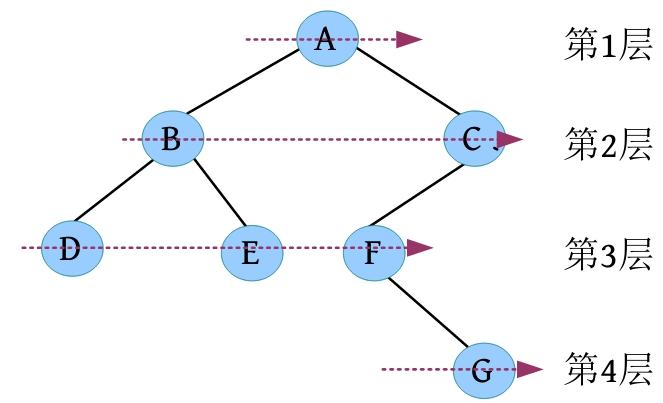
**广度优先搜索：分支限界法**

在图的应用中已讲过图的广度优先搜索，树上的广度优先搜索实际上就是层次遍历。首先遍历第1层，然后第2层……同一层按照从左向右的顺序访问，直到最后一层。一棵树如下图所示，首先遍历第1层A；然后遍历第2层，从左向右遍历B、C；再遍历第3层，从左向右遍历D、E、F；再遍历第4层G。



**分支限界法**通常**以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树**。首先将根节点加入活节点表中，接着从活节点表中取出根节点，使其成为当前扩展节点，一次性生成其所有孩子节点，判断对孩子节点是舍弃还是保留，**舍弃那些得不到可行解或最优解的节点**，将其余节点保留在活节点表中。再从活节点表中取出一个活节点作为当前扩展节点。重复上述过程，直到找到所需的解或活节点表为空时为止。每一个活节点最多只有一次机会成为扩展节点。

**活节点表的实现通常有两种形式**：一种是**普通的队列**，即**先进先出队列**；另一种是**优先级队列**，按照某种优先级决定哪个节点为当前扩展节点。

根据活节点表的不同，分支限界法分为以下两种：**队列式分支限界法**和**优先队列式分支限界法**。

分支限界法的解题过程如下：

**（1）定义解空间。**解空间的大小对搜索效率有很大的影响，首先要定义合适的解空间，确定解空间包括解的组织形式和显约束。解的组织形式规范为一个**n元组{x1,x2,…,xn}**，具体问题表达的含义不同。**显约束**是对**解分量的取值范围**的限定。

**（2）确定解空间的组织结构**。对解空间的组织结构通常用解空间树形象地表达，根据解空间树的不同，解空间分为**子集树**、**排列树**、**m叉树**等。

**（3）搜索解空间。**分支限界法指按照**广度优先搜索策略**，一次性生成所有孩子节点，**根据约束函数和限界函数判定对孩子节点是舍弃还是保留**，如果保留，则将其依次放入活节点表中，活节点表是普通队列或优先队列。然后从活节点表中取出一个节点，继续扩展，直到找到所需的解或活节点表为空时为止。如果对该问题只求**可行解**，则只需设定**约束函数即可**；如果求**最优解**，则需要设定**约束函数和限界函数**。

在优先队列分支限界法中还有**一个关键问题**，即**优先级的设定**：**选择什么值作为优先级？如何定义优先级？因为优先级的设计直接决定算法的效率**。

**一、队列式广度优先搜索**

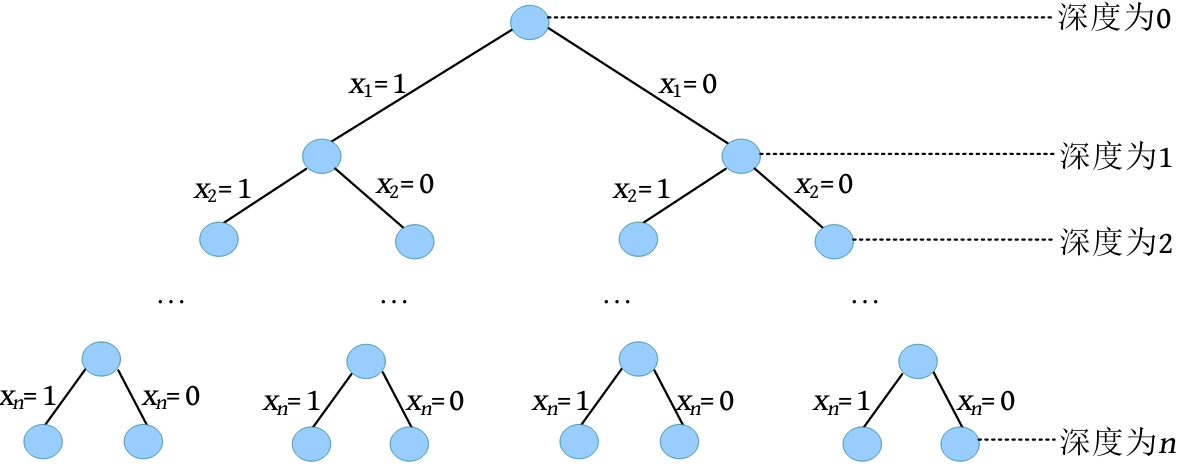
有**n个物品和1个背包**，每个物品i对应的价值都为vi、重量都为wi，背包的容量为W（也可以将重量设定为体积）。每个物品只有一件，要么**装入**，要么**不装入**，**不可拆分**。如何选取物品装入背包，使背包所**装入物品的总价值最大**？

上述问题是典型的**01背包问题**，已经用回溯法求解过，在此先用普通队列式分支界限法求解，然后用优先队列式分支界限法求解，体会这两种算法的不同之处。

**1. 算法设计**

**（1）定义问题的解空间。**背包问题属于典型的01背包问题，问题的解是从n个物品中选择一些物品，使其在不超过容量的情况下价值最大。每个物品都有且只有两种状态：要么被装入背包，要么不被装入背包。那么是第i个物品被装入背包能够达到目标，还是不被装入能够达到目标呢？显然还不确定。因此，可以用变量xi表示第i种物品是否被装入背包的状态，如果用“0”表示不被装入背包，用“1”表示被装入背包，则xi的取值为0或1。第i个物品被装入背包，xi=1；不被装入背包，xi=0。该问题解的形式是一个n元组，且每个分量的取值都为0或1。由此可得，问题的**解空间为{x1,x2,…,xi,…,xn}，其中显约束xi =0或1，i=1,2,3…n。**

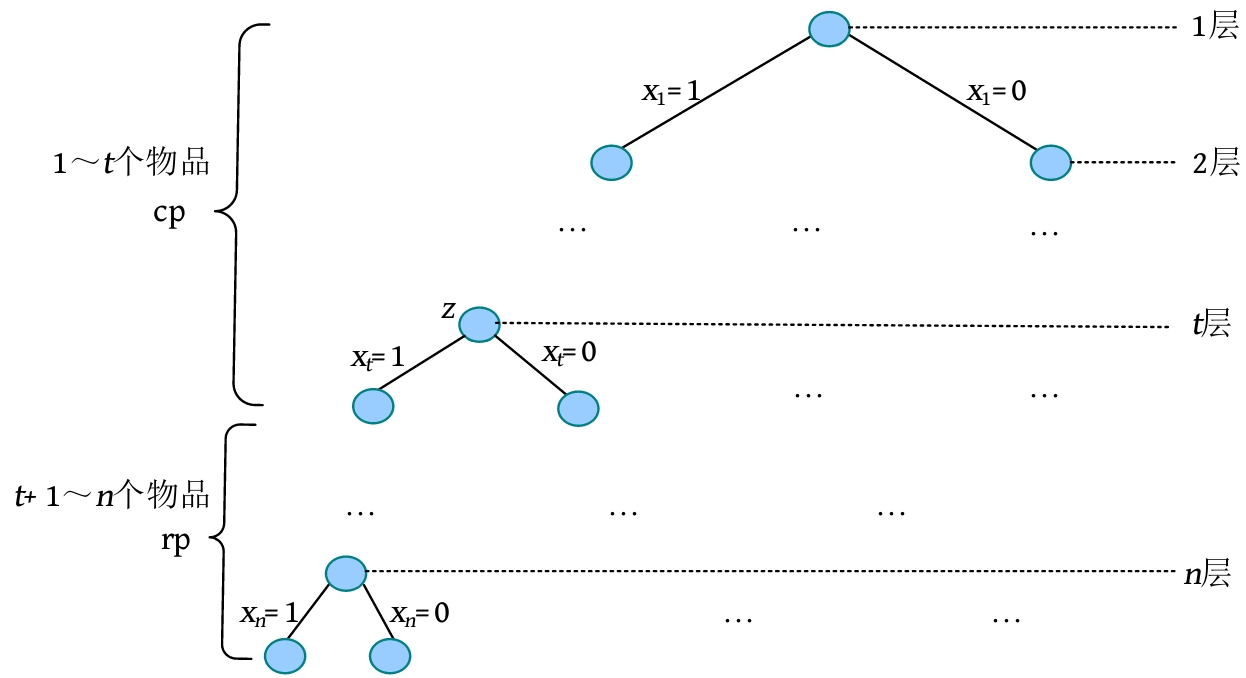
**（2）确定解空间的组织结构。**问题的解空间描述了2n种可能的解，也可以说是n个元素组成的集合的所有子集个数。**解空间树为子集树**，解空间树的深度为问题的规模n，如下图所示。



**（3）搜索解空间。**根据解空间的组织结构，对于任何一个中间节点z（中间状态），从根节点到z节点的分支所代表的状态（是否装入背包）已确定，从z到其子孙节点的分支的状态待确定。也就是说，如果z在解空间树中所处的层次是t，则说明从第1种物品到第t-1种物品的状态已确定，只需沿着z的分支扩展确定第t种物品的状态，前种物品的状态就确定了。在前t种物品的状态确定后，对当前已装入背包的物品的总重量用cw表示，对总价值用cp表示。

**• 约束条件。**判断第i个物品被装入背包后总重量是否超出背包容量，如果超出，则为不可行解；否则为可行解。约束条件为**cw+w[i]≤W**。其中w[i]为第i个物品的重量，W为背包容量。

**• 限界条件。**已装入物品的价值高不一定就是最优的，因为还有剩余物品未确定。目前还不确定第t+1种物品到第n种物品的实际状态，因此只能**用估计值**。假设第t+1种物品到第n种物品都被装入背包，对**第t+1种物品到第n种物品的总价值用rp来表示**，因此**cp+rp**是所有从根出发经过中间**节点z的可行解的价值上界**，如下图所示。



如果价值上界小于当前搜索到的最优值（对最优值用bestp表示，初始值为0），则说明从中间节点z继续向子孙节点搜索不可能得到一个比当前更优的可行解，没有继续搜索的必要；反之，继续向z的子孙节点搜索。

**限界条件为cp+rp≥bestp**。

**注意：回溯法中的背包问题，限界条件不带等号，因为bestp被初始化为0，首次到达叶子时才会更新bestp，因此只要有解，就必然存在至少一次到达叶子。**而在**分支限界法中**，**只要cp>bestp，就立即更新bestp，如果在限界条件中不带等号，就会出现无法到达叶子的情况，比如解的最后一位是0时，例如(1,1,1,0)，就无法找到这个解向量。***因为在最后一位是0时，cp+rp=bestp，而不是cp+rp>bestp，如果限界条件不带等号，就无法到达叶子，得不到解(1,1,1,0)。*该算法均设置了到叶子节点判断更新最优解和最优值。

**搜索过程：**从根节点开始，以**广度优先的方式**进行搜索。根节点首先成为活节点，也是当前扩展节点。一次性生成所有孩子节点，由于在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；由于在子集树中约定右分支上的值为“0”，因此沿着扩展节点的右分支扩展，则代表不装入物品。此时判断是否满足约束条件和限界条件，如果满足，则将其加入队列中；反之，舍弃。然后从队列中取出一个元素，作为当前扩展节点……直到搜索过程队列为空时为止。

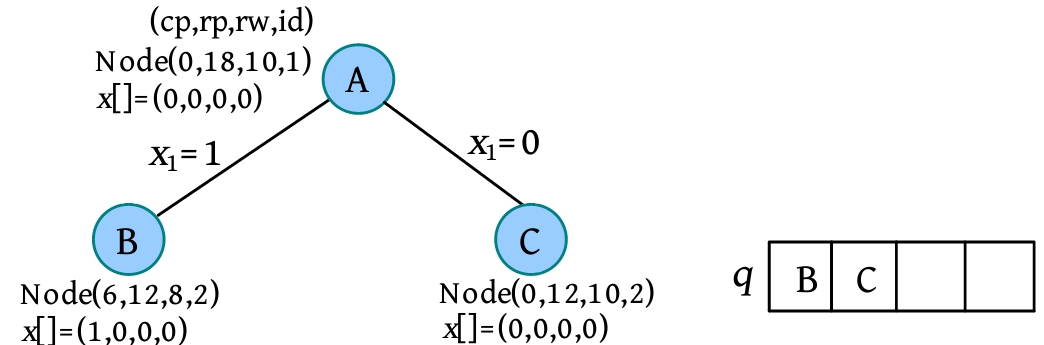
**2.图解**

有**一个背包和4个物品**，每个物品的重量和价值都如下图所示，背包的**容量W=10**。求在不超过背包容量的前提下，把哪些物品放入背包才能获得最大价值。

**（1）初始化。**sumw和sumv分别用来统计所有物品的总重量和总价值。sumw=13，sumv=18，sumw>W，因此不能全部装完，需要搜索求解。初始化当前放入背包的物品价值cp=0，当前剩余物品价值rp=sumv，当前剩余容量rw=W，当前处理物品序号为1且当前最优值bestp=0。解向量x[]=(0,0,0,0)，创建一个根节点Node(cp,rp,rw,id)，将其标记为A并加入先进先出队列q中。cp为装入背包的物品价值，rp为剩余物品的总价值，rw为剩余容量，id为物品号，x[]为当前解向量，如下图所示。

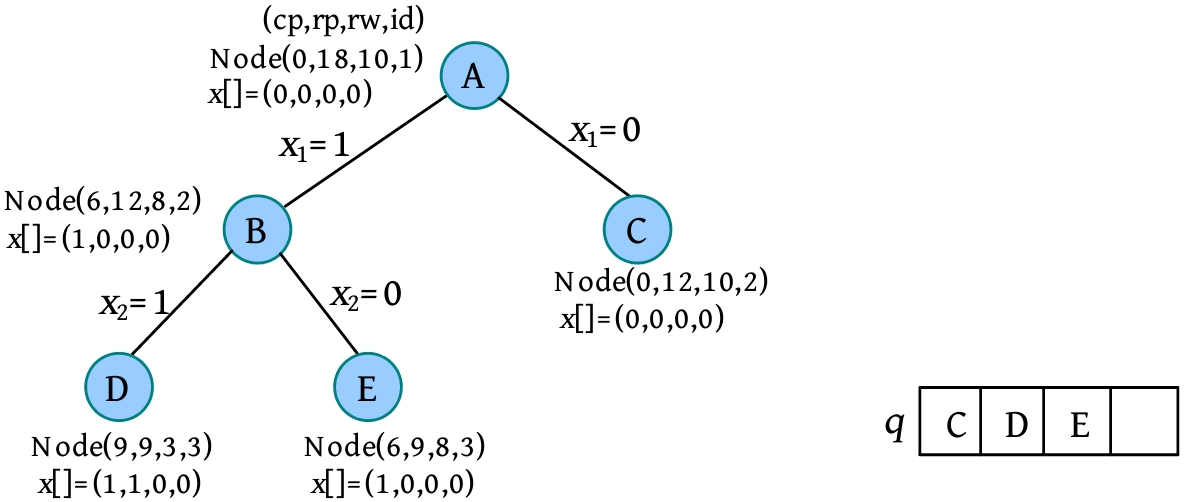


**（2）扩展节点A。**队头元素A出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=10>goods[1].weight=2，剩余容量大于1号物品的重量，满足约束条件，可以被放入背包，cp=0+6=6，rp=18−6=12，rw=10−2=8，t=2，x[1]=1，解向量更新为x[]=(1,0,0,0)，生成左孩子B并将其加入q队列，更新bestp=6。再扩展右分支，cp=0，rp=18−6=12，cp+rp≥bestp=6，满足限界条件，不放入1号物品，cp=0，rp=12，rw=10，t=2，x[1]=0，解向量为x[]=(0,0,0,0)，创建新节点C并将其加入q队列中，如下图所示。

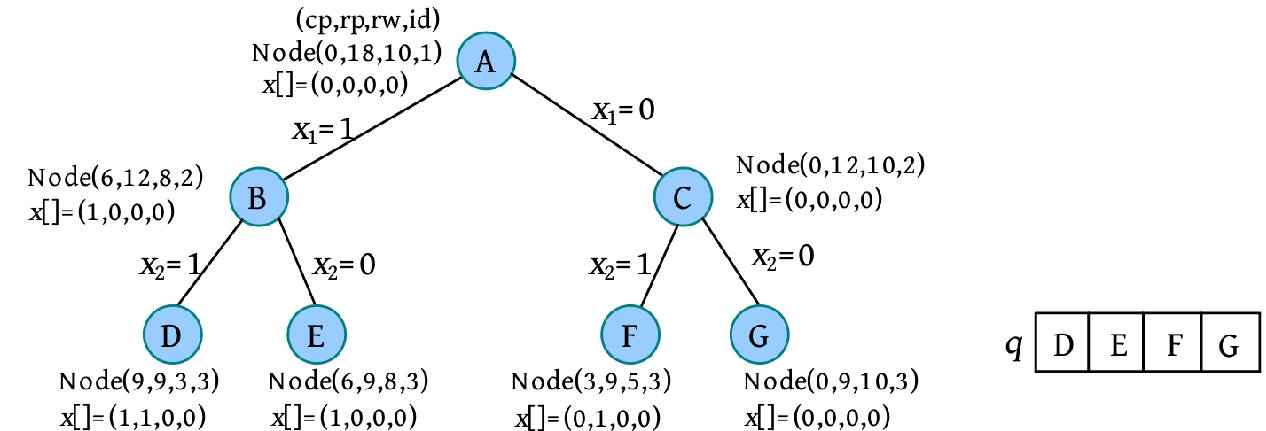


**（3）扩展节点B。**队头元素B出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=8>goods[2]. weight=5，剩余容量大于2号物品的重量，满足约束条件，cp=6+3=9，rp=12−3=9，rw=8−5=3，t=3，x[2]=1，解向量更新为x[]=1,1,0,0)，生成左孩子D并将其加入q队列中，更新bestp=9。

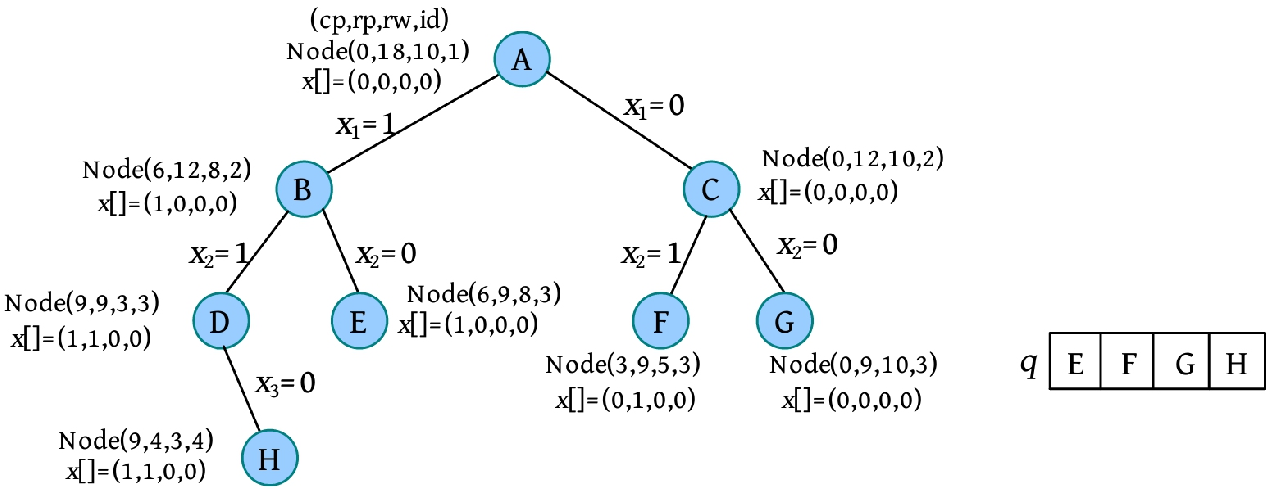
再扩展右分支，cp=6，rp=12−3=9，cp+rp≥bestp=9，满足限界条件，t=3，x[2]=0，解向量为x[]=(1,0,0,0)，生成右孩子E并将其加入q队列中，如下图所示。



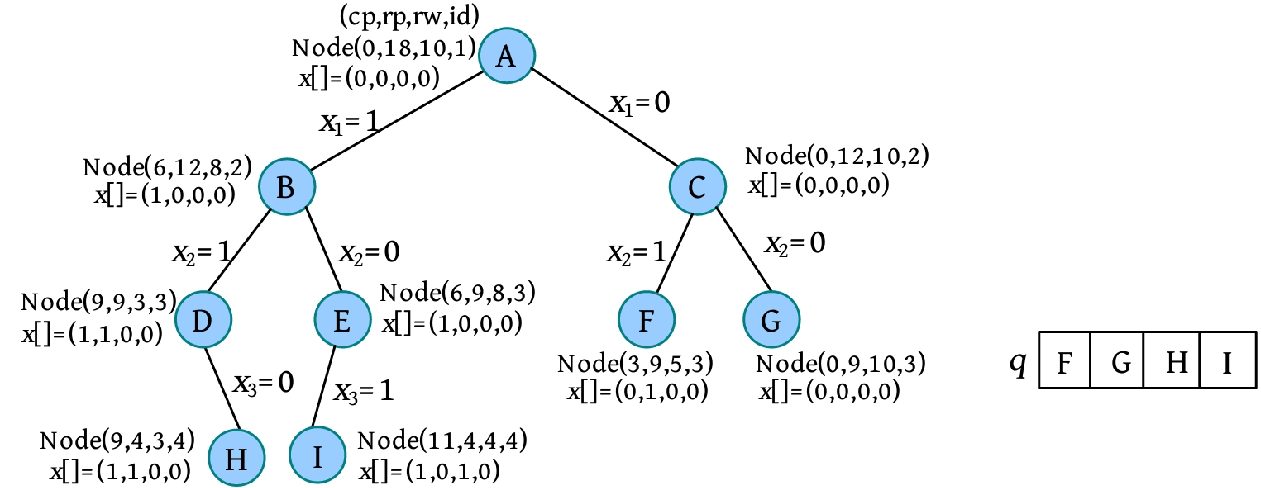
**（4）扩展节点C。**队头元素C出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=10>goods[2].weight=5，剩余容量大于2号物品的重量，满足约束条件，cp=0+3=3，rp=12−3=9，rw=10−5=5，t=3，x[2]=1，解向量更新为x[]=(0,1,0,0)，生成左孩子F并将其加入q队列中。再扩展右分支，cp=0，rp=12−3=9，cp+rp≥bestp=9，满足限界条件，rw=10，t=3，x[2]=0，解向量为x[]=(0,0,0,0)，生成右孩子G并将其加入q队列中，如下图所示。



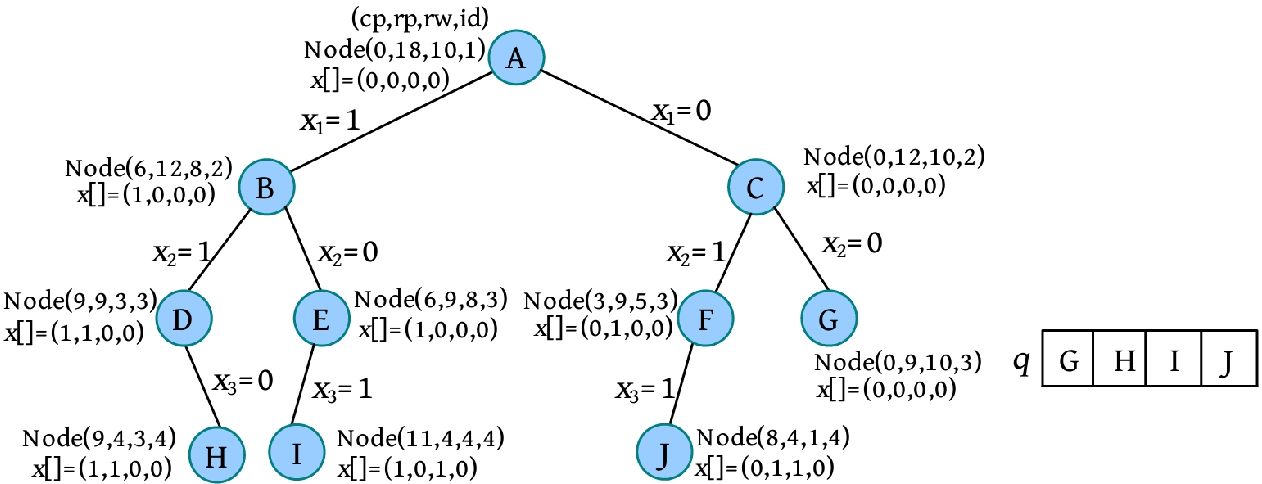
（5）扩展节点D。队头元素D出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=3>goods[3]. weight=4，剩余容量小于3号物品的重量，不满足约束条件，舍弃左分支。再扩展右分支，cp=9，rp=9−5=4，cp+rp≥bestp=9，满足限界条件，t=4，x[3]=0，解向量为x[]=(1,1,0,0)，生成右孩子H并将其加入q队列中，如下图所示。



**（6）扩展节点E。**队头元素E出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=8>goods[3].weight=4，剩余容量大于3号物品的重量，满足约束条件，cp=6+5=11，rp=9−5=4，rw=8−4=4，t=4，x[3]=1，解向量更新为x[]=(1,0,1,0)，生成左孩子I并将其加入q队列中，更新bestp=11。再扩展右分支，cp=6，rp=9−5=4，cp+rp<bestp=11，不满足限界条件，舍弃，如下图所示。

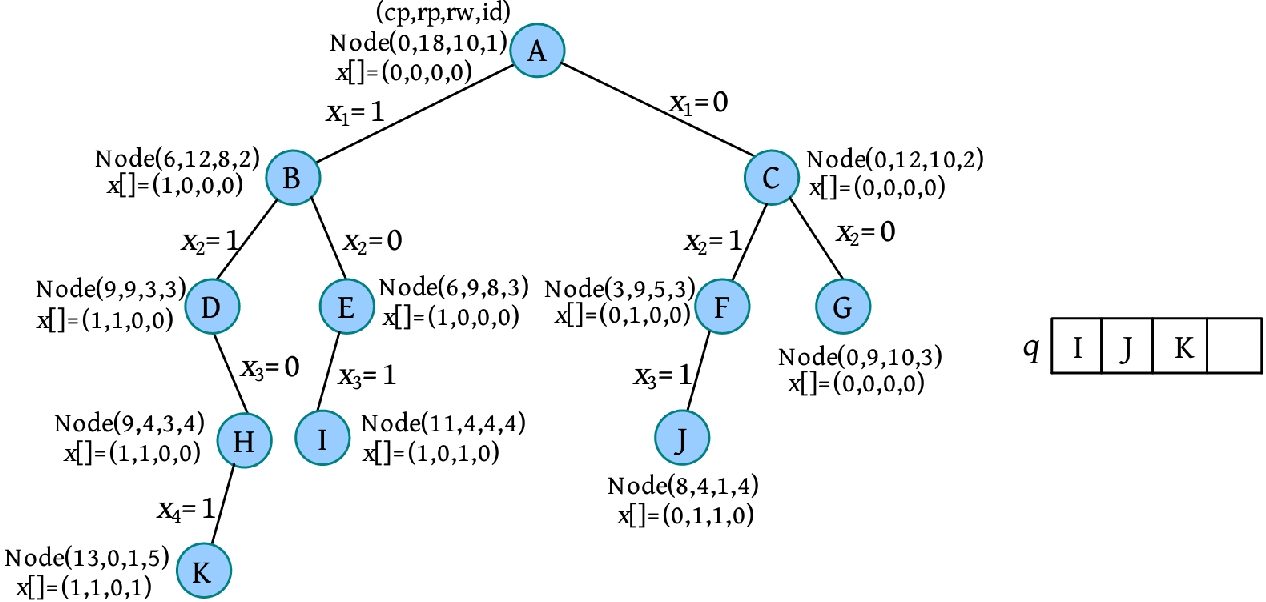


**（7）扩展节点F。**队头元素F出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=5>goods[3].weight=4，剩余容量大于3号物品的重量，满足约束条件，cp=3+5=8，rp=9−5=4，rw=5−4=1，t=4，x[3]=1，解向量更新为x[]=(0,1,1,0)，生成左孩子J并将其加入q队列中。再扩展右分支，cp=3，rp=9−5=4，cp+rp<bestp=11，不满足限界条件，舍弃，如下图所示。

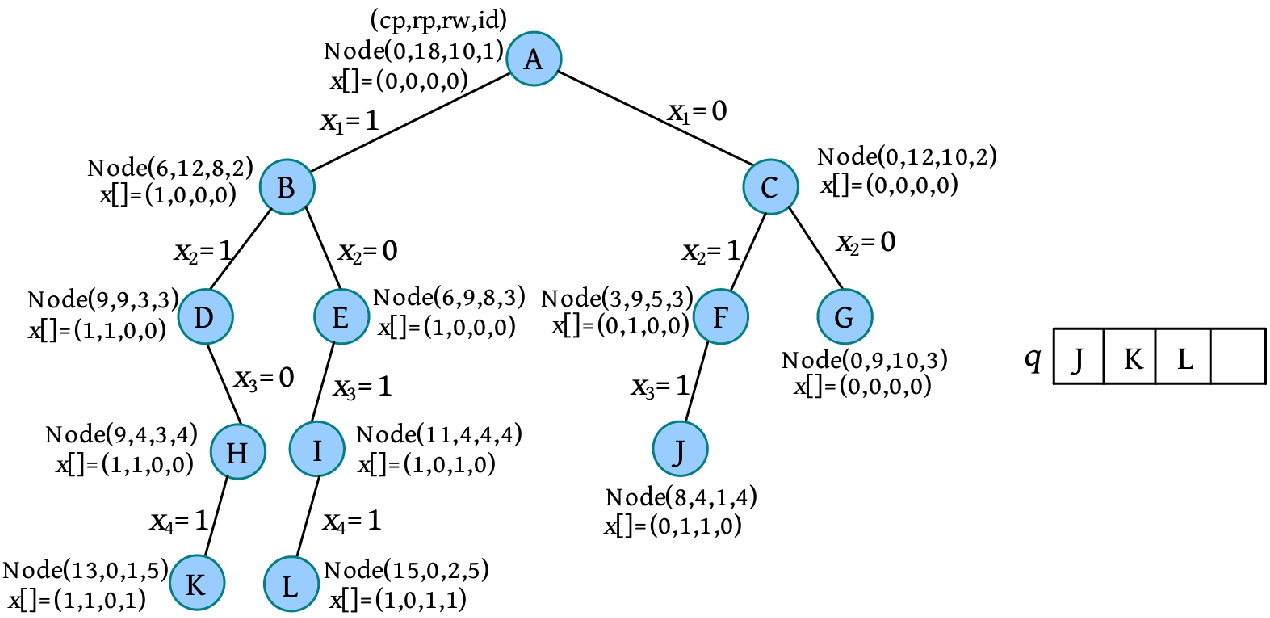


**（8）扩展节点G。**队头元素G出队，该节点的cp+rp<bestp=11，不满足限界条件，不扩展。

**（9）扩展节点H。**队头元素H出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=3>goods[4].weight=2，剩余容量大于4号物品的重量，满足约束条件，令cp=9+4=13，rp=4−4=0，rw=3−2=1，t=5，x[4]=1，解向量更新为x[]=(1,1,0,1)，生成左孩子K并将其加入q队列中，更新bestp=13。再扩展右分支，cp=9，rp=4−4=0，cp+rp<bestp，不满足限界条件，舍弃，如下图所示。

****

**（10）扩展节点I。**队头元素I出队，该节点的cp+rp≥bestp，满足限界条件，可以扩展。rw=4>goods[4].weight=2，剩余容量大于4号物品的重量，满足约束条件，cp=11+4=15，rp=4-4=0，rw=4−2=2，t=5，x[4]=1，解向量更新为x[]=(1,0,1,1)，生成左孩子L并将其加入q队列中，更新bestp=15。再扩展右分支，cp=11，rp=4−4=0，cp+rp<bestp，不满足限界条件，舍弃，如下图所示。

****

**（11）队头元素J出队**，该节点的cp+rp<bestp=15，不满足限界条件，不再扩展。

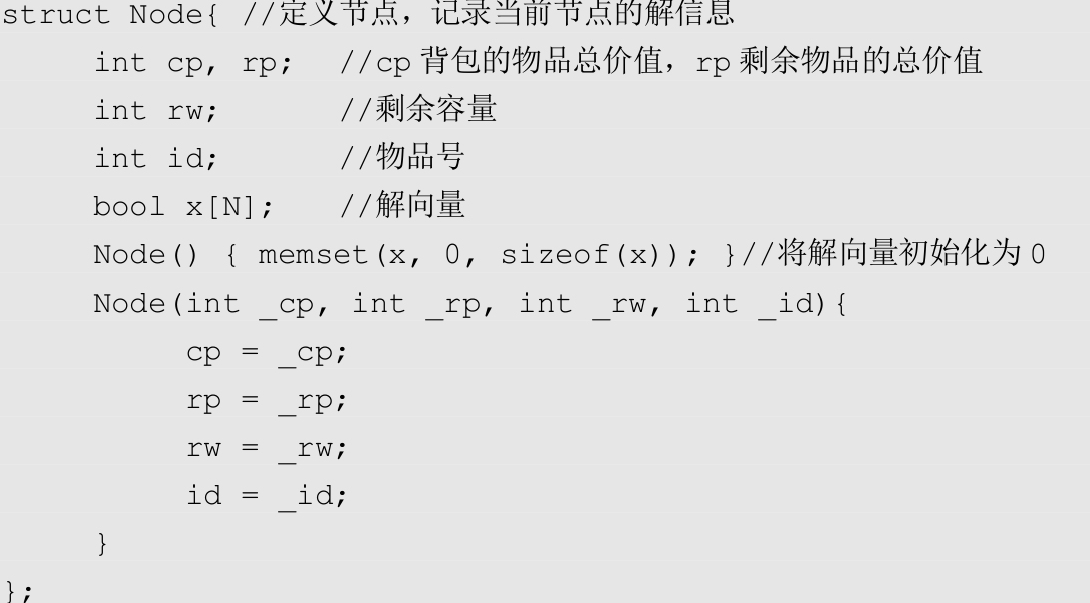
**（12）队头元素K出队**，扩展节点K，t=5，已经处理完毕，cp<bestp，不是最优解。

**（13）队头元素L出队**，扩展节点L，t=5，已经处理完毕，cp=bestp，是最优解，输出该解向量(1,0,1,1)。

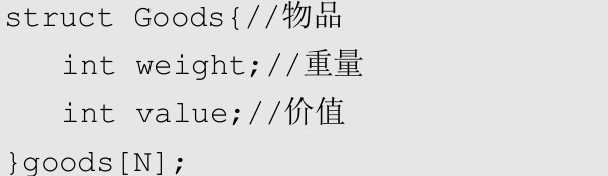
**（14）队列为空，算法结束**。

**3. 算法实现**

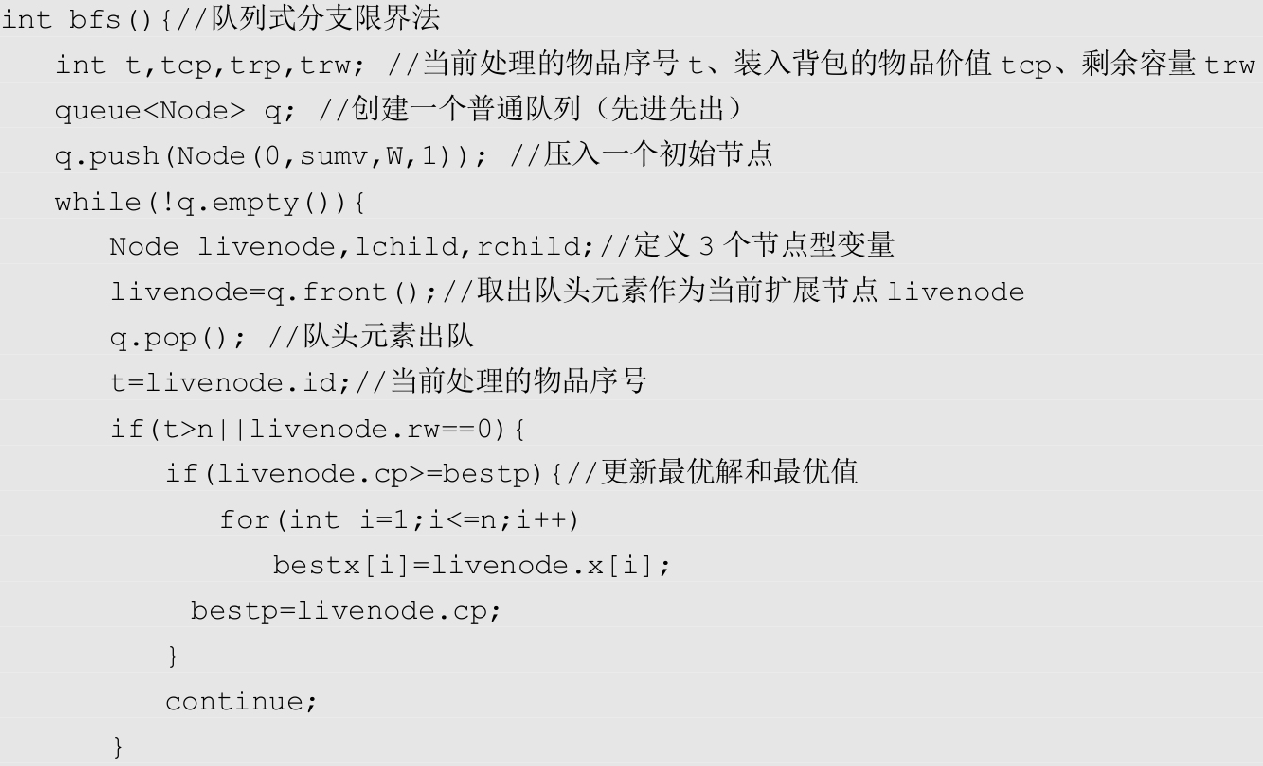
**（1）定义节点结构体。**



**（2）定义物品结构体。**在前面处理背包问题时，使用了两个一维数组w[]、v[]分别存储物品的重量和价值，在此使用一个结构体数组来存储这些重量和价值。

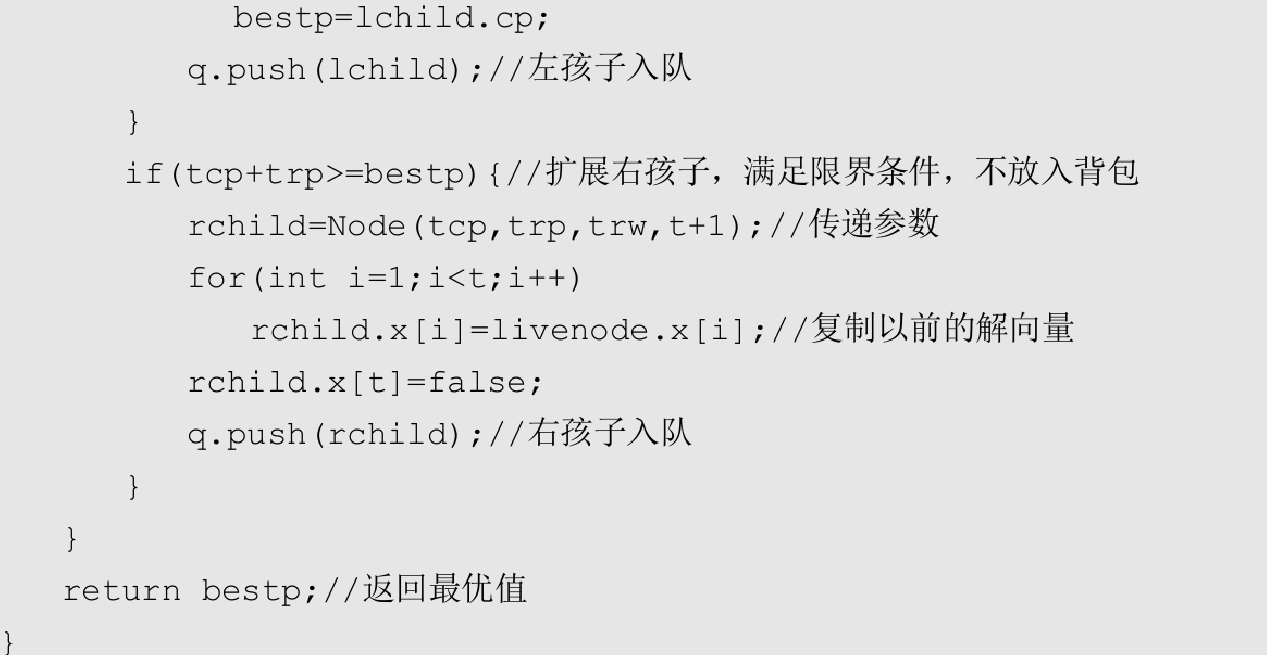


**（3）搜索解空间。**首先创建一个普通队列（先进先出），然后将根节点加入队列中，如果队列不空，则取出队头元素livenode，得到当前处理的物品序号，如果当前处理的物品序号大于n，则说明搜到最后一个物品了，不需要往下搜索。如果当前的背包没有剩余容量（已经装满）了，则不再扩展。如果当前放入背包的物品价值大于或等于最优值（livenode.cp≥bestp），则更新最优解和最优值。判断是否满足约束条件，满足则生成左孩子，判断是否更新最优值，左孩子入队，不满足约束条件则舍弃左孩子；判断是否满足限界条件，满足则生成右孩子，右孩子入队，不满足限界条件则舍弃右孩子。



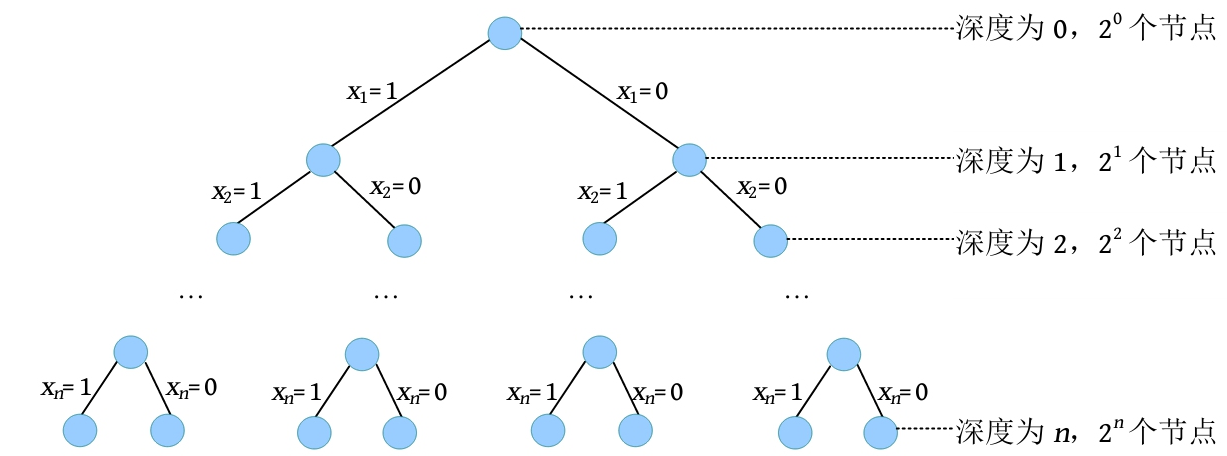






**4. 算法分析**

**时间复杂度：**算法的运行时间取决于它在搜索过程中生成的节点数。而限界函数可以大大减少所生成的节点个数，避免无效搜索，加快搜索速度。左孩子需要判断约束函数，右孩子需要判断限界函数，那么在最坏情况下有多少个左孩子和右孩子呢？规模为n的子集树在最坏情况下的状态如下图所示。



总的节点个数为20+21+…+2n=2n+1−1，减去树根节点再除以2，就得到左右孩子的个数，左右孩子的个数=(2n+1−1−1)/2=2n−1。约束函数时间复杂度为O(1)，限界函数时间复杂度为O(1)。在最坏情况下有O(2n)个左孩子需要调用约束函数，有O(2n)个右孩子需要调用限界函数，所以计算背包问题的分支限界法的时间复杂度为**O(2n+1)**。

**空间复杂度：**空间主要耗费在Node节点里面存储的变量和解向量上，因为最多有O(2n+1)个节点，而每个节点的解向量都需要O(n)个空间，所以空间复杂度为**O(n×2n+1)**。其实**让每个节点都记录解向量的办法是很笨的**，我们**可以用指针记录当前节点的左右孩子和父亲，到达叶子时逆向找其父亲，直到根节点，就得到了解向量，这样空间复杂度降为O(n)**。

**二、　优先队列式广度优先搜索**

优先队列优化以当前节点的上界为优先值，把普通队列改成优先队列，这样就得到了优先队列式分支限界法。

**1. 算法设计**

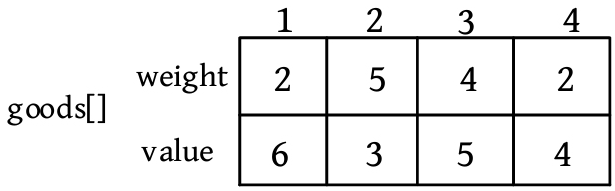
优先级的定义为活节点代表的部分解所描述的已装入物品的价值上界，上界越大，优先级越高。活节点的价值上界up=活节点的cp+剩余物品装满背包剩余容量的最大价值rp'。

**约束条件：cw+w[i]≤W。**

**限界条件：up=cp+rp'≥bestp。**

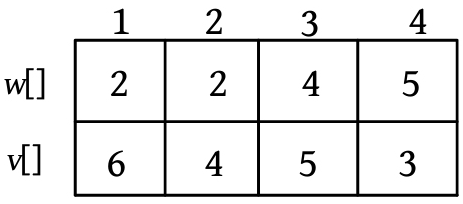
**2. 图解**

有一个背包和4个物品，每个物品的重量和价值如下图所示，背包的容量W=10。求在不超过背包容量的前提下，把哪些物品放入背包才能获得最大价值。



**（1）初始化。**sumw和sumv分别用来统计所有物品的总重量和总价值。sumw=13，sumv=18，sumw>W，因此不能全部装完，需要搜索求解。

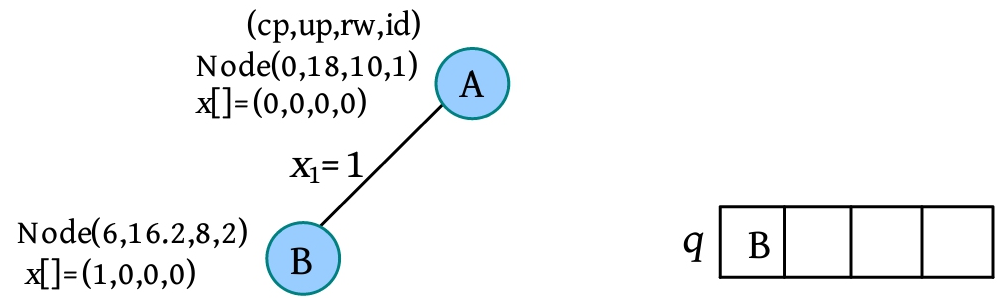
**（2）按价值重量比非递增排序。**排序后的结果如下图所示。为了程序处理方便，把排序后的数据存储在w[]和v[]数组中。后面的程序在该数组上操作即可，如下图所示。



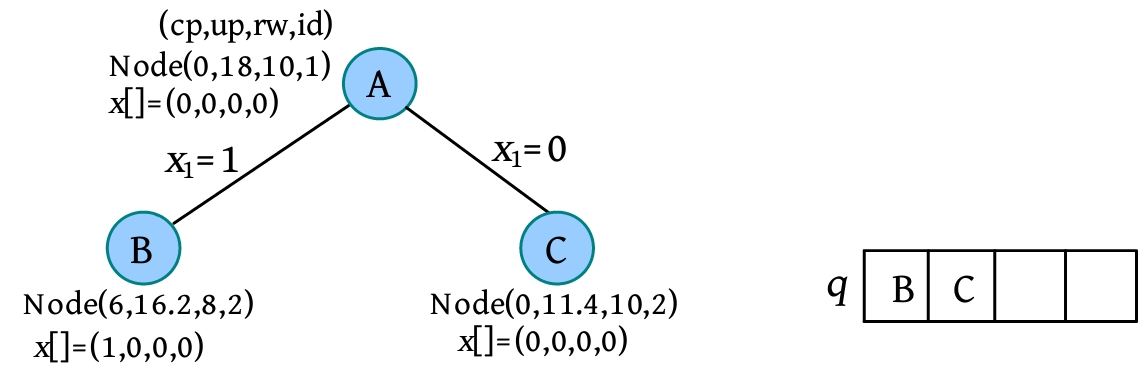
**（3）创建根节点A。**初始化当前放入背包的物品重量cp=0，当前价值上界up=sumv，当前剩余容量rw=W，当前处理物品序号为1，当前最优值bestp=0。最优解初始化为x[]=(0,0,0,0)，创建一个根节点Node(cp,up,rw,id)，标记为A，加入优先队列q中，如下图所示。



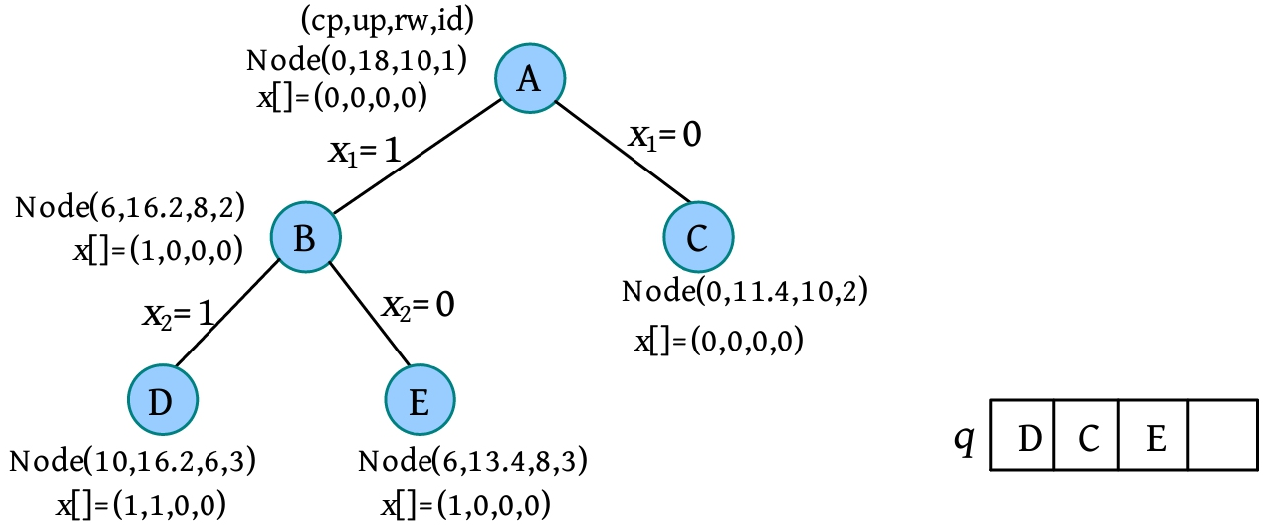
**（4）扩展节点A。**队头元素A出队，该节点的up≥bestp，满足限界条件，可以扩展。rw=10>w[1]=2，剩余容量大于1号物品的重量，满足约束条件，可以放入背包，生成左孩子，令cp=0+6=6，rw=10-2=8。那么上界怎么算呢？up=cp+rp'=cp+剩余物品装满背包剩余容量的最大价值rp'。剩余容量还有8，可以装入2、3号物品，装入后还有剩余容量2，只能装入4号物品的一部分，装入的价值为剩余容量×单位重量价值，即2×3/5=1.2，rp'=4+5+1.2=10.2，up=cp+rp'= 16.2。在此需要注意，背包问题属于01背包问题，物品要么装入，要么不装入，是不可以分割的，这里为什么还会有部分装入的问题呢？很多读者看到这里都有这样的疑问，在此不是真的部分装入了，只是算上界而已。令t=2，x[1]=1，解向量更新为x[]=(1,0,0,0)，创建新节点B并将其加入q队列中，更新bestp=6，如下图所示。



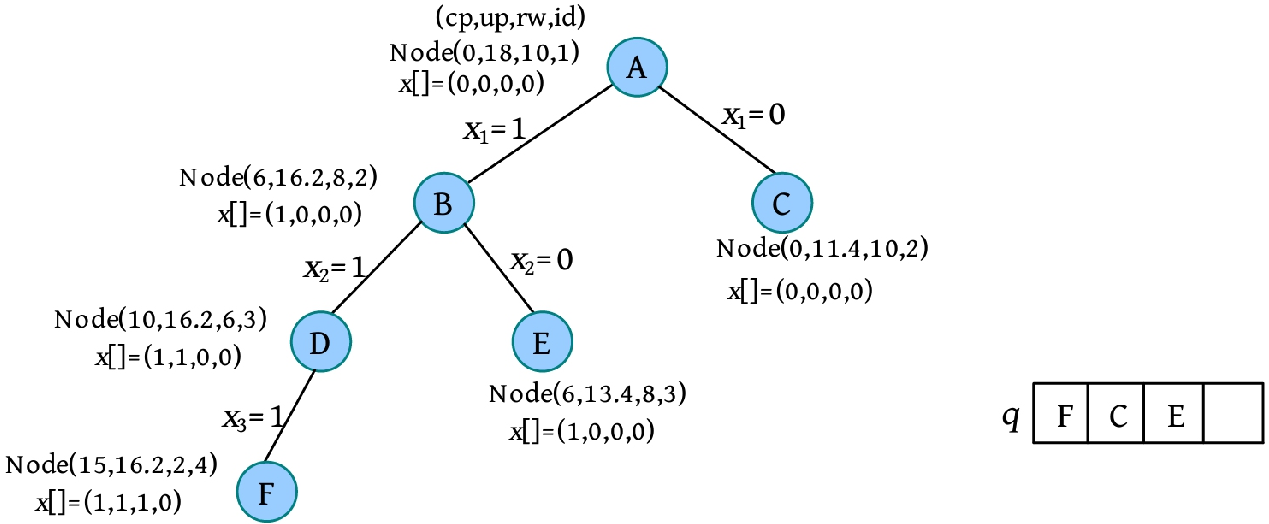
再扩展右分支，cp=0，rw=10，剩余容量可以装入2、3号物品，装入后还有剩余容量4，只能装入4号物品的一部分，装入的价值为剩余容量单位重量价值，即4×3/5=2.4，rp'=4+5+2.4= 11.4，up=cp+rp'=11.4，up>bestp，满足限界条件，令t=2，x[1]=0，解向量更新为x[]=（0,0,0,0），生成右孩子C并将其加入q队列中，如下图所示。



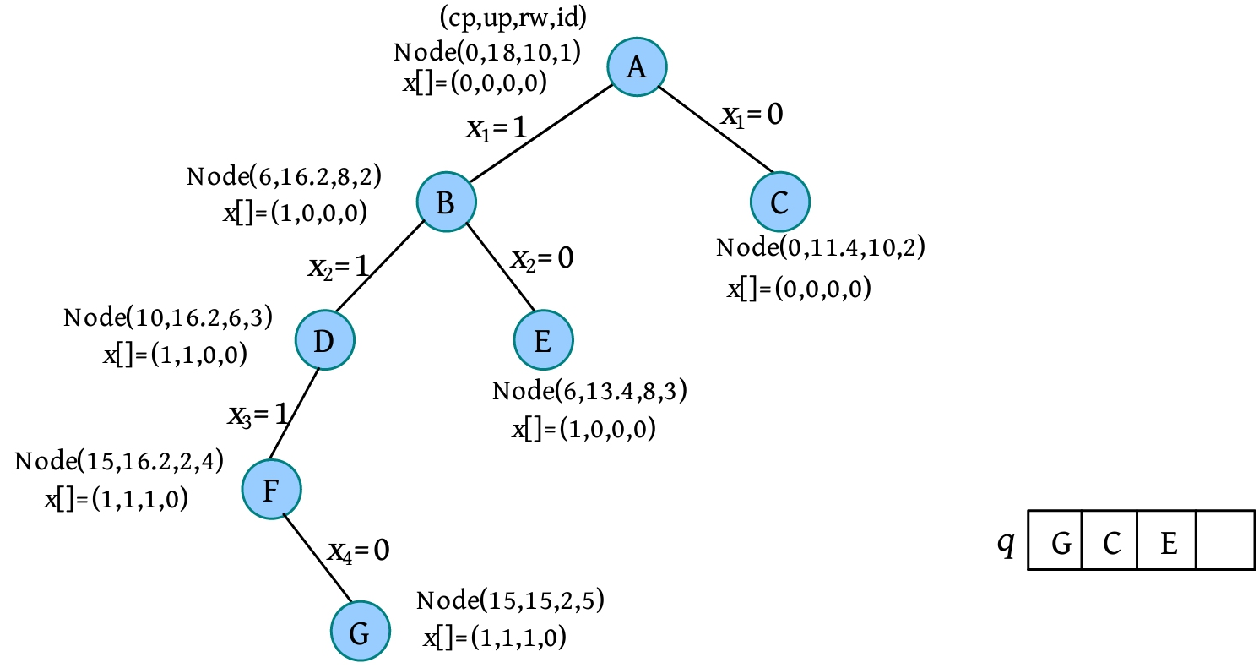
**（5）扩展节点B。**队头元素B出队，该节点的up≥bestp，满足限界条件，可以扩展。剩余容量rw=8>w[2]=2，大于2号物品的重量，满足约束条件，令cp=6+4=10，rw=8−2=6，up=cp+rp'=10+5+2×3/5=16.2，t=3，x[2]=1，解向量更新为x[]=(1,1,0,0)，生成左孩子D并将其加入q队列中，更新bestp=10。再扩展右分支，cp=6，rw=8，剩余容量可以装入3号物品，4号物品部分装入，up=cp+rp'=6+5+3×4/5=13.4，up>bestp，满足限界条件，令t=3，x[2]=0，解向量为x[]=(1,0,0,0)，生成右孩子E并将其加入q队列中。注意：q为优先队列，其实是用堆实现的，如果不想搞清楚，则只需知道每次up值最大的节点出队即可，如下图所示。



**（6）扩展节点D。**队头元素D出队，该节点的up≥bestp，满足限界条件，可以扩展。剩余容量rw=6>w[3]=4，大于3号物品的重量，满足约束条件，令cp=10+5=15，rw=6−4=2，up=cp+rp'=10+5+2×3/5=16.2，t=4，x[3]=1，解向量更新为x[]=(1,1,1,0)，生成左孩子F并将其加入q队列中，更新bestp=15。再扩展右分支，cp=10，rw=8，剩余容量可以装入4号物品，up=cp+rp'=10+3=13，up<bestp，不满足限界条件，舍弃右孩子，如下图所示。



**（7）扩展节点F。**队头元素F出队，该节点的up≥bestp，满足限界条件，可以扩展。剩余容量rw=2<w[4]=5，不满足约束条件，舍弃左孩子。再扩展右分支，cp=15，rw=2，虽然有剩余容量，但物品已经处理完毕，已没有物品可以装入，up=cp+rp'=15+0=15，up≥bestp，满足限界条件，令t=5，x[4]=0，解向量为x[]=(1,1,1,0)，生成右孩子G并将其加入q队列中，如下图所示。



**（8）扩展节点G。**队头元素G出队，该节点的up≥bestp，满足限界条件，可以扩展。t=5，已经处理完毕，bestp=cp=15，是最优解，解向量为x[]=(1,1,1,0)。注意：虽然解是(1,1,1,0)，但对应的物品原来的序号是1、4、3。节点G出队。

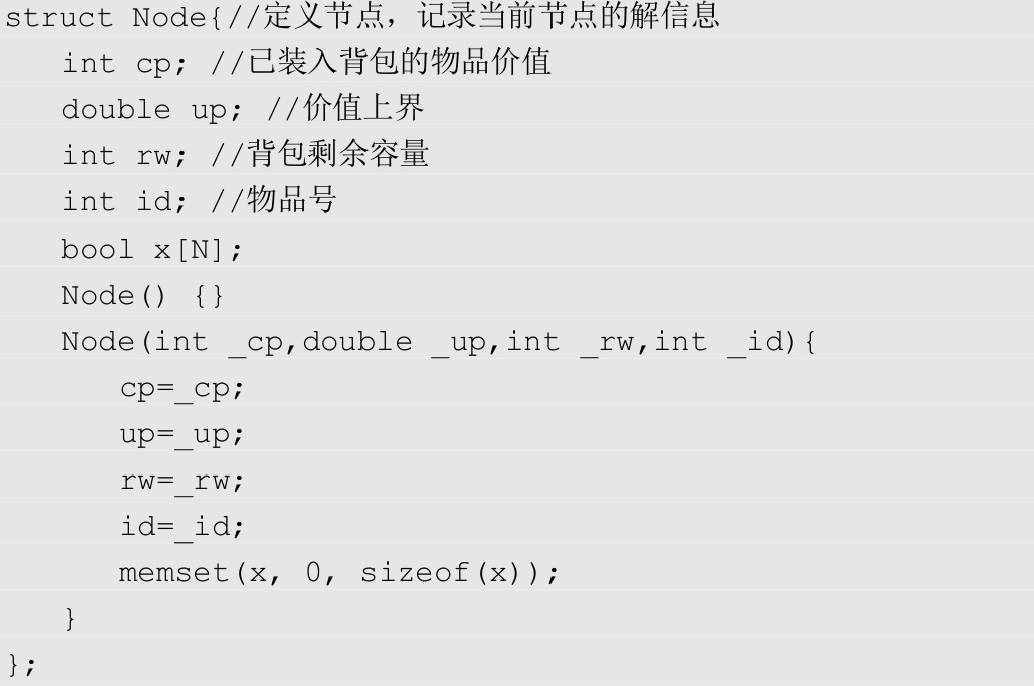
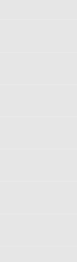
**（9）队头元素E出队**，该节点的up<bestp，不满足限界条件，不再扩展。

**（10）队头元素C出队**，该节点的up<bestp，不满足限界条件，不再扩展。

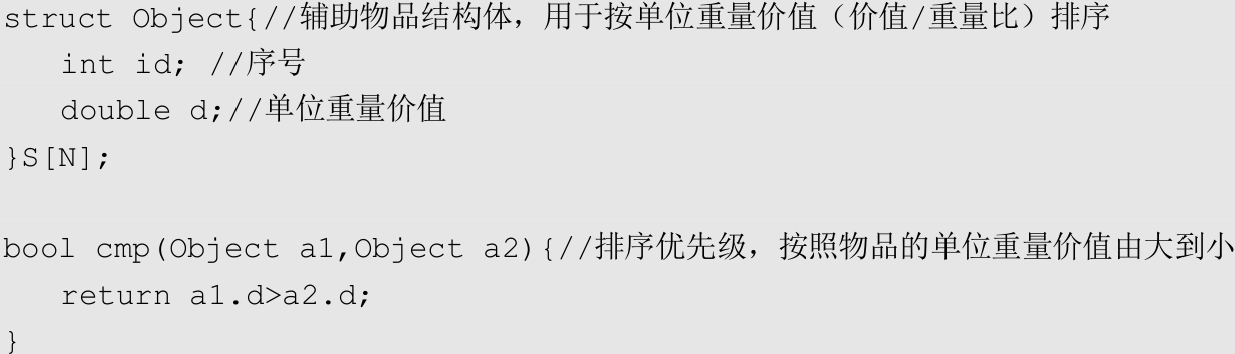
**（11）队列为空**，算法结束。

**3. 算法实现**

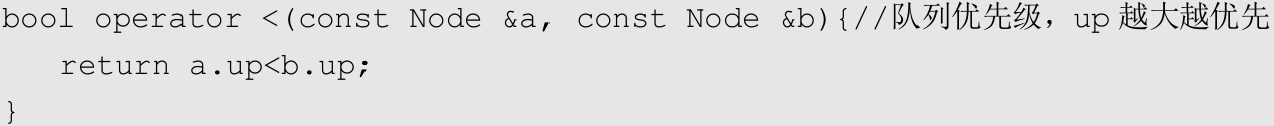
**（1）定义节点和物品结构体。**



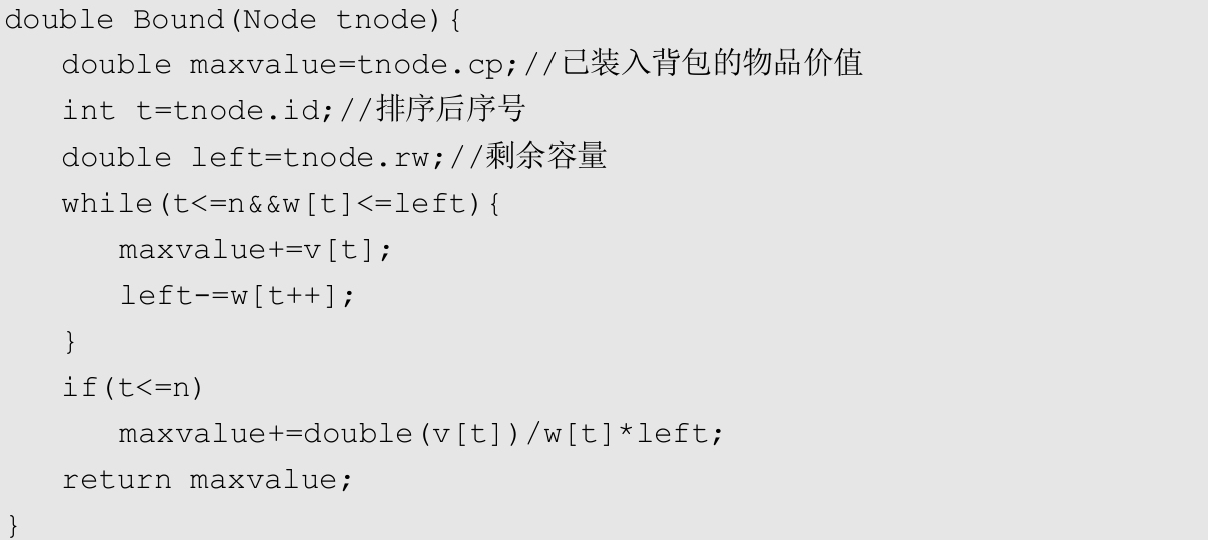
**（2）定义辅助结构体和排序优先级（从大到小排序）。**

****

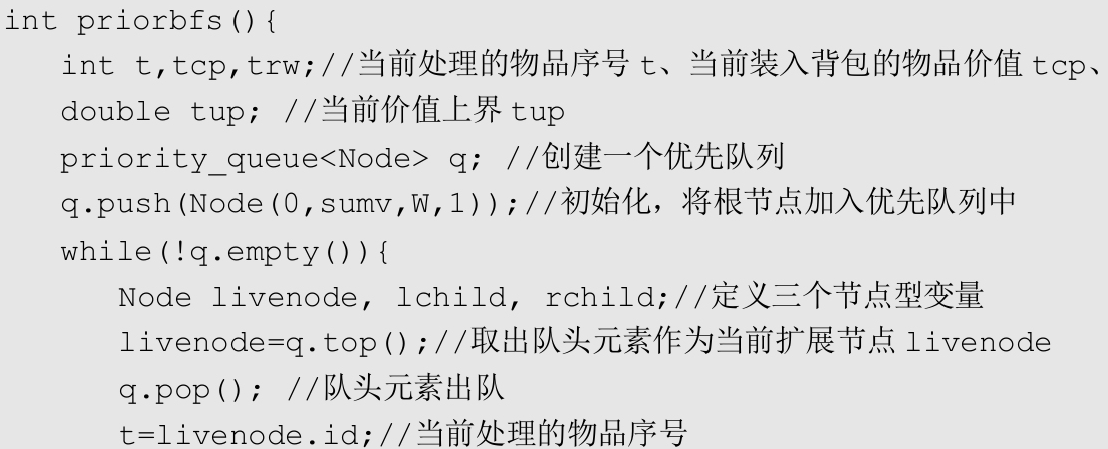
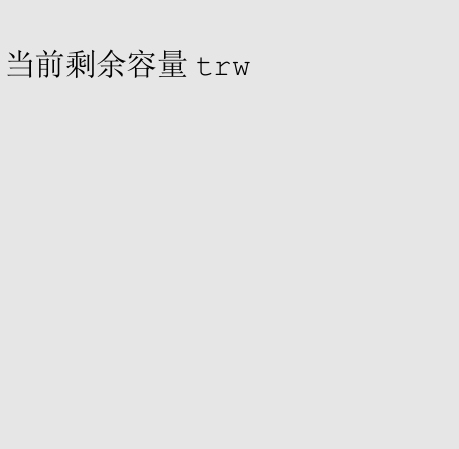
**（3）定义队列的优先级。**

****

**（4）计算节点的上界。**

****

**（5）优先队列分支限界法。**

****







**4. 算法分析**

虽然在算法复杂度数量级上，优先队列的分支限界法算法和普通队列的算法相同，但从图解可以看出，采用优先队列式的分支限界法算法生成的节点数更少，找到最优解的速度更快。