算法设计与分析

第6章　分支限界法

谢晓芹
哈尔滨工程大学计算机科学与技术学院

学习要点

- 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
  - 队列式(FIFO)分支限界法
  - 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
  - 0-1背包问题
  - 旅行售货员问题
  - 装载问题
  - 单源最短路径问题
  - 批处理作业调度问题

分支限界法的基本思想

- 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(最有利的)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

分支限界法的基本思想

- 分支限界方法找最优解的效率比回溯法高
- 原因在于
  - 采用了最小代价估值函数指导搜索，在活节点表中，选择有最小代价估值的节点作为扩展节点。即总是向最有可能获得最优解的子树上扩展。
  - 并且采用限界函数U杀死活节点表中不可能成为最优解的节点，提高算法的效率。

分支限界法的基本思想

- 算法会为每一节点x计算一个界,任何可能在以x为根的子树中生成的结点所给出的解,其值都不可能超出这个界
- 两个比较值
  - 对于一棵解空间树上的每一个节点所代表的部分解,要计算出通过这个部分解繁衍出的任何解在目标函数上的最佳值边界
  - 目前求得的最佳解的值
- 分支限界法的主要思想
  - 如果边界值不能超越最佳解,这个节点就是一个没有希望的节点.
- 常见的两种分支限界法
  - （1）队列式分支限界法
  - （2）优先队列式（堆式）分支限界法

根据从活结点表中获取下一扩展结点的不同方式

分支限界法的基本思想

- （1）队列式分支限界法
  - 按照队列先进先出（FIFO）或者后进先出（LIFO）原则选取下一个结点为扩展结点
  - 不足:对结点的选择规则相当死板，具有一定的 “盲目”性。这种选择规则不利于快速检索到一个能够到达答案的结点
- （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。常用方法是LC(Least Cost)方法
    - 最大优先队列：使用最大堆，体现最大效益优先
    - 最小优先队列：使用最小堆，体现最小费用优先
  - 对活结点使用一个“有智能”的排序函数来选取下一个结点，往往可以加快获取答案的速度

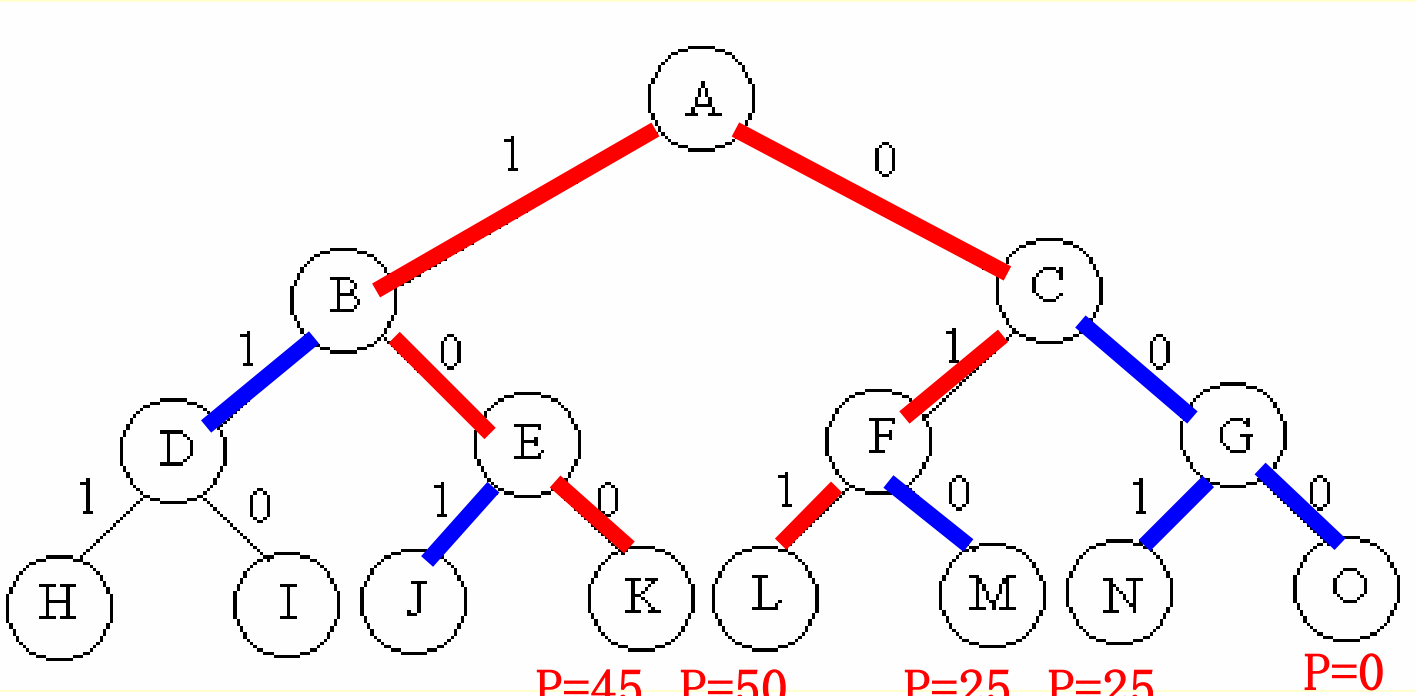
0-1背包问题

- 给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
  - 输入：C>0, w_i>0, v_i >0, 1≤ i≤n
  - 输出：$(x_1, x_2, \ldots, x_n)$, $x_i \in \{0, 1\}$, 满足

$$\max \sum_{i=1}^{n} v_i x_i, \quad \sum_{i=1}^{n} w_i x_i \leq C$$

0-1背包问题-FIFO队列式

- 实例
  - n=3，C=30，w={16，15，15}，p={45，25，25}
  - 可行性约束函数：$\sum_{i=1}^{n} w_i x_i \leq C$
- 分析
  - 步骤1 定义解向量空间
  - 步骤2 解空间结构
    - 子集树



A

B,C

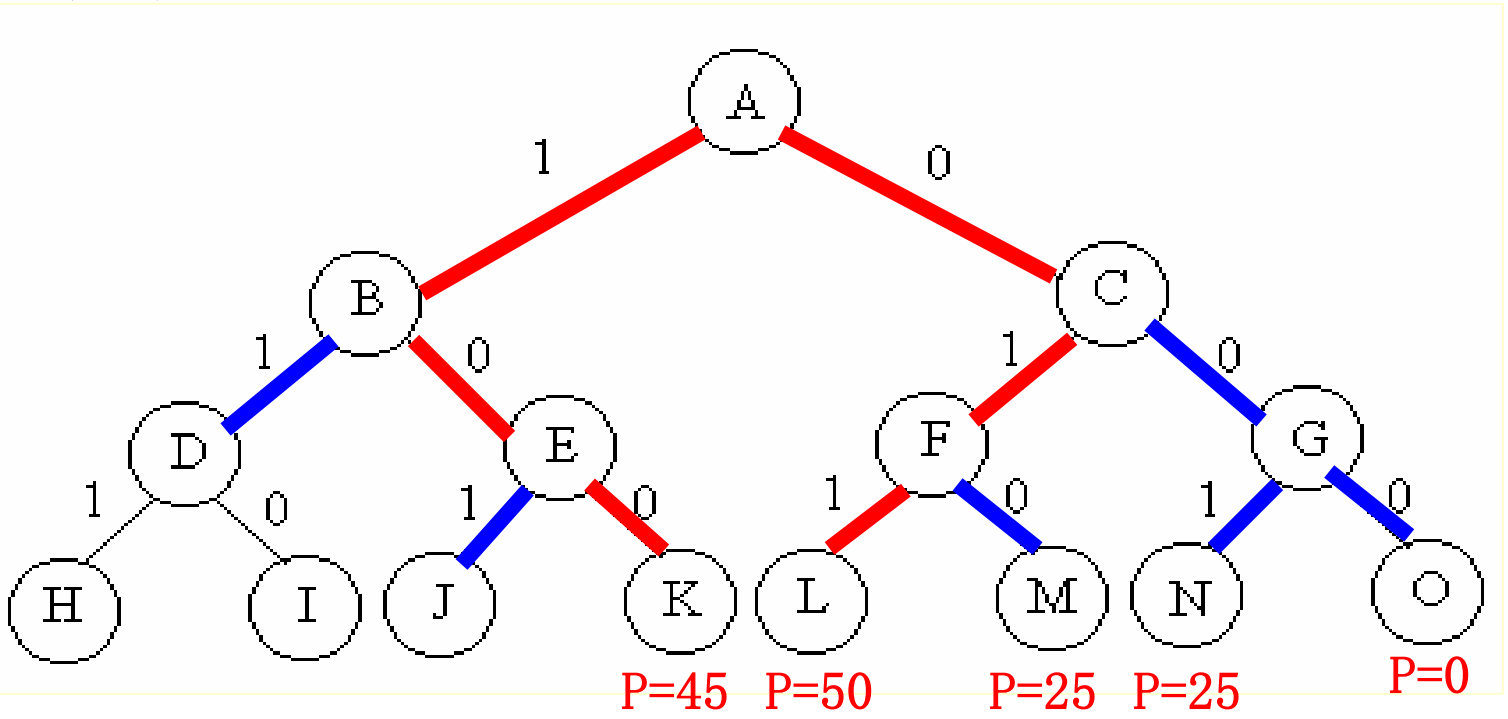
E,F

K,L

0-1背包问题-FIFO队列式

- 实例
  - n=3，C=30，w={16，15，15}，p={45，25，25}
  - 可行性约束函数：$\sum_{i=1}^{n} w_i x_i \leq C$
- 分析
  - 步骤3 广度优先遍历 + 剪枝
    - 叶子结点 i=n+1
    - 非叶子结点：
      - 左子树：可行性约束
      - 右子树：限界约束
    - 入活结点队列的情况
      - 可行解
      - 可能最优解
    - 取结点的方法

➢ FIFO队列式分支限界法的基本思想



A

B,C

E,F

K,L

0-1背包问题-FIFO队列式

- 分支限界搜索过程

```
Queue Q; Q.add(-1);
while (i != n+1) {      // 非叶结点
    // 检查当前扩展结点的左儿子结点
    Typew 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];
      Q.add(cp+p[i], cw+w[i], i+1);}//加入活结点队列

    up = Bound(i+1); // 检查当前扩展结点的右儿子结点
    if (up >= bestp)            // 右子树可能含最优解
        Q.add(cp, cw, i+1);

    //   取下一个扩展节点（略）

}
```

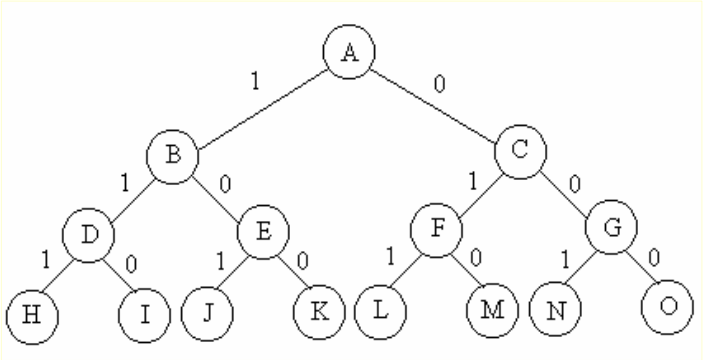
cw:当前装包重量
cp: 当前装包价值

可行性约束

更新bestp

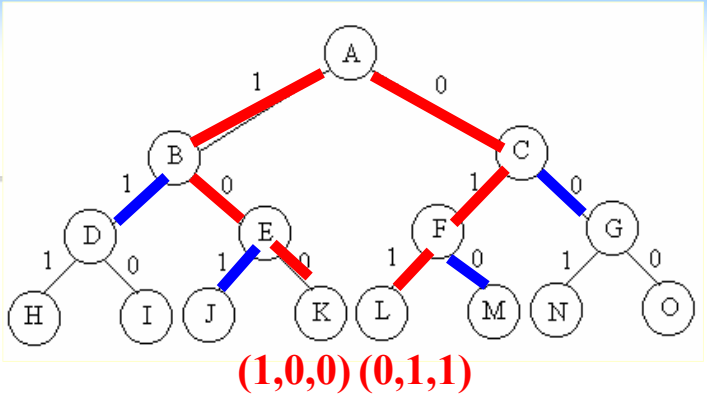
将新的活结点插入到子集树和优先队列中
i+1: 层数

上界约束



0-1背包问题-FIFO队列式

例子： n=3，C=30，w={16，15，15}，p={45，25，25}



活结点队列	扩展结点	处理过程	当前价值cp
空	A	A→B,C	cp=45
CB	B	B→D,E	
EC	C	C→F,G	
FE	E	E→ J,K	
KF	F	F→L,M	
LK	K	K为叶子结点,可行解(1,0,0)	Bestp=45
L	L	L为叶子结点,可行解(0,1,1)	bestp=50
空			

Con(B)=16<30 ✓ bound(C)=50>45 ✓ bestp=45

Constraint(D)=31 X, bound(E)=68.33>45 ✓

Constraint(F)=15<30 ✓, bound(G)=25<45 X

Constraint(J)=31 X, bound(K)=45=45 ✓

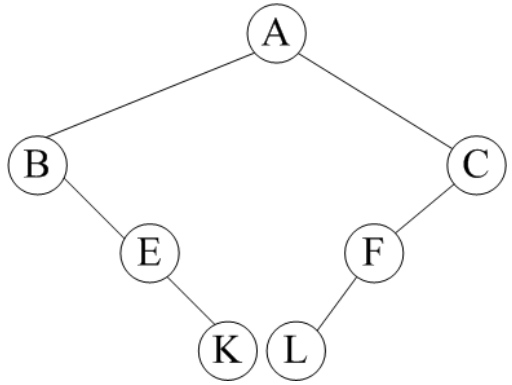
Constraint(L)=30<=30 ✓ bound(M)=25<45 X

Bestp=45

bestp=50

0-1背包问题-FIFO队列式

- 扩展结点生成顺序
  - ABCEFKL
- 实际生成的子集树
- 注意
  - 叶子结点入了活结点表，也入了扩展结点表
  - 剪枝在程序上的实现就是不入活结点表
  - 入活结点表的不同情况
    - 满足约束条件：左孩子
    - 满足限界条件：右孩子
  - 取结点方法
    - FIFO



0-1背包问题-优先队列式

- 优先队列式分支限界法的基本思想
  - 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
  - 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
  - 结点的优先级
    - 价值上界： 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

贪心思想

0-1背包问题-优先队列式

- 实例
  - n=3，C=30，w={16,15,15}，p={45,25,25}

➢ 优先队列式分支限界法的基本思想

提早更新bestP

✓ **使用最大堆**

✓ **上界 up** = 68.3

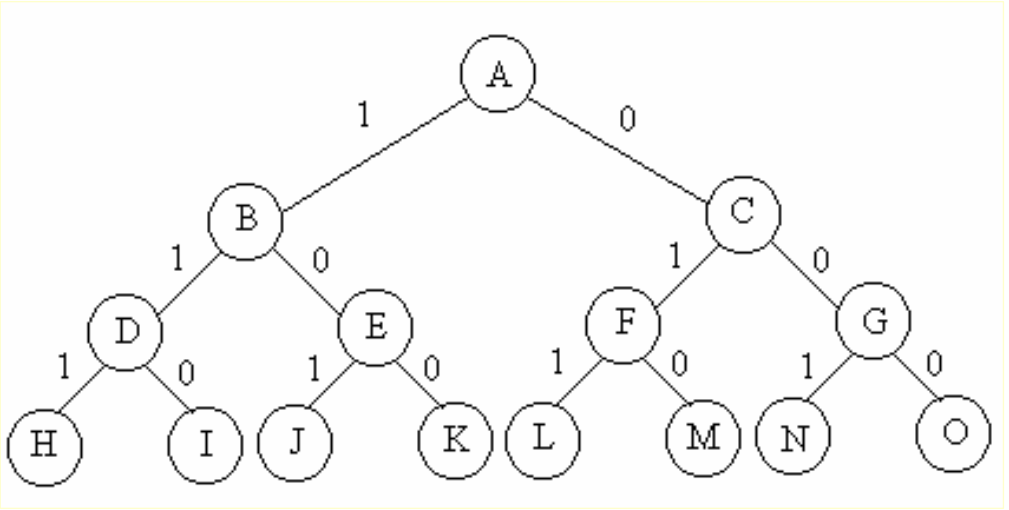
Bound(int i)实现

✓ **下界 L** = 45

贪心算法实现

✓ **bestp**为当前最优值，则

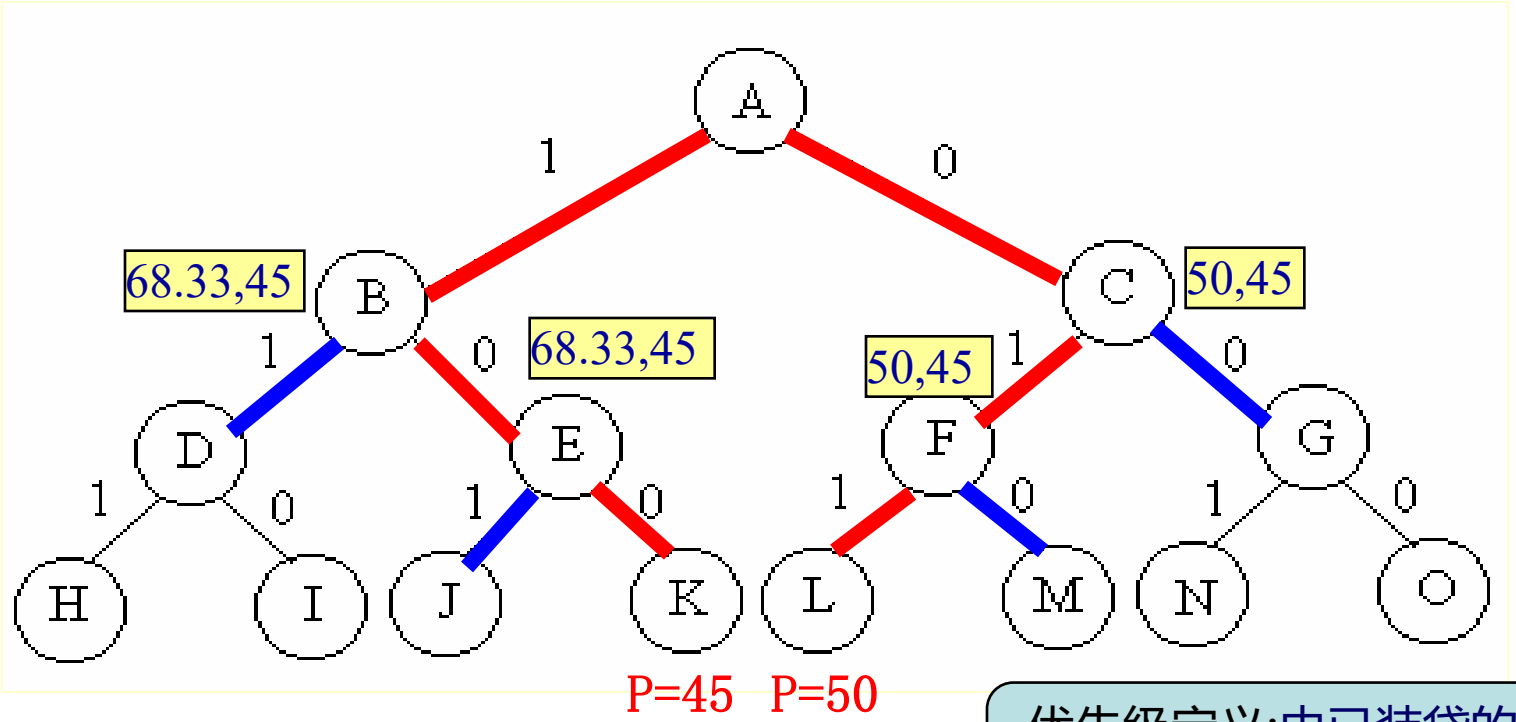
L=45 ≤ bestp ≤ up=68.3



0-1背包问题-优先队列式

- 实例
  - n=3，C=30，w={16,15,15}，p={45,25,25}

➢ 优先队列式分支限界法的基本思想



优先队列

~~A, level=1~~

~~B, level=2~~

~~E, level=3~~

~~F, level=3~~

L, level=4

K, level=4

优先级定义: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

0-1背包问题-优先队列式

- 分支限界搜索过程

cw:当前装包重量
cp: 当前装包价值

```
while (i != n+1) {                  // 非叶结点
    // 检查当前扩展结点的左儿子结点
    Typew 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], true, i+1);}//加入活结点队列

    up = Bound(i+1); // 检查当前扩展结点的右儿子结点
    if (up >= bestp)                // 右子树可能含最优解
       AddLiveNode(up, cp, cw, false, i+1);
    //  取下一个扩展节点（略）
}
```

可行性约束

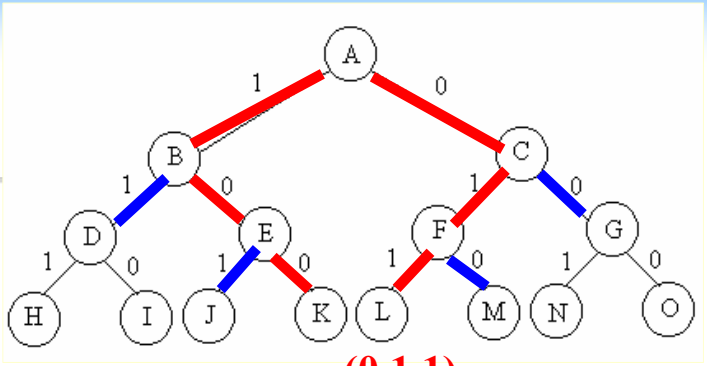
将一个新的活结点插入到子集树和优先队列中
i+1: 层数

true：左孩子

上界约束

false：右孩子

0-1背包问题-优先队列式



例子： n=3，C=30，w={16，15，15}，p={45，25，25}

优先级定义:结点的价值上界（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）

活结点队列	扩展结点	处理过程	当前价值cp
空	A	A→B,C	bestp=45
BC	B	B→D,E	
EC	E	E→ J,K	
CK	C	C→ F,G	
FK	F	F→L,M	Bestp=50
LK	L	L为叶子结点,可行解(0,1,1)	bestp=45

Bestp=0, cp=0

Cons(B)=16<30 ✓ up(B)=68.33 bound(C)=50>45 ✓ up(C)=50

Bestp=45

Cons(D)=31 X, bound(E)=68.33>45 ✓ up(E)=68.33

Cons(J)=31 X, bound(K)=45=45 ✓ up(K)=45

Cons(F)=16<30 ✓ cp+p(2)=25<45, up(F)=50, bound(G)=25>45 X

不更新,Bestp=45

Cons(L)=30<=30 cp+p[3]=50>45, 更新 Bestp=50 ✓

Up(L)=50, bound(M)=25<45 X

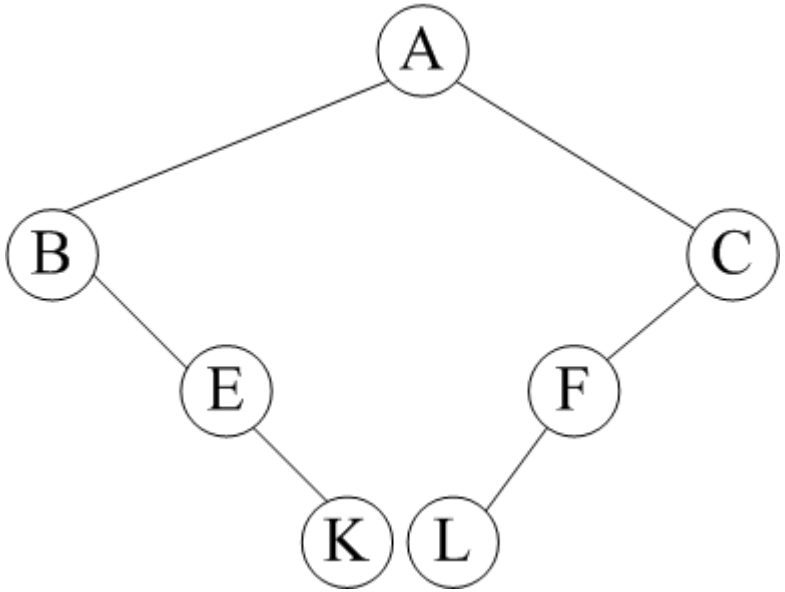
L：i=4为叶子结点，跳出while循环，为解(0,1,1) ,价值50

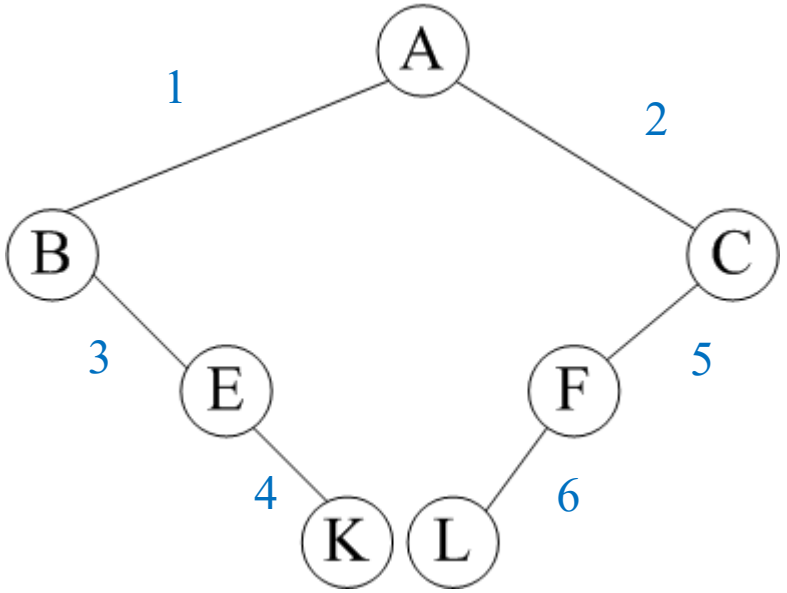
思考：可以采用其他优先级吗？ 如：当前所获得的价值V

0-1背包问题-优先队列式

- 扩展结点生成顺序
  - ABECFL

- 实际生成的子集树





分支限界法与回溯法比较

- 相同点
  - 都是在问题的解空间树上搜索问题解的算法
- 不同点
  - 求解目标不同：
    - 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，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。
  - 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优先级优先的方式搜索解空间树

分支限界法与回溯法比较

不同点	回溯法	分支限界法
求解目标	找出树中满足约束条件的所有解	找出满足约束条件的一个解或找出使目标函数达到极大(小)的最优解
搜索方式	深度优先	广度优先或优先级优先
扩展结点	多次机会成为扩展结点: 扩展结点变为活结点后又可成为扩展结点	每个活结点只有一次机会成为扩展结点
树结点的生成顺序	生成最近一个有希望结点的单个子女	选择其中最有希望的结点,并生成它的所有子女
行进方向	随机性	方向性: 活结点表,搜索朝着解空间树上有最优解的分支推进

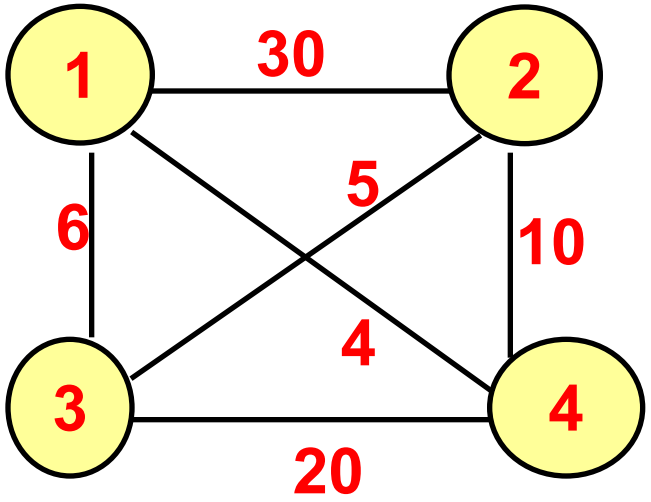
旅行售货员问题

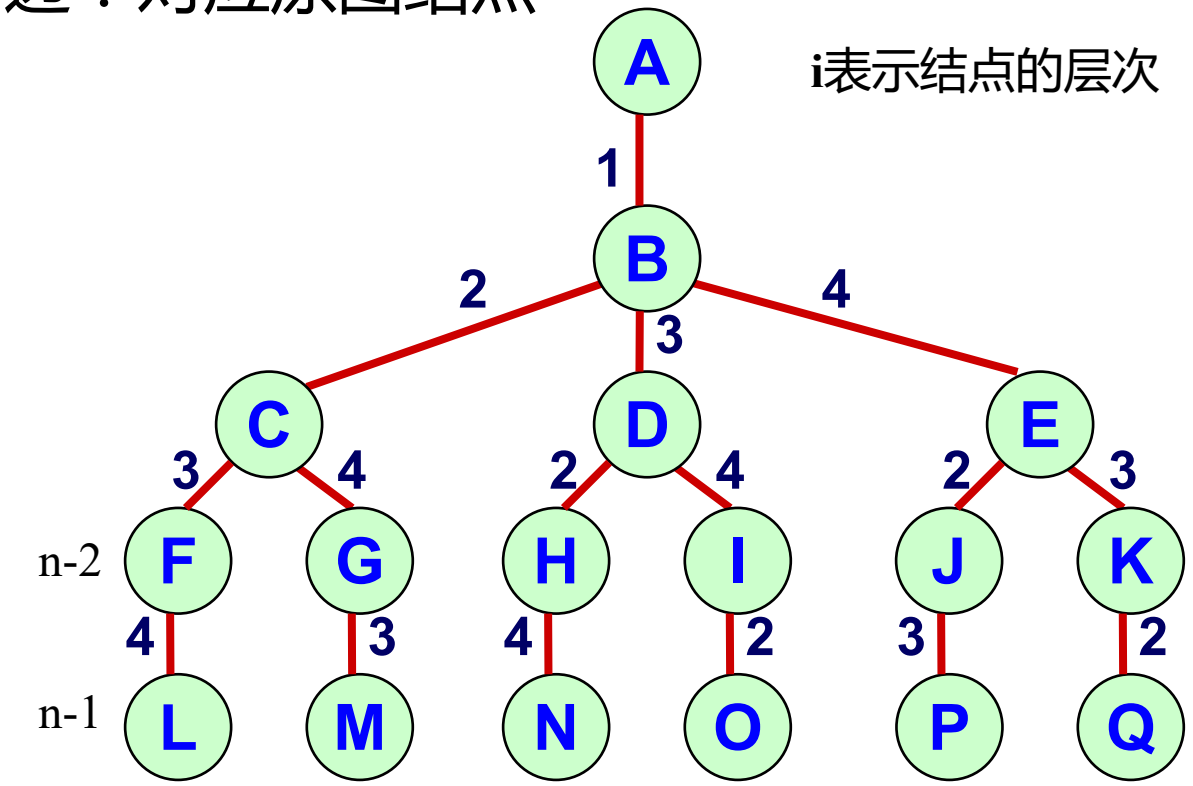
- 问题描述
  - 给定n个顶点的带权图G=(V，E),图中各边的权为正数，图中的一条周游路线是包括V中的每个顶点在内的一条回路，一条周游路线的费用是这条路线上所有边的权之和
  - 旅行商问题(Traveling Salesperson)是要在图G中找出一条有最小费用的周游路线。此问题是NP完全问题
    - 输入：G=(V,E), $c_{ij} > 0,(i,j) \in E$
    - 输出：从图中任一顶点v_i出发的最小耗费的周游路线

旅行售货员问题

- 分析：
  - 解向量空间： (1,2,3,4,1),(1,3,4,2,1),…
  - 解空间树为：排列树

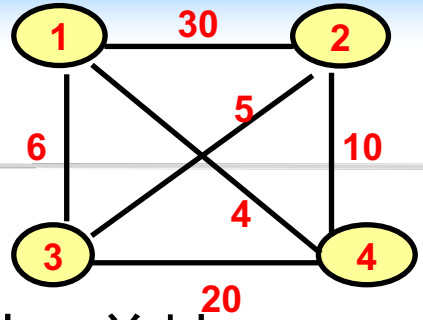
    边：对应原图结点

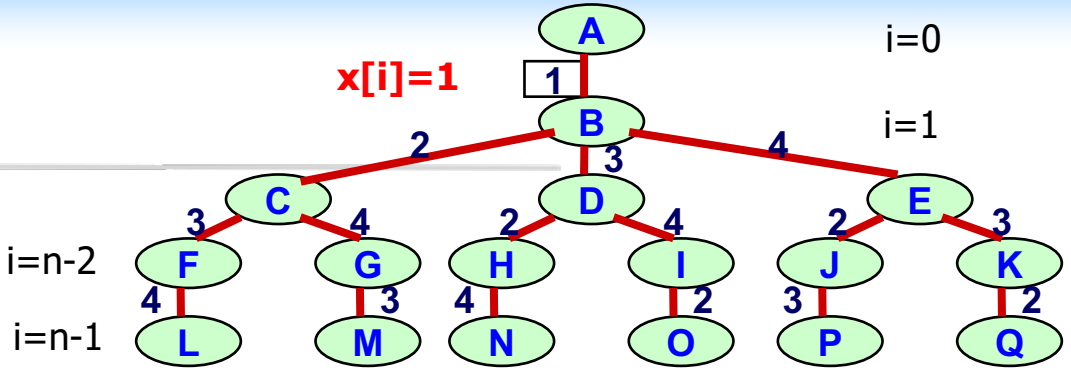




i表示结点的层次

旅行售货员问题





- 分析
  - 遍历：广度(最小代价)优先+剪枝
    - 接近解的结点(当前结点是叶子结点的父亲)： i=n-2 （从0开始）

      a(n-2,n-1),a(n-1,1)有边→判断是否构成回路　是解？

      cc+a(n-2,n-1)+a(n-1,1)<最小代价bestC?　是最优解？

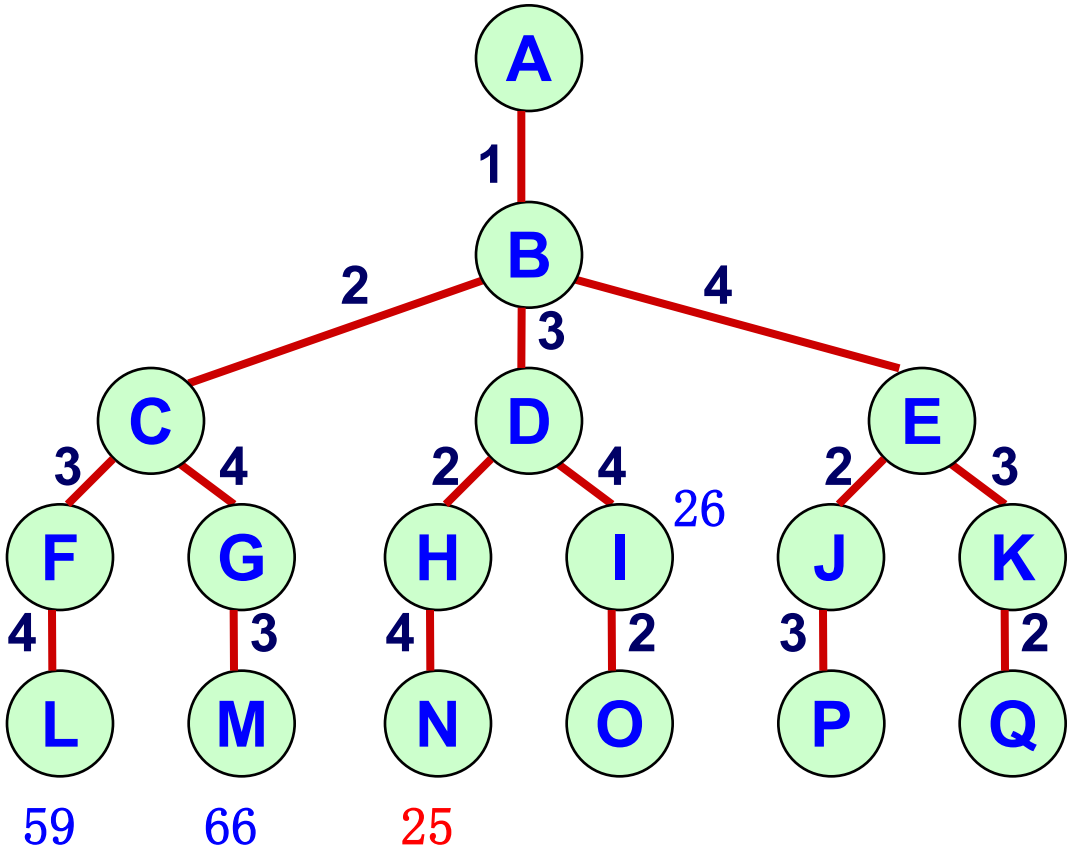
      剪枝： 当前费用>bestC

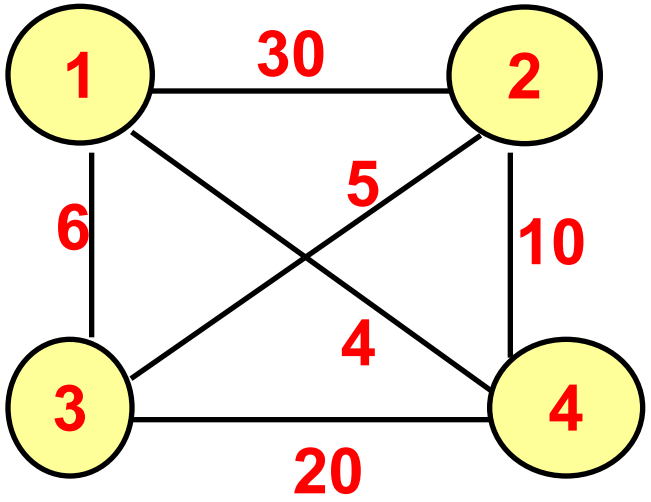
      i表示结点的层次
    - 中间结点： i<n-2 （从0开始）

      剪枝条件：当前费用cc>bestC
    - 结束条件： i=n-1
  - 改进
    - 剪枝：当前结点扩展后得到的最小费用的下界>当前最优值 ，剪去
    - 把每个结点的下界作为优先级

旅行售货员问题

- 实例——FIFO队列式

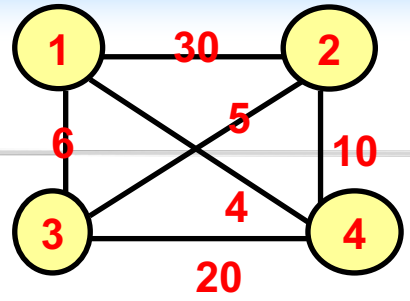


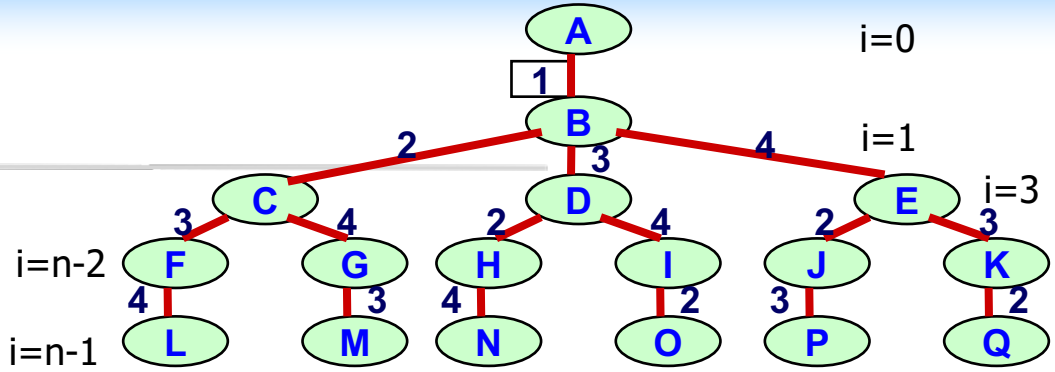


- ✓ B
- ✓ C, D, E
- ✓ F, G, H, I, J, K
- ✓ L, M, N, P, Q

旅行售货员问题

- FIFO队列式搜索过程





活结点队列	扩展结点	处理过程
空	B	B→C,D,E
EDC	C	C→F,G
GFED	D	D→H,I
IHGFE	E	E→J,K
KJIHGF	F	F→L
LKJIHG	G	G→M
LKJIH	H	H→N
NLKJI	I	I→O
NLKJ	J	J→P
NLK	K	K→Q
NL	L	L为叶子
N	N	N为叶子
空, 结束		

bestP=0

CDE均可行✓ ，入活

F: (2,3)有边✓, G（2,4) ✓

H:, ✓ J: ✓

J: ✓ K: ✓

L:叶子,(3,4)(4,1)有边,解(1,2,3,4,1) v=59, ✓ **bestP=59**

M:叶子,(4,3)(3,1)有边,解(1,2,4,3,1) v=66>59, X

N:叶子, 解(1,3,2,4,1), v=25<59 ✓ **bestP=25**

O:叶子, 解(1,3,4,2,1), v=66>59 X

P:叶子, 解(1,4,2,3,1), v=25=25 X

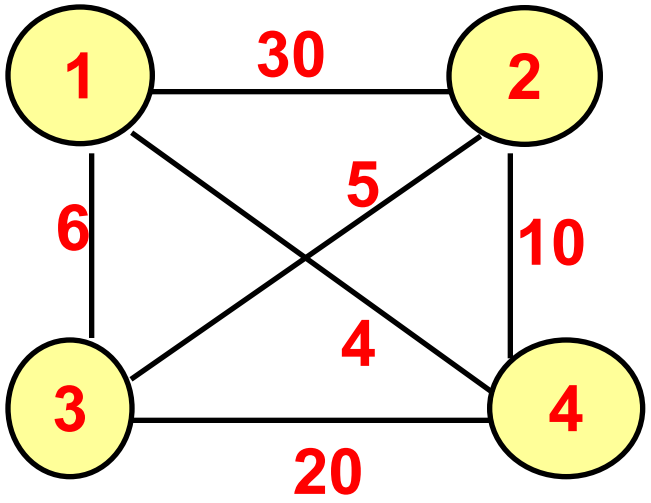
Q:叶子, 解(1,4,3,2,1), v=59>25 X

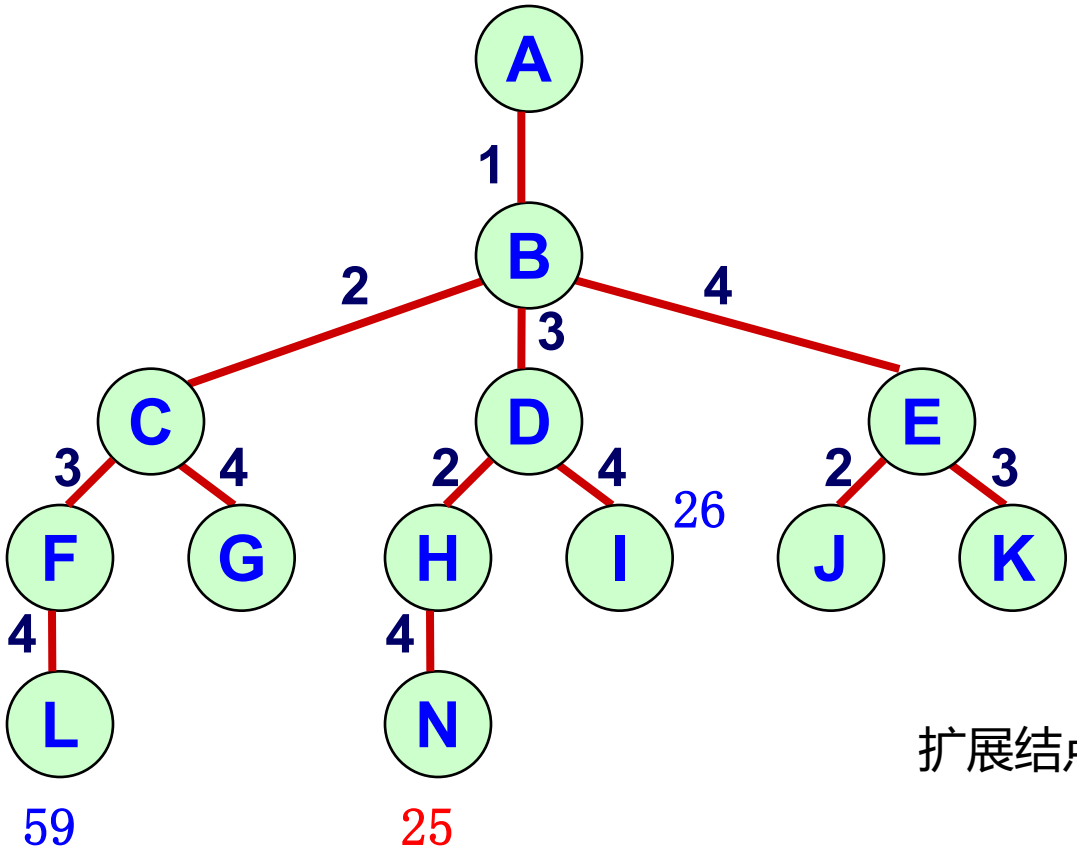
i=n-1, L:叶子, 解(1,4,3,2,1), v=59不是最优解 不输出

N:叶子, 解(1,3,2,4,1), v=25为最优解 输出

旅行售货员问题

- 实例——FIFO队列式
  - 生成排列树

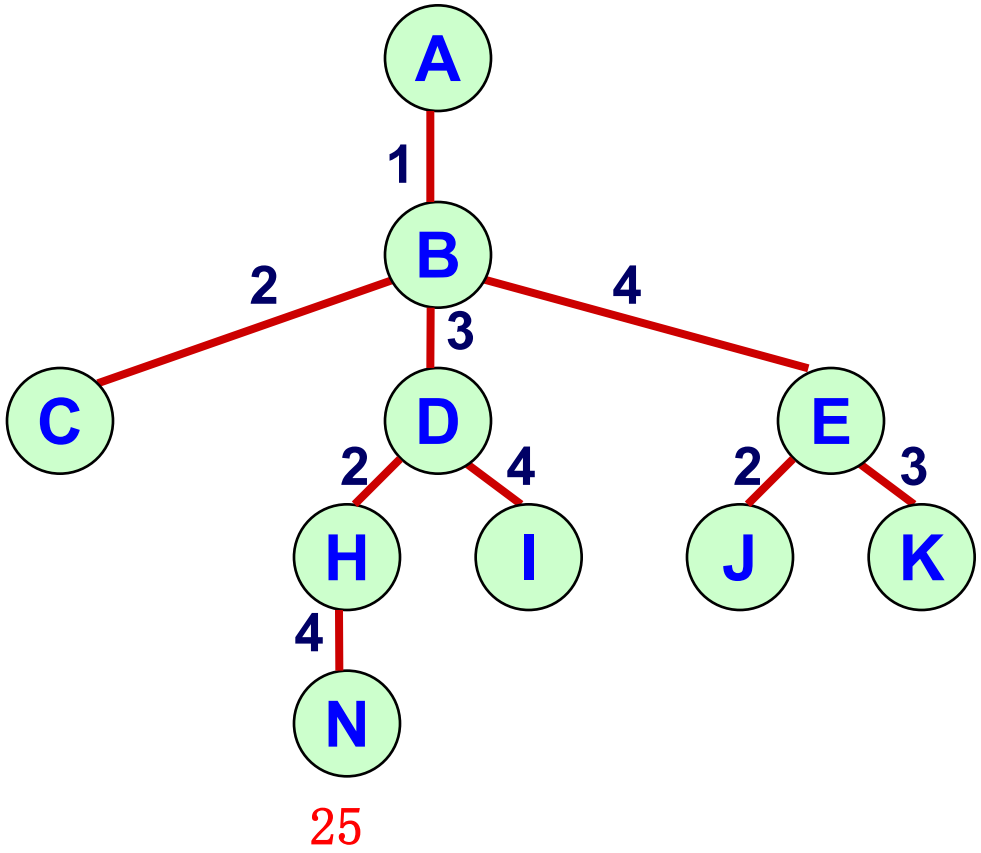


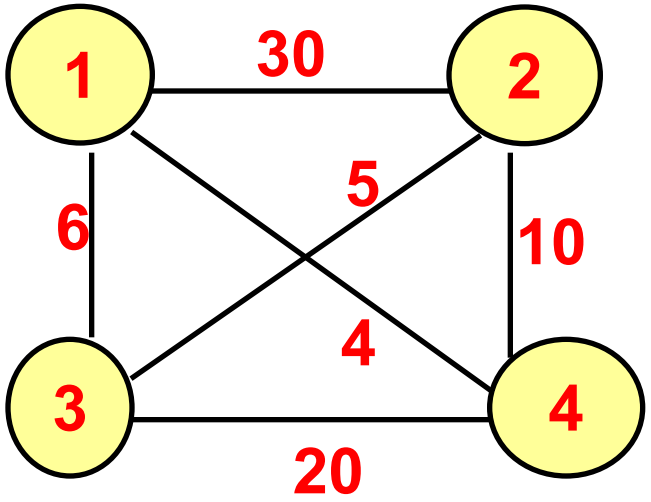


扩展结点表为：BCDEFGHIJKLN

旅行售货员问题

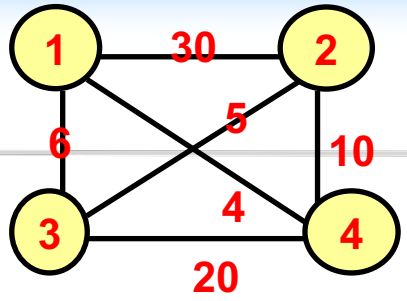
- 实例——优先队列式



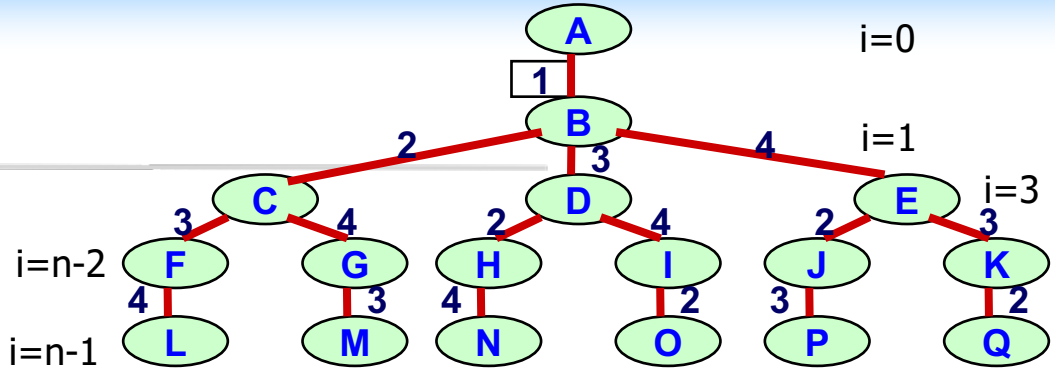


- B
- E, D, C
- D, J, K, C
- H, J, K, I, C
- J, K, N, I, C
- K, N, I, C
- N, I, C

旅行售货员问题



- 实例——优先队列式
  - 优先级：结点的当前费用



活结点队列(极小堆)	扩展结点	处理过程
空	B	B→C,D,E
ECD	E	E→J,K
DJKC	D	D→H,I
HJKIC	H	H→N
JKNIC	J	J→P
KNIC	K	K→Q
NIC	N	N为叶子

CDE均可行✓ ，入堆,V(C)=30, V(D)=6,V(E)=4

J,K可行✓, V(J)=14,V(K)=24 (堆顶D)

H,I可行 ✓, V(H)=11, V(I)=26 (堆顶H)

H: n-2, 为叶子父结点,, N为叶子(1,3,2,4,1)可行解 ✓ **bestC=25** V(N)=25

J: n-2, P为叶结点,(1,4,2,3,1)可行解, V(P)=25=bestC, 不更新 X

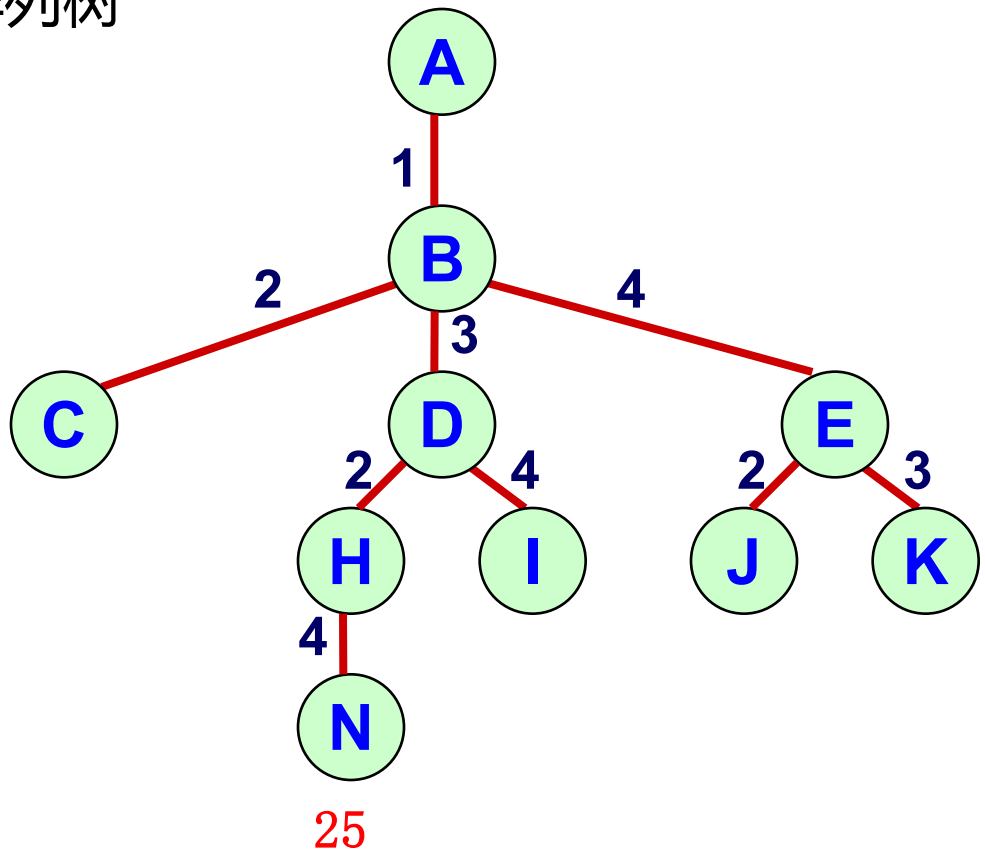
K:n-2,Q为叶结点, (1,3,3,2,1),可行解,V(Q)=59>bestC X

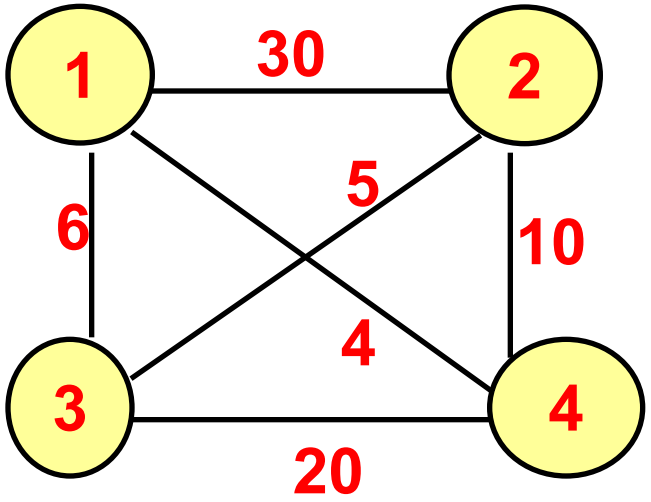
N:n-1, 跳出whilie循环，结束

扩展结点顺序：BEDHJKN

旅行售货员问题

- 实例——优先队列式
  - 优先级：结点的当前费用
    - 生成排列树





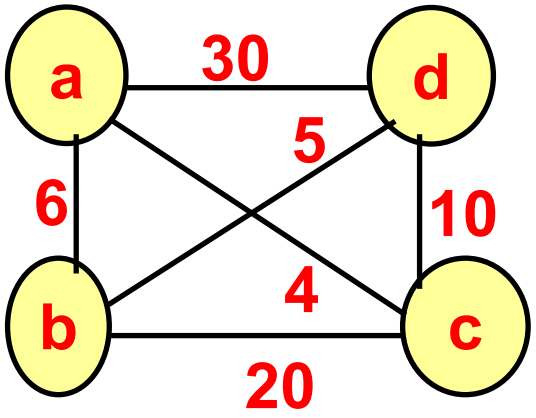
扩展结点顺序：BEDHJKN

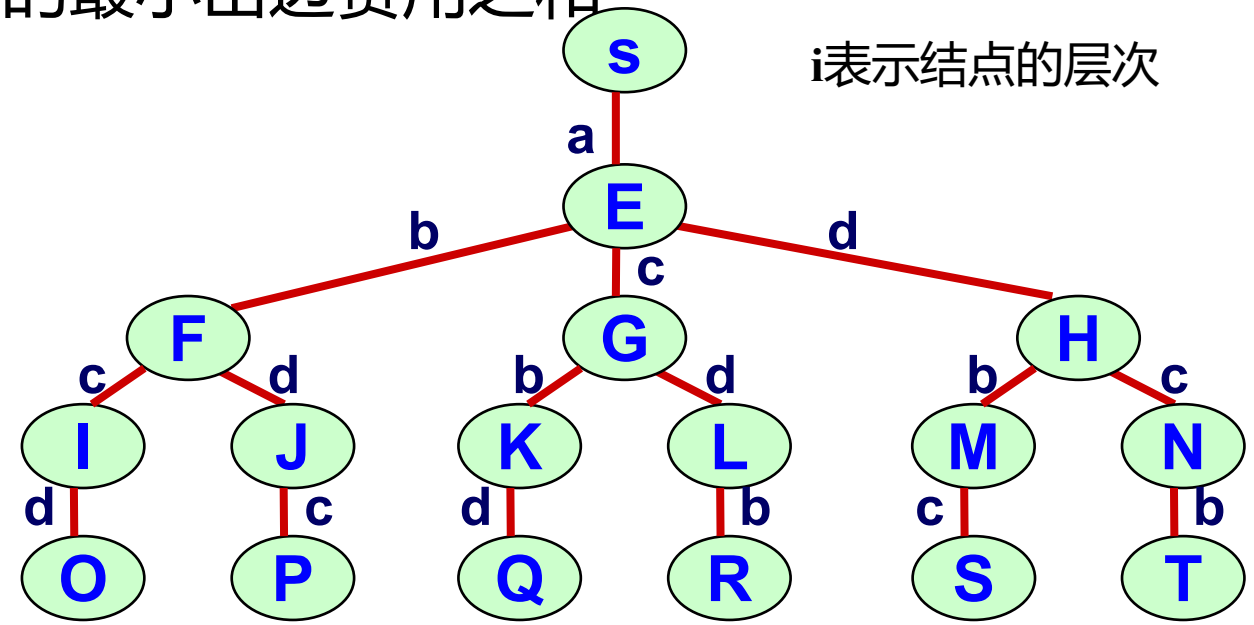
注意：不同的优先级定义会有不同扩展结点顺序

旅行售货员问题

- 实例——优先队列式：改进的优先级
  - 已知路径$P=\{v_1, v_2, v_3, \ldots, v_i\}$
  - 优先级为下界函数值
    - $\boldsymbol{Bound(E)} = cc + \sum_{j=i}^{n} Minout(v_j)$
    - $Minout(v_j)$是指每个顶点$\boldsymbol{v_j}$的最小出边
  - rcost定义为从源点s到n的最小出边费用之和

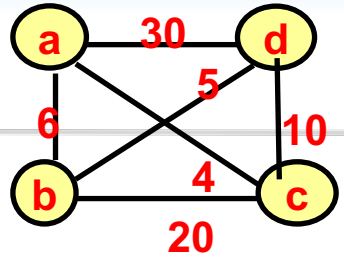
限界函数的改进能保证第一个找到的解就为最优解

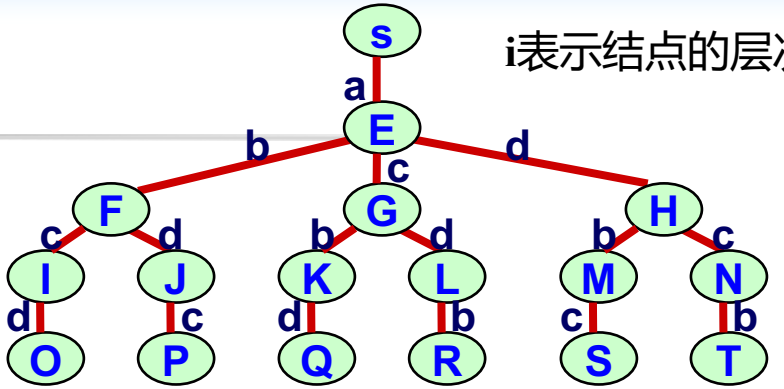




i表示结点的层次

旅行售货员问题





i表示结点的层次

- 改进的优先队列式
  - 优先级为下界函数值
    - $\boldsymbol{Bound(E)} = cc + \sum_{j=i}^{n} Minout(v_j)$ （剩余顶点）
      - cc: 当前路径长度
      - $Minout(v_j)$是指每个顶点$\boldsymbol{v_j}$的最小出边

Minout(a)=4, Minout(b)=5, Minout(c)=4, Minout(d)=5

Bound(E)=0+ $\sum_{j=i}^{4} Minout(v_j)$=18

Bound(F)=ab+min(b)+min(c)+min(d)=6+5+4+5=20

Bound(G)=ac+min(c)+min(b)+min(d)=4+4+5+5=18

Bound(H)=ad+min(d)+min(c)+min(b)=30+5+4+5=44

Bound(K)=ac+cb+min(b)+min(d)=4+20+5+5=34

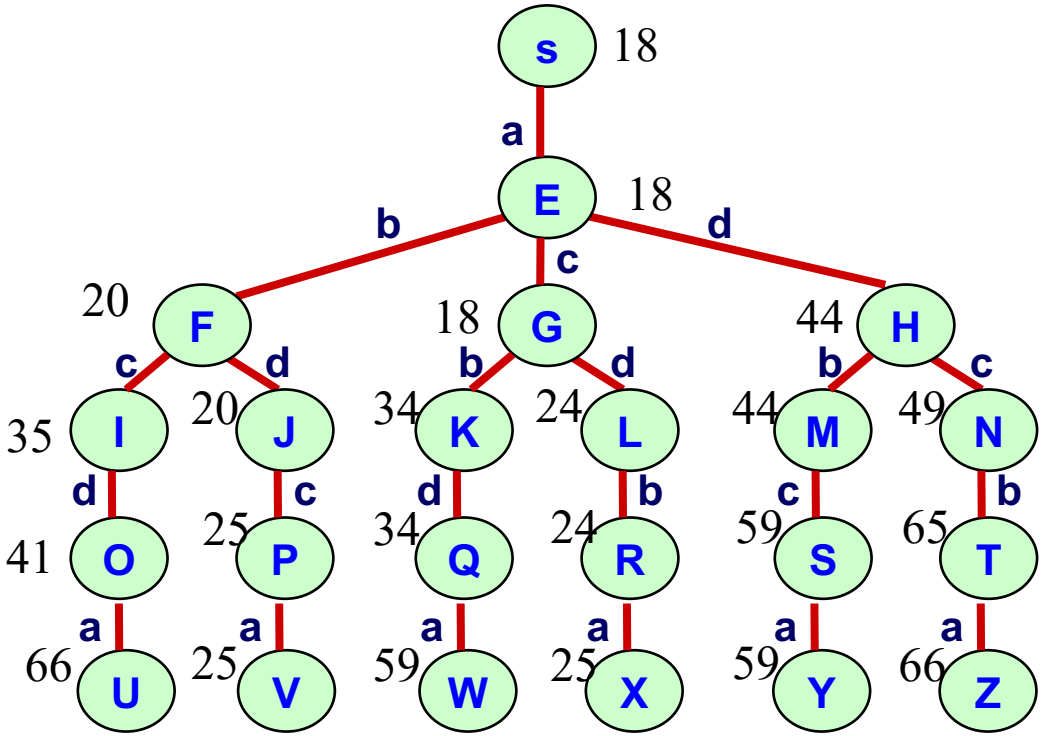
Bound(M)=ad+db+min(b)+min(c)=30+5+5+4=44

……..

Bound(U)=ab+bc+cd+da=6+20+10+30=66

Bound(Z)=ad+dc+cb+ba=30+10+20+6=66

….



装载问题

- 有一批共n个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱i的重量为w_i ，且

$$\sum_{i=1}^{n} w_i \leq c_1 + c_2$$

- 装载问题： 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

装载问题

- 有一批共n个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱i的重量为w_i ，且 $\sum_{i=1}^{n} w_i \le c_1 + c_2$
- 装载问题： 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
- 分析：
  - 与0-1背包问题的分支限界有什么相同点？
  - 相同点：
    - 可行性判定相同
    - 是否最优解判定：检查上界值
  - 不同点：
    - 装载没有最大价值的约束
    - 最优解判断剪枝函数不用贪心思想

装载问题

- 队列式分支限界法

搜索子集空间树

```
while (true) {
    // 检查左儿子结点
    if (Ew + w[i] <= c) // x[i] = 1
        EnQueue(Q, Ew + w[i], bestw, i, n);
    // 右儿子结点总是可行的
    EnQueue(Q, Ew, bestw, i, n); // x[i] = 0
    Q.Delete(Ew);          // 取下一扩展结点
}
```

bestw:当前最优解
Ew:当前扩展结点相应重量

EnQueue: 将活结点加入到活结点队列中

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

装载问题

- 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
    - 当ew+r ≤bestw时，可将其右子树剪去。
  - 提前更新bestw值的时间
    - 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。因为结点所相应的重量仅仅在搜索进入左子树时增加

装载问题

- 算法的改进

```
// 检查左儿子结点
  Type wt = Ew + w[i];     // 左儿子结点的重量
  if (wt <= c) {           // 可行结点
      if (wt > bestw) bestw = wt;          提前更新bestw
        // 加入活结点队列
      if (i < n) Q.Add(wt);
  }
// 检查右儿子结点
  if (Ew + r > bestw && i < n)          剪枝判断
      Q.Add(Ew);           // 可能含最优解
  Q.Delete(Ew);            // 取下一扩展结点
```

装载问题

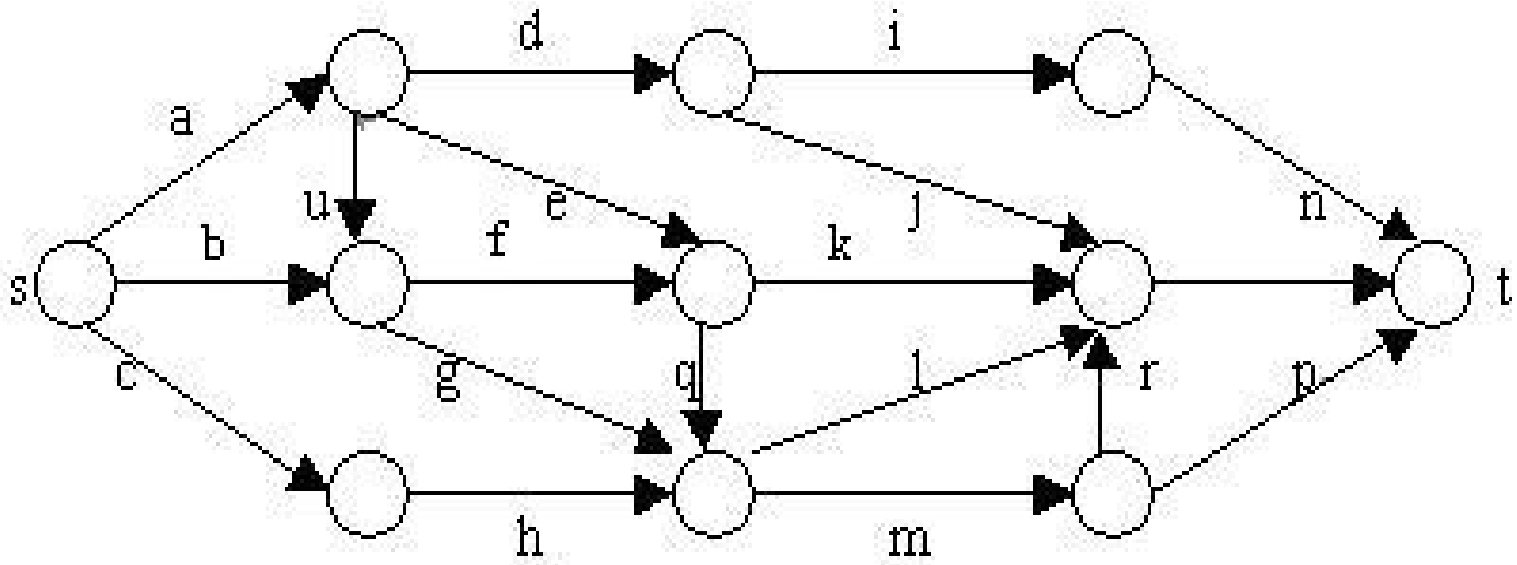
- 构造最优解
  - 为了在算法结束后能方便地构造出与最优值相应的最优解，算法必须存储相应子集树中从活结点到根结点的路径
  - 为此目的，可在每个结点处设置指向其父结点的指针，并设置左、右儿子标志
  - 找到最优值后，可以根据parent回溯到根节点，找到最优解

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
  - 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

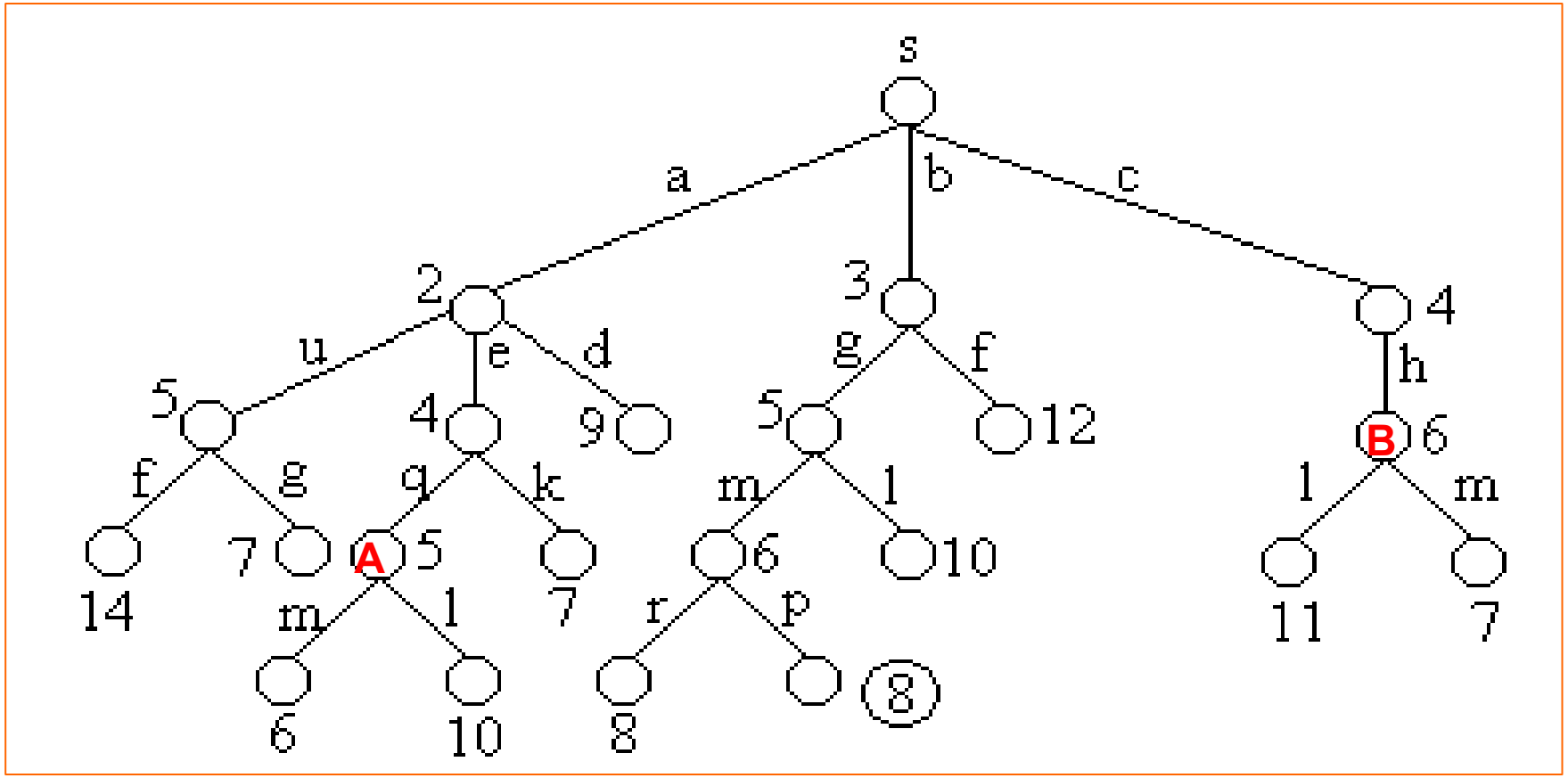
单源最短路径问题

- 问题描述
  - 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

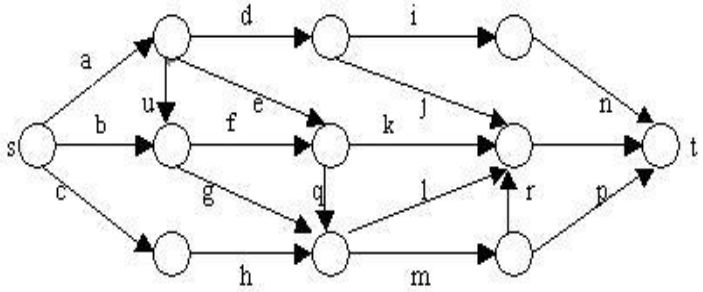


单源最短路径问题

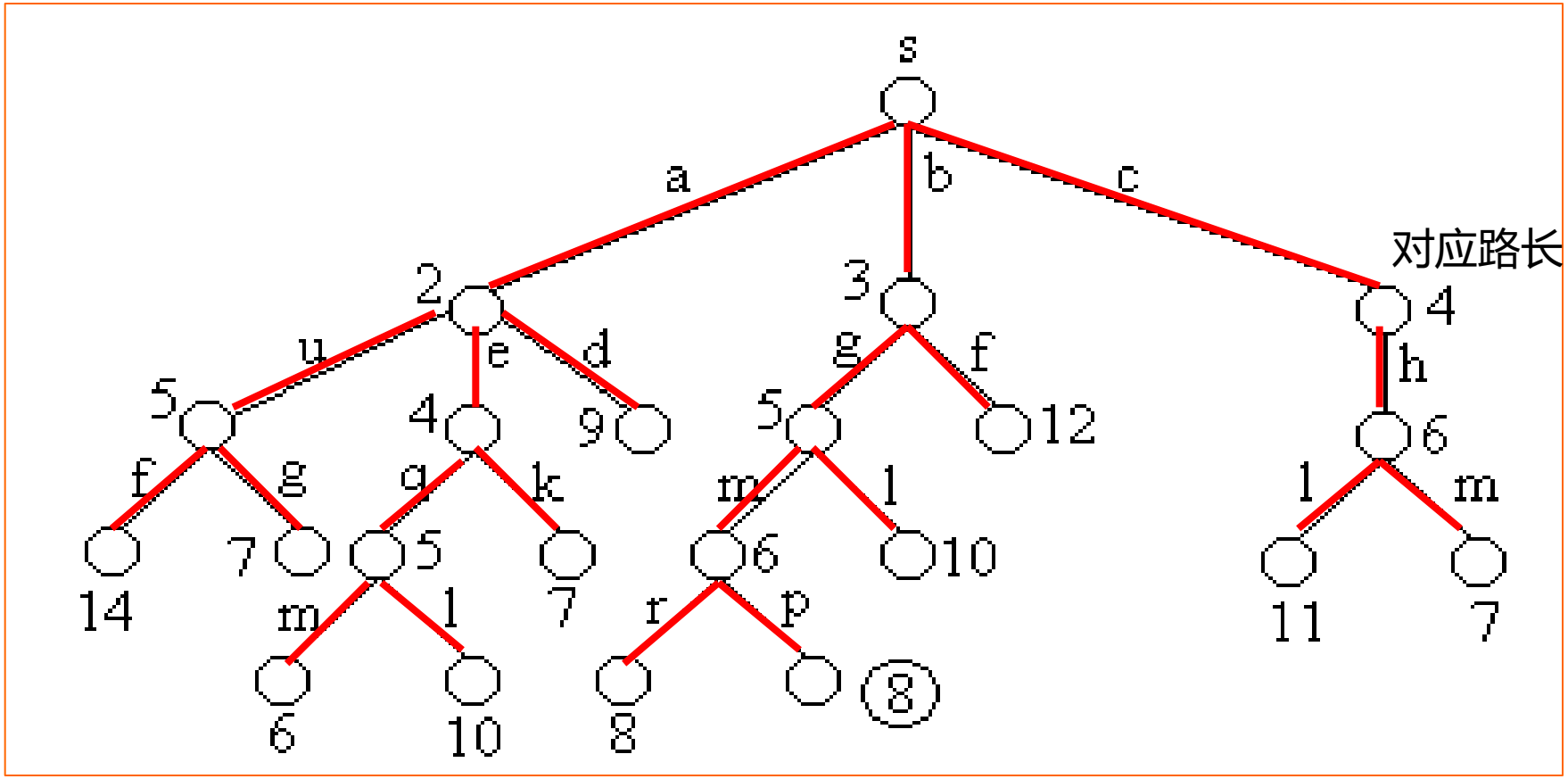
- 优先队列式分支限界法求解
  - 优先级：结点所对应的当前路长（结点旁数字）
  - 解空间树为：

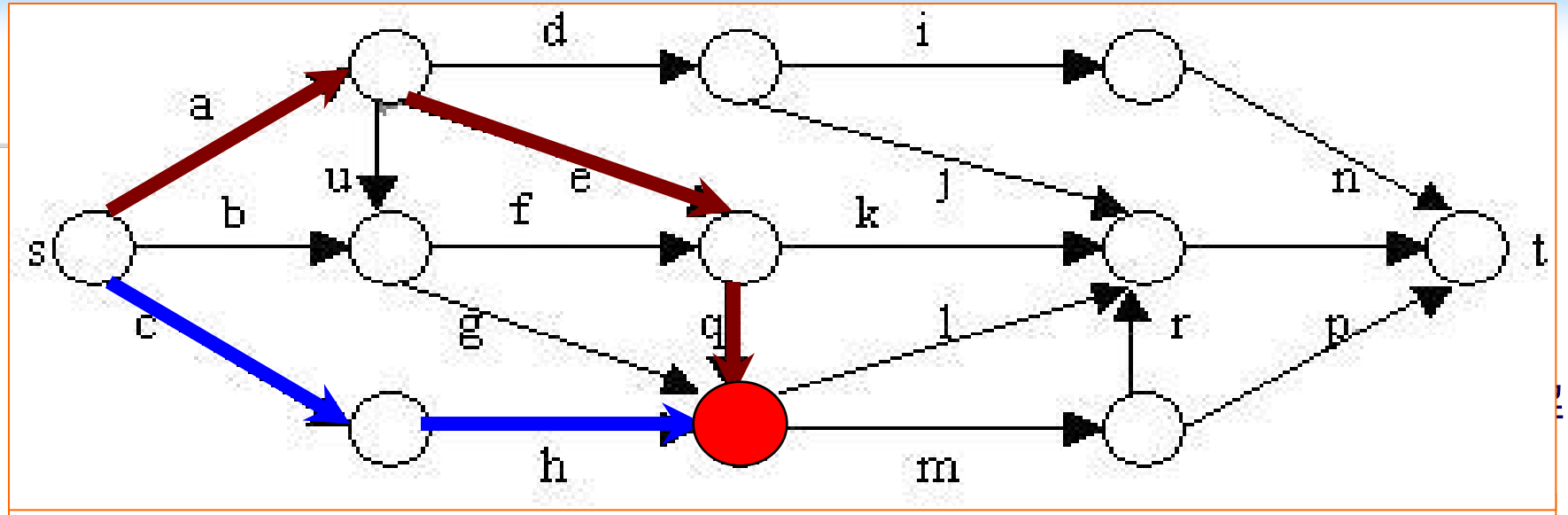


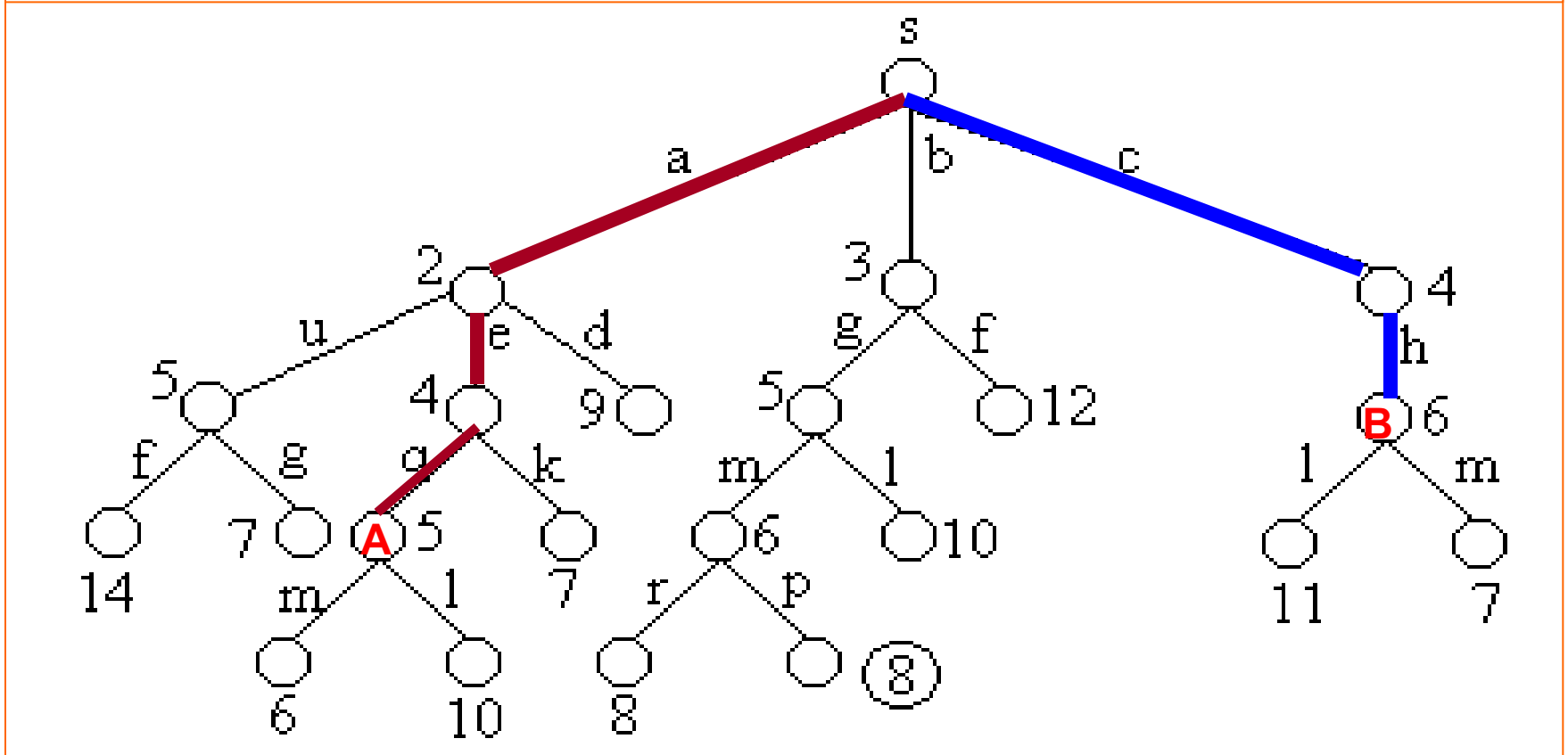
单源最短路径问题



- 活结点表：极小堆
- 解空间树：




d
i
a
u
e
j
n
b
f
k
s
t
c
g
q
l
r
p
h
m

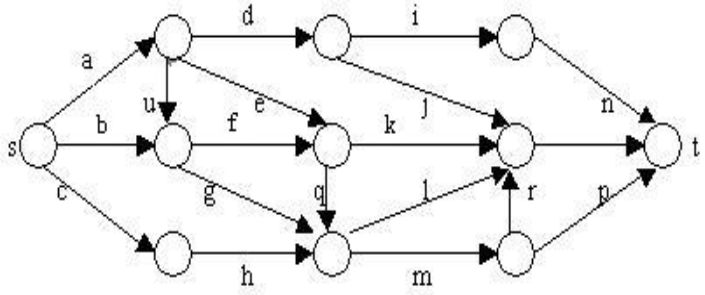

s
a
b
c
2
3
4
u
e
d
g
f
h
5
4
9
5
12
B
6
f
g
q
k
m
l
1
m
14
7
A
5
7
6
10
11
7
m
l
r
p
6
10
8
8

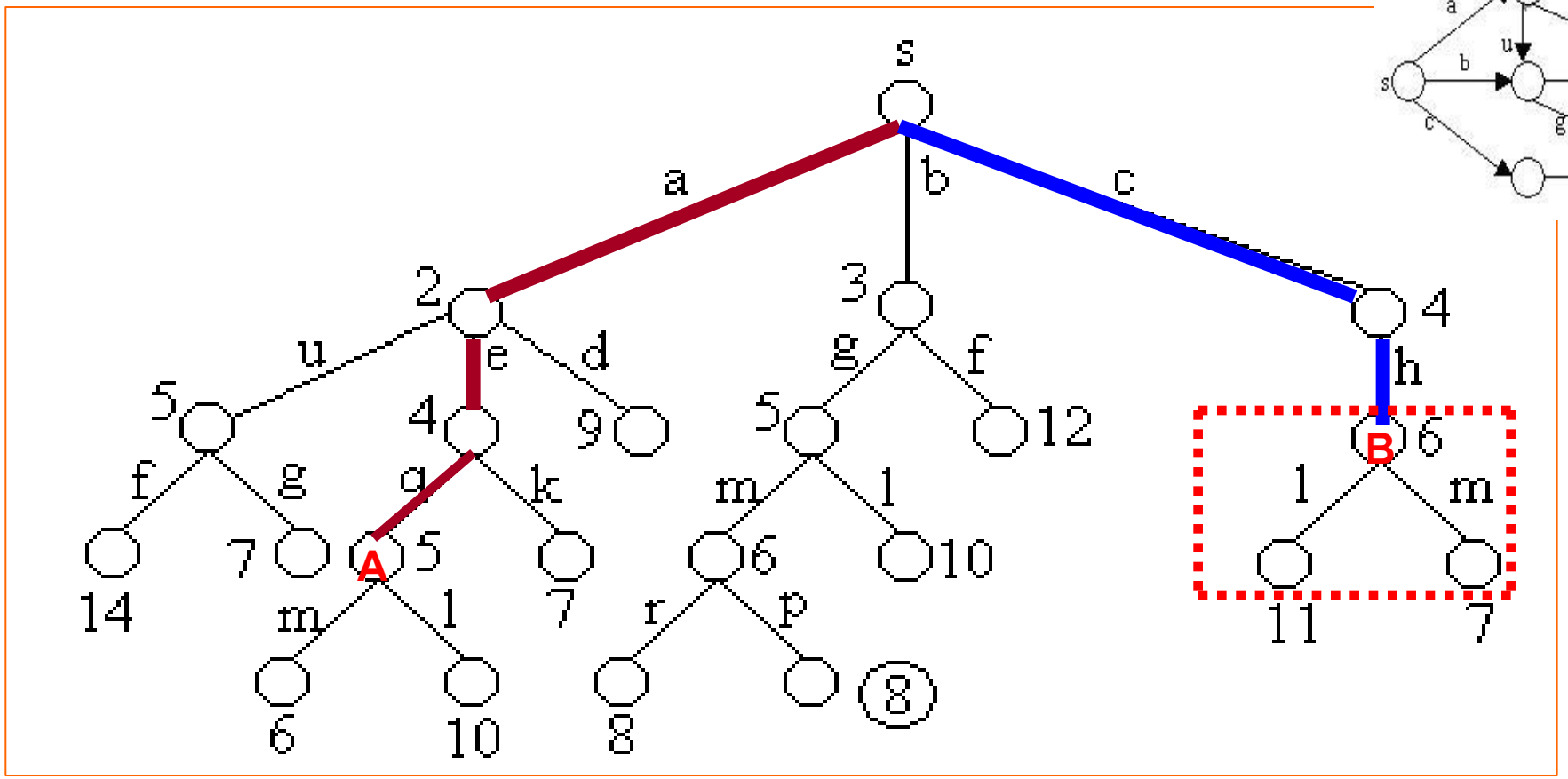
单源最短路径问题

- 结点A控制结点B

剪枝策略

  - 如果解空间树中以结点A为根的子树中所含的解优于以结点B为根的子树中所含的解，则称结点A控制结点B，以结点B为根的子树可以剪去。





单源最短路径问题

- 算法思想：解此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。
  - 初始化：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
  - While循环：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
  - While循环：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
    - 剪枝条件
  - While循环：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
    - 跳出循环条件

单源最短路径问题

- 算法描述

```
while (true) {  //搜索问题解空间
    for (int j = 1; j <= n; j++)  //依次检查与E.i相邻所有点
      if ((c[E.i][j]<inf)&&(E.length+c[E.i][j]<dist[j])) {
       // 顶点i到顶点j可达，且满足控制约束
          dist[j]=E.length+c[E.i][j];   //更新优先级
          prev[j]=E.i;
          // 加入活结点优先队列
          MinHeapNode<Type> N;
          N.i=j;
          N.length=dist[j];
          H.Insert(N); }
    try {H.DeleteMin(E);}        // 取下一扩展结点
    catch (OutOfBounds) {break;}  // 优先队列空
}
```

E.length: 源点到该点距离
c:图的邻接矩阵

剪枝条件：顶点i和j间有边，且此路径长小于原先从源点到j的路径长

Length: 优先队列的优先级

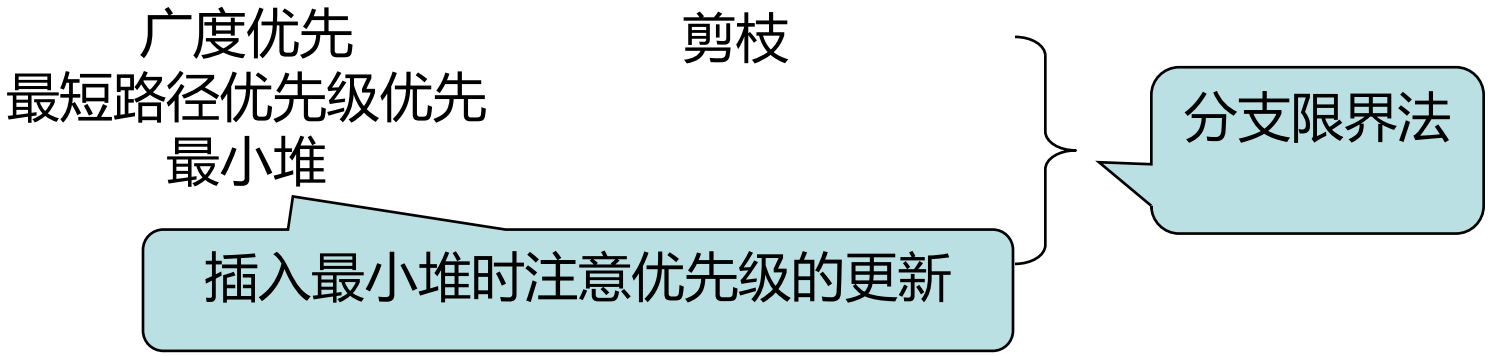
单源最短路径问题

- 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

- Dijkstra算法：贪心选择扩展顶点集合S

- 优先级的分支限界法：优先级是结点所对应的当前路长
  - 活结点队列:最小堆H



单源最短路径问题

- 算法描述的比较

```
BranchBand(){
…
while (true) {
    for (int j = 1; j <= n; j++)
      if((c[E.i][j]<inf)&&(E.length+c[E.i][j]<dist[j])) {
       // 顶点i到顶点j可达，且满足控制约束
           dist[j]=E.length+c[E.i][j];
                    prev[j]=E.i;
           // 加入活结点优先队列
           MinHeapNode<Type> N;
           N.i=j;
           N.length=dist[j];
           H.Insert(N);
       }
    try {H.DeleteMin(E);} // 取下一扩展结点
    catch (OutOfBounds) {break;}
                  // 优先队列空
}
```

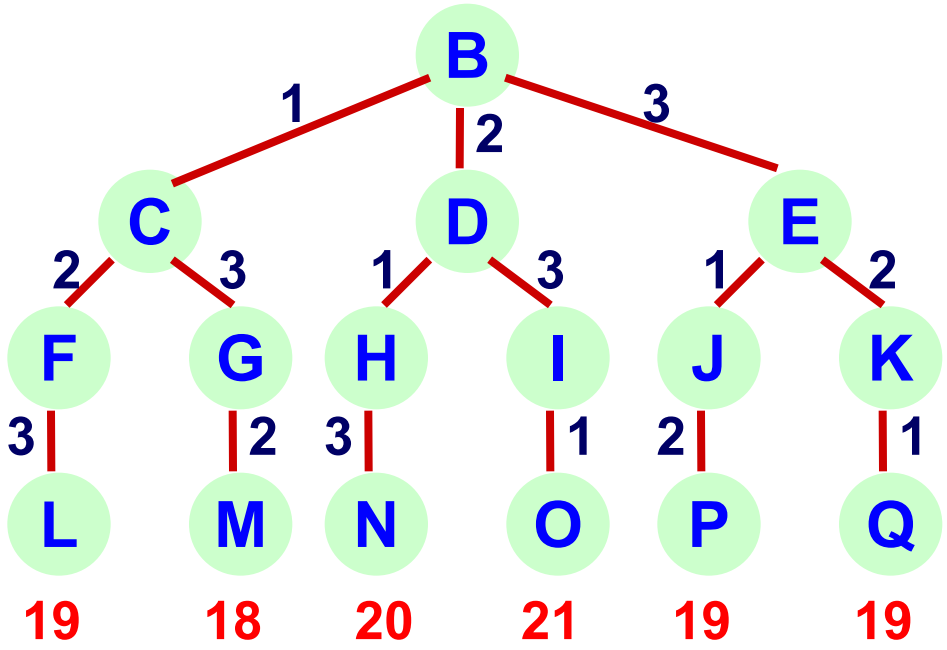
```
Dijkstra(int n, int v, long dist[],int prev[], long **c){
        ……//初始化
      for(int i=1; i<n; i++){   //何时终止
        for (int j=1; j<=n;j++){   //1）贪心选点
          if (!s[j]&&(dist[j]<temp)){
                    //特殊路径最短
            u=j;  temp=dist[j]}
        }
        s[u]=true;                    //2）加点
        for(int j=1; j<=n;j++){       //3）改权
          if((!s[j])&&(c[u][j]<maxint))  {
                    //从u到j有路径
            long newdist=dist[u]+c[u][j];
            if(newdist<dist[j]){
              dist[j]=newdist; pre[j]=u}
          }
        }
      }
}
```

批处理作业调度问题

- 输入
  - 给定n个作业的集合{J_1,J_2,...,J_n}。每个作业必须先由机器1处理，然后由机器2处理。
  - 作业j_i需要机器j的处理时间为t_{ji}。
- 输出
  - 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
    - 所有作业在机器2上完成处理的时间和称为该作业调度的完成时间和。
    - 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。

批处理作业调度问题

- 实例
  - 排列树

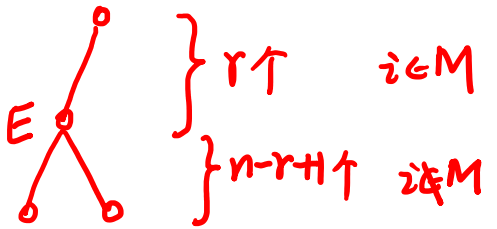


t_{ji}	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3

➢ 最佳调度方案是1,3,2，其完成时间和为18

批处理作业调度问题

- 在结点E处相应子树中叶结点完成时间和的下界
  - 每个E对应一个已安排的作业集M



- 以E为根的子树中所包含叶结点的完成时间和：

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

- 1) 机器1 紧凑: $\sum_{i \notin M} F_{2i} \geq \sum_{k=r+1} \left(F_{1p_r} + (n-k+1)t_{1p_k} + t_{2p_k}\right) = S_1$

机器1，$J_1 J_2 J_3 J_4$

机器2. t_{2p_k}

- 2) 机器2 紧凑: $\sum_{i \notin M} F_{2i} \geq \sum_{k=r+1} max\left(F_{2p_r}, F_{1p_r} + \min_{i \notin M} t_{1i}\right) + (n-k+1)t_{2p_k} = S_2$

机器1：$\min t_{1i}$

机器2：

- 两种情况下的极小值：

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

批处理作业调度问题

- 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p_k，使t_{1p_k}在k>=r+1时依非减序排列，S_1则取得极小值。同理如果选择p_k使t_{2p_k}依非减序排列，则S_2取得极小值

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

    - ✓ 这可以作为优先队列式分支限界法中的限界函数

讨论

- 分支限界法的难点
  - 如何选择结点的生成顺序?
  - 如何得到一个好的边界函数?

- 什么样的边界函数才是好的?
  - 函数容易计算
  - 又不能过于简单,因为它要尽可能的削剪解空间树的分支
  - 通过实验得到两者之间的一种平衡

小结

- 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
  - 队列式(FIFO)分支限界法
  - 优先队列（堆）式分支限界法

- 重点
  - 0-1背包问题
  - 旅行商问题
  - 限界函数的计算方法
  - 优先级的定义