第 13 章 动态规划进阶

13.1 LIS 和 LCS

LIS：Longest Increasing Subsequence，最长上升子序列
LCS：Longest Common Subsequence，最长公共子序列

1893：【提高】最长上升子序列 LIS（2）

解法一：求以 a[i]结尾的最长上升子序列的长度

a 数组：存放元素

1	10	4	5	3	12	6	2
1	2	3	4	5	6	7	8

dp 数组：存储状态，以 a[i]结尾的最长上升子序列的长度

1	2	2	3	2	4	4	2
1	2	3	4	5	6	7	8

dp[i]=1（边界条件）

状态转移方程：dp[i]=max(dp[j]+1,dp[i]) j 是 1..i-1 之间的数，a[j]<a[i]

问题：由于时间复杂度是 $O(n^2)$，如果 n=100000，时间会超限！

解法二：将原来的 dp 数组的存储以每个数结尾的 LIS 序列的长度，修改为存储上升子序列长度为 i 的上升子序列的最小末尾数值。

原理：LIS 长度如果已经确定，那么如果这种长度的子序列的结尾元素越小，后面可能续的元素会更多！

a 数组：存放元素

1	10	4	5	3	12	6	2
1	2	3	4	5	6	7	8

dp 数组：存储 LIS 长度为 i 的上升子序列的最小末尾数值。

1	2	5	6				
1	2	3	4	5	6	7	8

时间复杂度：$n*\log_2 n$

```
#include <bits/stdc++.h>
using namespace std;

//dp:长度为 i 的 LIS 的最后一位最小值是多少
int a[100100],dp[100100];
int i,n,l,r,mid;

int main(){
```

```
    //读入
    scanf("%d",&n);
    for(i = 1;i <= n;i++){
        scanf("%d",&a[i]);
    }

    //边界
    dp[1] = a[1];
    int len = 1;//LIS 的长度
    //从第 2 个数开始求解
    for(i = 2;i <= n;i++){
        //如果 a[i]比 dp 最后一位大,a[i]直接续上去,增加 LIS 的长度
        if(a[i] > dp[len]){
            len++;
            dp[len] = a[i];
        }else{
            //二分查找到 dp 数组中第 1 个>=a[i]的元素下标，替换（dp 数组一定是递增的）
            l = 1;
            r = len;
            while(l <= r){
                mid = (l + r) / 2;
                if(a[i] <= dp[mid]) r = mid - 1;
                else l = mid + 1;
            }

            //替换
            dp[l] = a[i];
        }
    }

    printf("%d",len);
}
```

1821:【基础】最长公共子序列（LCS）(1)

我们可以用 dp[i][j]来表示第一个串的前 i 位，第二个串的前 j 位的 LCS 的长度，那么递推出状态转移方程：

如果当前的 a [i]和 b[j]相同（即是有新的公共元素） 这说明该元素一定位于公共子序列中。因此，现在只需要找：a 数组 1~i-1 和 b 数组 1~j-1 的最长公共子序列：

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

如果不相同，说明最后一个元素肯定不是公共子序列中的元素，那么考虑找 a 数组 1~i-1 和 b 数组 1~j 的 LCS，或者找：a 数组的 1~i 和 b 数组的 1~j-1 的 LCS，那么，状态转义方程如下：

dp[i][j]=max(dp[i-1][j],dp[i][j-1])

a	1	2	3	4				
	1	2	3	4	5	6		
b	3	1	2	4				
	1	2	3	4	5	6		

dp：记录第一个串的前 i 位，第二个串的前 j 位的最长公共子序列的长度。

	1	2	3	4
1	0	1	1	1
2	0	1	2	2
3				
4				

a[i]==b[j],方程：dp[i-1][j-1]+1
a[i]!=b[j],方程：max(dp[i][j-1],dp[i-1][j])

```
#include <bits/stdc++.h>
using namespace std;

/*
a[i]==b[j],方程：dp[i-1][j-1]+1
a[i]!=b[j],方程：max(dp[i][j-1],dp[i-1][j])
*/
const int N = 1010;//常量，表示数组大小
int a[N],b[N],dp[N][N];
int n,i,j;

int main() {
    cin>>n;
    for(i = 1;i <= n;i++) cin>>a[i];
    for(i = 1;i <= n;i++) cin>>b[i];

    //递推
    for(i = 1;i <= n;i++){
        for(j = 1;j <= n;j++){
            if(a[i] == b[j]) dp[i][j] = dp[i-1][j-1] + 1;
            else dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
        }
    }

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

1822：【提高】最长公共子序列（LCS）(2)

解法一存在的问题：数组开不了这么大，即使数组能开这么大，时间也超限。

原理：

1、因为两个序列都是 1~n 的全排列，那么两个序列元素互异且相同，也就是说只是位置不同；

2、通过 c 数组将 b 序列的数字在 a 序列中的位置求出；

3、如果 b 序列每个元素在 a 序列中的位置递增，说明 b 中的这个数在 a 中的这个数整体位置偏后，可以考虑纳入 LCS；

4、从而就可以转变成求用来记录新的位置的 c 数组中的 LIS。

a	3	2	1	4	5			
	1	2	3	4	5	6		
b	2	1	4	3	5			

c[a[i]]=i

1 2 3 4 5 6

求出 b 数组的每个数在 a 数组中的位置!

c 数组：记录 a 数组的每个数的位置!

c	3	2	1	4	5			
	1	2	3	4	5	6		

b[i]在 a 中的位置：c[b[i]]

dp								
	1	2	3	4	5	6		

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100100;
int a[N],b[N],c[N],dp[N];
int n,i;

int main() {
    cin>>n;
    for(i = 1;i <= n;i++){
        scanf("%d",&a[i]);
        c[a[i]] = i;//求出 a 数组的每个数的位置
    }

    for(i = 1;i <= n;i++) scanf("%d",&b[i]);

    //求 b 数组的每个数在 a 数组的位置(c[b[i]])的 LIS
    dp[1] = c[b[1]];//边界
    int len = 1;
    int l,r,mid;
    //从第 2 个数开始讨论
    for(i = 2;i <= n;i++){
        //增加 LIS 的长度
        if(c[b[i]] > dp[len]){
            len++;
            dp[len] = c[b[i]];
        }else{
            l = 1;
            r = len;
            while(l <= r){
                mid = (l + r) / 2;
                if(c[b[i]] <= dp[mid]) r = mid - 1;
                else l = mid + 1;
            }

            dp[l] = c[b[i]];
        }
    }

    cout<<len;
    return 0;
}
```

13.2 常见各类背包问题

1282：【提高】简单背包问题（01 背包）

问题描述：有 n 件物品和容量为 m 的背包 给出 i 件物品的重量以及价值 求解让装入背包的物品重量不超过背包容量 且价值最大 。
特点:这是最简单的背包问题，特点是每个物品只有一件供你选择放还是不放。

1、二维解法
设 dp[i][j]表示前 i 件物品 总重量不超过 j 的最大价值 可得出状态转移方程
dp[i][j]=max{dp[i-1][j-w[i]]+v[i], dp[i-1][j]}
代码：
```
#include <bits/stdc++.h>
using namespace std;
/*
  动态转移方程:
  dp[i][j]=max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])
*/
//c:代表背包容量
//dp[i][j]:有 i 件物品，背包容量为 j 的情况下存储的最大价值
int c,dp[110][20100],w[110],v[110],i,j,n;

int main(){
     //读入
     cin>>c>>n;
     for(i = 1;i <= n;i++){
          cin>>w[i]>>v[i];
     }

     //递推求 dp 数组
     //i: 代表物品数量
     for(i = 1;i <= n;i++){
          //在 i 件物品，讨论背包容量分别是 1~c 的情况下，最大价值
          //j:代表背包容量
          for(j = 1;j <= c;j++){
               //如果能放得下
               if(w[i] <= j){
                    dp[i][j] = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]]);
               }else{
                    //放不下
                    dp[i][j] = dp[i-1][j];
               }
          }
     }

     //输出 n 件物品，背包容量为 c 的最大价值
     cout<<dp[n][c];
     return 0;
}
```

在一些情况下 题目的数据会很大 因此 dp 数组不开到一定程度是没有办法 ac。

2、一维解法
设 dp[j]表示重量不超过 j 公斤的最大价值 可得出状态转移方程
dp[j]=max{dp[j], dp[j-w[i]]+v[i]}

代码：
```
#include <bits/stdc++.h>
using namespace std;
```

```
/*
二维数组：dp[i][j]=max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])
一维数组滚动优化：
状态转移方程：dp[j]=max(dp[j],v[i]+dp[j-w[i]])
*/
int w;//背包容量
int dp[20010];
int n,wi,vi;

int main(){
    cin>>w>>n;
    //遍历每个物品
    for(int i = 1;i <= n;i++){
        cin>>wi>>vi;
        //倒过来循环
        for(int j = w;j >= wi;j--){
            //讨论每个物品选和不选的两个状态
            //取能到的价值的最大值
            dp[j] = max(dp[j],dp[j-wi] + vi);
        }
    }

    cout<<dp[w];
    return 0;
}
```

1780：【基础】疯狂的采药（完全背包）

1、完全背包状态转移方程：

f[i][j] = max(f[i-1][j], f[i-1][j-k*w[i]]+k*v[i])

(1 <= k <= w/w[i])

2、上面的式子可以变换为：（加上 k=0 的状态，也就是不取第 i 件物品）（步骤二）

f[i][j] = max(f[i-1][j-k*w[i]]+k*v[i])

(0 <= k <= w/w[i])

下面开始变形：

3、把 k=0 拿出来单独考虑，即比较在【不放第 i 种物品】、【放第 i 种物品 k 件(k>=1)中结果最大的那个 k】这两种情况下谁的结果更大

f[i][j] = max(f[i-1][j], max(f[i-1][j-k*w[i]]+k*v[i]))

(k >= 1)

4、考虑上式【放第 i 种物品】这种情况：放的话至少得放 1 件，先把这确定的 1 件放进去，即：在第 i 件物品已经放入 1 件的状态下再考虑放入 k(k>=0)件这种物品的结果是否更大。（如果 k=1，说明第 i 种物品放了 2 件，因为前提状态是必然有一件物品已经放入）

f[i][j] = max(f[i-1][j], max(f[i-1][(j-w[i])-k*w[i]]+k*v[i])+v[i])

(k >= 0)

将第 2 步的 j 换成 j-w[i]

f[i][j-w[i]] = max(f[i-1][(j-w[i])-k*w[i]]+k*v[i])

结合之前第二步的式子，可以发现，上式的后半部分就等于 f[i][j-w[i]]+v[i]，于是得出最终状态转移方程：

f[i][j] = max(f[i-1][j], f[i][j-w[i]]+v[i])

解法一：使用二维数组求解，但要注意内存是否足够使用!!!

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

```
using namespace std;
int n,maxw;
int dp[1010][10010],i,j;
int w[10100],v[10100];

int main()
{
    cin>>maxw>>n;
    //读入 n 个物品的重量和价值
    for(i = 1;i <= n;i++){
    cin>>w[i]>>v[i];
    }

    for(i = 1;i <= n;i++){
        for(j = 1;j <= maxw;j++){
            if(j < w[i]){
                dp[i][j] = dp[i-1][j];
            }else{
                dp[i][j] = max(dp[i-1][j],dp[i][j-w[i]]+v[i]);
            }
        }
    }

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

解法二：采用一维数组优化，注意此处要用顺推！

f[j]=max(f[j],f[j-w[i]]+v[i])

```
#include <bits/stdc++.h>
using namespace std;

/*
01 背包：每种物品有 1 个，可选或不选
 dp[i][j] = max(dp[i-1][j],dp[i-1][j-w[i]]+v[i])
 滚动优化：从背包容量 c，降序循环到当前物品重量 wi
 dp[j] = max(dp[j],dp[j-w[i]]+v[i])

完全背包：每种物品有无限多个
 dp[i][j] = max(dp[i-1][j],dp[i-1][j-k*w[i]]+k*v[i])
 经过变换等同于：
 dp[i][j] = max(dp[i-1][j],dp[i][j-w[i]]+v[i])
 一维数组滚动优化：从当前物品重量 wi 正序循环到背包容量 c
 dp[j] = max(dp[j],dp[j-w[i]]+v[i])
*/
int dp[100100],c,wi,vi,n,i,j;

int main(){
    cin>>c>>n;
    //读入 n 个物品
    for(i = 1;i <= n;i++){
        cin>>wi>>vi;
        //正序循环
        for(j = wi;j <= c;j++){
            dp[j] = max(dp[j],dp[j-wi]+vi);
        }
    }

    cout<<dp[c];
    return 0;
}
```

1888：【基础】多重背包（1）

写法一：将多重背包的 si 个物品分别装入 w 和 v 数组，直接转换为 01 背包！

```
#include <bits/stdc++.h>
using namespace std;

/*
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
*/
int n,c;//c 背包容量
int v[10010],w[10010];
int dp[110];
int vi,wi,si,k;//k 代表数组下标

int main(){
    cin>>n>>c;
    for(int i = 1;i <= n;i++){
        cin>>vi>>wi>>si;
        //第 i 个物品有 si 件，都存入数组
        for(int j = 1;j <= si;j++){
            k++;
            v[k] = vi;
            w[k] = wi;
        }
    }

    //01 背包
    for(int i = 1;i <= k;i++){
        //逆序从背包容量循环到当前物品体积
        for(int j = c;j >= v[i];j--){
            dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
        }
    }

    cout<<dp[c];
    return 0;
}
```

解法二：在做 01 背包时，体现一下有 Si 件物品这个条件。

```
#include <bits/stdc++.h>
using namespace std;

/*
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
*/
int n,c;//c 背包容量
int v[110],w[110],s[110];
int dp[110];

int main() {
    cin>>n>>c;
    for(int i = 1; i <= n; i++) {
        cin>>v[i]>>w[i]>>s[i];
    }

    //01 背包
    //有 n 个物品
    for(int i = 1; i <= n; i++) {
        for(int k = 1;k <= s[i];k++) {
```

```
            //逆序从背包容量循环到当前物品体积
            for(int j = c; j >= v[i]; j--) {
                dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
            }
        }

    }

    cout<<dp[c];
    return 0;
}
```

1889：【提高】多重背包（2）

关于 DP 要理解的关键点：

1、DP 的本质

求有限的集合中的最值（个数）

本质上，**DP 代表了走到阶段 i 的所有路线的最优解**；

2、DP 需要思考的点：

（1）DP 的状态是什么？状态要求什么：最大、最小、数量？

（2）DP 的状态计算？

状态转义方程；

求解方法：a、递推　b、考虑阶段 i（最后一个阶段的值）的值是如何得来的；

（3）DP 的边界是什么？

关键术语：阶段、状态、决策（状态转移方程）、边界；

以数塔问题（1216:【基础】数塔问题）为例，理解 DP 的本质，再理解 01 背包的本质（1282:【提高】简单背包问题）；

经典的 DP 模板题要熟练掌握，熟记状态转义方程！

本题解题的关键点：二进制优化（类似压缩的思想）

（1）有 n 个不同的物品，要讨论 2^n 种选择的可能（每个物品选或者不选）；

（2）一个物品有 n 件，虽然要讨论 2^n 种选择的可能，但由于 n 个物品是一样的，那么就减少了讨论数量，比如：有 4 个物品，如果是不同物品的选 2 个，选 1 2、2 3 是不同的选择，但如果是相同的物品，选哪两个就都是一样的了。

因此，n 个物品，要讨论的可能就分别是：选 0 个、选 1 个、选 2 个、选 3 个…选 n 个。

（3）要将 0~n 个不同的选择表达出来，比较简单的方法是将 n 二进制化。

比如：**整数 7**，只需要用 1 2 4 三个数任意组合，就能组合出 0~7 这 8 种可能。

再比如：整数 10，只需要用 1 2 4 3（注意最后一个数），就能组合出 0~10 这 11 种可能，这样 n 这个值就被二进制化了。

因此如果要讨论 10 个一样的物品，就转化为讨论 4 个不同的物品了；而 n 个一样的物品，就转化为 $\log_2 n$ 个不同的物品进行讨论。

dp[j]=max{dp[j], dp[j-w[i]]+v[i]}

```
#include <bits/stdc++.h>
using namespace std;

const int N = 20010;
int v[N],w[N],dp[2010];
int n,m;//n 种物品，背包容量为 m
int vi,wi,si;
int k = 0;

int main(){
    cin>>n>>m;
    for(int i = 1;i <= n;i++){
        cin>>vi>>wi>>si;
        /*
          对 si 二进制化，比如：有 10 件一样的物品
          我们转换为有 4 件不同的物品：1 2 4 3
          这 4 种物品的体积分别是：1*vi 2*vi 4*vi 3*vi
        */
        int t = 1;//权重，表示 2 的次方
        while(t <= si){
            k++;
            v[k] = t * vi;
            w[k] = t * wi;
            si = si - t;
            t = t * 2;
        }

        //如果二进制化有剩余，存入
        if(si > 0){
            k++;
            v[k] = si * vi;
            w[k] = si * wi;
        }
    }

    //01 背包
    for(int i = 1;i <= k;i++){
        for(int j = m;j >= v[i];j--){
            dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
        }
    }

    cout<<dp[m];
    return 0;
}
```

1905：【提高】混合背包


```
#include <bits/stdc++.h>
using namespace std;

const int N = 20000;
int v[N],w[N],s[N];
int vi,wi,si;
int k = 0;//表示存入数组的数据量
int dp[1010];
int n,m;

int main(){
    cin>>n>>m;
    for(int i = 1;i <= n;i++){
        cin>>vi>>wi>>si;
        //如果是多重背包，做二进制拆分
```

```
        if(si > 0){
            int t = 1;
            while(t <= si){
                k++;
                w[k] = t * wi;
                v[k] = t * vi;
                s[k] = -1;//转换为 01 背包
                si = si - t;
                t = t * 2;
            }

            if(si > 0){
                k++;
                w[k] = si * wi;
                v[k] = si * vi;
                s[k] = -1;//01 背包
            }
        } else{
            k++;
            w[k] = wi;
            v[k] = vi;
            s[k] = si;
        }
    }

    //计算
    //循环 k 个物品
    for(int i = 1;i <= k;i++){
        //判断是 01 背包还是完全背包
        if(s[i] == -1){
            for(int j = m;j >= v[i];j--){
                dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
            }
        } else{
            for(int j = v[i];j <= m;j++){
                dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
            }
        }
    }

    cout<<dp[m];
    return 0;
}
```