**0-1背包的限界函数**

假设求解最大化问题，解向量为X=(x1, x2, …, xn)，其中，xi的取值范围为某个有穷集合Si，|Si|=ri（1≤i≤n）。在使用分支限界法搜索问题的解空间树时，首先根据限界函数估算目标函数的界[down, up]，然后从根结点出发，扩展根结点的r1个孩子结点，从而构成分量x1的r1种可能的取值方式。对这r1个孩子结点分别估算可能取得的目标函数值bound(x1)，其含义是以该孩子结点为根的子树所可能取得的目标函数值不大于bound(x1)，也就是部分解应满足：bound(x1)≥bound(x1, x2)≥ … ≥bound(x1, x2, …, xk)≥ … ≥bound(x1, x2, …, xn)若某孩子结点的目标函数值超出目标函数的界，则将该孩子结点丢弃；否则，将该孩子结点保存在待处理结点表PT中。从表PT中选取使目标函数取得极大值的结点作为下一次扩展的根结点，重复上述过程，当到达一个叶子结点时，就得到了一个可行解X=(x1, x2, …, xn)及其目标函数值bound(x1, x2, …, xn)。如果bound(x1, x2, …, xn)是表PT中目标函数值最大的结点，则bound(x1, x2, …, xn)就是所求问题的最大值，(x1, x2, …, xn)就是问题的最优解；如果bound(x1, x2, …, xn)不是表PT中目标函数值最大的结点，说明还存在某个部分解对应的结点，其上界大于bound(x1, x2, …, xn)。于是，用bound(x1, x2, …, xn)调整目标函数的下界，即令down=bound(x1, x2, …, xn)，并将表PT中超出目标函数下界down的结点删除，然后选取目标函数值取得极大值的结点作为下一次扩展的根结点，继续搜索，直到某个叶子结点的目标函数值在表PT中最大。

int MaxKnapsack(){

while(非叶节点){//没有找到结果

if(当前体积<=背包体积){//左分支剪枝函数

if(当前价值>最大价值)//向上更新最大价值

最大价值 = 当前价值；

add(左节点)；//左节点加入优先队列 }

bound()；//计算上限

if（上限>最大价值）{//当不选择该结点时，后面全部按照最优选择计算，结果>最大价值时，才考虑。 add（右节点）;//右节点加入优先队列}

heapnode N；//堆排序

H->deletemax(N);//从优先队列中取下一个活结点。}

}

//计算上限

int bound(int i){

int left = 剩余容量；

int b = 价值上限变量;

while(i<=n && 第i个物品重量<=剩余容量){

left -=第i个物品重量；

b+=第i个物品价值；

i++；

}

if(i<=n){//如果第i个物品重量>剩余容量，则选择“部分”该物品:贪心策略

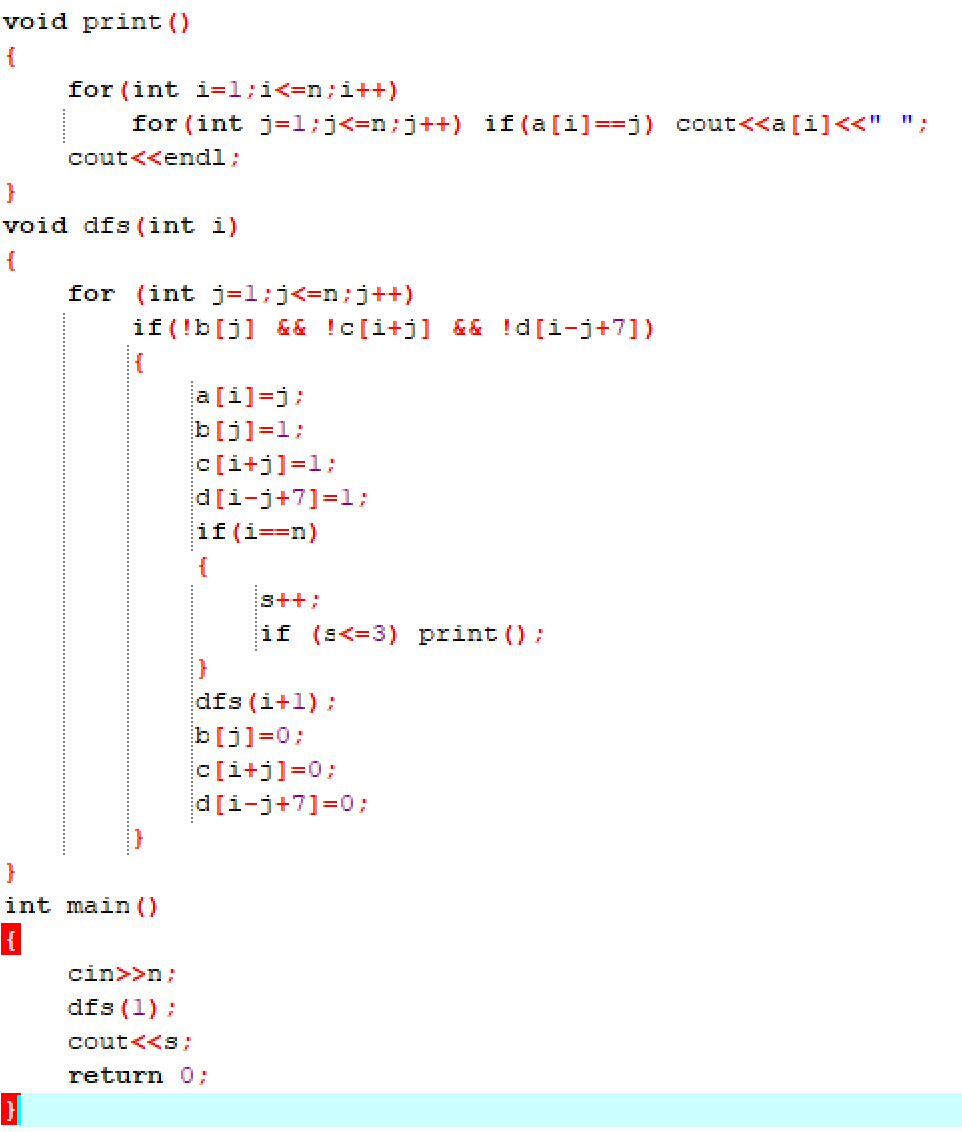
b+=第i个物品单位价值\*剩余容量；

}

return b;

}

**N后问题**



**改进回溯算法**

选择合适的限界函数和约束函数，如旅行商问题和背包问题分别采用了不同的限界函数来约束搜索，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。

　　此外，回溯法以深度优先的方式搜索解空间，而分支界限法则以广度优先的方式或以最小耗费优先的方式搜索解空间。

搜索策略：在当前节点（扩展节点）处，先生成其所有的儿子节点（分支），然后再从当前的活节点（当前节点的子节点）表中选择下一个扩展节点。为了有效地选择下一个扩展节点，加速搜索的进程，在每一个活节点处，计算一个函数值（限界），并根据函数值，从当前活节点表中选择一个最有利的节点作为扩展节点，使搜索朝着解空间上有最优解的分支推进，以便尽快地找出一个最优解。

从活结点表中选择下一扩展结点的不同方式导致不同的分支限界法。最常见的有以下两种：

1.队列式（FIFO）分支限界法

队列式分支限界法将活节点表组织成一个队列，并将队列的先进先出原则选取下一个节点为当前扩展节点。

2.优先队列式分支限界法

优先队列式分支限界法将活节点表组织成一个优先队列，并将优先队列中规定的节点优先级选取优先级最高的下一个节点成为当前扩展节点。如果选择这种选择方式，往往将数据排成最大堆或者最小堆来实现。

**0-1背包优先级**

采用优先队列的方式

算法的思想

首先，要对输入数据进行预处理，将各物品依其单位重量价值从大到小进行排列。

在实现时，由Bound计算当前结点处的上界。在解空间树的当前扩展结点处，仅当要进入右子树时才计算右子树的上界Bound，以判断是否将右子树剪。进入左子树时不需要计算上界，因为其上界与其父节点上界相同。

在优先队列分支限界法中，结点的优先级定义为：以结点的价值上界作为优先级（由bound函数计算出）

步骤

算法首先根据基于可行结点相应的子树最大价值上界优先级，从堆中选择一个节点（根节点）作为当前可扩展结点。

检查当前扩展结点的左儿子结点的可行性。

如果左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中。

当前扩展结点的右儿子结点一定是可行结点，仅当右儿子结点满足上界函数约束时,才将它加入子集树和活结点优先队列。

当扩展到叶节点时，算法结束，叶子节点对应的解即为问题的最优值。

**旅行商问题**

定义一个与顶点数量同样大的数组MinOut，它用来存放从该节点到达其他有路径结点的最小费用。再用MinSum计算这个最小费用数组的费用总和，可以确定这个总和一定是最小最小的费用，但是这个费用对应的路径不一定能够走得通，假设s是我们正在或者刚到达的城市，假设从第s到第n个城市的最小费用用rcost定义，它的初始值为MinSum。每次到达一个城市，MinSum就减去s城市对应的最小费用即减去MinOut[s]，并且这个计算是从第一个城市开始的，于是有：

rcost = E.rcost - MinOut[E.x[E.s]];

时定义优先级为lcost，它指的是从第一个城市到第s个城市的费用(这是一个确定的值)，它加上rcost这时候不就是走完回路的最小费用，而它就是这条路径的下界，也就是我们优先级的定义：

lcost = cc + rcost;//cc指的是到达s城市之前的费用