

基础 DP

POJ 3176: Cow Bowling 数字三角形问题，DP 方程不再赘述。代码如下

```
/*

*/
#define method_1
#ifdef method_1
/*
数字三角形问题，DP 方程不再赘述。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=350+5;
const ll INF=0x3f3f3f3f3f3f3f3fll;
int n,a[maxn][maxn];
ll d[maxn][maxn],ans;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Cow Bowling.in","r",stdin);
    memset(d,-INF,sizeof(d));
    cin>>n;
    for(int i=1;i<=n;i++) for(int j=1;j<=i;j++) cin>>a[i][j];
    d[1][1]=(ll)a[1][1];ans=(ll)a[1][1]; //注意这里 ans 不能初始化为-INF，
    否则 n=1 的情况会出问题
    for(int i=2;i<=n;i++){
```

```

        for(int j=1;j<=i;j++){
            if(j-1>=1) d[i][j]=max(d[i-1][j-1],d[i-1][j])+a[i][j];
            else d[i][j]=d[i-1][j]+a[i][j];
            if(i==n) ans=max(ans,d[n][j]);
        //      cout<<d[i][j]<<" ";
        }
    //      cout<<endl;
    }
    cout<<ans;
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 2229: Sumsets 题意:给出一个整数 n ,求解该整数 n 有多少种由 2 的幂次之和组成的方案。法一可以直接分析出递推式。解题思路: 1.可以将 n 用二进制表示。 $n=1$,只有 1 种表示方法。 $n=2$, $10(2)$, 二进制表示下, 可以分拆成 $\{1,1\}$, $\{10\}$ 有两种表示方法。 $n=3$, $11(2)$, 可以分拆成 $\{1,1,1\}$, $\{10,1\}$ 。 $n=4$, $100(2)$, $\{1,1,1,1\}$, $\{10,1,1\}$, $\{10,10\}$, $\{100\}$ 。总结: 如果所求的 n 为奇数, 那么所求的分解结果中必含有 1, 因此, 直接将 $n-1$ 的分拆结果中添加一个 1 即可为 $s[n-1]$ 。如果所求的 n 为偶数, 那么 n 的分解结果分两种情况 1.含有 1 这种情况可以直接在 $n-1$ 的分解结果中添加一个 1 即可, $s[n-1]$ 。2.不含有 1 那么, 分解因子的都是偶数, 将每个分解的因子都除以 2, 刚好是 $n/2$ 的分解结果, 并且可以与之一一对应, 这种情况有 $s[n/2]$ 。法二用完全背包实现。为了避免 $1+2+1$ 和 $1+1+2$ 这样的重复计数。外层从小到大循环物品(这样小物品在前尝试, 大物品在后尝试), 内层正序循环则保证了完全背包。代码如下

```

/*
题意:给出一个整数 n,求解该整数 n 有多少种由 2 的幂次之和组成的方案。
*/
#define method_1
#ifdef method_1
/*
法一可以直接分析出递推式。

```

解题思路:

1. 可以将 n 用二进制表示。

$n=1$, 只有 1 种表示方法。

$n=2$, $10(2)$, 二进制表示下, 可以分拆成 $\{1,1\}$, $\{10\}$ 有两种表示方法。

$n=3$, $11(2)$, 可以分拆成 $\{1,1,1\}$, $\{10,1\}$ 。

$n=4$, $100(2)$, $\{1,1,1,1\}$, $\{10,1,1\}$, $\{10,10\}$, $\{100\}$ 。

总结: 如果所求的 n 为奇数, 那么所求的分解结果中必含有 1, 因此, 直接将 $n-1$ 的分拆结果中添加一个 1 即可 为 $s[n-1]$ 。

如果所求的 n 为偶数, 那么 n 的分解结果分两种情况

1. 含有 1 这种情况可以直接在 $n-1$ 的分解结果中添加一个 1 即可, $s[n-1]$ 。

2. 不含有 1 那么, 分解因子的都是偶数, 将每个分解的因子都除以 2, 刚好是 $n/2$ 的分解结果, 并且可以与之——对应, 这种情况有 $s[n/2]$ 。

*/

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000000+5;
const int INF=0x3f3f3f3f;
const ll mod=1000000000ll;
//const int num[]={0,1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536,131072,262144,524288}; //a[1]~a[20]
int n;
ll d[maxn]={0},ans=0ll; //d[i]表示 i 分解的方案数
int main() {
    ios::sync_with_stdio(false);
    //freopen("Sumsets.in","r",stdin);
    /*
    int base=1;
```

```

    for(int i=1;i<=20;i++) base*=2,cout<<base<<" ";
    */
    cin>>n;
    d[1]=1ll;
    for(int i=2;i<=n;i++){
        if(i&1) d[i]=d[i-1];
        else d[i]=(d[i-1]+d[i/2])%mod;
//    D(d[i]);
    }
    cout<<d[n];
    return 0;
}
#endif
#ifdef method_2
/*
法二用完全背包实现。
为了避免 1+2+1 和 1+1+2 这样的重复计数。
外层从小到大循环物品（这样小物品在前尝试，大物品在后尝试），内层正序循环则
保证了完全背包。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000000+5;
const int INF=0x3f3f3f3f;
const ll mod=1000000000ll;
const int num[]={0,1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,
16384,32768,65536,131072,262144,524288};//a[1]~a[20]

```

```

int n;
ll d[maxn]={0},ans=0ll; //d[i]表示 i 分解的方案数
int main() {
    ios::sync_with_stdio(false);
    //freopen("Sumsets.in","r",stdin);
    /*
    int base=1;
    for(int i=1;i<=20;i++) base*=2,cout<<base<<" ";
    */
    cin>>n;
    d[0]=d[1]=1ll;
    for(int i=1;i<=20;i++){
        for(int j=2;j<=n;j++){
            if(j>=num[i]) d[j]+=d[j-num[i]];
//            d[j]%=mod; 这样写会比下面这一行慢上一倍
            while(d[j]>=mod) d[j]-=mod;
        }
    }
    cout<<d[n];
    return 0;
}
#endif
#ifdef method_3
/*

*/

#endif

```

POJ 2385: Apple Catching $d[i,j]$ 表示到第 i 分钟，移动了 j 次后获得的的最多苹果数。显然根据 j 的奇偶性可以直接知道 **bessie** 所在的苹果树。初值： $d[0,0] = 0$ 。其他也都为 0。目标： $\max\{d[t,i]\}, 0 \leq i \leq w$ 。转移方程：每一阶段都有两种选择，一种是移动，一种是不移动。如果要判断 i 时刻移动了 j 次后是否能接到苹果，只要用 $(j \& 1) == (a[i] - 1)$ 即可。即 $d[i,j] = \max\{d[i-1,j-1], d[i-1,j] + (j \& 1) == (a[i] - 1)\}$ 代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
 $d[i,j]$ 表示到第  $i$  分钟，移动了  $j$  次后获得的的最多苹果数。显然根据  $j$  的奇偶性可以直接知道 bessie 所在的苹果树。
初值： $d[0,0]=0$ 。其他也都为 0。

```

目标: $\max\{d[t,i]\}, 0 \leq i \leq w$ 。

转移方程: 每一阶段都有两种选择, 一种是移动, 一种是不移动。如果要判断 i 时刻移动了 j 次后是否能接到苹果, 只要用 $(j \& 1) == (a[i] - 1)$ 即可。

即 $d[i,j] = \max\{d[i-1,j-1], d[i-1,j] + ((j \& 1) == (a[i] - 1))\}$

```
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxt=1000+5;
const int maxw=30+5;
const int INF=0x3f3f3f3f;
int d[maxt][maxw],t,w,a[maxt],ans;
int main() {
    ios::sync_with_stdio(false);
    // freopen("Apple Catching.in","r",stdin);
    cin>>t>>w;
    for(int i=1;i<=t;i++) cin>>a[i],a[i]--;
    for(int i=1;i<=t;i++){
        for(int j=0;j<=w;j++){
            if(j>=1) d[i][j]=max(d[i-1][j-1],d[i-1][j])+((j&1)==a
[i]);
            else d[i][j]=d[i-1][j]+((j&1)==a[i]);
        }
    }
    for(int i=0;i<=w;i++) ans=max(ans,d[t][i]);
    cout<<ans;
    return 0;
}
```

```

#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 3616: Milking Time](#) 法一 对于每个区间 $[l, r]$ 建边 $l \rightarrow r+R$ 。然后就在相邻的区间间建边，边权为 0，保证顶点编号递增即可。为了便于计算，额外建边 $0 \rightarrow$ 第一个区间左端点和 最后一个区间右端点 $\rightarrow n+r$ 。起点为 0，终点 $n+r$ ，跑最长路即可。法二 将区间按照左端点升序排序，就是一个 LIS 问题。代码如下

```

/*

*/
#define method_2
#ifdef method_1
/*
对于每个区间 $[l, r]$ 建边  $l \rightarrow r+R$ 。然后就在相邻的区间间建边，边权为 0，保证顶点
编号递增即可。
为了便于计算，额外建边  $0 \rightarrow$  第一个区间左端点和 最后一个区间右端点  $\rightarrow n+r$ 。
起点为 0，终点  $n+r$ ，跑最长路即可。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "

```

```

#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000000*2+5;
const int maxm=1000+5;
const int INF=0x3f3f3f3f;
int n,m,r,vis[maxn],d[maxn],head[maxn],tot=1;
struct node{
    int from,to,c;
}edge[maxm<<2];
void add(int from,int to,int c){
    edge[++tot].from=head[from],head[from]=tot,edge[tot].c=c,edge[tot].to=to;
}
queue<int>q;
void spfa(){
    // memset(d,INF,sizeof(d));
    memset(vis,0,sizeof(vis));
    d[0]=0;q.push(0);vis[0]=1;
    while(q.size()){
        int x=q.front();q.pop();vis[x]=0;
        for(int i=head[x];i;i=edge[i].from){
            int y=edge[i].to,c=edge[i].c;
            if(d[x]+c>=d[y]){
                d[y]=d[x]+c;
                if(!vis[y]){
                    q.push(y);
                    vis[y]=1;
                }
            }
        }
    }
}
// for(int i=0;i<=n+r;i++) D(d[i]);
}
int main() {
    ios::sync_with_stdio(false);
    //freopen("Milking Time.in","r",stdin);
    cin>>n>>m>>r;
    int u,v,c;
    for(int i=1;i<=m;i++){
        cin>>u>>v>>c;add(u,v+r,c);vis[u]=vis[v+r]=1;
    }
    u=1,v=0;

```

```

        while(1){
            while(!vis[u]&&u<n+r) u++;
            add(v,u,0);
            if(u>=n+r) break;
            v=u,u++;
        }
        spfa();
        cout<<d[n+r];
        return 0;
    }
#endif
#ifdef method_2
/*
将区间按照左端点升序排序，就是一个 LIS 问题。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000000*2+5;
const int maxm=1000+5;
const int INF=0x3f3f3f3f;
int n,m,r,d[maxm],ans=0;
struct node{
    int s,t,c;
    bool operator<(const node &h)const{return s<h.s;}
}seg[maxm];
int main() {
    ios::sync_with_stdio(false);

```

```

//freopen("Milking Time.in","r",stdin);
cin>>n>>m>>r;
int u,v,c;
for(int i=1;i<=m;i++){
    cin>>seg[i].s>>seg[i].t>>seg[i].c;
}
sort(seg+1,seg+m+1);
// for(int i=1;i<=m;i++) d[i]=seg[i].c;
for(int i=1;i<=m;i++){
    d[i]=seg[i].c;
    for(int j=1;j<=i-1;j++){
        if(seg[j].t+r<=seg[i].s) d[i]=max(d[i],d[j]+seg[i].c);
    }
    ans=max(ans,d[i]);
}
cout<<ans;
return 0;
}
#endif
#ifdef method_3
/*

*/

#endif

```

POJ 3280: Cheapest Palindrome $d[i,j]$ 表示将 $[i,j]$ 变成回文串的最小代价。初值： $d[i,i]=0$ ，其他为 INF 。目标： $d[1,m]$ 。转移方程：1，若 $a[i]==a[j]$ ， $d[i,j]=d[i+1,j-1]$ 2，若 $a[i]!=a[j]$ ， $d[i,j]=\min\{d[i+1,j]+in[a[i]],d[i+1,j]+del[a[i]],d[i,j-1]+in[a[j]],d[i,j-1]+del[a[j]]\}$ 代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
 $d[i,j]$ 表示将 $[i,j]$ 变成回文串的最小代价。
初值： $d[i,i]=0$ ，其他为  $INF$ 。
目标： $d[1,m]$ 。
转移方程：
1，若  $a[i]==a[j]$ ， $d[i,j]=d[i+1,j-1]$ 
2，若  $a[i]!=a[j]$ ， $d[i,j]=\min\{d[i+1,j]+in[a[i]],d[i+1,j]+del[a[i]],d[i,j-1]+in[a[j]],d[i,j-1]+del[a[j]]\}$ 

```

```

*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=26+5;
const int maxm=2000+5;
const int INF=0x3f3f3f3f;
int n,m,d[maxm][maxm];
string a;
map<char,int>in,del;
void dp(){
    memset(d,INF,sizeof(d));
    for(int i=1;i<=m;i++) d[i][i]=d[i][i-1]=0;
    for(int len=2;len<=m;len++){
        for(int i=1;i<=m-len+1;i++){
            int j=i+len-1;
            // D(i);D(j);E;
            if(a[i]==a[j]) d[i][j]=d[i+1][j-1];
            else{
                int temp1=min(in[a[i]],del[a[i]]);
                d[i][j]=min(d[i+1][j]+temp1,d[i][j]);
                int temp2=min(in[a[j]],del[a[j]]);
                d[i][j]=min(d[i][j-1]+temp2,d[i][j]);
            }
        }
    }
}
int main() {

```

```

ios::sync_with_stdio(false);
//freopen("Cheapest Palindrome.in","r",stdin);
cin>>n>>m;
cin>>a;a.insert(0,"0");
char ch;int c1,c2;
for(int i=1;i<=n;i++){
    cin>>ch>>c1>>c2;
    in[ch]=c1,del[ch]=c2;
}
dp();
/*
for(int i=1;i<=m;i++){
    for(int j=i;j<=m;j++){
        D(d[i][j]);
    }
    E;
}
*/
cout<<d[1][m];
return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

优化递推式

POJ 1742: Coins

$f[i]$ 表示组成 i 元是否可行。

初值: $f[0]=1$

目标: 所有 $f[i]=1$ 的个数

转移方程:

设 $used[j]$ 表示 $F[j]$ 在阶段 i 时为 1 至少需要用多少枚第 i 种硬币。

若 $f[j-A[i]]=1$, 且 $used[j-A[i]]<C[i]$, 则 $used[j]=used[j-A[i]]+1$,

$f[j]=1$ 。

代码如下

```
/*
```

```
*/
```

```
#define method_1
```

```
#ifdef method_1
```

```
/*
```

$f[i]$ 表示组成 i 元是否可行。

初值: $f[0]=1$

目标: 所有 $f[i]=1$ 的个数

转移方程:

设 $used[j]$ 表示 $F[j]$ 在阶段 i 时为 1 至少需要用多少枚第 i 种硬币。

若 $f[j-A[i]]=1$, 且 $used[j-A[i]]<C[i]$, 则 $used[j]=used[j-A[i]]+1$,

$f[j]=1$ 。

```
*/
```

```
#include<iostream>
```

```
#include<cstdio>
```

```
#include<algorithm>
```

```
#include<cmath>
```

```
#include<set>
```

```
#include<map>
```

```
#include<queue>
```

```
#include<stack>
```

```
#include<vector>
```

```
#include<cstring>
```

```
#include<cstdlib>
```

```
#include<iomanip>
```

```
#include<ctime>
```

```
#include<string>
```

```
#define D(x) cout<<#x<<" = "<<x<<" "
```

```
#define E cout<<endl
```

```
using namespace std;
```

```
typedef long long ll;
```

```
typedef pair<int,int>pii;
```

```
const int maxn=100+5;
```

```
const int maxm=100000+5;
```

```
const int INF=0x3f3f3f3f;
```

```
int f[maxm],used[maxm],n,m,A[maxn],C[maxn],ans;
```

```
void init(){
```

```

        memset(f,0,sizeof(f));f[0]=1;ans=0;
    }
    int main() {
        ios::sync_with_stdio(false);
        //freopen("Coins.in","r",stdin);
        while(cin>>n>>m){
            init();
            if(n==0&&m==0) break;
            for(int i=1;i<=n;i++) cin>>A[i];
            for(int i=1;i<=n;i++) cin>>C[i];
            for(int i=1;i<=n;i++){
                memset(used,0,sizeof(used));
                for(int j=1;j<=m;j++){
                    if(!f[j]&&j-A[i]>=0){
                        if(f[j-A[i]]==1&&used[j-A[i]]<C[i]){
                            used[j]=used[j-A[i]]+1,f[j]=1;
                        }
                    }
                }
            }
            for(int i=1;i<=m;i++) if(f[i]) ans++;
            cout<<ans<<endl;
        }
        return 0;
    }
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 3046: Ant Counting](#)

```

/*

*/
#define method_2
#ifdef method_1
/*
多重集组合数。
f[i,j]表示前 i 种蚂蚁，选出 j 个的方案数。直接跑多重背包+滚动数组。
600msAC。
转移方程:  $f[i,j]=\sum f[i-1,j-k], 0\leq k\leq \min(j, \text{num}[i])$ 
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000+5;
const int maxa=100000+5;
const int INF=0x3f3f3f3f;
const int mod=1e6;
int t,a,s,b,num[maxn],f[2][maxa],tot=0;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Ant Counting.in","r",stdin);
    cin>>t>>a>>s>>b;
    int temp;

```

```

    for(int i=1;i<=a;i++){
        cin>>temp;num[temp]++;
    }
    f[0][0]=1;
    for(int i=1;i<=t;i++){
        memset(f[i&1],0,sizeof(f[i&1])); //滚动数组这里要先清空
        tot+=num[i];
        for(int j=0;j<=tot;j++){ //卡常数方式 这里将 tot 改成 a 就会 TLE
            for(int k=0;k<=min(j,num[i]);k++){
                f[i&1][j]=(f[i&1][j]+f[i-1&1][j-k])%mod;
            }
        }
    }
    int ans=0;
    for(int i=s;i<=b;i++) ans=(ans+f[t&1][i])%mod;
    cout<<ans;
    return 0;
}
#endif
#ifdef method_2
/*
对 method_1 的转移方程优化。
 $f[i,j]=f[i,j-1]+f[i-1,j]-f[i-1,j-1-num[i]]$ ，同时注意出现负数的情况即可。
100msAC。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>

```

```

#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000+5;
const int maxa=100000+5;
const int INF=0x3f3f3f3f;
const int mod=1e6;
int t,a,s,b,num[maxn],f[2][maxa];
int main() {
    ios::sync_with_stdio(false);
    //freopen("Ant Counting.in","r",stdin);
    cin>>t>>a>>s>>b;
    int temp;
    for(int i=1;i<=a;i++){
        cin>>temp;num[temp]++;
    }
    f[0][0]=f[1][0]=1; //这个转移方程中 f[1,1]可能来自 f[0,0],f[0,1](一定为0),f[1,0]
    for(int i=1;i<=t;i++){
        // memset(f[i&1],0,sizeof(f[i&1])); //同理 滚动数组这里不要清空
        for(int j=1;j<=a;j++){
            if(j-1-num[i]>=0) f[i&1][j]=f[i&1][j-1]-f[i-1&1][j-1-num[i]]+f[i-1&1][j];
            else f[i&1][j]=f[i-1&1][j]+f[i&1][j-1];
            f[i&1][j]=(f[i&1][j]+mod)%mod; //防止负数
        }
    }
    int ans=0;
    for(int i=s;i<=b;i++) ans=(ans+f[t&1][i])%mod;
    cout<<ans;
    return 0;
}
#endif
#ifdef method_3
/*

*/

```

#endif

[POJ 3181: Dollar Dayz](#)

完全背包即可。

但要高精度。

method_1 结果为 WA。

代码如下

```
/*

*/

#define method_2
#ifdef method_1
/*
要高精度。
method_1 结果为 WA。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000+5;
const int maxk=100+5;
const int INF=0x3f3f3f3f;
int d[maxn],n,k;
int main() {
```

```

        ios::sync_with_stdio(false);
        //freopen("Dollar Dayz.in","r",stdin);
        cin>>n>>k;
        d[0]=1;
        for(int i=1;i<=k;i++) for(int j=i;j<=n;j++) d[j]+=d[j-i];
        cout<<d[n];
        return 0;
    }
#endif
#ifdef method_2
/*

*/

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000+5;
const int maxp=100+5;
const int INF=0x3f3f3f3f;
int n,k;
struct bigint{
    int a[maxp],n;
    int &operator[](int x){return a[x];}
    bigint(){for(int i=0;i<maxp-5;i++)a[i]=0;n=0;}

```

```

void print(){for(int i=n;i>=1;i--)printf("%d",a[i]);}
friend bigint operator+(bigint x,bigint y){
    bigint res;res.n=x.n>y.n?x.n:y.n; int i;
    for(i=1;i<=res.n;i++)res[i]=x[i]+y[i];
    for(i=1;i<=res.n;i++)res[i+1]+=res[i]/10,res[i]%=10;
    if(res[res.n+1])res.n++; return res;
}
}d[maxn];
int main() {
    ios::sync_with_stdio(false);
    //freopen("Dollar Dayz.in","r",stdin);
    cin>>n>>k;
    /*
    d[0]=1;
    for(int i=1;i<=k;i++) for(int j=i;j<=n;j++) d[j]+=d[j-i];
    cout<<d[n];
    */
    d[0].n=1,d[0].a[1]=1;
    for(int i=1;i<=k;i++) for(int j=i;j<=n;j++) d[j]=d[j]+d[j-i];
    d[n].print();
    return 0;
}
#endif
#ifdef method_3
/*

*/

#endif

```

进阶 DP

[POJ 1065: Wooden Sticks](#)

Dilworth 定理

<https://blog.csdn.net/litble/article/details/85305561>

因此本题就是按 l 升序排序，若 l 相同则按 w 升序排序。然后对 w 求最长严格下降子序列($w_1 \leq w_2$ 的反面是 $w_1 > w_2$)的长度

代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
Dilworth 定理
https://blog.csdn.net/litble/article/details/85305561
因此本题就是按 l 升序排序，若 l 相同则按 w 升序排序。然后对 w 求最长严格下降子
序列( $w_1 \leq w_2$  的反面是  $w_1 > w_2$ )的长度
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=5000+5;
const int INF=0x3f3f3f3f;
int T,n,d[maxn],ans;
struct node{
    int l,w;
    bool operator<(const node& h)const{return
(l<h.l)|| (l==h.l&&w<h.w);}
}seg[maxn];
void init(int n){
    for(int i=1;i<=n;i++) d[i]=1;
    ans=0;
}
int main() {
    ios::sync_with_stdio(false);

```

```

//freopen("Wooden Sticks.in","r",stdin);
cin>>T;
while(T--){
    cin>>n;
    init(n);
    for(int i=1;i<=n;i++) cin>>seg[i].l>>seg[i].w;
    sort(seg+1,seg+n+1);
    for(int i=1;i<=n;i++) for(int j=1;j<i;j++)
if(seg[i].w<seg[j].w) d[i]=max(d[i],d[j]+1);
    for(int i=1;i<=n;i++) ans=max(ans,d[i]);
    cout<<ans<<endl;
//    cout<<d[n]<<endl; 这里不是 d[n]啊
}
return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 1631: Bridging signals](#) 和 Wooden Sticks 原理类似，不过由于数据量较大，所以 dp 时要用 $n\log n$ 的算法。代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
和 Wooden Sticks 原理一样，不过由于数据量较大，所以 dp 时要用  $n\log n$  的算法。

*/
#include<cstdio>
#include<iostream>
#include<algorithm>
const int maxn=40000+10;

```

```

int n,T;
int d[maxn]; //d[i]表示 1~i 中最长的不下降子序列的长度
struct node {
    int a,b;
} qj[maxn];
bool operator<(const node &a1,const node &a2) {
    if(a1.a!=a2.a) return a1.a<a2.a;
    return a1.b<a2.b;
}
int main() {
    //freopen("Bridging signals.in","r",stdin);
    scanf("%d",&T);
    while(T--) {
        scanf("%d",&n);
        int x,y;
        for(int i=1; i<=n; i++) {
            d[i]=1;
            scanf("%d",&y);
            qj[i].a=i;
            qj[i].b=y;
        }
        std::sort(qj+1,qj+n+1);
// for(int i=1;i<=n;i++){
//     std::cout<<qj[i].b<<" ";
//     printf("%d ",qj[i].b);
// }
// std::cout<<std::endl;
// printf("\n");
        int ans=1;
        d[1]=qj[1].b; //转移方程是单调的 但是由于区间不连续 所以不能用单
        调队列
        for(int i=2; i<=n; i++) {
            // begin=qj[i].a;
            // end=qj[i].b;
            int wz=std::lower_bound(d+1,d+ans+1,qj[i].b)-d; //二分找
            到插入位置
            d[wz]=qj[i].b;
            if(wz>ans) { //插在了最后 说明答案可以更新
                ans++;
            }
//     d[i]=std::max(temp+1,d[i]);
// }
// for(int i=1;i<=n;i++){
//     std::cout<<qj[i].b<<" ";

```

```

//      printf("%d ",d[i]);
//      ans=std::max(d[i],ans);
//  }
//  std::cout<<std::endl;
//  printf("\n");
//  std::cout<<d[n];
//      printf("%d\n",ans);
//  }
//  return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 3666: Making the Grade](https://www.jianshu.com/p/2c07da54278b) 题解链接

<https://www.jianshu.com/p/2c07da54278b> 代码如下

```

/*
https://www.jianshu.com/p/2c07da54278b
*/
#define method_2
#ifdef method_1
/*

*/

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>

```

```

#include<cstdlib>
#include<iomanip>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=2000+5;
const int INF=0x3f3f3f3f;
int n,a[maxn],b[maxn],f[maxn][maxn],cnt;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Making the Grade.in","r",stdin);
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        b[i]=a[i];
    }
    sort(b+1,b+n+1);
    cnt=unique(b+1,b+n+1)-b-1;
    /*
    for(int i=1;i<=n;i++){
        a[i]=lower_bound(b+1,b+cnt+1,a[i])-b;
    }
    */
    int ans=INF;
    memset(f,INF,sizeof(f));
    for(int i=1;i<=cnt;i++) f[0][i]=0;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=cnt;j++){
            for(int k=1;k<=j;k++){
                f[i][j]=min(f[i-1][k]+abs(a[i]-b[j]),f[i][j]);
            }
            if(i==n) ans=min(ans,f[n][j]);
        }
    }
    cout<<ans;
    return 0;
}
#endif
#ifdef method_2
/*

*/

```

```

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=2000+5;
const int INF=0x3f3f3f3f;
int n,a[maxn],b[maxn],f[maxn][maxn],cnt;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Making the Grade.in","r",stdin);
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        b[i]=a[i];
    }
    sort(b+1,b+n+1);
    cnt=unique(b+1,b+n+1)-b-1;
    /*
    for(int i=1;i<=n;i++){
        a[i]=lower_bound(b+1,b+cnt+1,a[i])-b;
    }
    */
    int ans=INF;
    memset(f,INF,sizeof(f));
    for(int i=0;i<=cnt;i++) f[0][i]=0;
    for(int i=1;i<=n;i++){
        int val=f[i-1][0];
        for(int j=1;j<=cnt;j++){
            val=min(val,f[i-1][j]);
            f[i][j]=val+abs(a[i]-b[j]);
            if(i==n) ans=min(ans,f[n][j]);
        }
    }
}

```

```

    }
}
cout<<ans;
return 0;
}
#endif
#ifdef method_3
/*

*/

#endif

```

POJ 2392: Space Elevator 这道题因为每个点的限制不一样，所以需要从限制小的先进行 **dp**（按照 **a[i]** 升序排序后 **dp**）。因为限制大的可以放在限制小的上面，但是限制小的无法再往大的上面堆。**f[i,j]** 表示使用前 **i** 个 **block**，能否达到高度 **j**。初值： $f[0,0] = 1$ 目标：条件： $f[n,i] = 1$ ，输出最大的满足条件的 **i** 转移方程： $f[i,j] = f[i-1, j-k*h[i]]$ $j \leq a[i]$, $0 \leq k \leq c[i]$ 这个多重背包在实现时，可以省略 **f** 数组的第一维。同时由于 **c[i]** 范围很小，所以不用二进制或者单调队列优化。代码如下

```

/*

*/

#define method_1
#ifdef method_1
/*
这道题因为每个点的限制不一样，所以需要从限制小的先进行 dp（按照 a[i] 升序排序后 dp）。
因为限制大的可以放在限制小的上面，但是限制小的无法再往大的上面堆。
f[i,j] 表示使用前 i 个 block，能否达到高度 j。
初值： $f[0,0]=1$ 
目标：条件： $f[n,i]=1$ ，输出最大的满足条件的 i
转移方程： $f[i,j]=f[i-1,j-k*h[i]]$   $j \leq a[i]$ ,  $0 \leq k \leq c[i]$ 
这个多重背包在实现时，可以省略 f 数组的第一维。同时由于 c[i] 范围很小，所以不用二进制或者单调队列优化。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>

```

```

#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxk=400+5;
const int maxai=4000+5;
const int INF=0x3f3f3f3f;
int f[maxai],h[maxk],a[maxk],c[maxk],k;
struct node{
    int h,a,c;
    bool operator<(const node& h)const{return a<h.a;}
}block[maxk];
int main() {
    ios::sync_with_stdio(false);
    //freopen("Space Elevator.in","r",stdin);
    cin>>k;
    for(int i=1;i<=k;i++) cin>>block[i].h>>block[i].a>>block[i].c;
    sort(block+1,block+k+1);
    f[0]=1;
    for(int i=1;i<=k;i++){
        for(int j=1;j<=block[i].c;j++){ //循环数量放在第二重循环里头
            for(int l=block[i].a;l<=block[i].h;l++){
                f[l]=f[l-block[i].h]; //注意这里不是 f[l-j*block[i].
h]
            }
        }
    }
    for(int i=maxai-5;i>=0;i--){
        if(f[i]){
            cout<<i;
            return 0;
        }
    }
    return 0;
}
#endif

```

```

#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 2184: Cow Exhibition 01 背包。 $d[i,j]$ 表示前 i 个奶牛，TS 为 j 时最大的 TF。（第一维可省略，第二维需要数组下标平移）如果 S 是负值的话会导致重复计算，所以此时需要调转循环方向。**PS：**这题深搜可过。只需要加两个剪枝即可。剪枝一： s 和 f 都大于等于 0 必选。剪枝二： s 和 f 都小于 0 必不选。代码如下

```

/*
PS：这题深搜可过。只需要加两个剪枝即可。
剪枝一： $s$  和  $f$  都大于等于 0 必选。
剪枝一： $s$  和  $f$  都小于 0 必不选。
*/
#define method_1
#ifdef method_1
/*
01 背包。
 $d[i,j]$  表示前  $i$  个奶牛，TS 为  $j$  时最大的 TF。（第一维可省略，第二维需要数组下标平移）
如果  $S$  是负值的话会导致重复计算，所以此时需要调转循环方向。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>

```

```

#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=100+5;
const int base=100*1000+5;
const int INF=0x3f3f3f3f;
int n,s[maxn],f[maxn],d[base*2+1005],ans=0;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Cow Exhibition.in","r",stdin);
    cin>>n;
    for(int i=1;i<=n;i++) cin>>s[i]>>f[i];
    memset(d,-INF,sizeof(d));d[base]=0;
    for(int i=1;i<=n;i++){
        if(s[i]>=0){
            //本来应该在[-base+s[i],base]之间 dp, 坐标平移后就变成了[s
[i],base]
            for(int j=base*2;j>=s[i];j--) d[j]=max(d[j],d[j-s[i]]+f
[i]);
        }
        else{
            for(int j=s[i];j<=base*2+s[i];j++) //注意这里不是 for(int
j=s[i];j<=base*2;j++) 这样 j-s[i]就会到达不合法的区域
            d[j]=max(d[j],d[j-s[i]]+f[i]);
        }
    }
    for(int i=base;i<=base*2;i++){ //只有[base,base*2]是合法的
        if(d[i]>=0) ans=max(ans,d[i]+i-base);
    }
    cout<<ans;
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

```

*/

#endif