

[Hawstein's Blog](#)

- [Home](#)
- [Archive](#)
- [Categories](#)
- [Sitemap](#)
- [About](#)

- [Subscribe](#)

动态规划之背包问题（一）

March 1, 2013

作者：Hawstein

出处：<http://hawstein.com/posts/dp-knapsack.html>

声明：本文采用以下协议进行授权：[自由转载-非商用-非衍生-保持署名|Creative Commons BY-NC-ND 3.0](#)，转载请注明作者及出处。

一切都要从一则故事说起。

话说有一哥们去森林里玩发现了一堆宝石，他数了数，一共有 n 个。但他身上能装宝石的就只有一个背包，背包的容量为 C 。这哥们把 n 个宝石排成一排并编上号： $0, 1, 2, \dots, n-1$ 。第 i 个宝石对应的体积和价值分别为 $V[i]$ 和 $W[i]$ 。排好后这哥们开始思考：背包总共也就只能装下体积为 C 的东西，那我要装下哪些宝石才能让我获得最大的利益呢？

OK，如果是你，你会怎么做？你斩钉截铁的说：动态规划啊！恭喜你，答对了。那么让我们来看看，动态规划中最最重要的两个概念：状态和状态转移方程在这个问题中分别是什么。

我们要怎样去定义状态呢？这个状态总不能是凭空想象或是从天上掉下来的吧。为了方便说明，我们先实例化上面的问题。一般遇到 n ，你就果断地给 n 赋予一个很小的数，比如 $n=3$ 。然后设背包容量 $C=10$ ，三个宝石的体积为5，4，3，对应的价值为20，10，12。对于这个例子，我想智商大于0的人都知道正解应该是把体积为5和3的宝石装到背包里，此时对应的价值是 $20+12=32$ 。接下来，我们把第三个宝石拿走，同时背包容量减去第三个宝石的体积（因为它是装入背包的宝石之一），于是问题的各参数变为： $n=2$ ， $C=7$ ，体积 $\{5, 4\}$ ，价值 $\{20, 10\}$ 。好了，现在这个问题的解是什么？我想智商等于0的也解得出了：把体积为5的宝石放入背包（然后剩下体积2，装不下第二个宝石，只能眼睁睁看着它溜走），此时价值为20。这样一来，我们发现， $n=3$ 时，放入背包的是0号和2号宝石；当 $n=2$ 时，我们放入的是0号宝石。这并不是一个偶然，没错，这就是传说中的“全局最优解包含局部最优解”（ $n=2$ 是 $n=3$ 情况的一个局部子问题）。绕了那么大的圈子，你可能要问，这都哪跟哪啊？说好的状态呢？说好的状态转移方程呢？别急，它们已经呼之欲出了。

我们再把上面的例子理一下。当 $n=2$ 时，我们要求的是前2个宝石，装到体积为7的背包里能达到的最大价值；当 $n=3$ 时，我们要求的是前3个宝石，装到体积为10的背包里能达到的最大价值。有没有发现它们其实是一个句式！OK，让我们形式化地表示一下它们，定义 $d(i, j)$ 为前 i 个宝石装到剩余体积为 j 的背包里

能达到的最大价值。那么上面两句话即为： $d(2, 7)$ 和 $d(3, 10)$ 。这样看着真是爽多了，而这两个看着很爽的符号就是我们要找的状态了。即状态 $d(i, j)$ 表示前 i 个宝石装到剩余体积为 j 的背包里能达到的最大价值。上面那么多的文字，用一句话概括就是：根据子问题定义状态！你找到子问题，状态也就浮出水面了。而我们最终要求解的最大价值即为 $d(n, C)$ ：前 n 个宝石（ $0, 1, 2, \dots, n-1$ ）装入剩余容量为 C 的背包中的最大价值。状态好不容易找到了，状态转移方程呢？顾名思义，状态转移方程就是描述状态是怎么转移的方程（好废话！）。那么回到例子， $d(2, 7)$ 和 $d(3, 10)$ 是怎么转移的？来，我们来说说2号宝石（记住宝石编号是从0开始的）。从 $d(2, 7)$ 到 $d(3, 10)$ 就隔了这个2号宝石。它有两种情况，装或者不装入背包。如果装入，在面对前2个宝石时，背包就只剩下体积7来装它们，而相应的要加上2号宝石的价值12， $d(3, 10) = d(2, 10-3) + 12 = d(2, 7) + 12$ ；如果不装入，体积仍为10，价值自然不变了， $d(3, 10) = d(2, 10)$ 。记住， $d(3, 10)$ 表示的是前3个宝石装入到剩余体积为10的背包里能达到的最大价值，既然是最大价值，就有 $d(3, 10) = \max\{d(2, 10), d(2, 7) + 12\}$ 。好了，这条方程描述了状态 $d(i, j)$ 的一些关系，没错，它就是状态转移方程了。把它形式化一下： $d(i, j) = \max\{d(i-1, j), d(i-1, j-V[i-1]) + W[i-1]\}$ 。注意讨论前 i 个宝石装入背包的时候，其实是在考查第 $i-1$ 个宝石装不装入背包（因为宝石是从0开始编号的）。至此，状态和状态转移方程都已经有了。接下来，直接上代码。

```
for(int i=0; i<=n; ++i){
    for(int j=0; j<=C; ++j){
        d[i][j] = i==0 ? 0 : d[i-1][j];
        if(i>0 && j>=V[i-1]) d[i][j] >= d[i-1][j-V[i-1]]+W[i-1];
    }
}
```

$i=0$ 时， $d(i, j)$ 为什么为0呢？因为前0个宝石装入背包就是没东西装入，所以最大价值为0。if语句里， $j \geq V[i-1]$ 说明只有当背包剩余体积 j 大于等于 $i-1$ 号宝石的体积时，我才考虑把它装进来的情况，不然 $d[i][j]$ 就直接等于 $d[i-1][j]$ 。 $i>0$ 不用说了吧，前0个宝石装入背包的情况是边界，直接等于0，只有 $i>0$ 才有必要讨论，我是装呢还是不装呢。简单吧，核心算法就这么一丁点，接下来上完整代码knapsack.cpp。

```
/**0-1 knapsack d(i, j)表示前i个物品装到剩余容量为j的背包中的最大重量**/
#include<cstdio>
using namespace std;
#define MAXN 1000
#define MAXC 100000

int V[MAXN], W[MAXN];
int d[MAXN][MAXC];

int main(){
    freopen("data.in", "r", stdin); //重定向输入流
    freopen("data.out", "w", stdout); //重定向输出流
    int n, C;
    while(scanf("%d %d", &n, &C) != EOF){
        for(int i=0; i<n; ++i) scanf("%d %d", &V[i], &W[i]);

        for(int i=0; i<=n; ++i){
            for(int j=0; j<=C; ++j){
                d[i][j] = i==0 ? 0 : d[i-1][j];
                if(i>0 && j>=V[i-1]) d[i][j] >= d[i-1][j-V[i-1]]+W[i-1];
            }
        }
        printf("%d\n", d[n][C]); //最终求解的最大价值
    }
    fclose(stdin);
    fclose(stdout);
}
```

```

    return 0;
}

```

其中freopen函数将标准输入流重定向到文件data.in， 这比运行程序时一点点手输要方便许多，将标准输出流重定向到data.out。 data.in中每组输入的第一行为宝石数量n及背包体积C，接下来会有n行的数据，每行两个数对应的是宝石的体积及价值。本测试用例data.in如下：

```

5 10
4 9
3 6
5 1
2 4
5 1
4 9
4 20
3 6
4 20
2 4
5 10
2 6
2 3
6 5
5 4
4 6

```

data.out为算法输出结果，对应该测试用例，输出结果如下：

```

19
40
15

```

好，至此我们解决了背包问题中最基本的0/1背包问题。等等，这时你可能要问，我现在只知道背包能装入宝石的最大价值，但我还不知道要往背包里装入哪些宝石啊。嗯，好问题！让我们先定义一个数组x，对于其中的元素为1时表示对应编号的宝石放入背包，为0则不放入。让我们回到上面的例子，对于体积为5，4，3，价值为20，10，12的3个宝石，如何求得其对应的数组x呢？（明显我们目测一下就知道x={1 0 1}，但程序可目测不出来）OK，让我们还是从状态说起。如果我们将2号宝石放入了背包，那么是不是也就意味着，前3个宝石放入背包的最大价值要比前2个宝石放入背包的价值大，即： $d(3, 10) > d(2, 10)$ 。再用字母代替具体的数字（不知不觉中我们就用了不完全归纳法哈），当 $d(i, j) > d(i-1, j)$ 时， $x(i-1)=1$;OK，上代码：

```

//输出打印方案
int j = C;
for(int i=n; i>0; --i){
    if(d[i][j] > d[i-1][j]){
        x[i-1] = 1;
        j = j - v[i-1]; //装入第i-1个宝石后背包能装入的体积就只剩下j - v[i-1]
    }
}
for(int i=0; i<n; ++i) printf("%d ", x[i]);

```

好了，加入这部分内容，knapsack.cpp变为如下：

```

/**0-1 knapsack d(i, j)表示前i个物品装到剩余容量为j的背包中的最大重量**/
#include<cstdio>
using namespace std;

```

```

#define MAXN 1000
#define MAXC 100000

int V[MAXN], W[MAXN], x[MAXN];
int d[MAXN][MAXC];

int main(){
    freopen("data.in", "r", stdin);
    freopen("data.out", "w", stdout);
    int n, C;
    while(scanf("%d %d", &n, &C) != EOF){
        for(int i=0; i<n; ++i) scanf("%d %d", &V[i], &W[i]);
        for(int i=0; i<n; ++i) x[i] = 0; //初始化打印方案

        for(int i=0; i<=n; ++i){
            for(int j=0; j<=C; ++j){
                d[i][j] = i==0 ? 0 : d[i-1][j];
                if(i>0 && j>=V[i-1]) d[i][j] >= d[i-1][j-V[i-1]]+W[i-1];
            }
        }
        printf("%d\n", d[n][C]);

        //输出打印方案
        int j = C;
        for(int i=n; i>0; --i){
            if(d[i][j] > d[i-1][j]){
                x[i-1] = 1;
                j = j - V[i-1];
            }
        }
        for(int i=0; i<n; ++i) printf("%d ", x[i]);
        printf("\n");
    }
    fclose(stdin);
    fclose(stdout);
    return 0;
}

```

data.out输出结果变为:

```

19
1 1 0 1 0
40
1 0 1 0
15
1 1 0 0 1

```

至此, 好像该解决的问题都解决了。当一个问题找到一个放心可靠的解决方案后, 我们往往就要考虑一下是不是有优化方案了。为了保持代码的简洁, 我们暂且把宝石装包方案的求解去掉。该算法的时间复杂度是 $O(nC)$, 即时间都花在两个for循环里了, 这个应该是没办法再优化了。再看看空间复杂度, 数组d用来保存每个状态的值, 空间复杂度为 $O(nC)$; 数组V和W用来保存每个宝石的体积和价值, 空间复杂度为 $O(n)$ 。程序总的空间复杂度为 $O(nC)$, 这个是可以进一步优化的。首先, 我们先把数组V和W去掉, 因为它们没有保存的必要, 改为一边读入一边计算:

```

int V = 0, W = 0;
for(int i=0; i<=n; ++i){
    if(i>0) scanf("%d %d", &V, &W);
    for(int j=0; j<=C; ++j){

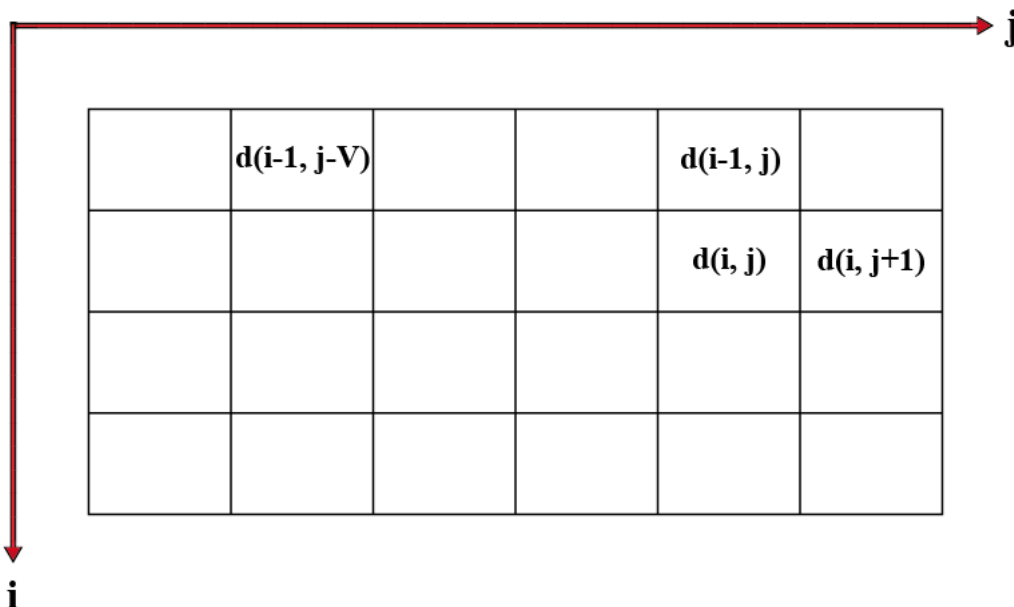
```

```

    d[i][j] = i==0 ? 0 : d[i-1][j];
    if(j>=V && i>0) d[i][j] >?= d[i-1][j-V]+W;
}
}

```

好了，接下来让我们继续压榨空间复杂度。保存状态值我们开了一个二维数组d，在看过把一维数组V和W变为一个变量后，我们是不是要思考一下，有没有办法将这个二维数组也压榨一下呢？换言之，这个二维数组中的每个状态值我们真的有必要都保存么？让我们先来看一下以下的一张示意图（参照《算法竞赛入门经典》P169的图画的）



由上面那一小段优化过后的代码可知，状态转移方程为： $d(i, j) = \max\{d(i-1, j), d(i-1, j-V) + W\}$ ，也就是在计算d(i, j)时我们用到了d(i-1, j)和d(i-1, j-V)的值。如果我们只用一个一维数组d(0)~d(c)来保存状态值可以么？将i方向的维数去掉，我们可以将原来二维数组表示为一维数据：d(i-1, j-V)变为d(j-V)，d(i-1, j)变为d(j)。当我们要计算d(i, j)时，只需要比较d(j)和d(j-V)+W的大小，用较大的数更新d(j)即可。等等，如果我要计算d(i, j+1)，而它恰好要用到d(i-1, j)的值，那么问题就出来了，因为你刚刚才把它更新为d(i, j)了。那么，怎么办呢？按照j递减的顺序即可避免这种问题。比如，你计算完d(i, j)，接下来要计算的是d(i, j-1)，而它的状态转移方程为 $d(i, j-1) = \max\{d(i-1, j-1), d(i-1, j-1-V) + W\}$ ，它不会再用到d(i-1, j)的值！所以，即使该位置的值被更新了也无所谓。好，上代码：

```

memset(d, 0, sizeof(d));
for(int i=0; i<=n; ++i){
    if(i>0) scanf("%d %d", &V, &W);
    for(int j=C; j>=0; --j){
        if(j>=V && i>0) d[j] >?= d[j-V]+W;
    }
}

```

优化后的完整代码如下，此时空间复杂度仅为 $O(C)$ 。

```

/**0-1 knapsack d(i, j)表示前i个物品装到剩余容量为j的背包中的最大重量**/
#include<stdio>
#include<stdlib>
#include<string>
using namespace std;

int main(){
    freopen("data.in", "r", stdin);
    freopen("data.out", "w", stdout);
    int n, C, V = 0, W = 0;
    while(scanf("%d %d", &n, &C) != EOF){
        int* d = (int*)malloc((C+1)*sizeof(int));
        memset(d, 0, (C+1)*sizeof(int));

        for(int i=0; i<=n; ++i){
            if(i>0) scanf("%d %d", &V, &W);
            for(int j=C; j>=0; --j){
                if(j>=V && i>0) d[j] >?= d[j-V]+W;
            }
        }
        printf("%d\n", d[C]);
        free(d);
    }
    fclose(stdin);
    fclose(stdout);
    return 0;
}

```

OK，背包问题暂时先讲这么多，以后接着讲。

Random Posts

- 26 Mar 2016 » [Scala 周报 \[20160320 - 20160326\]](#)
- 19 Mar 2016 » [Scala 周报 \[20160313 - 20160319\]](#)
- 12 Mar 2016 » [Scala 周报 \[20160306 - 20160312\]](#)
- 05 Mar 2016 » [Scala 周报 \[20160228 - 20160305\]](#)
- 14 Feb 2016 » [Scala 周报 \[20160214 - 20160220\]](#)

19 Comments Hawstein's Blog

Login ▾

Recommend 7 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name



Don7 Hao · 3 years ago

>?=?是什么意思?

1 ^ | ▾ · Reply · Share ·



Jiejing Zhang → Don7 Hao · 3 years ago

c++里面应该是: $d[i][j] = \max(d[i-1][j-V[i-1]]+W[i-1], d[i][j]);$

^ | v . Reply . Share ›



Benedict Jin · 7 months ago

给力，非常棒！

^ | v . Reply . Share ›



Yang · 7 months ago

不错的文章，但是请楼主把0/1 背包这个定义放到最前面，因为我一直以为每个宝石可以拿多次。

^ | v . Reply . Share ›



Richard Luo · 10 months ago

非常详细的解释。非常感谢博主的文章！

^ | v . Reply . Share ›



Lu Li · a year ago

写得真好!!! 谢谢~

^ | v . Reply . Share ›



Longqi Zhang · a year ago

为什么说宝石编号的时候是0, 1, 2。描述d(i,j)时候用的是1, 2, 3, 这样让我很困惑。

^ | v . Reply . Share ›



warren zhang · a year ago

$d(i, j) = \max\{d(i-1, j), d(i-1, j-V[i-1]) + W[i-1]\}$ 不正确。应该是: $d(i, j) = \max\{d(i-1, j), d(i-1, j-V[i]) + W[i]\}$ i.e, $d(i, j)$ 取决于不取第i个宝石，或者取第i个宝石。

你的数组定义不清楚，所以看的时候有点不明白。

^ | v . Reply . Share ›



Spyder → **warren zhang** · a year ago

宝石是按0, ..., n-1编号的, $v[i-1]$ 就是i-1号宝石（第i个宝石）的价值

^ | v . Reply . Share ›



kazaf f → **Spyder** · 5 months ago

$j-V[i]$ 含义为包剩余空间-宝石价值? 因该是 $d(i, j) = \max\{d(i-1, j), d(i-1, j-w[i]) + v[i]\}$ 吧?

^ | v . Reply . Share ›



肖帆 · 2 years ago

很受启发，谢谢博主，顺便转走^_^

^ | v . Reply . Share ›



庄瑞铭 · 2 years ago

谢谢分享

^ | v . Reply . Share ›



Richard Xu · 2 years ago

写的很好，很有帮助:

"如果我要计算 $d(i, j+1)$ ，而它恰好要用到 $d(i-1, j)$ 的值，那么问题就出来了，因为你刚刚才把它更新为 $d(i, j)$ 了。"

这个地方想不通，总感觉这种情况不会出现， $d(i, j)$ 只会依赖 $\{(i-1, 1), (i-1, j)\}$ ；所以觉得 $d(i-1, j)$ 应该跟 $d(i, j)$ 无关；

有人能帮忙解释一下吗？谢谢！

^ | v . Reply . Share ›



申伟 → **Richard Xu** · 2 years ago

$d(i, j+1) = \max\{d(i-1, j+1), d(i-1, j+1-V)+W\}$ 我觉得这只是一个假设而已，不是说 $d(i, j+1)$ 一定用到 $d(i-1, j)$ ，如果 $V=1$ ，那么就可能用到 $d(i-1, j)+W$

^ | v . Reply . Share ›



夏际威 · 3 years ago

23333看你的文章真是笑死我了。讲的好

^ | v . Reply . Share ›



pjfgsskl · 3 years ago

很详细~

^ | v . Reply . Share ›



virusdefender · 3 years ago

有帮助，十分感谢~

^ | v . Reply . Share ›

**Zing** · 3 years ago

如果程序中能加多点注释就更好了！

^ | v · [Reply](#) · [Share](#) ›**马天翼** · 4 years ago

博主对dp的讲解非常的透彻~

^ | v · [Reply](#) · [Share](#) ›

[✉ Subscribe](#) [➤ Add Disqus to your siteAdd DisqusAdd](#) [🔒 Disqus' Privacy PolicyPrivacy PolicyPrivacy](#)Powered by [Jekyll](#) and [Bootstrap](#). Last updated at 2016-04-08 02:20:17 +0000.