背包问题算法入门

（一） 部分背包问题

相较于 01 背包，部分背包的物品**可以分割**，使用**贪心算法**求解即可。本文不进行讨论。

（二） 01 背包问题

01 背包问题（01 knapsack problem）：一共有 N 件物品，第 i（i 从 1 开始）件物品的重量为 w[i]，价值为 v[i]。在总重量不超过背包承载上限 W 的情况下，能够装入背包的最大价值是多少？

- dp[i][j]表示将前 i 件物品装进限重为 j 的背包可以获得的最大价值，0<=i<=N, 0<=j<=W
- 那么我们可以将 dp[0][0...W]初始化为 0，表示将前 0 个物品（即没有物品）装入书包的最大价值为 0。
- 即状态转移方程为：

dp[i][j] = max(dp[i−1][j], dp[i−1][j−w[i]]+v[i]) // j >= w[i]

dp[i][j]=dp[i-1][j] //j<w[i]

```
cin>>m>>n;
for(int i=1;i<=n;i++)
        cin>>w[i]>>c[i];
for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
                dp[i][j]=dp[i-1][j];
                if(j>=w[i])
                        dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+c[i]);
        }
}
cout<<dp[n][m];
```

- 01 背包问题伪代码(空间优化版) ：

dp[0,...,W] = 0

for i = 1,...,N

for j = W,...,w[i] // 必须逆向枚举!!!

dp[j] = max(dp[j], dp[j−w[i]]+v[i])

```
for(int i=1;i<=n;++i){
    for(int j=m;j>=w[i];--j){//需要逆序
        dp[j]=max(dp[j],dp[j-w[i]]+c[i]);
    }
//  for(int i=1;i<=m;i++) cout<<dp[i]<<" ";
//  cout<<endl;
}
cout<<dp[m];
```

时间复杂度为 O(NW)，空间复杂度为 O(W)。

（三） 完全背包问题

完全背包（unbounded knapsack problem）与 01 背包不同就是**每种物品可以有无限多个**：一共有 N 种物品，每种物品有无限多个，第 i（i 从 1 开始）种物品的重量为 w[i]，价值为 v[i]。在总重量不超过背包承载上限 W 的情况下，能够装入背包的最大价值是多少？

- dp[i][j]表示将前 i 种物品装进限重为 j 的背包可以获得的最大价值，0<=i<=N，0<=j<=W
- 初始状态也是一样的，我们将 dp[0][0...W]初始化为 0，表示将前 0 种物品（即没有物品）装入书包的最大价值为 0。那么当 i > 0 时 dp[i][j]也有两种情况：
- 状态转移方程为：

dp[i][j] = max(dp[i−1][j], dp[i][j−w[i]]+v[i]) // j >= w[i]

与 01 背包问题唯一不同就是 max 第二项不是 dp[i-1]而是 dp[i]

```
for(int i=1;i<=n;++i){
    for(int j=1;j<=m;++j){
        dp[i][j]=dp[i-1][j];
        if(j>=w[i]){
            dp[i][j]=max(dp[i][j],dp[i][j-w[i]]+c[i]);
        }
        for(int k=0;j>=k*w[i];k+=1){//类似多重背包
            dp[i][j]=max(dp[i][j],dp[i][j-k*w[i]]+k*c[i]);
        }
    }
}
```

- 空间优化

dp[0,...,W] = 0

for i = 1,...,N for j = w[i],...,W // 必须正向枚举!!!

dp[j] = max(dp[j], dp[j−w[i]]+v[i])

```
for(int i=1;i<=n;++i){
    for(int j=w[i];j<=m;++j){//必须正向枚举
        dp[j]=max(dp[j],dp[j-w[i]]+c[i]);
    }
}
```

（四） 多重背包问题

多重背包（bounded knapsack problem）与前面不同就是**每种物品是有限个**：一共有 N 种物品，第 i（i 从 1 开始）种物品的数量为 n[i]，重量为 w[i]，价值为 v[i]。在总重量不超过背包承载上限 W 的情况下，能够装入背包的最大价值是多少？

- 分析和完全背包的分析二差不多，也是从装入第 i 种物品多少件出发：装入第 i

种物品 0 件、1 件、...n[i]件（还要满足不超过限重）。所以状态方程为：

k 为装入第 i 种物品的件数

k <= min(n[i], j/w[i])

dp[i][j] = max{(dp[i-1][j – k*w[i]] + k*v[i]) for every k}

- 全背包问题思路二伪代码(空间优化版)

dp[0,...,W] = 0

for i = 1,...,N for j = W,...,w[i] // **必须逆向枚举!!!**

for k = [0, 1,..., min(n[i], j/w[i])]

dp[j] = max(dp[j], dp[j−k*w[i]]+k*v[i])

- 可以将多重背包转换成 01 背包问题，采用二进制思路将第 i 种物品分成了 log(n[i]) 件物品。

```
for(int i=1;i<=n;++i){
    for(int j=1;j<=m;++j){
        dp[i][j]=dp[i-1][j];
        for(int k=0;k<=s[i] && j>=k*v[i];++k){//枚举物品的数量
            dp[i][j]=max(dp[i][j],dp[i-1][j-k*v[i]]+k*w[i]);
        }
    }
}
```

- 空间优化代码：

```
for(int i=1;i<=n;++i){
    for(int j=m;j>=v[i];--j){//多重背包的空间优化，务必逆向枚举
        for(int k=0;k<=s[i] && j>=k*v[i];++k){
            dp[j]=max(dp[j],dp[j-k*v[i]]+k*w[i]);
        }
    }
}
cout<<dp[m];
```

（五） 混合背包问题

一个旅行者有一个最多能装 V 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...,Wn，它们的价值分别为 C1,C2,...,Cn。有的物品只可以取一次（01 背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

```
//混合背包：有些物品01背包、有些完全背包无限次、有些多重背包若干次
int n,m,w[35],c[35],p[35],dp[35][205];
int main(){
	cin>>m>>n;
	for(int i=1;i<=n;++i){
		scanf("%d%d%d",&w[i],&c[i],&p[i]);
		if(p[i]==0) p[i]=m/w[i];
	}
	for(int i=1;i<=n;++i){
		for(int j=1;j<=m;++j){
			dp[i][j]=dp[i-1][j];
			for(int k=1;k<=p[i] && j>=k*w[i];++k){
				dp[i][j]=max(dp[i][j],dp[i-1][j-k*w[i]]+k*c[i]);
			}
		}
	}
	cout<<dp[n][m];
```

- 空间优化代码：

```
//混合背包：有些物品01背包、有些完全背包无限次、有些多重背包若干次
//空间优化
int n,m,w[35],c[35],p[35],dp[205];
int main(){
	cin>>m>>n;
	for(int i=1;i<=n;++i){
		scanf("%d%d%d",&w[i],&c[i],&p[i]);
		if(p[i]==0) p[i]=m/w[i];
	}
	for(int i=1;i<=n;++i){
		for(int j=m;j>=w[i];--j){//必须逆向枚举
			for(int k=1;k<=p[i] && j>=k*w[i];++k){
				dp[j]=max(dp[j],dp[j-k*w[i]]+k*c[i]);
			}
		}
	}
	cout<<dp[m];
```

（六） 分组背包问题

物品被分成了若干组，每组只能选其中一个物品。一本通 1272

分组背包问题 ：本质上可以认为是 01 背包，因为一个组里只能选一个，由原来的选不选变为选哪个，决策变了而已！

分组的背包问题首先判断一个分组当中的一件物品，同 01 背包一样，此物品存在两种状态，取与不取，若取此物品，则继续判断下一组的第一件物品，若不取此物品，则继续判断本组下一件物品，若该物品为本组最后一件物品，则判断下一组。也就是说设 f[k][v]表示前 k 组物品花费费用 v 能取得的最大权值，则有：f[k][v]=max{f[k-1][v],f[k-1][v-c[i]]+w[i] | 物品 i 属于组 k}。

```
cin >> W >> N >>T;
memset(dp,0,sizeof(dp));
memset(group,0,sizeof(group));
for(i = 1; i <= N;i++)  {
    cin >> w[i] >> c[i] >> p ;
    group[p][++group[p][0]] = i;//把这个物品存在p组的a[p][0]位置.a[p][0]用于存储物品p组物品个数.

}
for(i = 1 ; i <= T;i++)//第一层遍历的是组数
{
    for(j = W; j >= 0; j--)//第二层遍历的是价值（倒序）
    {
        for(k = 1; k <= group[i][0];k++)//第三层遍历的是每一组的物品。
        {
            int q = group[i][k];//q为第几个物品
            if(j >= w[q])
            dp[j] = max(dp[j],dp[j-w[q]]+c[q]);
        }

    }
}
printf("%d\n",dp[W]);
```

（七） 二维背包

```
using namespace std;
int n,m,k,i,j,t,minm;
int v[1010],u[510],dp[1010][510];
int main(){
    scanf("%d%d%d",&n,&m,&k);
    minm=m;
    for(i=1;i<=k;i++) scanf("%d%d",&v[i],&u[i]);
    for(i=1;i<=k;i++)
        for(j=n;j>=v[i];j--)
            for(t=m;t>=u[i];t--)
                dp[j][t]=max(dp[j][t],dp[j-v[i]][t-u[i]]+1);
    for(i=0;i<=n;i++)
        for(j=0;j<=m;j++)
            if(dp[i][j]==dp[n][m]&&j<minm) minm=j;
    printf("%d %d\n",dp[n][m],m-minm);
    return 0;
}
```