四皇后问题

四皇后问题

在n×n格的棋盘上放置彼此不受攻击的n个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。n后问题等价于在n×n格的棋盘上放置n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。

	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

四皇后问题

显然，棋盘的每一行上可以而且必须摆放一个皇后，所以，n皇后问题的可能解用一个n元向量$X=(x_1, x_2, \ldots, x_n)$表示，其中，$1 \leqslant i \leqslant n$并且$1 \leqslant x_i \leqslant n$，即第$i$个皇后放在第$i$行第$x_i$列上。

由于两个皇后不能位于同一列上，所以，解向量X必须满足约束条件1：

$$x_i \neq x_j$$

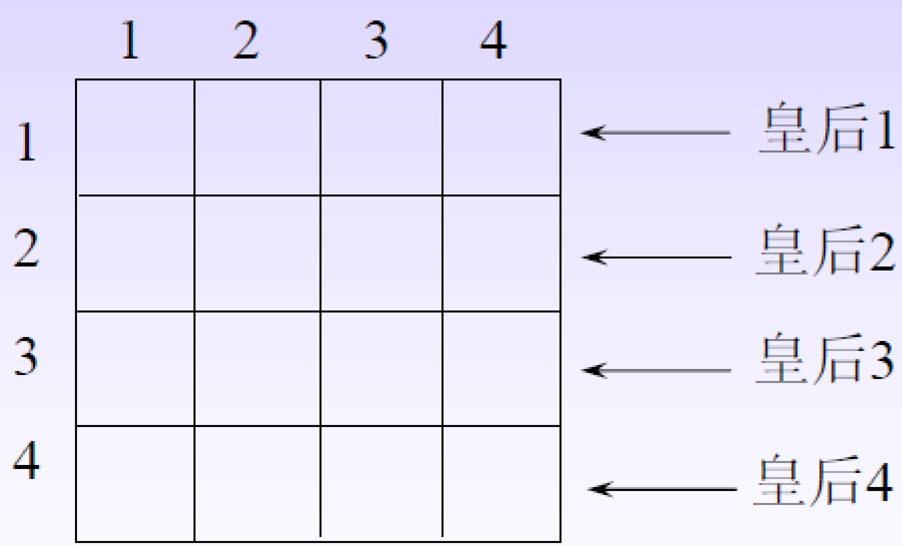
若两个皇后摆放的位置分别是(i, x_i)和(j, x_j)，在棋盘上斜率为-1的斜线上，满足条件$i-j= x_i-x_j$，在棋盘上斜率为1的斜线上，满足条件$i+j= x_i+x_j$，综合两种情况，由于两个皇后不能位于同一斜线上，所以，解向量X必须满足约束条件2：

$$|i-x_i| \neq |j-x_j|$$

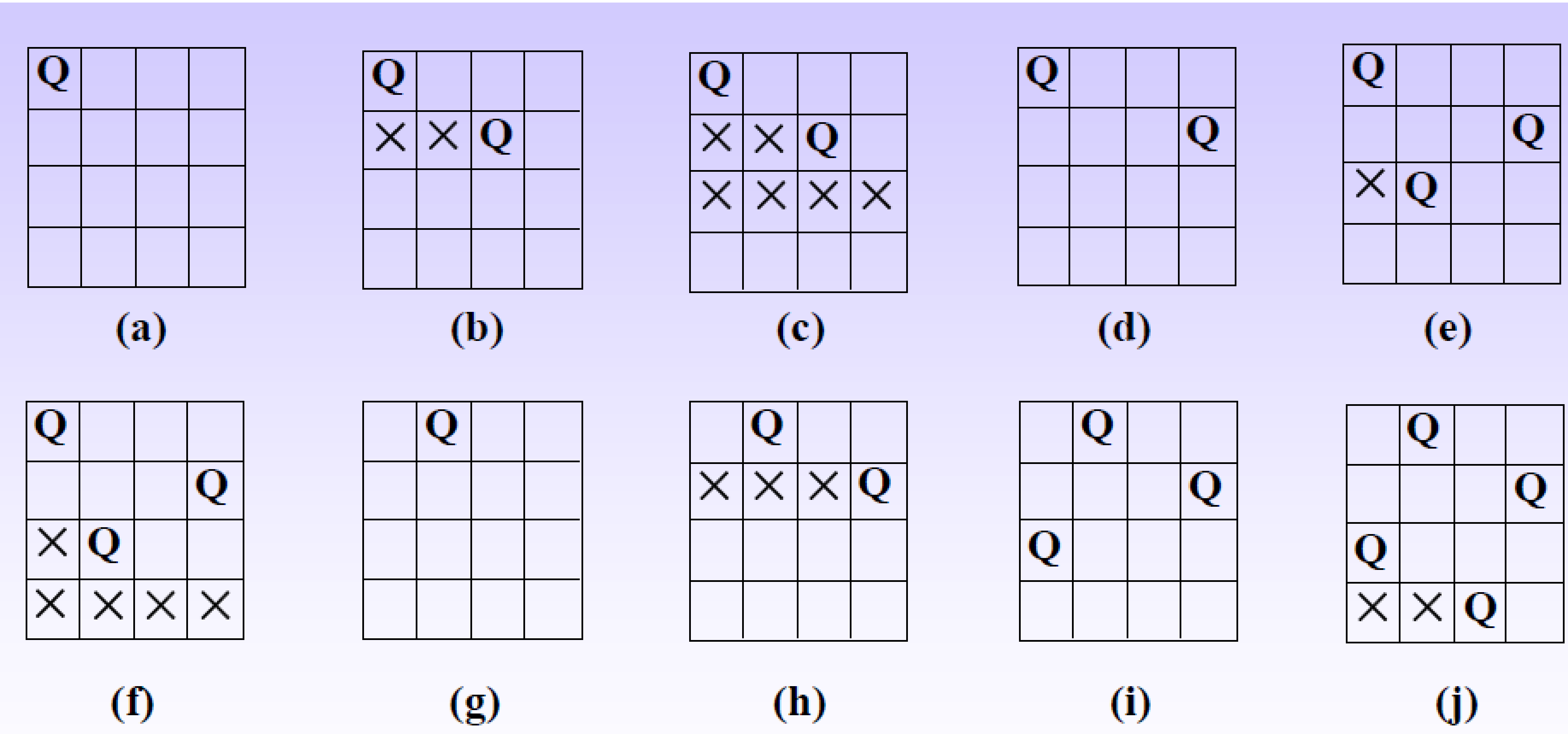
四皇后问题

为了简化问题，下面讨论四皇后问题。

四皇后问题的解空间树是一棵**完全4叉树**，树的根结点表示搜索的初始状态，从根结点到第2层结点对应皇后1在棋盘中第1行的可能摆放位置，从第2层结点到第3层结点对应皇后2在棋盘中第2行的可能摆放位置，依此类推。



四皇后问题



八皇后问题

```
bool Queen::Place(int k)
{
  for (int j=1;j<k;j++)
    if ( (abs(k-j)==abs(x[j]-x[k])) || (x[j]==x[k]) ) return false;
  return true;
}

void Queen::Backtrack(int t)
{
  if (t>n) sum++;
    else
      for (int i=1;i<=n;i++) {
        x[t]=i;
        if (Place(t)) Backtrack(t+1);
      }
 }
```

回溯法效率分析

通过前面具体实例的讨论容易看出，回溯算法的效率在很大程度上依赖于以下因素:

(1)产生x[k]的时间;

(2)满足显约束的x[k]值的个数;

(3)计算约束函数**constraint**的时间;

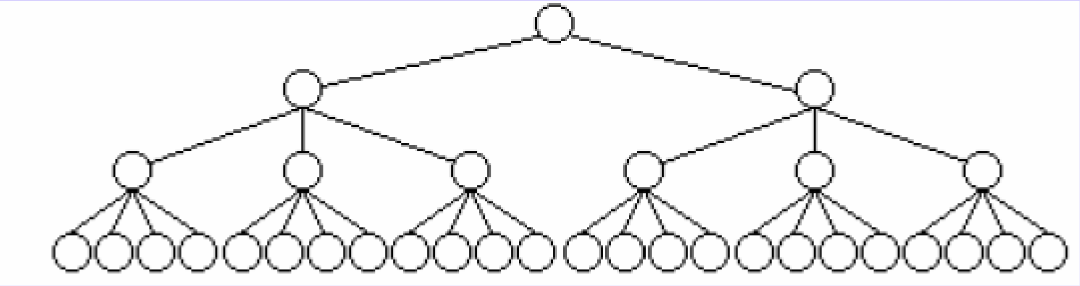
(4)计算上界函数**bound**的时间;

(5)满足约束函数和上界函数约束的所有x[k]的个数。

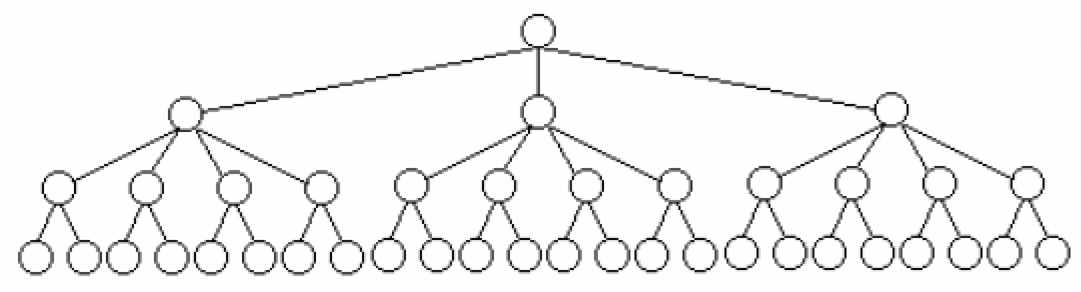
好的约束函数能显著地减少所生成的结点数。但这样的约束函数往往计算量较大。因此，**在选择约束函数时通常存在生成结点数与约束函数计算量之间的折衷。**

重排原理

对于许多问题而言，在搜索试探时选取x[i]的值顺序是任意的。**在其它条件相当的前提下，让可取值最少的x[i]优先。**从图中关于同一问题的2棵不同解空间树，可以体会到这种策略的潜力。



(a)



(b)

图(a)中，从第1层剪去1棵子树，则从所有应当考虑的3元组中一次消去12个3元组。对于图(b)，虽然同样从第1层剪去1棵子树，却只从应当考虑的3元组中消去8个3元组。前者的效果明显比后者好。

用概率方法估算回溯法所产生的结点数

基本思想：假定约束函数是静态的（即在回溯法的执行过程中，约束函数不随算法所获得信息的多少而动态改变），在解空间树上产生一条随机路径，然后沿此路径估算解空间树中满足约束条件的结点总数m。设x是所产生随机路径上的一个结点，且位于解空间树的第i层，对于x的所有孩子结点，计算出满足约束条件的结点数m_i，路径上的下一个结点从x的满足约束条件的m_i个孩子结点中随机选取，这条路径一直延伸，直到叶子结点或者所有孩子结点均不满足约束条件为止。

随机路径中含有的结点总数计算方法

假设第1层有m_0个满足约束条件的结点，每个结点有m_1个满足约束条件的孩子结点，则第2层上有m_0m_1个满足约束条件的结点，同理，假设第2层上的每个结点均有m_2个满足约束条件的孩子结点，则第3层上有$m_0m_1m_2$个满足约束条件的结点，依此类推，第n层上有$m_0m_1m_2 \ldots m_{n-1}$个满足约束条件的结点，因此，这条随机路径上的结点总数为：$m_0 + m_0m_1 + m_0m_1m_2 + \ldots + m_0m_1m_2 \ldots m_{n-1}$。

Q			
		Q	

(4, 2)=4+4×2
=12

	Q		
			Q
Q			
		Q	

(4, 1, 1, 1)=4+4+4+4
=16

		Q	
Q			
			Q
	Q		

(4, 1, 1, 1)=4+4+4+4
=16

			Q
	Q		

(4, 2)=4+4×2
=12

随机路径中含有的结点总数计算方法

在使用概率估算方法估算搜索空间的结点总数时，为了估算得更精确一些，可以选取若干条不同的随机路径（通常不超过20条），分别对各随机路径估算结点总数，然后再取这些结点总数的平均值。例如，上页图所示4皇后问题，搜索空间的结点数取4条随机路径结点总数的平均值，结果为14。而4皇后问题的解空间树中的结点总数为65，则回溯法求解4皇后问题产生的搜索空间的结点数大约是解空间树中的结点总数的14/65≈21.5%，这说明回溯法的效率大大高于穷举法。

旅行售货员(TSP)问题

旅行售货员(TSP)问题

- 组合优化领域著名、经典问题。上世纪20年代，由著名数学家、经济学家Karl Menger提出
- 问题描述:
- 旅行商从驻地出发，经过每个需要访问的城市一次且只有一次，并最终返回出发点。

  如何安排路线，使旅行总路程最短?
- 应用:

  军事、通信、电路板设计、大规模集成电路、基因排序等领域具有广泛应用

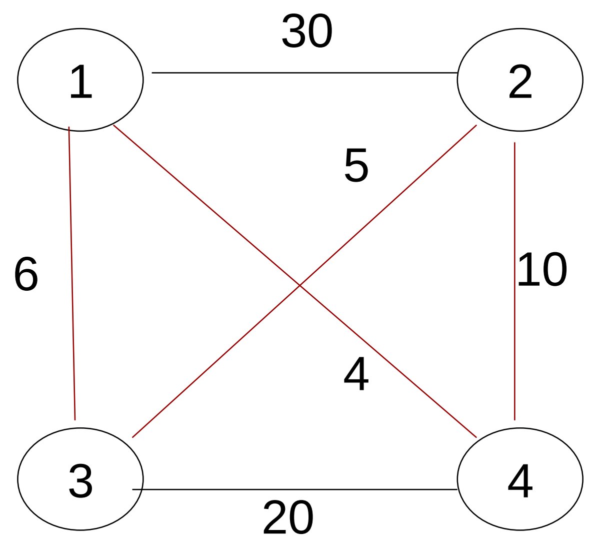
TSP问题的回溯求解

1. 问题形式化定义:

n个城市组成的带权无向图G=(V,E)，顶点V对应于城市，边E对应于城市间路径，要求找出一条旅行线路，每个城市只经历一次

2. 回溯法求解:

排列树问题



TSP问题的回溯求解

- 旅行回路总费用极小化

$$\min\{\sum_{i=1}^{n-1} w(v_i, v_{i+1}) \qquad + \qquad w(v_n, v_1)\}$$

式中，权$w(v_i,v_j)$（或：w(i, j)）表示城市i与j间的直接距离，w(vi,vj)=∞表示城市i与j间无直接路径。

将图中n个顶点编号为1,2,…,n;

以顶点1为起点，旅行回路描述为：1, x_1, x_2, .., x_n, 1;

其中， x_1, x_2, .., x_n为顶点2,3,4,…, n的一个排列。

因此，解空间大小为(n-1)!

TSP问题的回溯求解

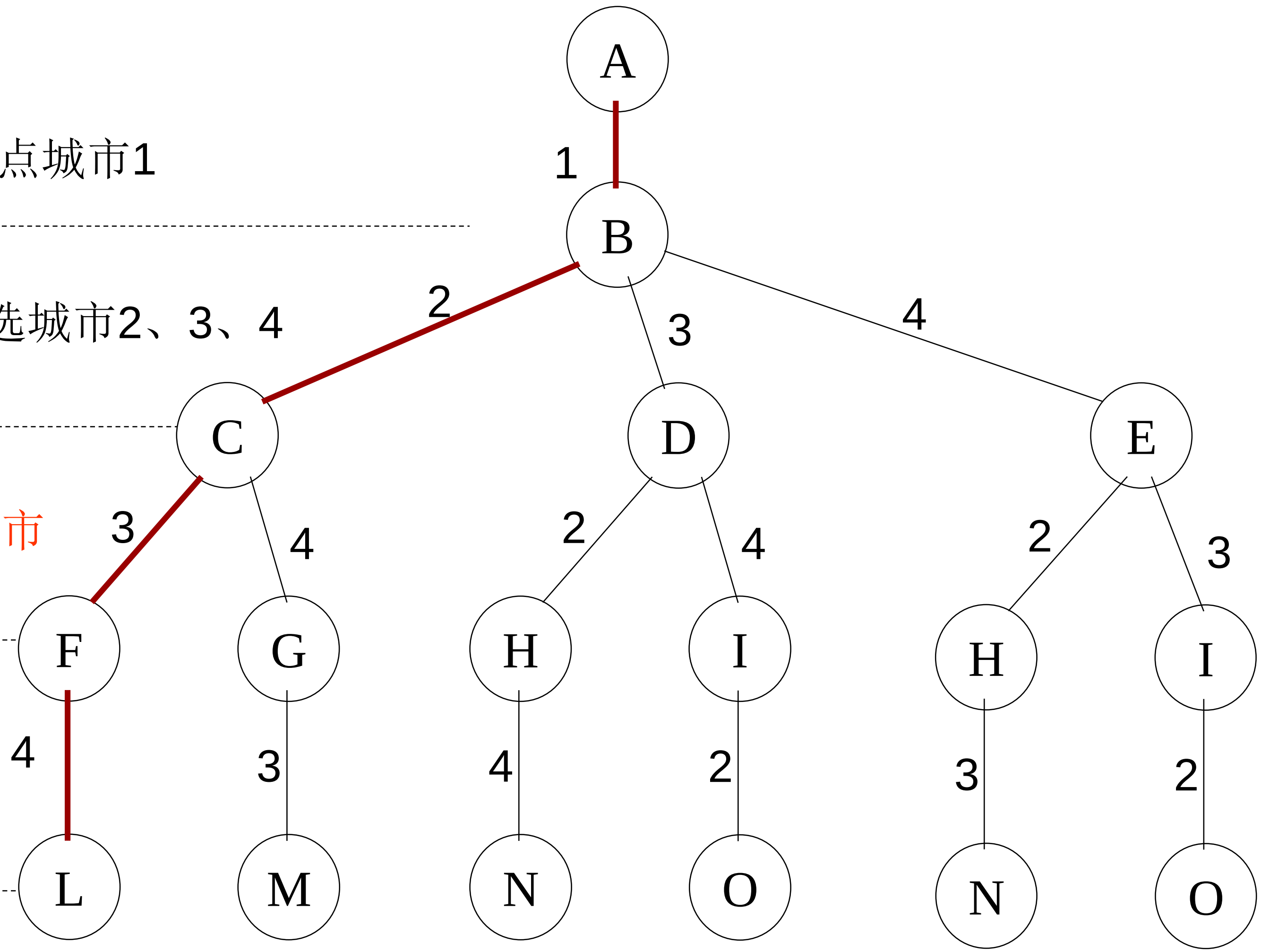
求解过程：以深度优先的方式，从树根结点开始，依次扩展树结点，直到达到叶结点——搜索过程中动态产生解空间

第1步：起点城市1

第2步：可选城市2、3、4

第3步：可选城市

第4步

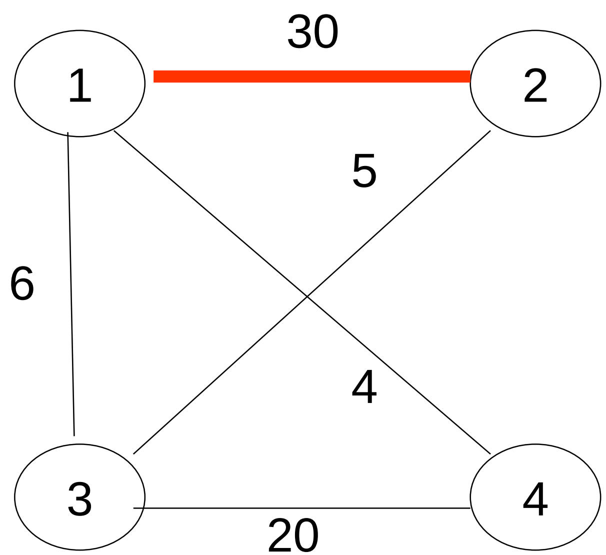


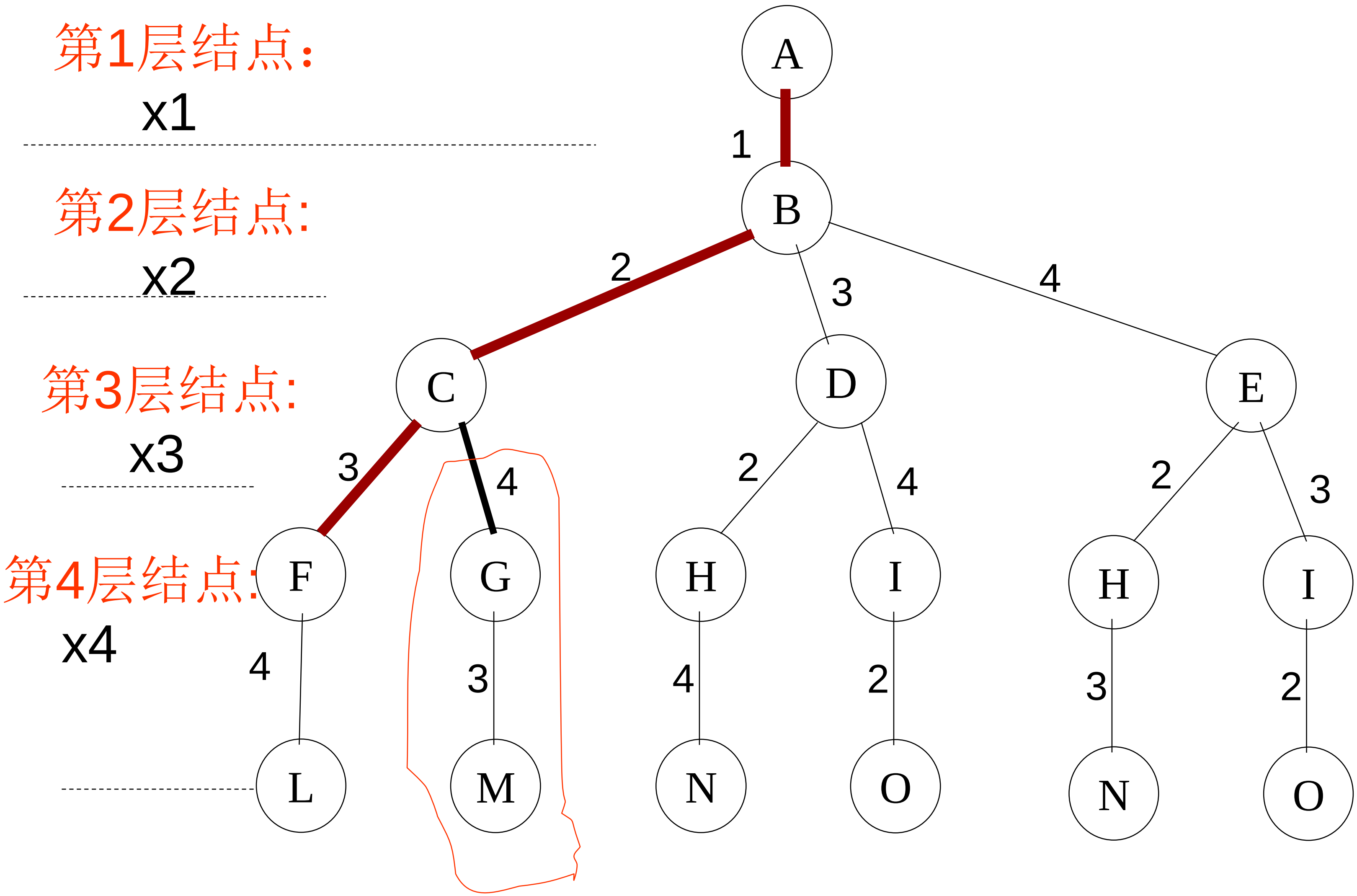
设置约束条件（函数）

剪枝条件/约束1：如果当前正在考虑的顶点j与当前路径中的末端结点i没有边相连，即w[i, j]= ∞, 则不必搜索j所在分支

E.g. 当前已有的部分路径为<1,2, ?, ?>, ,路径末端结点为2;

根据路径组成规则，下一步可考虑将顶点3、4加入到部分路中。但是，顶点2与4间无边，w(2,4)= ∞, 因此在解空间树中，可以不必考虑顶点4所在分支——见下页




第1层结点:
x1
第2层结点:
x2
第3层结点:
x3
第4层结点:
x4
A
1
B
2
3
4
C
D
E
3
4
2
4
2
3
F
G
H
I
H
I
4
3
4
2
3
2
L
M
N
O
N
O

此分支可剪枝

令到第i层结点为止，构造的部分解路径为

<1, x[2], x[3], …, x[i−1], x[i], ?, ? ,?>,

该路径的权值总和为:

$$cw(i) = \sum_{j=2}^{i} w(x[j-1], x[j])$$

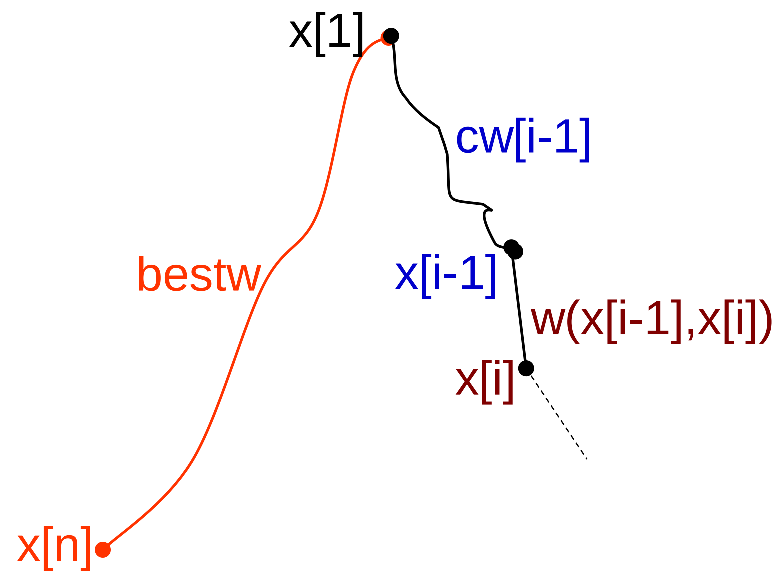
其中，x[1]=1;

假设：已经知道直到第i-1层的部分解

<1, x[2],x[3],…,x[i-1], ?, ? ,?>,

从第i-1层结点选择顶点x[i]，并向该分支往下搜索的界限函数为：

$$B(i) = cw(i\text{-}1) + w(x[i\text{-}1],x[i])$$



由此得到剪枝/约束条件2:

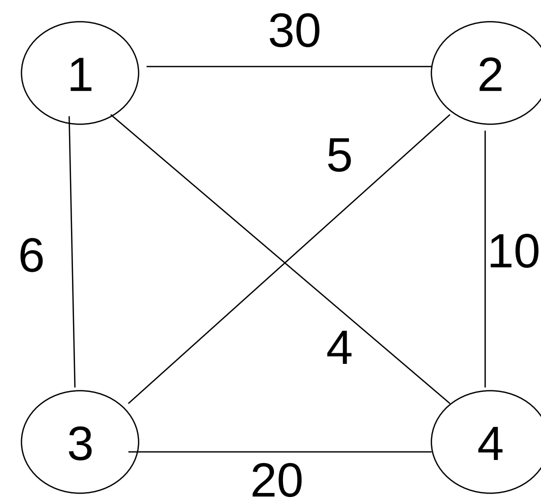
如果B(i) ≥ bestw, 则停止搜索x[i]分支及其下面的层，

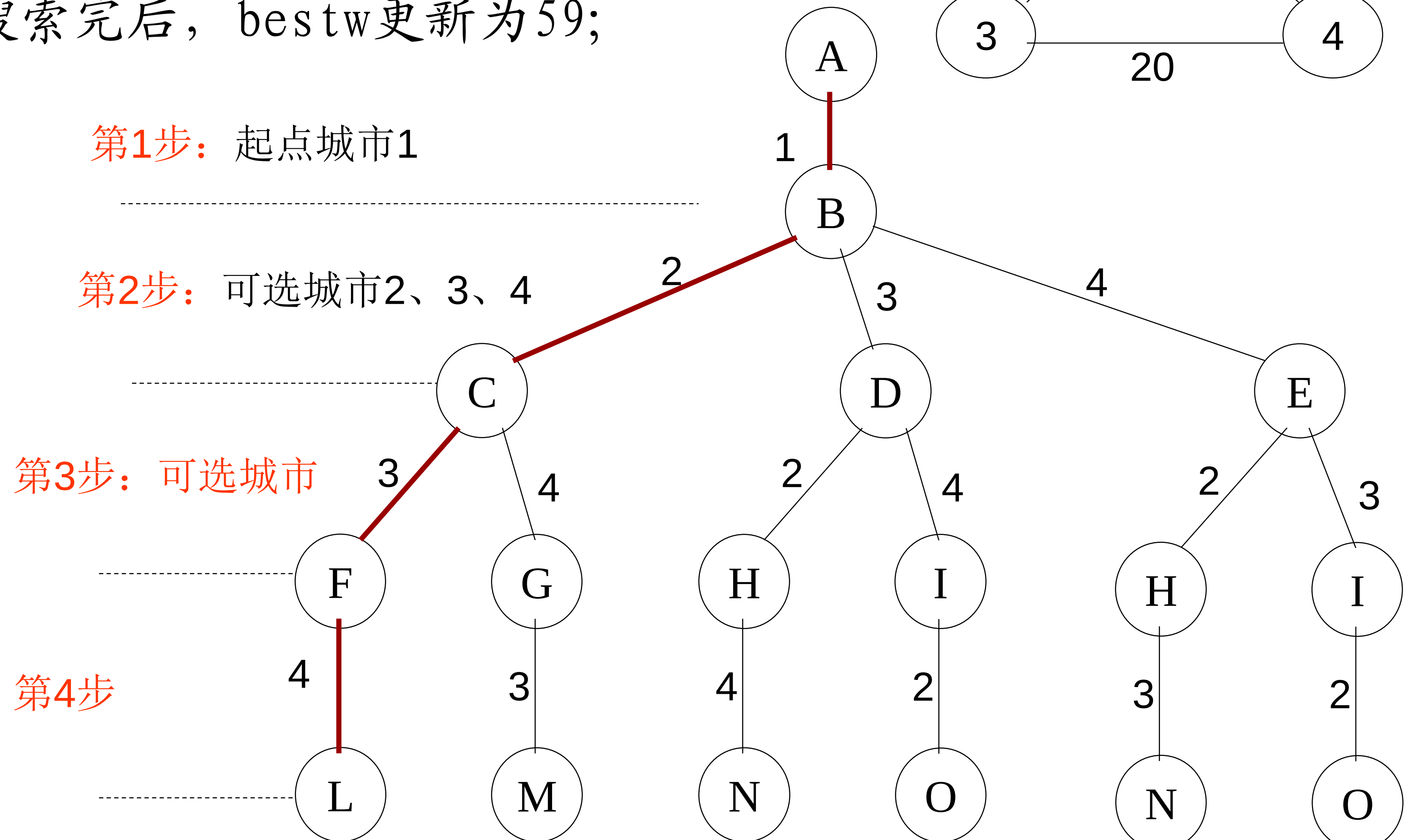
其中，bestw代表到目前为止,在前面的搜索中，从其它已经搜索过的路径中，找到的**最佳完整回路**的权和（总长度）

举例

对右图，在树空间中搜索。

1. 开始时，bestw=∞;
2. 深度优先得到第一条路径<1, 2, 3, 4, 1>。
   搜索该路径时，bestw不变。
   搜索完后，bestw更新为59;





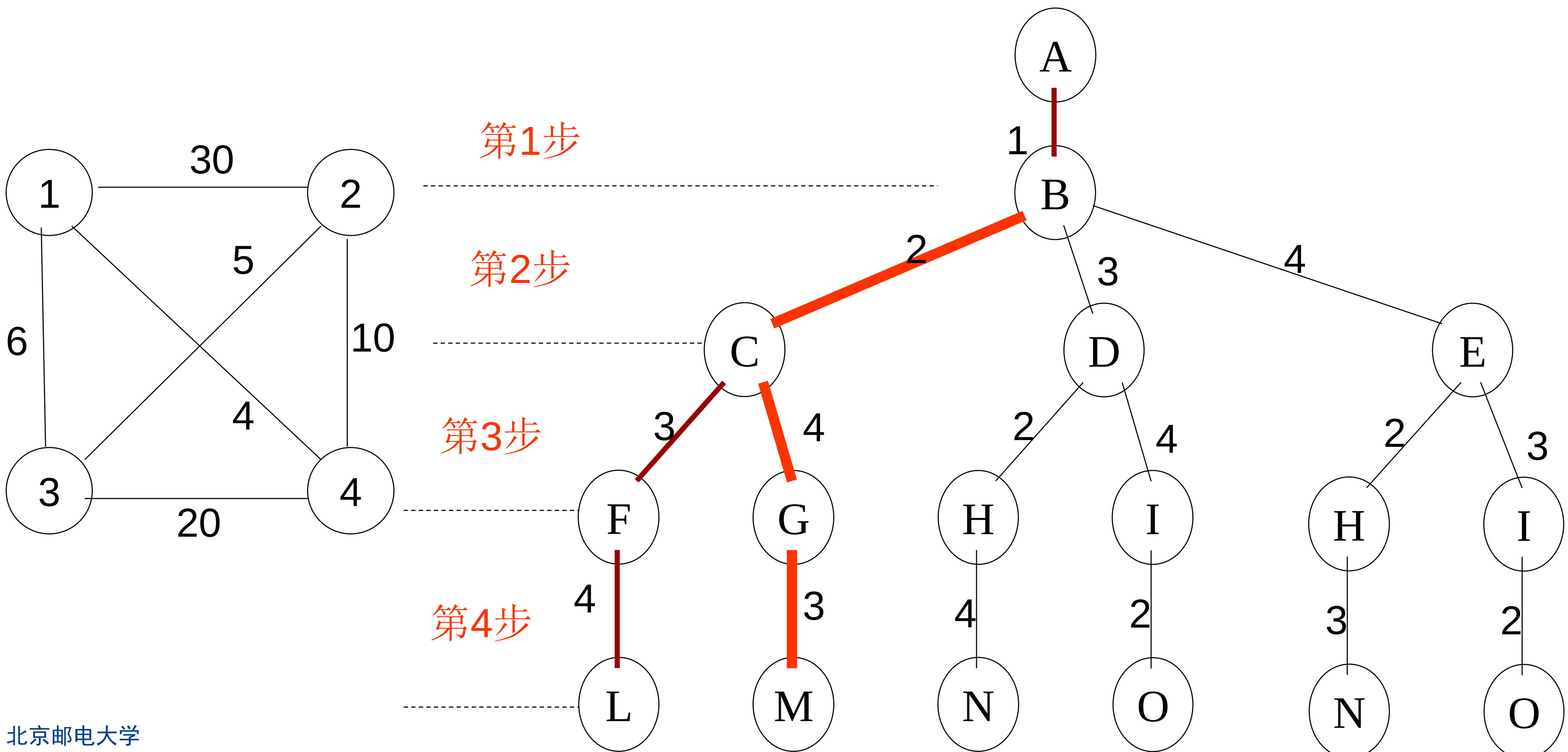
3. 按深度优先，搜索第2条路径<1, 2, 4, 3, 1>;

搜索过程中，不断比较、判断部分路径的费用≥bestw=59? 决定是否继续搜索下去。

对部分路径<1, 2, 4, ?>，其代价=30+10=40，可以搜索结点G下的分支;

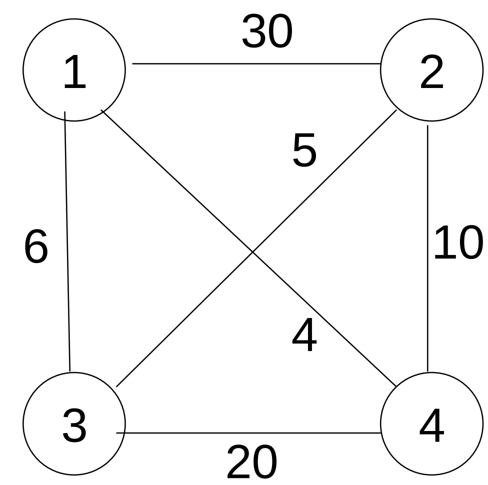
搜索完该分支后，该路径总长度=30+10+20+6=66 ≥bestw=59;

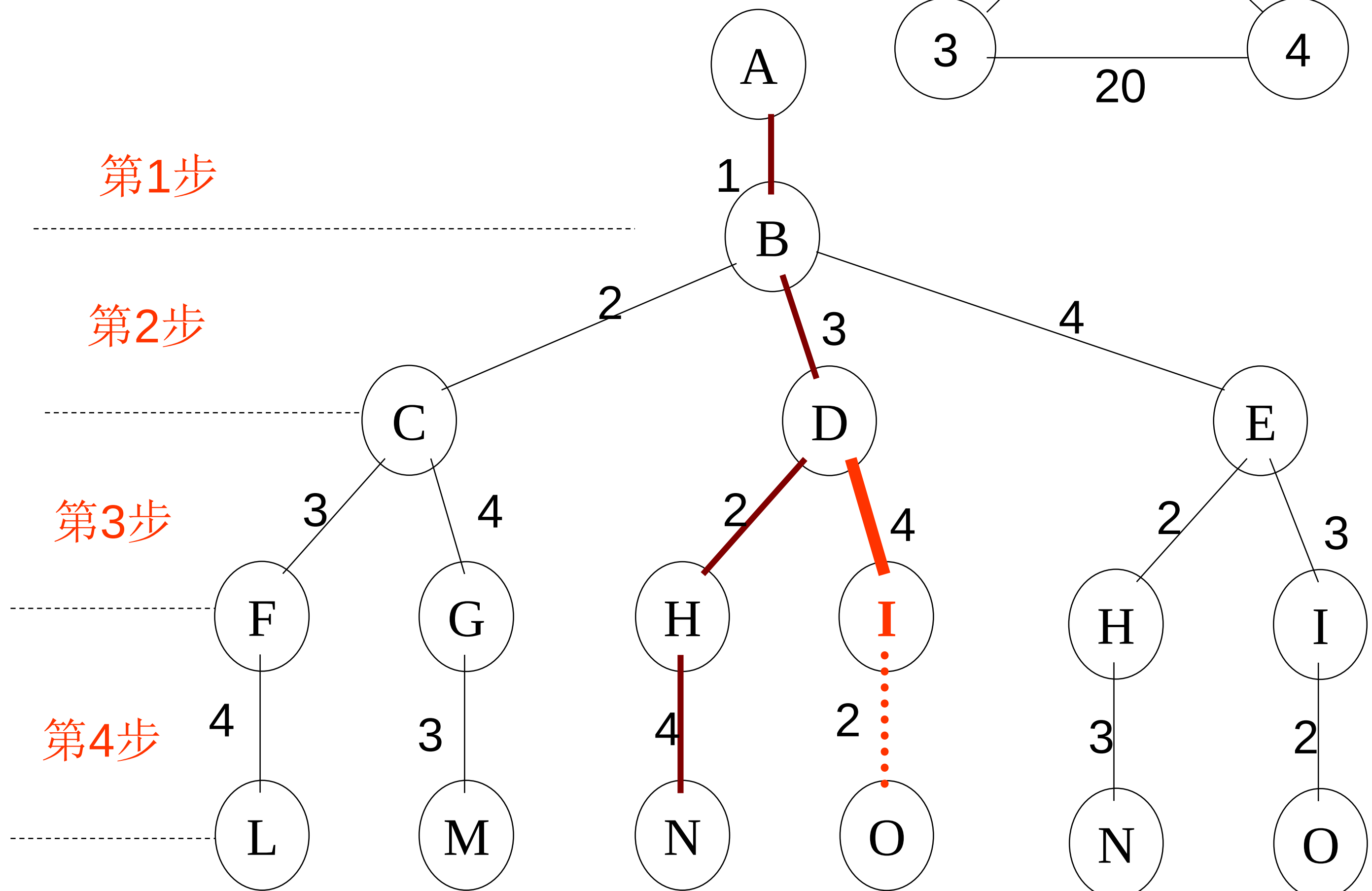
该路径不如以前找到的第一条路径，bestw不变。



4。该问题最优解：<1, 3, 2, 4, 1>，<1, 4, 2, 3, 1>，对应的bestw=25.

假设：搜索另一分支<1, 3, 4, ?>，当前路径对应的结点为I，长度=6+20=26 ≥bestw=25，结点I之下的路径被舍弃

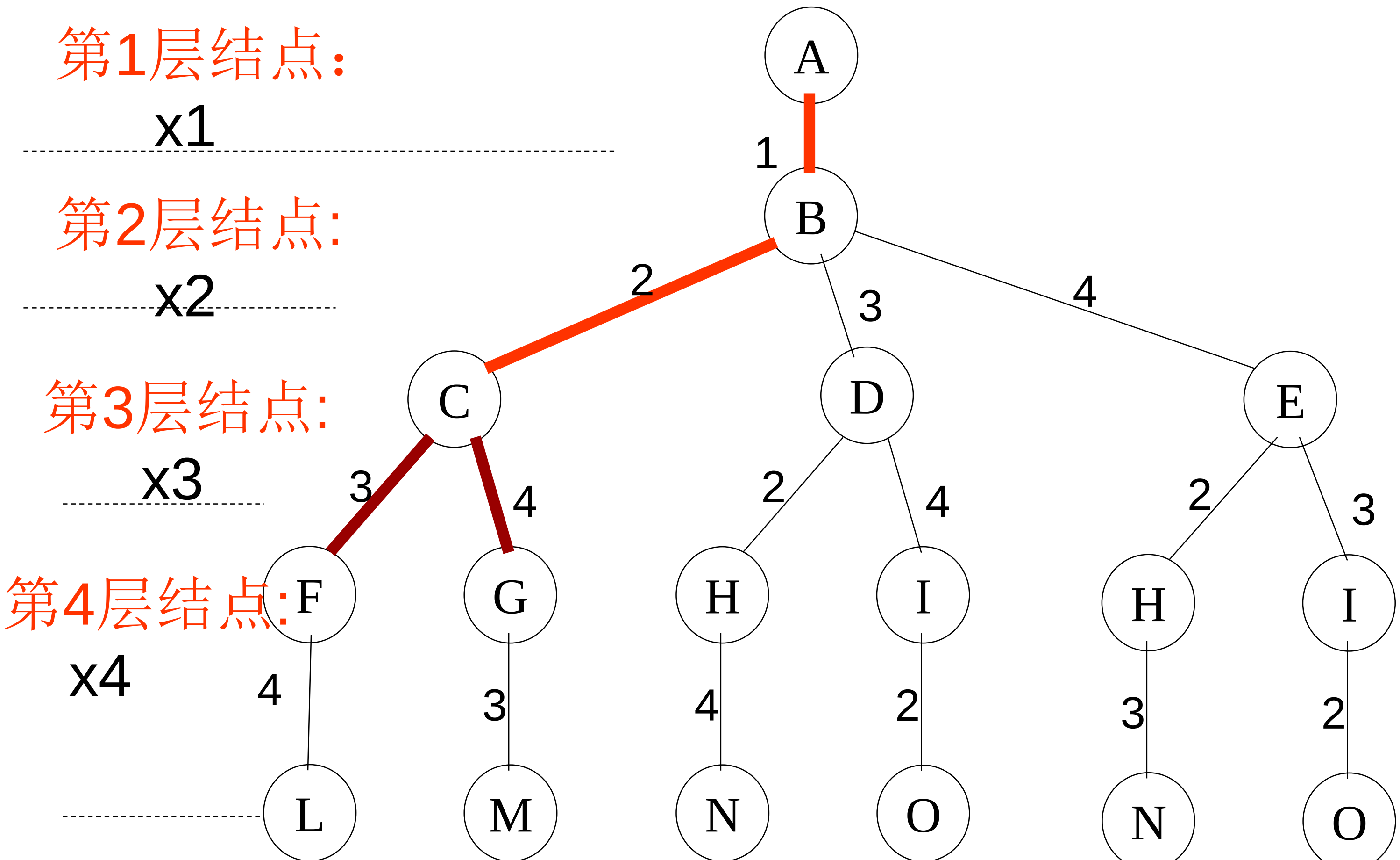




TSP问题的递归解法

BacktrackTSP(i):处理第i层结点，已经得到的部分解为:
<1, x[2],…, x[i-1], ?, …, ?>，需要选择第i个城市

e.g. i=3, 已经有<1, x[2], ?, ?>=<1,2,?,?>，需要在选则第3个城市。



bestx[1: n] : 记录最佳路径; cw: 当前部分路径的总长

BacktrackTSP(i)

1. if i=n then //已经搜索到叶节点，已经选择了最后1个城市

{

2. if w(x[n-1], x[n]) ≠∞ and w(x[n],1) ≠∞

// 最后1个城市与前一城市相连、与第1个城市相连，组成1个满足条件的回路

3. then //比较当前回路与以前最佳回路，看当前回路是否更优

4. if cw + w(x[n-1], x[n]) + w(x[n],1) < bestw

5. then { //当前回路更优，更新最优搜索结果

6. bestw = cw + w(x[n-1], x[n]) + w(x[n],1) ;

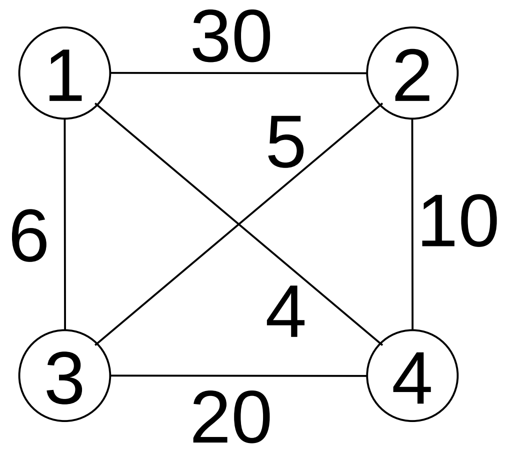
7. for j=1 to n do bestx[j]=x[j]

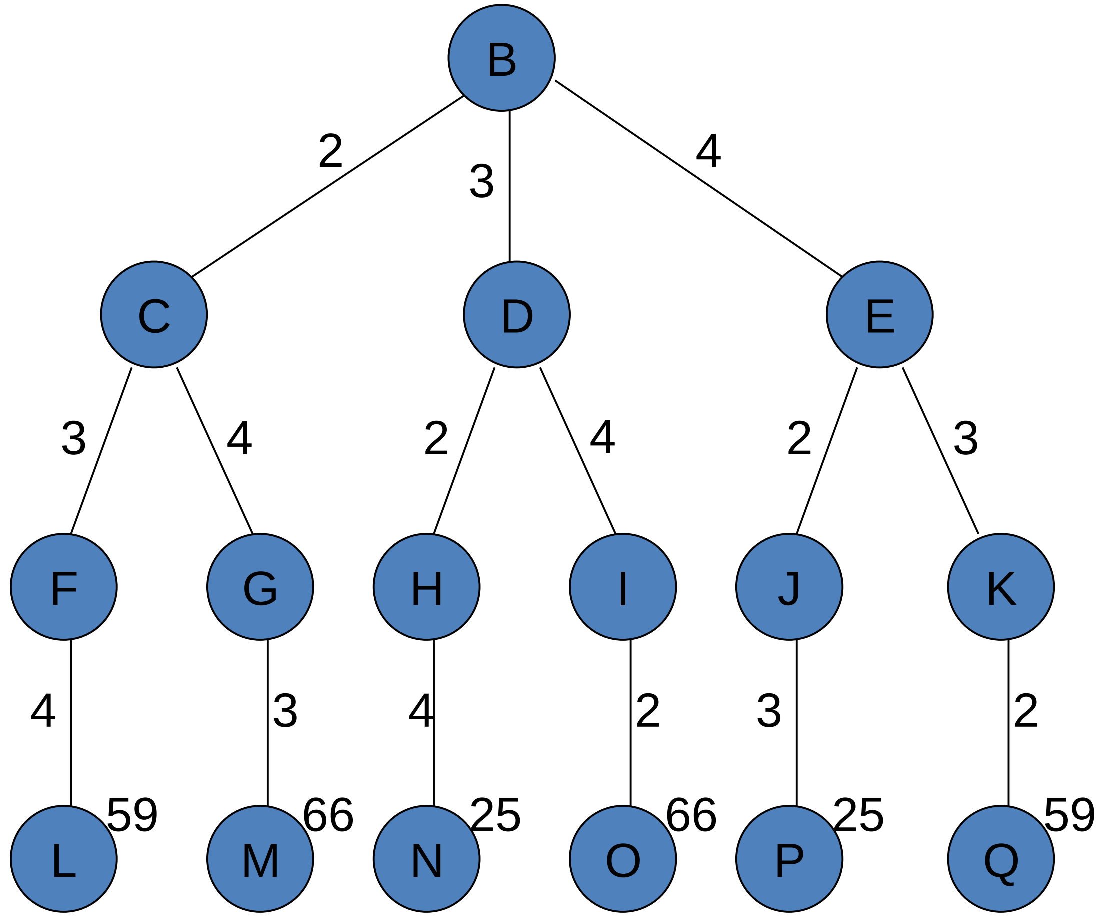
}

```
1.  if  i=n  then    //已经搜索到叶节点，已经选择了最后1个城市
      {
        ….
7.                       for j=1 to n  do  bestx[j]=x[j]
      }
8.      else  //如果搜索没有到叶节点, 当前得到的部分路径
                     <1, x[2],…, x[i-1], ?, …, ?>
        {
9.        for j=i to n   do  //考察x[i]的各个可能取值
10.           if w(x[i-1], x[j]) ≠∞ and cw + w(x[i-1], x[j])< bestw
              // 向当前部分路径加入新城市x[j]后，  x[j]与x[i-1]相
                连，且扩展后的部分路径<1, x[2],…, x[i-1], x[j], …, ?>
                  的成本小于当前最优回路的长度
11.              then  //继续扩展、搜索次路径
```

```
9.      for j=i to n   do  //考察x[i]的各个可能取值
10.         if w(x[i-1], x[j]) ≠∞ and cw + w(x[i-1], x[j])< bestw
            // 向当前部分路径加入新城市x[j]后， x[j]与x[i-1]相
              连，且扩展后的部分路径<1, x[2],…, x[i-1], x[j], …, ?>
                的成本小于当前最优回路的长度
11.           then   //继续扩展、搜索次路径
              {
12.             swap(x[i], x[j])         //加入第i个城市
13.             cw = cw + w(x[i-1], x[i]) //更新扩展后的路径的代价
14.             BacktrackTSP(i+1)        //递归搜索以x[i]为根的
                                          后续子树
15.             cw = cw - w(x[i-1], x[i]) //搜索失败，回溯，回到
                     第9步，为搜索x[i]的另一个取值x[j+1]准备
16.             swap(x[i], x[j])
                }
```

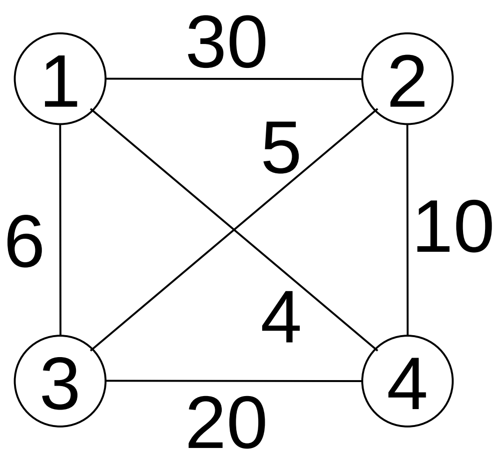
回溯法解旅行售货员问题

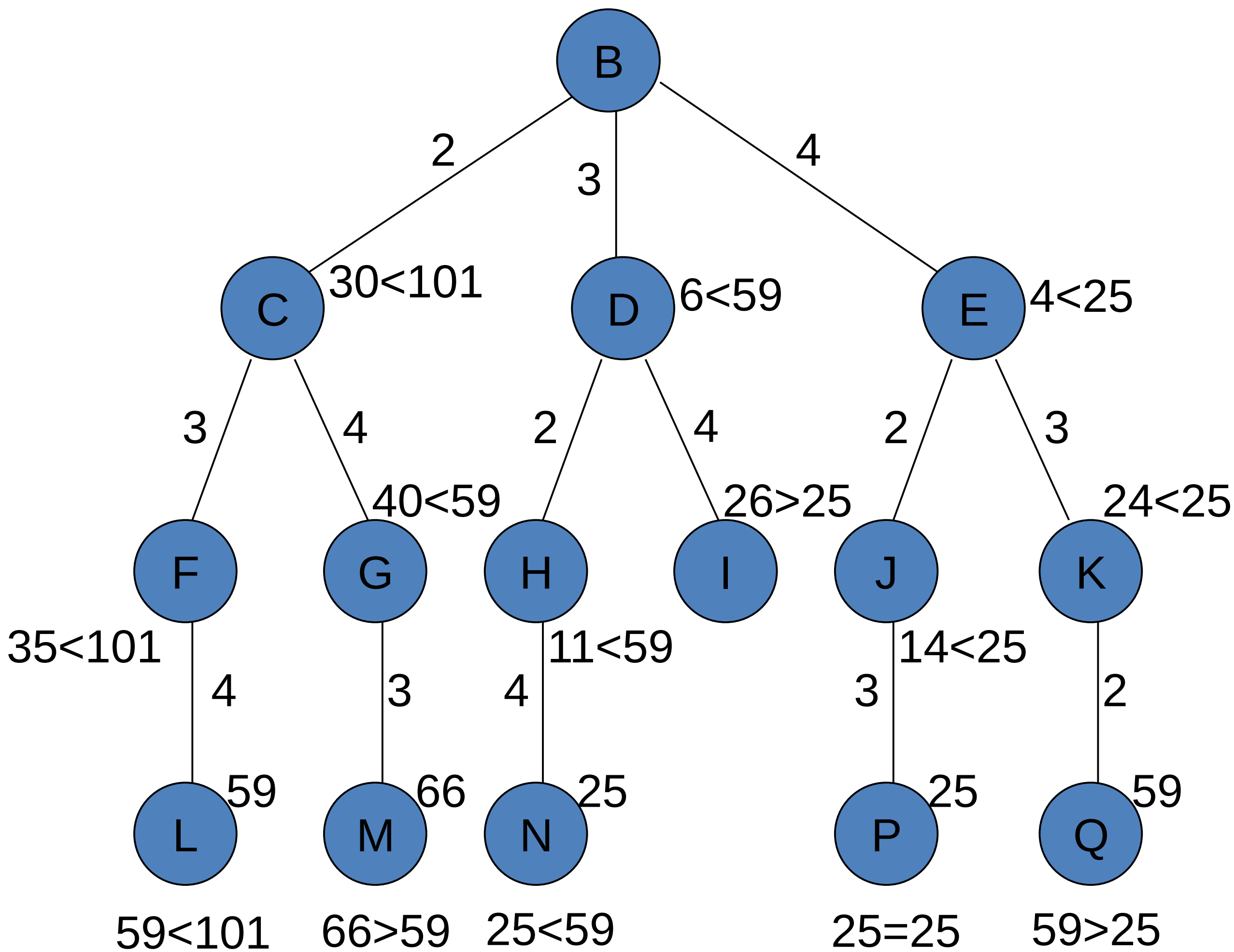




回溯法解旅行售货员问题

带下界函数：





编程作业

- N皇后问题
- 旅行售货员问题