第七章 分枝限界法

算法基本思想

0/1背包问题

电路板布线问题

LC－搜索

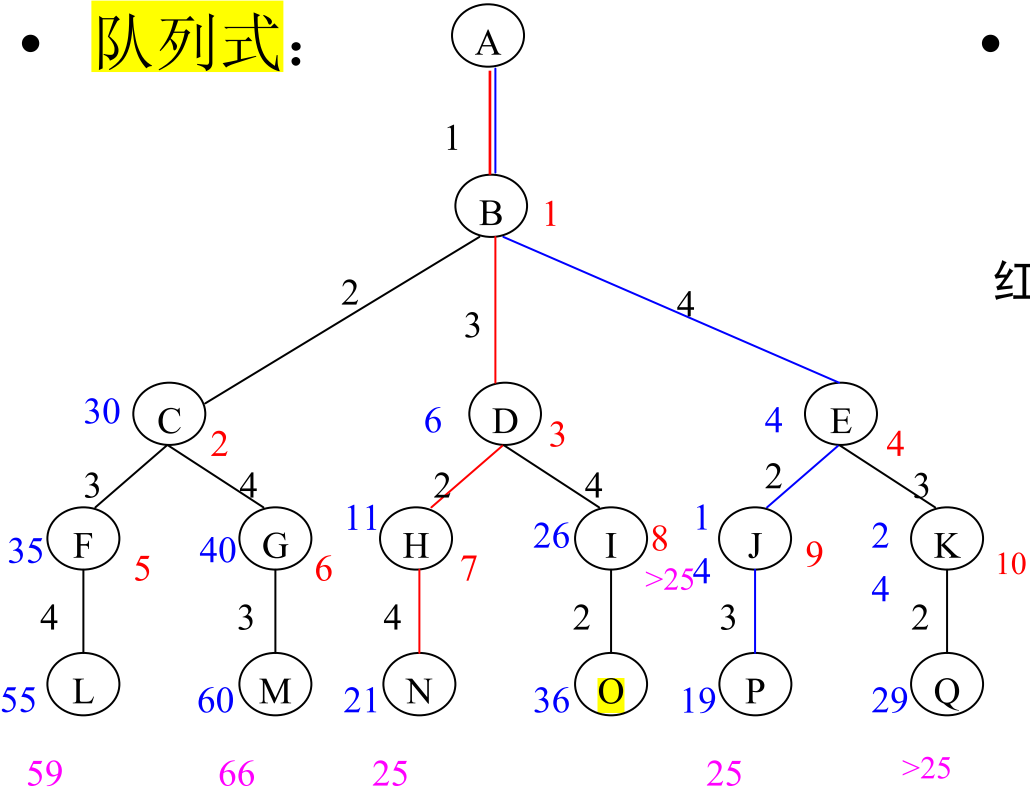
旅行商问题

算法的基本思想

- 在解空间中搜索，生成状态空间树；
- 采用宽度优先搜索，用表记录活节点
- 在扩展节点处，首先生成其所有的儿子节点,将那些导致不可行解或导致非最优解的儿子节点舍弃，其余儿子节点加入活节点表中。然后，从活节点表中取出一个节点作为当前扩展节点，重复上述节点扩展过程。
- **队列式分枝限界算法**：将活节点组织成先进先出（FIFO）或后进先出(LIFO)队列，不满足约束条件的节点不放入队列。
- **优先队列式分枝限界算法**：将活节点组织成最大堆（或最小堆），优先级高的首先取作当前扩展节点。
- 节点的优先级常常根据目标函数确定，最大化问题常引用一个可能获得的最大目标值的一个上界；最小化问题则使用可能获得最小目标值的一个下界。这两个界都是动态确定的。
- 达到最小成本搜索是确定节点优先级的根本目的。

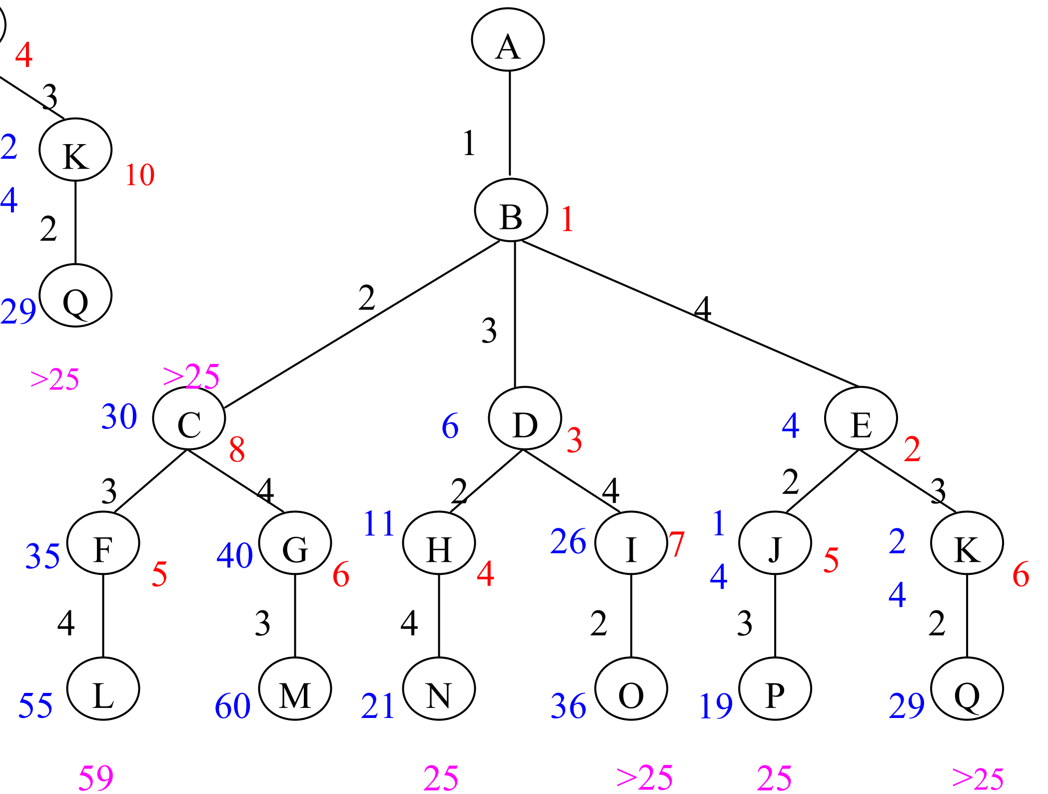
旅行商问题的两种分枝定界算法

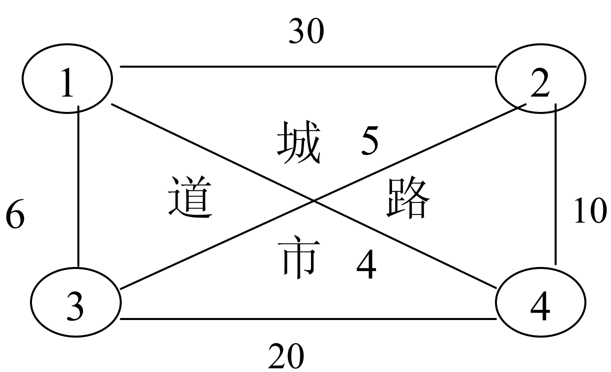
- 队列式：



- 优先队列式：当前路径长度短的节点优先级高。

蓝色表示该节点的费用
红色表示该结点成为扩展节点的序号
粉色表示相应周游的费用





0/1背包问题的优先队列式分枝定界算法

用优先队列式分枝定界法解0/1背包问题需要确定：

1) 解空间树中节点的结构；

2) 如何生成一个给定节点的儿子节点；

3) 如何组织活节点表；

4) 如何识别答案节点。

每个节点X有六个信息段：

Parent：节点X的父亲节点连接指针；

Level：标志出节点X在解空间树中的深度；

Tag：用来标记输出最优解的各个分量x_i；

CC ：记录背包在节点X处(状态下)背包的剩余空间；

CV：记录背包在节点X处(状态下)背包内物品的价值；

CUB：背包在节点X处可能达到的物品价值上界估值 Puv。

目标值动态预测prev：到目前为止所知道的最佳目标值。

0/1背包问题的优先队列式分枝定界算法(2)

- 活节点表的组织：采用优先队列

  信息段CUB中的值做为确定该节点优先级的依据；

  如果Puv(X)≤prev，则杀死节点X，即X不放入节点表。

- 六个辅助子程序

  LUBound：计算当前被搜索节点的Pvl和Pvu值；

  NewNode：生成新节点，给各个信息段置入适当的值，并将此节点加入节点表；

  GetNode：取一个可用节点；

  Init：对可用节点表和活节点表置初值；

  Largest：在活节点表中取一个具有最大Pvu值节点作为当前扩展节点；

  Finish：打印出最优解的值和此最优解中的物品标号。

0/1背包问题的优先队列式分枝定界算法(3)

```
proc LCKNAP(P,W,M,N)//物品序号
  //满足： P[i]/W[i]≥P[i+1]/W[i+1]；
  real M, Pvl, Pvu, cap, cv, prev ;
  real P[1..N],W[1..N];
  integer ANS，X，N；
  Init；//初始化可用节点及活节点表
  GetNode(E)；//生成根节点
  Parent(E):=0;  Level(E):=0;
  CC(E):=M; CV(E)=0;
  LUBound(P,W,M,0,N,1,Pvl,Pvu);
  prev:=Pvl; CUB(E):=Pvu;
  Tag(E):=0;
  loop
    i:=Level(E)+1, cap:=CC(E),
    cv:=CV(E);
    case:
       i=N+1: //解节点
         if cv > prev then
•            prev:=cv; ANS:=E;
•          end{if}
•        else: //E是内部节点,有两个儿子
•          if cap≥W[i] then //左儿子可行
•            NewNode(E,i,1,cap-W[i],
•                         cv+P[i],CUB(E));
•          end{if}
•          LUBound(P,W,cap,cv,N,i+1,
•                         Pvl,Pvu);
•          if Pvu>prev then //右儿子会活
•            NewNode(E,i,0,cap,cv,Pvu);
•            prev:=max(prev,Pvl-ε);
•          end{if}
•      end{case}
•      if 不再有活节点 then exit; end{if}
•      Largest(E);//取下一个扩展节点
•    until CUB(E)≤prev
•    Finish(cv,ANS,N);
end{LCKNAP}
```

生成新节点与解的输出

- 程序生成新节点算法

NewNode(par,lev,t,cap,cv
,ub)
//生成一个新节点J，并
//把它加到活节点表
GetNode(J);
Parent(J):=par;
Level(J):=lev;
Tag(J):=t;
CC(J):=cap;
CV(J):=cv;
CUB(J):=ub;
Add(J);
end{NewNode}

- 打印答案程序

Finish(CV,ANS,N)//输出解
- **real** CV;
- **global** Tag,Parent;
- print('OBJECTS IN
- KNAPSACK ARE')
- **for** j **from** N **by** –1 **to** 1 **do**
- **if** Tag(ANS)=1 **then**
- print(j);
- **end{if}**
- ANS:=Parent(ANS);
- **end{for}**

end{Finish}

计算当前状态下的可能取得最大效益值的上、下界

LUBound(P,W,cap,cv,N,k,Pvl,Pvu) // k为当前节点的级（level），cap是背包当前

- //的剩余容量，cv是当前背包中物品的总价值，还有物品k,…,N要考虑
- Real rw; //随时记录本函数执行过程中背包的剩余容量
- Pvl:=cv; rw:=cap;
- **for** i **from** k **to** N **do**
-   **if** rw<W[i] **then** Pvu:=Pvl+rw*P[i]/W[i];
-     // 从第k件到第N件至少有一件物品不能装进背包的情形出现
-     **for** j **from** i+1 **to** N **do**
-       **if** rw≥W[j] **then**
-         rw:=rw-W[j]; Pvl:=Pvl+P[j];
-       **end{if}**
-     **end{for}**
-     **return** //此时Pvl < Pvu
-   **end{if}**
-   rw:=rw-W[i]; Pvl:=Pvl+P[i];
- **end{for}**
- Pvu:=Pvl; // 从第k件物品到第N件物品都能装进背包的情形出现,

end{LUBound}

0/1背包问题的优先队列式分枝定界算法(4)

- 例子 n=4,

P=(10,10,12,18),

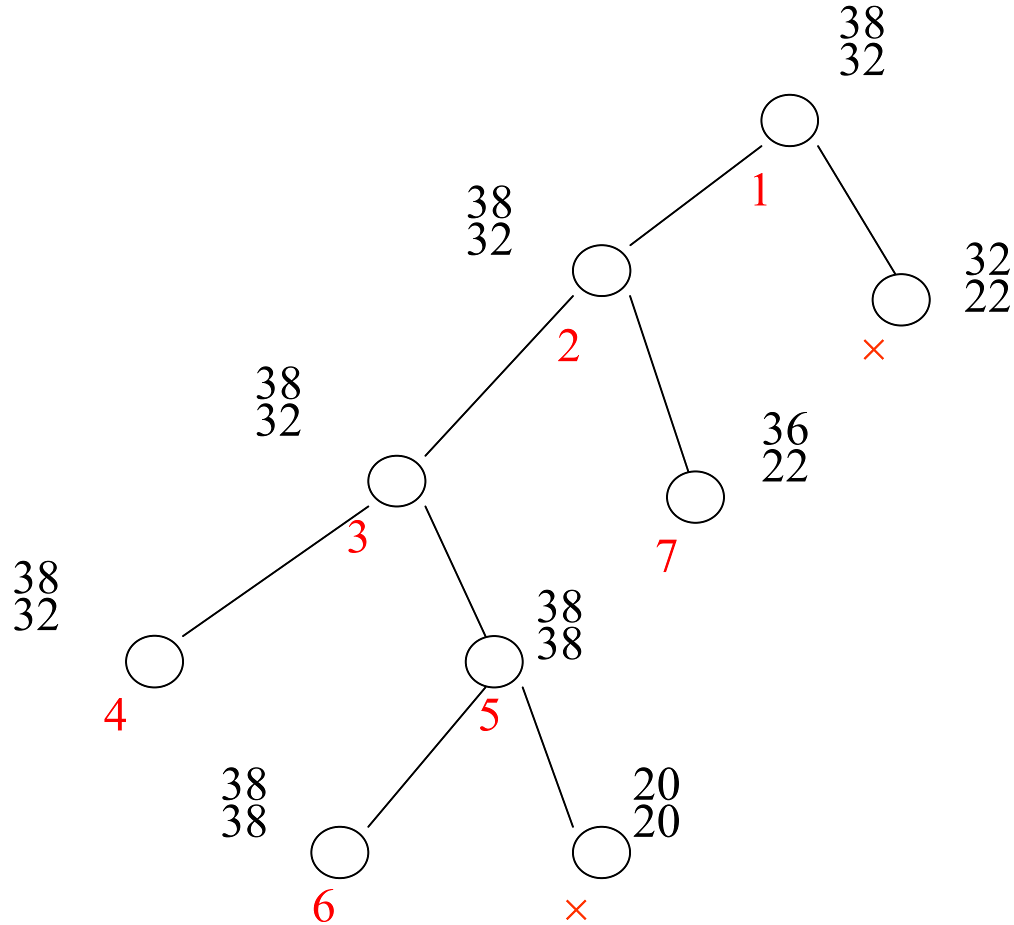
W=(2,4,6,9),

M=15.

试绘出算法

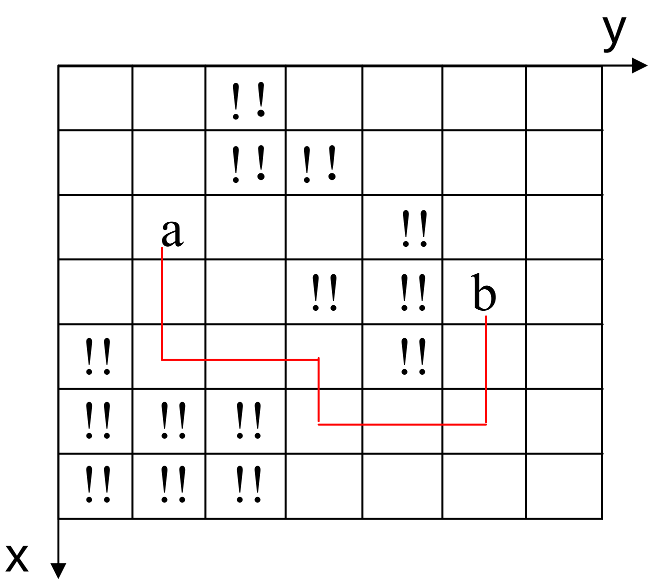
LCKNAP

求最优解的检索过程。



电路板布线问题

- 问题：印刷电路板将布线区域分成n×m个方格(阵列)，某些方格有禁入标记。确定连接两个指定方格a和b间的最短折线布置方案。



3	2	!!				
2	1	!!	!!			
1	a	1	2	!!		
2	1	2	!!	!!	b	
!!	2	3	4	!!	8	9
!!	!!	!!	5	6	7	8
!!	!!	!!	6	7	8	

- 方格位置的类Position：私有成员 row, col；如a=(3,2),b=(4,6)
- 平移offset：右(0)、下(1)、左(2)、上(3)，如向右平移一步表示为：offset[0].row=0 and offset[0].col=1。
- 方格状态： grid[i][j]=0可以通过， grid[i][j]=1禁止通过。

布线问题的队列式分枝限界算法(1)

```
bool FindPath(Position start,
              Position finish,
              int& PathLen,
              Position * &path)
{//计算从起点位置start到目标位置
 //finish的最短布线路径.找到最短布
 //线路径则返回true,否则返回false
 if((start.row= =finish.row) &&
              (start.col= =finish.col)
 {PathLen=0; return true;}
   //start=finish
   //设置方格阵列"围墙"
 for(int i=0; i<= m+1; i++)
   grid[0][i]=grid[n+1][i]=1;
    //顶部和底部
 for(int i=0; i<= n+1; i++)
   grid[i][0]=grid[i][m+1]=1;
   //左翼和右翼
```

- //初始化相对位移
- Position offset[4];
- offset[0].row=0; offset[0].col=1;//右
- offset[1].row=1; offset[1].col=0;//下
- offset[2].row=0; offset[2].col=-1;//左
- offset[3].row=-1; offset[3].col=0;//上
- int NumOfNbrs=4;//相邻方格数
- Position here,nbr;
- here.row=start.row;
- here.col=start.col;
- grid[start.row][start.col]=2;
- //标记可达方格位置
- LinkedQueue<Position> Q;
- do {//标记相邻可达方格
- for(int i=0; i<NumOfNbrs; i++){
- nbr.row=here.row + offset[i].row;
- nbr.col=here.col+offset[i].col;

布线问题的队列式分枝限界算法(2)

- if(grid[nbr.row][nbr.col]= =0){
- //该方格未被标记
- grid[nbr.row][nbr.col]
- =grid[here.row][here.col]+1;
- if((nbr.row= =finish.row) &&
- (nbr.col= =finish.col)) break;
- //完成布线
- Q.Add(nbr);}
- }
- //是否到达目标位置finish？
- if((nbr.row= =finish.row) &&
- (nbr.col= =finish.col)) break;
- //活结点队列是否非空？
- if(Q.IsEmpty()) return false;//无解
- Q.Delete(here);//取下个扩展结点
- }while(true);
- //构造最短布线路径
- PathLen=grid[finish.row][finish.col]-2;
- path=new Position[PathLen];
- //从目标位置finish开始向起始位置
- //回溯
- here=finish;
- for(int j=PathLen-1; j>=0; j--){
- path[j]=here; //找前驱位置
- for(int i=0; i<NumOfNbrs; i++){
- nbr.row=here.row+offset[i].row;
- nbr.col=here.col+offset[i].col;
- if(grid[nbr.row][nbr.col]==j+2)
- break;
- }
- here=nbr;//向前移动

}

return true;

}

最小成本搜索

- **优先队列式分枝定界算法优先级函数的确定**

  理想的当前扩展节点X

  1) 以X为根的子树中含有问题的答案节点；

  2) 在所有满足条件1)的活节点中，X距离答案节点“最近”。

  节点计算代价的度量

  (i) 在生成一个答案节点之前，子树X需要生成的节点数；

  (ii) 以X为根的子树中，离X最近的那个答案节点到X的路径长度。

  理想的优先级函数c(.)

  a) 如果X是答案节点，则c(X)是解空间树中由根节点到X的成本(即所用的代价，如深度、计算复杂度等)；

  b) 如果X不是答案节点，而且以X为根的子树中不含答案节点，则c(X)定义为∞；

  c) 如果X不是答案节点，但是以X为根的子树中含答案节点，则c(X)是具有最小成本的答案节点的成本。

  优先级估计函数 $\hat{c}(.) = f(X) + g(X)$

  f(X)－－解空间树根节点到X的成本；g(X)－－X到答案节点的计算成本。

- **最小成本搜索**

  根据成本估计函数选择下一个扩展节点的策略总是选取$\hat{c}(\cdot)$值最小的活节点作为下一个扩展节点。

十五谜团问题

- **问题**

在一个分成4×4的棋盘上排列15块号牌，其中会出现一个空格。棋盘上号牌的一次合法移动是指将与空格相邻的一块号牌移入空格。15迷问题要求通过一系列合法移动，将号牌的初始排列转换成自然排列。

1	3	4	15
2	5	12	
7	6	11	14
8	9	10	13

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

	#		#
#		#	
	#		#
#		#	

空格在#号位 t=1 否则 t=0

- 16个数字的排列：P=(1,3,4,15,2,16,5,12,7,6,11,14,8,9, 10,13) ，逆序数为

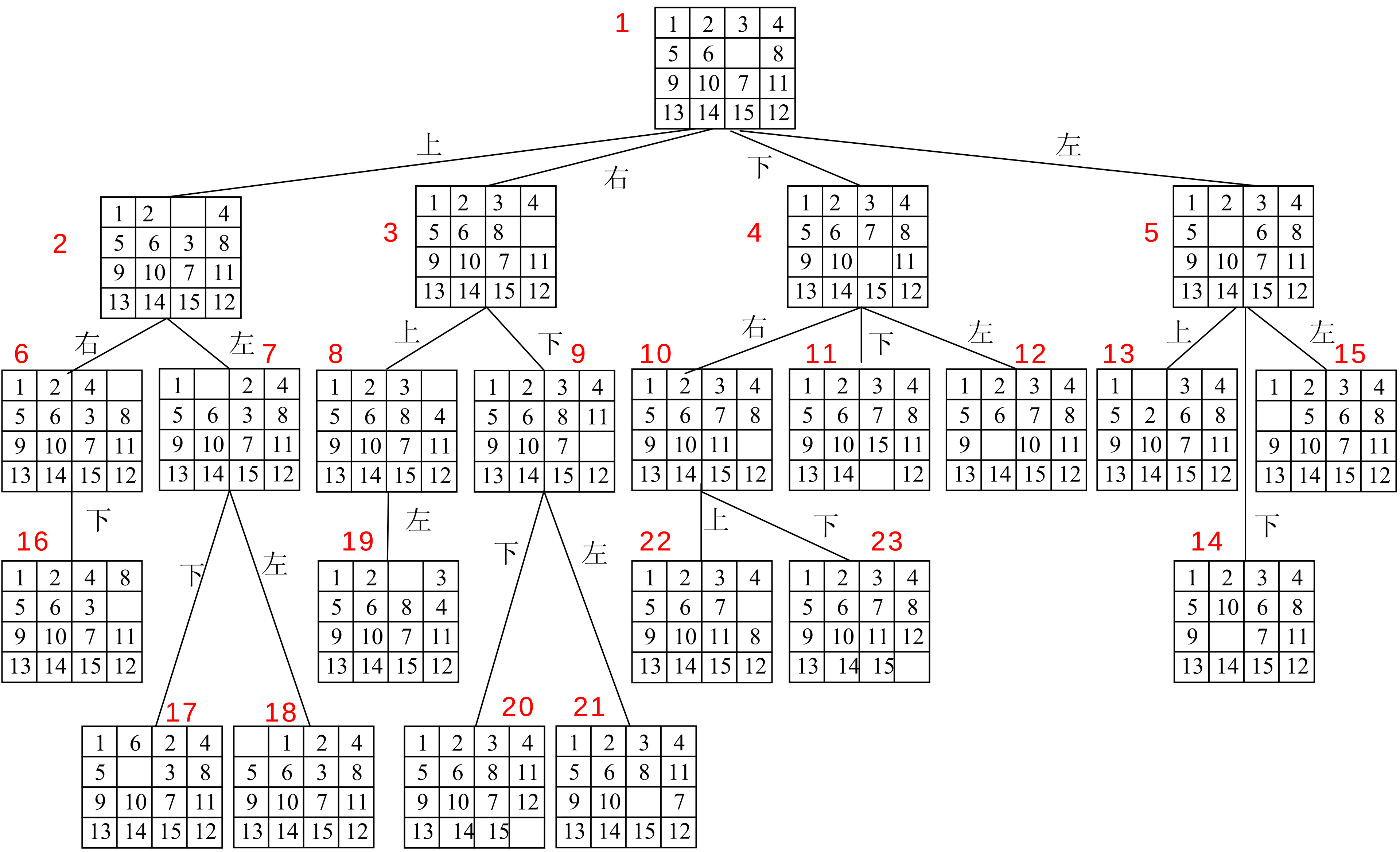
$$\tau(P) = \sum_{1\le i\le 16} \text{Less}(i), \quad \tau(P) + t \text{ 需是偶数,}$$

Less(i)是排列P中位于i后面且号码比i小的数的个数。

- **优先级函数**： $\hat{c}(.) = f(X) + g(X)$

f(x)是由根到节点X的路径的长度；g(X)=排列X的不在自然位置的号牌数目

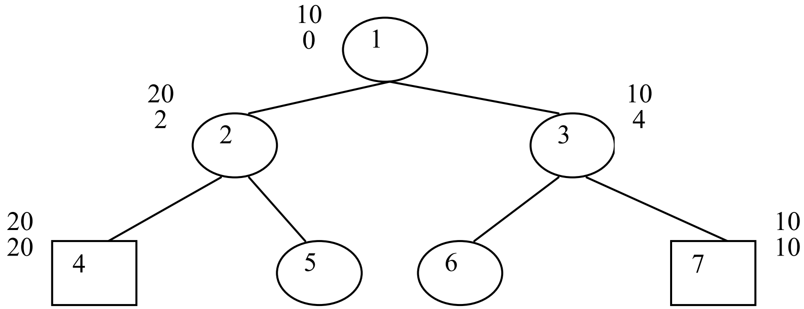
LC—搜索过程



1
上
右
下
左
2
3
4
5
右
左
上
下
右
下
左
上
左
6
7
8
9
10
11
12
13
15
下
下
左
左
下
左
上
下
下
16
19
22
23
14
17
18
20
21

成本估计函数应满足的条件

- 按照成本估价函数$\hat{c}(X)$确定的优先级进行搜索，所得到的答案节点未必是最小成本答案节点。



- **定理** 7.4.1 在有限的解空间树中，如果对每对节点X和Y都有
- “$c(X) < c(Y)$” $\Rightarrow$ “$\hat{c}(X) < \hat{c}(Y)$”
- 则按照最小成本估计函数搜索能够达到最小成本答案节点。

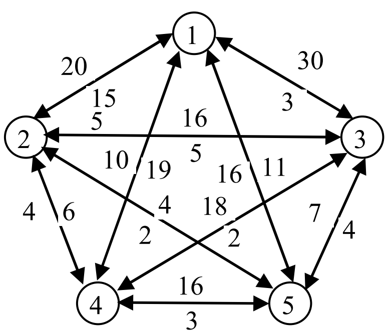
- 一般情况下，对于成本估计函数有一个基本要求：
- $\hat{c}(X) \le c(X)$，对任意节点X；
- $\hat{c}(X) = c(X)$，当X是答案节点时。

最小化问题的LC一分枝限界算法

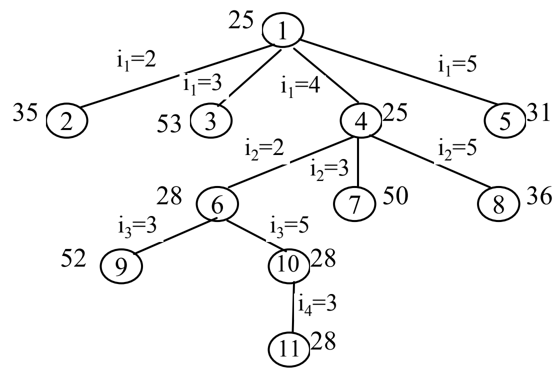
```
proc LCBB(T, ĉ,u,ε,cost) //假定解空间
  //树T包含一个解节点且
  // ĉ(X) ≤ c(X) ≤ u(X)。c(X)是最小
  //成本函数，ĉ(X)是成本估价函数
  //u(X)是限界函数;
  // cost(X)是X所对应的解的成本。ε
  //是一个充分小的正数。
  E:=T; Parent(E):=0;
  if T 是解节点 then
    U:=min(cost(T),u(T)+ ε); ans:=T;
  else U:=u(T)+ε; ans:=0;
  end{if}
  //将活节点表初始化为空集;
  loop
    for E 的每个儿子X do
      if ĉ(X)<U && X是一个可行节点
      then
        Add(X); Parent(X):=E;
        case:
          X是解节点 && cost(X)< U:
            U:=min(cost(X),u(X)+ε);
            ans:=X;
          u(X)+ε < U:
            U:=u(X)+ε;
        end{case}
      end{if}
    end{for}
    if 不再有活节点 or 下一个扩展
      节点满足 ĉ ≥ U then
      print('least cost=',U);
      while ans≠0 do
        print(ans); ans:=Parent(ans);
      end{while}
      return;
    end{if}
    Least(E);
  end{loop}
end{LCBB}
```

旅行商问题的LC-分枝限界算法

- 具有5个城市的旅行商问题



由小到大权值标于外侧，由大到小标于内侧



节点外面的数字是$\hat{c}$ 值

$$A=\begin{pmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

$$A(1)=\begin{pmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{pmatrix}$$

- 优先级函数

- $C(X)=\begin{cases}\text{从根到X的路径所定义的回路成本，当X是叶节点时；}\\ \text{子树X中一个最小成本叶节点的成本，当X不是也节点时。}\end{cases}$
- 定义$\hat{c}(X)$为根节点到节点X的路径的成本，显然有$\hat{c}(X) \le C(X)$。
- 简约矩阵：各行、列都至少有一个元素是零的非负矩阵。

$$\hat{c} = \sum_{i=1}^{n} r_i + \sum_{j=1}^{n} c_j$$

- (R,S)对应Hamilton回路中包含的边(i,j)。如果S不是叶节点，S的简约矩阵A_S可以通过修简R的简约矩阵AR而得：(1)将A_R中i行和j列的所有项都改为∞，防止任何其它离开顶点i的边，进入顶点j的边的使用。(2)将A_R的(j,1)元素置为∞，防止使用边(j,1)。(3)约简经过(1)、(2)两步操作后得到的矩阵。给出节点S的下界估值如下：
- $$\hat{c}(S) = \hat{c}(R) + A_R(i,j) + \tilde{c}$$
- 其中，$A_R(i,j)$ 是矩阵A_R的(i,j)元素，$\tilde{c}$是约简步骤(3)施行时减掉的总数。如果S是叶节点，则直接计算 $\hat{c}(S) = c(S)$ 即可，不必计算伴随矩阵。
- 每个上界估值可用u(x)=∞

- 伴随矩阵

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{pmatrix}$$

- A(2),路径 1,2　　A(3),路径 1,3　　A(4),路径 1,4

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{pmatrix}$$

- A(5),路径 1,5　　A(6), 路径 1,4,2　　A(7),路径 1,4,3

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{pmatrix} \quad \begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{pmatrix}$$

- A(8),路径 1,4,5　　A(9) 路径 1,4,2,3　　A(10),路径 1,4,2,5