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。

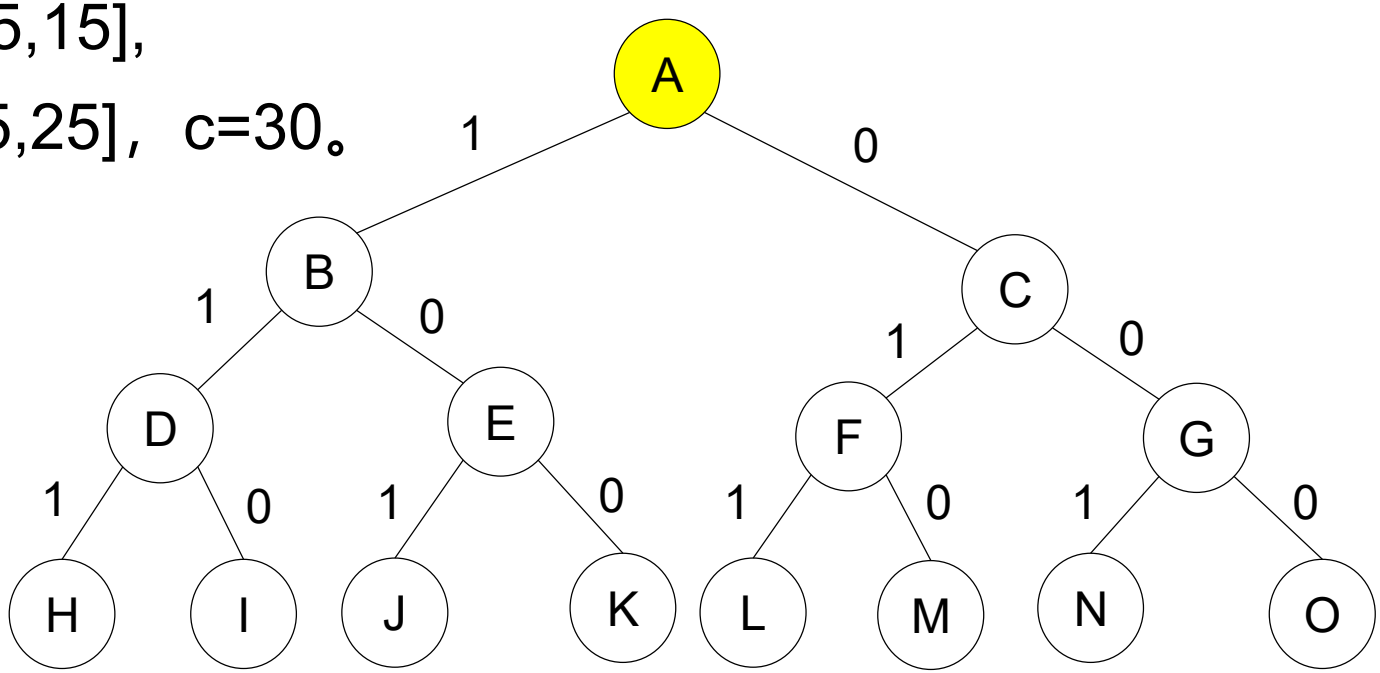
用堆实现优先队列，堆顶结点为优先级最高的结点。

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

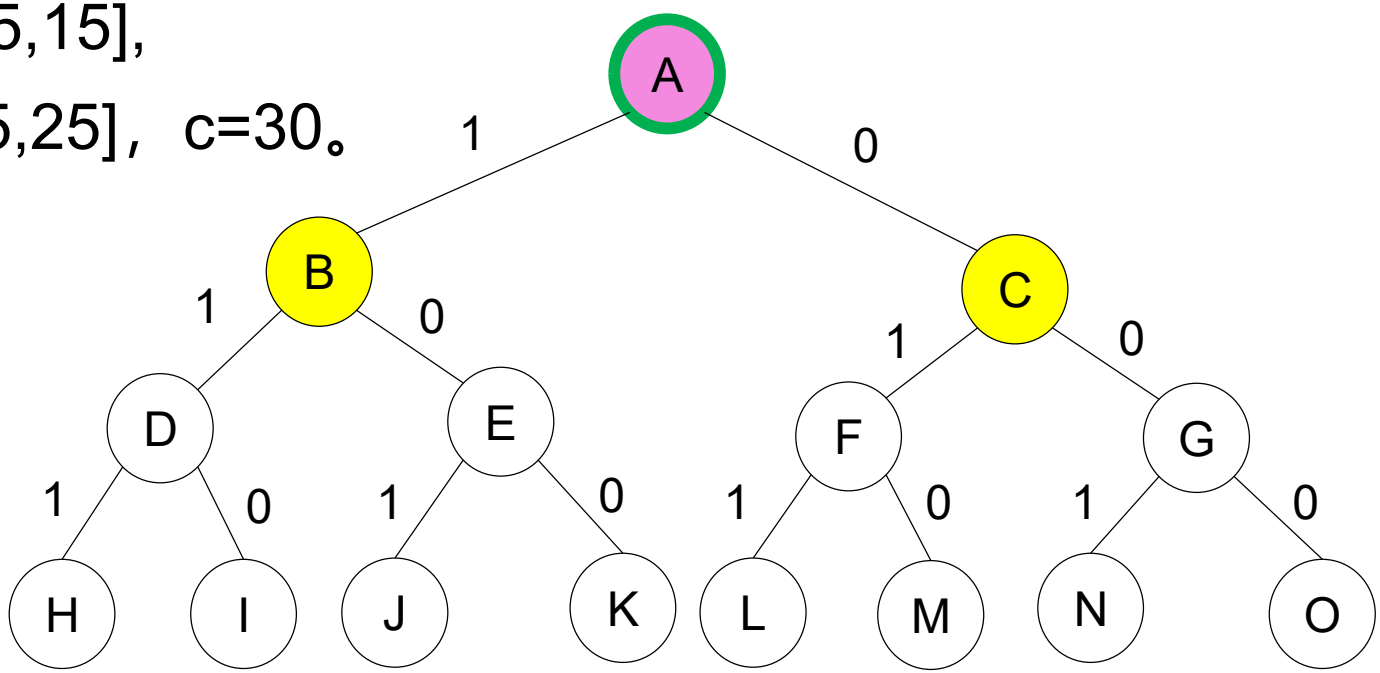
- 从A开始，初始时活结点队列为空，A是当前扩展结点；

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。



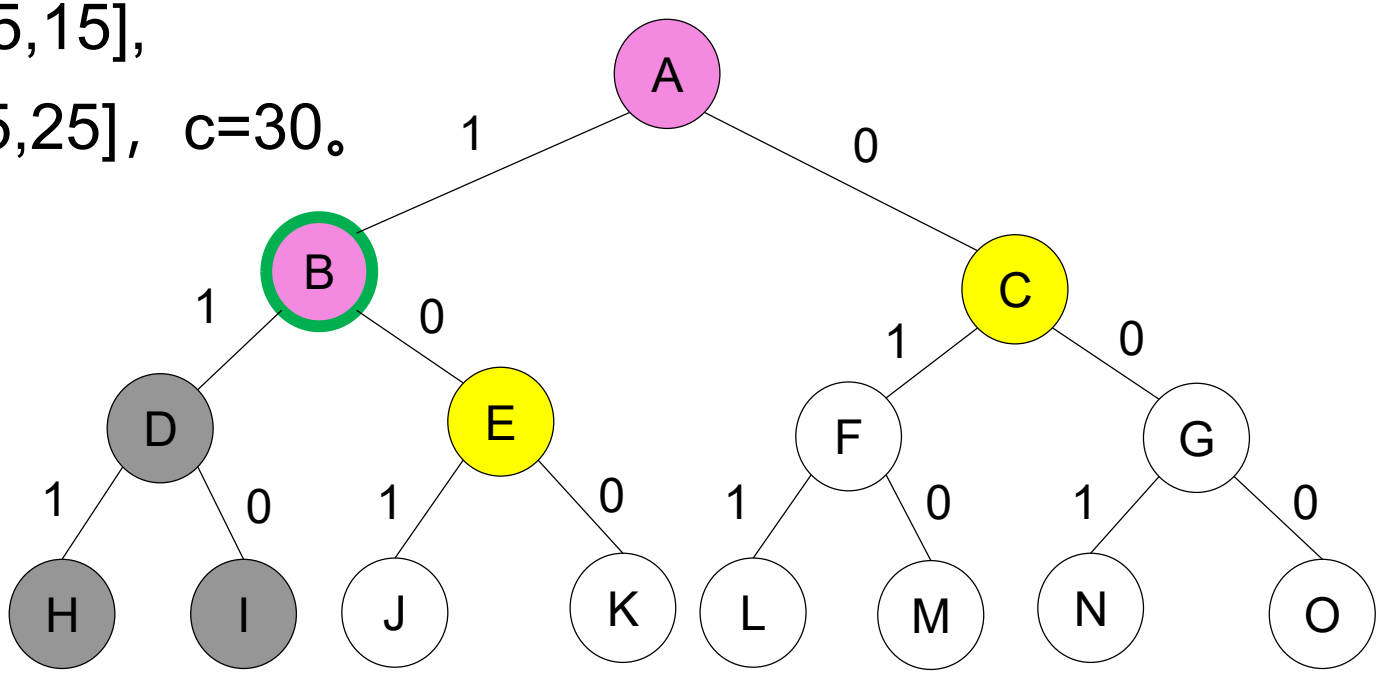
- A的2个儿子B和C均为可行结点，故将B和C依次加入队列，并舍弃A；(BC)

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。



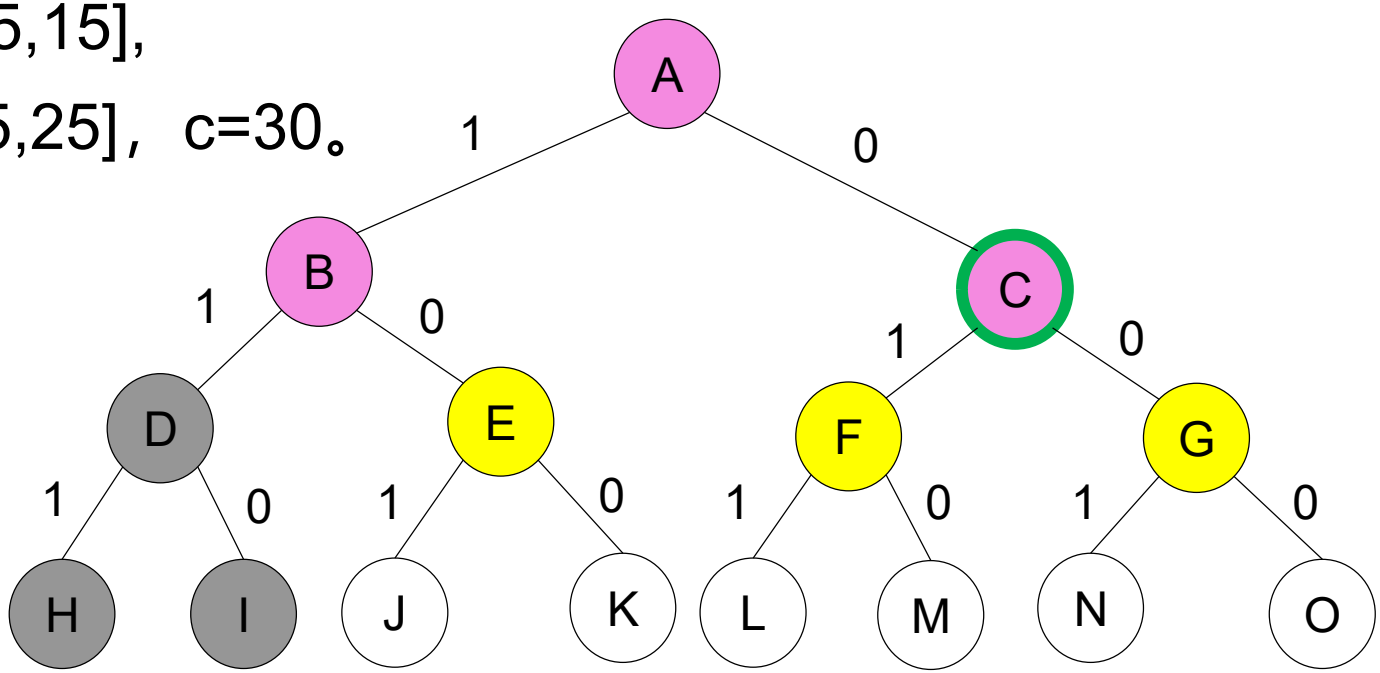
- 扩展B，它的2个儿子结点D和E，D不是可行结点舍去，E是可行结点加入队列；(CE)

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。



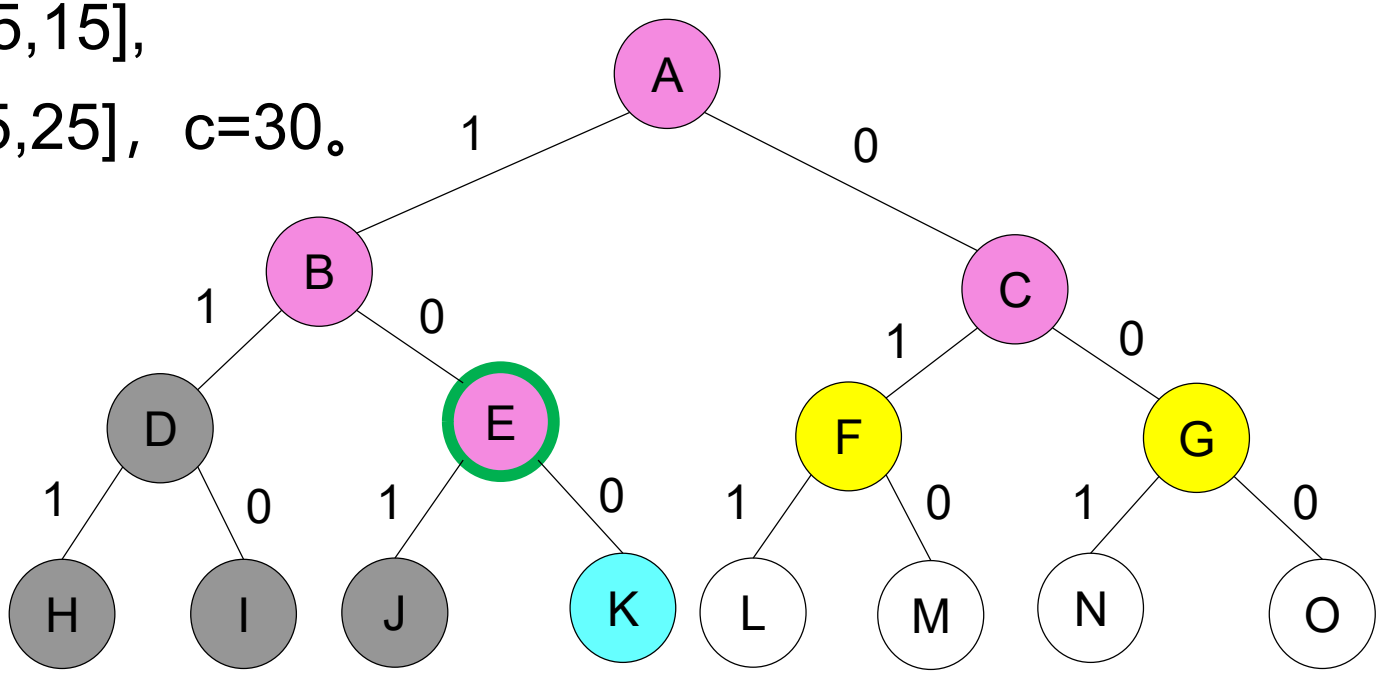
- C成为扩展结点，它的2个儿子结点F和G均为可行结点，依次加入队列；(EFG)

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。



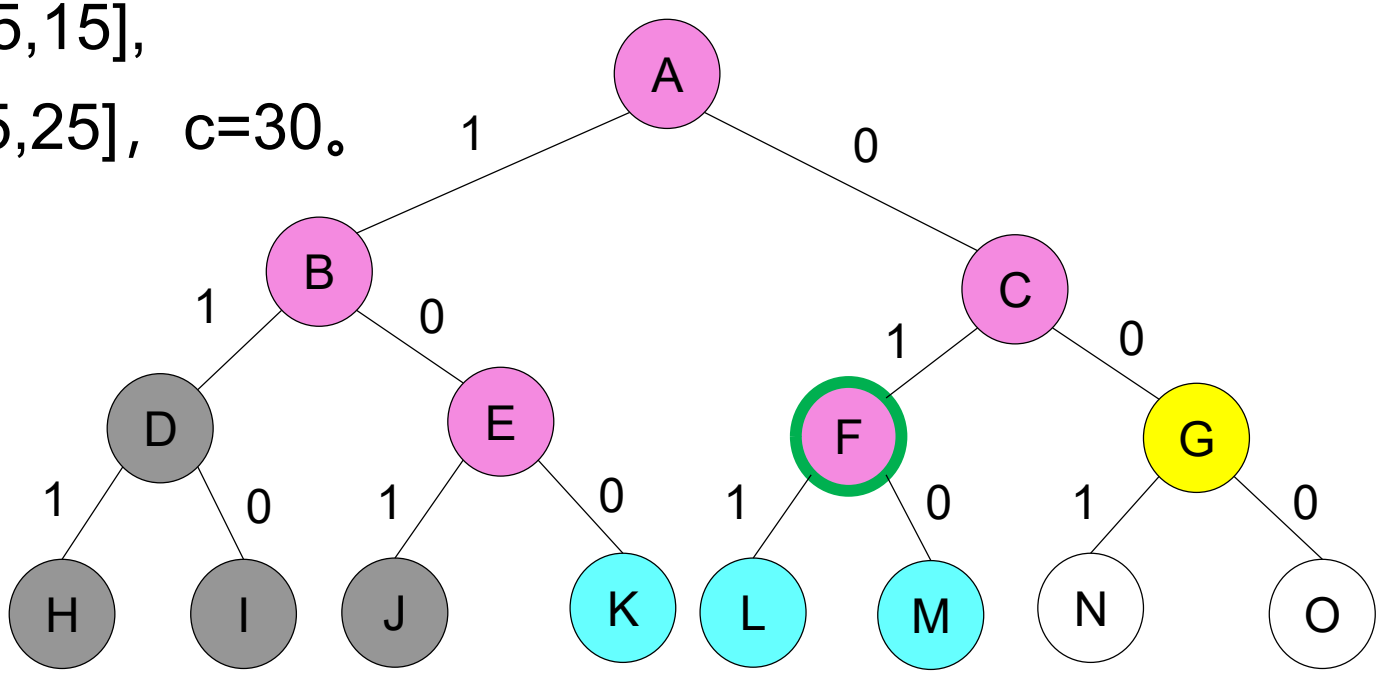
- 扩展E，得到J和K。J不可行，K是可行的叶结点，得到一个可行解，价值为45。(FG)

队列式分支限界法

0-1背包问题，n=3，

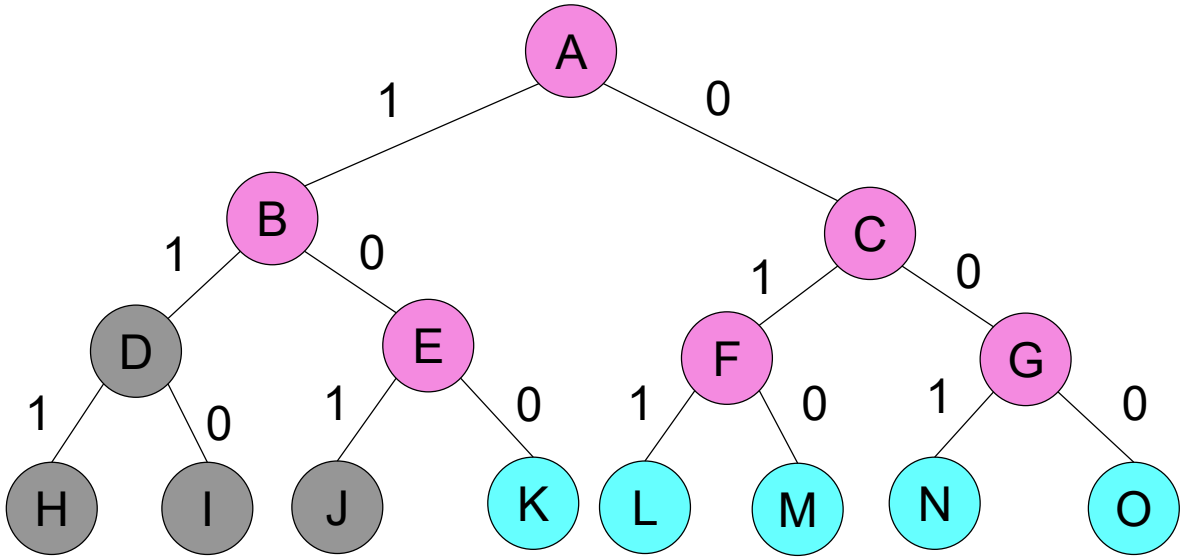
w=[16,15,15],

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。(G)

队列式分支限界法



结点访问顺序：ABCEFG

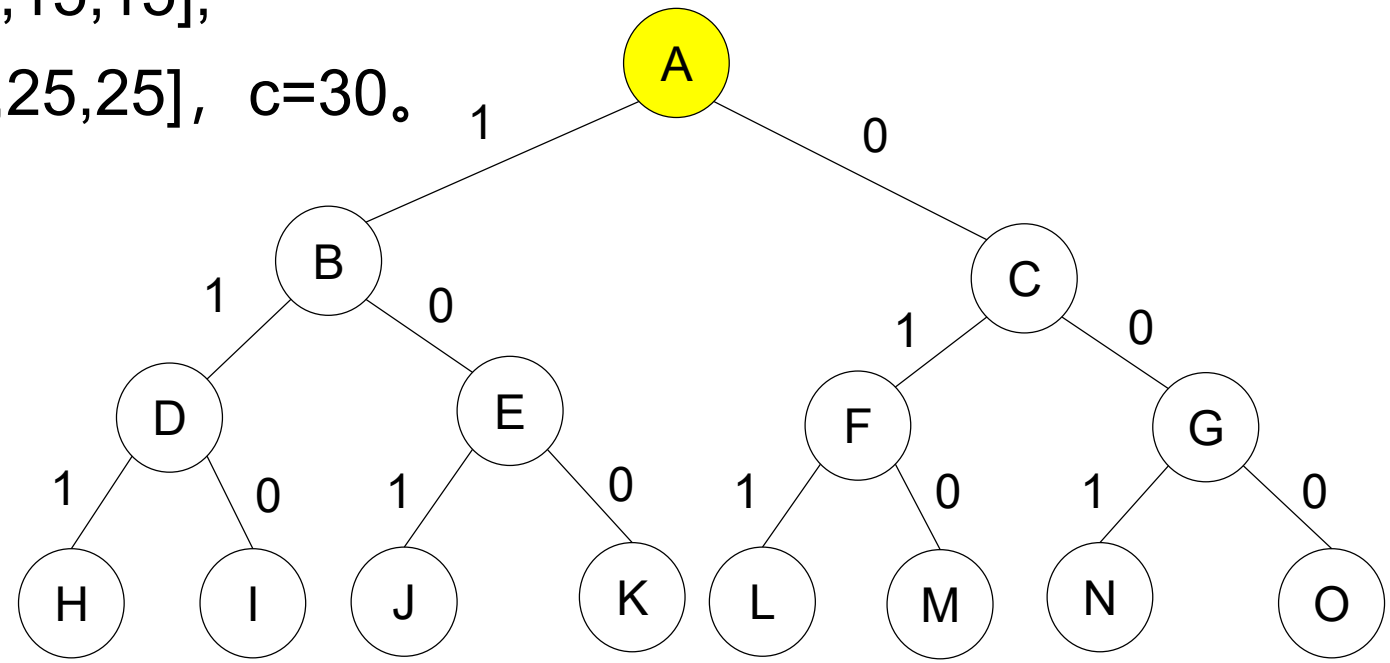
队列式分支限界法搜索空间树的方式与解空间树的广度优先遍历算法极为相似，唯一的不同之处是，队列式分支限界法不搜索以不可行结点为根的子树。

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。



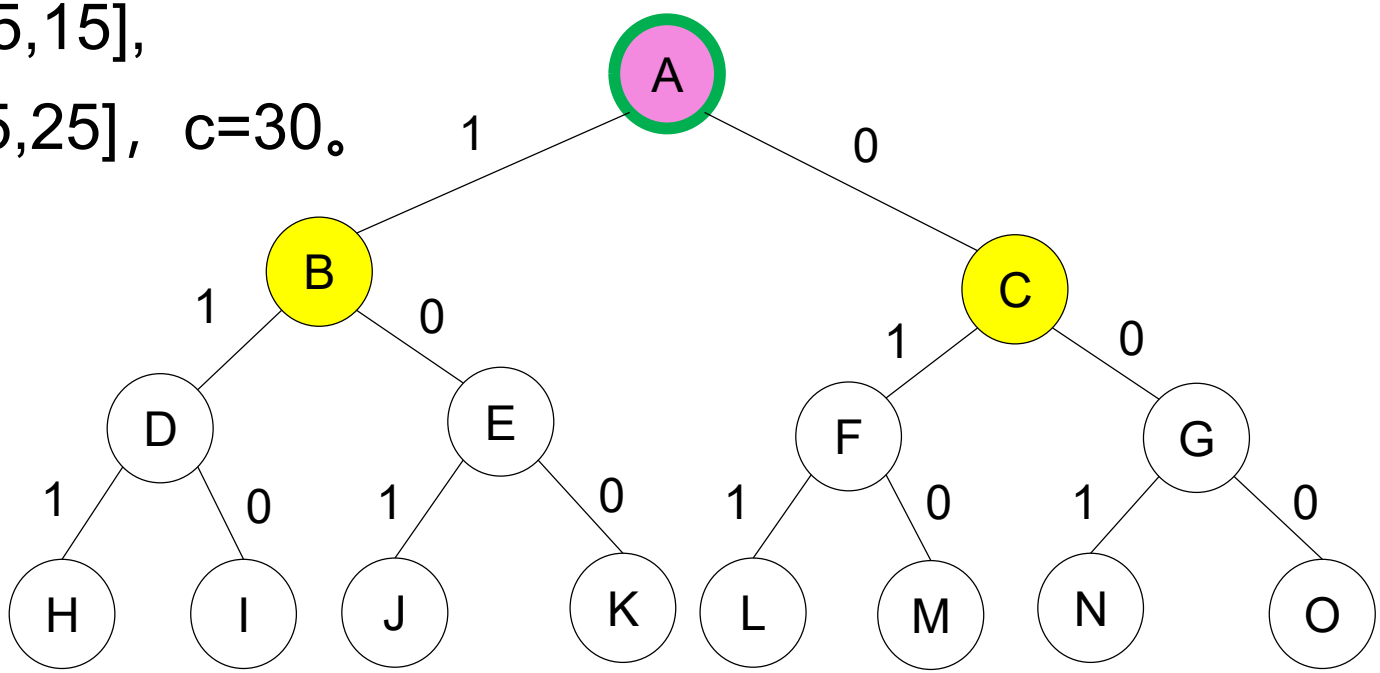
- 从A开始，用一个极大堆表示活结点表的优先队列，优先级是获得的价值。A是当前扩展结点。

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。



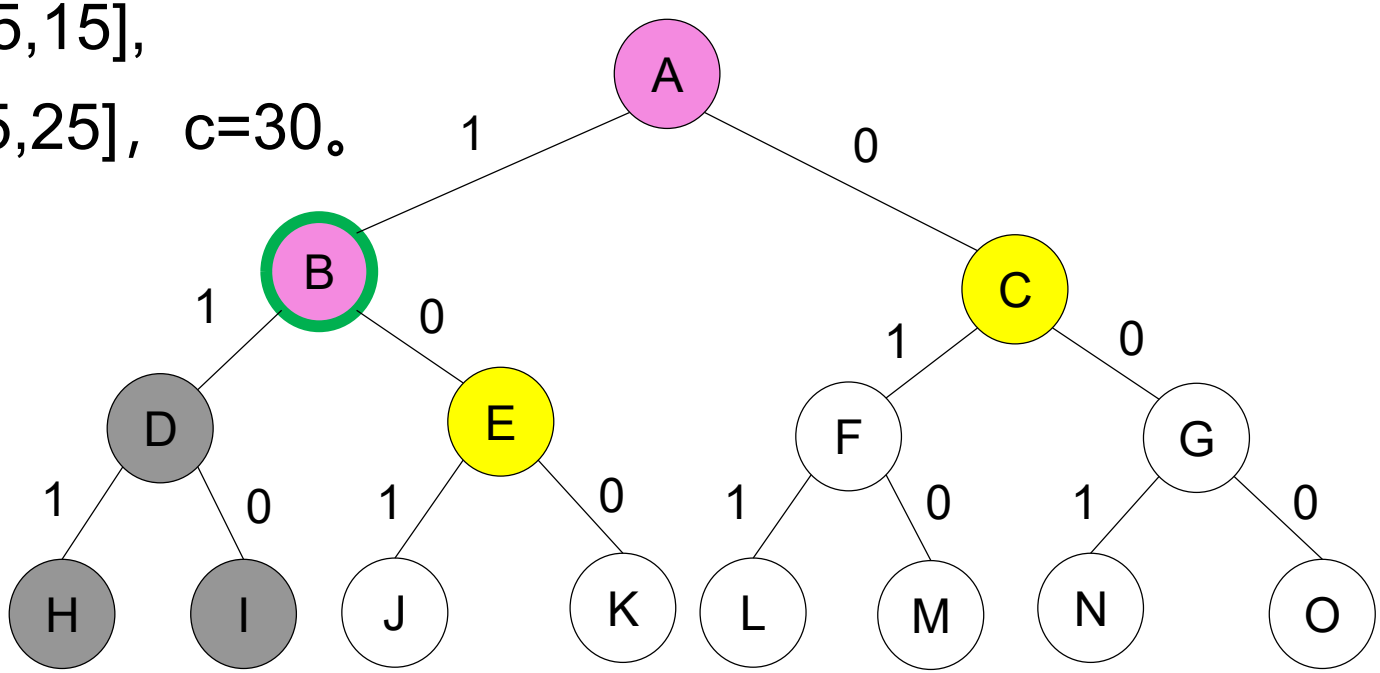
- 扩展A得到B和C，均为可行结点。B获得价值45，C获得价值0，所以B位于堆顶。

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。



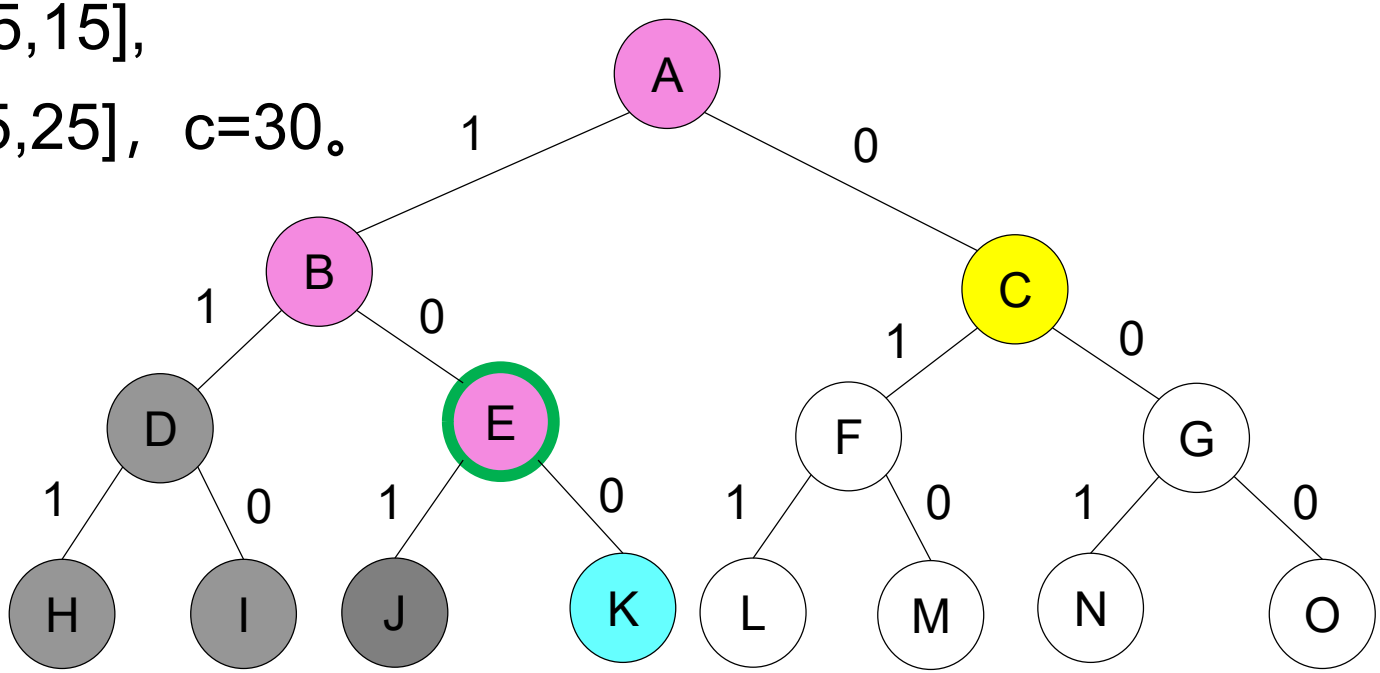
- 扩展B，它的2个儿子结点D和E，D不是可行结点舍去，E是可行结点，获得价值为45，位于堆顶；

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。



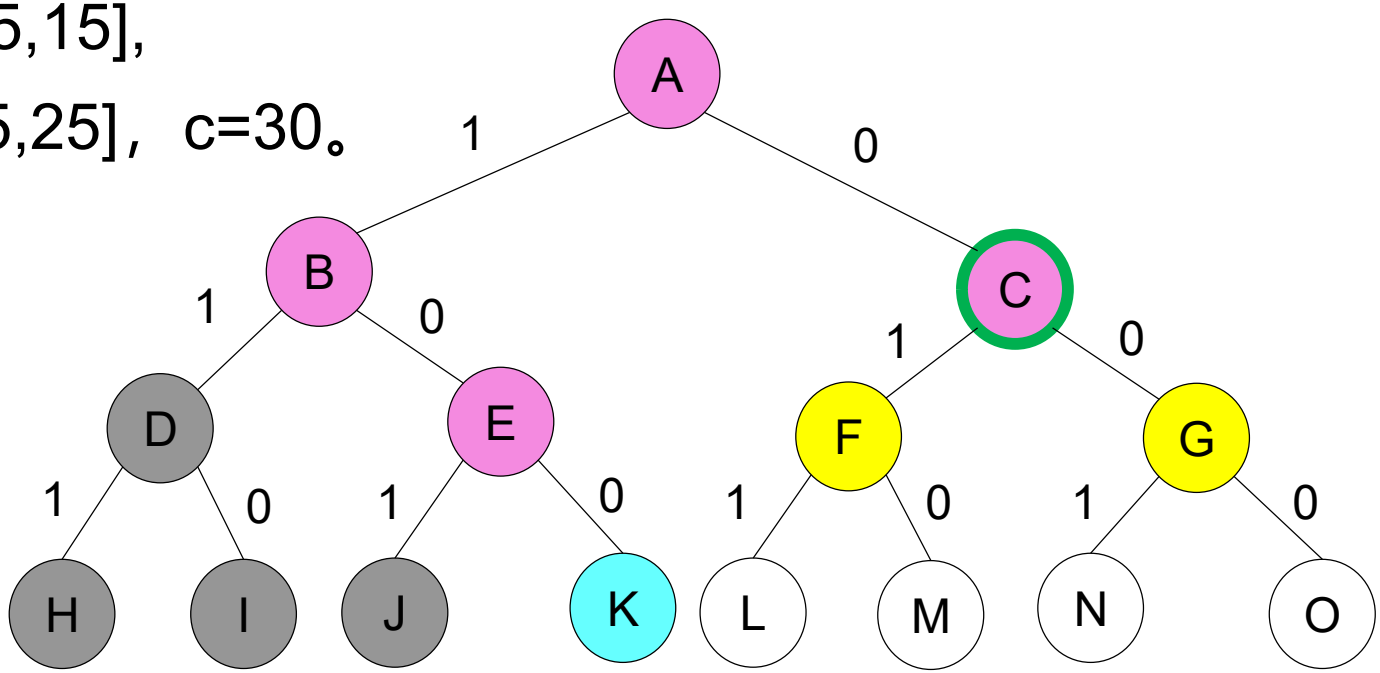
- 扩展E，得到J和K。J不可行，K是可行的叶结点，得到一个可行解，价值为45。

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。



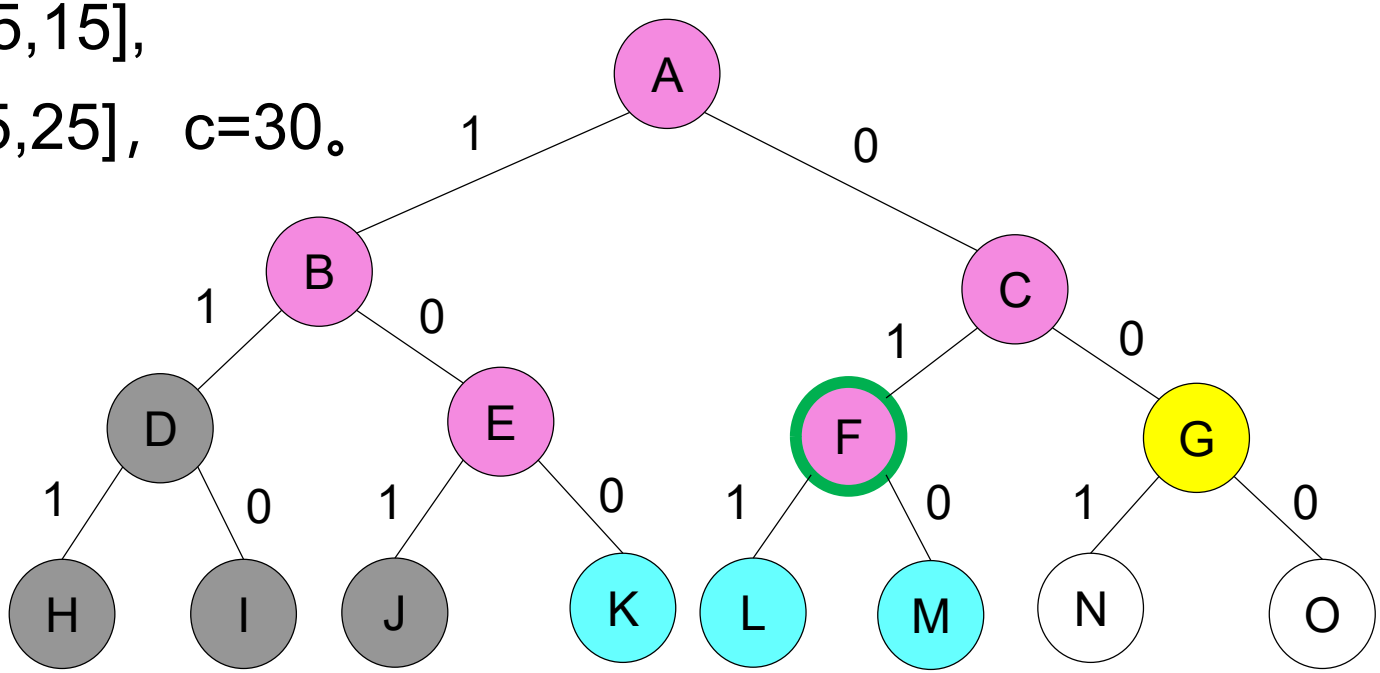
- 扩展堆中唯一元素C，它的2个儿子结点F和G均为可行结点，F获得价值25，G获得价值0，F位于堆顶；

优先队列式分支限界法

0-1背包问题，n=3，

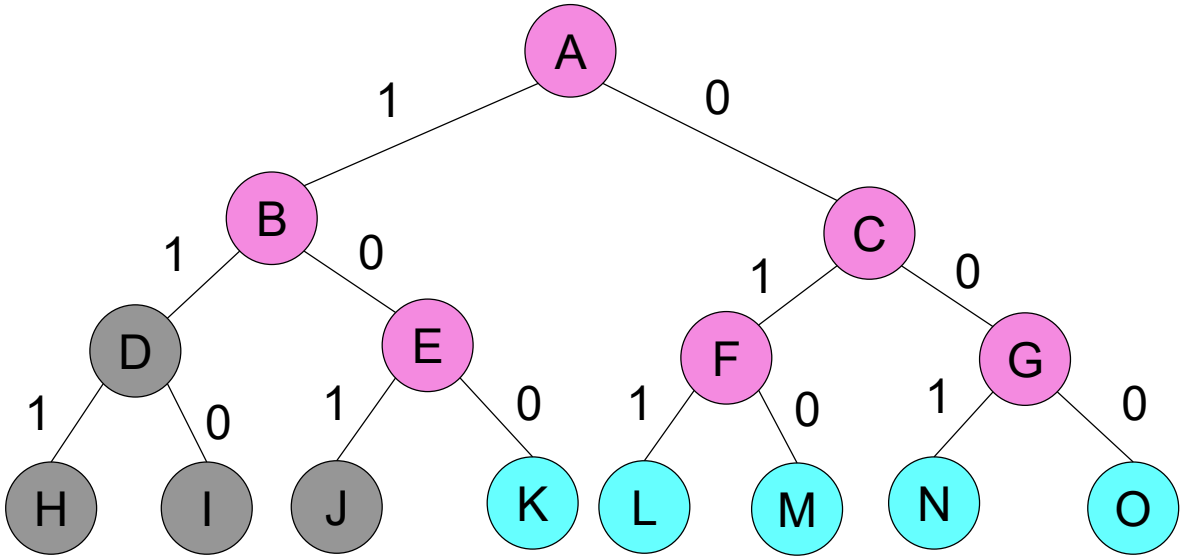
w=[16,15,15],

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

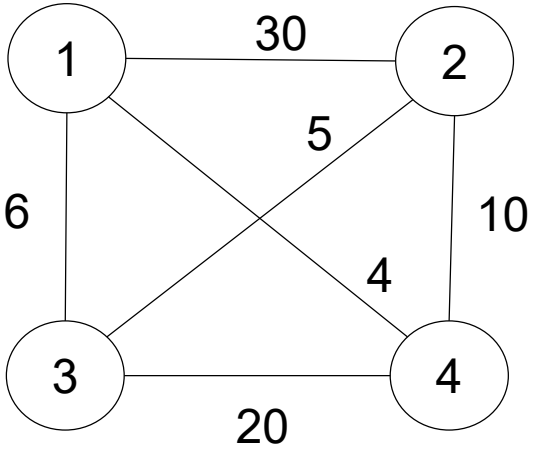
队列式分支限界法 结点访问顺序：ABCEFG

分支限界法

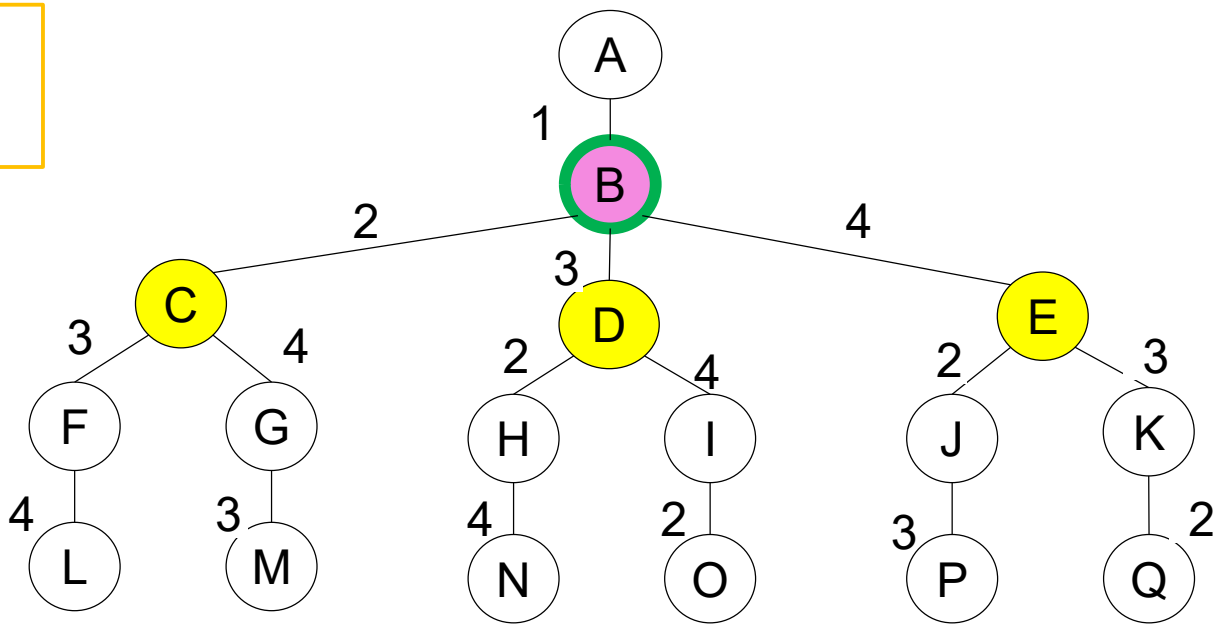
- 分支限界法与回溯法类似，也可用剪枝函数加速搜索；
- 剪枝函数给出每一个可行结点可能获得的最大价值的上界，如果该上界比当前最优值小，则剪去相应的子树；
- 将上界值作为优先级，选取当前扩展结点，有时可以加速找到最优解。

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。



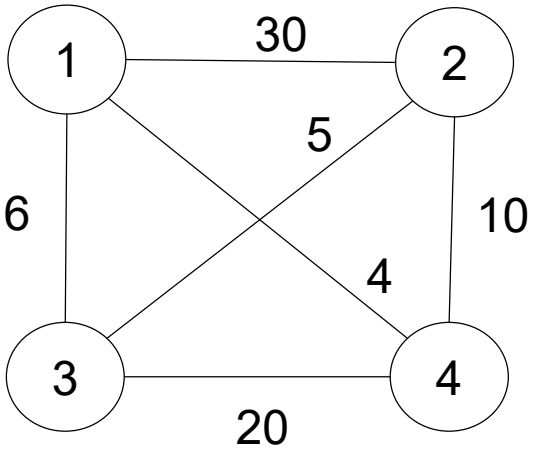
图G



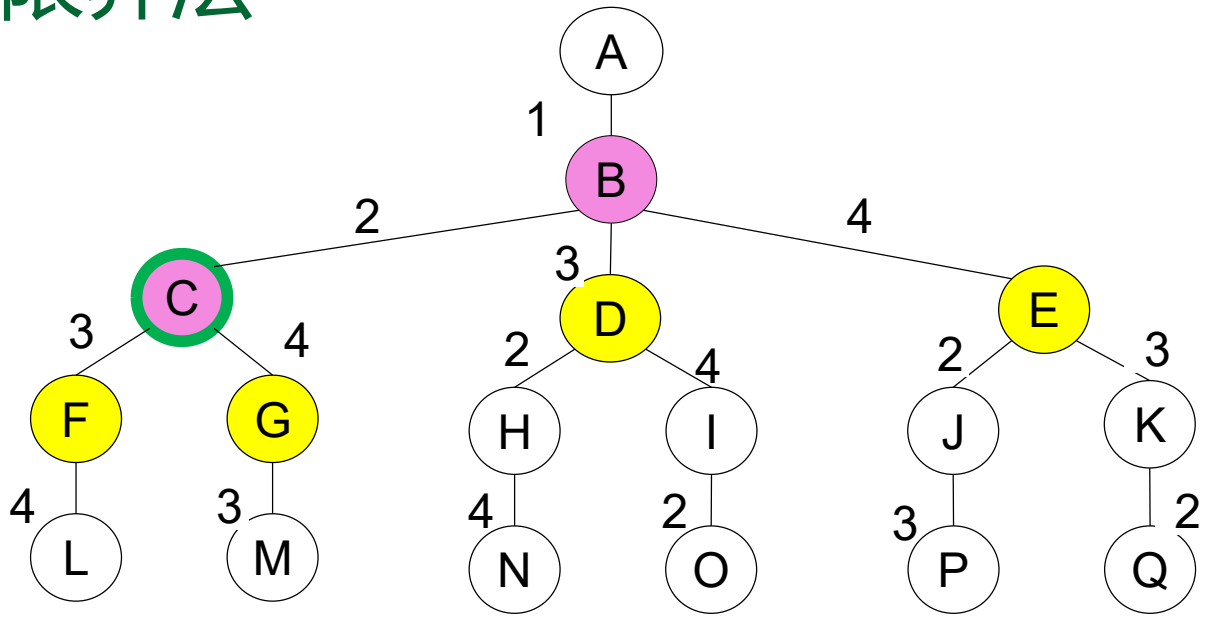
解空间 排列树

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



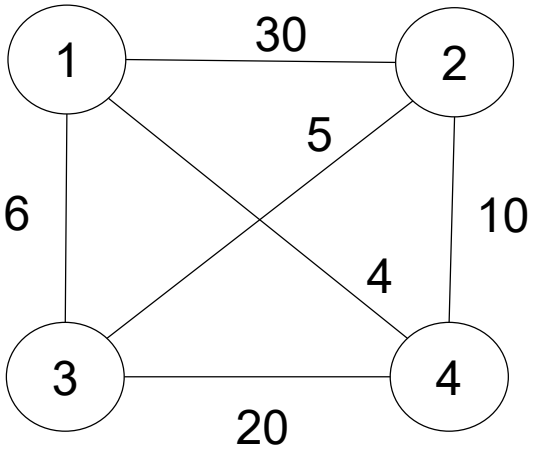
图G



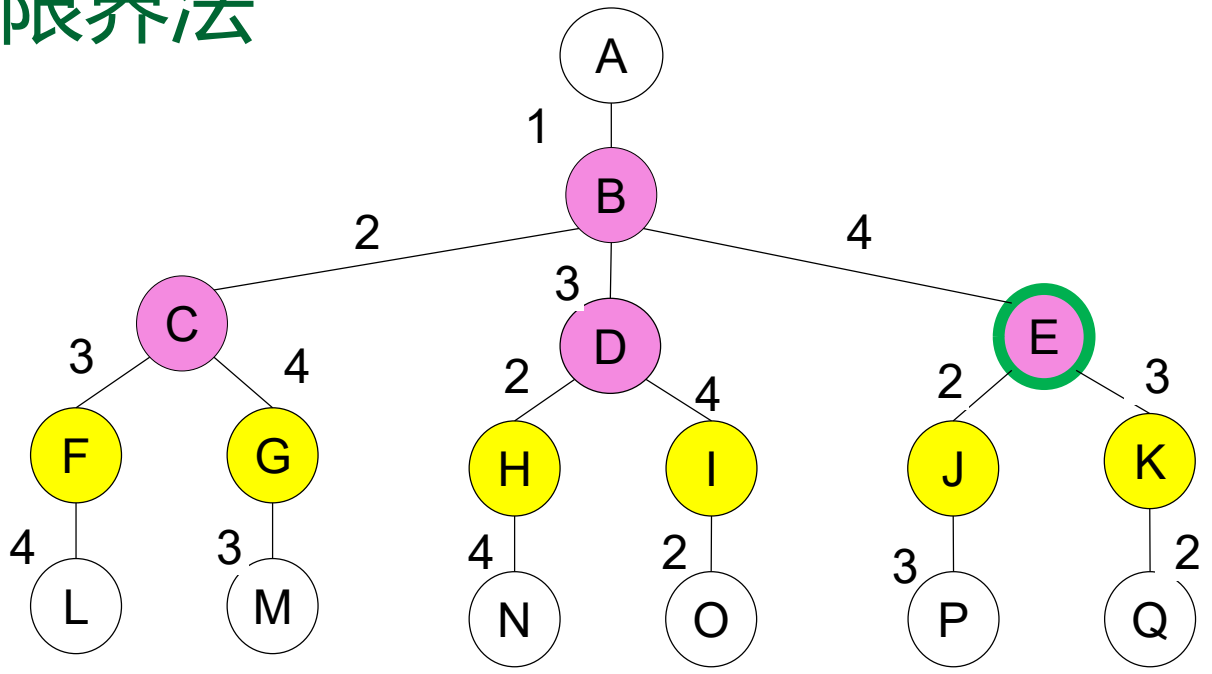
解空间 排列树

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



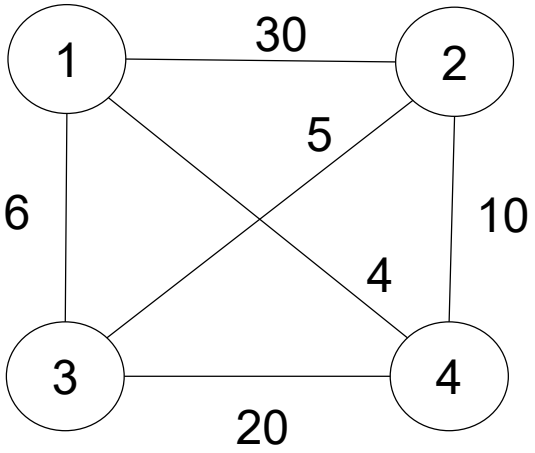
图G



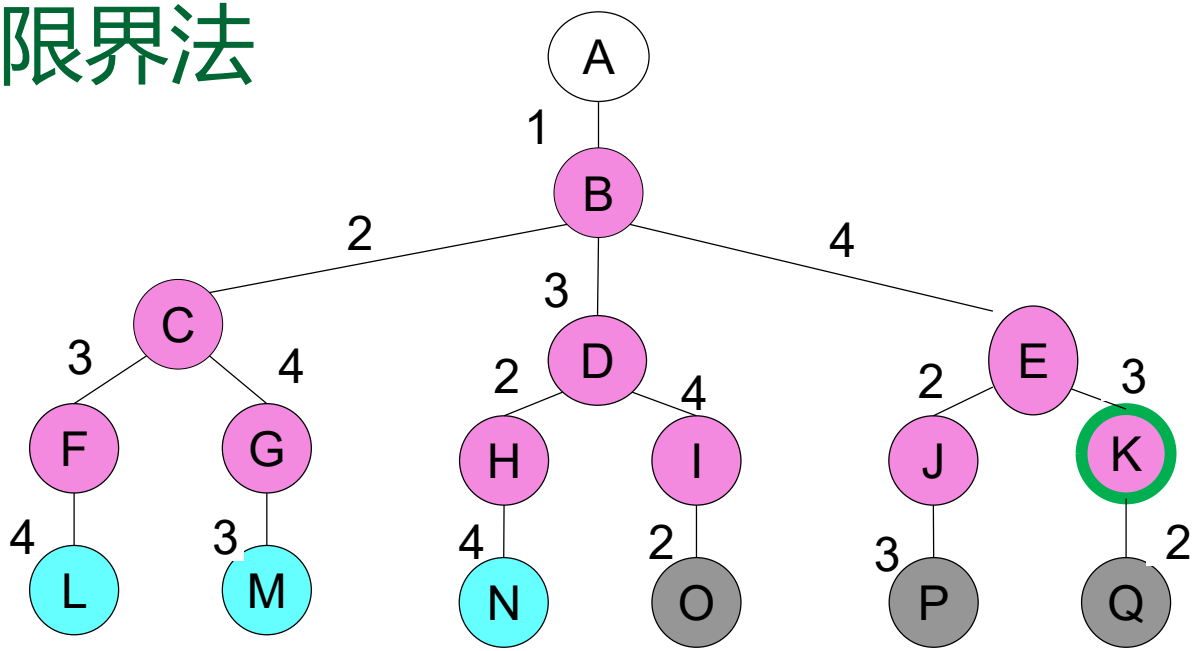
解空间 排列树

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

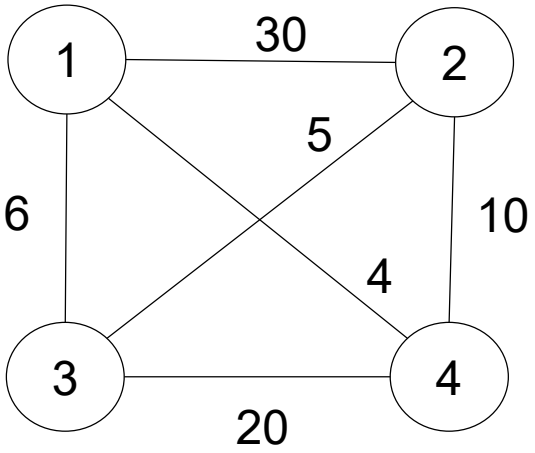
从F扩展至其儿子结点L为叶结点，找到了一条回路，费用为59；

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；

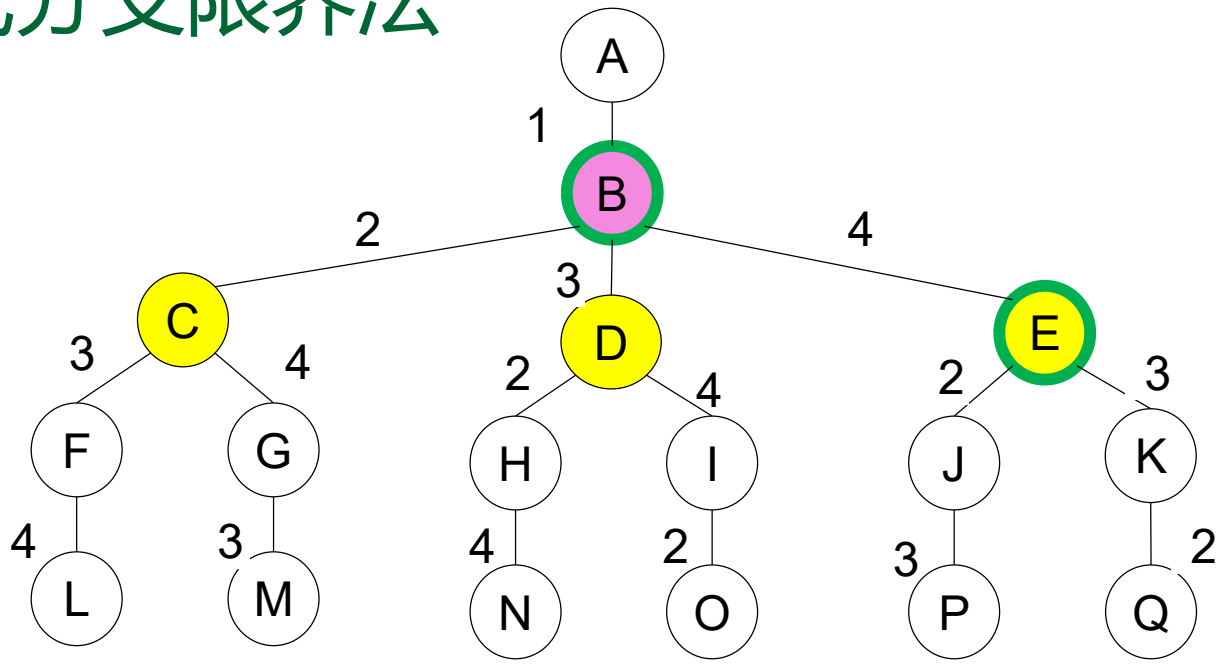
从H扩展至其儿子结点N为叶结点，找到了一条回路，费用为25；

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



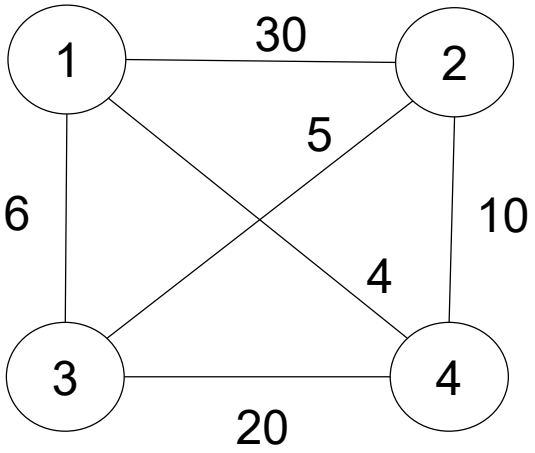
解空间 排列树

用极小堆存储活结点表，优先级是结点的当前费用。

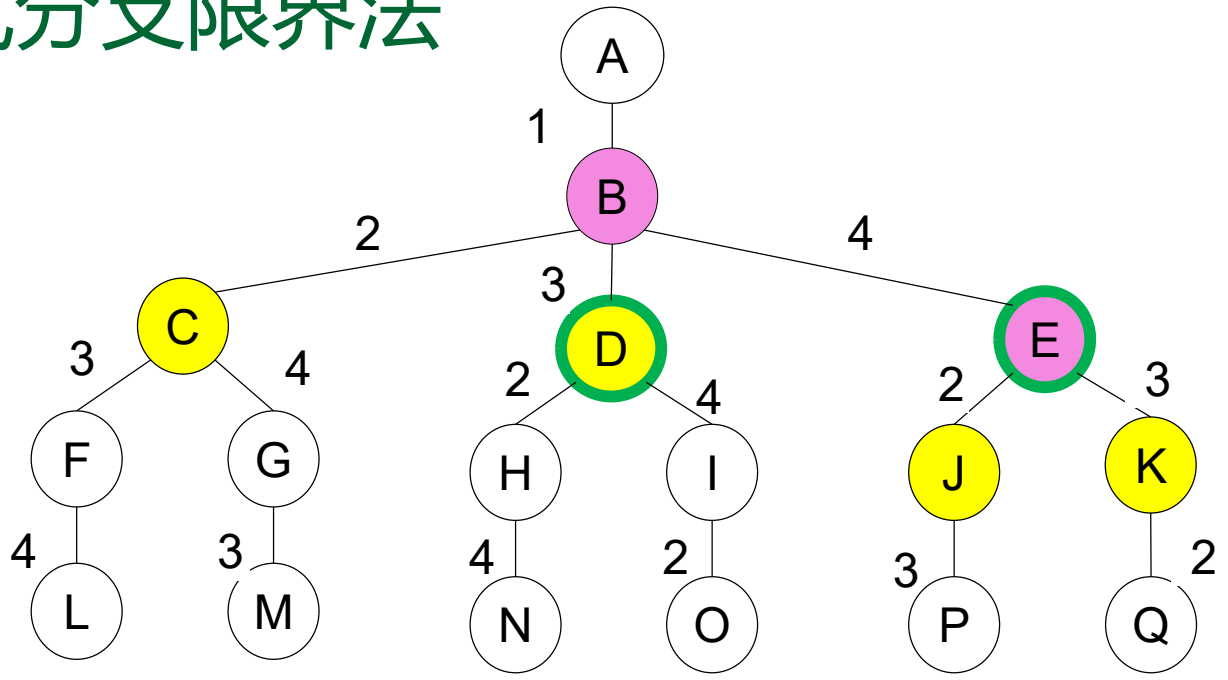
从根节点B开始，初始优先队列为空。

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。

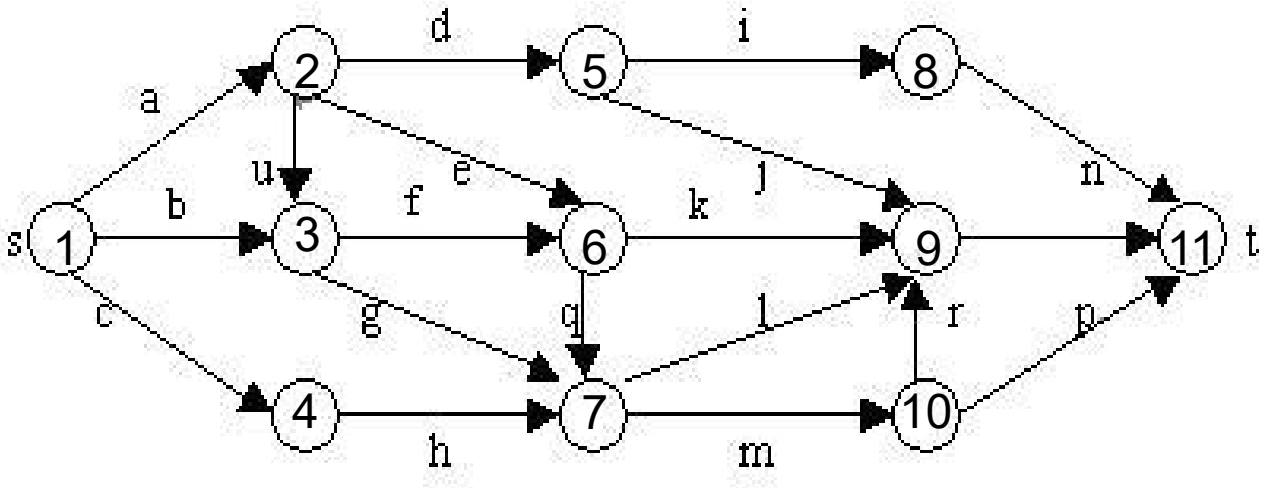
优先队列式分支限界法

- 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。

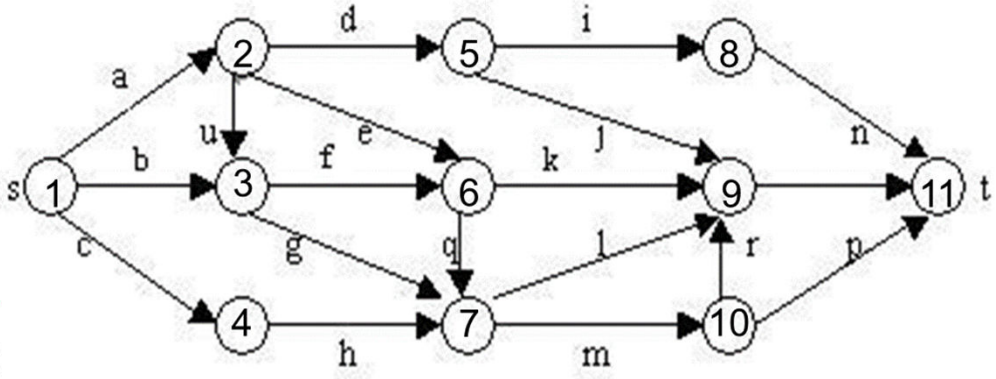
- 算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。
- 此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。
  - 如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j所相应路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。
- 这个结点的扩展过程一直继续到活结点优先队列为空时为止。

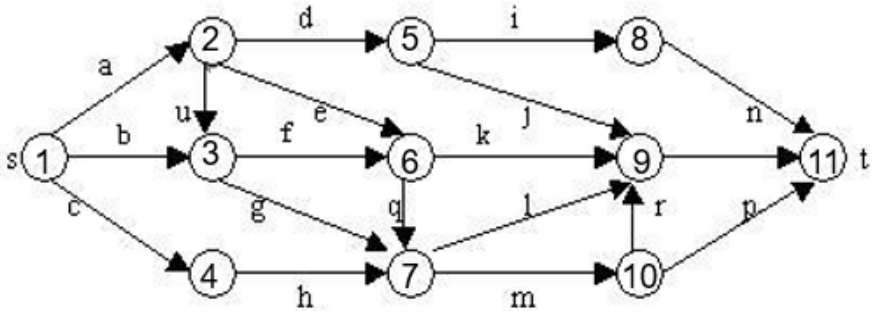
6.2 单源最短路径问题

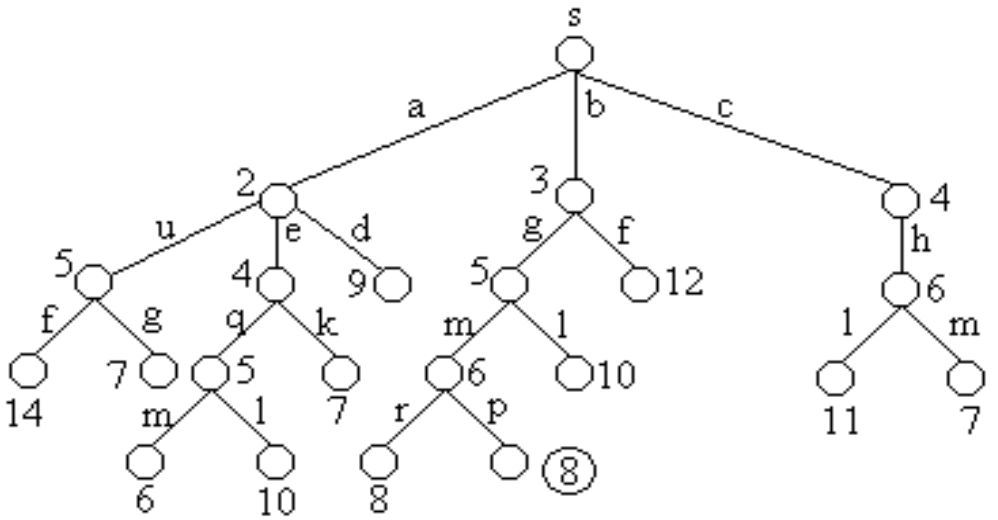
3. 剪枝策略

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;
- 依次方式继续搜索…

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; //节点编号
        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(); // 取下一拓展结点
}
```

6.3 装载问题

1. 问题描述

有一批共个集装箱要装上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。

6.3 装载问题

2. 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节点尾部标志-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

2. 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

2. 队列式分支限界法

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

```
int maxLoading(int w[], int c){
      Queue Q;  Q.add(-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) queue.remove()).intValue();  //
   取下一扩展结点
            if (ew == -1)
                  if (queue.isEmpty())
                        return bestw;
                  queue.put(new Integer(-1));      // 同层结点尾
   部标志
                  ew = queue.remove(); // 取下一扩展结点
            i++;   // 进入下一层
}
```

6.3 装载问题

3. 算法的改进

- 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

3. 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);
        // 取下一扩展结点
        ew=queue.remove();
```

6.3 装载问题

4. 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
 }
```

6.3 装载问题

找到最优值后，可以根据parent回溯到根节点，找到最优解。

```
// 构造当前最优解
for (int j = n; j > 0; j--) {
        bestx[j] = (e.leftChild) ? 1 : 0;
        e = e.parent;
}
```