第6章　分支限界法
(Branch and Bound)

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

（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。

(1)单源最短路径问题；(2)装载问题；(3)0-1背包问题；

(4)最大团问题；(5)旅行售货员问题；(6)批处理作业调度问题

6.1 分支限界法的基本思想

分支限界法与回溯法

- 求解目标：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。

- 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

- 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

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）原则选取下一个节点为扩展节点。

数据结构：队列

（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节点成为当前扩展节点。

数据结构：堆

最大优先队列：最大堆，体现最大效益优先

最小优先队列：最小堆，体现最小费用优先

队列式(FIFO)分支限界法

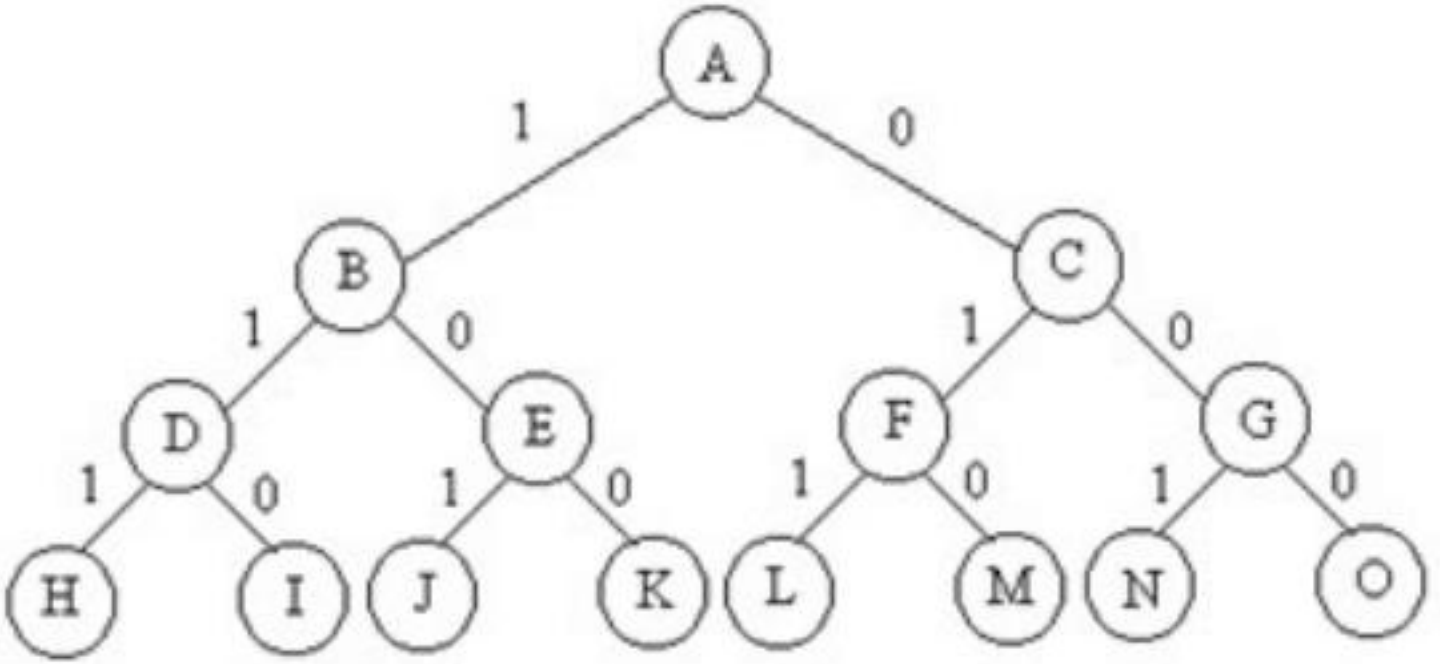
- 搜索策略：一开始，根结点是唯一的活结点，根结点入队。从活结点队中取出根结点后，作为当前扩展结点。对当前扩展结点，先从左到右地产生它的所有儿子，用约束条件检查，把所有满足约束函数的儿子加入活结点队列中。再从活结点表中取出队首结点为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

优先队列式分支限界法

- 搜索策略：对每一活结点计算一个优先级（某些信息的函数值），并根据这些优先级；从当前活结点表中优先选择一个优先级最高（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。再从活结点表中下一个优先级别最高的结点为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

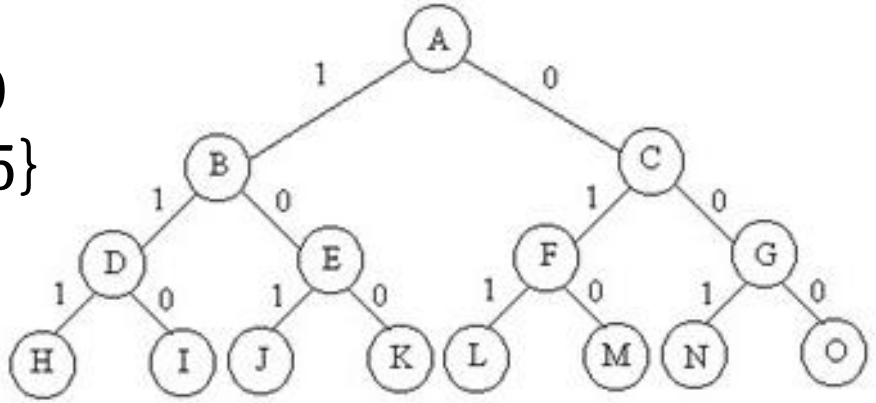
分支限界法搜索应用举例

- 0-1背包问题，当n=3时，w={16, 15, 15}，p={45, 25, 25}， c=30



分支限界法搜索应用举例

0-1背包问题，当n=3时， c=30
w={16, 15, 15}， p={45, 25, 25}

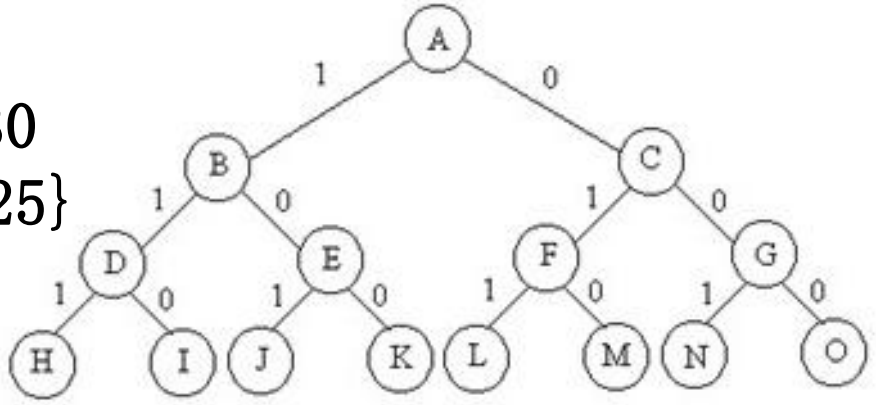


- 队列式分支限界法(处理法则:先进先出)：{}—>{A}—>{B,C}—>{C,D,E}(***D是不可行解，舍弃***)—>{C,E}—>{E,F,G}—>{F,G,J,K}(***J是不可行解，舍弃***)—>{F,G,K}—>{G,K,L,M}—>{K,L,M,N,O}—>{}

解空间广度搜索

分支限界法搜索应用举例

0-1背包问题，当n=3时， c=30
w={16, 15, 15}， p={45, 25, 25}



- 优先队列式分支限界法(处理法则:价值大者优先)：{}—>{A}—>{B(45),C(0)}—>{C(0)，D(x)，E(45)}—>{C(0)，E(45)}—>{C(0)，J(x)，K(45)}—>{C}—>{F(25)，G(0)}—>{G(0)，L(50)，M(25)}—>{G(0)，M(25)}—>{G}—>{N，O}—>{O}—>{}

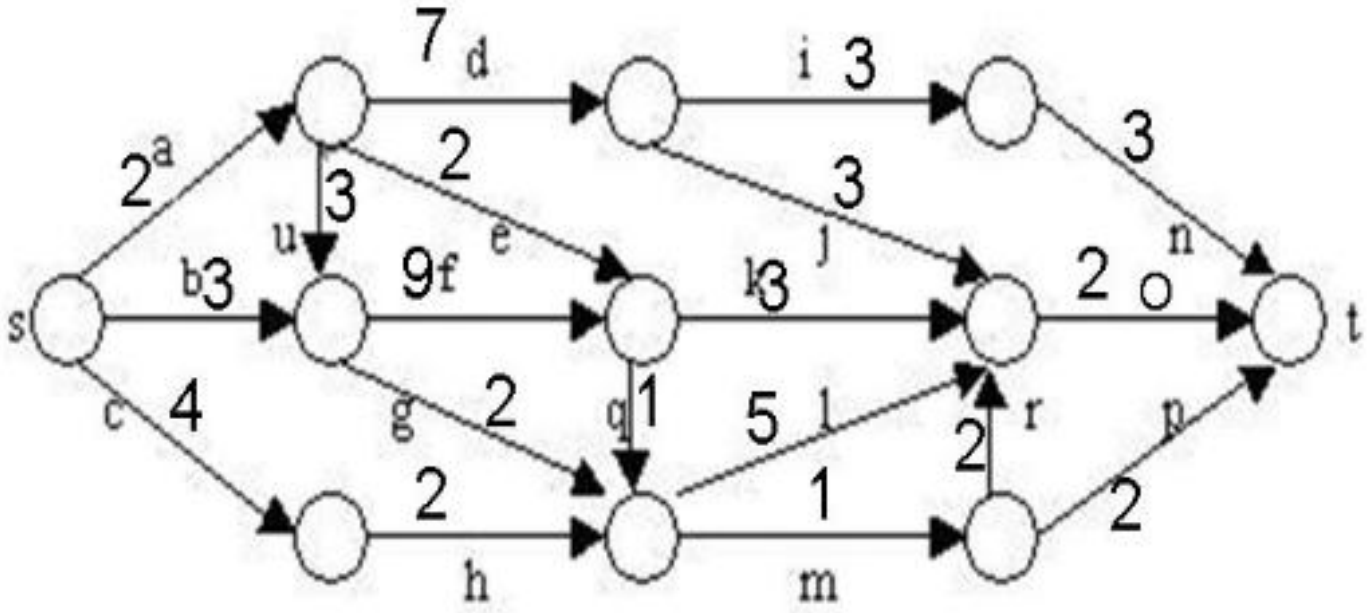
算法步骤

1. 定义解空间
2. 确定易于搜索的**解空间结构**（二叉树。。）
3. 确定结点的**数据结构**
4. 确定**约束条件**与**限界条件**用于剪枝
5. 确定活结点组织方法（队列/优先队列（**优先级函数**））
6. 确定答案节点识别方法

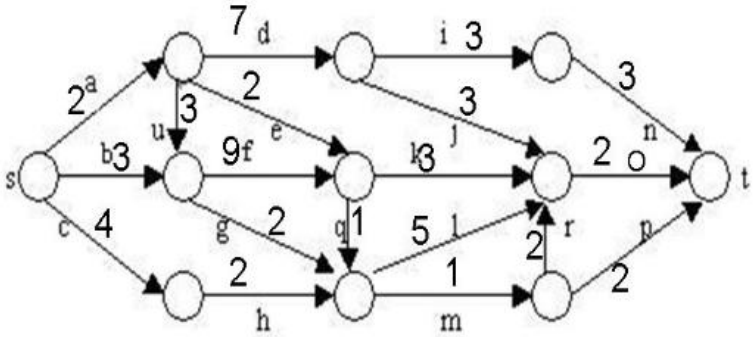
6.2 单源最短路径问题

1. 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权值。要求图G的从源顶点s到目标顶点t之间的最短路径。

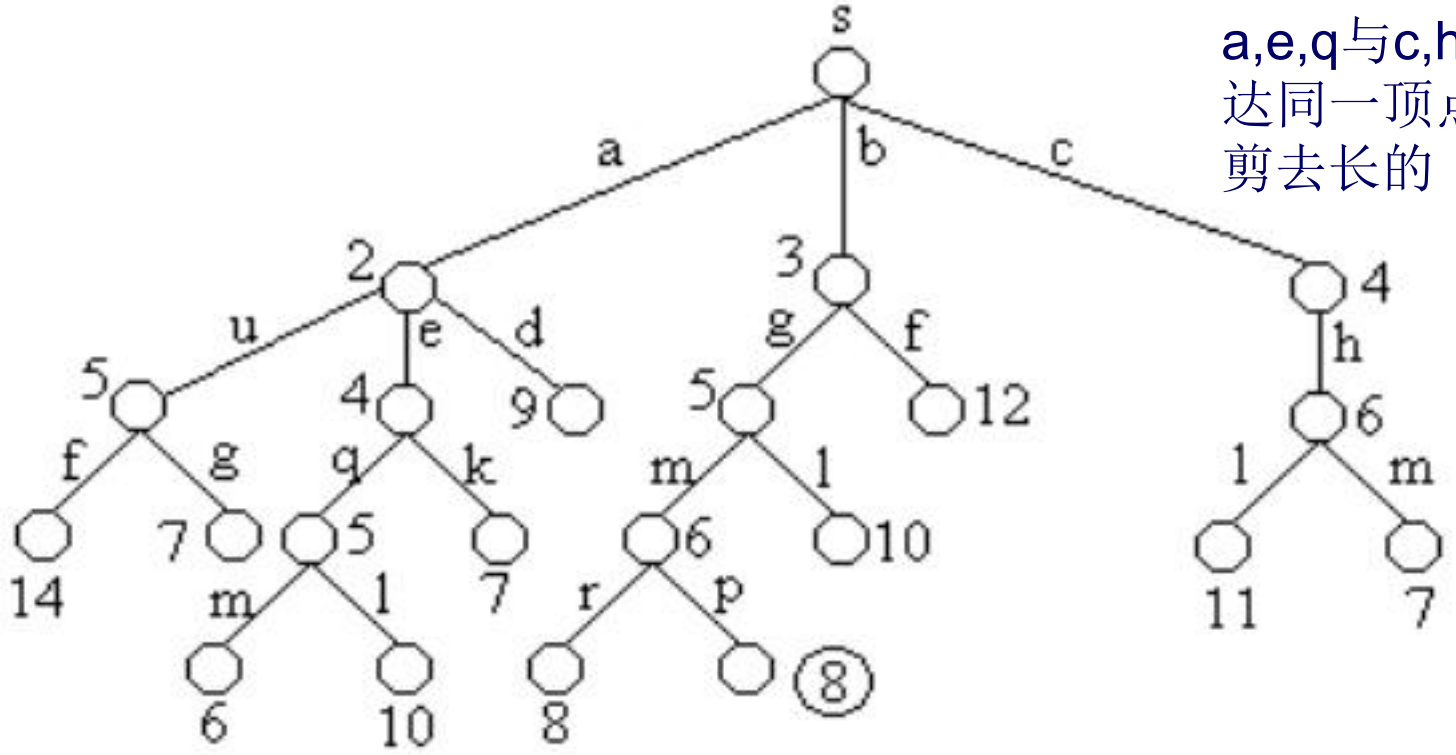


6.2 单源最短路径



1. 问题描述

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。



a,e,q与c,h到达同一顶点，剪去长的

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。

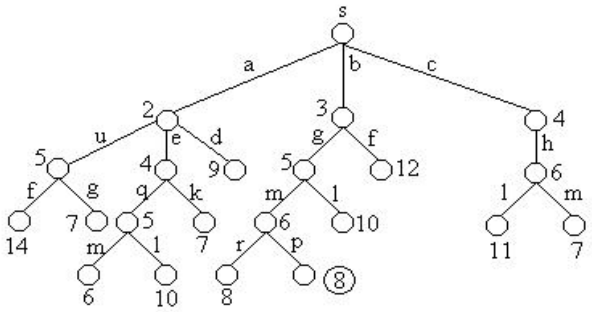
- 算法从图G的源顶点s和空优先队列开始。结点s被扩展后，它的儿子结点被依次插入堆中。此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所相应的路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。这个结点的扩展过程一直继续到活结点优先队列为空时为止。

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，因此可以将路长长的路径所对应的树中的结点为根的子树剪去。

6.2 单源最短路径问题



一次性生成所有儿子节点

```
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

6.3 装载问题

1. 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。

6.3 装载问题

2. 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。

6.3 装载问题

2. 队列式分支限界法

```
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      } }
```

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。则当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
 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;
        // 加入活结点队列
        if (i < n) Q.Add(wt);
}
```

提前更新bestw

```
// 检查右儿子结点
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
```

右儿子剪枝

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。

```
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
```

6.3 装载问题

4. 构造最优解

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 - 1; j > 0; j--) {
    bestx[j] = bestE->LChild;
    bestE = bestE->parent;
}
```

```
template<class Type>
void EnQueue(Queue<QNode<Type> *> &Q, Type wt, int i, int n, Type bestw, QNode<Type>*E,
             QNode<Type> * &bestE, int bestx[], bool ch) {       // 将活结点加入到活结点队列Q中
  if (i == n) {                              // 可行叶结点
    if (wt == bestw) {
      bestE = E;                             // 当前最优载重量
      bestx[n] = ch;
    }
    return;
  }
  // 非叶结点
  QNode<Type> *b;
  b = new QNode<Type>;
  b->weight = wt;
  b->parent = E;
  b->LChild = ch;
  Q.Add(b);
}
```

6.3 装载问题

5. 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。 （比剩余节点可能最好的还要好）

6.3 装载问题

树结点数据结构：

```
template <class Type>class HeapNode;
class bbnode {
   friend void AddLiveNode(MaxHeap<HeapNode<int>> &, bbnode *, int, bool, int);
   friend int MaxLoading(int*, int, int, int*);
   friend class AdjacencyGraph;
private:
   bbnode  *parent;                          // 指向父结点的指针
   bool  LChild;                             // 左儿子结点标志
};
```

6.3 装载问题

堆结点数据结构：

```
template<class Type>
class HeapNode {
   friend void AddLiveNode(MaxHeap<HeapNode<Type>> &, bbnode*, Type, bool, int);
   friend Type MaxLoading(Type*, Type, int, int*);
public:
   operator Type () const {  return uweight;  }
private:
   bbnode  *ptr;                          // 指向活结点在子集树中相应结点的指针
   Type  uweight;                         // 活结点优先级(上界)
   int  level;                            // 活结点在子集树中所处的层序号
};
```

6.3 装载问题

AddLiveNode:将新产生的节点加入优先队列中

```
template<class Type>
// 将活结点加入到表示活结点优先队列的最大堆H中
void AddLiveNode(MaxHeap<HeapNode<Type>>&H, bbnode *E, Type wt, bool ch, int lev) {
   bbnode  *b = new bbnode;
   b->parent = E;
   b->LChild = ch;
   HeapNode<Type> N;
   N.uweight = wt;
   N.level = lev;
   N.ptr = b;
   H.Insert(N);
}
```

6.3装载问题

MaxLoading:搜索

```
// 搜索子集空间树
while (i != n+1) {                                    // 非叶结点
   // 检查当前扩展结点的儿子结点
   if (Ew+w[i] <= c)                                  // 左儿子结点为可行结点
      AddLiveNode(H, E, Ew+w[i]+r[i], true, i+1);
   AddLiveNode(H, E, Ew+r[i], false, i+1);            // 右儿子结点
   HeapNode<Type> N;                                  // 取下一扩展结点
   H.DeleteMax(N);                                    // 非空
   i = N.level;
   E = N.ptr;
   Ew = N.uweight-r[i-1];
}
```

6.5 0-1背包问题

算法的思想

- 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。
- 在下面描述的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

- 用优先队列式分枝限界法解决0/1背包问题，要确定以下四个问题：
  - 解空间树中结点的结构；
  - 如何生成一个给定结点的儿子；
  - 如何组织活结点表；
  - 如何识别答案结点。

解空间树中结点的结构

解空间树中结点的结构

采用完全二叉树作为解空间树，每个结点包含：

- Parent：结点X的父亲结点连接指针；
- Level：结点X在解空间树中的级数，置$X_{Level(X)}=1$表示生成X的左儿子，置$X_{Level(X)}=0$表示生成X的右儿子；
- Cleft：记录背包在结点X处的可用空间(即剩余空间)，在确定X左儿子的可行性时用；
- Cp：记录在结点X处背包中已装物品的价值(或效益值)，Cp=$\sum_{1\le i<Level(X)} p_i x_i$ 。

如何生成儿子

- 检查当前扩展结点的左儿子结点的可行性。

  如果该左儿子结点满足约束条件，则它是可行的，将它加入到子集树和活结点优先队列中。

- 检查当前扩展结点的右儿子结点的可行性。

  右儿子一定是可行结点（满足约束条件），仅当右儿子结点满足限界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。

如何生成儿子

- 约束条件： $cw + w_k \leq C$，即$w_k \leq Cleft$
- 限界条件：在结点**X**所表示的状态下，可行解所能达到的可能值的上界用**bound**表示。也即是说，

  当 $x_1, \cdots, x_{l-1}$ 的值确定后，可行解

  $x_1, \cdots, x_{l-1}, x_l, \cdots x_n$ 所能达到的效益值的上界。
- 如果到目前为止所知道的解的最大效益值为**bestc**，当**bound(X)< bestc**时，就应该处死结点**X**。所以，**bound(X)**可以作为限界函数。

如何组织活结点表

即如何安排活结点的优先顺序。

定义的限界函数**bound(X)**为当前剩余最大单位价值的物品装满背包时的背包价值，也就是说**bound(X)**越大，该子树中产生最优解的可能越大，所以，**bound(X)**也是优先级函数。

每个活结点的优先顺序由**bound(X)**来确定

如何识别答案结点

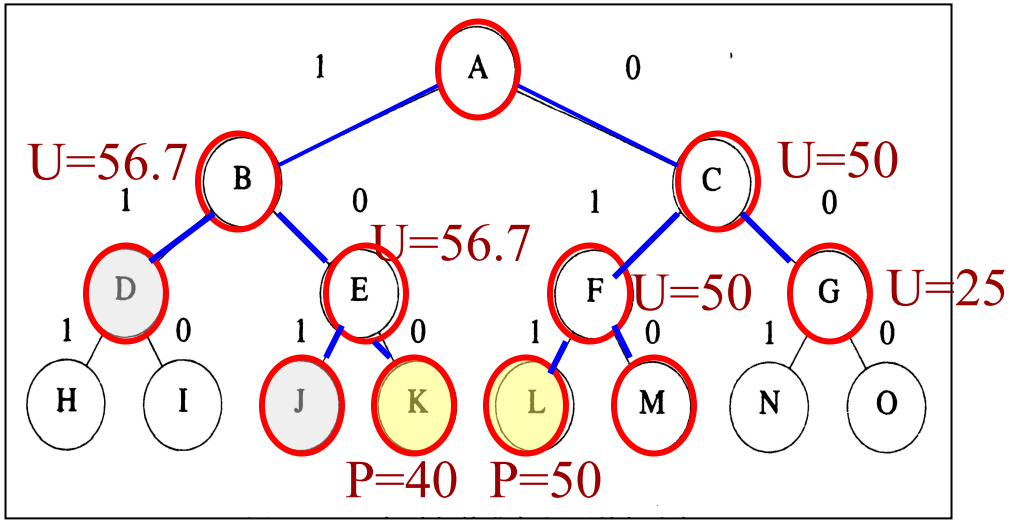
答案结点指使得背包价值最大的物品序列集合。

当搜索到的叶子结点的cp值大于所有活结点列表中结点的bound的时候或者活结点列表为空时找到最优值。将对应的X[]序列输出即为最优解。

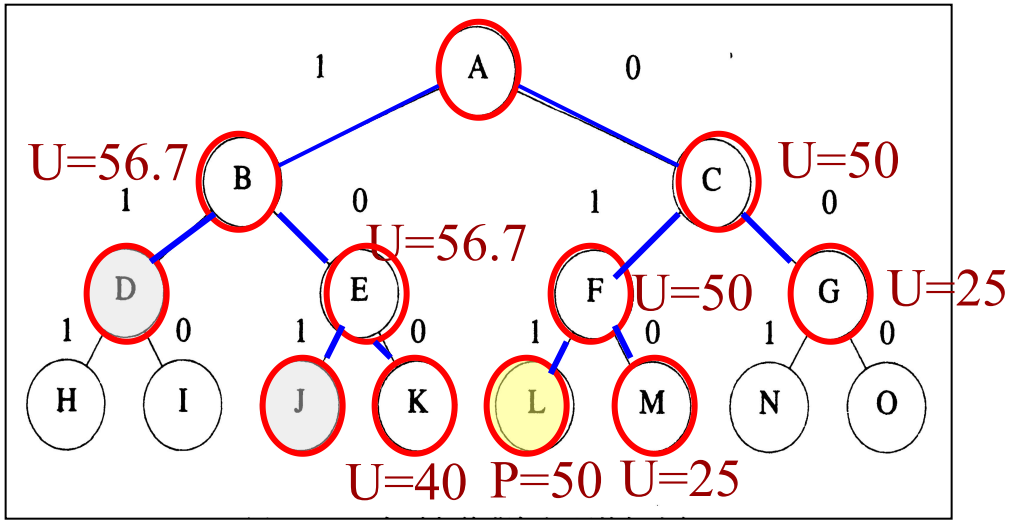
例题

设N=3, W=(20,15,15), P=(40,25,25), C=30

把Ubound=“当前背包价值量＋剩余物品按最大单位价值依次装入直至装满背包 ” 作为优先级。



例题：优先队列流程

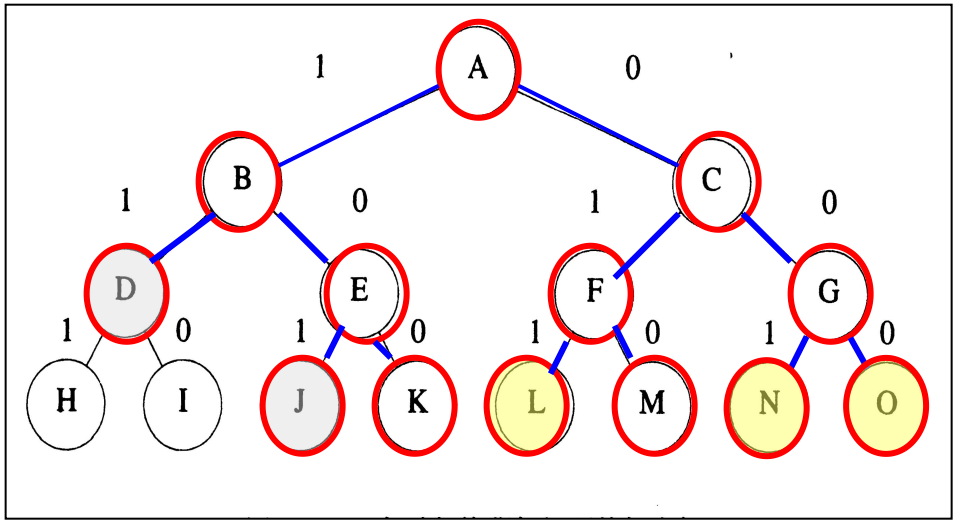


优先队列：

{A}->{B,C}->{E,C}->{C,K}->{F,K,G}->{L,K,G,M}->

L结点为叶子节点，最优

例题：队列流程



队列：运行到叶子结点时记录该节点的价值

{A}->{B,C}->{C,E}->{E,F,G}->{F,G,K}->{G,K,L,M}->{K,L,M,N,O}->{L,M,N,O}->{M,N,O}->{N,O}->{O}->最优L结点

0-1背包问题

算法描述：

初始化，将物品按照单位价值从大到小的顺序排列；

从第一个物品开始

左儿子加入后可否可满足控制约束

将顶点i加入到最大优先队列中

修改最大价值，修改上界条件

检查右儿子可否满足条件

将右儿子加入优先队列

修改上界条件

取下一扩展结点

当队列为空时

输出当前记录的路径

6.5 0-1背包问题

- 上界函数Bound

```
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
```

6.5 0-1背包问题

```
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）
}
```

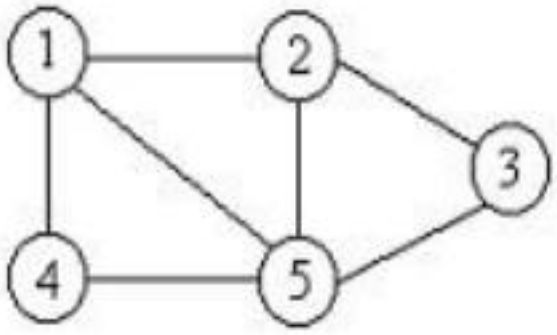
分支限界搜索过程

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。

- 下图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+1作为顶点数上界upperSize的值。

- 在此优先队列式分支限界法中，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。

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它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，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。

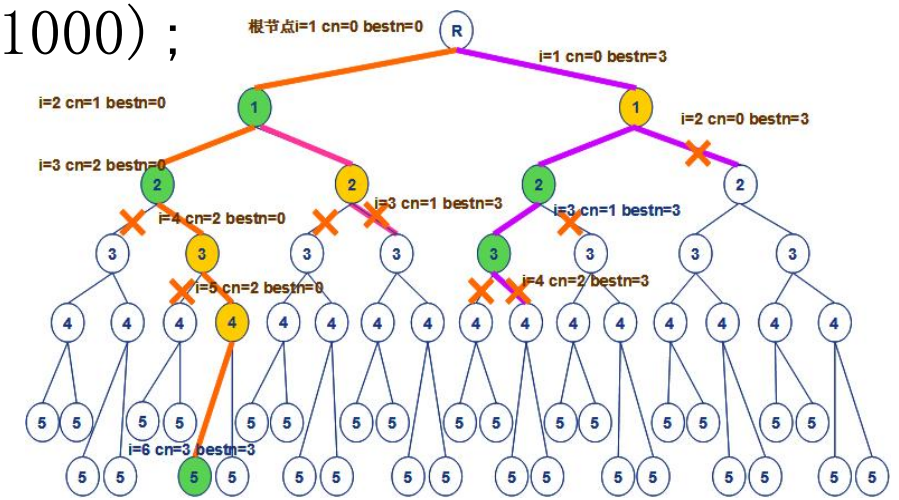
6.6 最大团问题

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。

```
//解最大团问题的优先队列式分支限界法
int Clique::BBMaxClique(int bestx[])  {
    MaxHeap<CliqueNode> H(1000);
    //初始化
    bbnode *E = 0;
    int i = 1,
    cn = 0,
    bestn = 0;

    //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;
            }
        }
```



```
    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;
}
```

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

    return bestn;
}
```

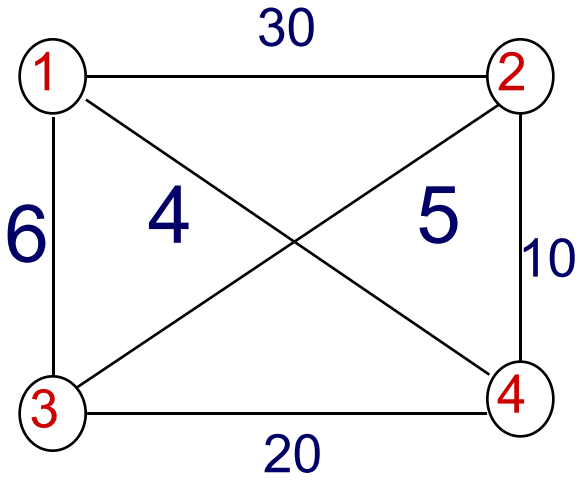
6.7 旅行售货员问题

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。图中各边的费用（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。周游路线的费用是这条路线上所有边的费用之和。
- 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。
- 这里可以用优先队列式实现方式。

TSP问题

优先队列式分支限界法



A:(-,-,-,-)

1 B:(1,-,-,-)

8 30 >25 C:(1,2,-,-)

3 6 D:(1,3,-,-)

2 4 E:(1,4,-,-)

35 F (1,2,3,-)

40 G(1,2,4,-)

4 11 H(1,3,2,-)

7 26 I(1,3,4,-)

5 14 J(1,4,2,-)

6 24 K(1,4,3,-)

55 L(1,2,3,4)

60 M(1,2,4,3)

21 N(1,3,2,4)

36 O(1,3,4,2)

19 P(1,4,2,3)

29 Q(1,4,3,2)

25

>25

25

>25

6.5 旅行售货员问题

- 算法开始时创建一个最小堆，用于表示活结点优先队列。(用一个最小优先队列来记录活节点)，队列中每个节点的类型为MinHeapNode。
- **每个结点中包括：**
  - x [0:n-1]：解的结构。
  - **s**：表示结点在排列树中的层次
  - cc：解空间树中从根节点到当前节点的耗费。
  - lcost：该节点子树的最小耗费下界。
  - rcost为x[s:n-1]中顶点最小出边费用和。图中每个顶点的最小出边费用用minout记录。
  - b＝cc＋rcost为当前结点的下界。

MinHeapNode

```
Class MinHeapNode{
    Type lcost,    //子树费用下界
    cc,            //当前费用
    rcost; //x[s:n-1]中顶点最小出边费用和
    int s, //根节点到当前结点的路径x[0:s]
    *x; //需要进一步搜索的顶点是x[s+1:n-1]
}
```

准备工作——计算MinOut

```
// 计算MinOut[i] = 顶点i的最小出边费用
   Type *MinOut = new Type [n+1];
    MinSum = 0; //最小出边费用和
    for (int i = 1; i <= n; i++) {
          Type Min = NoEdge;
          for (int j = 1; j <= n; j++)
                if (a[i][j] != NoEdge &&
                      (a[i][j] < Min || Min == NoEdge))
                        Min = a[i][j];
          if (Min == NoEdge) return NoEdge; // 此路不通
          MinOut[i] = Min;
          MinSum += Min;       }
```

准备工作——初始化

```
// 把E-节点初始化为树根
  MinHeapNode E;
  E.x = new int [n];
  for (i = 0; i < n; i++)
        E.x[i] = i + 1;
  E.s = 0; //根节点到当前节点的路径
  E.cc = 0; // 当前费用为0
  E.rcost = MinSum; //顶点最小出边费用和，下界
  bestc = NoEdge; // 目前没有找到旅行路径
```

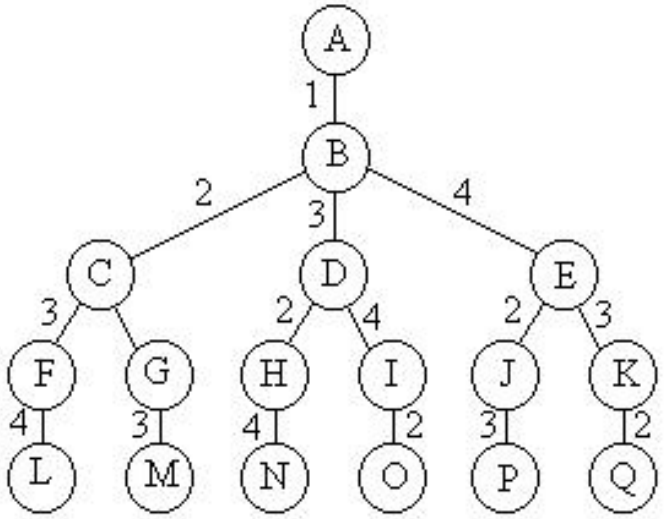
```
Class MinHeapNode{
    Type lcost,   //子树费用下界
    cc,         //当前费用
    rcost; //x[s:n-1]中顶点最小出边费用和
    int s, //根节点到当前结点的路径x[0:s]
    *x; //需要进一步搜索的顶点是x[s+1:n-1]
}
```

6.7 旅行售货员问题

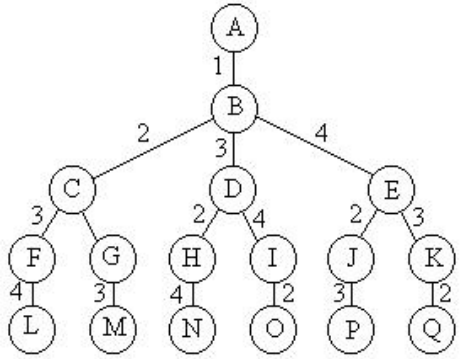
2. 算法描述

- 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。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。

- 算法在搜索排序空间树的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：



6.7 旅行售货员问题



2. 算法描述-搜索到树中某中间结点

- 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。如果该叶结点对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。

- 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。

算法

```
//s = n-2 时:
while (E.s < n - 1) {//搜索排列树,不是叶子
    if (E.s == n - 2) {// 叶子的父节点，再添加两条边即完成
        // 检查新的旅行路径是不是更好
       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))
        // 找到更优的旅行路径
        {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;
    } //舍弃扩展结点
```

算法

```
//s<n-2时:
else {// 产生孩子
        for (int i = E.s + 1; i < n; i++)
            if (a[E.x[E.s]][E.x[i]] != NoEdge) {
                // 可行的孩子, 限定了路径的耗费
                cc = E.cc + a[E.x[E.s]][E.x[i]];
                rcost = E.rcost - MinOut[E.x[E.s]]; //剩下节点的
                b = cc + rcost; //下限
               if (b < bestc || bestc == NoEdge) {
              // 子树可能有更好的叶子
              // 把根保存到堆中
```

算法

```
        MinHeapNode N;
        N.x = new int [n]; //新建立堆节点
        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) ;
    }
    } // 结束可行的孩子，释放空间
        delete [] E.x;
    } // 对本节点的处理结束
        try {H.DeleteMin(E);} // 取下一个E-节点
        catch (OutOfBounds) {break;} // 没有未处理的节点
        }
```

6.7 旅行售货员问题

2. 算法描述

- 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。

结果处理

```
if (bestc == NoEdge) return NoEdge; // 没有旅行路径
// 将最优路径复制到v[1:n] 中
for (i = 0; i < n; i++)
    v[i+1] = E.x[i];
while (true) {//释放最小堆中的所有节点
    delete [] E.x;
    try {H.DeleteMin(E);}
    catch (OutOfBounds) {break;}
    }
    return bestc;
```

6.9 批处理作业调度问题

t_{ji}	机器1	机器2
作业1	2（2）	1（3）
作业2	3（7）	1（8）
作业3	2（4）	3（7）

1. 问题的描述

- 给定n个作业的集合$J=\{J_1, J_2, \ldots, J_n\}$。每一个作业$J_i$都有两项任务要分别在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J_i需要机器j的处理时间为t_{ji}，$i=1, 2, \ldots, n$；$j=1, 2$。对于一个确定的作业调度，设是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。

2. 限界函数

在结点E处相应子树中叶结点完成时间和

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

机器1没有空闲时剩余任务的总完成时间和

剩余任务完成时间和的下界是：

$$\sum_{i \notin M} F_{2i} \geq \max\{S_1, S_2\}$$

机器2没有空闲时剩余任务的总完成时间和

注意到如果选择pk，使t_{1pk}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Pk使t_{2pk}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。

6.9 批处理作业调度问题

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

- 当E.s<n时，算法依次产生当前扩展结点E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。

6.9 批处理作业调度问题

3. 算法描述

```
while (E.s <= n ) {
 if (E.s == n ) {// 叶结点
   if (E.sf2 < bestc) {
     bestc = E.sf2;
     for (int i = 0; i < n; i++)
       bestx[i] = E.x[i];}
   delete [] E.x;}
```

当E.sf2<bestc时，更新当前最优值bestc和相应的最优解bestx

6.9 批处理作业调度问题

3. 算法描述

```
else {// 产生当前扩展结点的儿子结点
        for (int i = E.s; i < n; i++) {
            Swap(E.x[E.s],E.x[i]);
            int f1,f2;
            int bb=Bound(E,f1,f2,y);
            if (bb < bestc ) {
                MinHeapNode N;
                N.NewNode(E,f1,f2,bb,n);
                H.Insert(N);}
            Swap(E.x[E.s],E.x[i]);
            }
        delete [] E.x;}  // 完成结点扩展
```

当bb<bestc时，将儿子结点插入到活结点优先队列中

小 结

- 分支限界法是在解空间树上的广度优先搜索算法。
- 分为队列式和优先队列式两种分支限界法。
- 和回溯法解决问题的类型相似，但解决问题的方法不同。