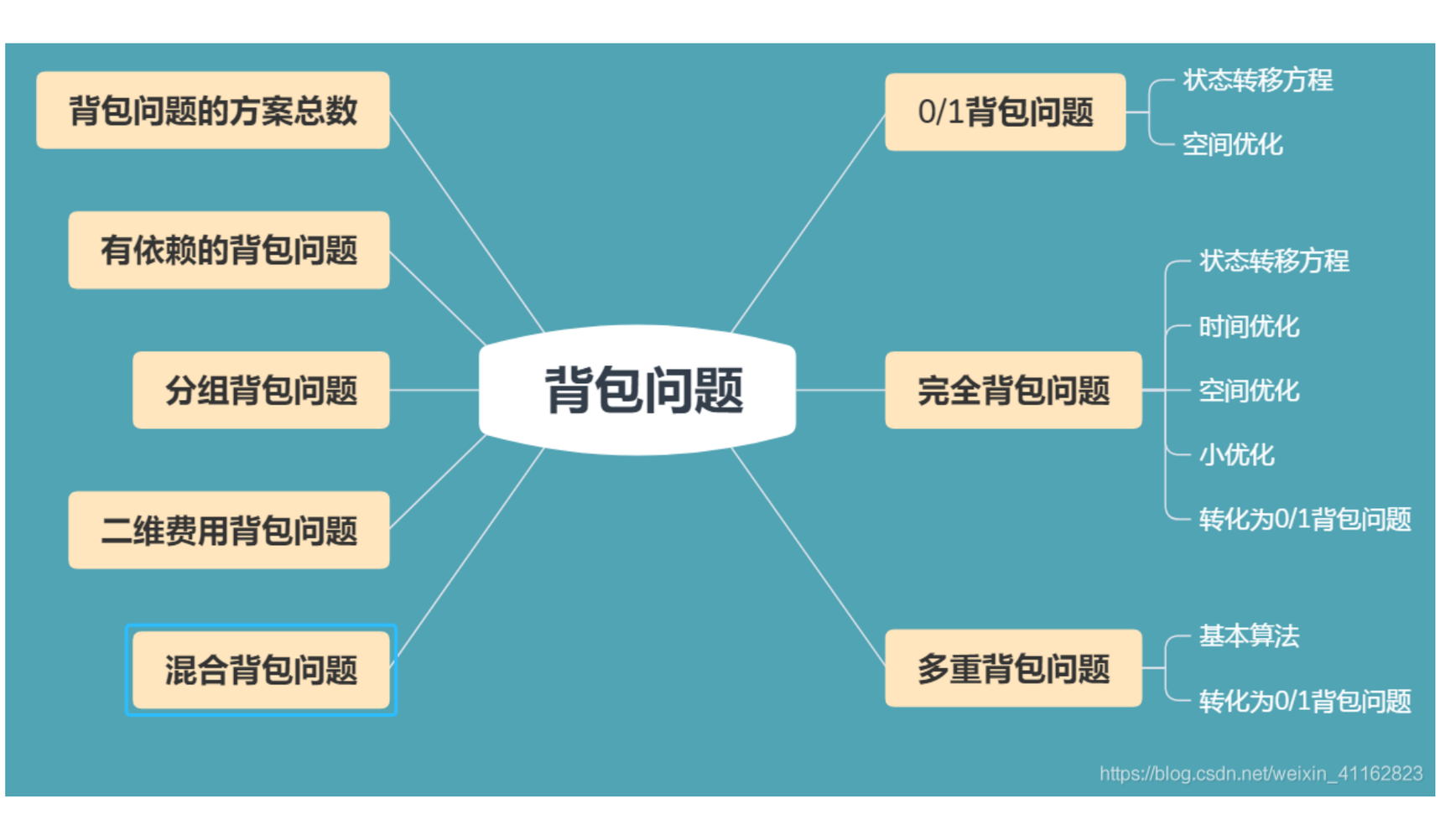
原创 迷亭1213 2019-02-24 16:18:10 53821 已收藏 612 版权
分类专栏： 动态规划 Algorithm ACM学习笔记专栏

> 本文为个人笔记，这算是我第三次系统的学背包问题。根据老师所写课件，所记为个人感悟以及知识点梳理，方便日后复习，如有不足之处请斧正，感谢。



1.背景：

1.1 什么是背包问题：背包问题指这样一类问题，题意往往可以抽象成：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。（来自百度百科）

1.2 背包问题的种类：就ACM或者其它算法竞赛而言，背包问题可以分为8种类型，其中最基础的是**0/1背包问题**。作为动态规划的典型问题，其状态转移方程往往需要认真理解并能自行推出。**这八种问题分别为：0/1背包问题、完全背包问题、多重背包问题、混合三种背包问题、二维费用背包问题、分组背包问题、有依赖的背包问题、求背包问题的方案总数。**

2.0/1背包问题

2.1 问题描述：有N件物品和一个容量为V的背包。第i件物品的费用（即体积，下同）是w[i]，价值是val[i]。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。

2.2 解题思路：用动态规划的思路，阶段就是“物品的件数”，状态就是“背包剩下的容量”，那么很显然f [i , v] 就设为从前 i 件物品中选择放入容量为 v 的背包最大的价值。那么状态转移方程为：

f[i][v]=max{ f[i-1][v],f[i-1][v-w[i]]+val[i] }。

这个方程可以如下解释：只考虑子问题“将前 i 个物品放入容量为 v 的背包中的最大价值”那么考虑如果不放入 i ，最大价值就和 i 无关，就是 f[i - 1][v] , 如果放入第 i 个物品，价值就是 f[i - 1][v - w[i]] + val[i]，我们只需取最大值即可。

2.3 空间优化：上述状态表示，我们需要用二维数组，但事实上我们只需要一维的滚动数组就可以递推出最终答案。考虑到用f[v]来保存每层递归的值，由于我们求f[i][v] 的时候需要用到的是f[i-1][v] 和 f[i-1][v - w[i]] 于是可以知道，只要我们在求f[v]时不覆盖f[v - w[i]]，那么就可以不断递推至所求答案。所以我们采取倒序循环，即v = m（m为背包总容积）伪代码如下：

```
for i = 1..N

   for v = V..0

        f[ v ] = max{ f[ v ],f[ v-w[i] ]+val[ i ] };
```

2.4 代码模板：（根据2.1问题作答）

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn = 1e4;
int f[maxn];
int w[maxn],val[maxn];
void solve(int n,int m){
    memset(f,0,sizeof f);
    for(int i = 1;i <= n;i++){
        for(int v = m;v > 0;v--){
            if(v >= w[i])
                f[v] = max(f[v],f[v-w[i]]+val[i]);
        }
    }
    printf("%d\n",f[m]);
}
int main(){
    int n,m;
    while(scanf("%d%d",&n,&m) != EOF){
        for(int i = 1;i <= n;i++) scanf("%d%d",w+i,val+i);
        solve(n,m);
    }
    return 0;
}
```

3.完全背包问题

3.1 问题描述：有N种物品和一个容量为V的背包，每种物品都有无限件可用。第i种物品的费用是w[i]，价值是val[i]。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。

3.2 解题思路：完全背包问题与0/1背包问题不同之处在于其每个物品是无限的，从每种物品的角度考虑，与它相关的策略就变成了取0件、1件、2件...。我们可以根据0/1背包的思路，对状态转移方程进行改进，令**f[i][v]表示前 i 种物品恰放入一个容量为 v 的背包的最大权值**。状态转移方程就变成了：

f[i][v] = max{ f[i-1][v-k*w[i]] + k*val[i] | 0 <= k*w[i] <= v}。

我们通过对0/1背包的思路加以改进，就得到了完全背包的一种解法，这种解法时间复杂度为O（n^3），空间复杂度为O（n^2）。

3.3 时间优化：根据上述f[i][v]的定义，其为前 i 种物品恰好放入容量为 v 的背包的最大权值。根据上述状态转移方程可知，我们假设的是**子结果f[i-1][v-k*w[i]]中并没有选入第 i 种物品**，所以我们需要逆序遍历（像0/1背包一样）来确保该前提；但是我们现在考虑“加选一件第 i 种物品”这种策略时，正**需要一个可能已经选入第 i 种物品的子结果f[i][v-w[i]]**，于是当我们**顺序遍历**时，就刚好达到该要求。这种做法，使我们省去了一层循环，即第 i 种物品放入的件数k，从而时间复杂度优化为O（n^2）。

3.4 空间优化：正如0/1背包的空间优化，上述状态转移方程已经优化为：

f[i][v]=max{f[i-1][v],f[i][v-w[i]]+val[i]}

将这个方程用一维数组实现，便得到了如下伪代码：

```
for i = 1..N

       for v = 0..V

          f[v] = max{f[v],f[v-w[i]] + val[ i ] };
```

3.5 小剪枝：完全背包问题有一个很简单有效的优化，是这样的：若两件物品i、j满足w[i] <= w[j]且val[i] >= val[j]，则将物品j去掉，不用考虑。这个优化的正确性显然：任何情况下都可将价值小费用高的j换成物美价廉的i，得到至少不会更差的方案。对于随机生成的数据，这个方法往往会大大减少物品的件数，从而加快速度。然而这个并不能改善最坏情况的复杂度，因为有可能特别设计的数据可以一件物品也去不掉。

3.6 转化为0/1背包问题：

既然01背包问题是最基本的背包问题，那么我们可以考虑把完全背包问题转化为01背包问题来解。最简单的想法是，考虑到第i种物品最多选V/w[i]件，于是可以把第i种物品转化为V/w[i]件费用及价值均不变的物品，然后求解这个01背包问题。这样完全没有改进基本思路的时间复杂度，但这毕竟给了我们将完全背包问题转化为01背包问题的思路：将一种物品拆成多件物品。

更高效的转化方法是：把第i种物品拆成费用为w[i]*2^k、价值为val[i]*2^k的若干件物品，其中k满足w[i]*2^k<V。这是二进制的思想，因为不管最优策略选几件第i种物品，总可以表示成若干个2^k件物品的和。这样把每种物品拆成O(log(V/w[i])+1)件物品，是一个很大的改进。

3.7 代码示例：完全背包问题模板

4.多重背包问题

4.1 问题描述：N种物品和一个容量为V的背包。第i种物品最多有n[i]件可用，每件费用是w[i]，价值是val[i]。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。

4.2 解题思路：这种类型的题目又和完全背包有些相似，不同的就在于其数量不是无限的。基本的方程只需将完全背包问题的方程略微一改即可，因为对于第i种物品有n[i]+1种策略：取0件，取1件……取n[i]件。令f[i][v]表示前i种物品恰放入一个容量为v的背包的最大权值，则：f[i][v]=max{ f[i-1][v-k*w[i]] + k*val[i] | 0<= k <= n[i]}。复杂度是O(V*∑n[i])。

4.3 转化为0/1背包问题：

把第i种物品换成n[i]件01背包中的物品，则得到了物品数为∑n[i]的01背包问题，直接求解，复杂度仍然是O(V*∑n[i])。

但是我们期望将它转化为01背包问题之后能够像完全背包一样降低复杂度。仍然考虑二进制的思想，我们考虑把第i种物品换成若干件物品，使得原问题中第i种物品可取的每种策略——取0..n[i]件——均能等价于取若干件代换以后的物品。另外，取超过n[i]件的策略必不能出现。

方法是：将第i种物品分成若干件物品，其中每件物品有一个系数，这件物品的费用和价值均是原来的费用和价值乘以这个系数。使这些系数分别为 1,2,4,...,2^(k-1),n[i]-2^k+1，且k是满足n[i]-2^k+1>0的最大整数**(注意：这些系数已经可以组合出1~n[i]内的所有数字)**。例如，如果n[i]为13，就将这种物品分成系数分别为1,2,4,6的四件物品。

分成的这几件物品的系数和为n[i]，表明不可能取多于n[i]件的第i种物品。另外这种方法也能保证对于0..n[i]间的每一个整数，均可以用若干个系数的和表示，这个证明可以分0..2^k-1和2^k..n[i]两段来分别讨论得出，并不难，希望你自己思考尝试一下。

这样就将第i种物品分成了O(logn[i])种物品，将原问题转化为了**复杂度为O(V*∑logn[i])的01背包问题**，是很大的改进。

4.4 参考模板：例题-庆功会

5.混合三种背包问题

5.1 问题描述：如果将01背包、完全背包、多重背包混合起来。也就是说，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）。应该怎么求解呢?

5.2 0/1背包与完全背包的混合：

考虑到在01背包和完全背包中最后给出的伪代码只有一处不同，故如果只有两类物品：一类物品只能取一次，另一类物品可以取无限次，那么只需在对每个物品应用转移方程时，根据物品的类别选用顺序或逆序的循环即可，复杂度是O(VN)。

伪代码如下：

```
for i=1..N

   if 第i件物品是01背包

      for v=V..0

         f[v] = max{ f[ v ] , f[ v-w[i] ] + val[i] };

   else if 第i件物品是完全背包

      for v=0..V

         f[v] = max{ f[v] , f[v-w[i] ] + val[i] };
```

5.3 再加上多重背包：

如果再加上有的物品最多可以取有限次，那么原则上也可以给出O(VN)的解法：遇到多重背包类型的物品用单调队列解即可。但如果不考虑超过NOIP范围的算法的话，用多重背包中将每个这类物品分成O(log n[i])个01背包的物品的方法也已经很优了。

5.4 例题：混合背包

6.二维费用背包问题

6.1 问题描述：二维费用的背包问题是指：对于每件物品，具有两种不同的费用；选择这件物品必须同时付出这两种代价；对于每种代价都有一个可付出的最大值（背包容量）。问怎样选择物品可以得到最大的价值。设这两种代价分别为代价1和代价2，第i件物品所需的两种代价分别为a[i]和b[i]。两种代价可付出的最大值（两种背包容量）分别为V和U。物品的价值为c[i]。

6.2 算法：

费用加了一维，只需状态也加一维即可。设f[i][v][u]表示前i件物品付出两种代价分别为v和u时可获得的最大价值。

状态转移方程就是：f [i][v][u]=max{f[i-1][v][u],f[i-1][v-a[i]][u-b[i]]+c[i]}。如前述方法，可以只使用二维的数组：当每件物品只可以取一次时变量v和u采用逆序的循环，当物品有如完全背包问题时采用顺序的循环。当物品有如多重背包问题时拆分物品。

6.3 物品总个数的限制：

有时，“二维费用”的条件是以这样一种隐含的方式给出的：**最多只能取M件物品。**这事实上相当于每件物品多了一种“件数”的费用，每个物品的件数费用均为1，可以付出的最大件数费用为M。**换句话说，设f[v][m]表示付出费用v、最多选m件时可得到的最大价值，则根据物品的类型（01、完全、多重）用不同的方法循环更新，最后在f[0..V][0..M]范围内寻找答案。**

另外，如果要求“恰取M件物品”，则在f[0..V][M]范围内寻找答案。

6.4 例题：潜水员

7.分组背包问题

7.1 问题描述：有N件物品和一个容量为V的背包。第i件物品的费用是w[i]，价值是c[i]。这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。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。

7.2 算法：

这个问题变成了每组物品有若干种策略：是选择本组的某一件，还是一件都不选。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-w[i]] + c[i] | 物品i属于第k组 }。

使用一维数组的伪代码如下：

```
for 所有的组k

   for v=V..0

      for 所有的i属于组k

              f[v]=max{f[v],f[v-w[i]]+c[i]}
```

注意这里的三层循环的顺序，“for v=V..0”这一层循环必须在“for 所有的i属于组k”之外。这样才能保证每一组内的物品最多只有一个会被添加到背包中。

另外，显然可以对每组中的物品应用完全背包中“一个简单有效的优化”（3.5）。

7.3 例题：分组背包

8.有依赖的背包问题

8.1 简化的问题：

这种背包问题的物品间存在某种“依赖”的关系。也就是说，i依赖于j，表示若选物品i，则必须选物品j。为了简化起见，我们先设没有某个物品既依赖于别的物品，又被别的物品所依赖；另外，没有某件物品同时依赖多件物品。

8.2 算法：

这个问题由**NOIP2006金明的预算方案**一题扩展而来。遵从该题的提法，将不依赖于别的物品的物品称为“主件”，依赖于某主件的物品称为“附件”。由这个问题的简化条件可知所有的物品由若干主件和依赖于每个主件的一个附件集合组成。

按照背包问题的一般思路，仅考虑一个主件和它的附件集合。可是，可用的策略非常多，包括：一个也不选，仅选择主件，选择主件后再选择一个附件，选择主件后再选择两个附件……无法用状态转移方程来表示如此多的策略。（事实上，设有n个附件，则策略有2^n+1个，为指数级。）

考虑到所有这些策略都是互斥的（也就是说，你只能选择一种策略），所以一个主件和它的附件集合实际上对应于**分组的背包**中的一个物品组，每个选择了主件又选择了若干个附件的策略对应于这个物品组中的一个物品，其费用和价值都是这个策略中的物品的值的和。但仅仅是这一步转化并不能给出一个好的算法，因为物品组中的物品还是像原问题的策略一样多。

再考虑**分组的背包**中的一句话： 可以对每组中的物品应用**完全背包**中“一个简单有效的优化”。这提示我们，对于一个物品组中的物品，所有费用相同的物品只留一个价值最大的，不影响结果。所以，我们可以对主件i的“附件集合”先进行一次01背包，得到费用依次为0..V-w[i]所有这些值时相应的最大价值f'[0..V-w[i]]。那么这个主件及它的附件集合相当于V-w[i]+1个物品的物品组，其中费用为w[i]+k的物品的价值为f'[k]+c[i]。也就是说原来指数级的策略中有很多策略都是冗余的，通过一次01背包后，将主件i转化为 V-w[i]+1个物品的物品组，就可以直接应用**分组的背包**的算法解决问题了。

更一般的问题是：依赖关系以图论中“森林”的形式给出（森林即多叉树的集合），也就是说，主件的附件仍然可以具有自己的附件集合，限制只是每个物品最多只依赖于一个物品（只有一个主件）且不出现循环依赖。

解决这个问题仍然可以用将每个主件及其附件集合转化为物品组的方式。唯一不同的是，由于附件可能还有附件，就不能将每个附件都看作一个一般的01 背包中的物品了。若这个附件也有附件集合，则它必定要被先转化为物品组，然后用分组的背包问题解出主件及其附件集合所对应的附件组中各个费用的附件所对应的价值。

事实上，这是一种树形DP，其特点是每个父节点都需要对它的各个儿子的属性进行一次DP以求得自己的相关属性。这已经触及到了“泛化物品”的思想。看完后，你会发现这个“依赖关系树”每一个子树都等价于一件泛化物品，求某节点为根的子树对应的泛化物品相当于求其所有儿子的对应的泛化物品之和。

8.3 小结：

NOIP2006的那道背包问题，通过引入“物品组”和“依赖”的概念可以加深对这题的理解，还可以解决它的推广问题。用物品组的思想考虑那题中极其特殊的依赖关系：物品不能既作主件又作附件，每个主件最多有两个附件，可以发现一个主件和它的两个附件等价于一个由四个物品组成的物品组，这便揭示了问题的某种本质。

9.求背包问题的方案总数

9.1 问题描述：对于一个给定了背包容量、物品费用、物品间相互关系（分组、依赖等）的背包问题，除了再给定每个物品的价值后求可得到的最大价值外，还可以得到装满背包或将背包装至某一指定容量的方案总数。

9.2 算法：

对于这类改变问法的问题，一般只需将状态转移方程中的max改成sum即可。例如若每件物品均是01背包中的物品，转移方程即为f[i][v] = sum{ f[i-1][v] , f[i-1][v-w[i]] + c[i] }，初始条件 f[0][0] = 1。

事实上，这样做可行的原因在于状态转移方程已经考察了所有可能的背包组成方案。

9.3 例题：货币系统

> **PS：**本来想写个人心得整理的，但无奈怎么总结都没原文写的易懂，尤其是后面几个没怎么接触过的背包问题，所以后面基本都是复制课件的，等练习练习再把updata吧。