第一章 动态规划 完成情况：22/29

包括数字三角形模型、最长上升子序列模型等内容 09/14 - 09/27

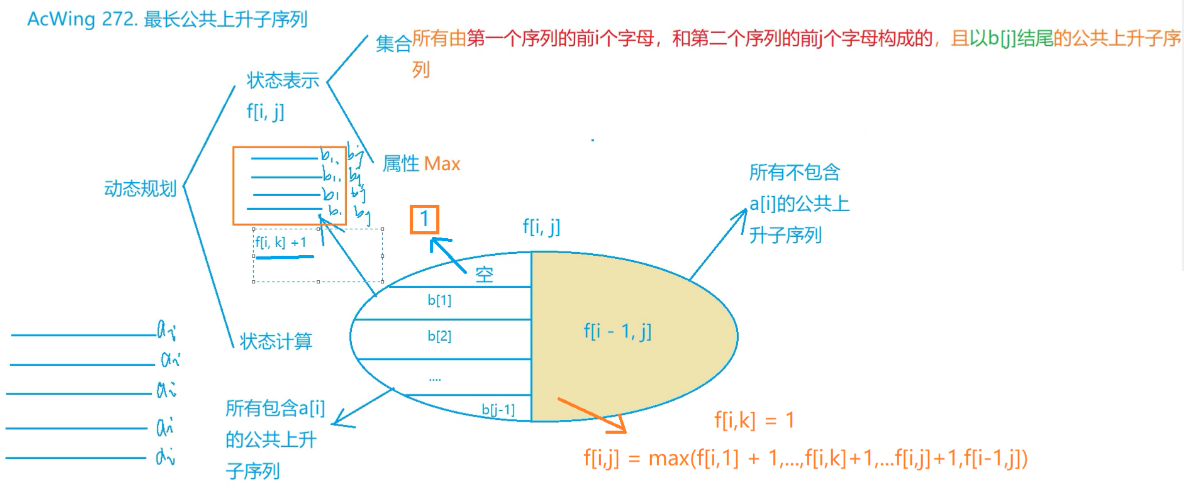
数字三角形模型	AcWing 1015. 摘花生	87人打卡
	AcWing 1018. 最低通行费	78人打卡
	AcWing 1027. 方格取数	60人打卡
最长上升子序列模型	AcWing 1017. 怪盗基德的滑翔翼	59人打卡
	AcWing 1014. 登山	56人打卡
	AcWing 482. 合唱队形	56人打卡
	AcWing 1012. 友好城市	46人打卡
	AcWing 1016. 最大上升子序列和	52人打卡
	AcWing 1010. 拦截导弹	40人打卡
	AcWing 187. 导弹防御系统	29人打卡
	AcWing 272. 最长公共上升子序列	27人打卡
背包问题	AcWing 423. 采药	25人打卡
	AcWing 1024. 装箱问题	24人打卡
	AcWing 1022. 宠物小精灵之收服	14人打卡
	AcWing 278. 数字组合	14人打卡
	AcWing 1023. 买书	12人打卡
	AcWing 1021. 货币系统	13人打卡
	AcWing 532. 货币系统	6人打卡
	AcWing 6. 多重背包问题 III	6人打卡
	AcWing 1019. 庆功会	8人打卡
	AcWing 7. 混合背包问题	7人打卡
	AcWing 8. 二维费用的背包问题	12人打卡
	AcWing 1020. 潜水员	6人打卡
	AcWing 1013. 机器分配	5人打卡
	AcWing 426. 开心的金明	19人打卡
	AcWing 10. 有依赖的背包问题	9人打卡
	AcWing 11. 背包问题求方案数	16人打卡
	AcWing 12. 背包问题求具体方案	11人打卡
	AcWing 734. 能量石	4人打卡
	AcWing 487. 金明的预算方案	10人打卡

最长上升子序列

[最长公共上升子序列]

动态规划分析图：

　　结合了最长上升子序列和最长公共子序列的知识



O(n^3)做法

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

const int N = 3010;

int a[N],b[N],f[N][N];

int main()
{
    int n;
    cin >> n;

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

    for(int i = 1; i<= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            f[i][j] = f[i-1][j];//不包含a[i]的情况
            if(a[i] == b[j])//包含a[i]的情况
            {
                f[i][j] = max(f[i][j],1);
                for(int k = 1; k < j; k++)
                    if(b[k] < b[j])
                        f[i][j] = max(f[i][j],f[i][k] + 1);
            }
        }
    }

    int res = 0;
    for(int i = 1; i <= n; i++) res = max(res,f[n][i]);

    cout << res << endl;

    return 0;
}
```

完全背包

[货币系统](#)

一道裸的完全背包问题，不过要求的属性变成了count

```
#include <iostream>
using namespace std;
typedef long long LL;

const int N = 3010;
int n,m;
LL f[N];

int main()
{
    cin >> n >>m;

    f[0] = 1;
    for(int i = 1; i <= n; i++)
    {
        int v;
        cin >> v;
        for(int j = v; j <= m; j++)
            f[j] += f[j-v];
    }

    cout << f[m] << endl;

    return 0;
}
```

[货币系统加强版](#)

最主要的是三个关于 **a[1...n],和b[1...m]** 两个数组的性质，有点线性代数感觉~~

```
性质 1 ： a[ ]中的所有数都可以由b[ ]中的数表示出来
性质 2 ： 在最优解中，b[ ]中的所有数一定是从 a[ ] 在中选择的
性质 3 ： b1,b2,b3...bn中的数一定不能被其他 b[ ] 中的数表示出来
```

因此，问题可以转化成一个完全背包问题。

如果 `a[i]` 不能被前 `a[1~i-1]` 凑出来，则 `a[i]` 将被添加到 `b[ ]` 中

但一定要把 `a[ ]` 数组排序，因为只有大的数才可能被小的数组合出来

因此，只需做一遍属性为count的完全背包，就可以判断。

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

const int N = 110,M = 25010;

int f[M],a[N],n,m;

int main()
{
```

```
    int T;
    cin >> T;
    while(T--)
    {
        int n;
        cin >> n;
        for(int i = 0; i < n; i++)
            cin >> a[i];

        sort(a,a + n);

        memset(f,0,sizeof f);
        f[0] = 1;

        int ans = 0, m = a[n-1];
        for(int i = 0; i < n; i++)
        {
            if(!f[a[i]])
                ans++;
            for(int j = a[i]; j <= m; j++)
                f[j] += f[j - a[i]];
        }

        cout << ans << endl;
    }

    return 0;
}
```

二维费用的背包问题

[二维费用的01背包问题]()

朴素版：

与普通的01背包一样，只不过多了一维

也可以优化成二维的01背包

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

const int N = 1010 , M = 110;
int f[N][M][M],v[N],w[N],m[N];

int main()
{
    int n,cv,cm;
    cin >> n >> cv >> cm;

    for(int i = 1; i <= n; i++)
        cin >> v[i] >> m[i] >> w[i];

    for(int i = 1; i <= n; i++)
        for(int j = 0; j <= cv; j++)
            for(int k = 0; k <= cm; k++)
```

```
            {
                f[i][j][k] = f[i-1][j][k];
                if( j >= v[i] && k >= m[i])
                    f[i][j][k] = max(f[i][j][k],f[i-1][j-v[i]][k-m[i]] + w[i]);
            }

    cout << f[n][cv][cm] << endl;

    return 0;
}
```

在背包问题里，体积最多是 j 、体积恰好是 j、体积至少是 j 是三种不同的情况

体积最多是 j 时， 应将f[i,j]全部初始化为 0 ， 并且 v >= 0

体积恰好是 j 时，应将f[0,0]初始化成 0，将其他的初始化为非法状态，并且 v >= 0

体积至少是 j 时，应将 f[0,0]初始化成 0，将其他初始化成非法状态，并且 v 不一定要大于 0 ， 因为是至少

[潜水员] 体积至少的情况

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

const int N = 50,M = 160;
int f[N][M];

int main()
{
    int n,m,k;
    cin >> n >> m;
    cin >> k;

    memset(f,0x3f,sizeof f);
    f[0][0] = 0;

    while(k--)
    {
        int v1,v2,w;
        cin >> v1 >> v2 >> w;
        for(int i = n; i >= 0; i--)
            for(int j = m; j >= 0; j--)
                f[i][j] = min(f[i][j],f[max(0,i - v1)][max(0,j - v2)] + w);
    }

    cout << f[n][m] << endl;

    return 0;
}
```

背包的具体选择方案

[求背包具体方案]

AcWing 12. 背包问题求具体方案

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

f[n, m]

最短路问题，求最短路径

其实是判断出每个物品是否被选

物品 i 的选法

- 只能选　必选它　（当前状态从f[i-1,j-v[i]]转移过来）
- 只能不选　必不选它　（从f[i-1,j]转移过来）
- 可选，可不选　一定要选一个（从f[i-1,j]或f[i-1,j-v[i]]转移过来，字典序选第一个）

遍历物品顺序不影响选法
因此方便输出，将物品从n到1遍历
如果不逆序遍历的话，可能得不到最小字典序

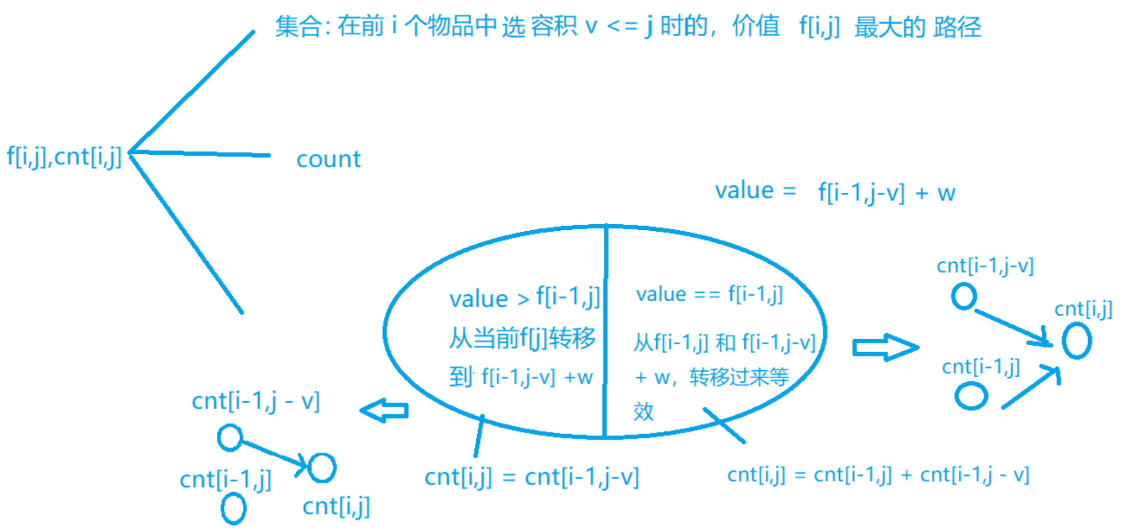
```cpp
#include <iostream>
using namespace std;

const int N = 1010;
int f[N][N], v[N], w[N], n, m;

int main()
{
    cin >> n >> m;

    for (int i = 1; i <= n; i++)
        cin >> v[i] >> w[i];

    for (int i = n; i >= 1; i--)
    {
        for (int j = 0; j <= m; j++)
        {
            f[i][j] = f[i + 1][j];
            if (j >= v[i])
                f[i][j] = max(f[i][j], f[i + 1][j - v[i]] + w[i]);
        }
    }

    int j = m;
    for (int i = 1; i <= n; i++)
        if (j >= v[i] && f[i][j] == f[i + 1][j - v[i]] + w[i])
        {
            cout << i << ' ';
            j -= v[i];
        }

    return 0;
}
```

背包问题求方案数

[背包问题求方案数]()

这和图论中求最短路路径数是相似的思路

01背包分析



初始时，所有最大价值f[i,j]都是 0 ，即全都不选，因此需要将 cnt[i,j] 全部置成 0 (也是一条路径)

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

const int N = 1010,mod = 1e9 + 7;

int f[N],cnt[N];

int main()
{
    int n,m;
    cin >> n >> m;

    for(int i = 0; i <= m; i++) cnt[i] = 1;

    for(int i = 1; i <= n; i++)
    {
        int v,w;
        cin >> v >> w;
        for(int j = m; j >= v; j--)
        {
            int value = f[j - v] + w;
            if(value > f[j]) //从f[i-1,j-v] + w转移过来
            {
                f[j] = value;
                cnt[j] = cnt[j - v];
            }
            else if(value == f[j])//可以从两个地方转移过来
            {
                cnt[j] = (cnt[j] + cnt[j - v]) % mod;
            }
        }
```

```
    }

    cout << cnt[m] << endl;

    return 0;
}
```

将集合设定为恰好的情况

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

const int N = 1010 , mod = 1e9 + 7;
int f[N],cnt[N];

int main()
{
    int n,m;
    cin >> n >> m;

    memset(f,-0x3f,sizeof f); //先将所有状态置成非法状态
    f[0] = 0; //将f[0,0]置成 0
    cnt[0] = 1; //cnt的定义是体积为 j 时的最优解的方案数(路径数)

    for(int i = 1; i <= n; i++)
    {
        int v,w;
        cin >> v >> w;
        for(int j = m; j >= v; j--)
        {
            int val = max(f[j],f[j - v] + w);
            int count = 0;
            if(val == f[j]) count += cnt[j]; //f[i,j]从 f[i-1,j]转移过来
            if(val == f[j - v] + w) count += cnt[j - v]; //f[i,j]从f[i-1,j-v] + w 转移过来
            f[j] = val;
            cnt[j] = count % mod;
        }
    }

    int res = 0;//因为定义的是体积恰好，而不是体积 <= 。所以最优解不一定是f[m]
    for(int i = 0; i <= m; i++) res = max(res,f[i]); //找到最解

    int ans = 0;
    for(int i = 0; i <= m; i++)
        if(res == f[i]) //找到最优解
            ans = (ans + cnt[i]) % mod; //累加体积为 i 时的最优解的方案数cnt[i]

    cout << ans << endl;

    return 0;
}
```

分组背包问题

[机器分配](#) [分组背包](#)

分组背包 + 输出方案

将n个公司看成n组物品，m台机器看成m体积，每个公司获得机器个数所创造的价值看成是每组背包的价值，所需要的体积就是获得机器的个数。

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

const int N = 20;

int f[N][N],w[N][N],way[N],n,m;

int main()
{
    cin >> n >> m;

    for(int i = 1; i<= n; i++)
        for(int j = 1; j <= m; j++)
            cin >> w[i][j];

    for(int i = 1; i <= n; i++)
        for(int j = 0; j <= m; j++)
        {
            f[i][j] = f[i-1][j];
            for(int k = 1; k <= j; k++) //枚举选择当前组 i 的第 k 个物品
                    f[i][j] = max(f[i][j],f[i-1][j - k] + w[i][k]);// k 也对应当前
物品的体积
        }

    cout << f[n][m] << endl;

    int j = m;
    for(int i = n; i >= 1 ; i--)
        for(int k = 0; k <= m; k++)  //必须从 0 开始不然会漏掉一些情况(f[i-1][j]转移过
来)
            if(j >= k && f[i][j] == f[i-1][j-k] + w[i][k])
            {
                way[i] = k;
                j -= k;
                break;
            }

    for(int i = 1; i <= n; i++)
        cout << i << ' ' << way[i] << endl;

    return 0;
}
```

[金明的预算方案](#)

该问题可以转化成一个分组背包求解

购买每个附件必须购买它的主件，因此我们可以枚举每个主件及该主件附件的选择方案。

假设该主件有 n 个附件，则该主件及其附件选择方案数有2^n种选法。

每一个主件和它的附件构成一个组，它们的选法方案对应该组的物品数量。

体积为价格，价值为 价格*重要程度

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

#define v first
#define w second

const int M = 32010, N = 65;

typedef pair<int,int> PII;

int f[M],m,n;
PII primer[N],combine[N][5];
vector<PII> attachment[N];

int main()
{
    cin >> m >> n;

    for(int i = 1; i <= n; i++)
    {
        int v,w,q;
        cin >> v >> w >> q;
        if(!q)  primer[i] = {v, v * w};
        else    attachment[q].push_back({v, v * w});
    }

    for(int i = 1; i <= n; i++) //求每一组的具体方案(物品)
    {
        if(primer[i].v) // 组 i 存在
        {   //枚举 每一种 选法(状态，组 i 的物品数量)
            for(int j = 0; j < 1 << attachment[i].size(); j++)//二进制状态表示
            {
                int v = primer[i].v , w = primer[i].w; //主件必选
                for(int k = 0; k < attachment[i].size(); k++) //如果没有附件则不会
执行该循环
                    if( j >> k & 1 ) //当前选法 j 中选了当前的 附件 k
                    {
                        v += attachment[i][k].v; //加上附件 k 的体积和价值
                        w += attachment[i][k].w;
                    }
                combine[i][j] = {v, w}; //存储当前组 i 的第 j 个物品的体积和价值
            }
        }
    }

    for(int i = 1; i <= n; i++) //分组背包
    {
        if(primer[i].v) //该组存在
        {
            for(int j = m; j >= 0; j--)
```

```
            {
                for(int k = 0; k < 1 << attachment[i].size(); k++) //枚举当前组 i
物品数量
                    if(j >= combine[i][k].v)
                        f[j] = max(f[j],f[j - combine[i][k].v] + combine[i]
[k].w);
            }
        }
    }

    cout << f[m] << endl;

    return 0;
}
```

混合背包问题

混合背包问题

混合的多重背包，顾名思义就是三种背包混合起来。

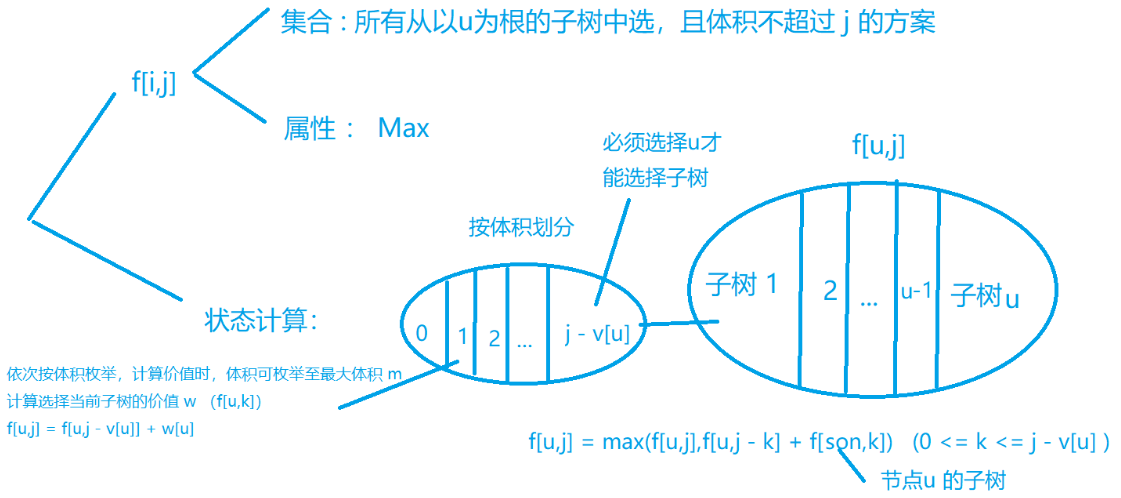
只需在枚举决策时，按对应的背包种类进行枚举

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

const int N = 1010;
int f[N];

int main()
{
    int n,m;
    cin >> n >> m;

    for(int i = 1; i <= n; i++)
    {
        int v,w,s;
        cin >> v >> w >> s;
        if(s == 0) //完全背包
        {
            for(int j = v; j <= m; j++)
                f[j] = max(f[j],f[j - v] + w);
        }
        else
        {
            if(s == -1) s = 1; // 01背包 ， 特殊的多重背包

            for(int k = 1; k <= s; k *= 2) // 多重背包 二进制优化
            {
                for(int j = m; j >= k * v; j--)
                    f[j] = max(f[j],f[j - k * v] + k * w);
                s -= k;
            }

            if(s) // 多重背包 二进制优化
```

```
                for(int j = m; j >= s * v; j--)
                    f[j] = max(f[j],f[j - s * v] + s * w);
        }
    }

    cout << f[m] << endl;

    return 0;
}
```

有依赖的背包问题

分组背包做法



将每个子树看成一组物品，**按体积来划分物品**，选择物品，其中每个物品的体积和价值可以是该费节点多个子树的体积和价值之和。

在递归到尽头才开始计算每颗子树（每一个分组）的物品价值 `w[i]` 即 `f[u][k]`

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

const int N = 110;
int n,m;
int f[N][N],v[N],w[N];
int h[N],e[N],ne[N],idx;

void add(int a,int b)
{
    e[idx] = b,ne[idx] = h[a], h[a] = idx++;
}

void dfs(int u)
{
    for(int i = h[u]; i != -1; i = ne[i]) //循环物品组
    {
        int son = e[i];
        dfs(son);
        //分组背包
```

```
        for(int j = m - v[u]; j >= 0; j--)//循环体积，想选择子节点必须选择它的父节点
            for(int k = 0; k <= j; k++) //循环决策，按体积分组
                f[u][j] = max(f[u][j],f[u][j - k] + f[son][k]);

    }
    //
    for(int j = m; j >= v[u]; j--) // 把当前节点 u 放入背包中，按体积划分，也是求 w
        f[u][j] = f[u][j - v[u]] + w[u];
    for(int j = 0; j < v[u]; j++) // 把体积小于当前层的状态置成 0
        f[u][j] = 0;                    // 不能省，因为每个子节点物品体积都不一样
}

int main()
{
    cin >> n >> m;

    memset(h,-1,sizeof h);

    int root;
    for(int i = 1; i <= n ; i++)
    {
        int p;
        cin >> v[i] >> w[i] >> p;
        if(p == -1) root = i;
        else add(p,i);
    }

    dfs(root);

    cout << f[root][m] << endl;

    return 0;
}
```

dfs构建新物品，0 1背包做法

```
#include <iostream>
#include <cstring>

using namespace std;

const int C = 105;

int h[C], ne[C], e[C], idx;

void add(int a, int b) {
    e[idx] = b;
    ne[idx] = h[a];
    h[a] = idx ++;
}

int N, V;
int v[C], w[C];
int f[C][C];
int size[2 * C], dfsx[2 * C];
int new_v[C], new_w[C];
int cnt;
```

```
//递归枚举每种选法
void dfs(int x) {
    dfsx[x] = ++cnt;
    size[cnt] = 1;
    new_v[cnt] = v[x];
    new_w[cnt] = w[x];
    for(int i = h[x]; i != -1; i = ne[i]) {
        int j = e[i];
        dfs(j);
        size[dfsx[x]] += size[dfsx[j]];
    }
}


int main() {
    cin >> N >> V;
    int root;
    memset(h,  -1, sizeof h);
    for (int i = 1; i <= N; i ++ ){
        int p;
        cin >> v[i] >> w[i] >> p;
        if (p == -1) root = i;
        else add(p, i);
    }

    dfs(root);

    for(int i = N; i >= 1; i --)
    {
        for(int j = 0; j <= V;j ++)
        {
            f[i][j] = f[i+size[i]][j];
            if(j >= new_v[i])
                f[i][j] = max(f[i][j],f[i+1][j-new_v[i]] + new_w[i]);

        }
    }

    cout << f[1][V];
    return 0;
}
链接：https://www.acwing.com/solution/content/5780/
```