算法分析与设计

第7章 分支限界法

主讲人：甘文生 PhD

Email: wsgan001@gmail.com

暨南大学网络空间安全学院

Fall 2021

Jinan University, China

第7章 分支限界法

内容提要：

- 理解分支限界法的剪枝搜索策略
- 掌握分支限界法的算法框架
  - （1）队列式 (FIFO) 分支限界法
  - （2）优先队列式分支限界法
- 通过应用范例学习分支限界法的设计策略

方法概述

与回溯法区别：

- **求解目标不同：**

  一般而言，回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是尽快地找出满足约束条件的一个解；

- **搜索方法不同：**

  回溯法使用深度优先方法搜索，而分支限界一般用宽度优先或最佳优先方法来搜索；

- **对扩展结点的扩展方式不同：**

  分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点；

- **存储空间的要求不同**

  分支限界法的存储空间比回溯法大得多，因此当内存容量有限时，回溯法解决问题成功的可能性更大。

方法概述

基本思想：

- 分支限界法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。i) 对已处理的各结点根据限界函数估算目标函数的可能取值，ii) 从中选出目标函数取得极大(极小) 值的结点优先进行广度优先搜索，iii) 不断地调整搜索方向，尽快找到解，裁剪那些不能得到最优解的子树以提高搜索效率。

- 特点：限界函数一般是基于问题的目标函数，适用于求解最优化问题。

方法概述

- 搜索策略：在扩展结点处，首先生成其所有的儿子结点(分支)，然后从当前的活结点表中选择下一个扩展结点。 思考：如何代码实现？

  为了有效地选择下一个扩展结点，以加速搜索的进程，在每一个活结点处，计算一个函数值(优先值)，并根据这些已计算出的函数值，从当前活结点表中选择一个最有利的结点作为扩展结点，使搜索朝着解空间树上有最优解的分支推进，以便尽快地找出一个最优解。

方法概述

- 求解步骤：

① 定义解空间(对解编码)；

② 确定解空间的树结构；

③ 按BFS等方式搜索：

a. 每个活结点仅有一次机会变成扩展结点；

b. 由扩展结点生成一步可达(即宽度搜索)的新结点；

c. 在新结点中，删除不可能导出最优解的结点； // 限界策略

d. 将剩余的新结点加入活动表(队列)中；

e. 从活动表中选择结点再扩展； //分支策略

f. 直至活动表为空；

方法概述

常见的两种分支限界法：

- 队列式(FIFO)分支限界法：从活结点表中取出结点的顺序与加入结点的顺序相同，因此活结点表的性质与队列相同；

- 优先队列(代价最小或效益最大)分支限界法：每个结点都有一个对应的耗费或收益，以此决定结点的优先级：
  - 如果查找一个具有最小耗费的解，则活结点可用小根堆来建立，下一个扩展结点就是具有最小耗费的活结点；
  - 如果希望搜索一个具有最大收益的解，则可用大根堆来构造活结点表，下一个扩展结点是具有最大收益的活结点。

解空间树的动态搜索

(1) 回溯求解0/1背包问题，剪枝能减少搜索空间，但整个搜索按深度优先机械地进行，是盲目搜索(因为该方法不能预测本结点以下的结点进行得如何)。

(2) 回溯求解TSP问题也是盲目的(虽有目标函数，也只有找到一个可行解后才有意义)。

解空间树的动态搜索

分支限界法首先确定一个合理的限界函数，并根据限界函数确定目标函数的界[down, up]；然后按照广度优先策略遍历问题的解空间树，在某一分支上，依次搜索该结点的所有孩子结点，分别估算这些孩子结点的目标函数的可能取值（注意：对于最小化问题，估算结点的down，对最大化问题，估算结点的up)。

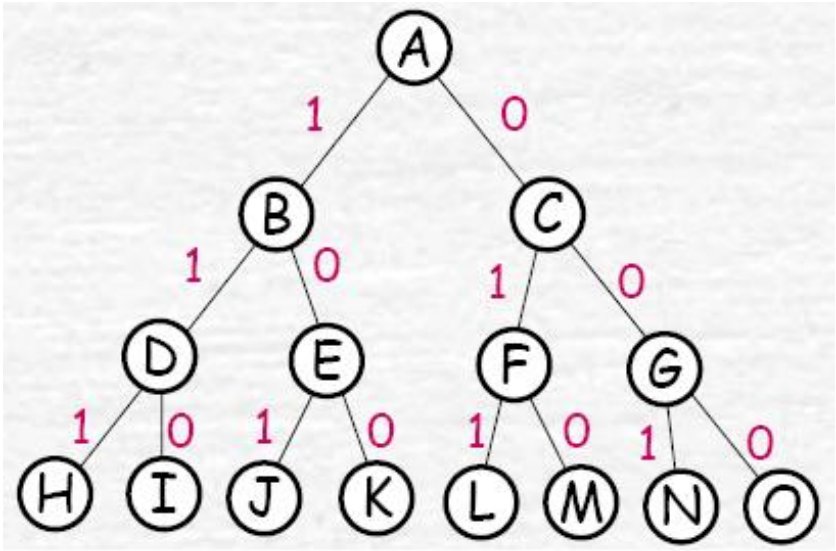
如果某孩子结点的目标函数值超出目标函数的上界或下界，则将其丢弃(即基于该结点生成的解不会比目前已得的更好)，否则入待处理表。

0-1背包问题

- 物品数量n = 3，重量w = (20, 15, 15)，价值v = (40, 25, 25)，背包容量c = 30，试装入价值和最大的物品？
- FIFO队列分支限界法求解：

① 解空间：{(0,0,0), (0,0,1), …, (1,1,1)}

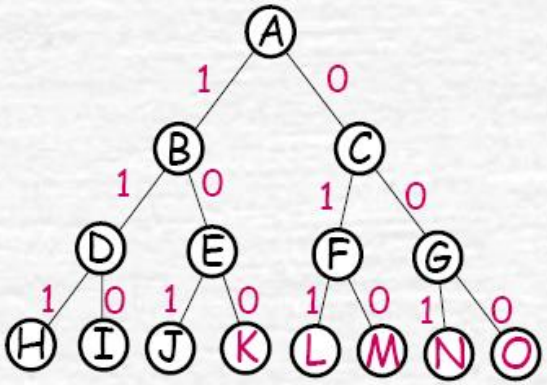
② 解空间树：



0-1背包问题

③ BFS搜索 (FIFO队列)

扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
A	B,C	BC		
B	D,E(D死结点)	CE		
C	F,G	EFG		
E	J,K(J死结点)	FG	K	40
F	L,M	G	L,M	50,25
G	N,O	ϕ	N,O	25,0



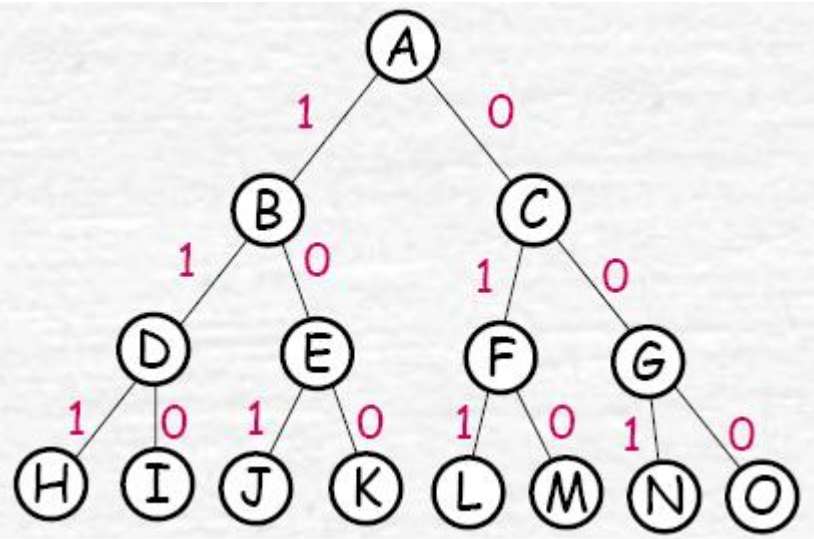
∴ 最优解为L，即(0,1,1)；解值为50

0-1背包问题

- 优先队列分支限界法求解：

① 解空间：{(0,0,0), (0,0,1), …, (1,1,1)}

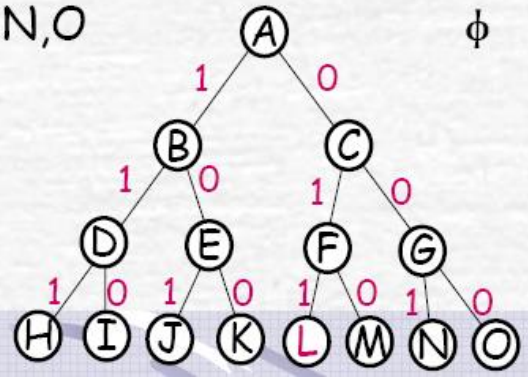
② 解空间树：



0-1背包问题

- BFS搜索 (优先队列：**按照价值率优先**)

扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
A	B,C	B C		
B	D,E(D死结点)	E C		
E	J,K(J死结点)	C	K	40
C	F,G	F G		
F	L,M	G	L	50(最优)
G	N,O	ϕ		



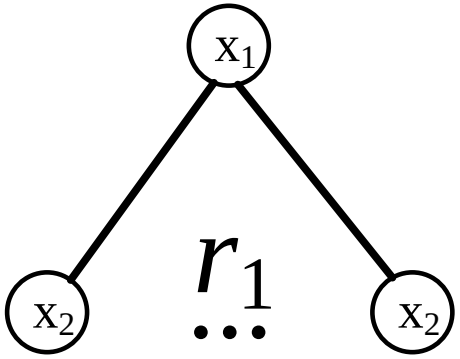
0-1背包问题

- **总结：**

  从0-1背包问题的搜索过程可看出：与回溯法相比，分支限界法可**根据限界函数不断调整搜索方向**，选择**最可能得最优解的子树**优先进行搜索→找到问题的解。

分支限界法的设计思路

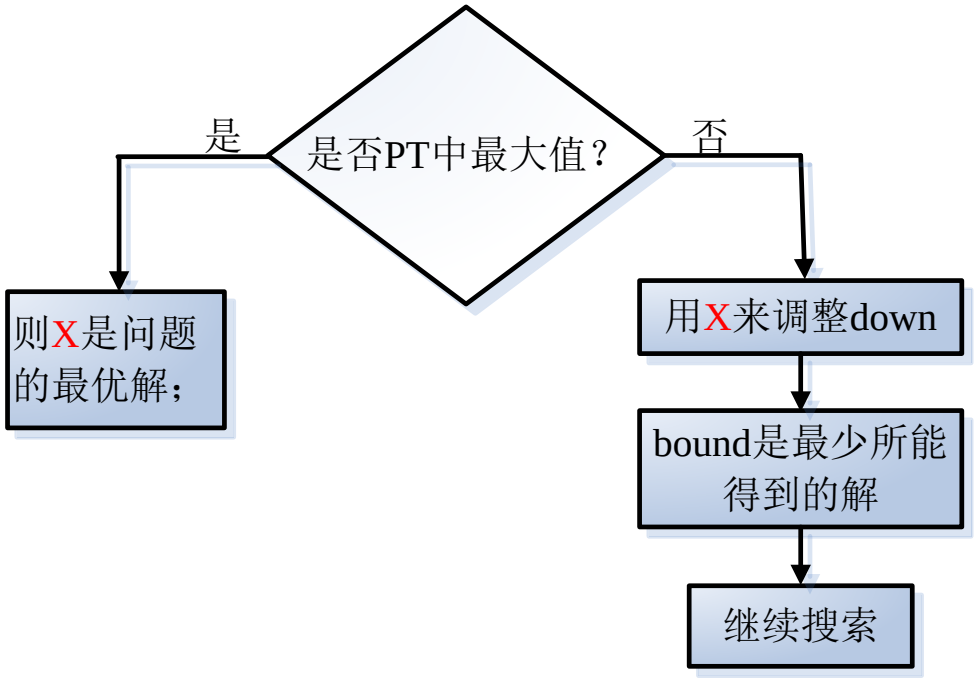
设求解最大化问题，解向量为$X=(x_1,\ldots,x_n)$，xi的取值范围为S_i，$|S_i|=r_i$。在使用分支限界搜索问题的解空间树时，先根据限界函数估算目标函数的界$[down, up]$，然后从根结点出发，扩展根结点的r_1个孩子结点，从而构成分量x_1的r_1种可能的取值方式。



对这r_1个孩子结点分别估算可能的目标函数$bound(x_1)$，其含义：以该结点为根的子树所有可能的取值不大于$bound(x_1)$，即：$bound(x_1) \geq bound(x_1, x_2) \geq \ldots \geq bound(x_1,\ldots, x_n)$

分支限界法的设计思路

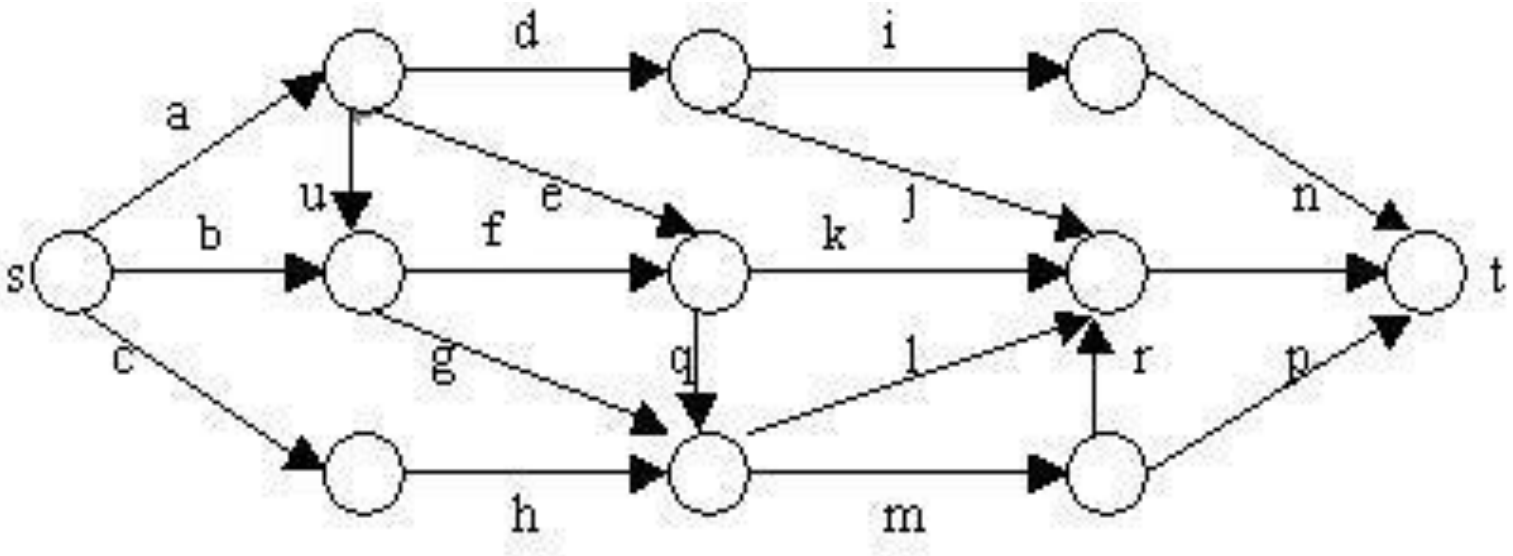
- 若某孩子结点的目标函数值超出目标函数的下界，则将该孩子结点丢弃；否则，将该孩子结点保存在待处理结点表PT中。
- 再取PT表中目标函数极大值结点作为扩展的根结点，重复上述。
- 直到一个叶子结点时的可行解$X=(x_1,\ldots,x_n)$，及目标函数值$bound(x_1,\ldots,x_n)$。



单源最短路径问题

- 问题描述：

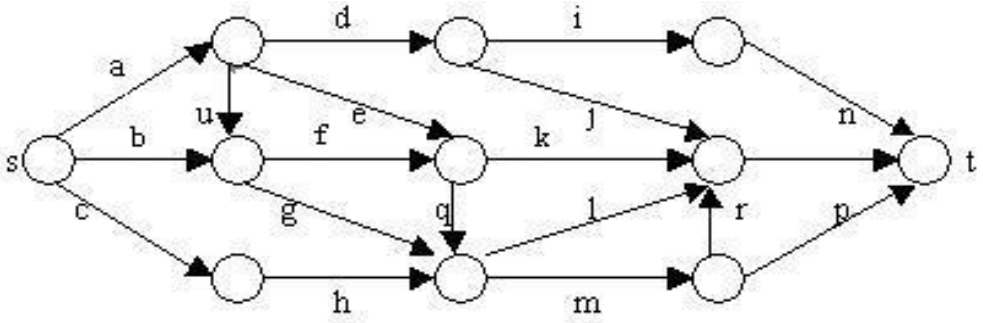
下面以一个例子来说明单源最短路径问题：在下图所给的有向图G中，每一边都有一个非负边权。要求图G的从源顶点s到目标顶点t之间的最短路径。

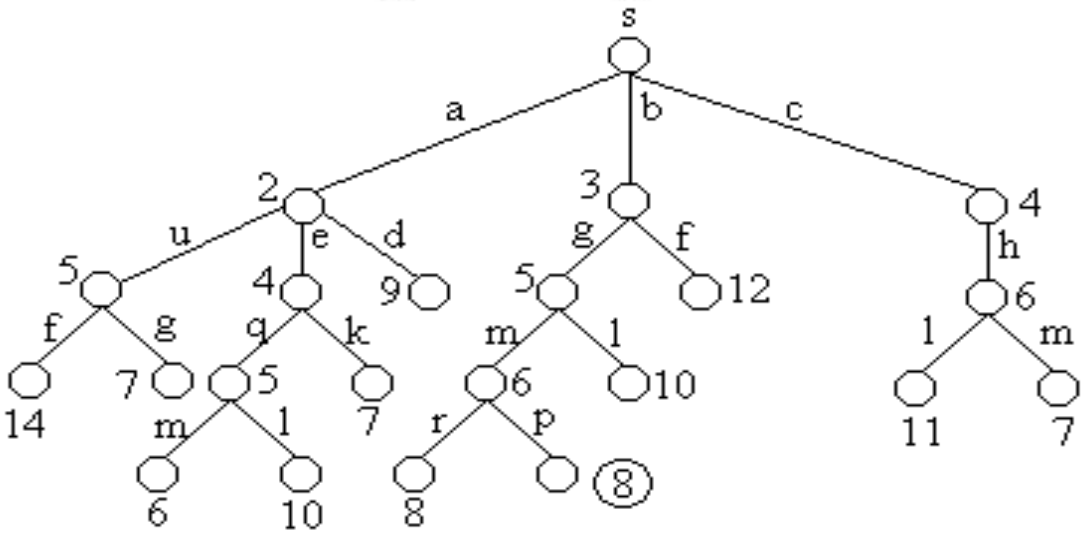


单源最短路径问题

- 解空间树：

下图是用优先队列式分支限界法解有向图G的单源最短路径问题产生的解空间树。其中，每一个结点旁边的数字表示该结点所对应的当前路长。





单源最短路径问题

- 算法思想：

解单源最短路径问题的优先队列式分支限界法用一极小堆来存储活结点表。其优先级是结点所对应的当前路长。

算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所相应的路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。这个结点的扩展过程一直继续到活结点优先队列为空时为止。

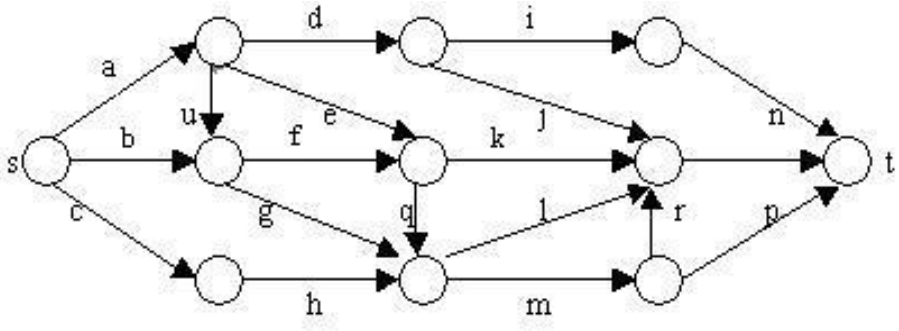
单源最短路径问题

- 剪枝策略：

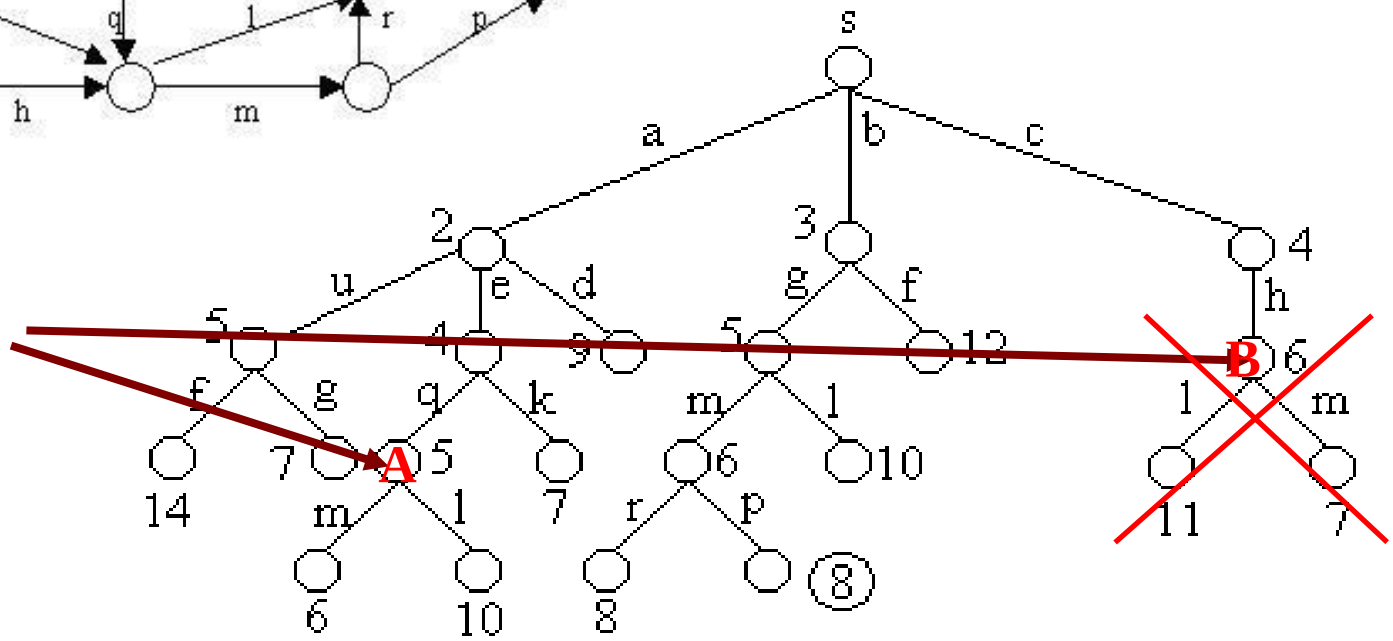
在算法扩展结点的过程中，一旦发现一个结点的下界不小于当前找到的最短路长，则算法剪去以该结点为根的子树。

在算法中，利用结点间的控制关系进行剪枝。从源顶点s出发，2条不同路径到达图G的同一顶点。由于两条路径的路长不同，因此可以将路长长的路径所对应的树中的结点为根的子树剪去。

单源最短路径问题



经过不同的路径到达相同的顶点



A优于B，B可剪枝

单源最短路径问题

- 算法代码：

```
while (true)
{ // 搜索问题的解空间，//while循环体完成对解空间内部结点的扩展
   for (int j=1;j<=n;j++)
        if(a[enode.i][j] < Float.MAX_VALUE  &&  enode.length+a[enode.i][j] < dist[j])
        { // 顶点i到顶点j可达，且满足控制约束，
           dist[j]=enode.length+a[enode.i][j];
           p[j]=enode.i;
           HeapNode node = new HeapNode(j,dist[j]);
           heap.put(node);     //加入活结点优先队列
        }
       if (heap.isEmpty()) break;
       else enode = (HeapNode) heap.removeMin();
 }
```

单源最短路径问题

Dijakstra 算法和分支限界法解决该问题的异同

- Dijakstra算法：每一步的选择为当前步的最优,复杂度为$O(n^2)$。

- 分支限算法：每一步的扩散为当前耗散度的最优。
  - 队列式分支限界法的搜索解空间树的方式类似于解空间树的宽度优先搜索，不同的是队列式分支限界法不搜索以不可行结点(即已经被判定不能导致可行解或不能导致最优解的结点)为根的子树。按照规则，这样的结点不被列入活结点表。A -> E -> Q -> M

  - 优先队列式分支限界法的搜索方式是根据活结点的优先级确定下一个扩展结点。结点的优先级常用一个与该结点有关的数值p来表示。最大优先队列规定p值较大的结点的优先级较高。在算法实现时通常用一个最大堆来实现最大优先队列，体现最大效益优先的原则。类似地，最小优先队列规定p值较小的结点的优先级较高。在算法实现时，常用一个最小堆来实现，体现最小优先的原则。采用优先队列式分支定界算法解决具体问题时，应根据问题的特点选用最大优先或最小优先队列，确定各个结点点的p值。

装载问题

问题描述：

有一批共n个集装箱要装上2艘载重量分别为C1和C2的轮船，其中集装箱i的重量为Wi，且 $\sum_{i=1}^{n} w_i \le c_1 + c_2$

装载问题要求确定是否有一个合理的装载方案可将这个集装箱装上这2艘轮船。如果有，找出一种装载方案。

容易证明：如果一个给定装载问题有解，则采用下面的策略可得到最优装载方案。

(1)首先将第一艘轮船尽可能装满；

(2)将剩余的集装箱装上第二艘轮船。

装载问题

- 将第一艘轮船尽可能装满等价于选取全体集装箱的一个子集，使该子集中集装箱重量之和最接近。由此可知，装载问题等价于以下特殊的0-1背包问题。

$$\max \sum_{i=1}^{n} w_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

装载问题

队列式分支限界法

在算法的while循环中，首先检测当前扩展结点的左儿子结点是否为可行结点。如果是则将其加入到活结点队列中。然后将其右儿子结点加入到活结点队列中(右儿子结点一定是可行结点)。2个儿子结点都产生后，当前扩展结点被舍弃。

活结点队列中的队首元素被取出作为当前扩展结点，由于队列中每一层结点之后都有一个尾部标记-1，故在取队首元素时，活结点队列一定不空。当取出的元素是-1时，再判断当前队列是否为空。如果队列非空，则将尾部标记-1加入活结点队列，算法开始处理下一层的活结点。

装载问题

- 代码：

```
while (true) {
    if (ew + w[i] <= c) // 检查左儿子结点，w[i]=1，ew存储当前扩展结点相应的载重量
        enQueue( ew + w[i], i );
    enQueue(ew, i);                                    //右儿子结点总是可行的
    ew = ((Integer) queue.remove()).intValue();        // 取下一扩展结点
    if (ew == -1)   //如果是本层的结尾结点
    {
        if (queue.isEmpty())   return bestw;
        queue.put(new Integer(-1));                    // 同层结点尾部标志
        ew = ((Integer) queue.remove()).intValue();    // 取下一扩展结点
        i ++;                                          // 进入下一层
     }
}
```

装载问题

算法改进

节点的左子树表示将此集装箱装上船，右子树表示不将此集装箱装上船。设bestw是当前最优解；ew是当前扩展结点所相应的重量；r是剩余集装箱的重量。则当ew+r≤bestw时，可将其右子树剪去，因为此时若要船装最多集装箱，就应该把此箱装上船。

另外，为了确保右子树成功剪枝，应该在算法每一次进入左子树的时候更新bestw的值。

装载问题

```
// 检查左儿子结点
int wt = ew + w[i];
if (wt <= c)
{   // 可行结点
    if (wt > bestw) bestw = wt;
    // 加入活结点队列
    if (i < n)
        queue.put(new Integer(wt));
}
```

提前更新bestw

右儿子剪枝

```
// 检查右儿子结点
if (ew + r > bestw && i < n)
// 可能含最优解
        queue.put(new Integer(ew));
ew=((Integer)queue.remove()).intValue();
// 取下一扩展结点
```

详细代码见王晓东教材p170－171

装载问题

构造最优解

为了在算法结束后能方便地构造出与最优值相应的最优解，算法必须存储相应子集树中从活结点到根结点的路径。为此目的，可在每个结点处设置指向其父结点的指针，并设置左、右儿子标志。找到最优值后，可以根据parent回溯到根节点，找到最优解。

```
private static class QNode
{
    QNode parent;        //父结点
    boolean leftChild;   //左儿子标志
    int weight;          //结点所相应的载重量
}
```

```
for (int j = n; j > 0; j--)
{
    bestx[j] = (e.leftChild) ? 1 : 0;
    e = e.parent;
}
```

装载问题

优先队列式分支限界法

解装载问题的优先队列式分支限界法用最大优先队列存储活结点表。活结点x在优先队列中的优先级定义为从根结点到结点x的路径所相应的载重量再加上剩余集装箱的重量之和。

优先队列中优先级最大的活结点成为下一个扩展结点。以结点x为根的子树中所有结点相应的路径的载重量不超过它的优先级。子集树中叶结点所相应的载重量与其优先级相同。

在优先队列式分支限界法中，一旦有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的解即为最优解。此时可终止算法。

0-1背包问题

- 算法思想：

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。

在下面优先队列分支限界法中，节点的优先级由已装袋的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和。

算法首先检查当前扩展结点的左儿子结点的可行性。如果该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。当扩展到叶节点时为问题的最优值。

0-1背包问题

```
MaxKnapsack( n, c, w[ ], p[ ] )
{   //优先队列式分支限界法，返回最大价值，n为物品数目，c为背包容量，w为物品重量，p为物品价值
    //算法开始之前，已经按照物品单位价值率按照降序顺序排列好了
    cw = 0, cp = 0;   //cw为当前装包重量，cp为当前装包价值
    bestp = 0;  //当前最优值
    i=1,  up = Bound(1) ;  //函数Bound(i)计算当前结点相应的价值上界
    while( i != n+1 ) { //非叶子结点
        //首先检查当前扩展结点的左儿子结点为可行结点
        if( cw + w[i] <= c ) { //左孩子结点为可行结点
            if( cp + p[i] > bestp )  bestp = cp + p[i];
            AddLiveNode( up, cp + p[i] + cw + w[i], true, i + 1);  //将左孩子结点插入到优先队列中
        }
        up = Bound( i+1);
        //检查当前扩展结点的右儿子结点
        if( up >= bestp )  //右子树可能包含最优解
            AddLiveNode( up, cp, cw, false , i + 1);   //将右孩子结点插入到优先队列中
        //从优先级队列（堆数据结构）中取下一个扩展结点N
        H->DeleteMax( N) ;
        i = N.level;
    }
}
```

分支限界搜索过程

0-1背包问题

```
Bound( i )
{ //计算结点所对应的价值的上界
    cleft = c - cw;      //剩余背包容量
    b = cp;              //价值上界
    //以物品单位重量价值递减顺序装填剩余容量
    while( i <= n  &&  w[i] <= cleft ){
          cleft -= w[i];          //w[i]表示i物品的重量
          b += p[i];              //p[i]表示i物品的价值
          i++;
    }
    //装填剩余容量装满背包
    if( i<=n) b += p[i]/w[i] * cleft;
    return b;
}
```

谢谢！

Q & A

作业：设有*n*件工作分配给*n*个人。将工作*j*分配给第*i*个人所需的费用为*cij*。试设计一个算法，为每个人都分配1件不同的工作，并使总费用达到最小。设计一个算法，计算最佳工作分配方案，使总费用达到最小。