第六章　分支限界法

第六章　分支限界法

本章主要知识点

- 6.1　分支限界法的基本思想
- 6.2　单源最短路径问题
- 6.3　装载问题
- 6.4　布线问题
- 6.5　0－1背包问题
- 6.6　最大团问题
- 6.7　旅行售货员问题
- 6.8　电路板排列问题
- 6.9　批处理作业调度

6.1 分支限界法的基本思想

1. 分支限界法基本思想

- 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。
- 在分支限界法中，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。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。
- 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

6.1 分支限界法的基本思想

2、分支限界法与回溯法的不同

（1）求解目标：一般情况下，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在某种意义下的最优解。

（2）搜索方式的不同：回溯法以深度优先的方式搜索解空间树，而分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树。

6.1 分支限界法的基本思想

3. 常见的两种分支限界法

（1）队列式(FIFO)分支限界法

按照队列先进先出（FIFO）原则选取下一个结点为扩展结点。

（2）优先队列式分支限界法

按照优先队列中规定的优先级选取优先级最高的结点成为当前扩展结点。

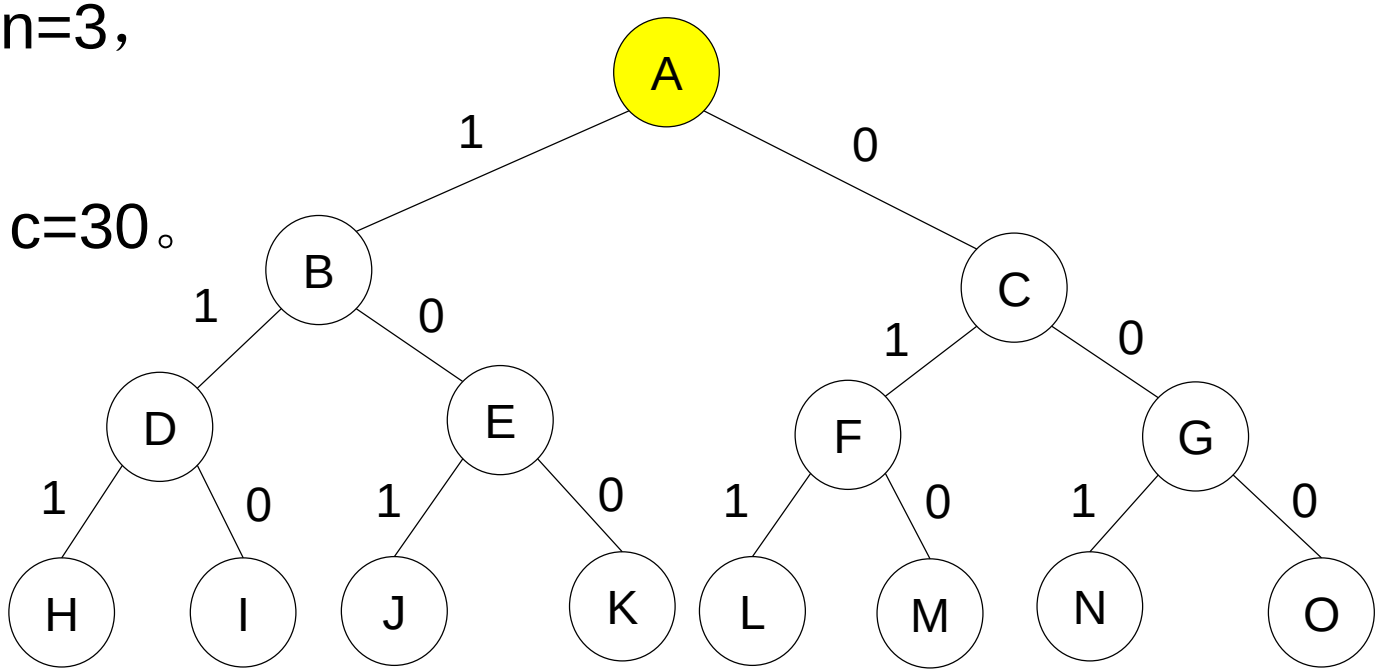
用堆实现优先队列，堆顶结点为优先级最高的结点。

队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



解空间 子集树

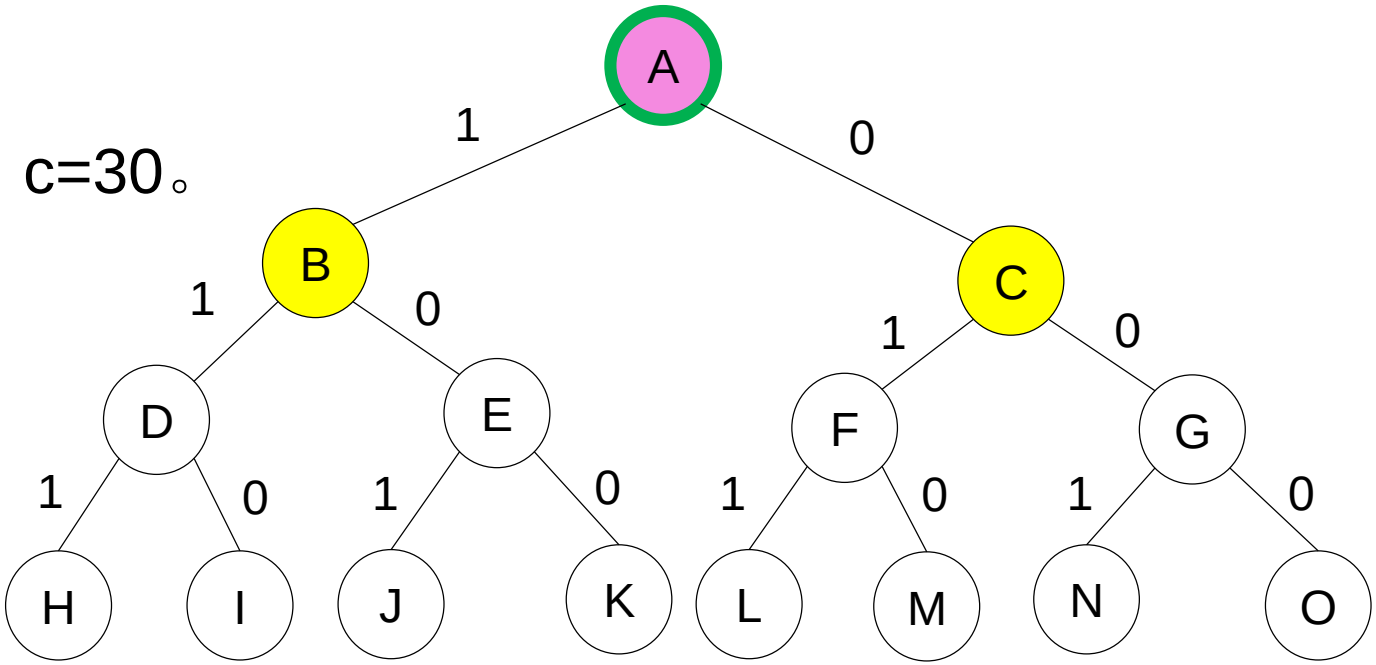
- 从A开始，初始时活结点队列为空，A是当前扩展结点；

队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



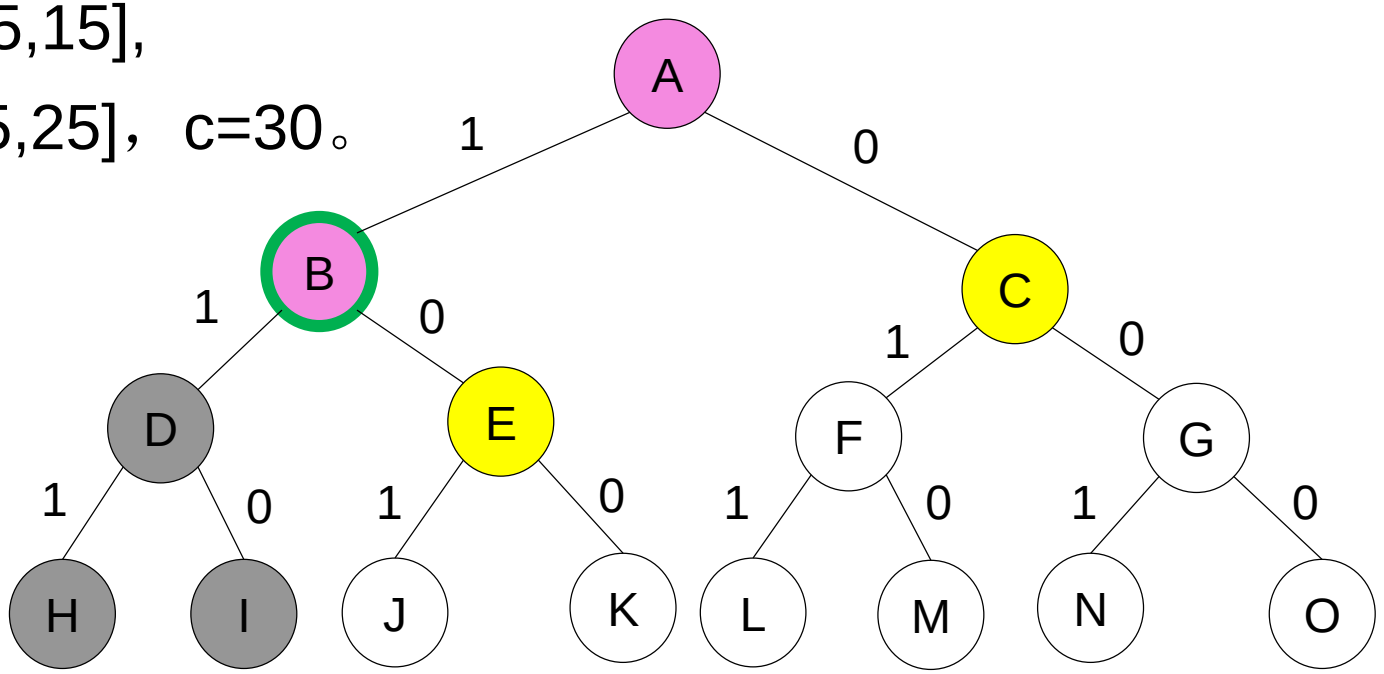
- A的2个儿子B和C均为可行结点，故将B和C依次加入队列，并舍弃A；(队列：B-C)

队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



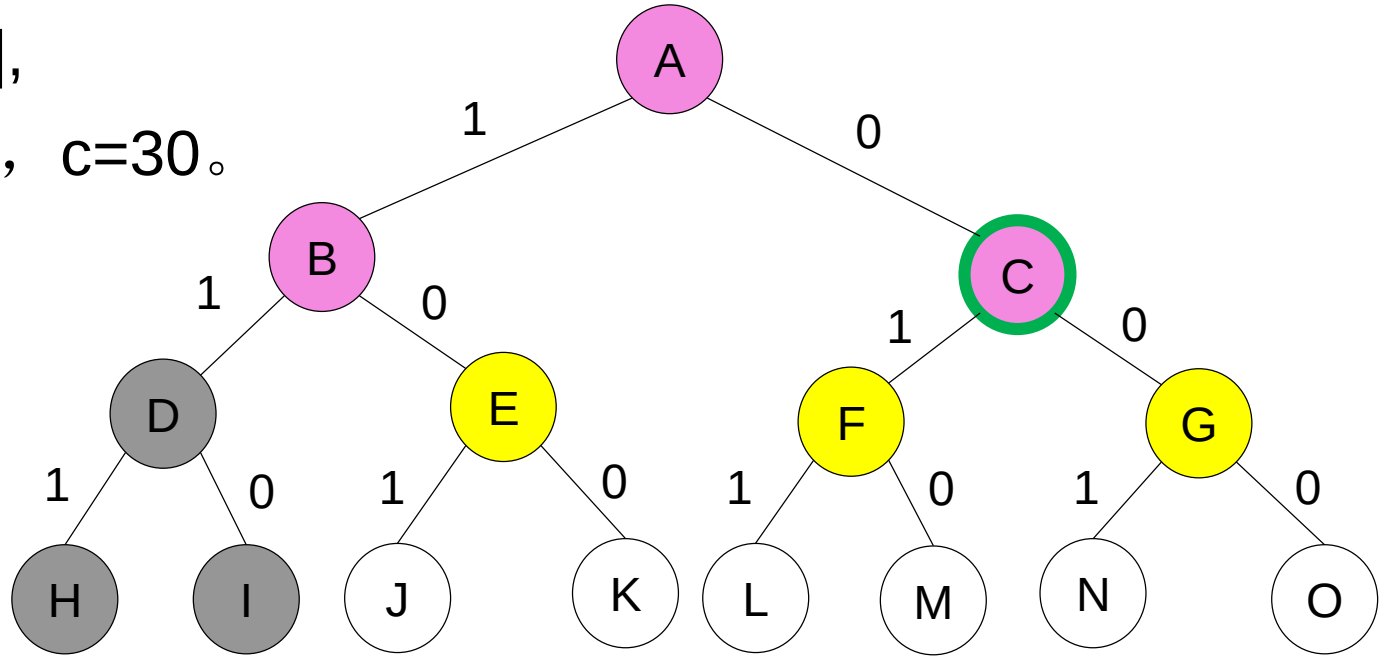
- 扩展B，它有2个儿子结点D和E，D不是可行结点舍去，E是可行结点加入队列；(队列： C-E)

队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



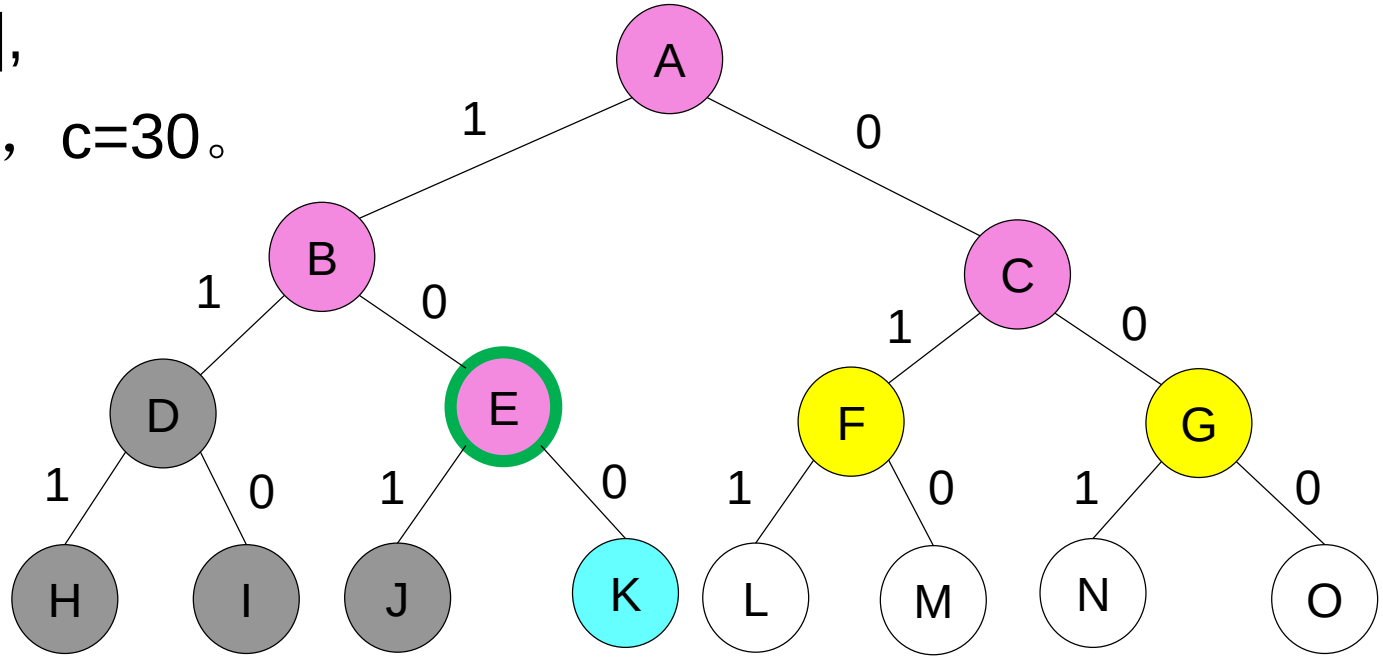
- C成为扩展结点，它的2个儿子结点F和G均为可行结点，依次加入队列；(队列： E-F-G)

队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



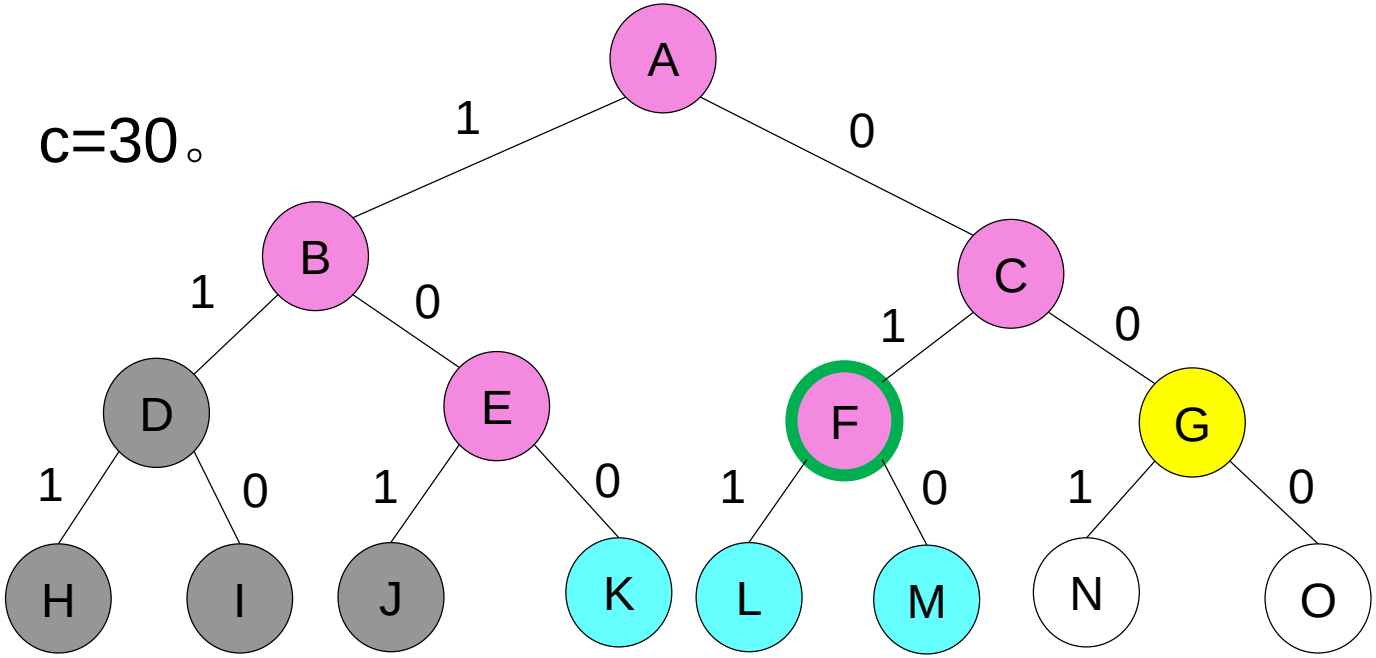
- 扩展E，得到J和K。J不可行，K是可行的叶结点，得到一个可行解，价值为45。(队列： F-G)

队列式分支限界法

0-1背包问题，n=3，

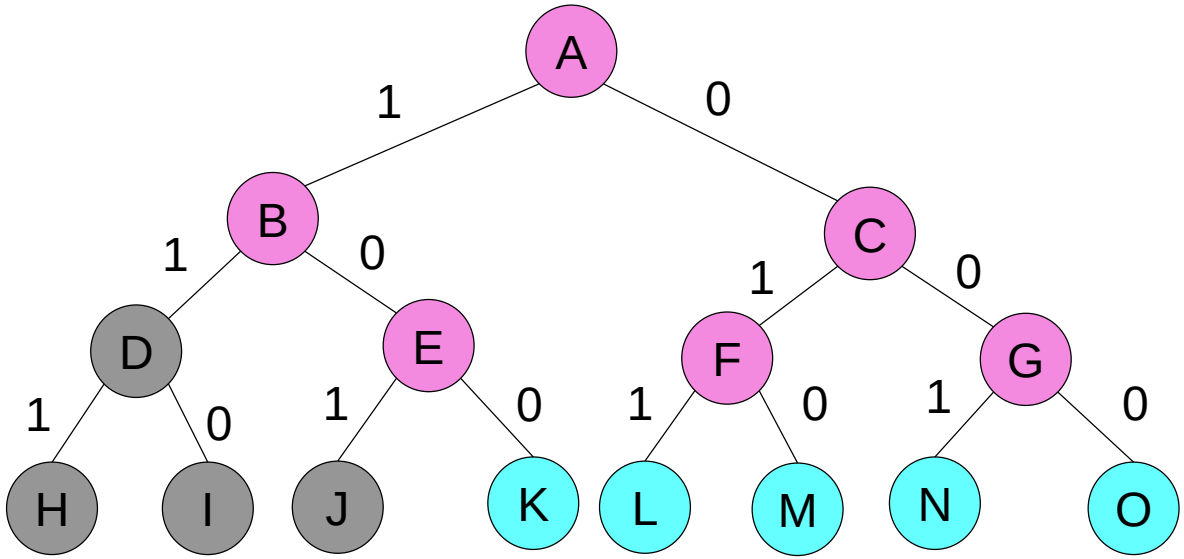
w=[16,15,15],

p=[45,25,25]，c=30。



- 扩展F，得到L和M，均为可行的叶结点。L获得价值为50的可行解，M获得价值为25的可行解。(队列： G-L-M)

队列式分支限界法



结点访问顺序：ABCEFG

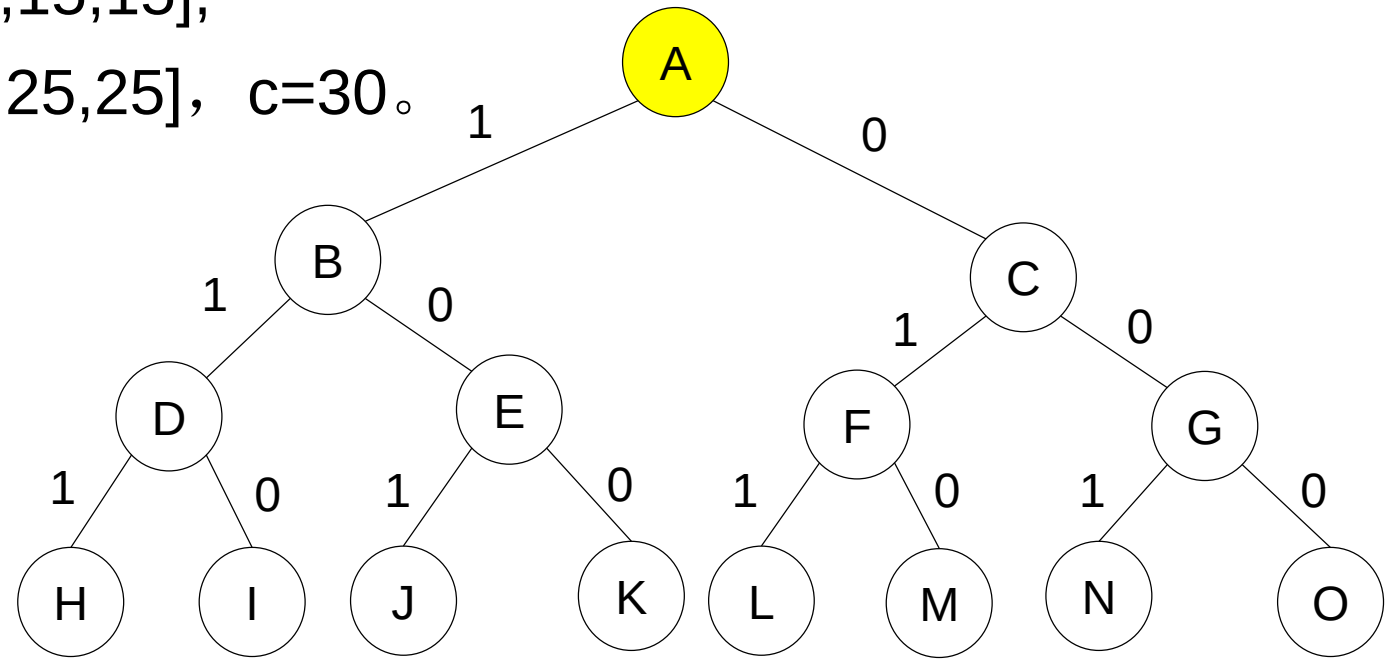
队列式分支限界法搜索空间树的方式与解空间树的广度优先遍历算法极为相似，唯一的不同之处是，队列式分支限界法不搜索以不可行结点为根的子树。

优先队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



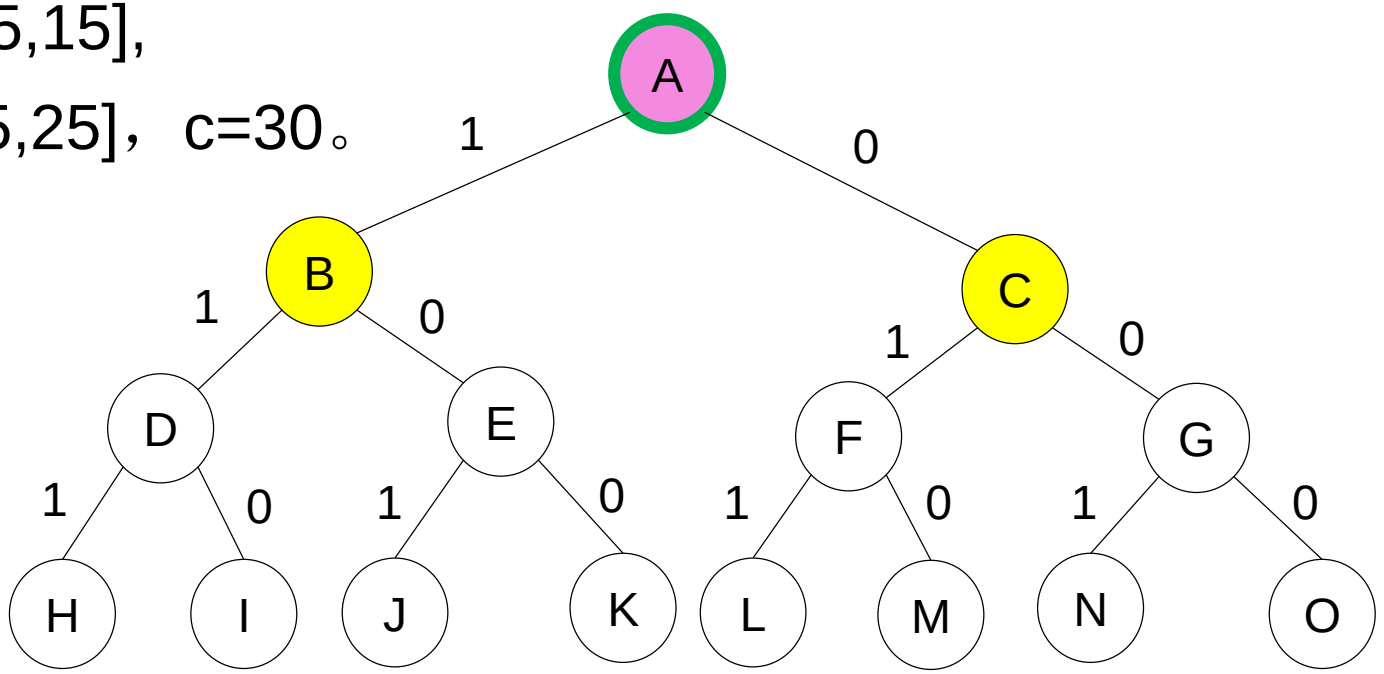
- 从A开始，用一个极大堆表示活结点表的优先队列，优先级是获得的价值。A是当前扩展结点。

优先队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



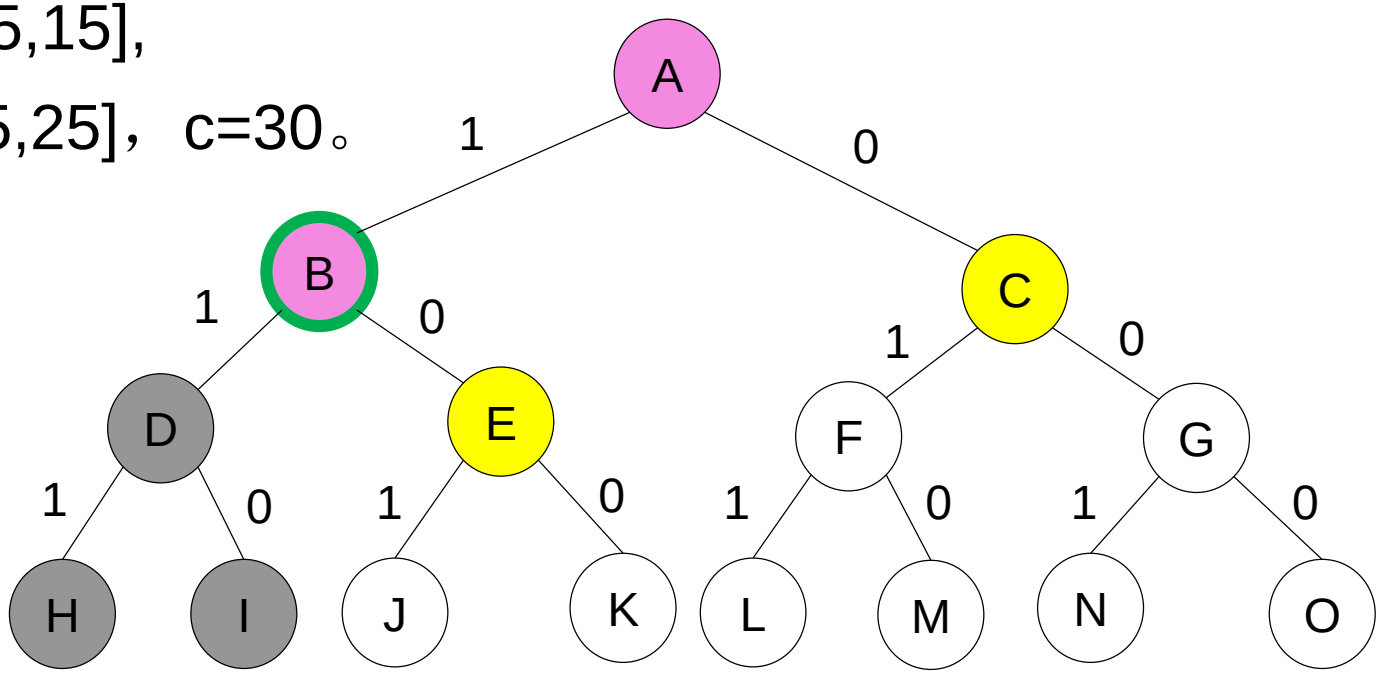
- 扩展A得到B和C，均为可行结点。B获得价值45，C获得价值0，所以B位于堆顶。

优先队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



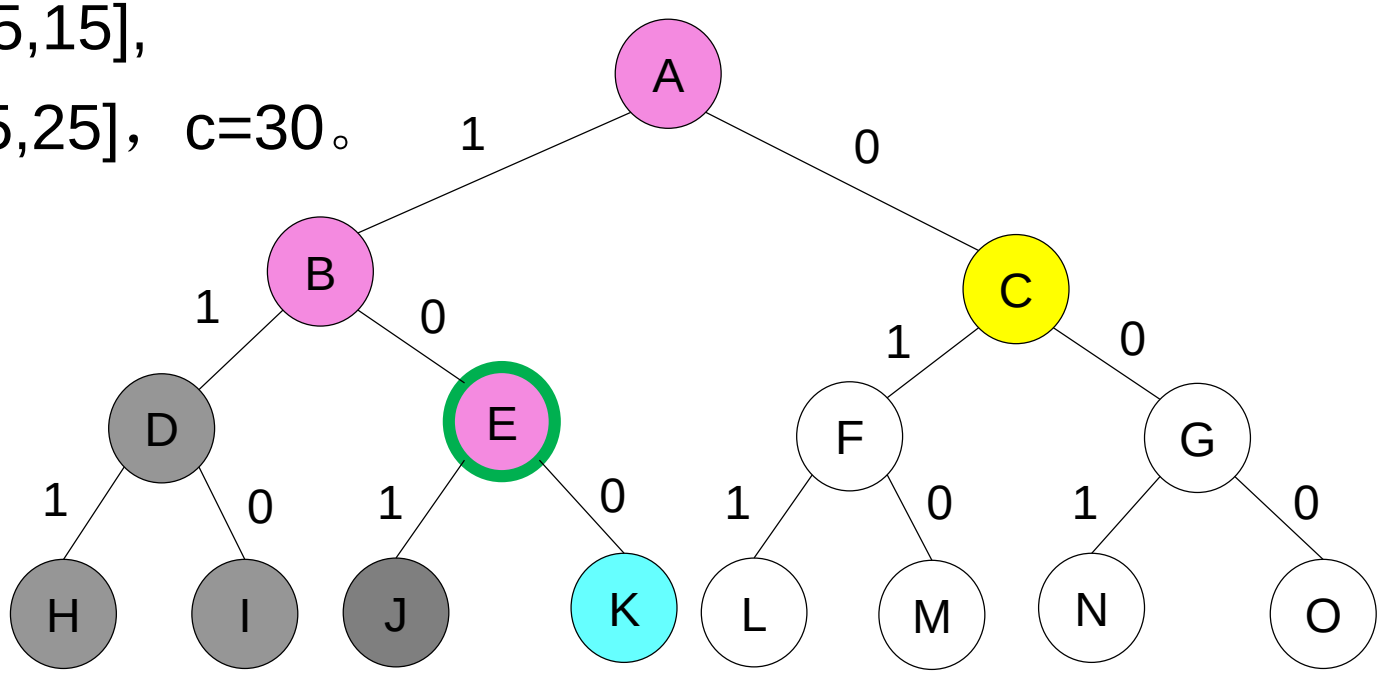
- 扩展B，它有2个儿子结点D和E，D不是可行结点舍去，E是可行结点，获得价值为45，位于堆顶；

优先队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



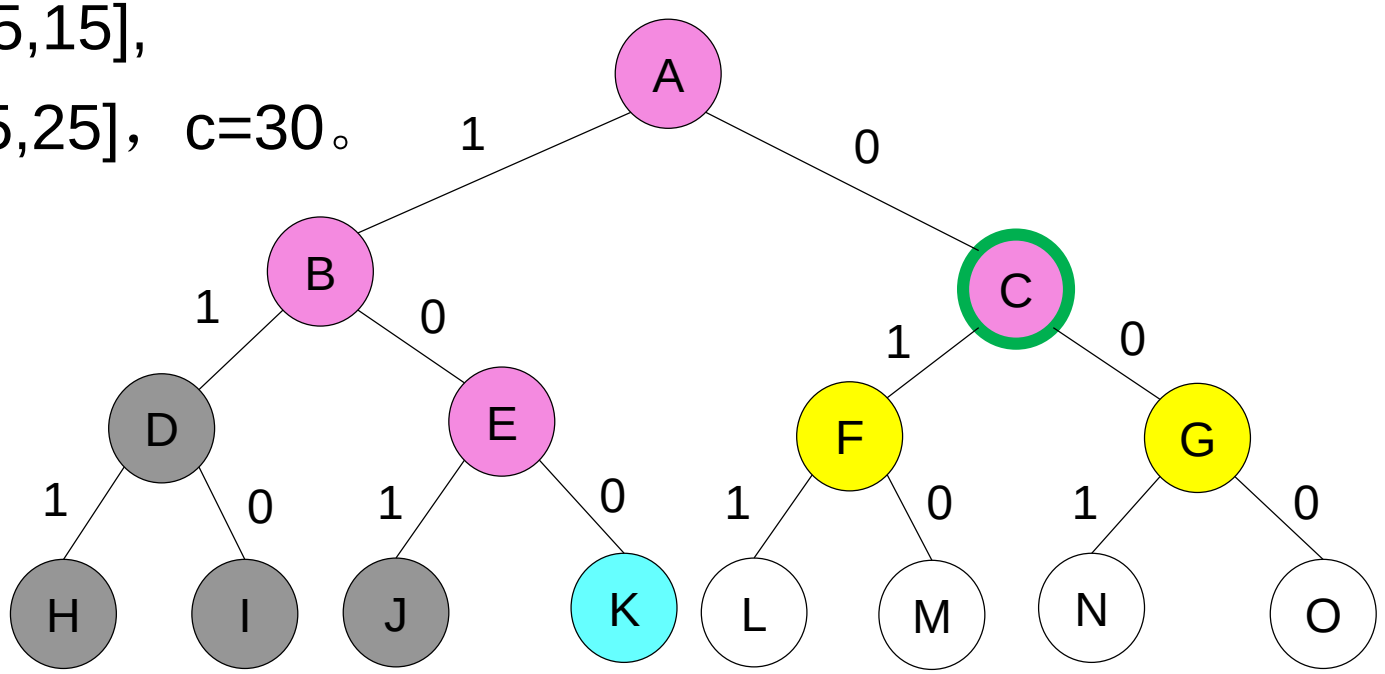
- 扩展E，得到J和K。J不可行，K是可行的叶结点，得到一个可行解，价值为45。

优先队列式分支限界法

0-1背包问题，n=3，

w=[16,15,15],

p=[45,25,25]，c=30。



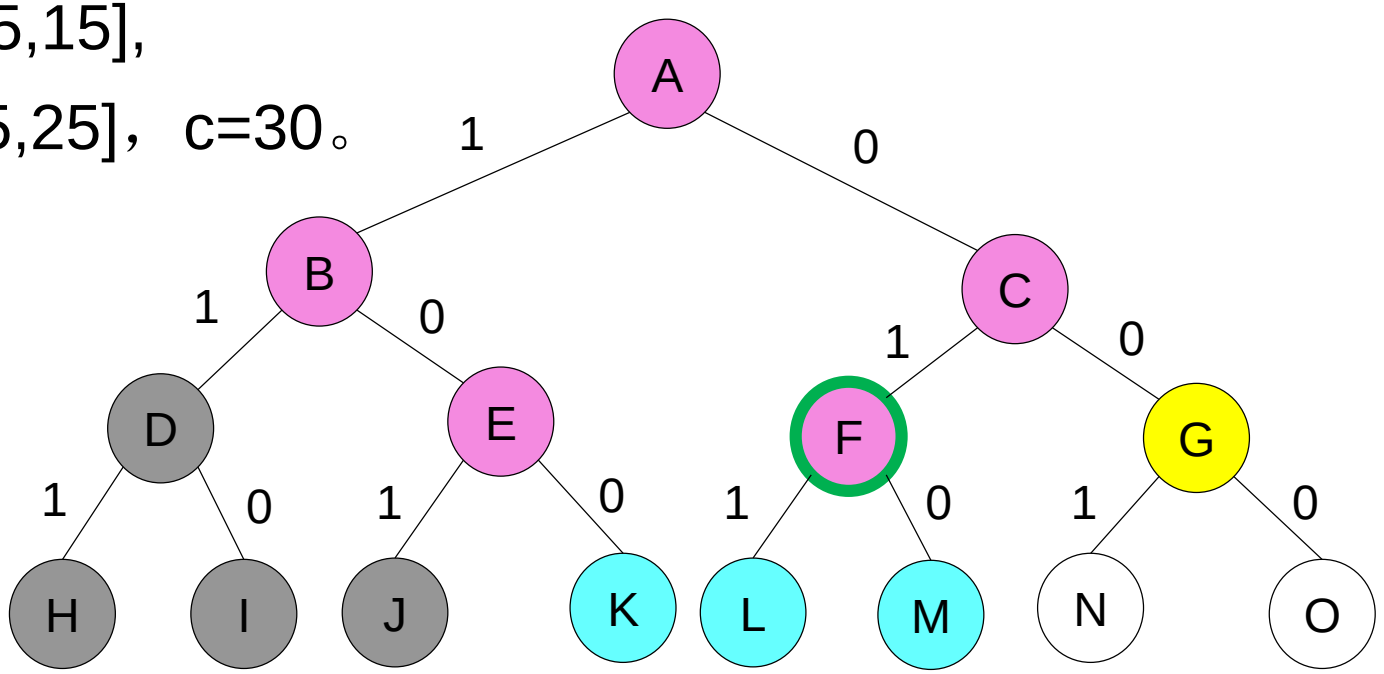
- 扩展堆中唯一元素C，它的2个儿子结点F和G均为可行结点，F获得价值25，G获得价值0，F位于堆顶；

优先队列式分支限界法

0-1背包问题，n=3，

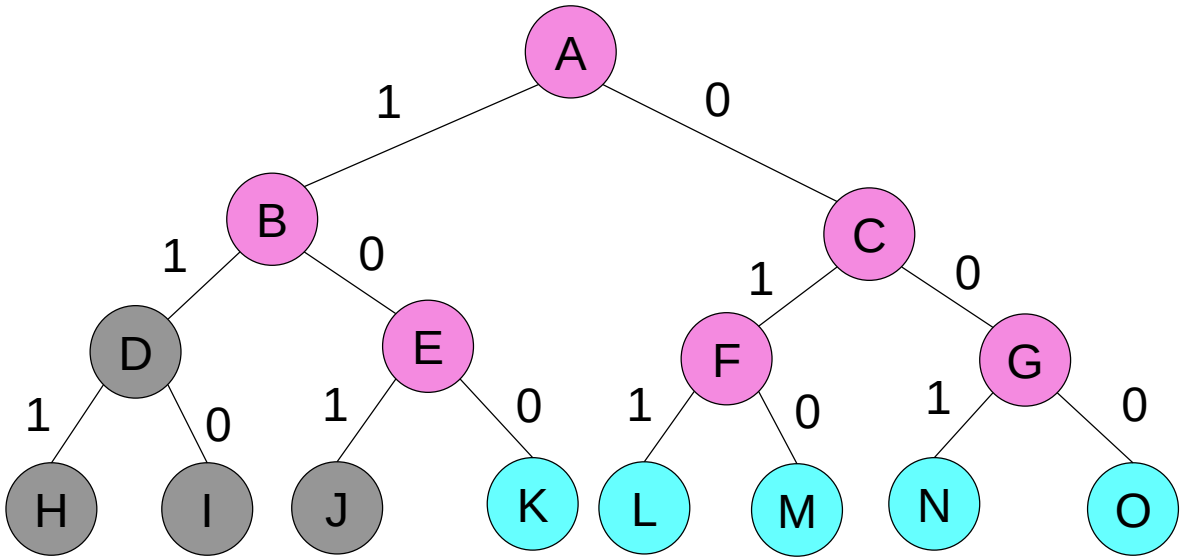
w=[16,15,15],

p=[45,25,25]，c=30。



- 扩展F，得到L和M，均为可行的叶结点。L获得价值为50的可行解，M获得价值为25的可行解。

优先队列式分支限界法



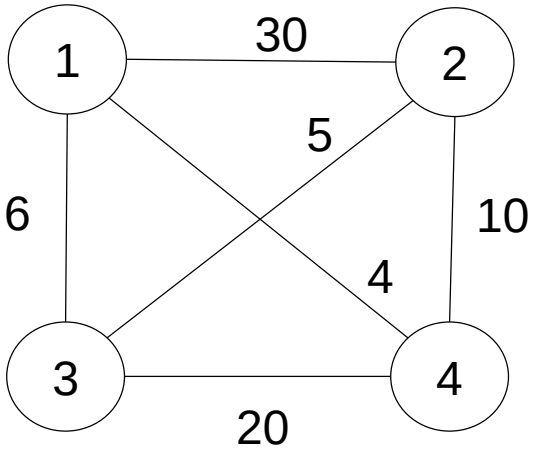
结点访问顺序：ABECFG

分支限界法

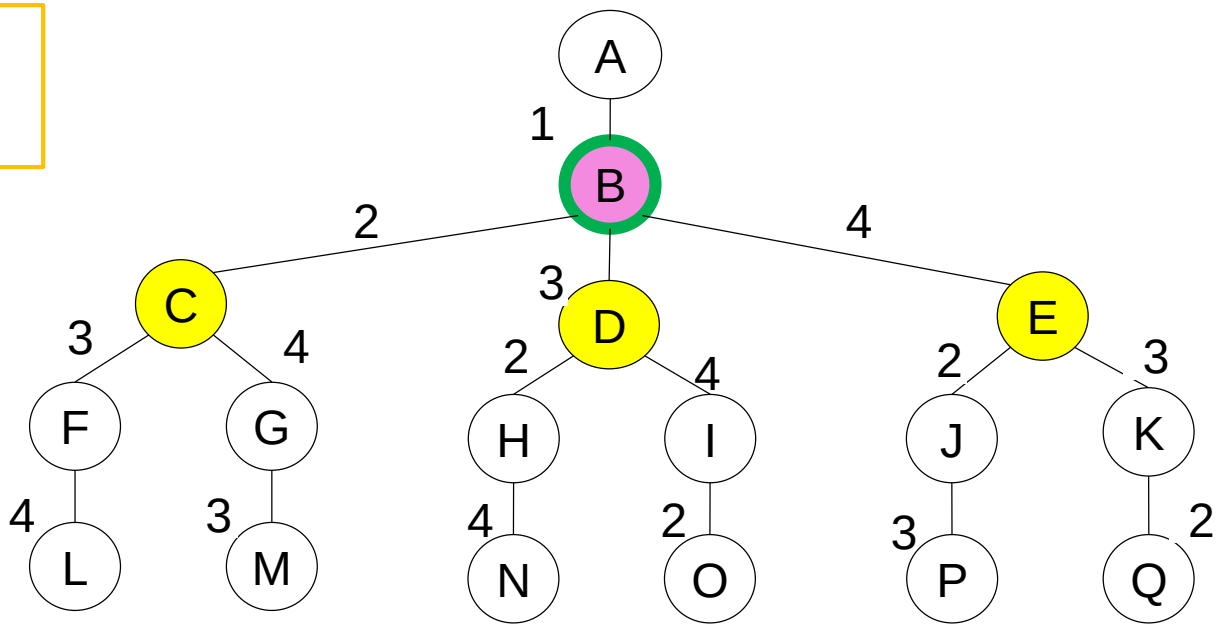
- 分支限界法与回溯法类似，也可用剪枝函数加速搜索；
- 剪枝函数给出每一个可行结点可能获得的最大价值的上界，如果该上界比当前最优值小，则剪去相应的子树；
- 将上界值作为优先级，选取当前扩展结点，有时可以加速找到最优解。

队列式分支限界法

售货员问题：从城市1出发，访问所有城市后回到1。



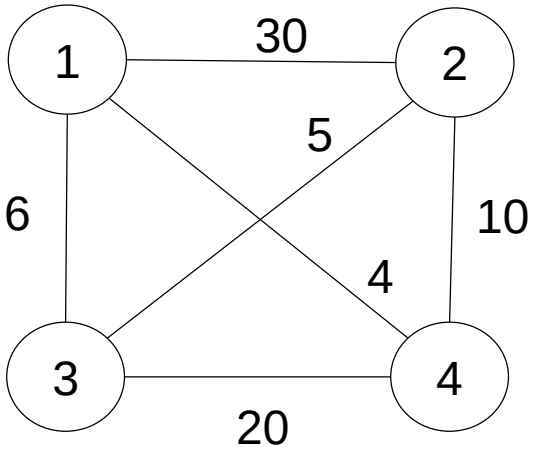
图G



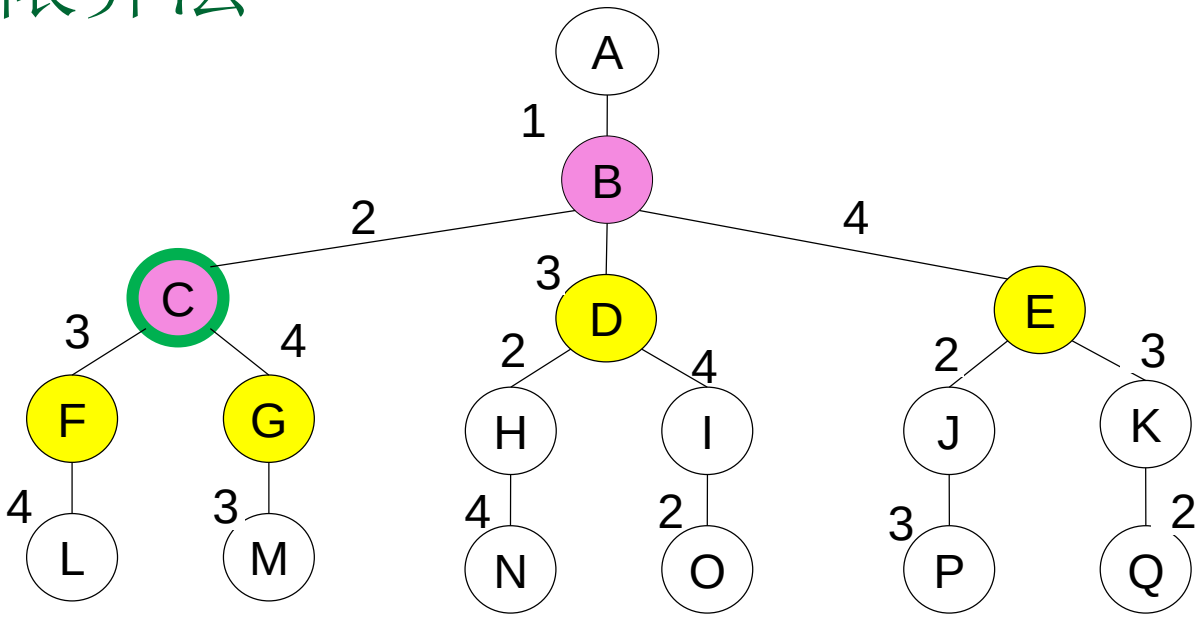
解空间 排列树

B是初始扩展结点，活结点队列为空。图G中顶点1和2、3、4均相连，因此B的儿子结点C、D、E均为可行结点，加入活结点队列中，并舍去B；

队列式分支限界法



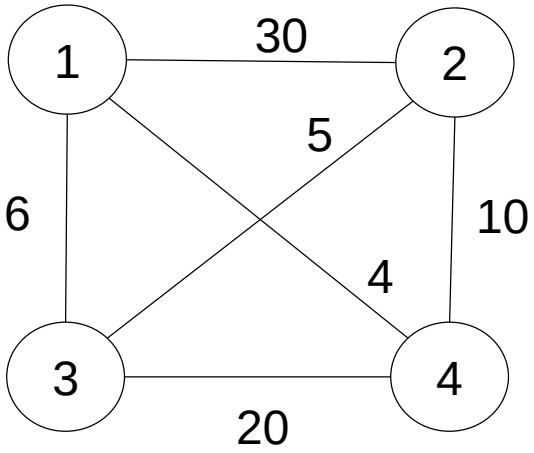
图G



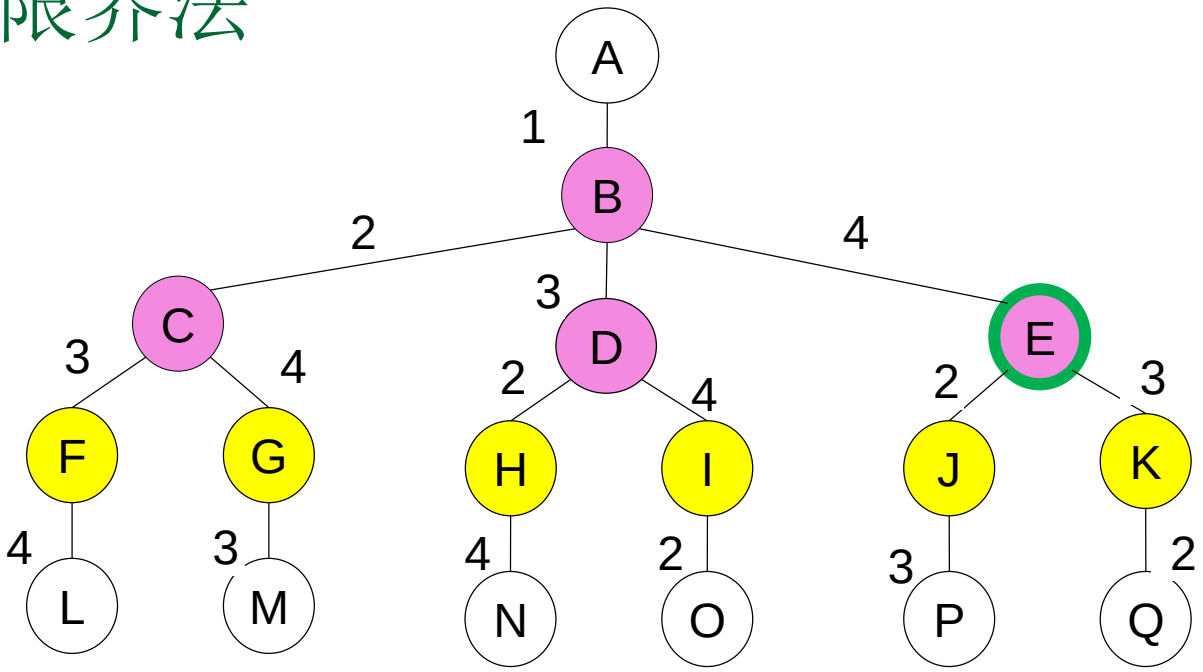
解空间 排列树

队首元素C成为扩展结点，图G中顶点2和3、4均相连，故C的儿子结点F和G均为可行结点，加入活结点队列中；

队列式分支限界法



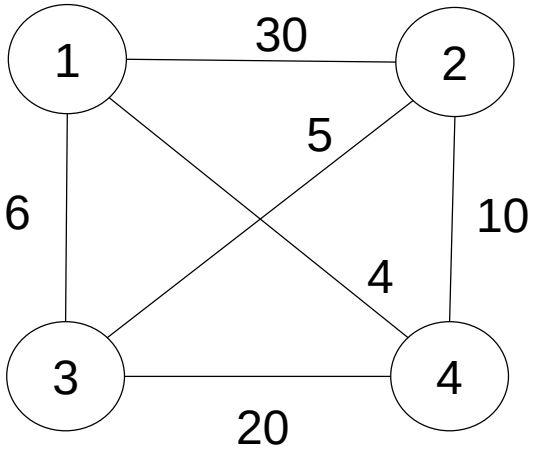
图G



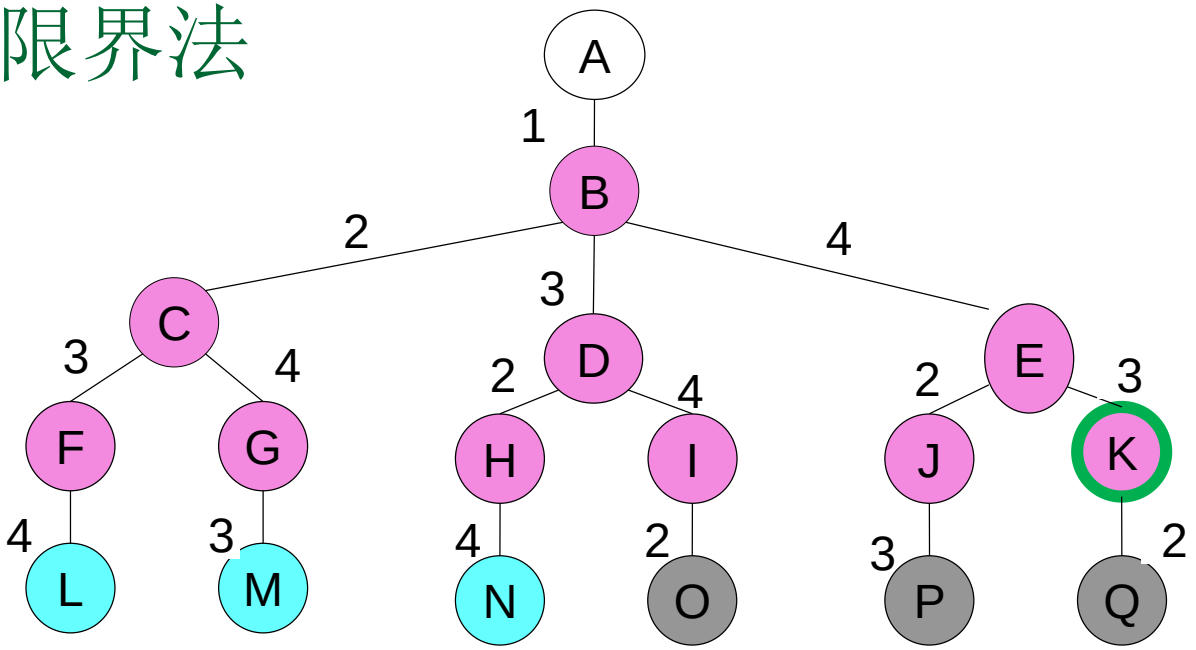
解空间 排列树

接下来D和E相继成为扩展结点，此时活结点队列中的结点依次为F、G、H、I、J、K。

队列式分支限界法



图G



解空间 排列树

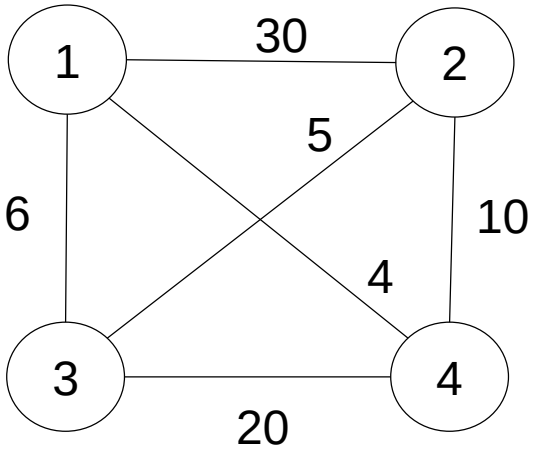
从F扩展至其儿子结点L为叶结点，找到了一条回路，费用为59；

从G扩展至其儿子结点M为叶结点，找到了一条回路，费用为71；

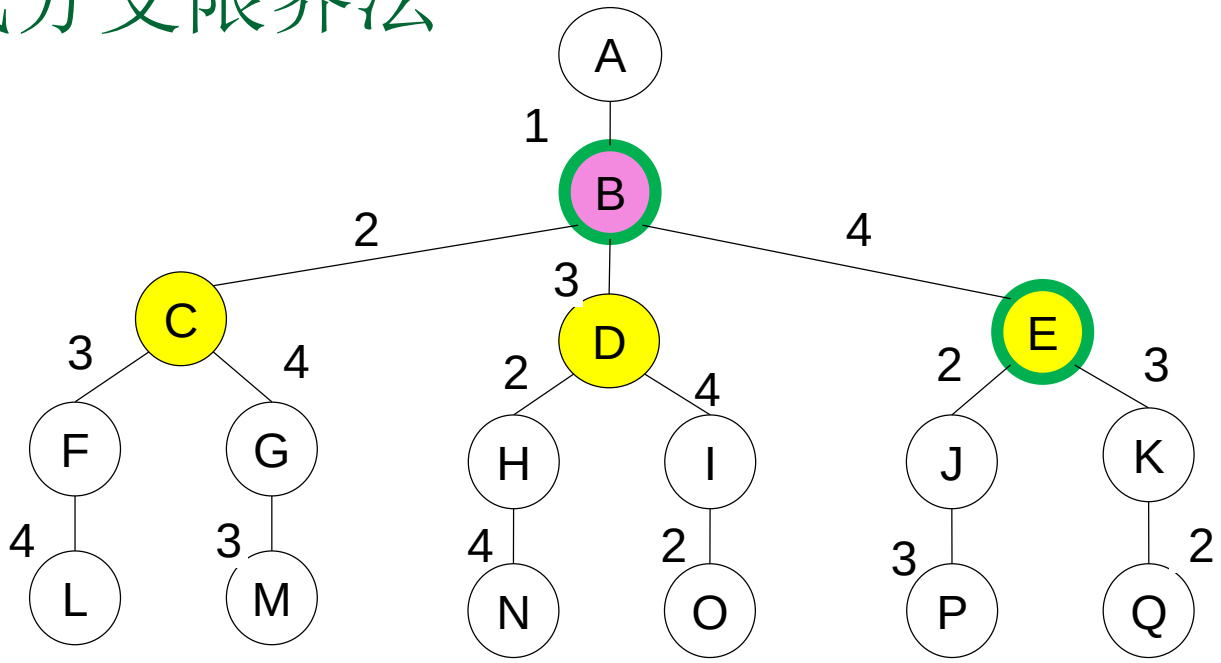
从H扩展至其儿子结点N为叶结点，找到了一条回路，费用为25；

从I、J、K扩展至O、P、Q的费用均超过当前最优值25，剪枝；

优先队列式分支限界法



图G



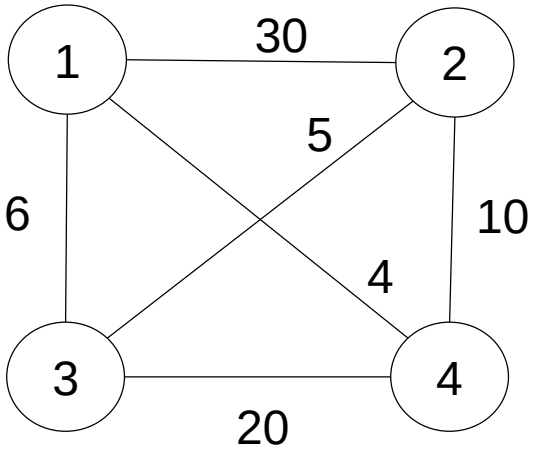
解空间 排列树

用极小堆存储活结点表，优先级是结点的当前费用。

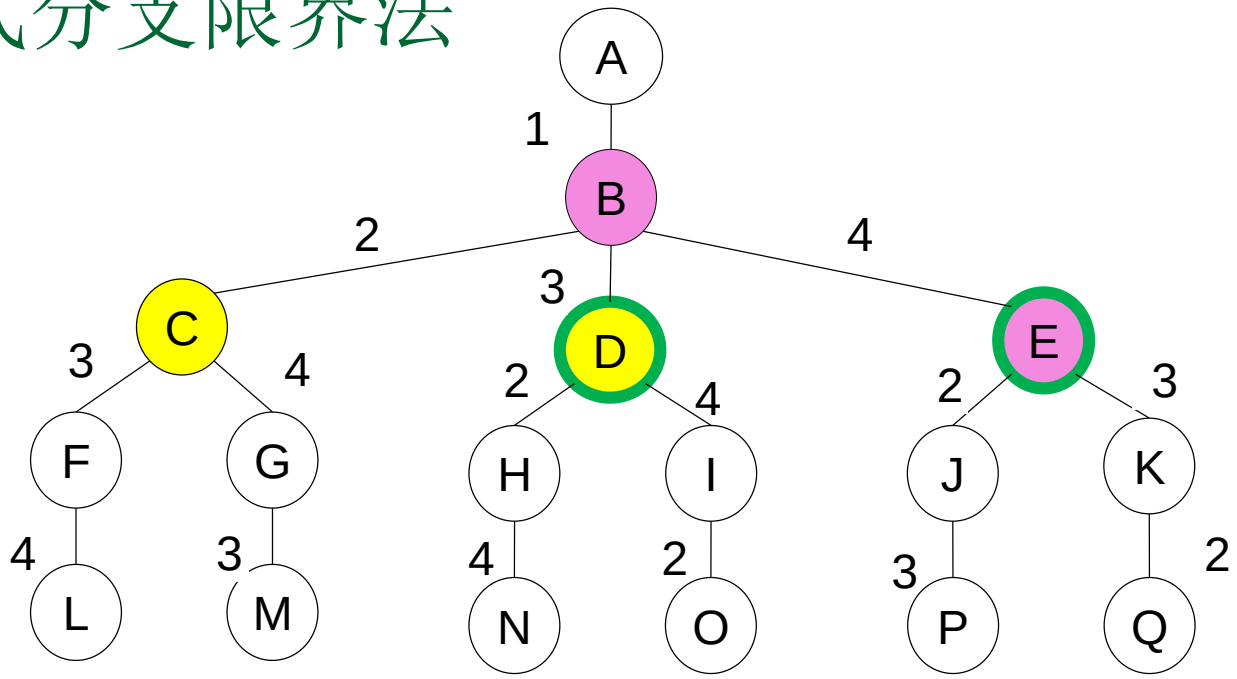
从根结点B开始，初始优先队列为空。

B的儿子结点C、D、E均为可行结点，插入堆中。此时E在堆中具有最小当前费用，成为堆顶元素。

优先队列式分支限界法



图G



解空间 排列树

扩展E，其儿子J和K均是可行结点，加入堆中，费用分别为14和24。此时D是堆顶元素。

依此方式，直至访问完所有结点。

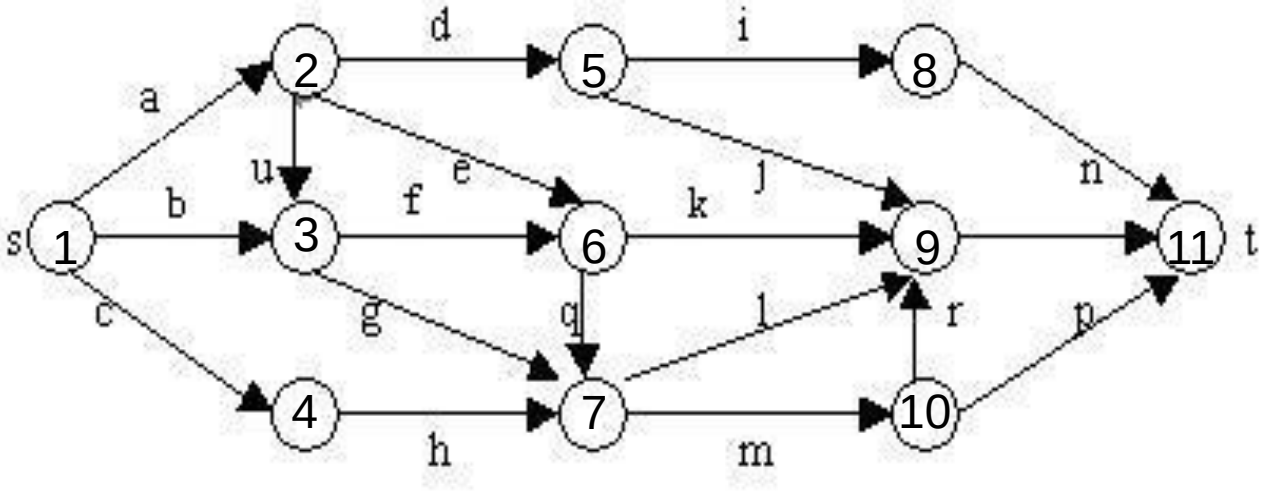
优先队列式分支限界法

- 此城市旅行售货员问题，也可用剪枝函数加速搜索；
- 剪枝函数给出每一个可行结点所需费用的最小价值的下界，如果该下界比当前最优值大，则剪去相应的子树；
- 将下界值作为优先级，选取当前扩展结点，有时可以加速找到最优解。

6.2 单源最短路径问题

1. 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。



边	权	边	权
a	2	h	2
b	3	l	5
c	4	m	1
d	7	r	2
e	2	p	2
f	9	u	3
g	2		

6.2 单源最短路径问题

2. 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。

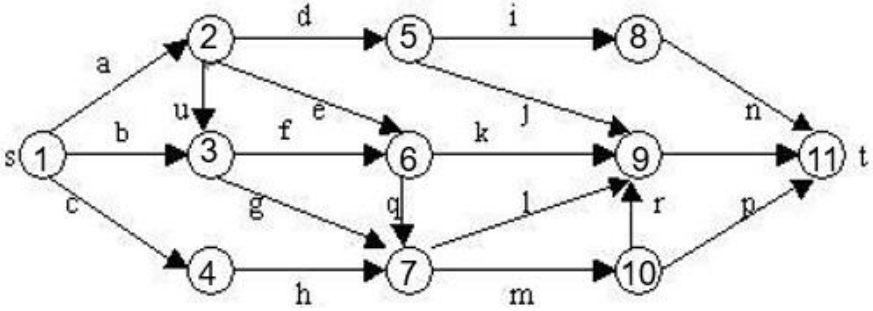
- 算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。
- 此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。
  - 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j所相应路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

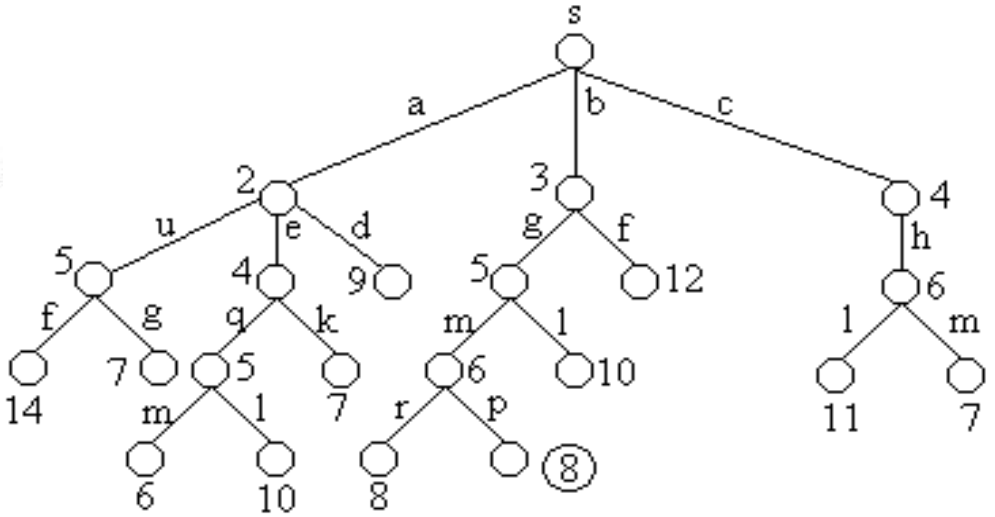
6.2 单源最短路径问题

3. 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，因此可以将路长较长的路径所对应的树中结点为根的子树剪去。





- s出发有三个子结点2,3,4被放入队列当中，路径为a,b,c，最短路长为2,3,4，分别获得当前最短路径，放入优先队列中；
- 选出堆顶结点2进行扩展至其子结点3,5,6，路径为a-u, a-d,a-e,路长为5,9,4，结点3没有获得当前最短路径，故不放入队列中，结点5和6放入优先队列中；
- 选出堆顶结点3进行扩展至其子结点6,7,路径为b-f和b-g，路长为12和5，结点6没有获得当前最短路径，故不放入队列;
- 依此方式继续搜索…

边	权	边	权
a	2	h	2
b	3	l	5
c	4	m	1
d	7	r	2
e	2	p	2
f	9	u	3
g	2		

6.2 单源最短路径问题

```
Class HeapNode{
    int i; //结点编号
    int length //当前路长
}
```

```
HeapNode enode = new HeapNode(0, 0);
while (true){  // 搜索问题的解空间
    for (int j=1;j<=n;j++)
        if(a[enode.i][j] <inf && 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;  //前驱结点
            enode = new HeapNode(j,dist[j]);
            heap.put(enode);   // 加入活结点优先队列
    if (heap.isEmpty())
        break;
    else
        enode = heap.removeMin();
}
```

6.3 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n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。

6.3 装载问题

1. 队列式分支限界法

队列Q用于存放活结点表，Q中元素值表示活结点所相应的当前载重量。当元素的值为-1时，表示队列已到达解空间树同一层结点的尾部。

- 初始时，活结点队列只包含同层结点尾部标志-1。
- 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。

6.3 装载问题

1. 队列式分支限界法

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。

6.3 装载问题

1. 队列式分支限界法

只求出最优值

```
int maxLoading(int w[], int c){
    Queue Q; Q.put(-1); int i=1, ew = 0,
    while(true)
        if(ew + w[i] <= c) // 检查左儿子结点
            enQueue(ew+w[i],  i);
        enQueue(ew, i);  //右儿子结点总是可行的
        ew = ((Integer)Q.remove()).intValue();  //取下一扩展结点
        if(ew == -1)
            if(Q.isEmpty())
                return bestw;
            Q.put(new Integer(-1));  // 同层结点尾部标志
            ew = Q.remove(); // 取下一扩展结点
            i++;   // 进入下一层
}
```

```
void enQueue(int wt,int i){
    if(i==n)
        if(wt>bestw)
            bestw = wt
    else
        Q.put(wt);
}
```

6.3 装载问题

1. 队列式分支限界法 算法的改进

- 结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ew+r≤bestw时，可将其右子树剪去;
- 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。

6.3 装载问题

1. 队列式分支限界法 算法的改进

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

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

6.3 装载问题

1. 队列式分支限界法

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。

```
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
}
```

找到最优值后，可以根据parent回溯到根结点，找到最优解。

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

6.3 装载问题

2. 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。

6.3 装载问题

子集树中结点类型为BBnode

```
class BBnode{
      BBnode parent;     //父结点
      Boolean leftChild; //左儿子结点标志 }
```

活结点优先队列用最大堆表示，堆中元素类型为HeapNode

```
class HeapNode{
      BBnode liveNode; //树结点
      int uweight; //优先级
      int level; //树中层序号 }
```

addLiveNode将新的活结点加入到子集树中，并插入到堆中

```
void addLiveNode(int up, int lev, BBnode par, boolean ch){
      BBnode b = new BBnode(par, ch);
      HeapNode node = new HeapNode(b, up, lev);
      heap.put(node); }
```

6.3 装载问题

addLiveNode(int up, int lev, BBnode par, boolean ch)

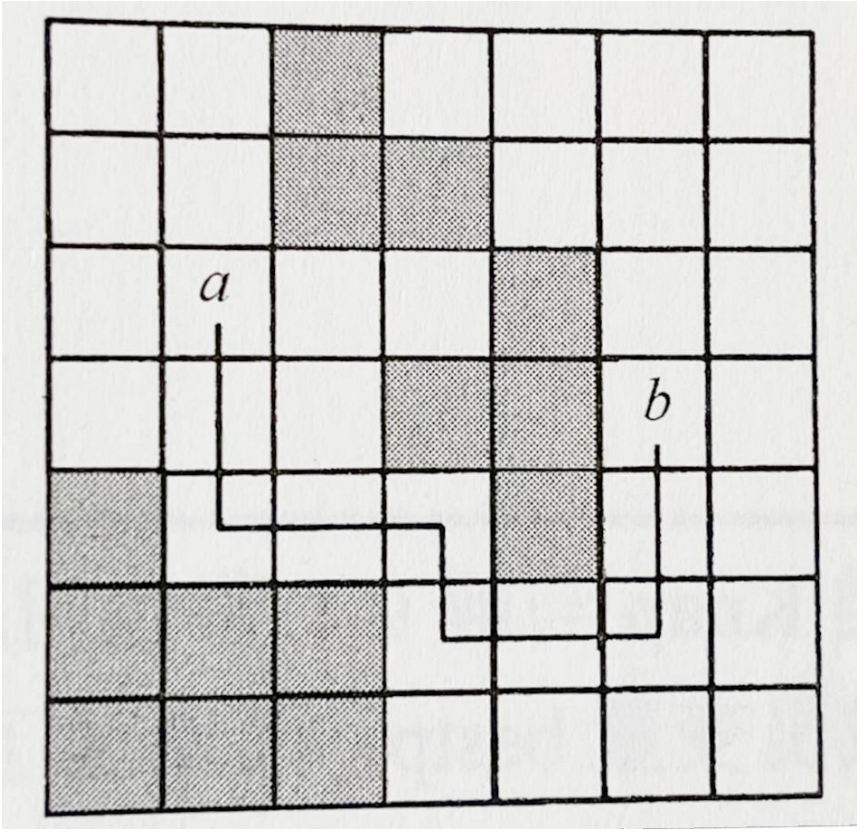
```
int maxLoading(int w[], int c){
      MaxHeap heap; BBnode e = null; int i = 1; int ew = 0;
      int r[] = new int[n+1];
      for(int j=n-1;j>0;j--)
            r[j] = r[j+1]+w[j+1];
      while(i!=n+1)
            if(ew+w[i]<=c)
                  addLiveNode(ew+w[i]+r[i], i+1, e, true);
            addLiveNode(ew+r[i], i+1, e, false);
            //取下一扩展结点……
      return ew;
}
```

2. 优先队列式分支限界法

改进策略：

- 在活结点插入优先队列前测试载重量上界是否大于当前最优值，如果不是，则剪枝；
- 将由于最优值的增加而产生的无效活结点从优先队列中删除。

6.4 布线问题



- 印刷电路板将布线区域划分成n*m个方格阵列；
- 电路布线问题要求确定连接方格a和方格b的中点的最短布线方案。
- 布线时，电路只能沿直线或直角布线。
- 已布了线的方格做了封锁标记，图中以灰色表示。

6.4 布线问题

算法的思想

- 解此问题的队列式分支限界法从起始位置a开始将它作为第一个扩展结点。与该扩展结点相邻并且可达的方格成为可行结点被加入到活结点队列中，并且将这些方格标记为1，即从起始方格a到这些方格的距离为1。
- 接着，算法从活结点队列中取出队首结点作为下一个扩展结点，并将与当前扩展结点相邻且未标记过的方格标记为2，并存入活结点队列。这个过程一直继续到算法搜索到目标方格b或活结点队列为空时为止。

6.4 布线问题

算法的思想

定义一个表示电路板上方格位置的类Position，它有两个成员row和col分别表示方格所在的行和列。

沿右、下、左、上四个方向的移动分别标记为0、1、2、3。

```
Position [] offset = new Position [4];
offset[0] = new Position(0,1);  // 右
offset[1] = new Position(1,0);  // 下
offset[2] = new Position(0,-1); // 左
offset[3] = new Position(-1,0); // 上
```

offset[i].row和offset[i].col给出了沿四个方向相对于当前方格的相对位移。

6.4 布线问题

算法的思想

二维数组grid表示所给的方格阵列，初始时，grid[i][j]=0表示该方格允许布线，grid[i][j]=1表示该方格被封锁。移动时，grid[i][j]表示方格（i，j）距起始方格的距离，起始位置距离标记为2，以区别于标记开放封锁状态的0和1，最后的实际距离为标记距离减2。

为处理方格边界，在方格阵列四周增加标记为”1”的附加方格。

```
for (int i = 0; i <= size + 1; i++){
      grid[0][i] = grid[size + 1][i] = 1;    //顶部和底部
      grid[i][0] = grid[i][size + 1] = 1;    //左翼和右翼
}
```

6.4 布线问题

```
for(int i=0; i<4; i++){
      nbr.row = here.row + offset[i].row;
      nbr.col = here.col + offset[i].col;
      if(grid[nbr.row][nbr.col] == 0)    //该方格未标记
          grid[nbr.row][nbr.col] = grid[here.row][here.col] + 1;
          if((nbr.row == finish.row) && (nbr.col == finish.col))
              break;
          q.put(new Position(nbr.row, nbr.col));
}
```

找到目标位置后，可以通过回溯方法找到这条最短路径，每次向标记距离比当前方格标记距离小1的相邻方格移动，直至到达起始方格为止。

6.5 0-1背包问题

算法的思想 (优先队列分支限界法)

- 将各物品依其单位重量价值从大到小进行排列。
- 结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。
- 当扩展到叶结点时为问题的最优值。

6.5 0-1背包问题

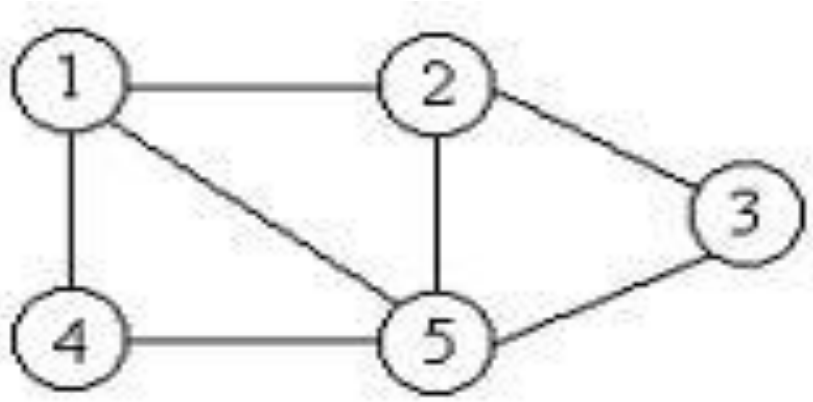
上界函数

```
double Bound(int i){
        double cleft = c - cw;  //cleft为剩余空间
        double b = cp;  //b已获得价值
        while(i<=n && w[i]<=cleft)
             cleft -= w[i];   //w[i]表示i所占空间
             b += p[i];    //p[i]表示i的价值
             i++;
        if(i<=n)
             b += p[i]/w[i]*cleft; //装填剩余容量装满背包
        return b;    //b为上界函数
}
```

AddLiveNode(double up,double cp,double cw,int lev,BBnode par,boolean ch)
up: 结点的价值上界；cp: 结点相应的价值；cw: 结点相应的重量；
lev: 活结点在子集树中所处的层序号 ; par:父结点; ch:左儿子标志

```
double MaxKnapsack{
    BBnode enode = null;
    int i=1; double cw=cp=0, bestp=0, up=Bound(1);
    while(i!=n+1)     // 非叶结点
        double wt = cw + w[i];
        if(wt<=c)      // 左儿子结点为可行结点
            if(cp+p[i]>bestp)
                bestp = cp + p[i];
                addLiveNode(up, cp+p[i], cw+w[i],i+1, enode, true);
        up = bound(i + 1);    //计算结点i+1所相应价值的上界
        if (up>=bestp)    //检查右儿子结点
            addLiveNode(up, cp, cw, i+1, enode, false);
            // 取下一个扩展结点 …
```

6.6 最大团问题



1. 问题描述

给定无向图G=(V，E)。如果

U⊆V，且对任意u，v∈U有(u，v)∈E，则称U是G的完全子图。G的完全子图U是G的团当且仅当U不包含在G的更大的完全子图中。G的最大团是指G中所含顶点数最多的团。

图G中，子集{1，2}是G的大小为2的完全子图。这个完全子图不是团，因为它被G的更大的完全子图{1，2，5}包含。{1，2，5}是G的最大团。{1，4，5}和{2，3，5}也是G的最大团。

6.6 最大团问题

2. 上界函数

用变量cliqueSize表示与该结点相应的团的顶点数；level表示结点在子集空间树中所处的层次；用cliqueSize +n-level作为顶点数上界upperSize的值。

在此优先队列式分支限界法中，用upperSize作为优先队列中元素的优先级。算法总是从活结点优先队列中抽取具有最大upperSize值的元素作为下一个扩展元素。

6.6 最大团问题

3. 算法思想

- 子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其cliqueSize的值为0。
- 算法在扩展内部结点时:
  - 首先考察其左儿子结点。在左儿子结点处，将顶点i加入到当前团中，并检查该顶点与当前团中其他顶点之间是否有边相连。当顶点i与当前团中所有顶点之间都有边相连时，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点。
  - 接着继续考察当前扩展结点的右儿子结点。当upperSize>bestn时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

6.6 最大团问题

3. 算法思想

- 算法的while循环的终止条件是遇到子集树中的一个叶结点(即n+1层结点)成为当前扩展结点。
- 对于子集树中的叶结点，有upperSize＝cliqueSize。此时活结点优先队列中剩余结点的upperSize值均不超过当前扩展结点的upperSize值，从而进一步搜索不可能得到更大的团，此时算法已找到一个最优解。

6.7 旅行售货员问题

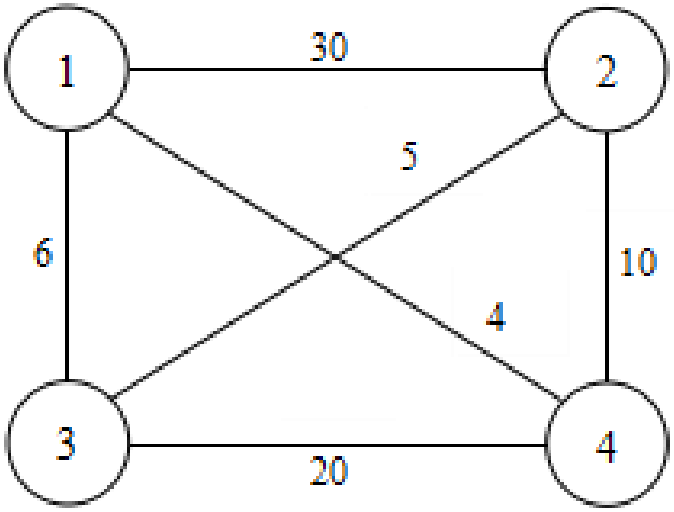
1. 问题描述

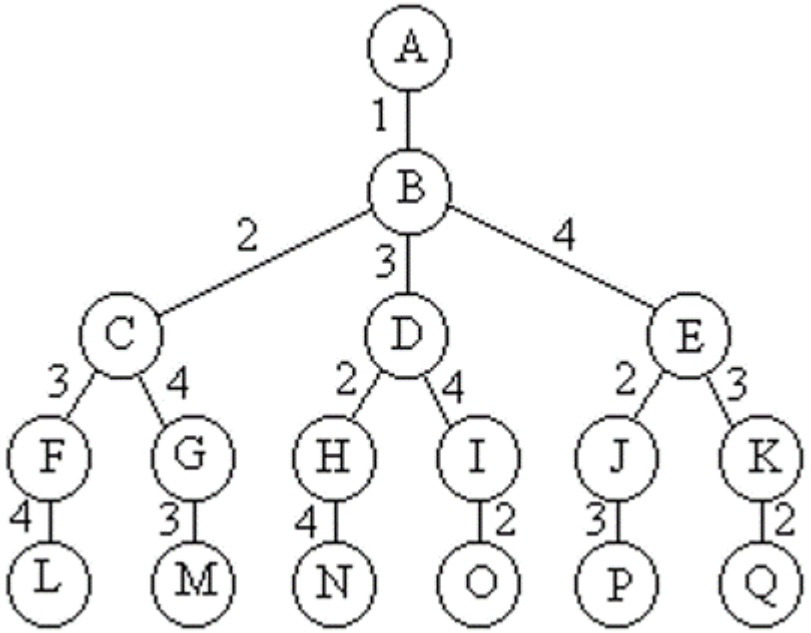
某售货员要到若干城市去推销商品，已知各城市之间的路程(或旅费)。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程(或总旅费)最小。

路线是一个带权图。图中各边的权为正数。图的一条周游路线是包括V中的每个顶点在内的一条回路。

旅行售货员问题的解空间可以组织成一棵树，从树的根结点到任一叶结点的路径定义了图的一条周游路线。旅行售货员问题要在图G中找出费用最小的周游路线。

6.7 旅行售货员问题



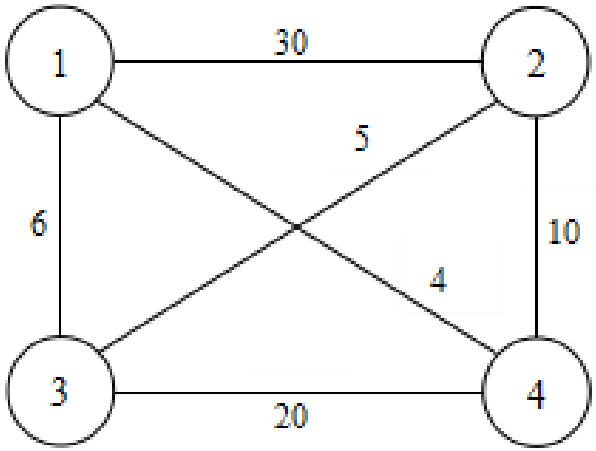


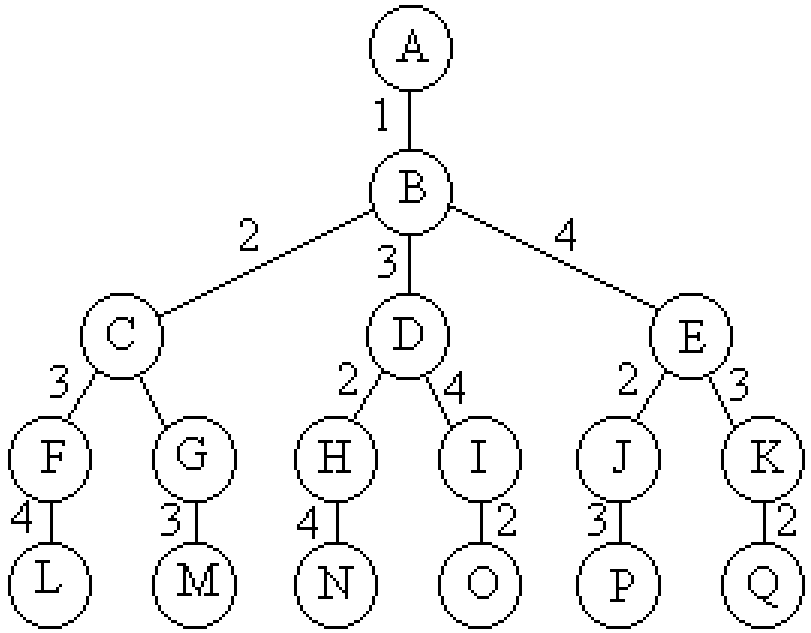
队列式：

活结点队列CDE，对于结点C、D、E，FG、HI、JK都是可行结点，于是队列中依次加入FG，HI，JK。

对于结点F，L为其叶结点并得到费用59。对于G扩展至M得到费用66。H扩展至N得到费用25，对于结点I，由于现有费用已经超过25，故无必要扩展。

6.7 旅行售货员问题





优先队列式：

使用优先队列来存储活结点，优先队列中的每个活结点都存储从根到该活结点的相应路径。

6.7 旅行售货员问题

2. 优先队列式算法描述

- 算法开始时创建一个最小堆，用于表示活结点优先队列。堆中每个结点的子树费用的下界lcost=cc+rcost，lcost值是优先队列的优先级。
- 接着算法计算出图中每个顶点的最小费用出边并用minout记录，处理结点i时，rcost=$\sum_{j=i+1}^{n} minout[j]$。如果所给的有向图中某个顶点没有出边，则该图不可能有回路，算法即告结束。如果每个顶点都有出边，则根据计算出的minout作算法初始化。

6.7 旅行售货员问题

2. 算法描述

算法的while循环体完成对排列树内部结点的扩展。对于当前扩展结点，算法分2种情况进行处理：

① 首先考虑s=n-2的情形，此时当前扩展结点是排列树中某个叶结点的父结点。如果该叶结点相应一条可行回路且费用小于当前最小费用，则将该叶结点插入到优先队列中，否则舍去该叶结点。

6.7 旅行售货员问题

2. 算法描述

对于当前扩展结点，算法分2种情况进行处理：

② 当s<n-2时，算法依次产生当前扩展结点的所有儿子结点。由于当前扩展结点所相应的路径是x[0:s]，其可行儿子结点是从剩余顶点x[s+1:n-1]中选取的顶点x[i]，且(x[s]，x[i])是所给有向图G中的一条边。对于当前扩展结点的每一个可行儿子结点，计算出其前缀(x[0:s]，x[i])的费用cc和相应的下界lcost。当lcost<bestc时，将这个可行儿子结点插入到活结点优先队列中。

6.7 旅行售货员问题

算法中while循环的终止条件是排列树的一个叶结点成为当前扩展结点。

当s=n-1时，已找到的回路前缀是x[0:n-1]，它已包含图G的所有n个顶点。因此，当s=n-1时，相应的扩展结点表示一个叶结点。此时该叶结点所相应的回路的费用等于当前费用cc和子树费用的下届lcost的值。剩余的活结点的lcost值不小于已找到的回路的费用。它们都不可能导致费用更小的回路。因此已找到的叶结点所相应的回路是一个最小费用旅行售货员回路，算法可以结束。

6.7 旅行售货员问题

```
class HeapNode{
      float lcost,  //子树费用的下界，优先级
            cc,      //当前费用
            rcost;  //x[s:n-1]中顶点最小出边费用和
      int s;  //结点在树中的层次
      int x[];  //搜索路径 }
```

Max表示Float.MaxValue

```
float bbTSP( ){
     MinHeap heap;
     float[ ] minOut = new float[n+1];
     float minSum = 0;
     for(int i=1;i<=n;i++) //找出每个结点的最小出边，并计算最小出边和
          float min = Max;
               for(int j=1;j<=n;j++)
                    if(a[i][j]<Max)&&a[i][j]<min)
                         min = a[i][j];
               if(min==Max)
                    return Max;
           minOut[i] = min;
           minSum += min;
}
```

6.7 旅行售货员问题

HeapNode(lcost, cc, rcost, s, x[])

```
float bbTSP( )续
     int [ ] x = new int[n];
     for(int i=0;i<n;i++)
          x[i] = i+1;
     HeapNode enode = new HeapNode(0, 0, minSum, 0, x);
     float bestc = Max;
     while(enode.s<n-1)
          x = enode.x;
          if(enode.s==n-2)
              if(a[x[n-2]][x[n-1]]<Max && a[x[n-1]][1]<Max
                    && enode.cc+a[x[n-2]][x[n-1]]+a[x[n-1]][1]<bestc)
                 bestc = enode.cc+ a[x[n-2]][x[n-1]]+a[x[n-1]][1];
                 enode.cc = bestc; enode.lcost  = bestc;
                 enode.s++;
                 heap.put(enode);
}
```

6.7 旅行售货员问题

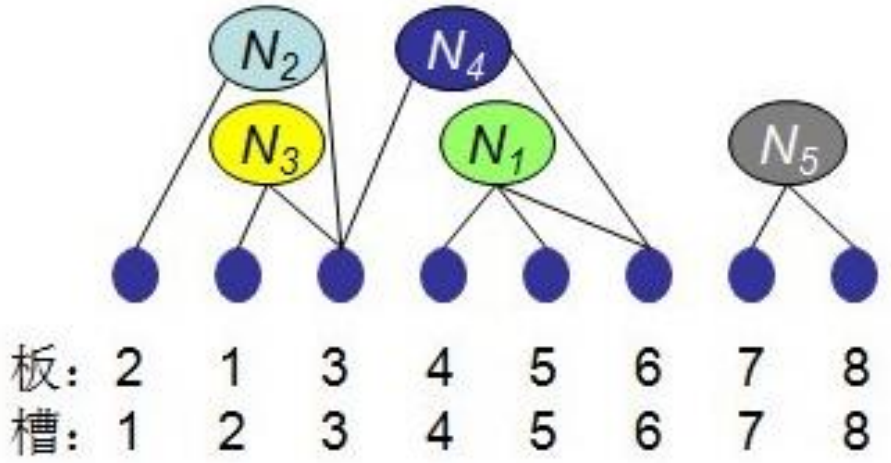
HeapNode(lcost, cc, rcost, s, x[])

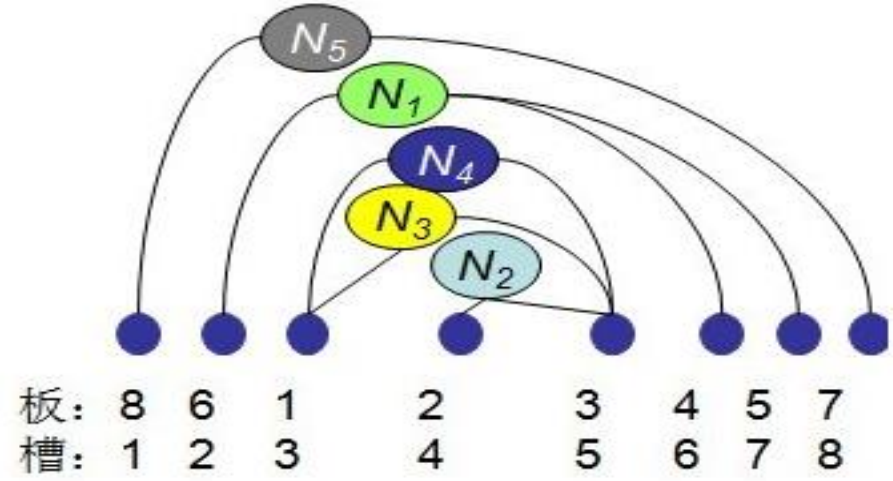
```
float bbTSP( ){ 续
   else //enode.s!=n-2
      for(int i=enode.s+1;i<n;i++)
         if(a[x[enode.s]][x[i]]<Max)
            float cc = enode.cc+a[x[enode.s]][x[i]];
            float rcost = enode.rcost – minOut[x[enode.s]];
            float b = cc+rcost;
            if(b<bestc)
               int[] xx = new int[n];
               for(int j=0;j<n;j++)
                  xx[j] = x[j];
               xx[node.s+1] = x[i];
               xx[i] = x[enode.s+1];
               HeapNode node = new HeapNode(b,cc,rcost,enode.s+1,xx);
               heap.put(node);
               //取下一扩展结点继续……
}
```

6.8 电路板排列问题

设B={1,2,…,n}是n块电路板的集合。集合L={N1, N2, …, Nm}是n块电路板的m个连接块。其中每个连接块是B的一个子集，且Ni中的电路板用同一根导线连接在一起。

电路板排列问题要求对于给定连接块，确定电路板的最佳排列，使其具有最小密度。





6.8 电路板排列问题

算法描述

- 解空间树是一棵排列树，采用优先队列式分支限界法找出最小密度布局。用当前密度作为结点的优先级。
- 算法开始时，将排列树的根结点置为当前扩展结点。在do-while循环体内算法依次从活结点优先队列中取出具有最小当前密度cd值的结点作为当前扩展结点，并加以扩展。
- 首先考虑s=n-1的情形，当前扩展结点是排列树中的一个叶结点的父结点。x表示相应于该叶结点的电路板排列。计算出与x相应的密度并在必要时更新当前最优值和相应的当前最优解。
- 当s<n-1时，算法依次产生当前扩展结点的所有儿子结点。对于当前扩展结点的每一个儿子结点node，计算出其相应的密度node.cd。当node.cd<bestd时，将该儿子结点N插入到活结点优先队列中。

6.9 批处理作业问题

1. 问题的描述

给定n个作业的集合J={J1, J2, ..., Jn}。每一个作业Ji都有2项任务要分别在2台机器上完成。每一个作业必须先由机器1处理，然后再由机器2处理。作业Ji需要机器j的处理时间为t_{ji}， i=1, 2, ..., n； j=1, 2。

对于一个确定的作业调度，设F_{ji}是作业i在机器j上完成处理的时间。则所有作业在机器2上完成处理的时间和 $f=\sum_{i=1}^{n}F_{2i}$ ，称为该作业调度的完成时间和。

批处理作业调度问题要求对于给定的n个作业，制定最佳作业调度方案，使其完成时间和达到最小。

6.9 批处理作业问题

2. 限界函数

解空间是一棵排列树，用优先队列式分支限界法。

排列树中每个结点E对应一个已安排的作业集M⊆{1,2，。。。。，n}。以结点E为根的子树所含叶结点的完成时间和可表示为：

$$f = \sum_{i \in M} F_{2i} + \sum_{i \notin M} F_{2i}$$

设|M| = r，且L是以E为根的子树中的叶结点，相应的作业调度为$\{p_k\}$(k=1,2,…,n)，p_k 是第k个安排的作业。

6.9 批处理作业问题

2. 限界函数

如果从E开始到叶结点L的路上，每个作业pk在机器1上处理完都能立即在机器2上开始处理，则对于L有：

$$\sum_{i\notin M} F_{2i} = \sum_{k=r+1}^{n}[F_{1Pr}+(n-k+1)t_{1Pk+}\ t_{2Pk}] = S1$$

如果不能做到这点，S1只会增加，从而有$\sum_{i\notin M} F_{2i} \geq S1$。

如果从E到L的路上，从作业p_{r+1}开始机器2没有空闲时间，则有：

$$\sum_{i\notin M} F_{2i} =$$

$$\sum_{k=r+1}^{n}[\max(F_{2Pr},\ F_{1Pr}+\min_{i\notin m} t_{1i})+(n-k+1)t_{2Pk}] = S2$$

从而有S2也是$\sum_{i\notin M} F_{2i}$的下届。

6.9 批处理作业问题

2. 限界函数

在结点E处相应子树中叶结点完成时间和的下界是：

$$f \geq \sum_{i \in M} F_{2i} + \max\{S_1, S_2\}$$

注意到如果选择Pk，使t_{1pk}在k>=r+1时依非减序排列，S1则取得极小值$\hat{S1}$。同理如果选择Pk使t_{2pk}依非减序排列，则S2取得极小值$\hat{S2}$。

$$f \geq \sum_{i \in M} F_{2i} + \max\{\hat{S}_1, \hat{S}_2\}$$

这可以作为优先队列式分支限界法中的限界函数，并用该下界作为结点的优先级。

6.9 批处理作业问题

3. 算法描述

算法的while循环完成对排列树内部结点的有序扩展。在while循环体内算法依次从活结点优先队列中取出具有最小bb值（完成时间和下界）的结点作为当前扩展结点，并加以扩展。

首先考虑已安排作业数enode.s=n的情形，当前扩展结点enode是排列树中的叶结点。enode.sf2是相应于该叶结点的完成时间和。当enode.sf2<bestc时更新当前最优值bestc和相应的当前最优解bestx。

当enode.s<n时，算法依次产生当前扩展结点enode的所有儿子结点。对于当前扩展结点的每一个儿子结点node，计算出其相应的完成时间和的下界bb。当bb< bestc时，将该儿子结点插入到活结点优先队列中。而当bb≥ bestc时，可将结点node舍去。

总结

回溯法

- 深度优先
- 约束函数和限界函数剪枝

分支限界法

- 队列式：广度优先
- 优先队列式：优先级
- 约束函数和限界函数剪枝

许多典型问题可以用两种不同的算法策略求解，对比体会算法的精髓和各自的优点。