差值博弈

我们经常会遇到一种博弈模型：两人轮流做游戏，每次可以获得一个分数，两人都采取最优策略，当游戏结束时，求先手得分-后手得分的最大值。

    我们可以用动态规划进行求解，设dp[i]表示先手面临状态i时，先手进行决策后，先手-后手的最大值。

    状态转移：枚举先手的决策，dp[i]可由dp[j]转移得到。对于先手来说，dp[j]为后手-先手的最大值，因此，-dp[j]为先手-后手的最小值。设转移代价为cost，dp[i]=max(-dp[j]+cost)。

    程序在进行递推时，状态要从后往前进行枚举。也可以采用记忆化搜索的方式，正向思考比较方便。

例1：<http://poj.org/problem?id=1678>

题目大意：有n个数字，两个人P1和P2做游戏。P1先拿走一个数字x1，a<=x1<=b，然后P2再拿走一个数字y1，a<=y1-x1<=b，P1再拿走一个数字x2，a<=x2-y1<=b。然后两个人轮流拿走一个数字，但拿走的数字减去前一个人拿走的数字必须在[a,b]中。当某个人没有数字可以拿走时，游戏结束。S1=P1拿走数字的总和，S2=P2拿走数字的总和。假设两人都以最优策略进行决策，求游戏结束时，S1-S2的最大值。

题解：

先把n个数字w[i]从小到大排序。

设dp[i]为后手拿走第i个数字后，先手-后手的最大值。（注意：dp[i]表示的是，先手面临状态i时，先手-后手的最大值。当前状态下，先手进行决策。）

状态转移：枚举所有可行的状态j，每一种可行的状态j表示先手选择第j个数字。

dp[i]=max(-dp[j]+w[j]) , a<=w[j]-w[i]<=b

由于dp[i]表示后手拿走第i个数字后S1-S2的最优值，所以答案需要枚举w[i]在[a,b]

i，ans=max(-dp[i]+w[i])。

在决策阶段中，max(-dp[j]+w[j]) 可以用单调队列优化到O(n)。  
  
程序：

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<climits>

using namespace std;

const int N=10005;

int w[N];

int dp[N];

int n,a,b;

void input()

{

int i;

scanf("%d%d%d",&n,&a,&b);

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

 scanf("%d",&w[i]);

}

void solve()

{

int i,j,res;

sort(w,w+n);

dp[n-1]=0;

for(i=n-2;i>=0;i--)

{

 if(w[i]<a)break;

 dp[i]=INT\_MIN;

 for(j=i+1;j<=n;j++)

 {

  if(w[j]-w[i]>b)break;

  if(w[j]-w[i]<a)continue;

  dp[i]=max(dp[i],-dp[j]+w[j]);

 }

 if(dp[i]==INT\_MIN)dp[i]=0;

}

res=INT\_MIN;

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

 if(w[i]>=a&&w[i]<=b)

  res=max(res,-dp[i]+w[i]);

if(res==INT\_MIN)res=0;

printf("%d\n",res);

}

int main()

{

//freopen("in.txt","r",stdin);

int t;

scanf("%d",&t);

while(t--)

{

 input();

 solve();

}

return 0;

}

例2：<http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemId=607>

题目大意：盒子里有n个圆饼，两个人轮流从盒子中拿出一些圆饼，至少拿一个，至多m个。当一个人在某一轮拿走盒子中所有的圆饼后，他吃掉自己拿走的圆饼，另一个人把他拿到的圆饼全部放回盒子中。然后游戏重新开始，由没有吃圆饼的那个人先开始。当所有的圆饼全部被吃掉后，游戏结束。假设双方都采取最优策略，求先手吃掉圆饼的最大值。

题解：

题目要求先手的最大值，可以转化为求先手-后手的最大值，因为总的圆饼数量是n，所以先手的最大值=（ans+n）/2。

设dp[i][j][k]为剩下没吃的圆饼数量为i，先手拿走了j个圆饼，后手拿走的k个圆饼，先手-后手的最大值。

状态转移：

1.先手可以一次性拿走盒子中所有的圆饼，即盒子中剩余的圆饼不足m

f[i][j][k]=max(-f[k][0][0]+i-k)，i-j-k<=m

2.先手不能一次性拿走所有的圆饼，枚举先手拿走的圆饼数

f[i][j][k]=max(-f[i][k][j+x])，1<=x<=m，j+k+x<i

先手的最大值=(f[n][0][0]+n)/2

此题采用记忆化搜索较为方便，可以省去很多无效状态的计算。

程序：

#include<iostream>

#include<cstdio>

#include<climits>

using namespace std;

const int N=105;

int f[N][N][N];

int n,m;

void dfs(int i,int j,int k)

{

int x;

if(f[i][j][k]!=INT\_MIN)return;

if(i==0)

{

 f[i][j][k]=0;

 return;

}

if(i-j-k<=m)

{

 dfs(k,0,0);

 f[i][j][k]=max(f[i][j][k],-f[k][0][0]+i-k);

}

for(x=1;x<=m&&j+k+x<i;x++)

{

 dfs(i,k,j+x);

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

}

}

void solve()

{

int i,j,k;

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

 for(j=0;j<=n;j++)

  for(k=0;k<=n;k++)

   f[i][j][k]=INT\_MIN;

dfs(n,0,0);

printf("%d\n",(f[n][0][0]+n)/2);

}

int main()

{

//freopen("in.txt","r",stdin);

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

{

 solve();

}

return 0;

}

例3：2013杭州现场赛I题

<http://acm.hdu.edu.cn/showproblem.php?pid=4778>

题目大意：有G种颜色的宝石，B（B<=21）个背包，每个背包中有一些宝石。Alice和Bob轮流每次拿走一个背包，把背包中的宝石放进熔炉中。当一名角色X把一个背包中的宝石放进熔炉中后，熔炉中某种颜色的宝石有S个，那么这S个宝石变成一颗魔法石，角色X获得这颗魔法石（如果形成了多颗魔法石，角色X获得多颗魔法石）。如果角色X在某一轮中获得了魔法石，那么他可以接着额外进行一轮游戏，如果在额外的一轮游戏中，又获得了魔法石，那么他又可以继续进行一轮游戏。游戏一共进行B轮，若两人都采取最优策略，求游戏结束后，Alice获得魔法石数量-Bob获得魔法石数量的最大值。Alice先进行游戏。

题解：

因为B<=21，所有可以把已选背包的状态进行状态压缩。

设dp[i]表示已经拿走背包的状态压缩为i，先手面临状态i，先手进行决策，先手获得的魔法石-后手获得的魔法石的最大值。

状态转移：

枚举剩下的背包j，设delta为先手当前拿走背包j后获得的魔法石数量，分两种情况转移：

1.delta>0，先手额外进行一轮游戏，转移不交换先后手次序

dp[i]=max(dp[i+pw[j]]+delta)，i&pw[j]=0

2.delta=0，交换先后手次序

dp[i]=max(-dp[i+pw[j]])，i&pw[j]=0

其中，pw[j]=2j

ans=dp[0]  
  
程序：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<climits>

using namespace std;

const int N=22;

const int M=2100000;

int pw[N];

int color[N][8];

int dp[M];

int num[M];

int cnt[M][8];

int G,B,S;

void init()

{

int i;

pw[0]=1;

for(i=1;i<=21;i++)

 pw[i]=pw[i-1]\*2;

}

void input()

{

int i,j,k,c,n;

memset(color,0,sizeof(color));

for(i=0;i<B;i++)

{

 scanf("%d",&n);

 for(j=0;j<n;j++)

 {

  scanf("%d",&c);

  c--;

  color[i][c]++;

 }

}

for(j=0;j<G;j++)

 cnt[0][j]=0;

num[0]=0;

for(i=1;i<pw[B];i++)

{

 for(j=0;j<B;j++)

  if(i&pw[j])break;

 num[i]=0;

 for(k=0;k<G;k++)

 {

  cnt[i][k]=cnt[i-pw[j]][k]+color[j][k];

  num[i]+=cnt[i][k]/S;

 }

}

}

void solve()

{

int i,j,inc;

for(i=0;i<pw[B];i++)

 dp[i]=INT\_MIN;

dp[pw[B]-1]=0;

for(i=pw[B]-1;i>0;i--)

 for(j=0;j<B;j++)

  if(i&pw[j])

  {

   inc=num[i]-num[i-pw[j]];

   if(inc>0)dp[i-pw[j]]=max(dp[i-pw[j]],dp[i]+inc);

   else dp[i-pw[j]]=max(dp[i-pw[j]],-dp[i]);

  }

printf("%d\n",dp[0]);

}

int main()

{

//freopen("in.txt","r",stdin);

init();

while(scanf("%d%d%d",&G,&B,&S)&&G!=0)

{

 input();

 solve();

}

return 0;

}