分支限界算法

陈长建

计算机科学系

作业

- 1. 算法实现题6-2 最小权顶点覆盖问题
- 2. 算法实现题6-6 n后问题（分支限界法）

提交时间：12月26日

第四次上机实验

- **院楼103，12月29日**上午8:30-12:00（现场验收）
- 实验离线题（离线准备）
  - 回溯算法实现题5-4运动员最佳配对问题
  - 分支限界法求解实现题6-3无向图的最大割问题
- 在线题
  - acm.hnu.edu.cn

分支限界法 VS 回溯法

（1）求解目标

- 回溯法：能够找出解空间树中满足约束条件的所有解
- 分支限界法：
  - 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某种意义下的最优解

（2）搜索方式

- 回溯法：深度优先
- 分支限界法：广度优先或以最小耗费优先

分支限界法的基本思想

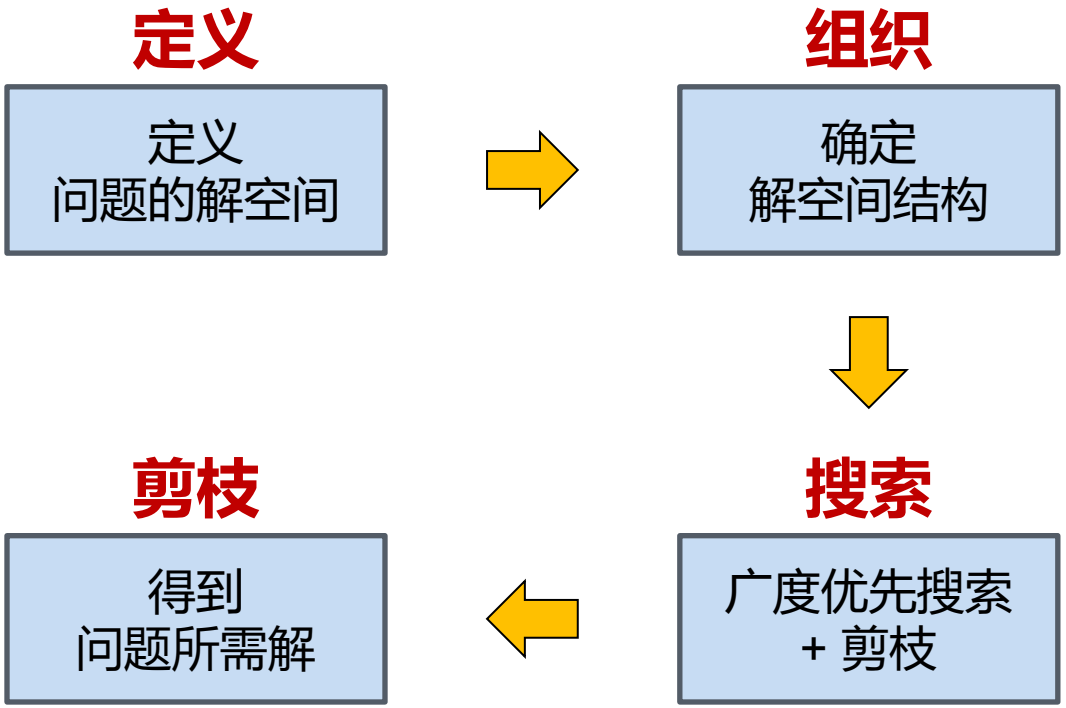
- 分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。
- 每一个活结点只有一次机会成为扩展结点。
  - 活结点一旦成为扩展结点，就一次性产生其所有子结点。
  - 在这些子结点中，导致不可行解或导致非最优解的子结点被舍弃，其余儿子结点被加入活结点表中。
- 从活结点表中取下一结点作为当前扩展结点，进行扩展。
- 直到找到所需的解或活结点表为空时为止。

分支限界法的基本思想

常见的两种分支限界法

- 队列式（FIFO）分支限界法
  - 按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。
- 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节点成为当前扩展节点。

分支限界法的基本步骤



难点：剪枝

(约束+限界)

例1- 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?
- 问题三要素：
  - 输入：$C > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$
  - 输出：$(x_1, x_2, \ldots, x_n), x_i \text{ in } \{0, 1\}$
  - 约束：$\sum_{1 \leq i \leq n} w_i x_i \leq C$，$\sum_{1 \leq i \leq n} v_i x_i$ 最大

算法思想

- 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。
- 优先队列分支限界法，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。

算法思想

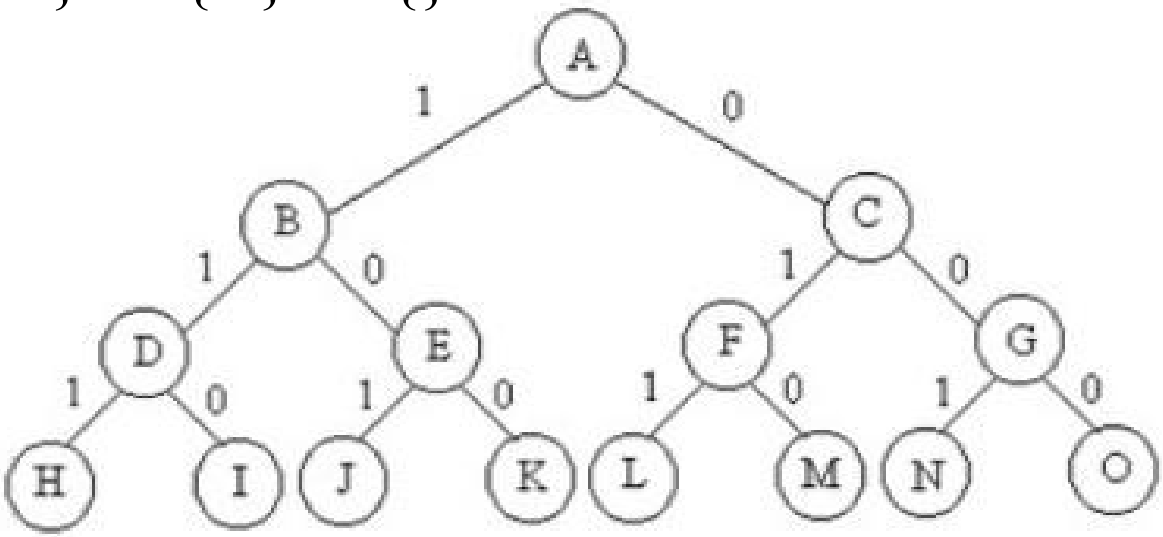
- 子集树
  - 首先检查当前扩展结点的左子结点的可行性。
  - 如果该左子结点是可行结点，则将它加入到子集树和活结点优先队列中。
  - 当前扩展结点的右子结点一定是可行结点，仅当右子结点满足上界约束时才将它加入子集树和活结点优先队列。
  - 当扩展到叶节点时为问题的最优值。

算法思想

- 例: 给定n=3，w={16,15,15}, p={45,25,25}, c=30。
  - 价值大者优先（为了简化计算）

算法思想

- 例: 给定n=3，w={16,15,15}, p={45,25,25}, c=30。
  - 价值大者优先（为了简化计算）
  - {}—>{A}—>{B, C}—>{C, D, E}—>{C, E}—>{C, J, K}—>{C}—>{F, G}—>{G, L, M}—>{G, M}—>{G}—>{N, O}—>{O}—>{}



示例代码

```
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);}
      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）
}
```

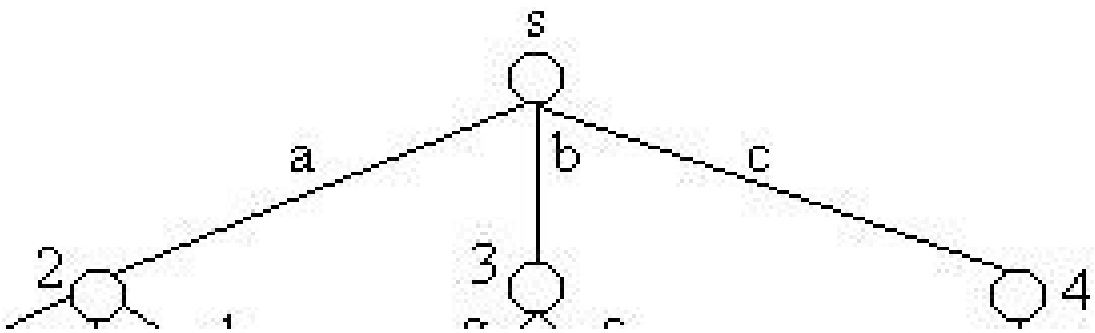
分支限界
搜索过程

示例代码

```
while (i <= n && w[i] <= cleft)      // n表示物品总数，cleft为剩余空间
{
        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
```

队列式 VS 优先队列式

- 队列式：子节点按生成顺序入队和出队
  - 例如，队列元素：可以有 {s}->{a, b, c}
- 优先队列式：子节点按优先级出队
  - 【若越大越好】{s}->{a, b, c}
  - 【若越小越好】{s}->{c, b, a}



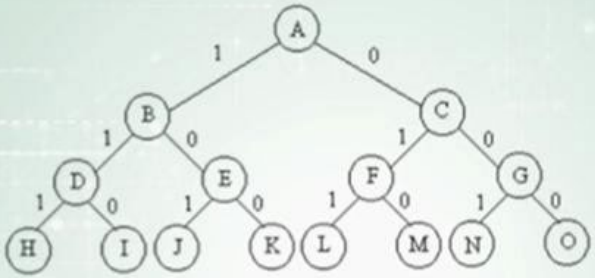
队列式 VS 优先队列式

- 队列式：子节点按生成顺序入队和出队
  - 均匀逐层推进
  - 约束+限界
- 优先队列式：子节点按优先级出队
  - 向最优解的方向快速推进
  - 优先级策略
  - 约束+限界

用限界函数计算优先级

解空间树 VS 搜索树

- 解空间树：不必考虑搜索策略
- 搜索树：考虑搜索策略+剪枝策略



	广度优先搜索	分支限界算法
图搜索视角	遍历整棵完全二叉树	添加剪枝策略，尽量限制搜索无效的分支
枚举视角	全局判定 $[x1, x2, \cdots, xn]$	局部预判 $[x1, ?, ?, \cdots]$ $[x1, x2, ?, \cdots]$

练习

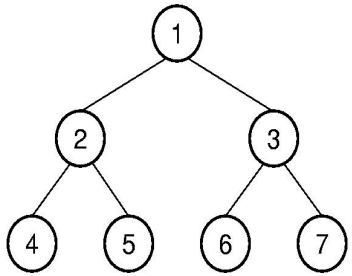
- 给定n=3，w={16,15,15}, p={45,25,25}, c=30。请用队列式分支限界法和优先队列式分支限界法求解该0-1背包问题，并对比其异同。

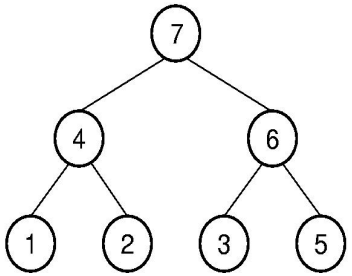
关键数据结构： 堆栈和队列

- 堆栈：先进后出
- 队列：先进先出
- 优先级队列: 顺序入队，高优先级先出队

优先级队列：最大/小堆（heap）

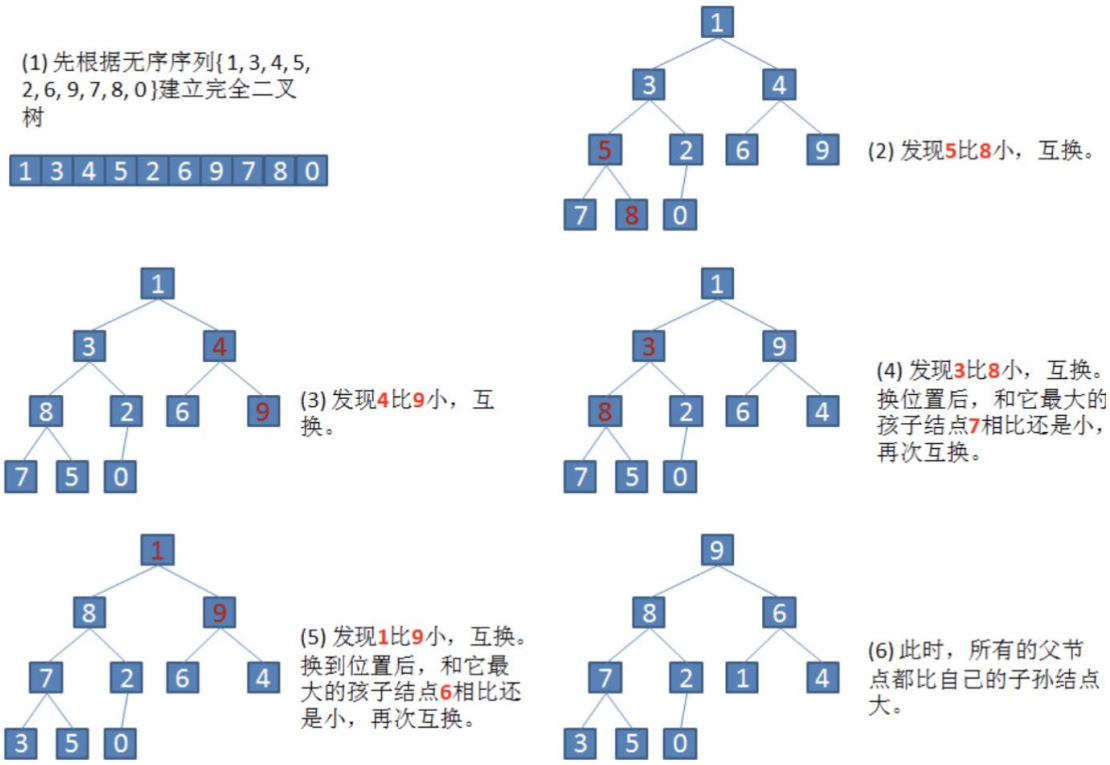
- 堆树或者是一棵空树，或者是具有下列性质的一棵完全二叉树（堆的局部有序特性），堆可以用来表现优先级
  - 最小值堆：每一个结点存储的值都小于或等于其子结点存储的值
  - 最大值堆：每一个结点的值都大于或等于其任意一个子结点存储的值





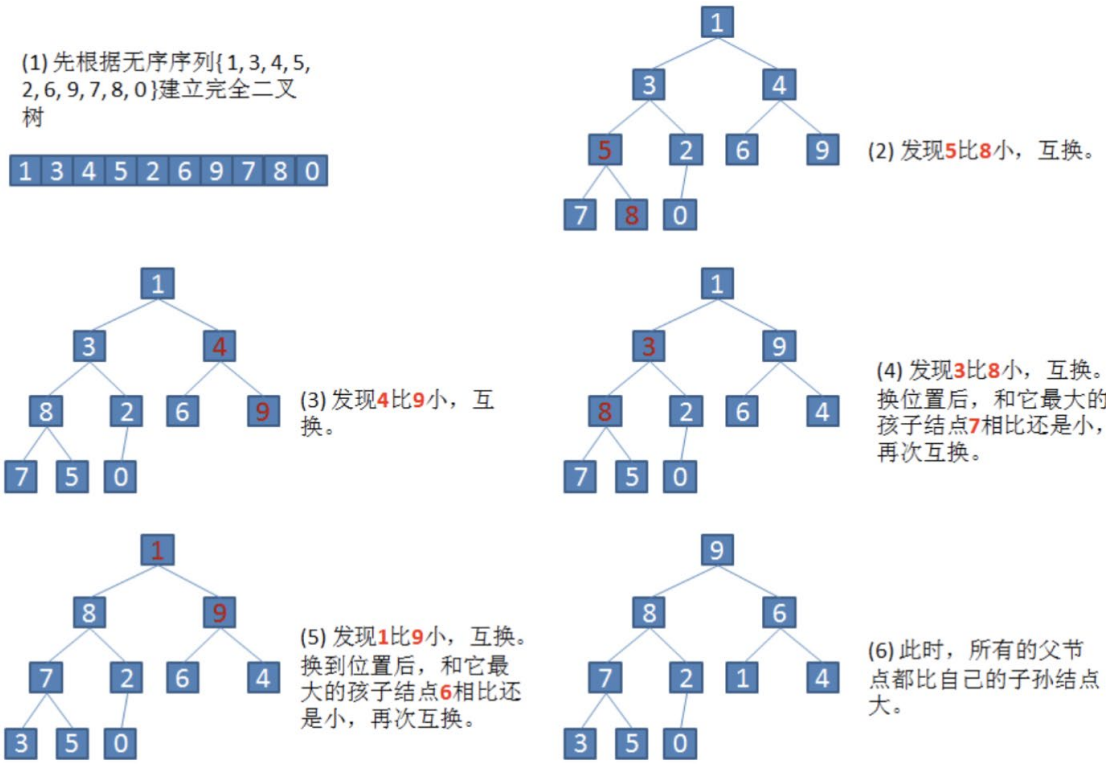
最大/小堆（heap）

- 堆的复杂度
  - 构建: O(N)[why?]
  - 插入: O(logN)



最大/小堆（heap）

- 堆的复杂度
  - 构建: O(N)
  - 插入: O(logN)
- 链表
  - 构建：O(N)
  - 插入：O(N)



例2-两艘船的装载问题

- 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，且满足

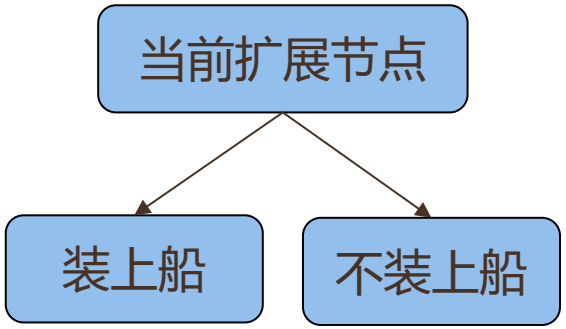
$$\sum_{i=1}^{n} w_i \leq c_1 + c_2$$

- 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。

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。

队列式分支限界法

- 首先检测当前扩展结点的左儿子结点是否为可行结点。若是，加入到活结点队列。再将其右儿子结点加入(一定是可行结点)。2个儿子结点都产生后，舍弃当前扩展结点。



- 非空判断及下一层扩展
  - 活结点队列中的队首元素被取出作为当前扩展结点。队列中每一层结点之后有一尾部标记-1。
  - 当取出的元素是-1时，再判断当前队列是否为空。若非空，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。

示例代码

```
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
    if (Ew == -1) {           // 同层结点尾部
        if (Q.IsEmpty()) return bestw;
        Q.Add(-1);            // 同层结点尾部标志
        Q.Delete(Ew);         // 取下一扩展结点
        i++; }                // 进入下一层
} }
```

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。
- 为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

示例代码

```
// 检查左子结点
Type wt = Ew + w[i];        // 左子结点的重量
    if (wt <= c) {          // 可行结点
        if (wt > bestw) bestw = wt;
        // 加入活结点队列
        if (i < n) Q.Add(wt);
 }

// 检查右子结点
if (Ew + r > bestw && i < n)
    Q.Add(Ew);              // 可能含最优解
    Q.Delete(Ew);           // 取下一扩展结点
```

提前更新bestw

右子节点剪枝

构造最优解

- 为方便构造与最优值相应的最优解，须存储子集树中从活结点到根结点的路径。为此，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。

```
class QNode
{
   QNode *parent; // 指向父结点的指针
   bool LChild;    // 左儿子标志
   Type weight;    // 结点所相应的载重量
}
```

```
// 构造当前最优解
for (int j = n - 1; j > 0; j--)
{
   bestx[j] = bestE->LChild;
   bestE = bestE->parent;
}
```

优先队列式分支限界法

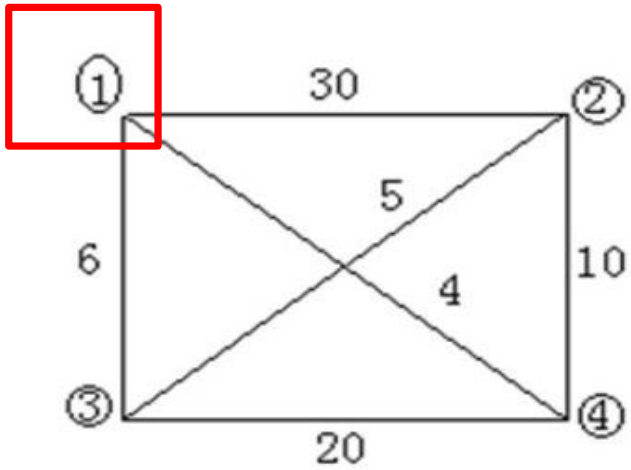
- 用最大优先队列存储活结点表。活结点x的优先级定义为从根结点到结点x的路径所相应的载重量再加上剩余集装箱的重量之和。
- 优先级最大的活结点成为下一个扩展结点。以结点x为根的子树中所有结点相应的路径的载重量不超过它的优先级。子集树中叶结点所相应的载重量与其优先级相同。
- 一旦有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的解即为最优解。此时可终止算法。

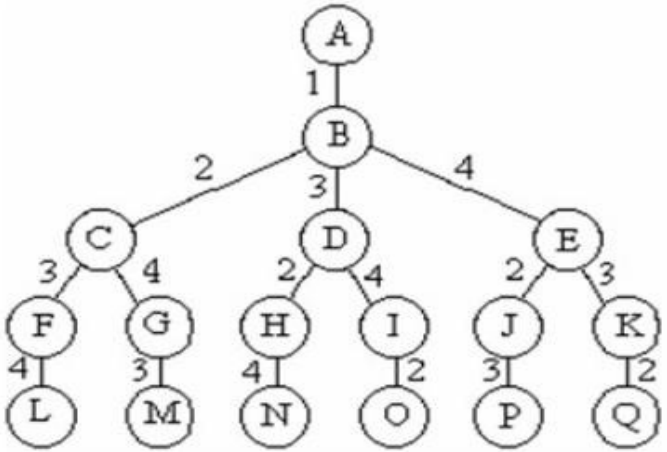
练习

- 3个集装箱，重量分别为 5, 20, 10，两艘船载重量分别为C1=30，C2=45。请分别用队列式分支限界法和优先队列式分支限界法求解该问题，并写出相应的节点序列。并对比其异同。

分支限界法案例3- 旅行商（TSP）问题

- 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（旅费），他要选定一条从驻地出发，经过每个城市一遍，最后回到驻地的路线，使总的路程（总旅费）最小。





算法描述（1/4）-初始化

- 创建一个最小堆表示活结点优先队列。堆中每个结点的子树费用的下界 lcost 值是优先队列的优先级。
- 计算出图中每个顶点的最小费用出边并用 minout 记录。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。

算法描述（2/4）-内部节点扩展

- 叶节点的父节点： s=n-2。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。

算法描述（3/4）-内部节点扩展

- 更上层节点：s<n-2，依次产生当前扩展结点的所有子结点。
  - 由于当前扩展结点所相应的路径是x[0:s]，其可行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 下界/优先级： lcost=cc+sum(minout)

算法描述（4/4)- 循环终止条件

- 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。

算法结束时的效果

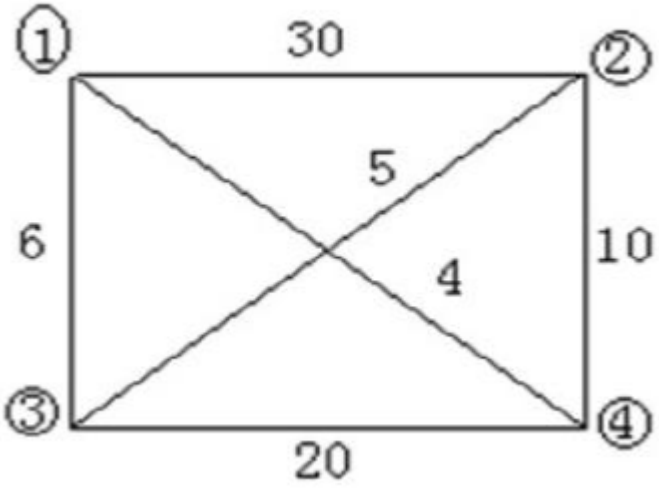
- 第一条到达叶节点的即最优解。
- 此时该叶结点所相应的回路的费用等于cc和lcost的值。
- 剩余的活结点的lcost值不小于已找到的回路的费用。它们都不可能导致费用更小的回路。因此已找到的叶结点所相应 的回路是一个最小费用旅行售货员回路，算法可以结束。
- 算法结束时返回找到的最小费用，相应的最优解由数组v 给出。

示例代码

```
//搜索排列空间树
while(E.s<n-1)//非叶结点
{
    if(E.s == n-2)//当前扩展节点是叶节点的父节点
    {
        //再加2条边构成回路
        //所构成回路是否优于当前最优解
        if(a[E.x[n-2]][E.x[n-1]]!=NoEdge && a[E.x[n-1]][1]!=NoEdge
            && (E.cc+a[E.x[n-2]][E.x[n-1]]+a[E.x[n-1]][1]<bestc
            || bestc == NoEdge))
        {
            //费用更小的回路
            bestc = E.cc + a[E.x[n-2]][E.x[n-1]]+a[E.x[n-1]][1];
            E.cc = bestc;
            E.lcost = bestc;
            E.s++;
            H.Insert(E);
        }
        else
        {
            delete[] E.x;//舍弃扩展节点
        }
    }
    else//产生当前扩展节点的儿子节点
    {
        if(a[E.x[E.s]][E.x[i]]!=NoEdge)
        {
            //可行儿子节点
            Type cc = E.cc + a[E.x[E.s]][E.x[i]];
            Type rcost = E.rcost - MinOut[E.x[E.s]];
            Type b = cc + rcost;//下界
            if(b<bestc || bestc == NoEdge)
            {
                //子树可能含有最优解
                //节点插入最小堆
                MinHeapNode<Type> N;
                N.x = new int[n];
                for(int j=0; j<n; j++)
                {
                    N.x[j] = E.x[j];
                }
                N.x[E.s+1] = E.x[i];
                N.x[i] = E.x[E.s+1];
                N.cc = cc;
                N.s = E.s + 1;
                N.lcost = b;
                N.rcost = rcost;
                H.Insert(N);
            }
        }
    }
    delete []E.x;//完成节点扩展
```

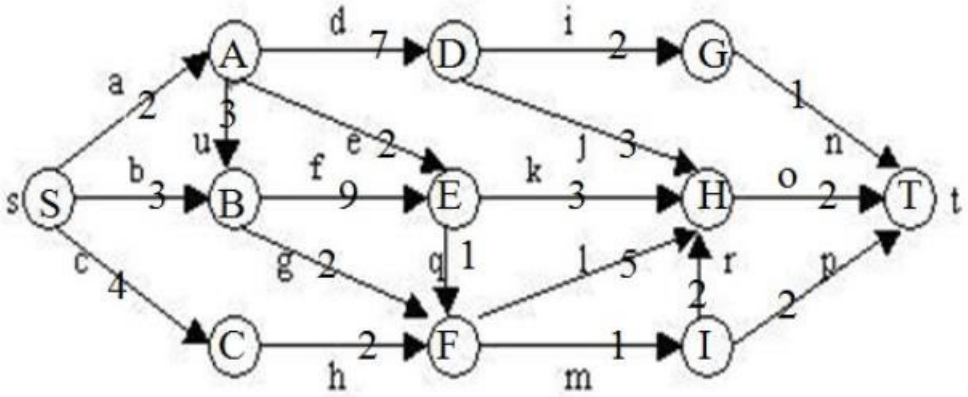
练习

- 优先级队列式分支限界法 求解如下TSP问题



分支限界法案例4- 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。
- 输入：一个有向加权连通图G = <V,E,w>
- 输出：从源点s到终点t的最短路径
- 约束条件：路径最短

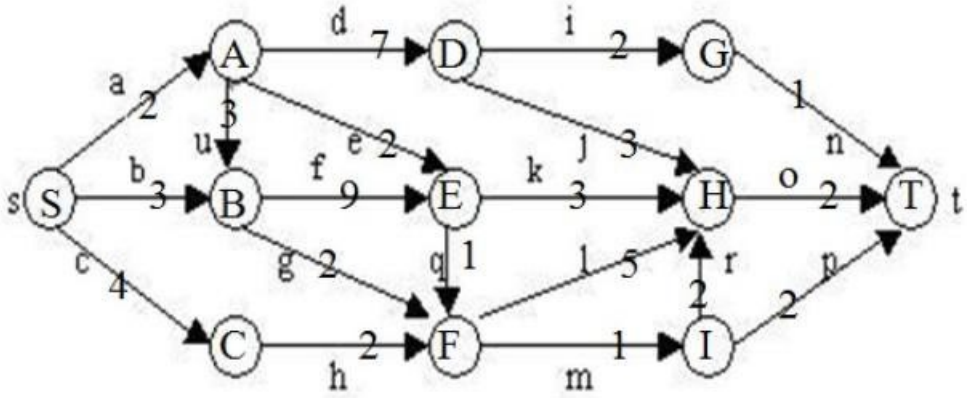


分支限界法案例4- 单源最短路径问题

图G

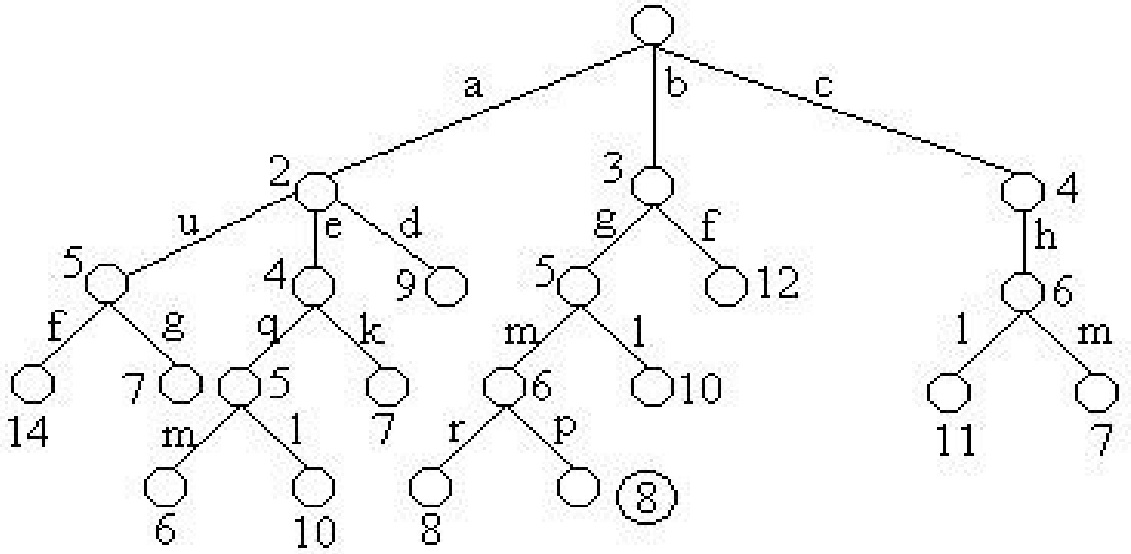
源点S

目标节点T



解空间树

路径长



优先队列式分支限界法-算法思想

- 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。
- 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。
- 继续扩展过程，直到活结点优先队列为空。

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。

剪枝策略

- 在扩展结点的过程中，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。
- 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。可将路长较长的路径所对应的子树剪去。

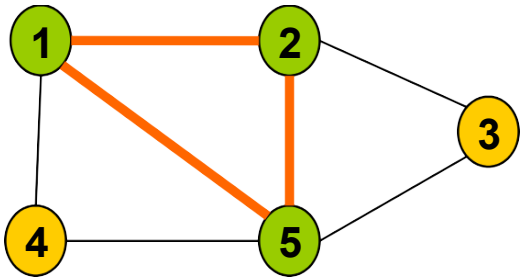
示例代码

```
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
    }
}
```

顶点i和j间有边，且此路径长小于原先从原点到j的路径长

分支限界法案例5- 最大团问题

- 团（clique）：社会团体，团体中的个体互相认 识。
- 最大团问题（Maximum Clique Problem, MCP）是图论中一个经典的组合优化问题，也是一类NP完全问题。



例：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。

上界函数及优先级

- 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
- cliqueSize+n-level+1作为顶点数上界upperSize的值
- upperSize实际上也是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

算法思想：子集树

- 子集树的根结点是初始扩展结点，其cliqueSize的值为0
- 扩展内部结点时，
  - 先考察左子结点。将顶点 i 加入到当前团中，并检查可行性，即该顶点与当前团中其它顶点是否都有边相连。
    - 可行结点，将加入到子集树中并插入活结点优先队列。
  - 继续考察当前扩展结点的右子结点。
    - 当upperSize>bestn时，右子树中可能含有最优解，此时将右子结点加入到子集树中并插入到活结点优先队列中。

算法思想：while循环的终止条件

- 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。

示例代码

```
//搜集子集空间树
while(i!=n+1)//非叶节点
{
    //检查顶点i与当前团中其他顶点之间是否有边相连
    bool OK = true;
    bbnode *B = E;
    for(int j=i-1; j>0; B=B->parent,j--)
    {
        if(B->LChild && a[i][j]==0)
        {
            OK = false;
            break;
        }
    }

//构造当前最优解
for(int j=n; j>0; j--)
{
    bestx[j] = E->LChild;
    E = E->parent;
}

return bestn;
```

```
    if(OK)//左儿子节点为可行结点
    {
        if(cn+1>bestn)
        {
            bestn = cn + 1;
        }
        AddLiveNode(H,cn+1,cn+n-i+1,i+1,E,true);
    }

    if(cn+n-i>=bestn)//右子树可能含有最优解
    {
        AddLiveNode(H,cn,cn+n-i,i+1,E,false);
    }

    //取下一扩展节点
    CliqueNode N;
    H.DeleteMax(N); //堆非空
    E = N.ptr;
    cn = N.cn;
    i = N.level;
}
```