动态规划题解

A- Boxer (Codeforces-1203E)

贪心即可，先对数组排序，从大到小遍历，把当前数尽可能的往+1的去放（给之前的数腾空间）。用vis数组维护一下某个数是否被占用。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>

using namespace std;

const int maxn = 2e5 + 5;

int a[maxn];

bool vis[maxn];

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", a + i);
    }
    sort(a,a+n);
    int cnt = 0;
    for(int i=n-1;i>=0;--i){
        if(a[i]==1){
            if(!a[2]){
                vis[2]=1;
                cnt++;
            }
            else if(!a[1]){
                cnt++;break;
            }
            else break;
        }
        else{
            if(!vis[a[i]+1]){
                vis[a[i]+1]=1;
                cnt++;
            }
            else if(!vis[a[i]]){
                vis[a[i]]=1;
                cnt++;
            }
            else if(!vis[a[i]-1]){
                vis[a[i]-1]=1;
                cnt++;
```

```
            }
        }
    }
    printf("%d", cnt);
    return 0;
}
```

B- Gas Pipeline (Codeforces-1207C)

普通DP，第一维是第i段及之前的最小消费，第二维代表当前段是状态0还是状态1，当前段为状态1时，要从之前的状态1转移，当前段为状态0时，它的状态0从之前的状态1或者状态0转移，它的状态1也可以从状态1或者状态0转移。注意一开始需要把dp数组memset成一个大值。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<set>
#include<queue>
#include<stack>
#include<vector>
using namespace std;

const int maxn=2e5+5;

typedef long long ll;

const int inf=0x3f3f3f3f;

char s[maxn];

ll dp[maxn][2];

int main()
{
    int t;
    scanf("%d",&t);
    while(t--){
        int n,a,b;
        scanf("%d%d%d",&n,&a,&b);
        scanf("%s",s);
        memset(dp,inf,sizeof(dp));
        dp[0][0]=b;
        for(int i=0;i<n;i++){
            if(s[i]=='0'){
                dp[i+1][0]=min(dp[i+1][0],dp[i][0]+a+b);
                dp[i+1][1]=min(dp[i+1][1],dp[i][0]+2*(a+b));

                dp[i+1][1]=min(dp[i+1][1],dp[i][1]+a+2*b);
                dp[i+1][0]=min(dp[i+1][0],dp[i][1]+2*a+b);
            }
            else{
                dp[i+1][1]=min(dp[i+1][1],dp[i][1]+a+2*b);
            }
```

```
        }
        cout<<dp[n][0]<<endl;
    }
    return 0;
}
```

C- Super Jumping! Jumping! Jumping! (HDU-1087)

LIS变形题，最长上升子序列的和，dp[i]表示当跳到i点时，它可以获得的和的最大值是多少?

两重循环：外层遍历所有的i，内层遍历0到i-1所有点j。如果a[j]比a[i]小，那就把dp[j]+a[i]来更新dp[i];最后再把a[i]和dp[i]更新一下哪个更大，并用dp[i]更新ans值。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include<stack>
#define INF 1E9
using namespace std;

const int inf=0x3f3f3f3f;
long long dp[1010];
long long a[1010];
int main()
{
    int n;
    while(scanf("%d",&n)&&n!=0){
        memset(dp,0,sizeof(dp));
        for(int i=0;i<n;i++){
            scanf("%d",a+i);
        }

        dp[0]=a[0];
        long long maxx=dp[0];
        for(int i=1;i<n;i++){
            for(int j=0;j<i;j++){
                if(a[j]<a[i]){
                    dp[i]=max(dp[i],dp[j]+a[i]);
                }
            }
            dp[i]=max(dp[i],a[i]);
            maxx=max(maxx,dp[i]);
        }

        cout<<maxx<<endl;
    }
    return 0;
}
```

D -Bone Collector (HDU-2602)

01背包裸题，课件有题解

E -免费馅饼 （HDU-1176)

数字三角形变形题，差别只是起点变成了5，可以往左右1步和原地三个状态去走，第一维设置为在i点，第二维设置为时间为j时的状态，用数字三角形的解法就能过题。注意处理边界（不要走出了范围）

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<set>
using namespace std;

const int mod=1e9+7;

const int maxn=1e5+5;
int dp[11][maxn];

int m[11][maxn];

int main()
{
    int n;
    while(scanf("%d",&n)){
        if(n==0) break;
        memset(dp,0,sizeof(dp));
        memset(m,0,sizeof(m));
        int maxt=0;
        for(int i=0;i<n;i++){
            int x,t;
            scanf("%d%d",&x,&t);
            m[x][t]++;
            maxt=max(t,maxt);
        }

        for(int j=maxt;j>=0;j--){
            for(int i=0;i<=10;i++){
                if(i==0) dp[i][j]=max(dp[i][j],m[i][j]+max(dp[i][j+1],dp[i+1]
[j+1]));
                else if(i==10) dp[i][j]=max(dp[i][j],m[i][j]+max(dp[i]
[j+1],dp[i-1][j+1]));
                else dp[i][j]=max(dp[i][j],m[i][j]+max(dp[i][j+1],max(dp[i+1]
[j+1],dp[i-1][j+1])));
            }
        }
        printf("%d\n",dp[5][0]);
    }
    return 0;
}
```

F -Piggy-Bank (HDU-1114)

完全背包裸题，课件有题解，注意输出即可

G -Ayoub and Lost Array （Codeforces-1105C）

题意：给你l-r区间，选出n个数使它们加起来的和能整除3，求有多少种情况，对1e9+7取模。

由于对3整除，所以先预处理算出l-r区间有多少个数取模是0，取模是1，取模是2;之后用DP，i表示前面i个数，j表示前i个数加起来的和对3整除是多少，dp[i][j]存放当前情况有多少种答案。由于当前的状态可以从之前的状态得出，例如要想得出当前取模是0的情况，可以从之前取模为0的情况加上一个取模为0的数得到，也可以从之前取模为1的情况加上一个取模为2数，也可以从之前取模为2的情况加上一个取模为1的数，最后输出dp[n][0]即可。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<set>
#include<queue>
#include<stack>
#include<vector>
#include<map>
using namespace std;

const int maxn=2e5+5;

const int mod=1e9+7;

typedef long long ll;

const int inf=0x3f3f3f3f;

ll dp[maxn][3];

int main()
{
    int n,l,r;
    while(~scanf("%d%d%d",&n,&l,&r)){
        memset(dp,0,sizeof(dp));
        int nn[3];
        memset(nn,0,sizeof(nn));
        int len=r-l+1;
        int che=len/3;
        for(int i=0;i<3;i++){
            nn[i]+=che;
        }
        int y=len%3;
        int num=l;
        for(int i=0;i<y;i++){
            nn[num%3]++;
            num++;
        }
        dp[1][0]=nn[0];dp[1][1]=nn[1];dp[1][2]=nn[2];
        for(int i=2;i<=n;i++){
            dp[i][0]=(dp[i-1][2]*nn[1]+dp[i-1][1]*nn[2]+dp[i-1][0]*nn[0])%mod;
            dp[i][1]=(dp[i-1][1]*nn[0]+dp[i-1][2]*nn[2]+dp[i-1][0]*nn[1])%mod;
            dp[i][2]=(dp[i-1][2]*nn[0]+dp[i-1][1]*nn[1]+dp[i-1][0]*nn[2])%mod;
        }
```

```
        printf("%lld\n",dp[n][0]%mod);
    }
    return 0;
}
```

H -Common Subsequence (POJ-1458)

最长公共子序列模板题。i表示第一个串的第i个位置，j表示第二个串的第j个位置,dp[i][j]表示在第一个串的1-i子串，和第二个串的1-j的子串的最长公共子序列是多少。如果a[i]==b[j]，那么就从dp[i-1][j-1]处加1转移到dp[i][j]，

否则就从dp[i-1][j]和dp[i][j-1]中的较大值转移。（要知道之前的状态到底是什么状态，为什么是这样转移的）

给一个blog好好领悟一下LCS到底为什么是这样：https://blog.csdn.net/weixin_40673608/article/details/84262695

博客其实有很多，关键要自己弄懂为什么要从那个位置转移。

```
#include<iostream>
#include<stdio.h>
#include<cstring>
#include<string>
using namespace std;

const int MAX = 1005;

int DP[MAX][MAX];

int main()
{
    string a;
    string b;
    while(cin >> a >> b)
    {
        int l1 = a.size();
        int l2 = b.size();
        memset(DP, 0, sizeof(DP));
        for(int i = 1; i <= l1; i++)
            for(int j = 1; j <= l2; j++)
                if(a[i - 1] == b[j - 1])
                    DP[i][j] = DP[i-1][j-1]+1;
                else
                    DP[i][j] = max(DP[i][j - 1], DP[i - 1][j]);
        printf("%d\n", DP[l1][l2]);
    }
    return 0;
}
```

I -Lost it! (Codeforces-1176C)

思维，意思是找4、8、15、16、23、42这样一个子序列到底有多少个，然后把除了这些子序列的数都删除，问要删多少个数。

由于输入的肯定是上述6个数，所以把他们在这个子序列的下标存下来（例如输入一个4就存一个0在a[i]）中，

从左往右遍历一遍a[i]，假设遇到一个0，就使s[0]的数量加一，如果之后遇到一个1，那么把s[0]的数量--，代表这个0已经被使用了，然后把s[1]的数量++，往后亦然；一直转移过去，统计一下s[5]的数量，最后答案就是n-6*s[5]。

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<set>
#include<algorithm>
using namespace std;

const int maxn=5e5+5;

typedef long long ll;

int a[maxn];

const int b[6]={4,8,15,16,23,42};

int main()
{
    int n;
    while(~scanf("%d",&n)){
        for(int i=1;i<=n;i++){
            scanf("%d",a+i);
            a[i]=lower_bound(b,b+6,a[i])-b;
        }

        int s[6];
        memset(s,0,sizeof(s));
        for(int i=1;i<=n;i++){
            if(a[i]==0){
                s[0]++;
            }
            else{
                if(s[a[i]-1]>0){
                    s[a[i]-1]--;
                    s[a[i]]++;
                }
            }
        }

        printf("%d\n",n-6*s[5]);
    }
    return 0;
}
```

J -Invoker (2019-CCPC秦皇岛区域赛I题2019.9.21) HDU-6739

题解：暴力DP即可，dp[i][j]中,i为前i项,j为6种排序情况，从dp[i-1]中的6种情况转移即可。

```
#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5+5;

typedef long long ll;

char s[maxn];

string f[26];

const int xl[10][10]={{0,1,2},{0,2,1},{1,0,2},{1,2,0},{2,0,1},{2,1,0}};

int dp[maxn][10];

int main()
{
    f['Y'-'A']="QQQ";
    f['V'-'A']="QQW";
    f['G'-'A']="QQE";
    f['C'-'A']="WWW";
    f['X'-'A']="QWW";
    f['Z'-'A']="WWE";
    f['T'-'A']="EEE";
    f['F'-'A']="QEE";
    f['D'-'A']="WEE";
    f['B'-'A']="QWE";

    while(~scanf("%s",s)){
        int n=strlen(s);
        memset(dp,0x3f,sizeof(dp));
        for(int i=0;i<6;i++){
            dp[0][i]=3;
        }

        for(int i=1;i<n;i++){
            for(int j=0;j<6;j++){
                for(int k=0;k<6;k++){
                    int tmp=0;
                    if((f[s[i-1]-'A'][xl[j][0]]==f[s[i]-'A'][xl[k][0]])&&(f[s[i-
1]-'A'][xl[j][1]]==f[s[i]-'A'][xl[k][1]])&&(f[s[i-1]-'A'][xl[j][2]]==f[s[i]-'A']
[xl[k][2]])) tmp=3;
                    else if((f[s[i-1]-'A'][xl[j][1]]==f[s[i]-'A'][xl[k][0]])&&
(f[s[i-1]-'A'][xl[j][2]]==f[s[i]-'A'][xl[k][1]])){
                        tmp=2;
                    }
                    else if(f[s[i-1]-'A'][xl[j][2]]==f[s[i]-'A'][xl[k][0]]){
                        tmp=1;
                    }

                    dp[i][k]=min(dp[i][k],dp[i-1][j]+3-tmp);
                }
            }
        }
```

```
        int maxx=0x3f3f3f3f;
        for(int i=0;i<6;i++){
            maxx=min(maxx,dp[n-1][i]);
        }
        printf("%d\n",maxx+n);
    }
    return 0;
}
//XDTBVV
```

K -Pebbles (HDU-2167)

状压DP，见第5次比赛（位运算）题解。

```
#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5+5;

int dp[20][1<<15];

int a[20][20];

char s[maxn];

int top=0;

int state[1<<15];

void init(){
    top=0;
    for(int i=0;i<(1<<15);i++){
        if(i&i<<1)
            continue;
        state[top++]=i;
    }
    memset(a,0,sizeof(a));
}

int main(){
    init();
    while(~scanf("%d",&a[0][0])){
        int n=1;
        char c;
        scanf("%c",&c);
        while(c!='\n'){
            scanf("%d",&a[0][n++]);
            scanf("%c",&c);
        }

        for(int i=1;i<n;i++){
            for(int j=0;j<n;j++)scanf("%d",&a[i][j]);
        }

        memset(dp,0,sizeof(dp));
```

```
        for(int i=0;i<top&&state[i]<(1<<n);i++){
            for(int j=0;j<n;j++){
                if(state[i]&(1<<j)) dp[0][i]+=a[0][j];
            }
        }

        for(int i=1;i<n;i++){
            for(int j=0;j<top&&state[j]<(1<<n);j++){
                for(int k=0;k<top&&state[k]<(1<<n);k++){
                    if(state[j]&state[k]||(state[j]&(state[k]<<1))||((state[j]
<<1)&state[k])) continue;
                    int sum=0;
                    for(int m=0;m<n;m++){
                        if(state[k] & (1<<m)) sum+=a[i][m];
                    }
                    dp[i][k]=max(dp[i][k],dp[i-1][j]+sum);
                }
            }
        }

        int ans=0;
        for(int i=0;i<top&&state[i]<(1<<n);i++){
            ans=max(dp[n-1][i],ans);
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

L - By Elevator or Stairs? (Codeforces-1249E)

dp[i][j] i表示当前层,j表示用电梯还是用楼梯，每一层从之前的电梯或者楼梯之间的最优值来转移。

```
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=2e5+5;

int a[maxn];

int b[maxn];

int dp[maxn][2];

int main(){
    int n,c;
    scanf("%d%d",&n,&c);
    memset(dp,0x3f,sizeof(dp));
    for(int i=1;i<n;i++){
        scanf("%d",a+i);
    }

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

```
        scanf("%d",b+i);
    }

    dp[0][0]=0;dp[0][1]=0;
    printf("0 ");
    for(int i=1;i<n;i++){
        dp[i][0]=min(dp[i][0],dp[i-1][0]+a[i]);
        dp[i][0]=min(dp[i][0],dp[i-1][1]+a[i]);

        dp[i][1]=min(dp[i][1],dp[i-1][0]+b[i]+c);
        if(i!=1)
            dp[i][1]=min(dp[i][1],dp[i-1][1]+b[i]);
        else
            dp[i][1]=min(dp[i][1],dp[i-1][1]+b[i]+c);
        printf("%d ",min(dp[i][0],dp[i][1]));
    }
    return 0;
}
```

M -All-you-can-eat(Atcoder - 5276)

01背包变种，当剩余容量不足够放一个更大容量的东西时，可以把这个东西放进去让它填满背包。

用二维的01背包，dp1[i][j]维护1-i个物品中，容量为j时的最大价值；dp2[i][j]维护i-n个物品中，容量为j时的最大价值。

运行完两个dp之后，枚举1-n每个物品i，假设必取当前枚举的物品i，再枚举装前1~i-1的背包容量j，总容量必须少1用来放当前枚举的物品，那么最大值为dp1[i-1][j]+b[i]+dp2[i+1][t-j-1]。

```
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn=3e3+5;

const ll mod=1e9+7;

int dp1[maxn][maxn];
int dp2[maxn][maxn];

int a[maxn];
int b[maxn];

int main(){
    int n,t;
    scanf("%d%d",&n,&t);
    for(int i=1;i<=n;i++){
        scanf("%d%d",a+i,b+i);
    }

    for(int i=1;i<=n;i++){
        for(int j=0;j<=t;j++){
            dp1[i][j]=dp1[i-1][j];
        }
        for(int j=a[i];j<=t;j++){
```

```
            dp1[i][j]=max(dp1[i][j],dp1[i-1][j-a[i]]+b[i]);
        }
    }

    for(int i=n;i>=1;i--){
        for(int j=0;j<=t;j++){
            dp2[i][j]=dp2[i+1][j];
        }
        for(int j=a[i];j<=t;j++){
            dp2[i][j]=max(dp2[i][j],dp2[i+1][j-a[i]]+b[i]);
        }
    }

    int ans=0;
    for(int i=1;i<=n;i++){
        for(int j=0;j<t;j++){
            ans=max(ans,dp1[i-1][j]+dp2[i+1][t-1-j]+b[i]);
        }
    }
    printf("%d",ans);
    return 0;
}
```