访问右孩子标志    */
    i++;
    flag=true;                  /*                      设定进入左子树的标志    */
   }
  }
 }
}
int main()
{
  int i,n;
  cin>>n;
  for(i=0;i<n;i++)
  {
    cin>>m;
    cout<<"m="<<m<<":"<<endl            ;
    if(m<=0)
      cout<<"impossible!"<<endl;
    else
    {
      init(m);
      Compt();
     }
   }
   return 0;
  }
```

7. 解：

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

```
int set[50];                /*                  记录元素的值   */
int used[50];               /*                  记录元素被使用的情况    */
int n;
int quarter;                /*                  平均值 */
int flag       ;                /*               记录是否已经找到解    */
void Track      （int ， int ， int ）；
int cmp(const void *a,const void *b)
{
   return *(int *)a-*(int *)b;
}
void find(int x)
{
   if(x>6)                  /*                  已经找到目标   */
   {
      flag=1;
      return;
   }
   int i;
   for(i=1;i<=n;i++) /*             找到第一个未使用过的元素作为下一个集合的第一个元素          */
   {
      if(used[i]==0)
         break;
   }
   used[i]=1;
   Track(x,i,set[i]);
   used[i]=0;          /*                  对于找不到的情况，按原路返回      */
}
void Track        （int num ， int k  ， int sum ）；
   {
   if(flag==1) return;
   if(sun==quarter)  /*             若当前集合的和与平均值相等，开始构建下一个集合         */
   {
      find(num+1);
      return;
   }
   for(int i=k+1;i<=n;i++) /*                  每次只需为当前集合引进上次引进元素之后的元素，
                                                强剪枝，可以省去很多重复判断      */
{
   if(used[i]==0&&sum+set[i]<=quarter)
   {
      used[i]=1;
      Track(num,i,sun+set[i]);
      used[i]=0;
   }
```

```
    }
}
int main()
    {
       int T,i;
       cin>>T;
       while(T--)
      {
         cin>>n;
      quarter=0;
      flag=0;
      memset(used,0,sizeof(used));
      for(i=0;i<=n;i++)
      {
         cin>>set[i];
         quarter+=set[i];
      }
if(quarter%6!=0)          /*                    若不能平均分成    6 份，直接得到结果    */
{
   cout<<"no"<<endl;
   continue;
}
       quarter/=6;
       qsort(set+1,n,sizeof(set[0],cmp);
       if(set[n]>quarter)/*                若最大份大于平均数,    可以判断一定不能平均分成      6 份 */
      {
        cout<<"no"<<endl;
        continue;
       }
       find(1);
       if(flag==1)
          cout<<"yes"<<endl;
       else cout<<"no"<<endl;
    }
    return 0;
}
```

8. 解：

```
#include<iostream>
#include<cstring>
#include<cstdlib>
#define H 13          /*          容器的高度  */
#define W 6           /*          容器的宽度  */
#define JH 3          /*          新增宝石的高度   */
#define JW 1           /*          新增宝石的宽度   */
```

```
using namespace std;
char table[H+2][W+2];            /*                容器中宝石的状态    */
bool isw[H][W];                  /*                判断当前格是否为空    */
char newj[JH][JW];               /*                新宝石的状态   */
bool flag        ;
const int step[4][2]={{1,0},{0,1},{1,1},{-1,1}};   /*                          定义  4 个方向的扩展   */
void downOp()                    /*                刷新一遍得到下一个状态    ( 宝石下落后  )*/
 {
    int i,j,x;
    for(j=0;j<W;j++)
    {
      x=H-1;
      for(i=H-1;i>=0;i--)     /*                珠宝下落，依次将为空的位置填满       */
      {
        if(isW[i][j])
          continue;
        table[x--][j]=table[i][j];
      }
      for(i=x;i>=0;i--)       /*                最后为空的赋值为‘     W’*/
        table[i][j]=                ‘ W’;
    }
}
void search()                    /*                消去宝石  */
 {
    flag=true;                   /*                默认本次操作没有消去宝石      */
    int i,j,k,t;
    for(i=0;i<H;i++)
      for(j=0;j<W;j++)
        isW[i][j]=false;
    for(i=0;i<H;i++)             /*                开始判断有没有能消去的宝石，如果有，消去         */
    {
      for(j=0;j<W;j++)
      {
        if(table[i][j]!=              ‘ W’) /*    约束判断，如果是     W则无需操作  */
        {
          for(k=0;k<4;k++)     /*4                  个方向扩展  */
{
   if(i+2*step[k][0]>=0) /*                        约束条件，将不符合条件、越界的舍去        */
   {
     if(table[i+step[k][0]][j+step[k][1]]== table[i][j]&&
        table[i+2*step[k][0]][j+2*step[k][1]]== table[i][j])
     {
        t=0;
        while(i+t*step[k][0]>=0&&
```

```
            table[i+t*step[k][0]][j+t*step[k][1]]== table[i][j])
                                        /*  找到该方向上的所有能消的宝石      */
        {
            isW[i+t*step[k][0]][j+t*step[k][1]]=true;
            t++;
        }
        flag=false;
      }
   }
          }
        }
      }
  }
       if(!flag)              /*                    如果有消去宝石，则刷新状态      */
         downOp()   ;
}
int main()
{
   int i,j,T,l,h,t;
   cin>>T;
   while(T--)
  {
for(i=0;i<H;i++)
{
   cin>>table[i];
}
for(i=0;i<JH;i++)
{
   cin>>newj[i];
}
cin>>l;
l=l-1;
for(i=H-1;i>=0;i--)
{
    if(table[i][1]==                ‘ W’)
    {
        h=i;
        break;
     }
 }
 for(i=h-JH+1;i<=h;i++)        /*                    将新增的宝石放入容器     */
{
    table[i][1]=newj[i-(h-JH+1)][0];
}
```

```
 flag=false;
 while(1)              /*                    开始进入循环，判断是否存在能消去的宝石        */
{
   search();
   if(flag)
      break;
 }
 for(i=0;i<H;i++)
 {
       for(j=0;j<W;j++)
             {
          cout>>table[i][j];
        }
       cout>>endl;
    }
  }
  return 0;
}
```

第6章　　随机化算法

1. 解：

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int main()
{
   int k=0;
   srand(unsigned int)time(NULL);
   for(k=0;k<10;k++)
     cout<<(float)rand()<<endl;
   return 0;
}
```

2. 解：

```
int pollard(int n)
{
int i,k,x,y,d=0;
random_seed(0);
i=1;
k=2;
x=random(1,n);
y=x;
while(i<n){
  i++;
```

```
x=(x*x-1)%n;
d=euclid(n,y-x);
if((d>1)&&(d<n))
   break;
else if(i==k){
   y=x;
   k*=2;
}
  }
      return d;
  }
```

对算法 Pollard 进行深入分析得到，执行算法的 while 循环的循环体 $\sqrt{d}$ 次后，就可以得到 n 的因子 d。因为 n 的最小因子 $d \le \sqrt{n}$，因此，这个算法的时间复杂度为 $O(\sqrt[4]{n})$。

3. 解：

```
template<class Type>
BOOL majority_monte<Type A[],Type &x,int n,double e)
{
   int t,s,i,j,k;
   BOOL flag=FALSE;
   random_seed(0);
 s=log(1/e);
 for(t=1;t<=s;t++){
   i=random(0,n-1);
   k=0;
   for(j=0;j<n;j++)
     if(A[i]==A[j])
        k++;
   if(k>n/2){
     x=A[i];
     flag=TRUE;
     break;
   }
}
return flag;
}
```

这个算法以所给的参数 e 计算出重复测试的次数 s，然后重复执行第 9 行开始的循环 s 次。如果一次也检测不到存在着主元素，就返回 FALSE；只要其中有一次检测到存在着主元素，就返回 TRUE。因此，这个算法的错误概率小于所给参数 e。这样，该算法的时间复杂度为 $O(n\log(1/e))$。

4. 解：

```
bool Queen::QueensLV(int stopVegas)
```

```
{
   RandomNumber rnd;
   int k=1;
   int count=1;
   while((k<=stopVegas)&&(count>0)){
      count=0;
      for(int i=1;i<=n;i++){
        x[k]=i;
        if(Place(k))
           y[count++]=i;
        }
        if(count>0)
          x[k++]=y[rnd.Random(count)];
          }
          return(count>0);
}
void nQueen(int n)
{
   Queen X;
   X.n=n;
   int *p=new int[n+1];
   int *q=new int[n+1];
for(int i=0;i<=n;i++)
{
   p[i]=0;
   q[i]=0;
}
X.y=p;
X.x=q;
int stop=5;
if(n>15)
  stop=n-15;
bool found=false;
while(!X.QueensLV(stop));
if(X.Backtrack(stop+1)){
   for(int i=1;i<=n;i++)
      cout<<p[i]<<" ";
   found=true;
}
cout<<endl;
delete[]p;
delete[]q;
return found;
}
```

用该算法解 8 皇后问题时，对于不同的 stopVegas 值，算法成功的概率 p，一次成功搜索访问的结点数平均值 s，一次不成功搜索访问的结点数平均值 e，以及反复调用算法使得最终找到一个解所访问的结点数的平均值 t=s+(1-p)*e/p 。

第7章 图论与网络流问题

1. 解：

```
#include<iostream>
using namespace std;
#define INF 65535                /*                假定 INF 为无穷大 */
const int MAXN=100;              /*                顶点数的上限 */
int n,e;                         /*                顶点数 n，边数 e*/
int g[MAXN][MAXN],flow[MAXN][MAXN]; /*g[i][j]          为顶点 vi 至顶点 vj 的容量 */
void init （）                    /*                首先输入有向图的信息 */
{ /* 若顶点 vi 至顶点 vj 有边，则设 g[i][j] 为容量；否则 g[i][j] 设为无穷大 */
   int i,j,l,a,b;
   for(i=0;i<=n;i++)             /*                网络初始化 */
     for(j=0;j<=n;j++)
       g[i][j]=INF;              /*                将网络矩阵中无边的元素设为无穷大 */
   for(i=0;i<=e;i++)             /*                输入边信息，构造网络矩阵 */
   {
      cin>>a>>b>>l
      g[a-1][b-1]=l;
      }
 }
int max_flow(int n,int c[][MAXN],int source,int sink,int flow[][MAXN])
                                                                  /* 求最大流 */
{
   int prev[MAXN],que[MAXN],d[MAXN],p,q,i,j,newc; /*          设置队列 que，用于存放未检查
                                                              的结点 */
   if （source=sink ）
       return INF;               /*          如果源点与汇点相同，则返回 */
   memset(flow,0,sizeof(flow)); /*          取一个平凡可行流 */
   while （1）
   {
     for(i=0;i<n;i++)
       prev[i]=0;                /*          标号第一个分量初始化 */
prev[i=source]=source+1;
d[i]=INF;                        /*          标号开始，先给源点标上（ 0， +∞），尚未检查 */
for(p=q=0;p<=q&&!prev[sink];i=que[p++])          /* 若汇点未标上号， 将 vi 的全部相邻结
                                                          点都标号，结束后队列中取下一个结点 */
 for(j=0;j<n;j++)                /*          寻找所有与 vi 连接，且未标号的结点 vi*/
   if(!prev[j]&&c[i][j]<INF)  /*          如果结点 vi 未标号，且 vi 与 vj 相连 */
   {
     if(newc=c[i][j]-flow[i][j]) /*          若 (vi,vj) 上， flow[i][j]<c[i][j] ，则给
```

```
                                                    vj   标号为 (vi,d(vj))*/
     prev[que[q++]=j]=i+1;          /*                  结点  vj  进入队列  */
     d[j]=d[i]<newc?d[i]:newc;
    else if(newc=flow[j][i])
        prev[que[q++]=j]=i-1;
    d[j]=d[i]<newc?d[i]:newc;
    }
if(!prev[sink])
    break;     /*              若所有标号都已检查过,     而标号过程无法进行时,      则算法结束  */
    for(i=sink;i!=source;) /*              当  t  被标上号时，表明得到一条从      s 至 t  的增广路经
                                                    Path，转入调入过程   */
      if(prev[i]>0)
        flow[prev[i]-1][i]+=d[sink];
        i=prev[i]-1;
      else flow[i][-prev[i]-1]-=d[sink];
      i=-prev[i]-1;
    }
    for(j=i=0;i<n;j+=flow[source][i++]);/*                  统计计算从源点发出的流量     */
    return j;
}
int main()
{
      int source,sink,MaxFlow;
      while(cin>>n>>e)              /*                  输入顶点数和边数   */
      {
        init();                 /*                  输入网络信息   */
        source=0       ;              /*              源 source  取 0,  汇  sink  取 n-1*/
        sink=n-1;
        MaxFlow=max_flow(n,g,source,sink,flow);
        cout<<MaxFlow<<endl;        /*              输出最大流   */
      }
      return 0;
   }
```

2. 解：

```
#include<cstdio>
#include<queue>
#include<algorithm>
#include<map>
using namespace std;
#define MAX 201                 /*          最大人数  */
#define MAX_L 201                /*          左部分点集的最大值    */
#define MAX_R 201                /*          右部分点集的最大值    */
struct edge
{
```

```
  int w,v;
};
int n,m;                    /*n                为人数,   m为关系数  */
int left    ,  right   ;          /*          二部图的左右点的数目    */
Edge edge[10000];           /*                关系边 */
bool con[MAX][MAX];         /*                关系矩阵  */
bool yes;                   /*                表示能否形成二部图    */
int used[MAX];      /*          记录建二部图时,    i   是否已经加入到二部图中,若加入,是属
                            于哪一条边  */
int at[MAX];        /*i              在二部图中各个部分的编号,左右部分编号从        1 开始 */
class MATCH
{
  private:
    bool map[MAX_L][MAX_R];  /*          表示左点集合里的    i  能否与右点集合的    j  匹配 */
    int count_left,count_right;
    int lMatchR[MAX_L],rMatchL[MAX_R];
    int prev[MAX_L];
    bool visited[MAX_R];
    void findAugment(int);
    void reverse(int,int);
    void init();
  public:
    bool build();
    int maxMatch();
};
void MATCH::init()
{
  for(int i=1;i<=count_left;i++)
    lMatchR[i]=-1;
  for(int i=1;i<=count_right;i++)
        rMatchL[i]=-1;
}
void MATCH::reverse(int left,int right)
{
  rMatchL[right]=left;
  while(prev[left]!=-1)
  {
    int nextR=lMatchR[left];
    rMatchL[nextR]=prev[left];
    lMatchR[left]=right;
    left=prev[left];
    right=nextR;
  }
  lMatchR[left]=right;
```

```
}
void MATCH::findAugment(int start)
{
   for(int i=1;i<=count_left;i++)
     prev[i]=-1;
   for(int i=1;i<=count_right;i++)
         visited[i]=false;
}
queue<int>Q;
Q.push(start);
bool found=false;
while(!Q.empty()&&!found)
{
   int from=Q.front();
   Q.pop();
   for(int to=1;to<=count_right&&!found;to++)
     if(map[from][to])
     {
       if(!visited[to])
       {
          visited[to]=true;
          if(rMatchL[to]==-1)
          {
             found=true;
             reverse(from,to);
          }
          else
          {
             Q.push(rMatchL[to]);
             prev[rMatchL[to]]=from;
          }
       }
    }
  }
}
int MATCH::maxMatch()
{
   init();
   for(int i=1;i<=count_left;i++)
      findAugment(i);
   int max=0;
   for(int i=1;i<=count_left;i++)
            if(lMatchR[i]!=-1)
               max++;
```

```
    return max;
}
bool MATCH::build()      /*                给左、右点集的大小赋值，初始化        map*/
{
    count_left=left;
    count_right=right;
      for(int i=1;i<=count_left;i++)
         for(int j=1;j<=count_right;j++)
            map[i][j]=false;
for(int i=0;i<m;i++)
{
    int w=edge[i].w;
    int v=edge[i].v;
    if(used[w]==1)
    {
       int temp=w;
       w=v;
       v=w;
     }
    map[at[w]][at[v]]=true;
 }
 return true;
}
void mark(int i,int side)
{
    if(!yes)
      return;
    for(int j=1;j<=n;j++)
      if(con[i][j])
      {
         if(used[j]==-1)
         {
            used[j]=1-side;
            mark(j,1-side);
          }
         else if(used[j]==side)
         {
            yes=false;
            return;
         }
       }
}
int main
{
```

```
MATCH match;
while(scanf("%d%d",&n,&m)!=EOF)
{
   for(int i=1;i<=n;i++)
   {
      for(int j=1;j<=n;j++)
        con[i][j]=false;
      used[i]=-1;
   }
   for(int i=0;i<m;i++)
   {
     scanf("%d%d",&edge[i].w,&edge[i].v);
     con[edge[i].w][edge[i].v]=con[edge[i].v][edge[i].w]=true;
   }
   yes=true;
   for(int i=1;i<=n&&yes;i++)
    if(used[i]==-1)
    {
       used[i]=0;
       mark(i,0);
     }
   if(!yes)
   {
     printf("No\n");
     continue;
   }
   left=right=0;          /*                    统计左右部分点数    */
   for(int i=0;i<=n;i++)
    if(used[i]==0)
    {
       left++;
       at[i]=left;
    }
    else
    {
       right++;
       at[i]=right;
     }
       match.build();
       printf("%d\n",match.maxMatch());
     }
  }
```

3. 解：

```
#include<iostream>
```

```
#include<stdio.h>
#include<stdlib.h>
using namespace std;
#define Course_MAX 101          /*          分别定义课程和学生最大数目      */
#define Stud_MAX 301
bool used[Stud_MAX];
int   link[Stud_MAX];    /*link[]          记录与学生关联的元素,    -1  表示没有关联； link[i]=j
                                表示边 (j,i)    并被标记，顶点匹配    */
int CourseNum,StudNum;          /*          定义课程数、学生数    */
bool map[Course_MAX][Stud_MAX];    /*map          定义课程与学生之间是否有关系      */
void Input()
{
  int i,j,k,x;
  scanf("%d%d",&CourseNum,&StudNum);  /*        输入课程数、学生数    */
  memset(link,0xff,sizeof(link));/*              表示将  X的所有不与   Y的边关联的顶点标上    -1*/
  memset(map,0,sizeof(map));        /*          初始化 */
  for(i=0;i<CourseNum;i++)
  {
    scanf("%d",&k);           /*              有 k 个学生选择课程    i*/
    for(j=0;j<k;j++)
    {
      scanf("%d",&x);         /*              输入学生编号   */
      map[i][x-1]=true;
    }
  }
}
bool DFS(int t)             /*              考察 X 中顶点  t  是否可以增广   */
{
  int i;
  for(i=0;i<StudNum;i++)      /*            对 Y 中所有顶点   i*/
  {
    if(!used[i]&&map[t][i])  /*             如果顶点  i  未使用过，并且   t  与  i  有边相连 */
    {
      used[i]=true          ;     /*      顶点  i  标上使用标记   */
      if(link[i]==-1||DFS(link[i])) /*              若顶点  i  没有被标记的边连到    t，或者  i
                                与 X 中顶点  link[i]    构成的边被标记但    X的顶点  link[i]    可增广 */
      {
        link[i]=t; /*               修改边的标记方式，使顶点     t  与  i  构成的边被标记   */
        return true           ;  /*   此时说明，增广可以进行    */
      }
    }
  }
  return false        ;
}
```

```
int main()
{
   int i,j,num,T;
   scanf("%d",&T);
   for(j=0;j<T;j++)
   {
      num=0;                    /*num                    为匹配数  */
      Input();
      for(i=0;i<CourseNum;i++)      /*                  选择一个顶点    i*/
      {
         memset(used,0,sizeof(used)); /*               左边  X 中所有顶点标上未使用标志     */
         if(DFS(i))
           num++;                 /*                  如果顶点   i  可以增广，那么匹配数加      1*/
      }
      if(num==CourseNum)
         printf("YES\n");          /*                  输出 */
      else printf("NO\n");
    }
    return 0;
  }
```

4. 解题思路：下面先从朝一行上的小行星开火讨论。假如对一行开一次火，清除该行上所有的小行星。由于这些小行星的列号各不相同，那么就要求所有有小行星的列都被射中。为此，以行的编号 x 为顶点向含有小行星的列编号表示的顶点连一条线。

这里，一条连线就表示一颗行星， 边的左右两端构成小行星的坐标。 选择第 1 行开火可以消灭两颗行星。 但选择第 2 行与第 3 行只能分别消灭一颗行星。 幸运的是还可以从列的方向开火。如果选择第 2 列开火可以消灭 2 颗行星。只要抓住关键点 xi 和 yi ，就能解决本题的最少开火次数问题。

本问题涉及一个著名的 “ 二部图的最小点覆盖数 ” 问题： 一个二分图 G 中的最大匹配数等于 G的最小点覆盖数。 只需将 N*N网格的行列编号分别作为一个二部图的左右两部分：行编号是集合 X 的点，列编号是集合 Y 的点，那么在网格中点 (x,y) 处有小行星即相当于自顶点 x 至顶点 y 有一条边，最终得到一个二部图 G。本问题求小行星全部被清理掉的最少次数，就是要使二部图 G的所有边都被覆盖的最少顶点数， 也就是二部图的最大匹配数。最后需要做的，就是计算这个二部图的最大匹配数。

5. 解：

```
#include<iostream>
#include<cmath>
using namespace std;
#define MAXN 51;
#define INF 10000000
int g[MAXN][MAXN],Boy[MAXN][MAXN],m,n;
void InputM()                   /*                  输入，构建矩阵    g*/
{
   int i,j,girl;
   for(i=0;i<m;i++)
```

```
  for(j=0;j<n;j++)
    cin>>Boy[i][j];
for(i=0;i<m;i++)
       for(j=0;j<n;j++)
{
   cin>>girl;
   g[j][i]=girl*Boy[j][i];
}
 }
int kuhn_munkres(int m,int n,int mat[][MAXN],int *match1,int *match2)
{
   int s[MAXN],t[MAXN],l2[MAXN],p,q,i,j,k;
   int l1[MAXN],ret=0;
   for(i=0;i<n;i++)
     l2[i]=0;          /*                取可行顶点标记函数     L：11、 12,12  数组取  0 值*/
    for(i=0;i<m;i++)     /*11[i]                取第  i  行中最大者  */
     for(l1[i]=-INF,j=0;j<n;j++)
       l1[i]=mat[i][j]>l1[i]?mat[i][j]:l1[i];
    memset(match1,0xff,sizeof(int)*n);    /*                数组  match1 与  match2  置-1*/
    memset(match2,0xff,sizeof(int)*n);
    for(i=0;i<m;i++)              /*                考察  X 中的顶点是否皆被匹配    */
    {
memset(t,0xff,sizeof(t));     /*                数组  t  置-1*/
q=0;                          /*                顶点  i  放入  S,   T 为空*/
s[0]=i;
for(p=0;p<=q&&match1[i]<0;p++) /*               确定相对应于    L 的等同子图及顶点和边     */
  for(k=s[p],j=0;j<n&&match1[i]<0;j++)
    if(l1[k]+l2[j]==mat[k][j]&&t[j]<0)    /*                        如顶点  j  未标记,  且是等同边  */
   {
     s[++q]=match2[j],t[j]=k;             /*  取得  Y 中顶点  j , 匹配的点   match2[j]   放入  S*/
     if(s[q]<0)                /*                若无关联  */
       for(p=j;p>=0;j=p)       /*                扩展匹配边集合   */
         match2[j]=k=t[j],p=match1[k],match1[k]=j;
   }
 if(match1[i]<0)               /*                取顶点  i  作为算法中的   u*/
 {
  p=INF;
  for(k=0;k<=q;k++)            /*                取 L(x)+L(y)-w(x,y)       最小者 */
    for(j=0;j<n;j++)
      if(t[j]<0&&l1[s[k]]+l2[j]-mat[s[k]][j]<p)
        p=l1[s[k]]+l2[j]-mat[s[k]][j];
  for(j=0;j<n;j++)
    l2[j]+=(t[j]<0)?0:p;  /*                修正当前的顶点标记函数      L:l1  、 l2*/
  for(k=0;k<=q;k++)
```

```
            l1[s[k]]-=p;
          i--;
        }
      }
      for(i=0;i<m;i++)
        ret+=mat[i][match1[i]];        /*                    计算最优匹配值   */
      return ret;
}
int main()
{
    int match1[MAXN],match2[MAXN],i;
    int MM;
    while(cin>>m)
    {
      n=m;
      InputM();
     MM=kuhn_munkres(m,n,g,match1,match2);
     cout<<MM<<endl;
    }
    return 0;
}
```

6. 解：

```
#define MAXN 210
struct matrix
{
  int c,f;                    /*                    容量，流量  */
}
matrix edge[MAXN][MAXN];          /*                流及容量（邻接矩阵）    */
int m,n;              /*          读入的排水沟（即弧）数目和汇合结点（即顶点）数目        */
int s,t;              /*          源点（结点  1)、汇点（结点  N）*/
int residual[MAXN][MAXN];/*       剩余网络  */
int qu[MAXN*MAXN],qs,qe; /*       队列、队列头和尾   */
int pre[MAXN];          /*pre[i]          为增广路上顶点   i  前面的顶点序号   */
int vis[MAXN];          /*BFS          算法中各顶点的访问标志    */
int maxflow,min_augment; /*       最大流流量、每次增广时的可改进量      */
void find_augment_path() /*BFS       求增广路  */
{
  int i,cu;           /*cu                为队列头结点   */
  memset(vis,0,sizeof(vis));
  qs=0;               /*s                 入队列  */
  qu[qs]=s;
  pre[s]=s;
  vis[s]=1;
  qe=1;
```

```
memset(residual,0,sizeof(residual));
memset(pre,0,sizeof(pre));
while(qs<qe&&pre[t]==0)
{
   cu=qu[qs];
   for(i=1;i<n;i++)
   {
      if(vis[i]==0)
      {
   if(edge[cu][i].c-edge[cu][i].f>0)
   {
      residual[cu][i]=edge[cu][i].c-edge[cu][i].f;
      pre[i]=cu;
      qu[qe++]=i;
      vis[i]=1;
   }
   else if(edge[cu][i].f>0)
   {
      residual[cu][i]=edge[cu][i].f;
      pre[i]=cu;
      qu[qe++]=i;
      vis[i]=1;
            }
         }
      }
      qs++;
   }
}
void augment_flow()                    /*                          计算可改进量   */
{
   int i=t;                            /*t                          为汇点 */
   int j;
   if(pre[i]==0)
   {
      min_augment=0;
      return;
   }
   j=0x7fffffff;
   while(i!=s)                         /*                          计算增广路上可改进量的最小值      */
   {
     if(residual[pre[i]][i]<j)
        j=residual[pre[i]][i];
     i=pre[i];
   }
```

```
    min_augment=j;
}
void update_flow()                  /*                          调整流量  */
{
    int i=t;                        /*t                           为汇点  */
    if(pre[i]==0)
        return;
    while(i!=s)
    {
        if(edge[pre[i]][i].c-edge[pre[i]][i].f>0)
            edge[pre[i]][i].f+=min_augment;
        else if(edge[pre[i]][i].f>0)
            edge[pre[i]][i].f+=min_augment;
        i=pre[i];
    }
}
void solve()
{
    s=1;
    t=n;
    maxflow=0;
    while(1)
    {
        find_augment_path();            /*BFS                         寻找增广路径   */
        augment_flow();                 /*                          计算可改进量   */
        maxflow+=min_augment;
        if(min_augment>0)
            update_flow();              /*                          更新流  */
        else return             ;
    }
}
int main()
{
    int i;
    int u,v,c;
    while(scanf("%d%d",&m,&n)!=EOF)
    {
        memset(edge,0,sizeof(edge));
        for(i=0;i<m;i++)
        {
            scanf("%d%d%d",&u,&v,&c);
            edge[u][v].c+=c;
        }
        solve();
        printf("%d\n",maxflow);
```

```
        }
        return 0;
    }
```

7. 解：

```
#include<cstdio>
#include<vector>
#include<cassert>
#include<algorithm>
using namespace std;
const int n=256;
const int INF=1<<28;
class edge{
    public:
      int u,v,cuv,cvu,flow,cost;
      edge(int cu,int cv,int ccu,int ccv,int cc):u(cu),v(cv),cuv(ccu),cvu(ccv),
          flow(0),cost(cc)){}
      int other(int p)const{return p==u?v:u;}
      int cap(int p)const{return p==u?cuv-flow:cuv+flow;}
      int ecost(int)const;
      void addFlow(int p,int f){flow+=(p==u?f:-f);}
    } ;
    int edge::ecost(int p)const{
     if(flow==0)
       return cost;
     else if(flow>0)
            return p==u?cost:-cost;
        else return p==u?-cost:cost;
}
class Network{
  private:
    vector<edge>eg;
    vector<edge*>net[N];
    edge *prev[N];
    int v,s,t;
    int flow,cost,phi[N],dis[N],pre[N];
    void initNet();
    void initFlow();
    bool dijkstra();
public:
  bool build();
  int getCost()const{return cost;}
  int getFlow()const{return flow;}
  int mincost(int,int);
};
```

```
void Network::initNet(){
  for(int i=0;i<v;i++)
    net[i].clear();
  for(int i=eg.size()-1;i>=0;i--){
    net[eg[i].u].push_back(&eg[i]);
    net[eg[i].v].push_back(&eg[i]);
  }
}
void Network::initFlow(){
  flow=cost=0;
  memset(phi,0,sizeof(phi));
  initNet();
}
bool Network::dijkstra (){
  for(int i=0;i<v;i++)
    dis[i]=INF;
  dis[s]=0;
  prev[s]=NULL;
  pre[s]=-1;
  bool vst[N]={false};
  for(int i=1;i<v;i++){
    int md=INF,u;
for(int j=0;j<v;j++)
  if(!vst[j]&&md>dis[j])
  {
     md=dis[j];
     u=j;
  }
if(md==INF)
  break;
vst[u]=true;
for(int j=net[u].size()-1;j>=0;j--){
  edge *ce=net[u][j];
  if(ce->cap(u)>0){
    int p=ce->other(u);
    int cw=ce->ecost(u)-phi[u]+phi[p];
      assert(cw>=0);
      if(dis[p]>dis[u]+cw){
        dis[p]=dis[u]+cw;
        prev[p]=ce;
        prev[p]=u;
      }
   }
}
```

```
    return(dis[t]!=INF);
}
int Network::mincost(int ss,int tt){
  s=ss;
  t=tt;
  initFlow();
  while(dijkstra()){              /*                  判断是否存在增广路，若否，则算法终止       */
    int ex=INF;
    for(int c=t;c!=s;c=pre[c])
      ex=ex<prev[c]->cap(pre[c])?ex:prev[c]->cap(pre[c]);
                                                  /* 增加增广路径上边的流量     */
   for(int c=t;c!=s;c=pre[c])
     prev[c]->addFlow(pre[c],ex);
   flow+=ex;
   cost+=ex*(dis[t]-phi[t]);
   for(int i=0;i<v;i++)
            phi[i]-=dis[i];          /*                  在增广路径上添加必要的逆向负权边       */
 }
 return cost;
}
bool Network::build(){           /*                  建图 */
  int n,m,p,k;
if(scanf("%d%d%d%d",&n,&m,&k,&p)==EOF)
   return false;
eg.clear();
int con[N][N];                  /*                  工作站的连接情况    */
int at[N];                      /*                  每条船的初始位置    */
for(int i=0;i<n;i++)
  scanf("%d",&at[i]);
for(int i=0;i<=m;i++)
  for(int j=0;j<=m;j++)
    con[i][j]=INF;
for(int i=0;i<k;i++)
{
   int a,b,c;
   scanf("%d%d%d",&a,&b,&c);
   con[a][b]=con[b][a]=c;
}
int to[N][N];                   /*                  港口与工作站的连接情况     */
for(int i=0;i<=n;i++)
  for(int j=0;j<=m;j++)
    to[i][j]=INF;
for(int i=0;i<p;i++)
{
```

```
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    to[a][b]=c;
}
for(int c=0;c<n;c++)
{
    int wt[N];
    bool sign[N];
    for(int i=0;i<=m;i++)
    {
    wt[i]=INF;
    sign[i]=true;
}
int root=at[c];
wt[root]=0;
sign[root]=false;
int min=root;
for(int vv=root;min!=0;vv=min)
{
    min=0;
    for(int ww=1;ww<=m;ww++)
      if(sign[ww])
     {
        if(con[vv][ww]!=INF)
        {
           if(con[vv][ww]+wt[vv]<wt[ww])
           {
              wt[ww]=con[vv][ww]+wt[vv];
           }
        }
        if(wt[ww]<wt[min])
           min=ww;
   }
if(min!=0)
{
   sign[min]=false;
   }
}
for(int i=1;i<=n;i++)
{
   int max=INF;
   for(int j=1;j<=m;j++)
   {
      int tt=to[i][j]+wt[j];
```

```
        if(max>tt)
            max=tt;
       }
       eg.push_back(edge(c+2,i+n+1,1,0,max));
     }
  }
v=2*n+2;
for(int i=0;i<=n;i++)
  eg.push_back(edge(0,i+2,1,0,0));
    for(int i=0;i<=n;i++)
      eg.push_back(edge(i+n+2,1,1,0,0));
    return true;
}
int main()
{
   Network net;
   while(net.build())
      printf("%d\n",net.mincost(0,1));
   return 0;
}
```