《L4-CSP-J 复赛进阶算法与数据结构》

Day05 动态规划-Dynamic Programming

背包动态规划（下）

主讲人：小武老师

可达信奥_小武老师_keda.ac

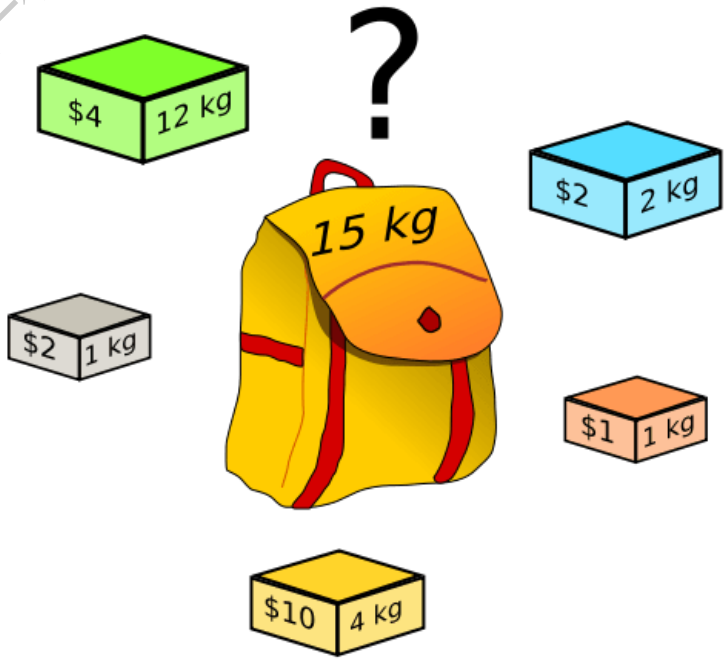
背包DP

背包问题是动态规划的经典问题之一。指的是在容积或者重量限制的情况下装入物品，物品有价值、体积、重量等属性，要求在满足限制的条件下，让背包中物品的价值之和最大。0-1背包，子集背包，完全背包、混合背包

可达信奥_小武老师_keda.ac

0-1 背包问题

给你一个可装载重量为 W 的背包和 N 个物品，每个物品有重量和价值两个属性。其中第 i 个物品的重量为 wt[i]，价值为 val[i]，现在让你用这个背包装物品，最多能装的价值是多少？



这个题目中的物品不可以分割，要么装进包里，要么不装，不能说切成两块装一半。这就是 0-1 背包这个名词的来历。

```
N   = 3, W = 4
wt  = [2, 1, 3]
val = [4, 2, 3]
```

最大价值？ 6

0-1 背包

dp[i][w] 当背包容量在w的时候，前i个物品放入时候背包中获得的最大价值

不同的阶段

V	W	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
4	1	0	4	4	4	4	4	4
8	3	0	4	4	8	12	12	12
4	5	0	4	4	8	12	12	12
6	1	0	6	10	10	14	18	18

当背包容量是w的时候，背包能够装的最大的价值是装最后一个物品的阶段，即最后一个物品如果重量大于背包容量，那么背包的最大价值就是上一阶段的值，即d[i-1][w]. 如果第i阶段的重量小于背包容量，那么背包的最大价值则是d[i-1][w], 和 d[i-1][w-wt[i]]+ v[i]两个里面的较大值

https://alchemist-al.com/algorithms/knapsack-problem

0-1 背包

动规标准套路

Step 1「状态」和「选择」
状态有两个，就是「背包的容量」和「可选择的物品」
选择就是「装进背包」或者「不装进背包」

Step 2 明确 dp 数组的定义
「状态」，有两个，也就是说我们需要一个二维 dp 数组
dp[i][w] 的定义如下：对于前 i 个物品，当前背包的容量为 w，背包获得的最大价值

Step 3 根据「选择」，思考状态转移的逻辑

$$dp[i][w] = \begin{cases} dp[i-1][w] & ,wt[i] > w \\ \max\{dp[i-1][w], dp[i-1][w-wt[i]] + v[i]\} & ,wt[i] \le w \end{cases}$$

0-1 背包

P0456. 01背包问题

$$dp[i][w] = \begin{cases} dp[i-1][w] & , wt[i] > w \\ \max\{dp[i-1][w], dp[i-1][w-wt[i]] + v[i]\} & , wt[i] \le w \end{cases}$$

```
int main(){
    cin >> m >> n;
    for(int i = 1; i <= n; i++) cin >> w[i] >> v[i];
    for(int i = 1; i <= n; i++){
        dp[i][0] = 0;
        for(int j = 1; j <= m; j++){
            dp[0][j] = 0;
            if(w[i] > j){
                dp[i][j] = dp[i-1][j];
            }else{
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i]]+v[i]);
            }
        }
    }
    cout << dp[n][m];
    return 0;
}
```

时间复杂度?

P0456. 01背包问题

一维数组优化

```
int main() {
    //背包容量m和物品数量n
    cin >> m >> n;
    //每个物品的重量和价值
    for (int i = 1; i <= n; i++) cin >> w[i] >> c[i];
    //设dp(v)表示重量不超过v公斤的最大价值
    for (int i = 1; i <= n; i++)
        for (int v = m; v >= w[i]; v--) {
            if (dp[v - w[i]] + c[i] > dp[v]) {
                dp[v] = dp[v - w[i]] + c[i];
            }
        }
    cout << dp[m];
    return 0;
}
```

$$dp[w] = \max\{dp[w], dp[w - wt[i]] + v[i]\}$$

完全背包

P0457. 完全背包问题

状态表示：dp[i][w] 表示将第i个物品放入容量为w的背包中可以获得的最大价值

v	w	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
4	1	0	4	8	12	16	20	24
8	3	0	4	8	12	16	20	24
4	5	0	4	8	12	16	20	24
6	1	0	6	12	18	24	30	36

$$dp[i][w] = \begin{cases} dp[i-1][w] & ,wt[i] > w \\ \max\{dp[i-1][w], dp[i][w-wt[i]] + v[i]\} & ,wt[i] \le w \end{cases}$$

完全背包

P0457. 完全背包问题

$$dp[i][w]=\begin{cases} dp[i-1][w] & ,wt[i]>w \\ \max\{dp[i-1][w],dp[i][w-wt[i]]+v[i]\} & ,wt[i]\le w \end{cases}$$

```
int main(){
    cin >> m >> n;
    for(int i = 1; i <= n; i++) cin >> w[i] >> v[i];
    for(int i = 1; i <= n; i++){
        dp[i][0] = 0;
        for(int j = 1; j <= m; j++){
            dp[0][j] = 0;
            if(w[i] > j){
                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][m];
    return 0;
}
```

时间复杂度？

完全背包

P0457. 完全背包问题

设有 n 种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为 M ，今从 n 种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于 M ，而价值的和为最大。

状态表示：dp[w] 表示将物品放入容量为w的背包中可以获得的最大价值

状态转移方程：$$dp[w] = \max\{dp[w], dp[w - wt[i]] + v[i]\}$$

$$dp[i][w] = \begin{cases} dp[i-1][w] & , wt[i] > w \\ \max\{dp[i-1][w], dp[i][w - wt[i]] + v[i]\} & , wt[i] \le w \end{cases}$$

完全背包

P0457. 完全背包问题

```
int main() {
//背包容量m和物品数量n
    cin >> m >> n;
    for (i = 1; i <= n; i++) cin >> w[i] >> c[i];
    //设 dp[v]表示重量不超过v公斤的最大价值
    for (i = 1; i <= n; i++)
       for (v = w[i]; v <= m; v++)
           if (dp[v - w[i]] + c[i] > dp[v]) dp[v] = dp[v - w[i]] + c[i];
    // dp[m]为最优解
    cout << "max=" << dp[m] << endl;
    return 0;
}
```

关于DP的本质

1、DP为什么会快？

2、DP的核心是什么？

3、关于终极的思考？我是谁？我从哪里来？我要到哪里去？

问题一：DP为什么会快？ DP自带剪枝

无论是DP还是暴力，我们的算法都是在可能解空间内，寻找最优解。

来看硬币问题：

暴力做法是枚举所有的可能解，这是最大的可能解空间。

DP是枚举有希望成为答案的解。这个空间比暴力的小得多。

DP舍弃了一大堆不可能成为最优解的答案。譬如硬币问题：

15 = 5+5+5 被考虑了。

15 = 5+5+1+1+1+1+1 从来没有考虑过，因为这不可能成为最优解。

问题二：DP的核心思想？

大事化小，小事化了。

DP的核心思想：

尽量缩小可能解的空间

在暴力算法中，可能解空间往往是指数级的大小；如果我们采用DP，那么有可能把解空间的大小降到多项式级。

一般来说，解空间越小，寻找解就越快。这样就完成了优化

↓ DP的操作过程

将一个大问题转化成几个小问题；

求解小问题；

推出大问题的解。

问题三：如何设计DP算法?

DP的设计步骤：

我是谁？ ——设计状态，表示局面 x

我从哪里来？ f(x)

我要到哪里去？ ——设计转移

一般而言，DP的难点，在初学时是如何设计状态；在学习深入一些之后，变成了如何设计转移；在省选/NOI级别，又变成了如何设计状态。

0-1 背包总结

1、DP的三个要素：状态、阶段、决策（转移/选择）

2、0-1背包和完全背包的区别

3、关键：状态表示+状态转移方程

课后习题与实验

Talk is cheap, show me the code !

下节课见啦！