第5章 回溯法

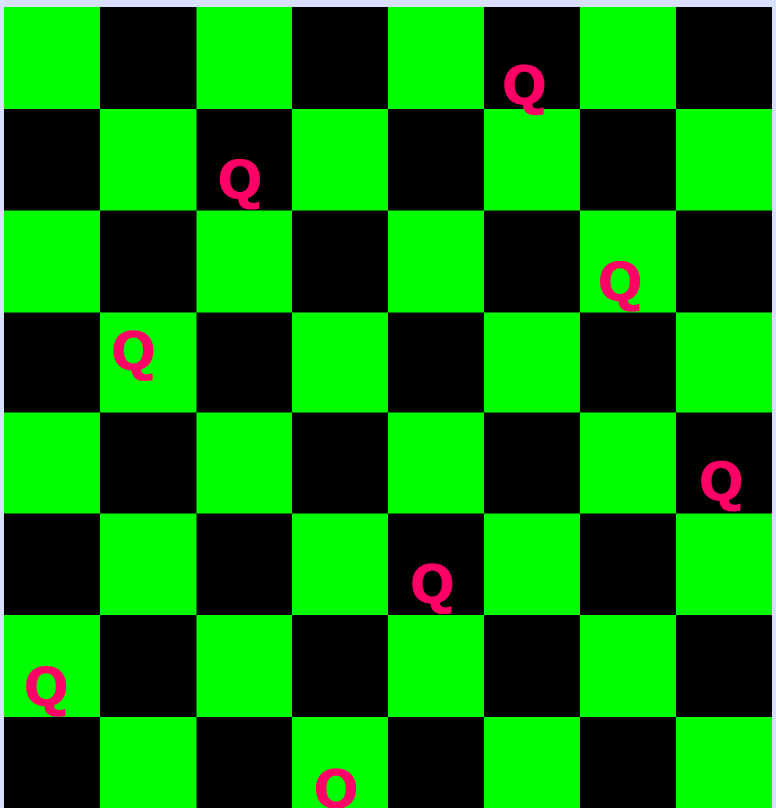
提　纲

- 回溯法的基本思想
- 装载问题
- *n*后问题
- 图的*m*着色问题
- 旅行售货员问题
- 总结

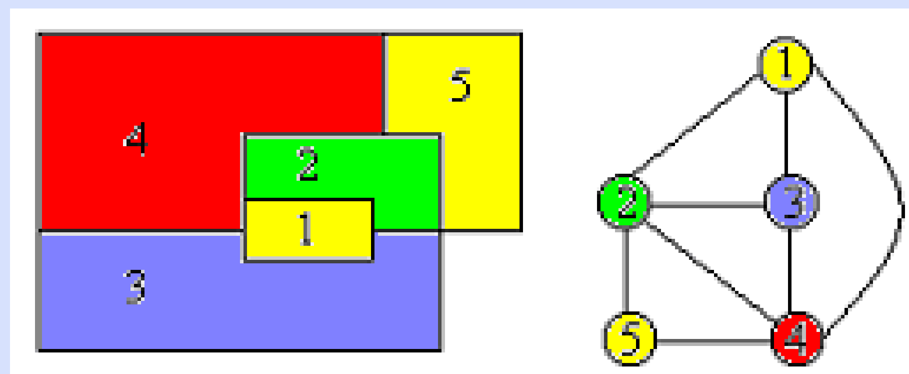
回溯法的基本思想

- 引例

八后问题

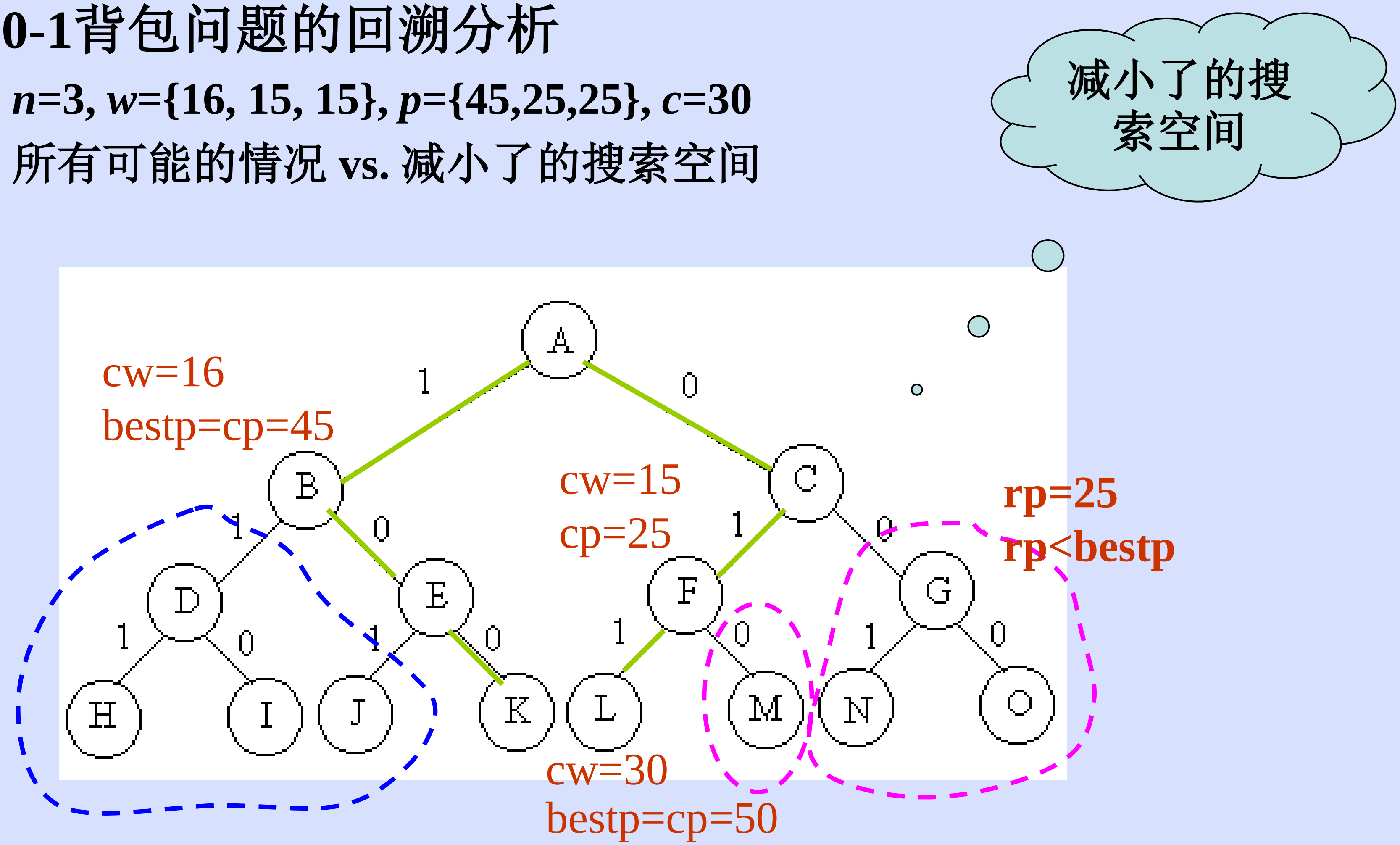


图的*m*着色问题



❖引例

- **0-1背包问题的回溯分析**
  - **- *n*=3, *w*={16, 15, 15}, *p*={45,25,25}, *c*=30**
  - **- 所有可能的情况 vs. 减小了的搜索空间**



❖基本思想

- 问题的提出
  - 很多问题不存在穷举搜索之外的解决该问题方法来
  - 很多问题通过穷举搜索数量巨大，但通过有限步，可以获得一个解
  - 产生开发系统化的搜索技术的需要，并希望能将搜索空间尽可能减少
  - 找出问题的解集、满足约束条件的最佳解、……
- 回溯法:以系统的方法隐含搜索所有可能解的技术
  - 有组织的搜索，常常可以避免搜索所有的可能性
  - 适用于解一些组合数（解空间）相当大的问题
  - 问题的解向量：回溯法希望一个问题的解能够表示成一个n元式 $(x_1, x_2, \ldots, x_n)$的形式

- 问题的解空间

  - 显约束：对分量x_i的取值限定
  - 解空间：解向量满足显式约束条件的所有元组；将解空间组织为树
  - 隐约束：为满足问题的解而对不同分量之间施加的约束

- 解空间树的生成

  - 深度优先的生成（搜索过程中动态生成）
  - 扩展结点、活结点、死结点

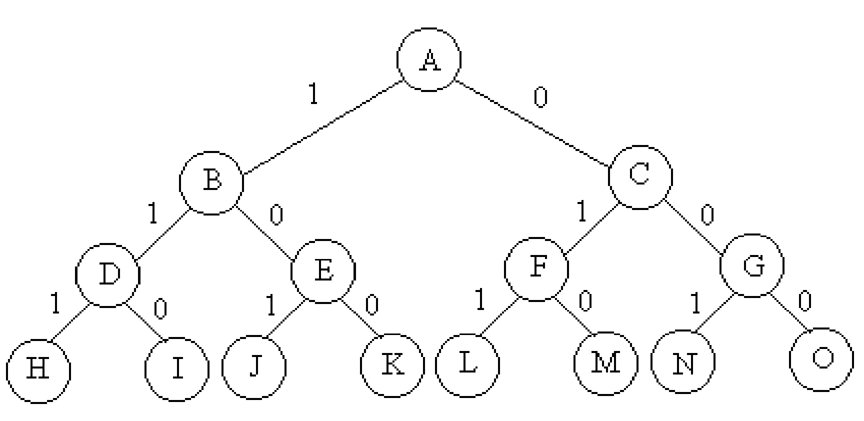
- 回溯法

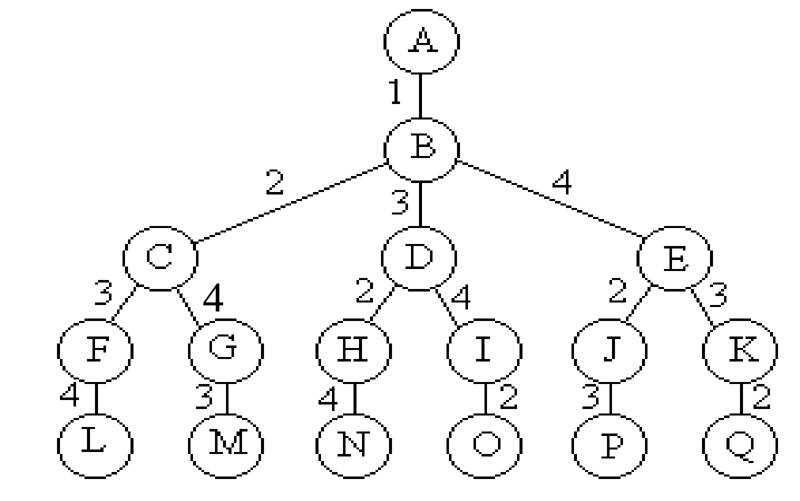
  - 避免无效搜索、提高效率——利用约束函数、限界函数来处死那些实际上不可能产生所需解的活结点，以减少问题的计算量

❖子集树与排列树

旅行售货员问题

0-1背包问题





遍历子集树需$O(2^n)$计算时间

```
void backtrack (int t){
  if(t>n) output(x);
  else
    for(int i=0;i<=1;i++){
      x[t]=i;
      if(legal(t))
      backtrack(t+1);
    }
}
```

遍历排列树需要$O(n!)$计算时间

```
void backtrack (int t){
  if(t>n) output(x);
  else
    for(int i=t;i<=n;i++){
      swap(x[t],x[i]);
      if(legal(t))
      backtrack(t+1);
      swap(x[t],x[i]);
    }
}
```

- **解题思路**
  - 确定解空间结构；
  - 深度优先搜索解空间+剪枝函数
- **常用剪枝函数**
  - 用约束函数在扩展结点处剪去不满足约束的子树（问题本身的约束）
  - 用限界函数剪去得不到最优解的子树（相对于已得到的解）
- **主要特征**
  - 不需要存储整棵搜索树，只需存储根到当前扩展结点的路径（隐式）
  - 设$h(n)$为从根到叶的最长路径长度
  - 显式：

    对于子集树解空间 —— $O(2^{h(n)})$

    对于排列树解空间 —— $O((h(n))!)$

提　纲

- 回溯法的基本思想
- 装载问题
- *n*后问题
- 图的*m*着色问题
- 旅行售货员问题
- 总结

装载问题

❖ 问题

- n个集装箱要装上两艘载重量分别为c_1和c_2的轮船，集装箱i重量为w_i
- 且：

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

问是否存在一种合理的装载方案将n个集装箱装上轮船？

(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。

$$\max \sum_{i=1}^{n} w_i x_i , \quad \text{s.t.} \ \sum_{i=1}^{n} w_i x_i \le c_1 , \quad x_i \in \{0,1\}, \quad 1 \le i \le n$$

- 回溯算法的基本思想

- 对于第一艘船的装载情况：一个集装箱要么装上、要么不装 — 1，0

解空间：子集树

- 剪枝函数：

（1）考查第j+1层扩展结点Z，考虑装载

$$\sum_{i=1}^{n} w_i x_i \le c_1.\text{即}cw = \sum_{i=1}^{j} w_i x_i, \text{若}cw + w_{j+1} > c_1$$

，则剪去Z的左子树（多米诺条件）

（2）不被装载：Z可扩展右子树（$bestw$为当前最优载重量）

$$cw = \sum_{i=1}^{j} w_i x_i, r = \sum_{i=j+2}^{n} w_i, \text{若}cw + r \le bestw$$

，则剪去右子树

- 装载问题的回溯算法

初始化：

```
for(int i=1; i<=n;i++)
 r=r+w[i];
```

调用backtrack：

```
cw=0; bestw=0;
backtrack(1);
return bestw;
```

计算时间：

- 搜索树叶结点数：2^n
- 时间复杂度：$O(2^n)$

```
void backtrack(int i)
 {
  if(i>n){
   if(cw>bestw)   bestw=cw;
   return;
  }
  r=r-w[i];   //第i+1到n的总量和
  if(cw+w[i]<=c){
    x[i]=1;
    cw=cw+w[i];
    backtrack(i+1);
    cw-=w[i];}
  if(cw+r>bestw){
    x[i]=0;
    backtrack(i+1);}
  r=r+w[i];
  }
```

提　纲

- 回溯法的基本思想
- 装载问题
- n后问题
- 图的m着色问题
- 旅行售货员问题
- 总结

n后问题

❖ 问题

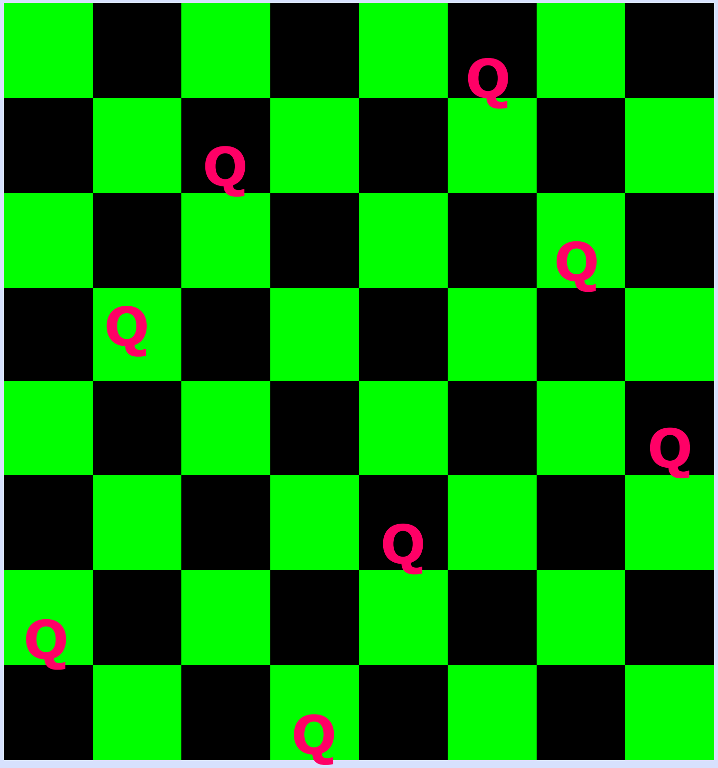
- 在$n \times n$格的棋盘上放置彼此不受攻击的n个皇后
- n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上

❖ 算法思想

- 解空间：完全n叉树
- 解向量：$(x_1, x_2, \ldots, x_n)$
- 每行放一个皇后， $x[i]$表示皇后i被放在第i行$x[i]$列，约束：

(1) $x[i] \neq x[j], j \in [1, i-1]$

(2) $|i - j| \neq |x[i] - x[j]|, j \in [1, i-1]$



n皇后问题的回溯算法

- 初始化：
  ```
  for(i=0;i<=n;i++) x[i]=0
  ```
- 调用：
  ```
  sum=0;
  backtrack(1)
  ```

```
boolean place (int k)
{
  for(int j=1;j<k;j++)
   if(|k-j|==|(x[j]-x[k]|) or |x[j]==x[k]|) return false;
   return true;
}

void backtrack (int t)
{
  if(t>n) sum++;
  else
    for(int i=1;i<=n;i++) {
      x[t]=i;
      if(place(t)) backtrack(t+1);
    }
}
```

- 时间复杂度：$O(n^n)$
- 和穷举方法相比？

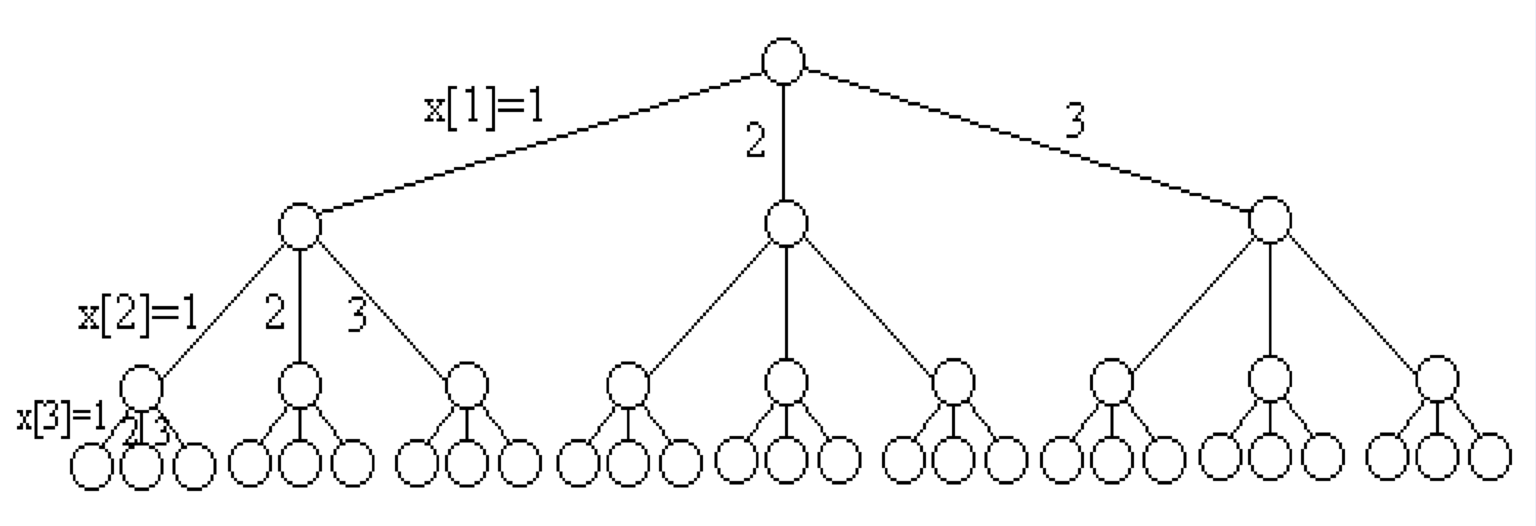
提　纲

- 回溯法的基本思想
- 装载问题
- *n*后问题
- 图的*m*着色问题
- 旅行售货员问题
- 总结

图的*m*着色

❖ 问题

- 用*m*种不同的颜色给无向图***G*=(*V*, *E*)**的各顶点着色，相邻顶点颜色不重复
- ***G***是***m***可着色的，找出不同着色法；不是则给出否定回答
- 例如: |*V*|=3, *m*=3的解空间树



- **算法思想**
  - 解空间树：完全m叉树
  - 解向量：$(x_1, x_2, \ldots, x_n)$，x_i表示顶点i所着颜色$x[i]$
  - 约束：顶点i与已着色的相邻顶点颜色不重复

    即：所有G中存在的边(i, j)，$x[i]\neq x[j]$，$j\in[1, n]$

```
boolean ok(int k)
{
 for(int j=1;j<=n;j++)
  if(a[k][j]&&(x[j]==x[k]))
   return false;
  return true;
}
```

```
void backtrack(int t){
 if (t>n) sum++;
 else
  for(int i=1;i<=m;i++){
   x[t]=i;
   if(ok(t))backtrack(t+1);
  }
}
```

计算时间

- 搜索树结点数：$O(m^n)$, 每个结点的颜色可用性检查 $O(n)$
- 时间复杂度：$O(nm^n)$

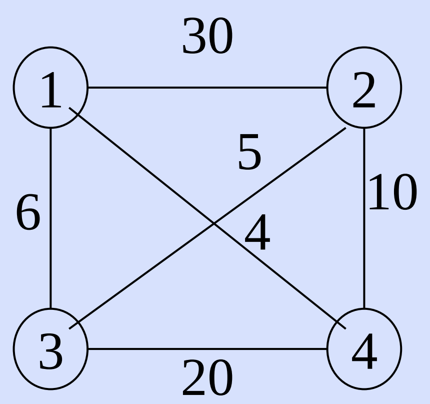
提 纲

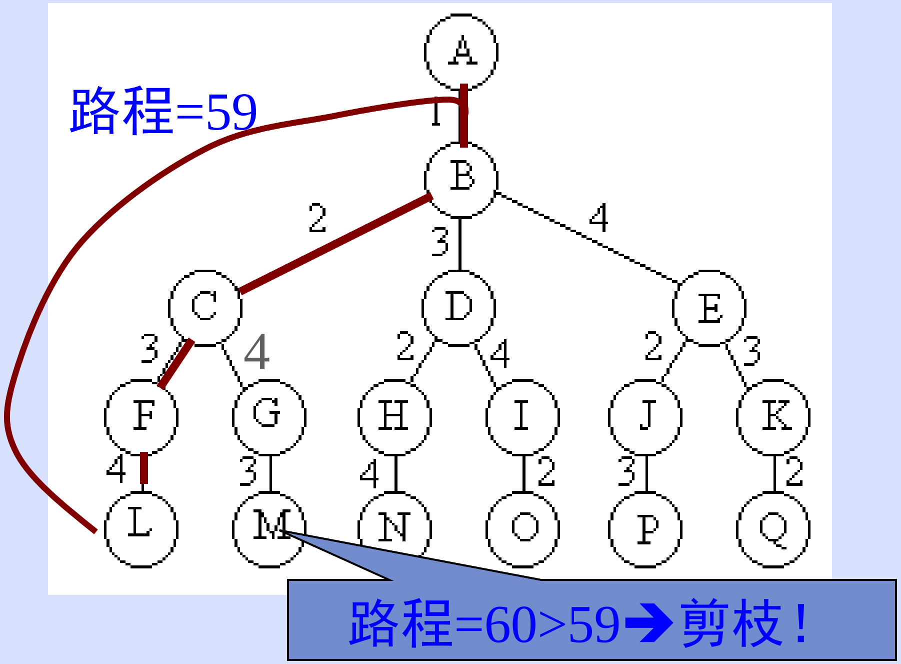
- 回溯法的基本思想
- 装载问题
- n后问题
- 图的m着色问题
- 旅行售货员问题
- 总结

旅行售货员问题

❖ 问题

- 售货员要到若干城市（图顶点）去推销商品，已知各城市之间的路程（旅费），选定一条从驻地（顶点1）出发，经过每个城市一次，最后回到住地（顶点1）的路线，使总的路程（旅费）最短（最小）
- 解空间：排列树





- 算法思想

- 解向量: ($x_1=1$, x_2, x_3, …, x_n, $x_1=1$)，长度为 $n+1$

- 考查第i层结点，约束和界：

$x[1:i]$的路程（旅费）< 当前最优值

$i=n$时，当前扩展结点是排列树的叶结点的父结点

(1) G中边$(x[n-1], x[n])$存在

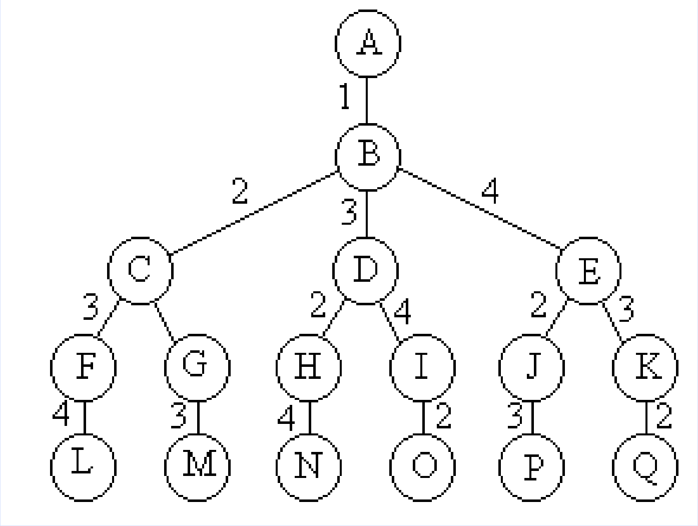
(2) G中边$(x[n], x[1])$存在

(3) 回路路程<当前已找到最优回路路程

旅行售货员问题的回溯算法

```
void backtrack(int i){
  if (i == n) {
   if ((x[n-1],x[n]）∈ E and (x[n],1)∈ E and
   cc+a[x[n-1],x[n]]+a[x[n],1]<bestc)){
     for(int j=1;j<=n;j++) bestx[j]=x[j];
     bestc = cc+a[x[n-1],x[n]]+a[x[n],1];
   }
  }else{
   for(int j=i;j<=n;j++)
    if ((x[i-1],x[j])∈ E and cc+a[x[i-1],x[j]]<bestc)){
     swap(x,i,j);
     cc+=a[x[i-1],x[i]];
     backtrack(i+1);
     cc-=a[x[i-1],x[i]];
     swap(x,i,j);
  }
}
```

- 调用：
 backtrack(2)



cc：当前搜索的部分路径长度，初始化值为0

bestc：已经求得的最好周游路径，初始值是系统最大浮点数

bestx：记录bestc具体路径（顶点序列）

计算时间

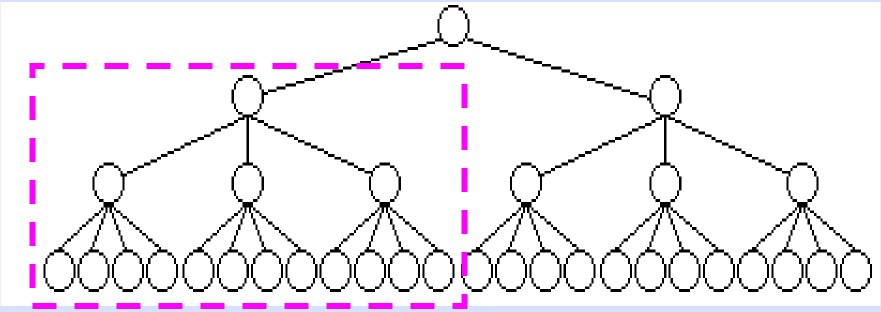
- 搜索树结点数：$O((n-1)!)$
- 时间复杂度：$O(n(n-1)!)=O(n!)$

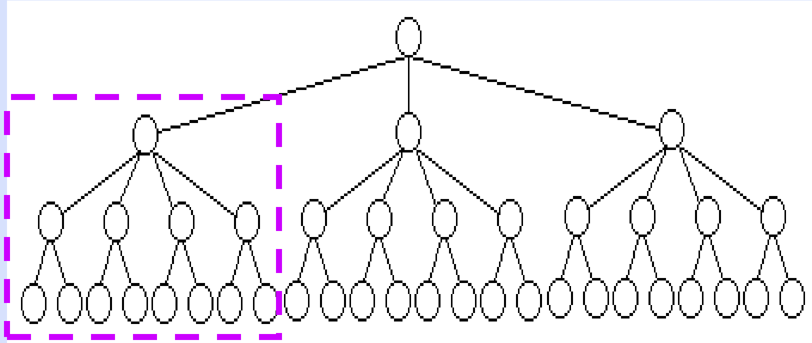
提　纲

- 回溯法的基本思想
- 装载问题
- n后问题
- 图的m着色问题
- 旅行售货员问题
- 总结

总结

- 回溯法效率分析
  - 好的约束函数能显著地减少所生成的结点数，往往计算量较大
  - 考虑生成结点数与约束函数计算量之间的折衷
- 重排原理
  - 尽可能减小搜索空间
  - 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
  - 在其他条件相当的前提下，让可取值最少的*x*[*i*]优先






谢谢大家！