0-1背包问题

动态规划法

首先我们要了解动态规划法的基本思想是什么，动态规划法与分治法类似，都是把大问题拆分成几个小问题，通过寻求大问题与小问题的递推关系，解决一个个小问题，最终达到解决原问题的效果。但不同的是，分治法在子问题和子子问题等上被重复计算了很多次，而动态规划则具有记忆性，通过填写表把所有已经解决的子问题答案纪录下来，在新问题里需要用到的子问题可以直接提取，避免了重复计算，从而节约了时间，所以在问题满足最优性原理之后，用动态规划解决问题的核心就在于填表，表填写完毕，最优解也就找到。

离不开一个关键词，拆分 ，就是把求解的问题分解成若干个子阶段，前一问题的结果就是求解后一问题的子结构。在求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解。

解题基本步骤

1. 分析最优子结构性质，刻画左右子结构特征
2. 构造递推公式
3. 计算最优值
4. 回溯得到最优解

动态规划法解**0-1**背包问题

给定n种物品和一背包。物品i的重量是W_i，其价值为V_i，背包容量为C 。一个物品要么全部装入背包，要么全部不装入背包，不允许部分装如背包，装入背包的物品的总重量不超过背包的容量。 n=5，C=8，W={2，2，7，5，4}，V={3，6，4，5，6}，问应该如何选择装入背包的物品，使得装入背包中的物品总价值最大。

我们用V_i表示第i个物品的价值，W_i表示第i个物品的体积，Val(i,j):表示当前背包容量j,前i个物品最佳组合对应的价值，同时背包问题抽象化(X_1，X_2，X_3，..X_i..X_n) X_i取0或1表是第i个物品选不选。

建立模型：$max(V_1X_1+V_2X_2+....V_nX_n)$;

建立约束条件：$W_1X_1+W_2X_2+W_3X_3+....W_nX_n < C$

寻找递推关系：

- 第i个物品的重量大于当前背包容量，装不下，此时价值与前i-1个的价值一样的，Val(i, j) = Val(i-1, j);
- 背包容量大于第i个物品时，到底装不装，所以在装与不装只之间选择一个，比较哪一个价值更高

  Val(i, j) = max{Val(i-1, j), Val(i-1, j-w(i)) + V(i)}

  其中Val(i-1,j)表示不装，这个比较容易理解。

  $Val(i-1,j-W_i)+V_i$ 这个我当时理解就比较困难，看了好多个教学视频才理解

  表示装了第i个商品，在装之前的状态就是$Val(i-1, j-W_i)$装入第i件之前就应该留有W_i的空间给第i件,$Val(i-1, j-W_i)$再由前面的递推得到，因此我们就可以得到递推关系

递推关系式：

$j<W_i$ $Val(i,j)=V(i-1,j)$ $j>=W_i$ $Val(i,j)=max\{Val(i-1,j)，Val(i-1,j-W_i)+V_i\}$

根据递推关系我们进行填表，找出最优解。

填表：

初始条件Val(0,j) = Val(i,0) = 0

(W,V)	I/J	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0	0
(2,3)	1	0	0	3	3	3	3	3	3	3
(2,6)	2	0	0	6	6	9	9	9	9	9
(7,4)	3	0	0	6	6	9	9	9	9	9
(5,5)	4	0	0	6	6	9	9	9	11	11
(4,6)	5	0	0	6	6	9	9	12	12	15

递推得全过程我就不一一写出来了，我写出了几个比较关键的点

Val(1,2) 也是一个转折点，此时j = 2>=W_1

Val(1,2) = max{Val(0,2)，Val(0,0)+3} = max{0,3}=3

....

Val(2,2)是一个转折点，此时j = 2>= W_2，所以根据递推关系，

Val(2,2) = max{Val(1,2) ， Val(1, 0)+V_2} = Val(2,2) = max{3,6}=6

.....

Val(2,4)也是一个转折点，此时j = 4 >= W_2 ，根据递推关系：

Val(2,4) = max{Val(1,4), Val(1,2) + V_2} = max{3,9} = 9

.....

Val(4,7) ,此时j = 7 >= W_4，根据递推关系：

Val(4,7) = max{Val(3,7)，Val(3,2)+V_4} = max{9，6 + 5} = 11

.....

Val(5, 6),此时j = 6 >= W_5，根据递推关系：

Val(5, 6) = max{Val(4,6)，Val(3,2) + V_2} = max{9，6+6} = 12

.....

Val(5, 8), 此时j = 8 >= W_5，根据递推关系回溯：

Val(5, 8) = max{Val(4,8)，Val(3,4) + V_5} = max{11，9+6} = 15

Val(3,4) = max{Val(2,4)，Val(2,2) + V_2} = max{3，3+6} = 9

Val(2,2) =max{Val(1,2)，Val(1,0) +V_1} = max{3，3} = 3

所以Val(5,8) = Val(1,0) + V_1 + V_2 + V_5 = 15

最终我们得到最优解为15, 选择1号2号5号物品

规模趋于**n**的时候时间复杂度

填写表格我们发现，表中每一个元素要么是有Val(i-1,j)得到，要么是由Val(i-1,j-W_i)+V_i得到，耗时O(1)；所以规模趋于n的时候，时间复杂度为O(nW)，n为物品数量，W为背包容量。

回溯法

首先我们要对回溯法有一个了解，回溯法就是一种有组织的系统化搜索技术，可以看作是蛮力法穷举搜索的改进。回溯法每次只构建可能解的一部分，然后评估这个部分解，如果这个部分有可能导致一个完全解，对其进一步搜索，否则，就不必继续构造这部分的解了，回溯法常常可以避免搜索所有可能的解，所以，它往往比满立法效率更高，适用于求解组合数组较大的问题。

解题基本步骤

（1）针对具体问题，定义问题的解空间；

（2）确定易于搜索的解空间结构（数据结构的选择）。

（3）一般以**DFS**的方式搜索解空间。

（4）在搜索过程中，可以使用剪枝函数等来优化算法。（剪枝函数：用约束函数和限界函数剪去得不到最优解的子树，0统称为剪枝函数。）

解空间：顾名思义，就是一个问题的所有解的集合。（但别忘了，这离我们要求的最优解还差很远！）

约束条件：有效解的要求，即题目的要求。

约束函数：减去不满足约束条件的子树的函数

限界函数：去掉得不到最优解的节点的函数

使用回溯法解决**0-1**背包问题

给定n种物品和一背包。物品i的重量是W_i，其价值为V_i，背包容量为C 。一个物品要么全部装入背包，要么全部不装入背包，不允许部分装如背包，装入背包的物品的总重量不超过背包的容量。 n=5，C=8，W={2，2，7，5，4}，V={3，6，4，5，6}，问应该如何选择装入背包的物品，使得装入背包中的物品总价值最大。

问题的解空间

(X_1，X_2，X_3，X_4，X_5) X_i=0或1

首先应该设置问题的约束条件和限界条件

01背包问题的解空间包括2^n个可能解，很显然不是每一个可能解的物品都能装入背包的，因此我们要设置约束条件判断可能得解描述的装入背包的物品总重量不超过背包容量，如果超出就是不可行解，否则就是可行解，搜索过程也不再搜索那些导致不可行解的节点，(X_i取0或取1，表示第i个物品装不装，用来判断装入背包的物品总重量是否大于背包容量)因此约束条件课表示为：

约束条件：

$$\sum_{i=1}^{n} W_i X_i <= W$$

添加限界条件来，来加快找出最优解。如何添加限界条件呢？ 我们使用cp表示当前已经装入背包的物品总价值，rp表示剩余不知道是否装入的物品的总价值，bestp表示点前已经找到的最优解的价值。所以限界条件可以表示为：

限界条件：

$$cp + rp > bestp$$

搜索过程： 先了解相关概念名词

活节点：还没生出孩子的节点 死节点：不能生孩子的节点 扩展节点：当前正在产生子节点的节点称为扩展节点

从根节点开始，以深度优先的方式进行搜索。根节点首先成为活节点，也是当前的扩展节点。由于子集树中约定左分支上的值为“1”，因此沿着扩展节点的左分支扩展，则代表装入物品，此时，需要判断是否能够装入该物品，即判断约束条件成立与否，如果成立，就进入左孩子节点，左孩子节点成为活节点，并且是当前的扩展节点，继续向纵深节点扩展；如果不成立，就剪掉扩展节点的左分支,沿着其右分支扩展。右分支代表物品不装人背包，肯定有可能导致可行解。但是沿着右分支扩展有没有可能得到最优解呢？这一点需要由限界条件来判断。如果限界条件满足，说明有可能导致最优解，即进入右分支，右孩子节点成为活节点，并成为当前的扩展节点，继续向纵深节点扩展;如果不满足限界条件，则剪掉扩展节点的右分支，开始向最近的活节点回溯。搜索过程直到所有活节点变成死节点后结束。

根据上述的搜索过程进行搜索

首先，从根节点开始，即根节点是活节点，也是当前节点的扩展节点。它代表初始状态即背包是空的。如下图1-1所示。

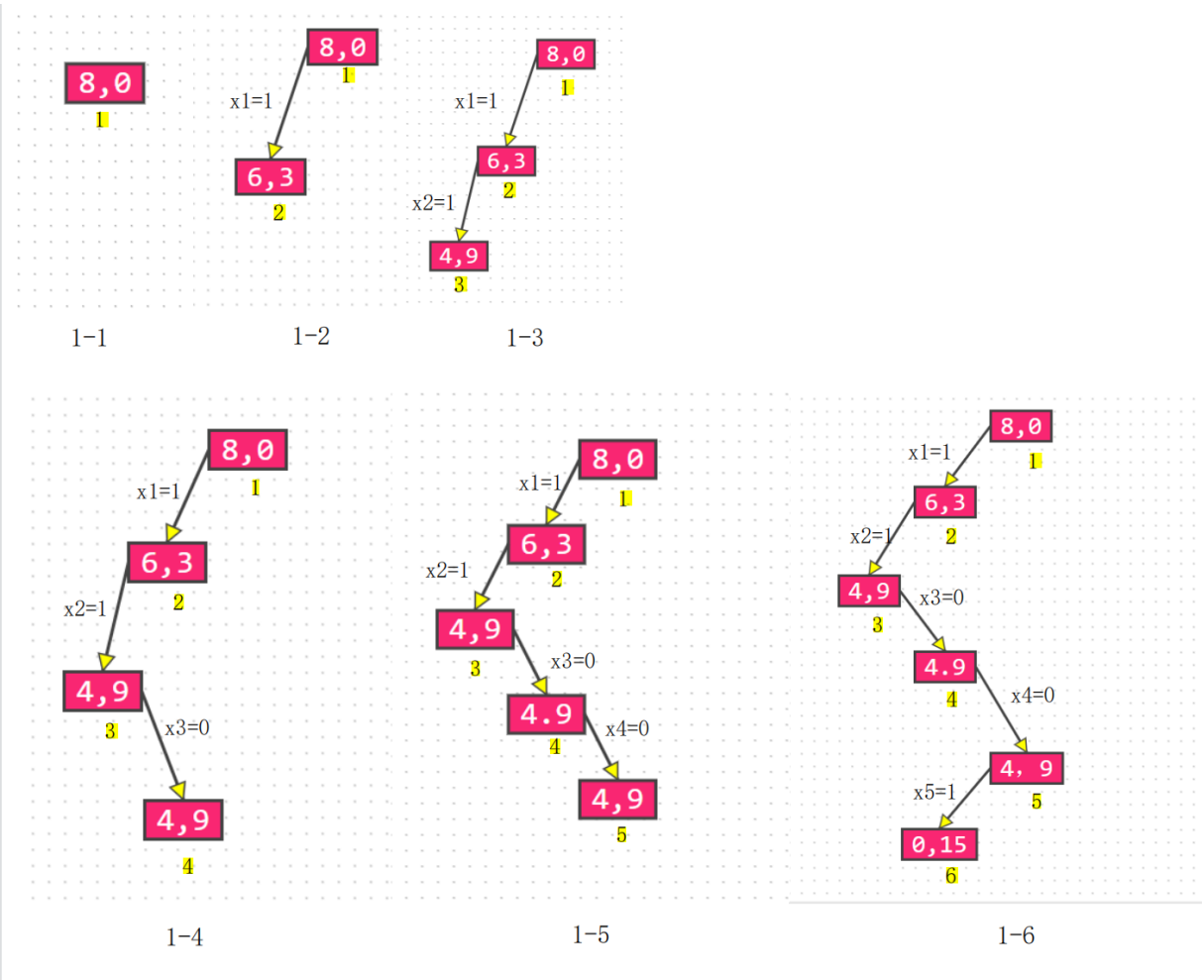
扩展节点1先沿着左分支扩展，此时需要判断约束条件，第一件物品重量为2，2<8，满足约束条件，因此节点2成为活节点，并成为当前的扩展节点。它代表物品1已经装入背包，背包剩余容量为6，背包内物品总价值为3，如图1-2所示。

扩展节点2继续沿着左分支扩展，此时需要判断第二件物品是否能装入背包，第二件物品重量为2，背包剩余容量6，2<6，所以第二件物品可以装入背包，因此第3个节点成为活节点，并成为当前的扩展节点。它代表物品1和物品2都已经装入背包，背包剩余容量为4，背包内物品总价值为9，如图1-3所示。

扩展节点3继续沿着左分支扩展，此时需要判断第三件物品是否能装入背包，第三件物品重量为7，7>4，显然第三件物品装不进去，故减掉扩展节点2的左分支，此时需要扩展节点沿着右分支继续扩展，判断限界条件 cp+rp>bestp，此时cp=9，rp=11，bestp=0，cp+rp>bestp，限界条件成立，则节点3沿右分支扩展的节点4成为活节点，并成为当前的扩展节点，扩展节点4，代表背包剩余容量为4，背包内物品总价值为9，如图1-4所示。

以此类推，节点4沿着左分支扩展，此时需要判断第四件物品是否能装入背包，第四件物品重量为5，5>4，显然第四件物品装不进去，故减掉扩展节点4的左分支，此时需要扩展节点沿着右分支继续扩展，判断限界条件 cp+rp>bestp，此时cp=9，rp=6，bestp=0，cp+rp>bestp，限界条件成立，则节点4沿右分支扩展的节点5成为活节点，并成为当前的扩展节点，扩展节点5，代表背包剩余容量为4，背包内物品总价值为9，如图1-5所示。

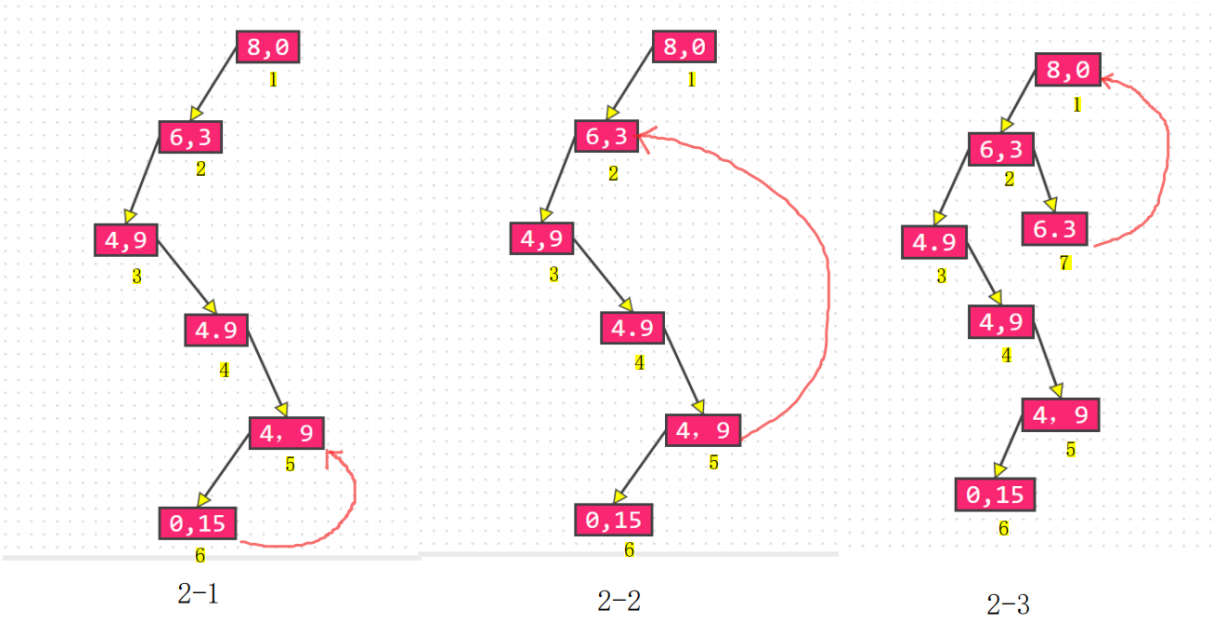
节点5继续沿着左分支扩展，此时需要判断第五件物品是否可以装入背包，第五件物品重量为4，4=4，可以刚好装进，节点6满足约束条件，此时节点6以经是叶子节点，故找到一个当前最优解，记录修改bestp的值，bestp=15，当前最优解为装入背包的总价值为15，如图1-6所示。



由于节点6已经是叶子节点，不再具备扩展能力，此时要回溯到离节点最近的节点5，节点5再次成为扩展节点，如图2-1所示。扩展节点沿着右分支继续扩展，此时要判断是否满足限界条件，cp=9，rp=0，bestp=15，显然不满足条件，故剪去5的右分支，此时节点3的左右分支均搜索完毕。

回溯到最近的活节点2，节点2再次成为扩展节点，如图2-2所示。节点2继续沿右分支扩展，此时要判断是否满足限界条件，cp=3，rp=15，cp+rp>bestp，所以满足限界条件。

节点7成为活结点，并成为当前结点的扩展结点，节点7沿着左分支扩展，判断是否满足约束条件，此时物品3重量为7，背包剩余容量为6，7>6，不满足，向右分支扩展，判断是否满足限界条件，此时cp=3，rp=11，bestp=15，cp+rp<bestp，所以不满足限界条件，剪去节点7的右分支。此时扩展节点2的左右分支搜索完毕。回溯到节点1，如图2-3所示。



扩展节点1沿着右分支继续扩展，判断是否满足限界条件，cp=0，rp=21，bestp=15，cp+rp>bestp，满足限界条件，则扩展节点8成为活节点，并成为当前的扩展节点，如图3-1所示。

扩展节点8沿着左分支继续扩展，判断约束条件，当前背包剩余容量8，物品2的重量为2，8>2，满足约束条件，扩展节点9成为活节点，并成为当前的扩展节点。

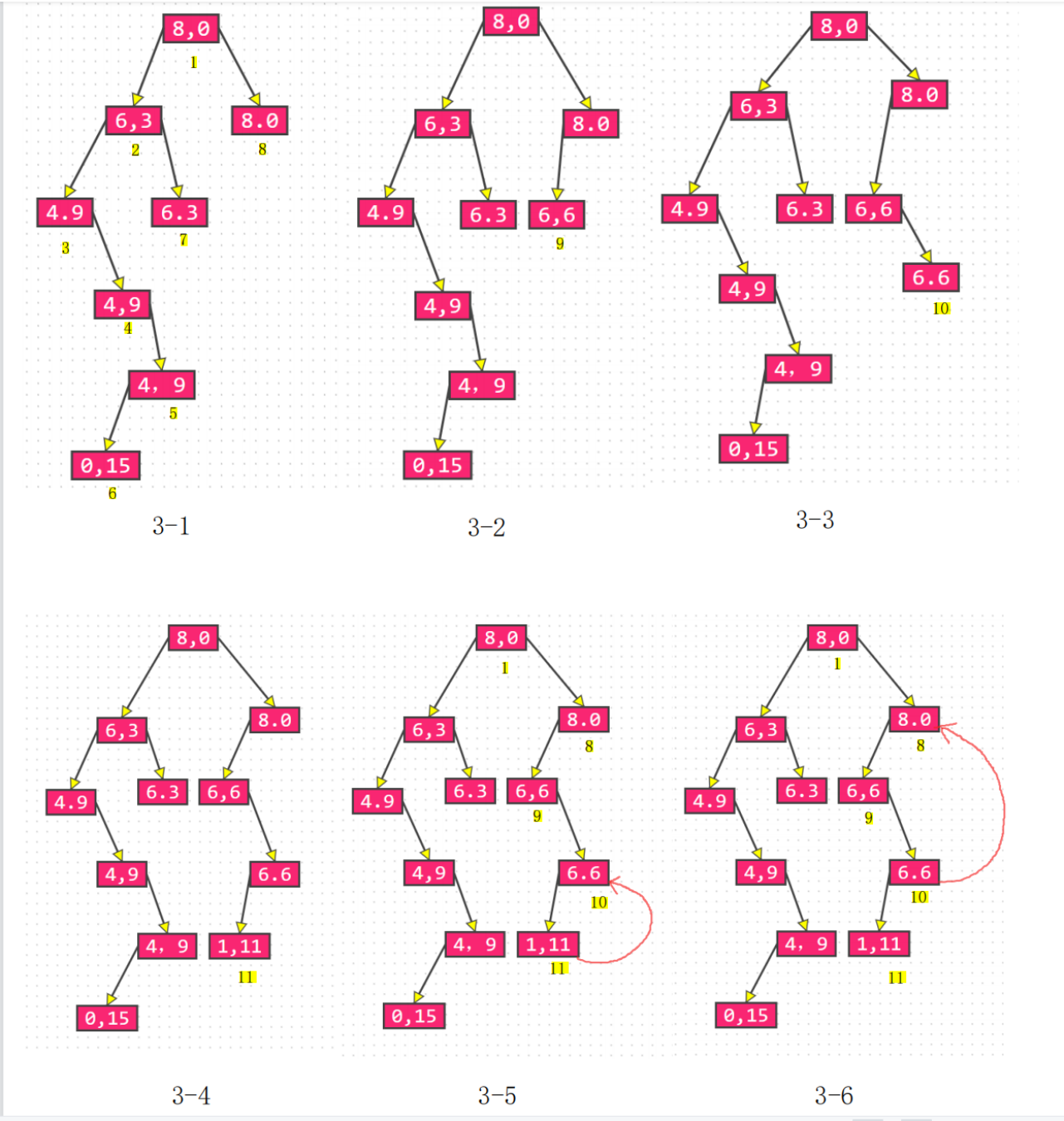
扩展节点9继续向左分支扩展，判断约束条件，此时背包剩余容量为6，物品3的重量为7，6<7，不满足约束条件，故剪去节点9的左分支，节点9继续向右分支扩展，判断限界条件，cp=6，rp=11，bestp=15，cp+rp>bestp，满足限界条件，故节点10成为活节点并成为当前结点的扩展结点。

节点10继续沿着左分支扩展，判断约束条件，此时背包剩余容量6，物品4重量为5，6>5，满足约束条件，节点11成为活节点并成为当前结点的扩展结点。

结点11继续沿着左分支扩展，判断约束条件，此时背包剩余容量为1，物品5总量为4，1<4，不满足约束条件，故剪去节点11的左分支，结点11继续沿着右分支扩展，判断限界条件此时cp=11，rp=0，bestp=15，cp+rp<bestp，不满足限界条件,故剪去节点11的右分支。节点11的左右分支搜索完毕，回溯到最近的活节点10，节点10又成为扩展节点。

节点10继续沿着右分支扩展，判断限界条件，cp=6，rp=6，bestp=15，cp+rp<bestp，不满足限界条件，故剪去节点10的右分支，如图所示。节点10的左右分支搜索完毕，回溯到最近的活节点8，节点8成为扩展节点。

节点8继续沿着右分支扩展，判断限界条件，cp=0，rp=15，cp+rp=bestp，不满足限界条件，故剪去节点8的右分支，此时节点8的左右分支搜索完毕，回到节点1，节点1成为扩展节点。



扩展节点1两个分支搜索完毕，它成为死结点，搜索结束。找到问题的最优路径(1,1,0,0,1)，即第一件第二件，第五件物品装入背包，物品总价值为15。

算法的改进

针对剩余物品，根据物品的单位重量价值，将单位价值高的优先装满背包，将背包剩余容量装满的值即为brp

$$\frac{3}{2}=1.5, \qquad \frac{6}{2}=3, \qquad \frac{4}{7}=0.57, \qquad \frac{5}{5}=1, \qquad \frac{6}{4}=1.5$$

将限界条件改为：

$$cp+brp>bestp$$

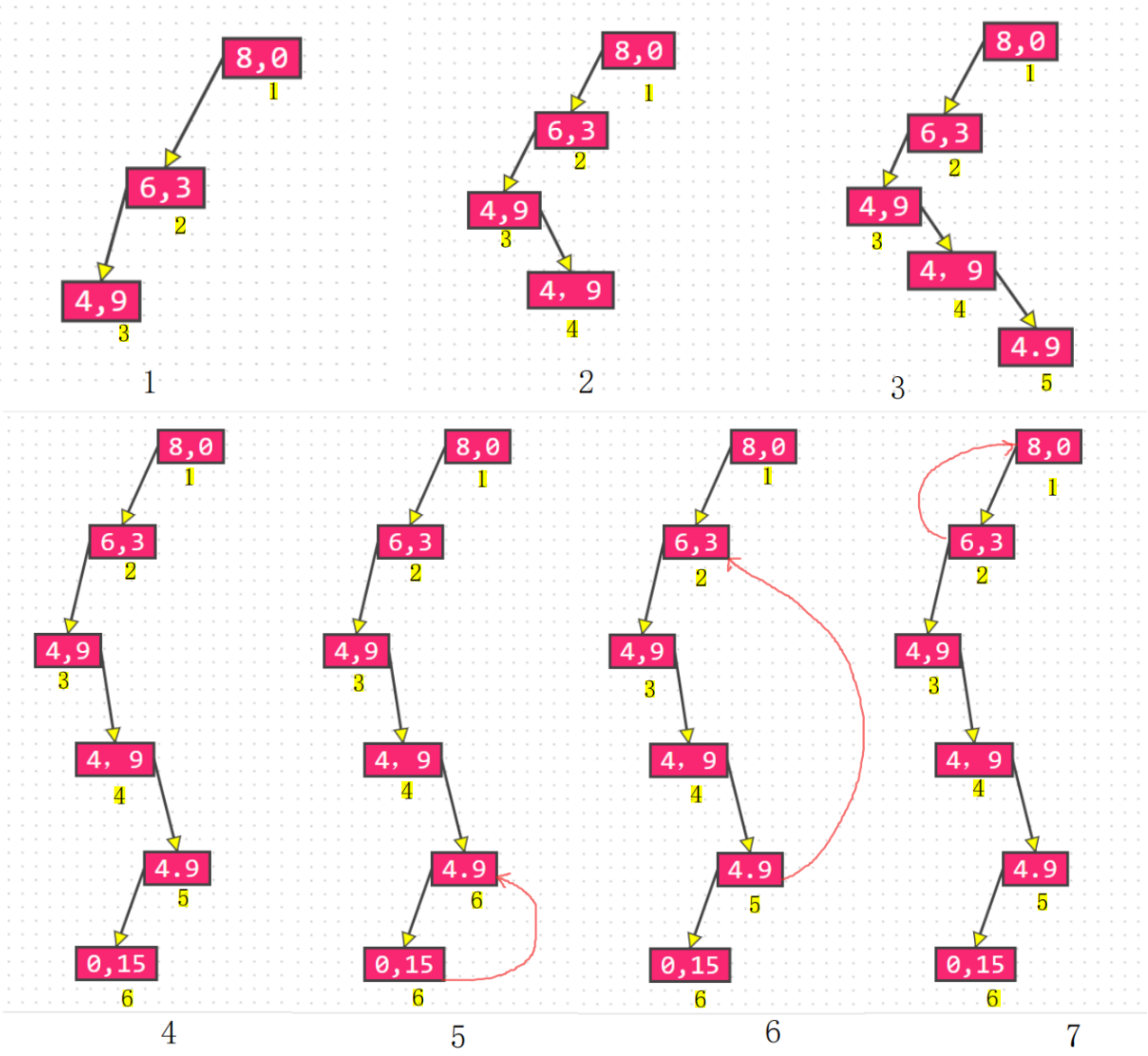
搜索过程：

前面几步搜索过程都一样，在搜寻右分支时判断的限界条件不一样

搜寻到节点3，节点3成为活节点，并且是当前的扩展节点，如图一所示。

扩展节点3继续沿着左分支扩展，此时需要判断第三件物品是否能装入背包，第三件物品重量为7，7>4，显然第三件物品装不进去，故减掉扩展节点2的左分支，此时需要扩展节点沿着右分支继续扩展，判断限界条件，cp=9，brp=6，bestp=0，cp+brp>bestp，限界条件成立，则节点3沿右分支扩展的节点4成为活节点，并成为当前的扩展节点，扩展节点4，代表背包剩余容量为4，背包内物品总价值为9，如图2所示。

节点4沿着左分支扩展，此时需要判断第四件物品是否能装入背包，第四件物品重量为5，5>4，显然第四件物品装不进去，故减掉扩展节点4的左分支，此时需要扩展节点沿着右分支继续扩展，判断限界条件 ，此时cp=9，brp=6，bestp=0，cp+rp>bestp，限界条件成立，则节点4沿右分支扩展的节点5成为活节点，并成为当前的扩展节点，扩展节点5，代表背包剩余容量为4，背包内物品总价值为9，如图3所示。



节点5继续沿着左分支扩展，此时需要判断第五件物品是否可以装入背包，第五件物品重量为4，4=4，可以刚好装进，节点6满足约束条件，此时节点6以经是叶子节点，故找到一个当前最优解，记录修改bestp的值，bestp=15，当前最优解为装入背包的总价值为15，如图4所示。

由于节点6已经是叶子节点，不再具备扩展能力，此时要回溯到离节点最近的节点5，节点5再次成为扩展节点，如图5所示。扩展节点沿着右分支继续扩展，此时要判断是否满足限界条件，cp=9，brp=0，bestp=15，显然不满足条件，故剪去5的右分支，此时节点3的左右分支均搜索完毕。

再回溯到最近的活节点2，节点2再次成为扩展节点，如图6所示。节点2继续沿右分支扩展，此时要判断是否满足限界条件，cp=3，brp=(6+2/5)，cp+rp=9.4<bestp，所以不满足限界条件，故剪去2的右分支，此时节点2的左右分支均搜索完毕。

再回溯到最近的活节点1，节点1再次成为扩展节点，如图7所示。节点1继续沿右分支扩展，此时要判断是否满足限界条件，cp=10，brp=(6+6+2/5)，cp+rp=12.4<bestp，所以不满足限界条件，故剪去1的右分支，此时节点1的左右分支均搜索完毕。

找到当前的最优解为(1,1,0,0,1)，即第一件第二件，第五件物品装入背包，最优值为15。

规模趋于**n**的时候，时间复杂度

判断约束函数需要O(1)，在最坏情况下有 2n-1个左孩子，约束函数耗时最坏为 $O(2^n)$。计算上界 限界函数需要O(n)时间,在最坏情况下有2^n-1个右孩子需要计算上 界，限界函数耗时最坏为$O(n2^n)$。 0-1背包问题的回溯算法所需的计算时间为$O(2^n)+ O(n2^n)=0(n2^n)$

分支限界法

解题基本步骤

1. 定义问题的解空间
2. 确定问题的解空间组织结构
3. 搜索解空间

确定问题的解空间

$(X_1，X_2，X_3，X_4，X_5)$ X_i=0或1

分支限界法求解**0-1**背包问题

给定n种物品和一背包。物品i的重量是W_i，其价值为V_i，背包容量为C 。一个物品要么全部装入背包，要么全部不装入背包，不允许部分装如背包，装入背包的物品的总重量不超过背包的容量。 n=5，C=8，W={2，2，7，5，4}，V={3，6，4，5，6}，问应该如何选择装入背包的物品，使得装入背包中的物品总价值最大。

队列式分支限界法

了解分支限界法，分支限界法类似于回溯法，也是一种在问题的解空间树中搜索问题解得一种算法，它常常以广度优先或最小耗费(最大效益)优先的方式搜索问题的解空间树。

和回溯法类似先定义问题的约束条件和限界条件

约束条件：

$$\sum_{i=1}^{n} W_i X_i <= W$$

限界条件：

cp表示当前已经装入背包的物品总价值，初始值为0；r'p表示剩余物品装入剩余背包容量能装入的最优值，初始值为所有物品的价值之和；bestp表示当前的最优解，初始值为0，当cp+r'p>bestp时更新bestp为cp。

$$cp + r'p > bestp$$

搜索过程：

深色的表示死结点，不在活节点列表中。

如下图，将根节点插入活结点表中，节点A是唯一的活结点，如图1所示。

从活结点表中取出A，节点A是当前的扩展节点，一次性生成它的两个孩子节点，B和C，结点B满足约束条件(8>2)，将bestp改为B结点的cp，即bestp=3，对于结点C，由于cp=0，r'p=24，bestp=3，cp+r'p>bestp，满足限界条件，依次将B，C插入到活节点列表中，A节点成为死结点，如图2所示。

从活节点列表中取出B，节点B是当前节点的扩展节点，一次性生成两个孩子节点，D和E，节点E满足约束条件(6>2)，将bestp更新为D节点的cp，即bestp=9，对于结点E，由于cp=3，r'p=15，bestp=9，cp+r'p>bestp，满足限界条件，依次将D，E插入活结点列表中，B节点成为死结点，如图3所示。

从活节点列表中取出C，节点C是当前节点的扩展节点，一次性生成两个孩子节点，F和G，节点F满足约束条件(8>2)，此时F节点的cp<bestp故不更新bestp，即bestp=9，对于结点G，由于cp=0，r'p=15，bestp=9，cp+r'p>bestp，满足限界条件，依次将D，E插入活结点列表中，C节点成为死结点，如图4所示。

从活节点列表中取出D，节点D是当前节点的扩展节点，一次性生成两个孩子节点，H和I，节点H不满足约束条件(4<7)，故将其舍弃，此时的bestp的值还是9，对于结点I，由于cp=9，r'p=11，bestp=9，cp+r'p>bestp，满足限界条件，故将I节点插入活结点列表中，D节点成为死结点，如图5所示。

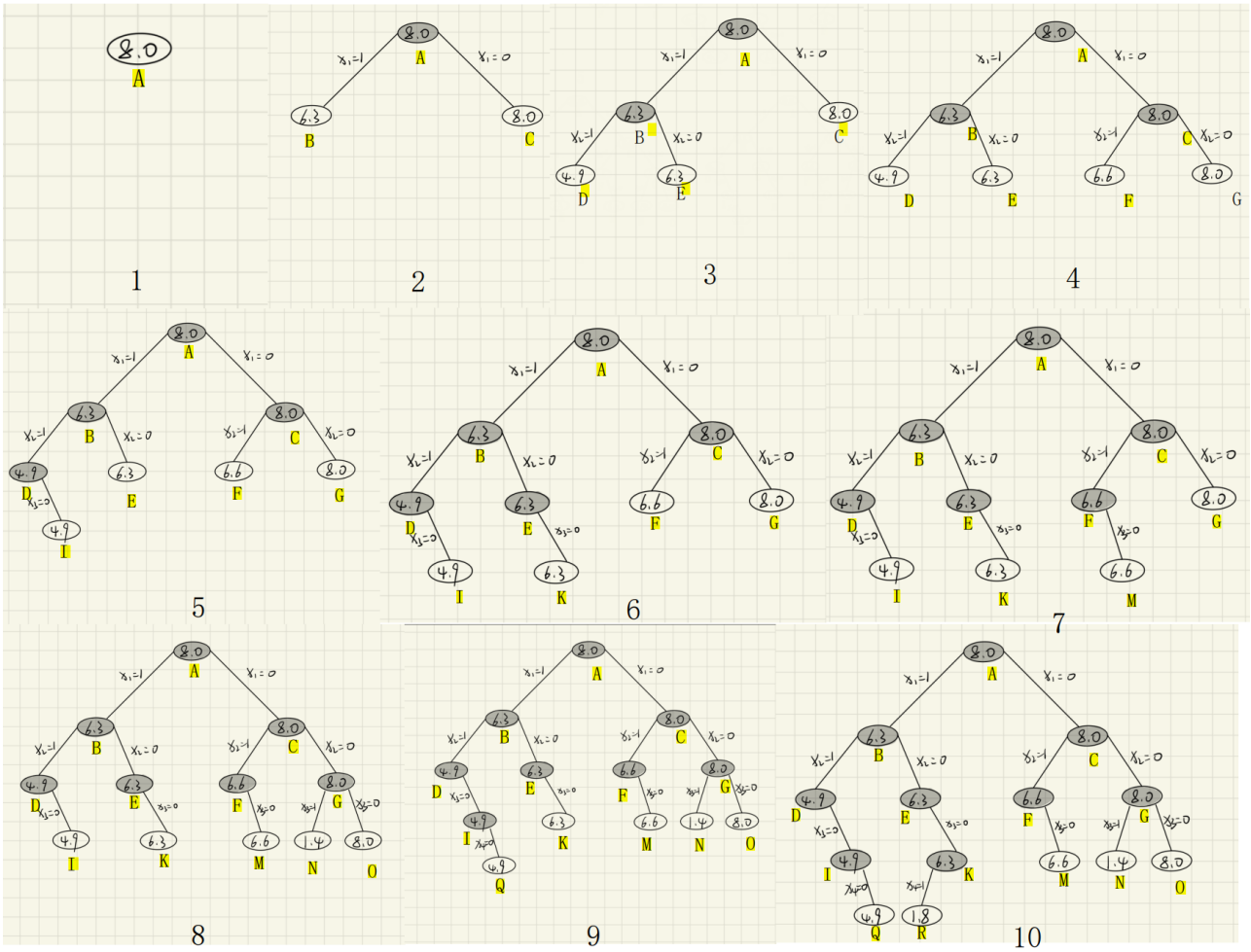
从活节点列表中取出E，节点E是当前节点的扩展节点，一次性生成两个孩子节点，J和K，节点J不满足约束条件(6<7)，故将其舍弃，此时的bestp的值还是9，对于结点K，由于cp=3，r'p=11，bestp=9，cp+r'p>bestp，满足限界条件，故将K节点插入活结点列表中，E节点成为死结点，如图6所示。

从活节点列表中取出F，节点F是当前节点的扩展节点，一次性生成两个孩子节点，L和M，节点L不满足约束条件(6<7)，故将其舍弃，此时的bestp的值还是9，对于结点M，由于cp=6，r'p=11，bestp=9，cp+r'p>bestp，满足限界条件，故将M节点插入活结点列表中，F节点成为死结点，如图7所示。

从活节点列表中取出G，节点G是当前节点的扩展节点，一次性生成两个孩子节点，N和O，节点N满足约束条件(8>7)，此时N节点的cp=4<bestp故不更新bestp，即bestp=9，对于结点O，由于cp=0，r'p=11，bestp=9，cp+r'p>bestp，满足限界条件，依次将O，N插入活结点列表中，G节点成为死结点，如图8所示。

从活节点列表中取出I，节点I是当前节点的扩展节点，一次性生成两个孩子节点，P和Q，节点Q不满足约束条件(4<5)，故将其舍弃，此时的bestp的值还是9，对于结点Q，由于cp=9，r'p=6，bestp=9，cp+r'p>bestp，满足限界条件，故将Q节点插入活结点列表中，I节点成为死结点，如图9所示。

从活节点列表中取出K，节点K是当前节点的扩展节点，一次性生成两个孩子节点，R和S，节点R满足约束条件(6>5)，此时R节点的cp=8<bestp，故不更新bestp，即bestp=9，对于结点S，由于cp=3，r'p=6，bestp=9，cp+r'p=bestp，不满足限界条件，将其剪去，故将R节点插入活结点列表中，K节点成为死结点，如图10所示。



从活节点列表中取出M，节点M是当前节点的扩展节点，一次性生成两个孩子节点，S和T，节点S满足约束条件(6>5)，此时S节点的cp=11,将bestp更新为S节点的cp，即bestp=11 ，对于结点T，由于cp=6，r'p=6，bestp=11，cp+r'p>bestp，满足限界条件，依次将S，T插入活结点列表中，M节点成为死结点，如图11所示。

从活节点列表中取出N，节点N是当前节点的扩展节点，一次性生成两个孩子节点，左节点不满足约束条件(1<5)，舍去，对于右，由于cp=4，r'p=6，bestp=11，cp+r'p<bestp，不满足限界条件，故将其舍去，N节点成为死结点，如图12所示。

从活节点列表中取出O，节点O是当前节点的扩展节点，一次性生成两个孩子节点，左节点W满足约束条件(8>5)，此时W节点的cp=5<bestp，不更新bestp，即bestp=11 ，对于右结点，由于cp=0，r'p=6，bestp=11，cp+r'p<bestp，不满足限界条件，故将其舍去，将W插入活结点列表中，O节点成为死结点，如图13所示。

从活节点列表中取出Q，节点Q是当前节点的扩展节点，一次性生成两个孩子节点，左节点Y满足约束条件(4=4)，此时Y节点的cp=15>bestp，故更新bestp，即bestp=15 ，对于右结点，由于cp=9，r'p=0，bestp=15，cp+r'p<bestp，不满足限界条件，将其剪去，故将Y节点插入活结点列表中，Q节点成为死结点，如图14所示。

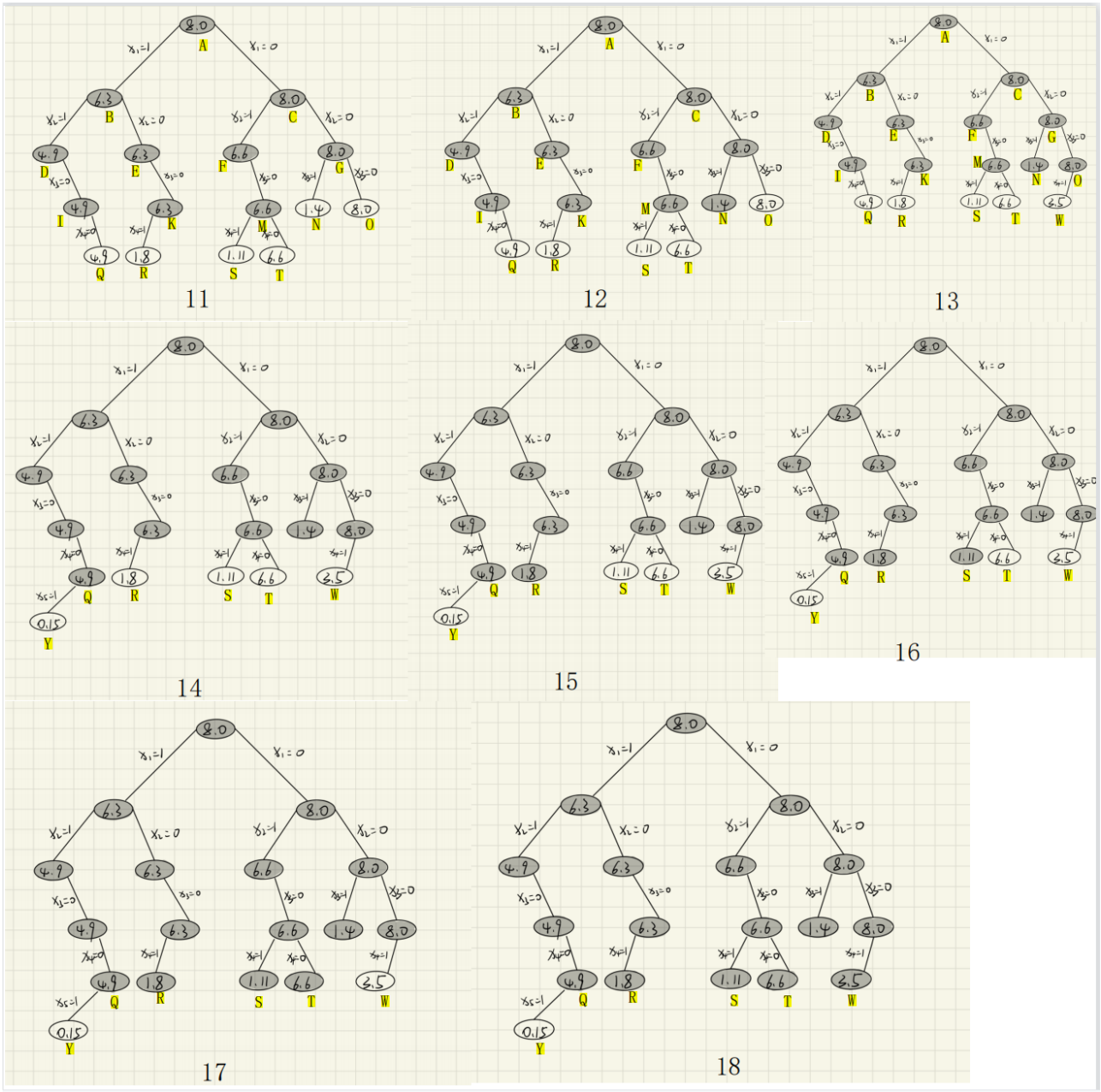
从活节点列表中取出节点R，节点R一次性生成两个孩子节点，左孩子节点不满足约束条件，右孩子不满足限界条件，故将其两个孩子节点舍去，节点R成为死节点。如图15所示。

同理从活节点列表中取出节点S，节点S一次性生成两个孩子节点，左孩子节点不满足约束条件，右孩子不满足限界条件，故将其两个孩子节点舍去，节点S成为死节点。如图16所示。

从活节点列表中取出节点T，节点T一次性生成两个孩子节点，左孩子节点不满足约束条件，右孩子不满足限界条件，故将其两个孩子节点舍去，节点T成为死节点。如图17所示。

从活节点列表中取出节点W，节点W一次性生成两个孩子节点，左孩子节点不满足约束条件，右孩子不满足限界条件，故将其两个孩子节点舍去，节点W成为死节点。如图18所示。

此时活节点列表为空，算法结束，找到了问题的最优解，即从根节点A出发到叶子节点Y的路径，(1,1,0,0,1),最优值为15。



优先队列式分支限界法

优先队列分支限界法与队列式分支限界法有所不同，它会有一个优先级定义

优先级定义：活结点代表的部分所描述的装入背包的物品价值上界，该上界越大，优先级越高。活结点的上界up=活结点的cp+剩余物品装满背包剩余容量的最优值r'p，这里的r'p和队列分支限界法的有所不同。

优先队列分支限界法的约束条件与队列式分支限界法约束条件相同：

$$\sum_{i=1}^{n} W_i X_i <= W$$

限界条件：

$$up = cp + r'p > bestp$$

初始时，将根节点A插入到活结点列表中，节点A是唯一的活结点，如图1所示。

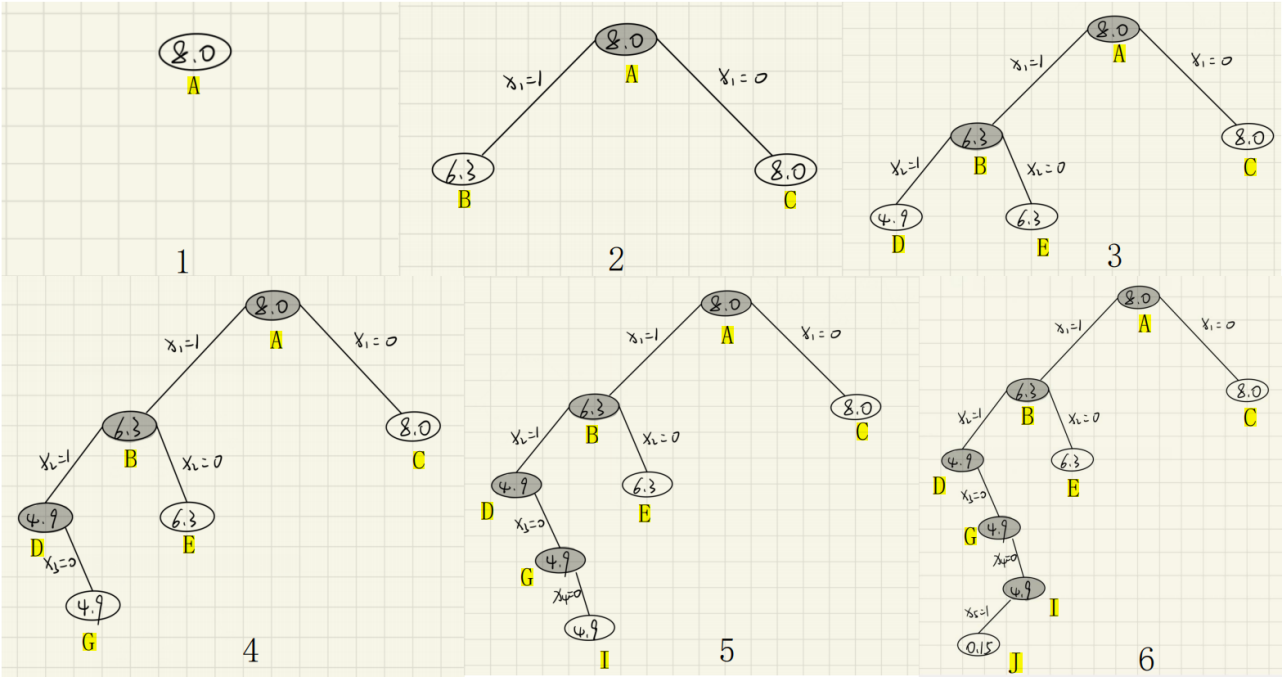
从活结点列表中选出A，节点A是点前的扩展节点，一次性生成两个孩子节点，B和C，节点B满足约束条件(8>2)，将节点B插入活节点列表中，其中B节点的up=cp+r'p=3+6+6=15，bestp=0<cp=3，更新bestp=3。 节点C的up=cp+r'p=0+6+6+(2/5)=12.4；节点C的up>bestp，故满足限界条件，将C节点插入活解点列表中。如图2所示.

选出活解点列表中优先级最的活节点(即up最大的)B，作为当前的扩展节点，一次性生成两个孩子节点，D和E，节点D满足约束条件(6>2)；将活节点D插入活节点列表中，此时节点D的up=cp+r'p=9+6=15，cp=9，更新bestp=9。 节点E的up=cp+r'p=3+6+(2/5)=9.4；节点E的up>bestp，满足限界条件，将节点E插入活节点列表中。如图3所示。

选出活解点列表中优先级最的活节点(即up最大的)D，作为当前的扩展节点，一次性生成两个孩子节点，F和G，节点F不满足约束条件(4<7)，故舍去；bestp还是9。 节点G的up=cp+r'p=9+6=15，节点G的up>bestp满足限界条件，将节点G插入活节点列表中，如图4所示。

选出活解点列表中优先级最的活节点(即up最大的)G，作为当前的扩展节点，一次性生成两个孩子节点，H和I，节点H不满足约束条件(4<5)，故舍去；bestp还是9。 节点I的up=cp+r'p=9+6=15，节点I的up>bestp满足限界条件，将节点I插入活节点列表中，如图5所示。

选出活解点列表中优先级最的活节点(即up最大的)I，作为点前节点的扩展节点，一次生成两个孩子，J和K，节点J满足约束条件(4=4),将解J插入活节点列表中，此时节点J的up=cp+r'p=15+0=15，节点J的cp=15>bestp，更新bestp的值为15。 节点K的up=cp+r'p=9+0=9，节点K的up<bestp不满足限界条件，故舍去。



此时节点J已经是叶子节点，搜索结束找到，找到问题的最优解，即从根节点到叶子节点的路径为(1,1,0,0,1),bestp=15。

规模趋于**n**的时候，时间复杂度

限界函数时间复杂度为O(n)，而最坏情况有$2^{(n+1)}-2$个节点，若 对每个节点用限界函数判断，则其时间复杂度为$O(n2^n)$.而算法中时间复杂度主要依赖限界函数，则算法的时间复杂度为$O(n2^n)$。

三种算法的区别和相同点

动态规划法：

是将待求解问题分解成若干个相互重叠的子问题，每个子问题对应决策过程的一个阶段，一般来说，子问题的重叠关系表现在对给定问题求解的递推关系（也就是动态规划函数）中，将子问题的解求解一次并填入表中，当需要再次求解此子问题时，可以通过查表获得该子问题的解而不用再次求解，从而避免了大量重复计算。

回溯法：

回溯法就是一种有组织的系统化搜索技术，可以看作是蛮力法穷举搜索的改进。回溯法每次只构建可能解的一部分，然后评估这个部分解，如果这个部分有可能导致一个完全解，对其进一步搜索，否则，就不必继续构造这部分的解了，回溯法常常可以避免搜索所有可能的解，所以，它往往比满立法效率更高，适用于求解组合数组较大的问题。

分支限界法：

分支限界法按广度优先策略遍历问题的解空间，在遍历过程种，对已经处理的每一个结点根据限界函数估算目标函数的可能值，从中选取使目标函数取得极值（极大或极小）的结点优先进行广度优先搜索，从而不断调整搜索方向，尽快找到问题的解。因为界限函数常常是基于问题的目标函数而确定的，所以，分支限界法适用于求解最优化问题。

首先是分支限界法和回溯法

相同点：

1. 均需要先定义问题的解空间,确定的解空间组织结构一般都是树或图。
2. 在问题的解空间树上搜索问题解。
3. 搜索前均需确定判断条件,该判断条件用于判断扩展生成的节点是否为可行节点。
4. 搜索过程中必须判断扩展生成的节点是否满足判断条件，如果满足,就保留该扩展生成的节点;否则，舍弃。

不同点：

1. 求解目标不同： 回溯法的求解目标是找出空间树中满足约束条件的所有解。 分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出（使某一目标函数值达到极大或极小的解，即在）某种意义下的最优解。
2. 搜索方式不同： 回溯法:深度优先遍历结点搜索解空间树。 分支限界法:广度优先或最小耗费有限搜索解空间树。
3. 存储空间不同： 分支限界法由于加入了活结点表，所以存储空间比回溯法大得多。因此当内存容量有限时，回溯法的成功率要大一些。 回溯一般使用堆栈，而分支限界使用队列或优先队列.
4. 扩展结点的方式不同： 回溯法中，活结点的所有可行子结点被遍历后才从栈中弹出;分支限界中，每个活结点只有一次机会变成扩展结点，一旦成为扩展结点便一次性生成其所有子结点。

区别小结：回溯法空间效率更高，分支限界法由于只需要求到一个解，所以往往更“快”。

动态规划法和回溯法

相同点：

在求解背包问题中定义的约束条件类似。都能的得到最优解。时间复杂度都比较高。都可以通过改进算法减小时间复杂度

不同点：

回溯法只需要搜索一次就可以判断问题的结果是否正确。动态规划法，问题与问题之间存在关联，也就是常说的子问题，需要记录子问题的一些结果。

动态规划是自底向上的求解问题，回溯法的关键就是回溯，并不是自底向上的搜索。

他们的时间复杂度和空间复杂度都不相同。

总结，动态规划法是过程主义；回溯法是结果主义。

动态规划法和分支限界法

相同点：

动态规划法和分支限界法对于0-1背包问题的约束条件相同，目标相同都是求解问题的最优解。都能的得到最优解。时间复杂度都比较高。

不同点：

动态规划法将一个问题拆解成许多个相互重叠的子问题，后面再次遇到相关的子问题使用前面子问题的解就行了无需再次求解，从而省去大量计算。分支限界法则是采用广度优先的搜索方式搜索解空间，通过约束条件和限界条件来对问题进行剪枝，来减少计算。

动态规划法可以找出问题的所有最优解，分支限界法的求解目标则是找出满足约束条件的一个解。

贪心算法是否可以求解**0-1**背包问题

贪心算法是指在求解问题时，总在做出当前看来最好的选择，也就是说，不从整体最优上加以考虑，算法得到的是在某种意义上的局部最优解。而0-1背包问题时寻找全局最优解。故贪心算法不能用来解决0-1背包问题。

贪心算法求解0-1背包问题基本步骤

1.计算每种物品单位重量的价值 V_i/W_i 2.依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。 3.若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包。 4.依此策略一直进行下去，直到背包装满为止。

对于0-1背包问题，贪心选择之所以不能得到最优解,是因为在这种情况下，无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此导出许多互相重叠的子问题。