5.4宽度优先搜索

- 思想（给定图G=(V，E)，它的初始状态是所有顶点均未被访问过，在图G中任选一个顶点v作为源点）
  - 先访问顶点v，并将其标记为已访问过；然后从v出发，依次访问v的邻接点（孩子结点）w1,w2,…,wt, 如果wi(i=1,2,…,t)未访问过，则标记wi为已访问过，将其插入到队列中；然后再依次从队列中取出w1,w2,…,wt，访问它们的邻接点。依此类推，直到图中所有和源点v有路径相通的顶点均已访问过为止；若此时图G中仍然存在未被访问过的顶点，则另选一个尚未访问过的顶点作为新的源点。重复上述过程，直到图中所有顶点均已访问过为止。

- 示例
  - 给定一个有向图，如图5-26所示，给出宽度优先搜索的一个序列。

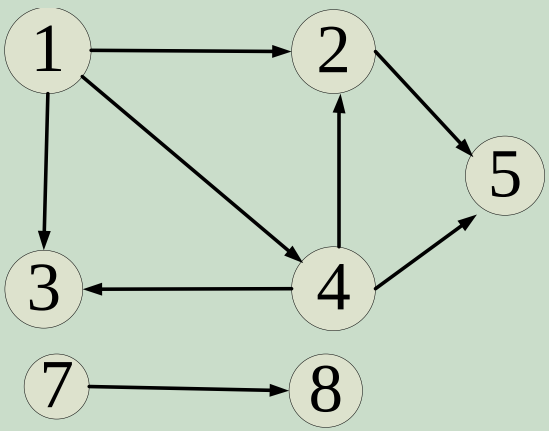


图5-26 有向图

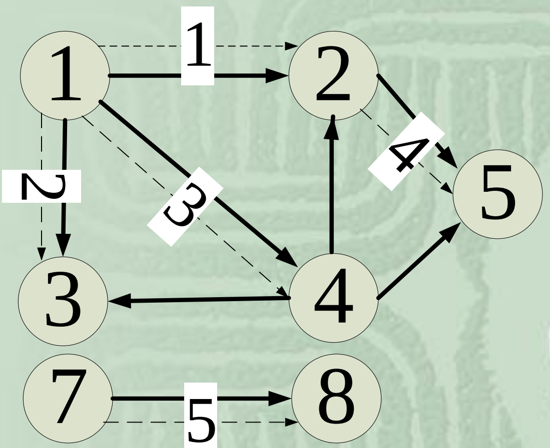


图5-27 搜索顺序

- 搜索顺序如图5-27所示
- 搜索序列为：1234578

算法描述

```
bool Visited[n+1];  //标记图中顶点有未被访问过
for(int i=1;i<=n;i++)  Visited[i]=0;  //所有顶点未被访问过
InitQueue(&Q);          //初始化空队

void  BFSV0 (int v0)     //从v0开始广度优先搜索所在的连通子图
{    visit(v0);   Visited[v0]=1;
     InsertQueue(&Q,v0);   // v0进队
     while (! Empty(Q))
     {     DeleteQueue(&Q, &v);  //队头元素出队
           for(int i=1;i<=n;i++)       //依次访问v的邻接点
           {     if(g[v][i]!=0)  w=i;
                 if (!Visited(w))
                 {    visit(w);     Visited[w]=1;
                      InsertQueue(&Q, w);
                 }
           }
     }
}
```

```
BFS( )   //检查整个图G
{   for(int i=1;i<=n;i++)
       if(Visited[i]==0)
          BFSV0(i);
}
```

5.5 分支限界法

- 思想
  - 从根开始，常以宽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。首先将根结点加入活结点表（用于存放活结点的数据结构），接着从活结点表中取出根结点，使其成为当前扩展结点，一次性生成其所有孩子结点，判断孩子结点是舍弃还是保留，舍弃那些导致不可行解或导致非最优解的孩子结点，其余的被保留在活结点表中。再从活结点表中取出一个活结点作为当前扩展结点，重复上述扩展过程，一直持续到找到所需的解或活结点表为空时为止。由此可见，每一个活结点最多只有一次机会成为扩展结点。

- **分类（根据活结点表的维护方式）**
  - 队列式分支限界法
  - 优先队列式分支限界法

- **求解步骤**
  - 定义问题的解空间
  - 确定问题的解空间组织结构（树或图）
  - 搜索解空间。搜索前要定义判断标准（约束函数或限界函数），如果选用优先队列式分支限界法，则必须确定优先级。

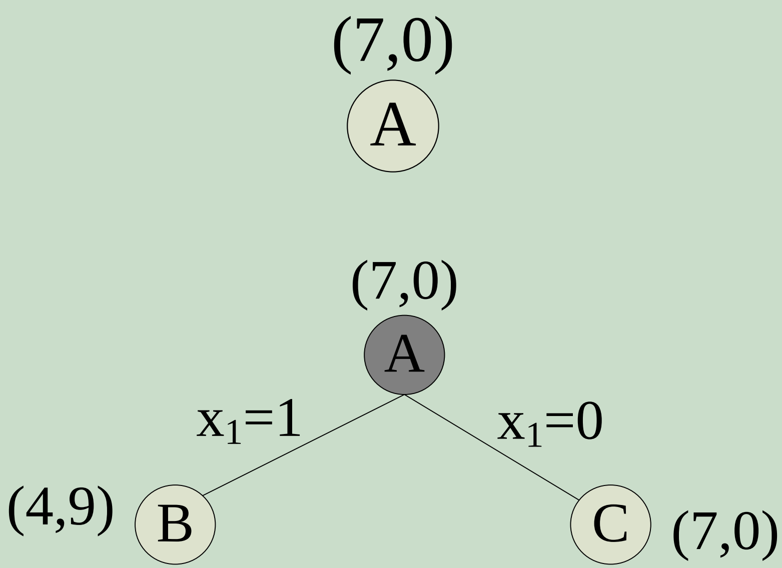
示例1：0-1背包问题

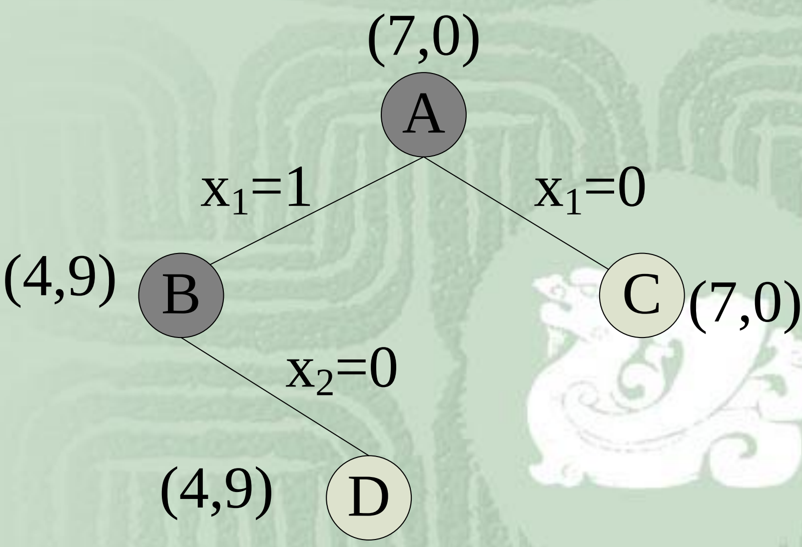
- **实例n=4，w=[3,5,2,1]，v=[9,10,7,4]，C=7。**
- **定义问题的解空间**
  - 解空间为(x1,x2,x3,x4)，xi=0或1(i=1,2,3,4)。
- **确定问题的解空间组织结构**
  - 该实例的解空间是一棵子集树，深度为4。
- **搜索解空间**
  - 约束条件 $\sum_{i=1}^{n} w_i x_i \le C$
  - 限界条件 **cp+rp>bestp**

队列式分支限界法

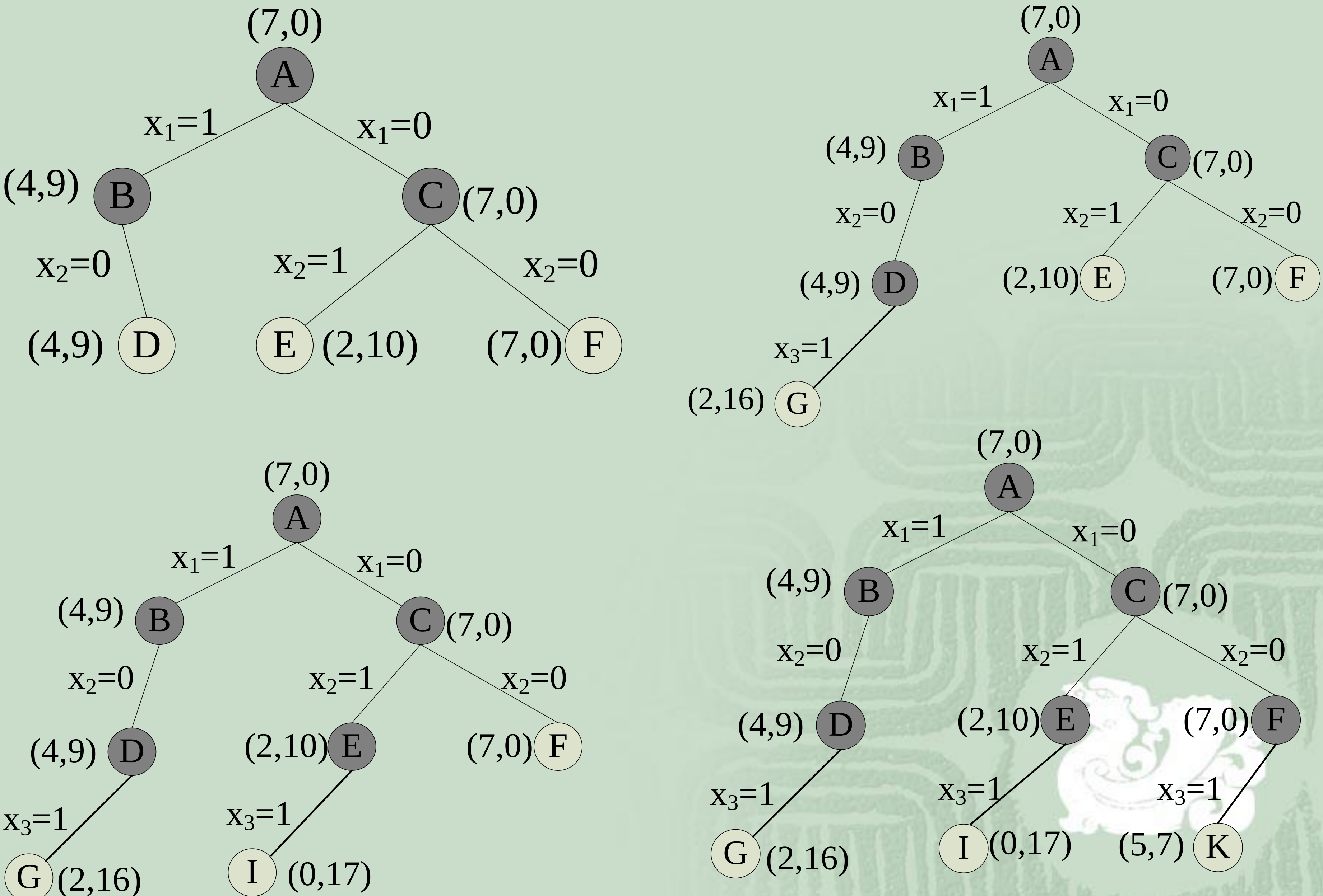
- **cp初始值为0；rp初始值为所有物品的价值之和；bestp表示当前最优解，初始值为0。**
- **当cp>bestp时，更新bestp为cp。**

w=[3,5,2,1]，v=[9,10,7,4]，C=7

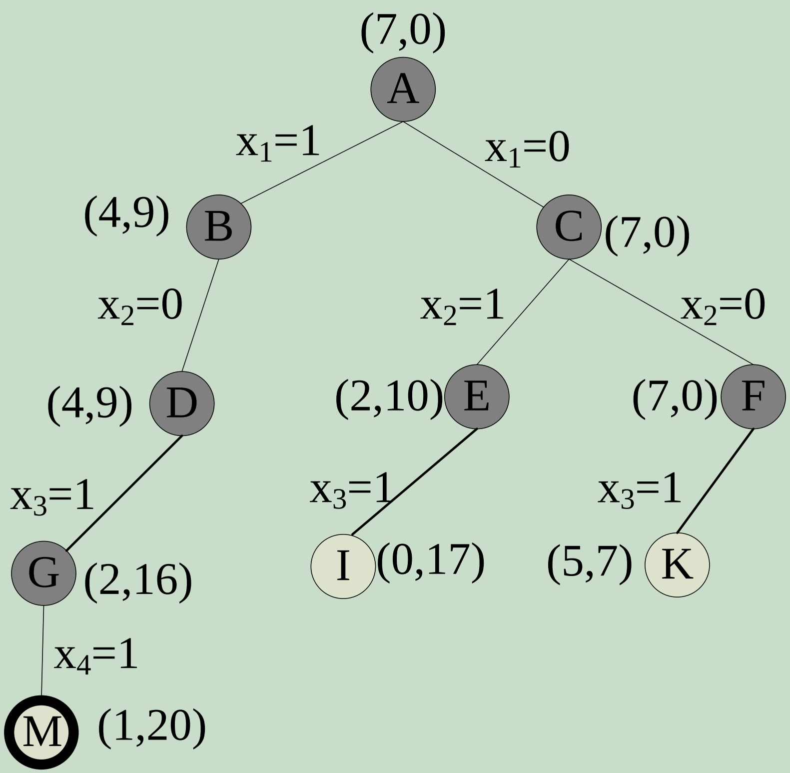


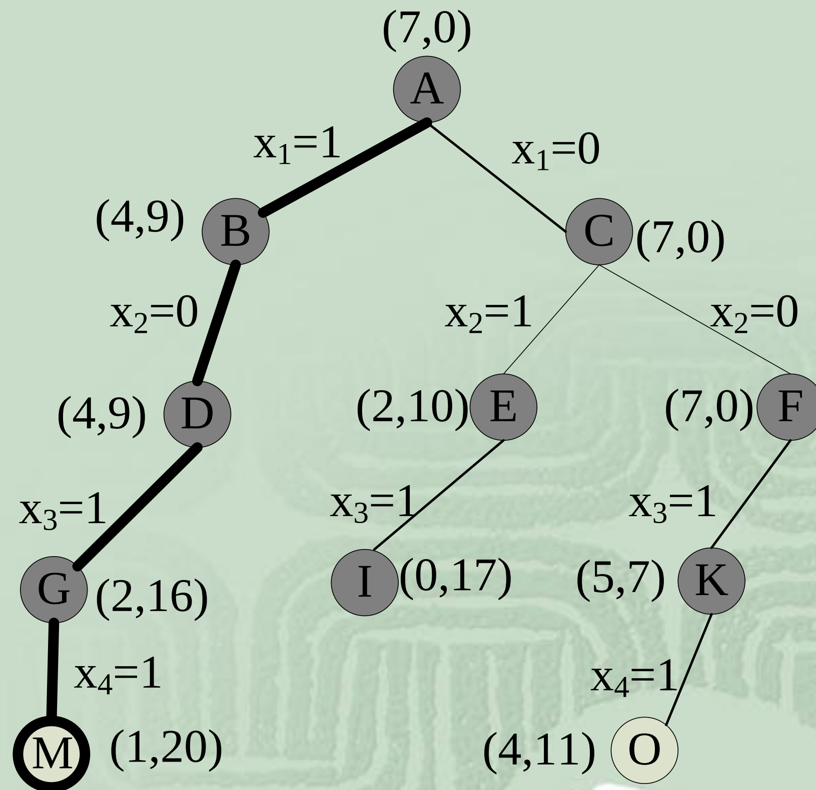


w=[3,5,2,1]，v=[9,10,7,4]，C=7



w=[3,5,2,1]， v=[9,10,7,4]， C=7

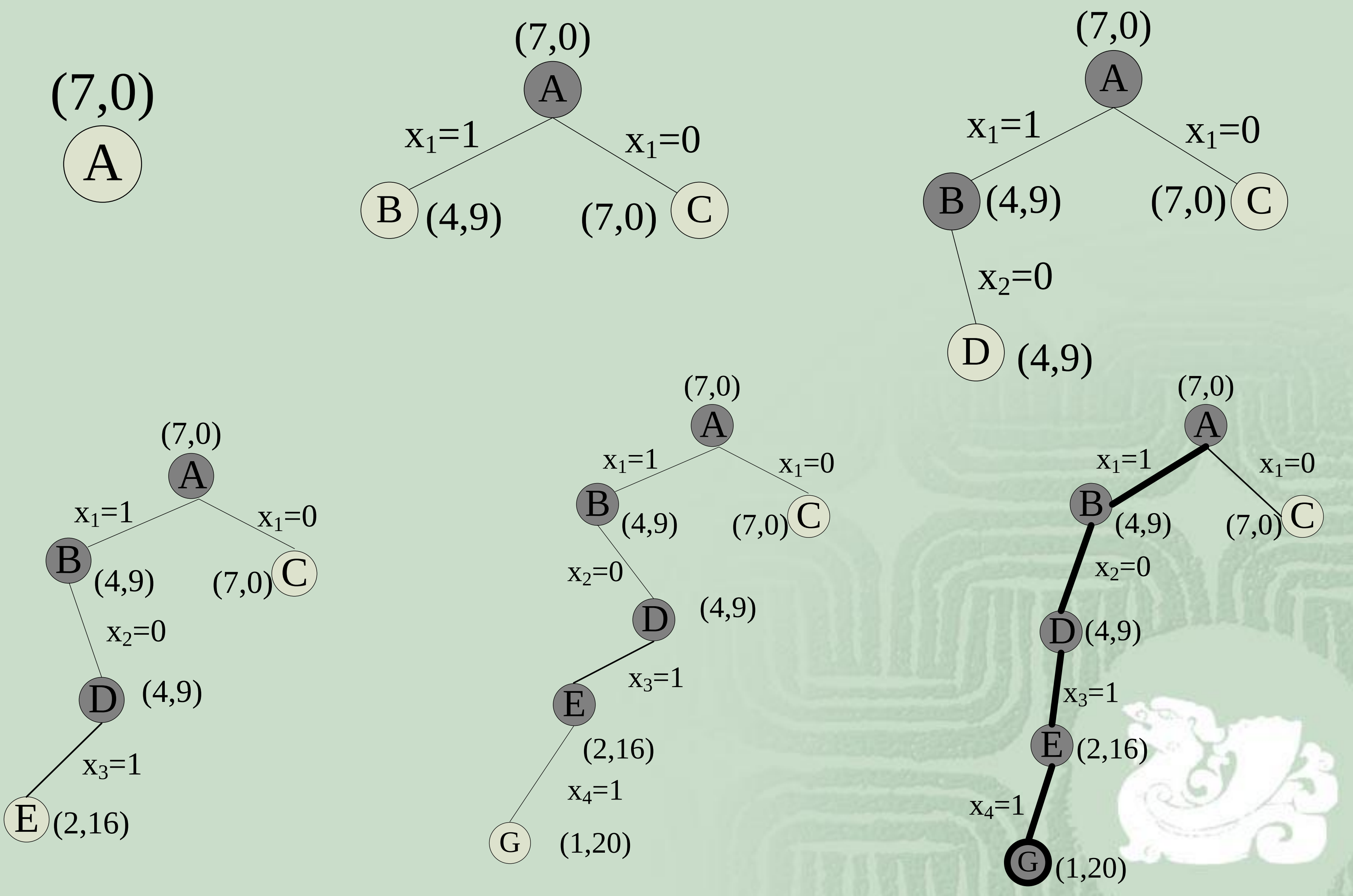




优先队列式分支限界法

- 优先级：活结点代表的部分解所描述的装入背包的物品价值上界，该价值上界越大，优先级越高。活结点的价值上界**up=cp+r'p**。
- 约束条件：同队列式
- 限界条件：**up=cp+r'p>bestp**。

w=[3,5,2,1]， v=[9,10,7,4]， C=7



算法描述

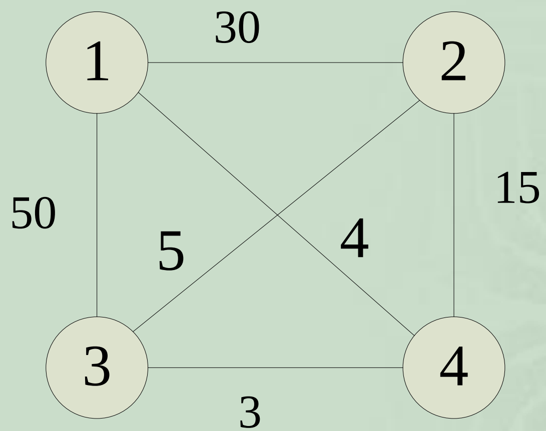
```
……
while (i != n+1)
{     wt = cw + w[i];
      if (wt <= c)
      {     if (cp+p[i] > bestp)   bestp = cp+p[i]; //左子树
             AddLiveNode(up, cp+p[i], cw+w[i], true, i+1);
      }
      up = Bound(i+1);
      if (up > bestp)      //右子树
           AddLiveNode(up, cp, cw, false, i+1);
      //取下一个扩展节点
      ……
}
```

思考：如何还原最优解

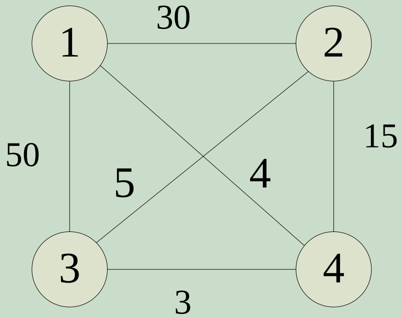
示例2：旅行售货员问题

问题描述

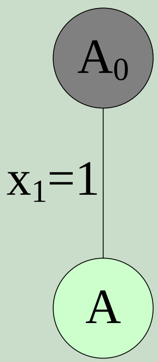
某售货员要到若干城市去推销商品，已知各城市之间的路程(或旅费)。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程(或总旅费)最小。

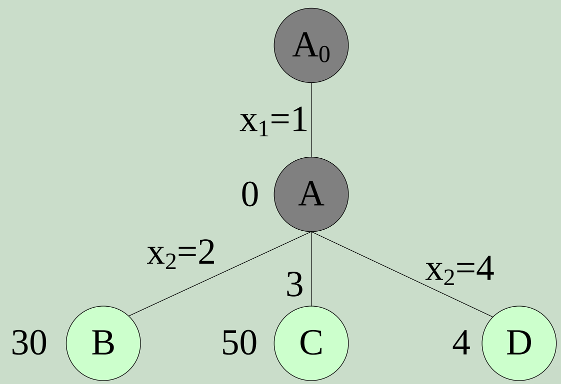


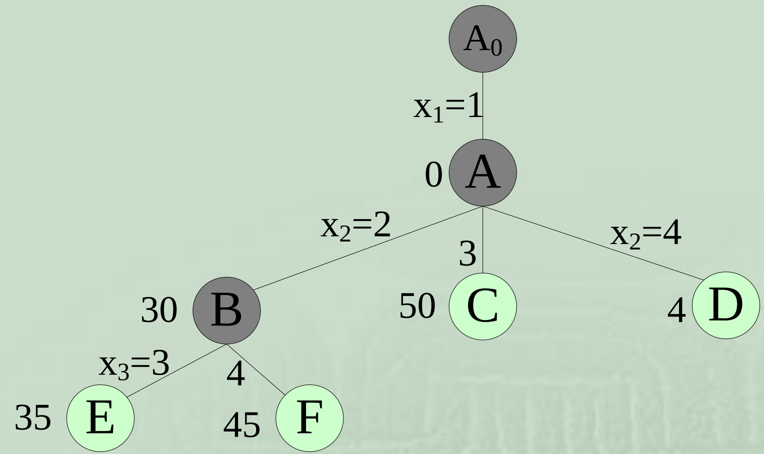
- **问题的解空间(x1,x2,x3,x4)：**
  - 其中令S={1,2,3,4}， x1=1，x2∈S-{x1}，x3∈S-{x1,x2}，x4 ∈ S-{x1,x2,x3}。
- **解空间的组织结构是一棵深度为4的排列树。**
- **搜索**
  - 约束条件g[i][j]!=∞，其中g是该图的邻接矩阵；
  - 限界条件：cl<bestl，其中cl表示当前已经走的路径长度，初始值为0；bestl表示当前最短路径长度，初始值为∞。

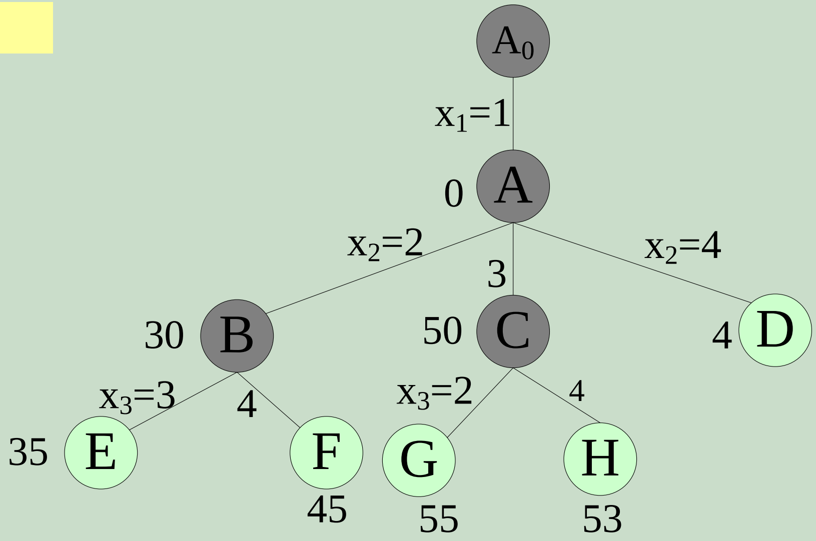

1
2
3
4
30
50
15
5
4
3

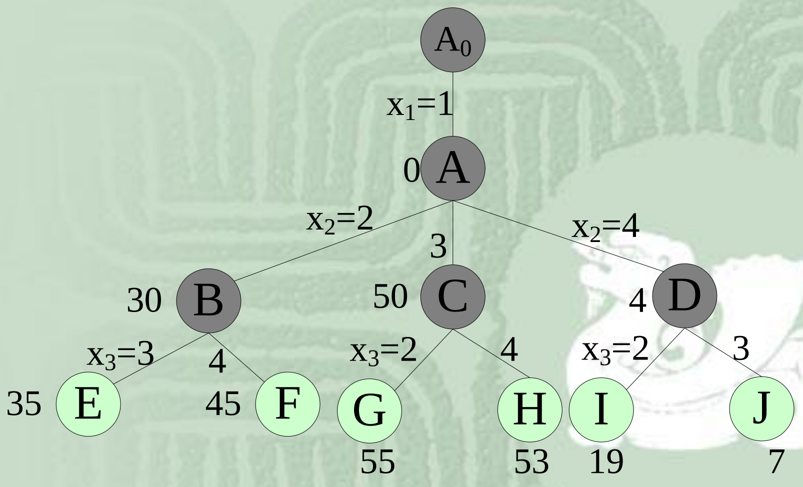
队列式分支限界法

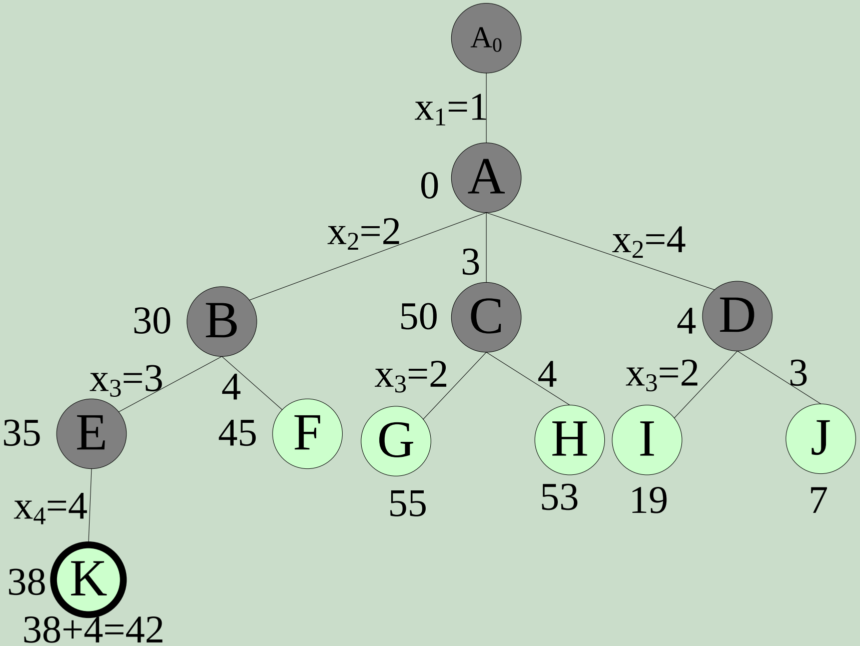

A_0
$x_1=1$
A


A_0
$x_1=1$
0
A
$x_2=2$
3
$x_2=4$
30
B
50
C
4
D

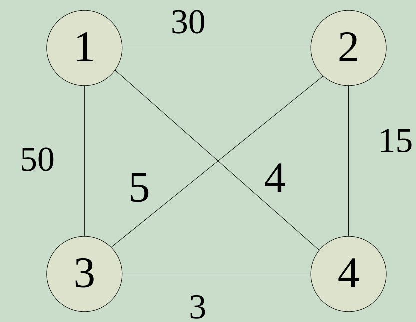

A_0
$x_1=1$
0
A
$x_2=2$
3
$x_2=4$
30
B
50
C
4
D
$x_3=3$
4
35
E
45
F


A_0
$x_1=1$
0
A
$x_2=2$
3
$x_2=4$
30
B
50
C
4
D
$x_3=3$
4
$x_3=2$
4
35
E
F
45
G
55
H
53

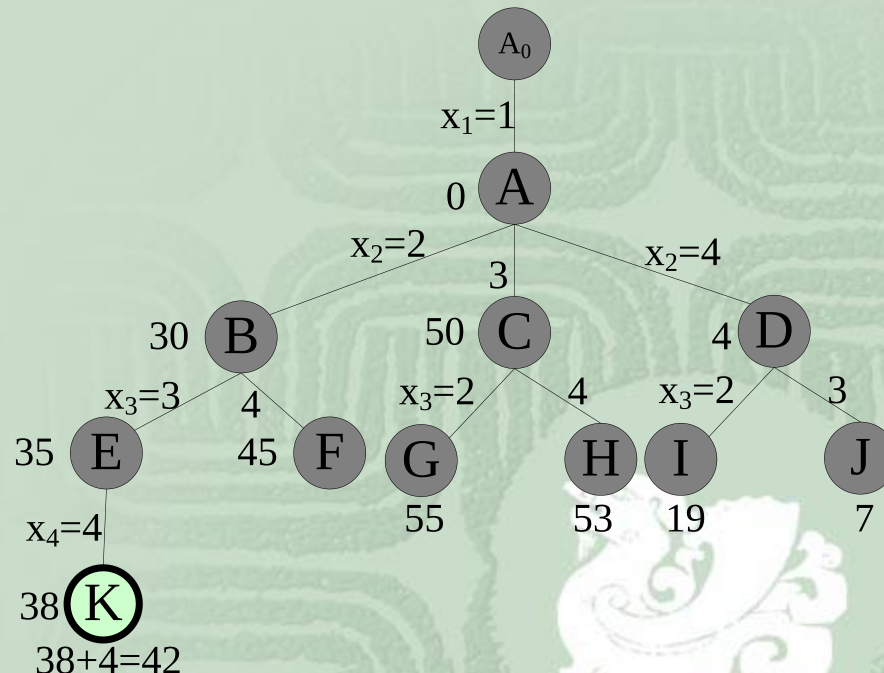

A_0
$x_1=1$
0
A
$x_2=2$
3
$x_2=4$
30
B
50
C
4
D
$x_3=3$
4
$x_3=2$
4
$x_3=2$
3
35
E
45
F
G
55
H
53
I
19
J
7



A0
x1=1
0
A
x2=2
3
x2=4
30
B
50
C
4
D
x3=3
4
x3=2
4
x3=2
3
35
E
45
F
G
H
I
J
55
53
19
7
x4=4
38
K
38+4=42



1
30
2
50
5
4
15
3
4
3

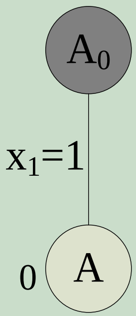


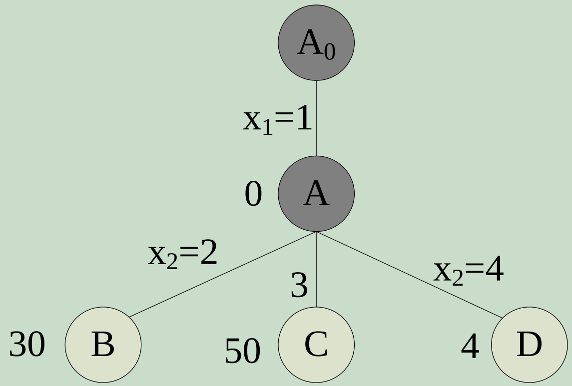
A0
x1=1
0
A
x2=2
3
x2=4
30
B
50
C
4
D
x3=3
4
x3=2
4
x3=2
3
35
E
45
F
G
H
I
J
55
53
19
7
x4=4
38
K
38+4=42

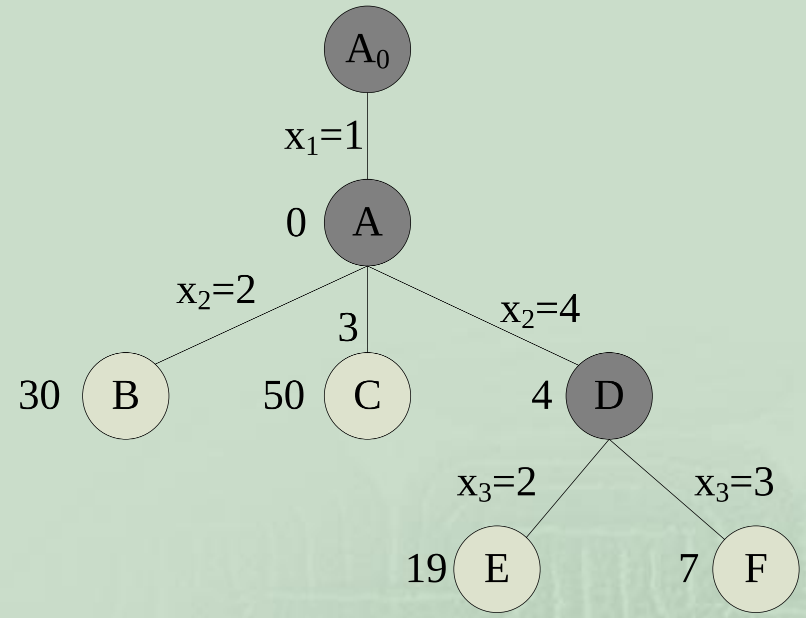
优先队列式分支限界法

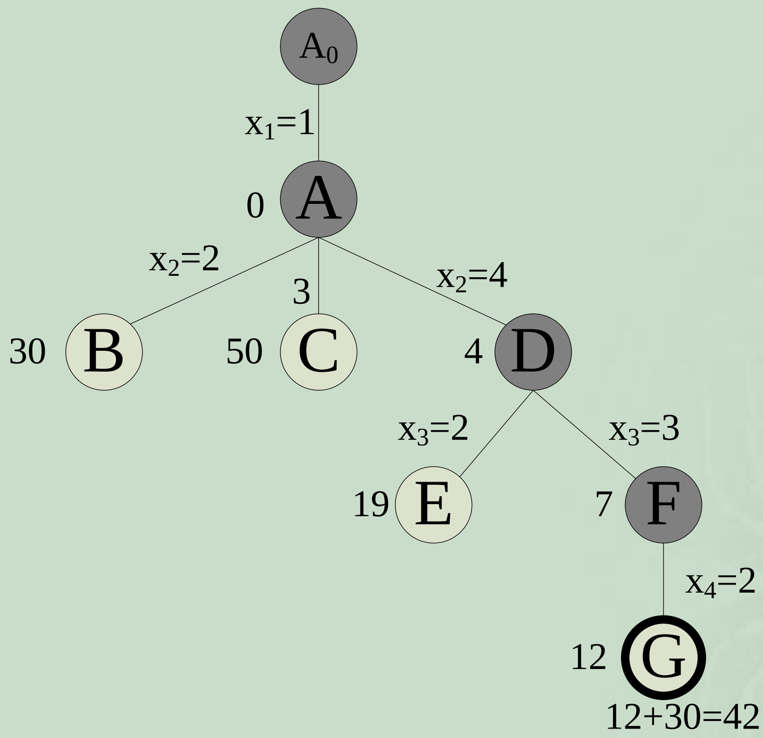
- 优先级：活结点所对应的已经走过的路径长度cl，长度越短，优先级越高。

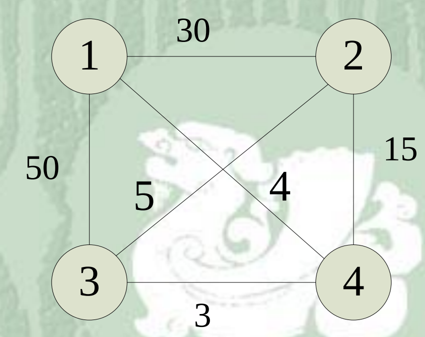
有更好的优先级吗？

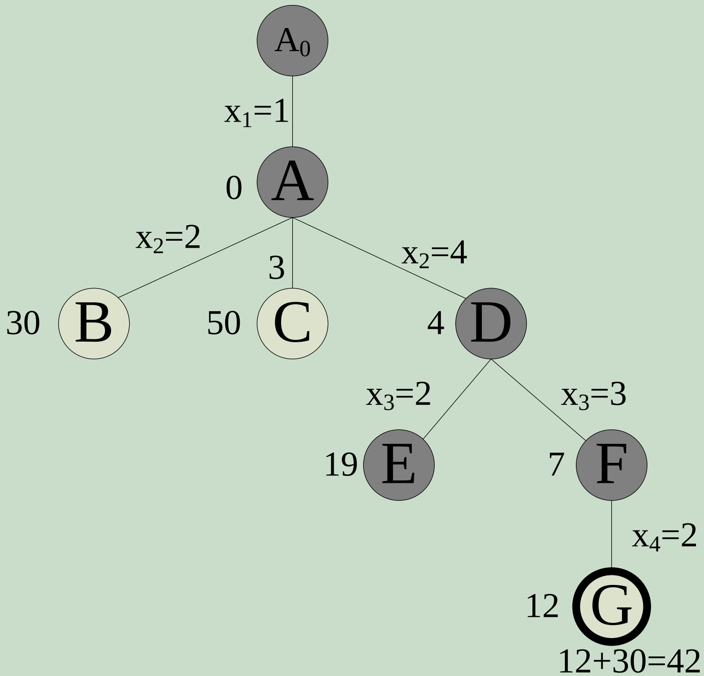

A0
x1=1
0
A

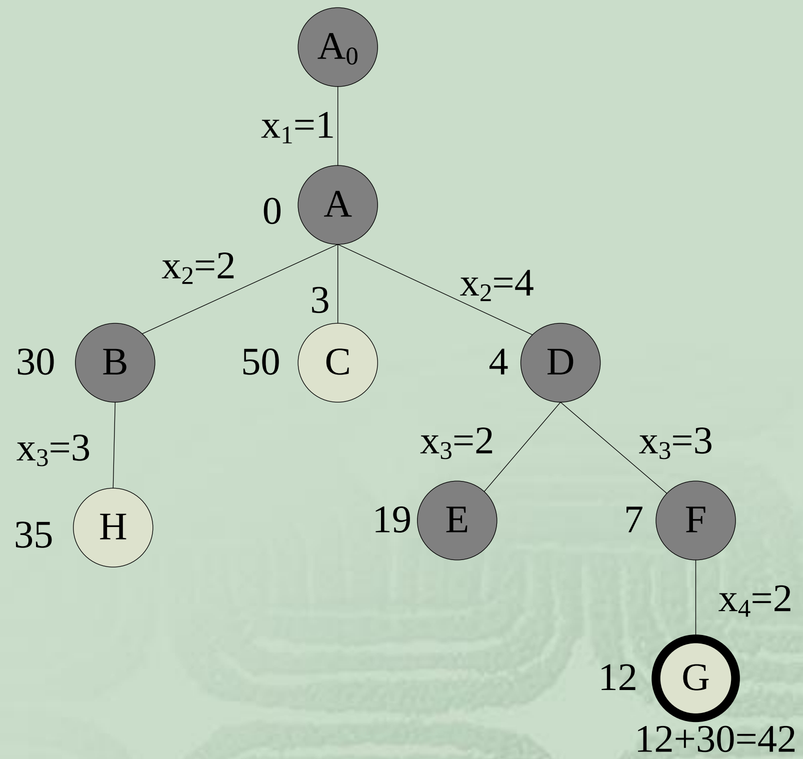

A0
x1=1
0
A
x2=2
3
x2=4
30
B
50
C
4
D

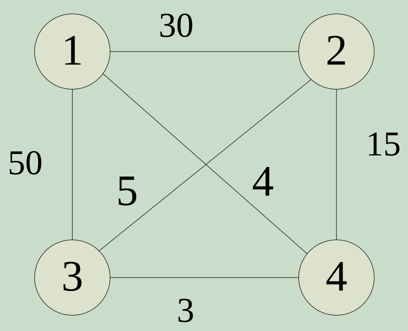

A0
x1=1
0
A
x2=2
3
x2=4
30
B
50
C
4
D
x3=2
x3=3
19
E
7
F

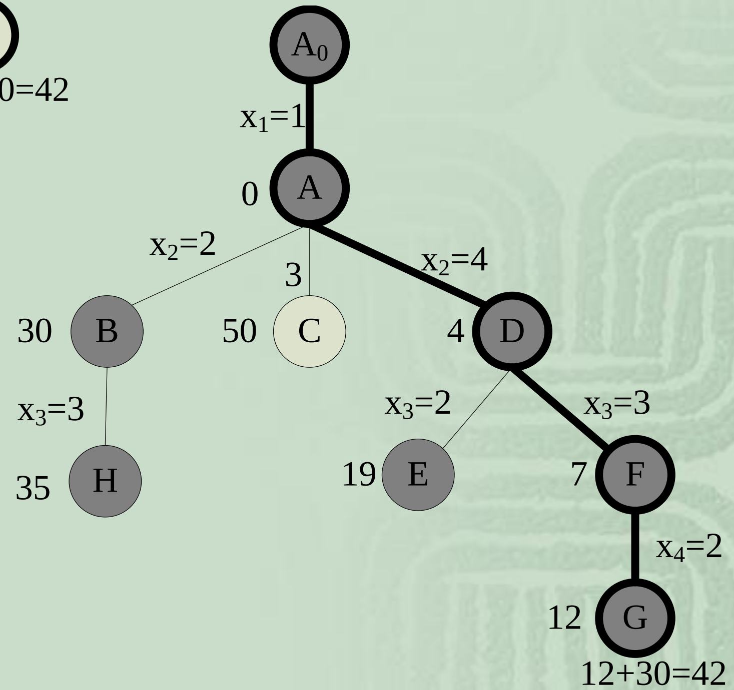

A0
x1=1
0
A
x2=2
3
x2=4
30
B
50
C
4
D
x3=2
x3=3
19
E
7
F
x4=2
12
G
12+30=42


1
30
2
50
5
4
15
3
3
4


A_0
$x_1=1$
0 A
$x_2=2$
3
$x_2=4$
30 B
50 C
4 D
$x_3=2$
$x_3=3$
19 E
7 F
$x_4=2$
12 G
12+30=42

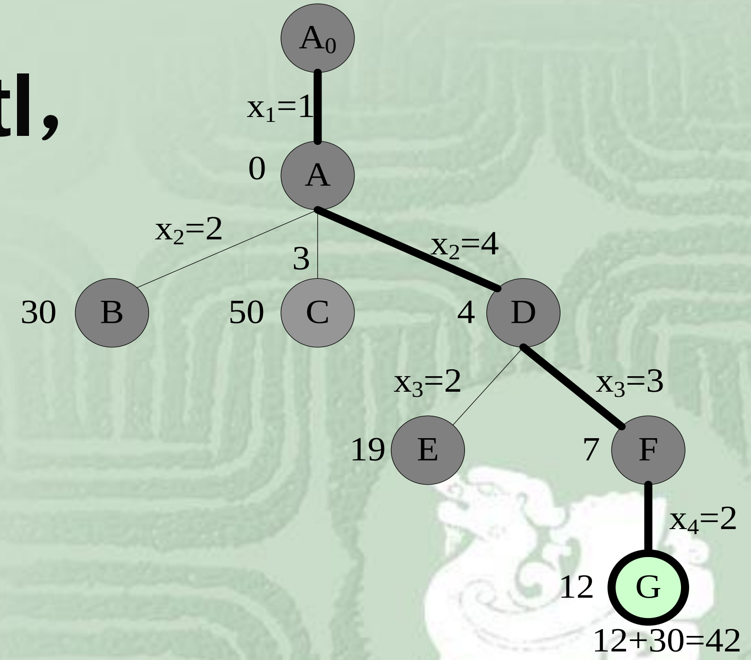

A_0
$x_1=1$
0 A
$x_2=2$
3
$x_2=4$
30 B
50 C
4 D
$x_3=3$
$x_3=2$
$x_3=3$
35 H
19 E
7 F
$x_4=2$
12 G
12+30=42


1
30
2
50
5
4
15
3
4
3


A_0
$x_1=1$
0 A
$x_2=2$
3
$x_2=4$
30 B
50 C
4 D
$x_3=3$
$x_3=2$
$x_3=3$
35 H
19 E
7 F
$x_4=2$
12 G
12+30=42

算法优化

- 估计路径长度的下界用zl表示，**zl=cl+rl**
- 优先级：活结点的**zl**，**zl**越小，优先级越高；
- 约束条件：同上
- 限界条件：**zl=cl+rl>bestl**，



```
while(E.s<=n) //非叶结点
{   if(E.s ==n)
   {   if(a[E.x[n-1]][E.x[n]]!=∞ && a[E.x[n]][1]!=∞ && (E.cl+a[E.x[n-1]][E.x[n]]
                                                       +a[E.x[n]][1]<bestl))
       {   bestl=E.cl+a[E.x[n-1]][E.x[n]]+a[E.x[n]][1];
           E.cl=bestl; E.zl=bestl; E.key=E.zl;
       }
   }
   else
   {   for(i=E.s;i<=n;i++)
         if(a[E.x[E.s-1]][E.x[i]]!=∞) //可行子结点
         {   double cl=E.cl+a[E.x[E.s-1]][E.x[i]];
             double rl=E.rl-MinOut[E.x[E.s-1]];
             Type b=cl+rl; //子结点的下界
             if(b<bestl)
             {   MinHeapNode  N; N.x=new int[n+1];
                 for(j=1;j<=n;j++)   N.x[j]=E.x[j];
                 N.x[E.s]=E.x[i]; N.x[i]=E.x[E.s]; N.cl=cl; N.s=E.s +1;
                 N.n=n; N.zl =b; N.key=N.zl; N.rl=rl; H.Insert(N);
             }
         }
   }
   H.DeleteMin(E); //取下一活结点扩展
}
```

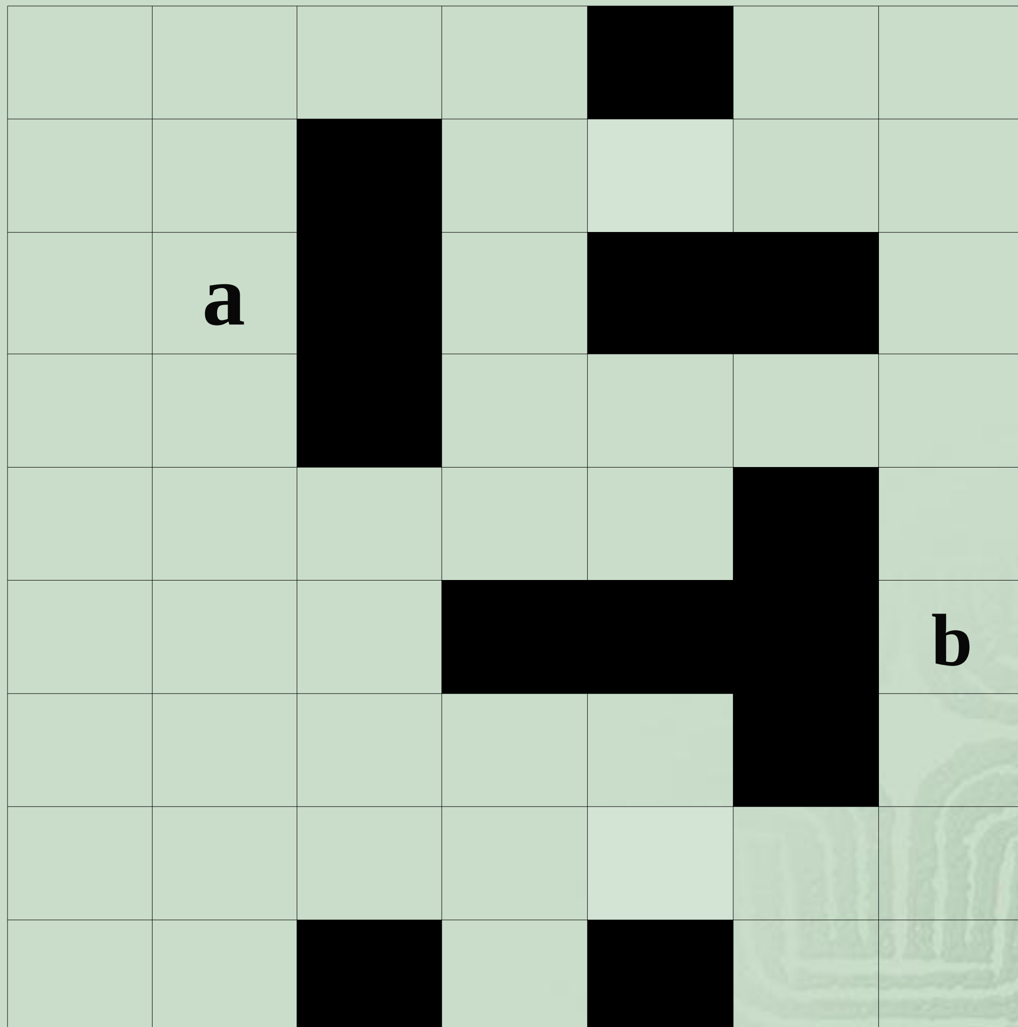
用规约矩阵计算下界。

思考：如何还原最优解

示例4： 布线问题

问题描述

- 在N×M的方格阵列中，指定一个方格的中点为a，另一个方格的中点为b，问题要求找出a到b的最短布线方案（即最短路径）。布线时只能沿直线或直角，不能走斜线。黑色的单元格代表不可以通过的封锁方格。如下图所示。



9×7阵列

问题分析

- 将方格抽象为顶点，中心方格和相邻四个方向（上、下、左、右）能通过的方格用一条边连起来。这样，可以把问题的解空间定义为一个图。
- 该问题是特殊的最短路径问题，特殊之处在于用布线走过的方格数代表布线的长度，布线时每布一个方格，布线长度累加1。
- 只能朝上、下、左、右四个方向进行布线

实例

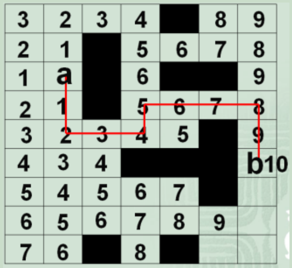
3	2	3	4		8	9
2	1		5	6	7	8
1	a		6			9
2	1		5	6	7	8
3	2	3	4	5		9
4	3	4				b10
5	4	5	6	7		
6	5	6	7	8	9	
7	6		8			

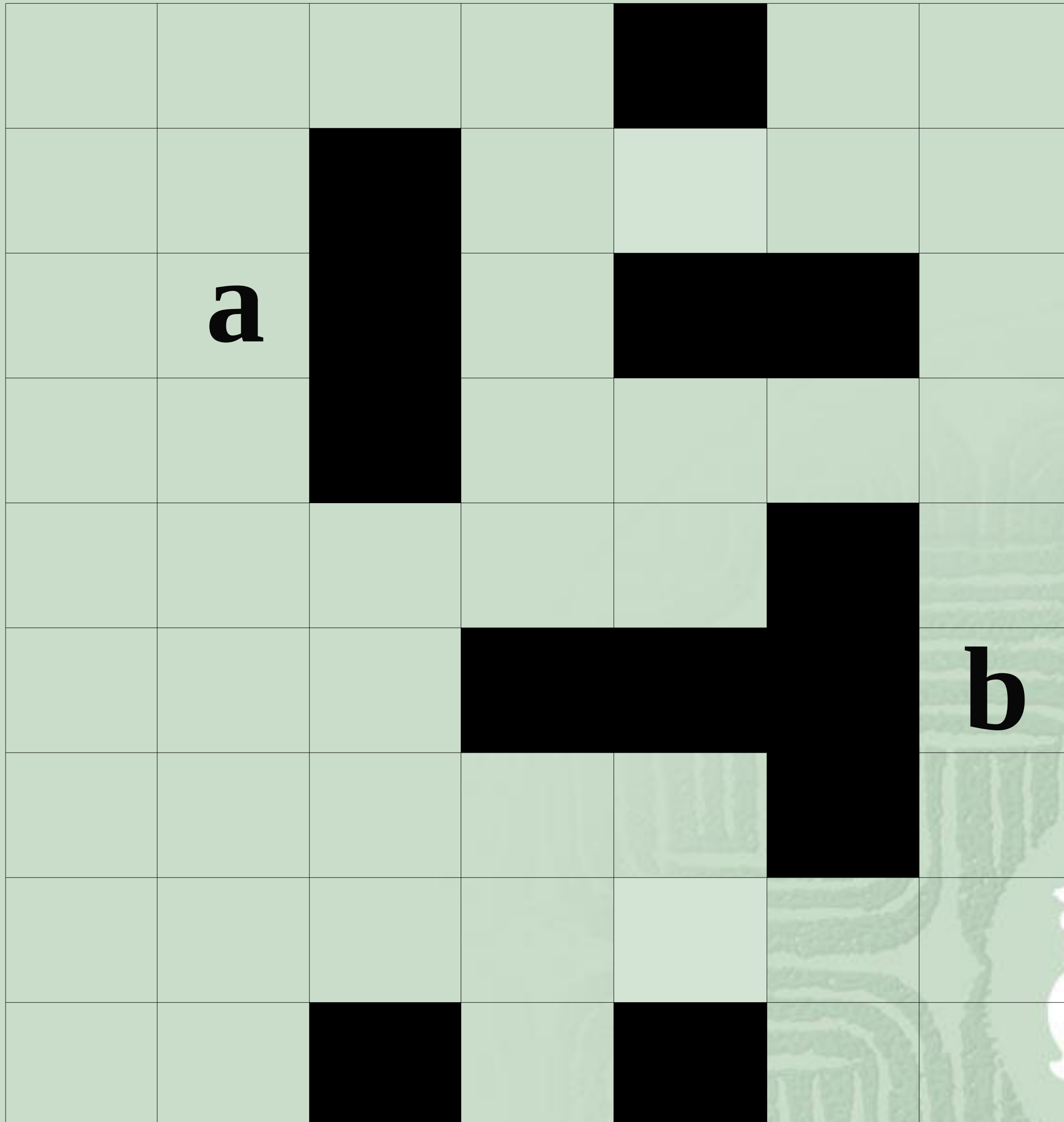
左

上

右

下


3 2 3 4 8 9
2 1 5 6 7 8
1 a 6 9
2 1 5 6 7 8
3 2 3 4 5 9
4 3 4 b10
5 4 5 6 7
6 5 6 7 8 9
7 6 8


a
b

算法描述

思考1：如何表示左、上、右、下四个方向？

```
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
```

定义移动方向的相对位移

思考2：如何表示阵列的边界？

```
for (int i = 0; i <= m+1; i++)
    grid[0][i] = grid[n+1][i] = -2; // 顶部和底部
for (int i = 0; i <= n+1; i++)
    grid[i][0] = grid[i][m+1] = -2; // 左翼和右翼
```

设置边界的围墙

思考3：如何表示可以布线的方格？

```
do{
    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]==-2) continue;
        if(grid[nbr.row][nbr.col]==-1)
            grid[nbr.row][nbr.col]=grid[here.row][here.col]+1;
        if((nbr.row==finish.row)&&(nbr.col==finish.col))
            break;
        Q.Add(nbr);      //此邻结点放入队列
    }
    if((nbr.row==finish.row)&&(nbr.col==finish.col))
        break;              //完成布线
    if(Q.Isempty()) return;
        Q.Delete(here);     //从队列中取下一个扩展结点
} while(true)
```

思考4：找到目标位置后，如何找到布线方案？

```
PathLen=grid[finish.row][finish.col];
path=new Position[Pathlen];
here=finish;
for(int j=PathLen-1; j>=0; j--)
{      path[j]=here;
       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)
                     break;
       }
       here=nbr;    //往回推进
}
```

思考5：如何能斜45度布线，怎么修改算法？

	A	B	C	D
1	a	1	2	3
2	1	1.4		3.4
3	2	2.4		4.4
4	3	3.4	3.8	b

a			
			b

分支限界法与回溯法的比较

- 相同点
  - 均需要先定义问题的解空间，确定的解空间组织结构一般都是树或图。
  - 在问题的解空间树上搜索问题解。
  - 搜索前均需确定判断条件，该判断条件用于判断扩展生成的结点是否为可行结点。
  - 搜索过程中必须判断扩展生成的结点是否满足判断条件，如果满足，则保留该扩展生成的结点，否则舍弃。

- 不同点
  - 搜索目标：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通常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，因此需要更复杂的数据结构。
  - 扩展方式不同：在回溯法搜索中，扩展结点一次生成一个孩子结点，而在分支限界法搜索中，扩展结点一次生成它所有的孩子结点。
  - 如果每层的分支数目过多，则不适合用分支限界，因为存在大量候选节点，且每个节点要保持历史选择，会造成活结点表空间暴涨。

思考题

- 1.单源最短路径问题
- 2.走棋（米字旗、拼图、华容道）
- 3. 地图上每块方格中都包含不等的矿产资源，你只能建立一个面积为S（S个方格）的基地，并且基地是个连通图，找出能够包含的最多资源。（连通性）

如何判断连通图中的空洞？
可以对非选择做连通性测试。