课程内容

NP完全性理论与近似算法	算法高级理论
随机化算法；线性规划与网络流	高级算法
递归分治；动态规划；贪心算法；回溯与分支限界	基础算法
算法分析与问题的计算复杂性	算法基础理论

第五章 回溯法

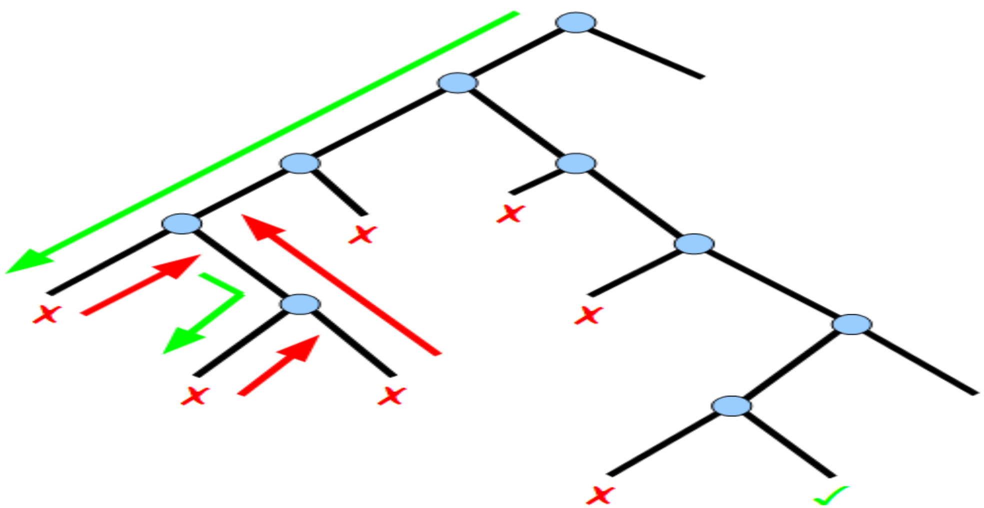
学习要点

- 理解回溯法的深度优先搜索策略
- 掌握用回溯法解题的算法框架
  - 递归回溯最优子结构性质
  - 迭代回溯贪心选择性质
  - 子集树算法框架
  - 排列树算法框架
- 通过应用范例学习回溯法的设计策略
  - *n*后问题、0-1背包问题 、旅行售货员问题
  - 装载问题
  - 图的着色问题

回溯法概述

回溯法

- 有许多问题，当需要找出它的解集或者要求回答什么解是满足某些约束条件的最佳解时，往往要使用回溯法。回溯法号称计算机算法中最通用的算法！
- 回溯法的基本做法是搜索，或是一种组织得井井有条的(带有系统性)，能避免不必要搜索(带有跳跃性)的穷举式搜索法。这种方法适用于解一些组合数相当大的问题。



回溯问题的解空间

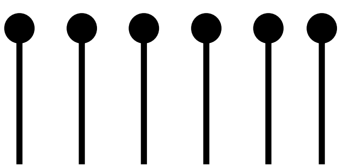
- **问题的解向量：** 回溯法希望一个问题的解能够表示成一个*n*元向量$(x_1, x_2,\ldots, x_n)$的形式
  - **显约束：** 对分量x_i的取值限定
  - **隐约束：** 为满足问题的解而对不同分量之间施加的约束
  - **解空间：** 对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间
- **注意：** 同一个问题可以有多种表示，有些表示方法更简单，所需表示的状态空间更小(存储量少，搜索方法简单)

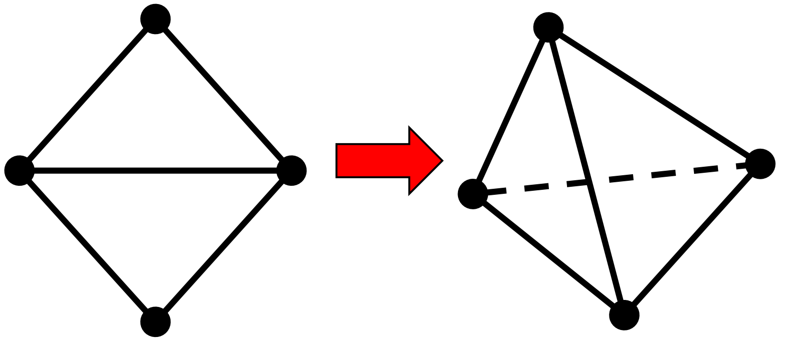
最优化问题（回顾Chpater3）

- 有n个输入（**解空间**），问题的解由这n个输入的一个子集组成，这个子集必须满足某些事先给定的条件，这些条件称为**约束条件**，满足约束条件的解称为问题的**可行解**。
- 满足约束条件的可行解可能不只一个，为了衡量这些可行解的优劣，事先给出一定的标准，这些标准通常以函数的形式给出，这些标准函数称为**目标函数**，使目标函数取得极值（极大或极小）的可行解称为**最优解**。
- 这类问题就称为**最优化问题**。

回溯问题的解空间

- **例如：** 有6根火柴，以之为边搭建4个等边三角形
  - 该问题易产生误导，它暗示是一个二维空间，为解决问题需拓展到三维。





  - 对任意一个问题，解的表示方式和它相应的解隐含了解空间及其大小。

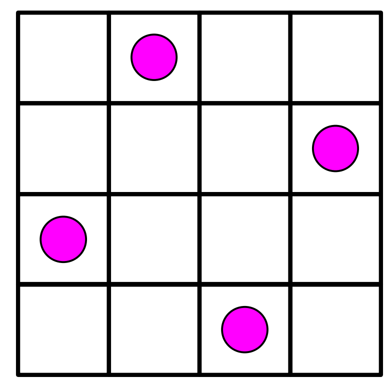
几个回溯法的例子

4后问题

问题：在4×4的方格棋盘上放置4个皇后，使得没有两个皇后在同一行、同一列、也不在同一条45度的斜线上。问有多少种可能的布局？

解是4维向量：$\langle x_1, x_2, x_3, x_4 \rangle$

解：⟨2,4,1,3⟩，⟨3,1,4,2⟩

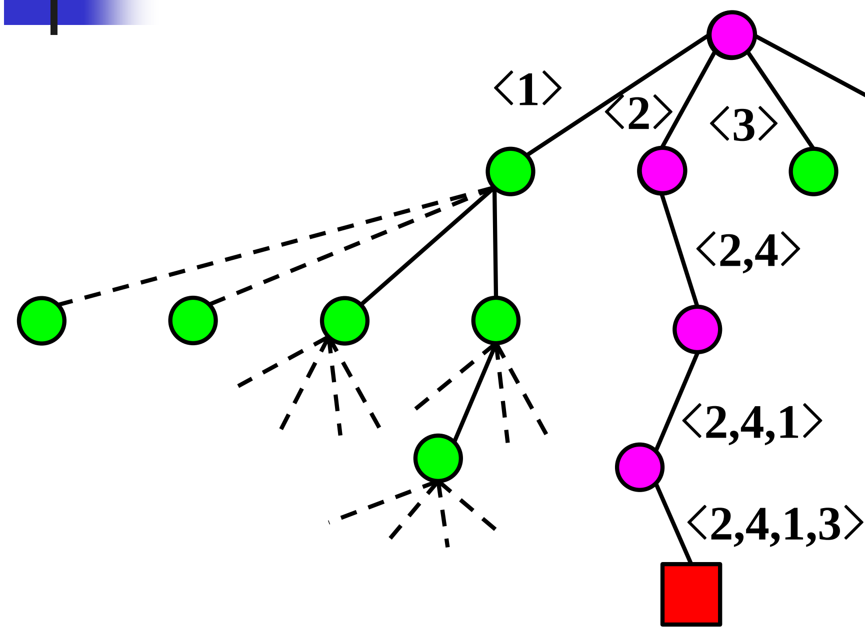


推广到8后问题

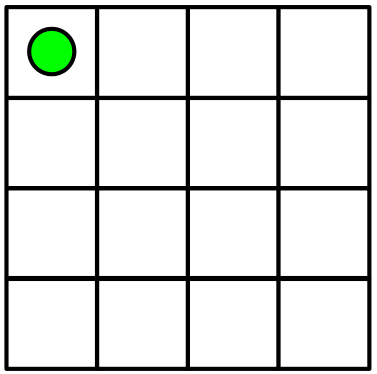
解：8维向量，有92个。

例如：⟨1,5,8,6,3,7,2,4⟩是解

搜索空间：4叉树



第1层代表第1个皇后可能选择的列号



- 每个结点有4个儿子，分别代表选择1，2，3，4列位置
- 第*i*层选择解向量中第*i*个分量值
- 最深层的树叶是解
- 按深度优先次序遍历树，找到所有解

0-1背包问题

问题： 有n种物品，每种物品的重量和价值分别为w_i，v_i。如果背包的最大承重限制是B，每种物品至多放1个。怎么样选择放入背包的物品使得背包所装物品价值最大？

实例： V={12,11,9,8}, W={8,6,4,3}, B=13

最优解： ⟨0,1,1,1⟩，价值：28，重量：13

算法设计

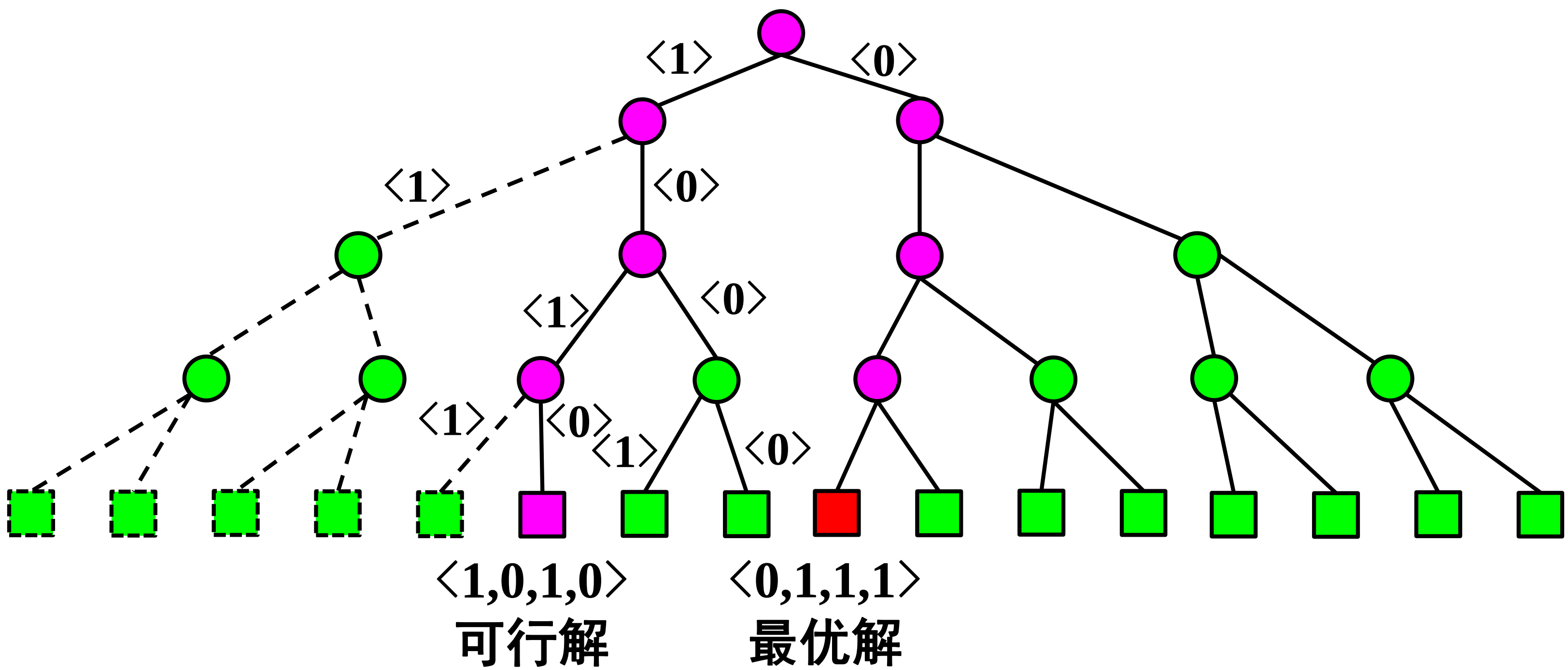
- **解**：n维0-1向量$\langle x_1, x_2, \ldots, x_n\rangle$
  $x_i = 1 \Leftrightarrow$ 物品i选入背包
- **结点**：$\langle x_1, x_2, \ldots, x_k\rangle$（部分向量）
- **搜索空间**：一棵0-1取值的二叉树，称为**子集树**，有2^n片树叶
- **可行解**：满足约束条件(不超重)的解
- **最优解**：可行解中价值达到最大的解

实例

- **输入：** **V={12,11,9,8}, W={8,6,4,3}, B=13**
- **2个可行解：**
  - **〈0,1,1,1〉，价值：28，重量：13**
  - **〈1,0,1,0〉，价值：21，重量：12**
- **最优解：〈0,1,1,1〉**

搜索空间

- 实例：V={12,11,9,8}, W={8,6,4,3}, B=13
- 搜索空间：子集树，2^n片树叶



旅行售货员问题

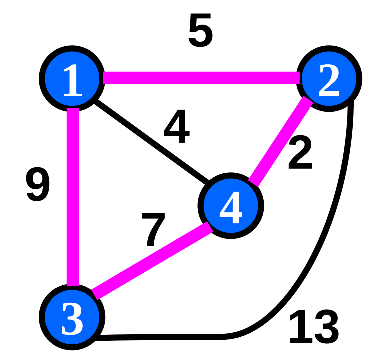
- **问题：** 一个售货员需要在n个城市销售商品，已知任两个城市之间的距离，求一条每个城市恰好经过一次的回路，使得总长度最小。
- **建模：** 城市集$C=\{c_1,c_2,\ldots,c_n\}$,距离$d(c_i,c_j)=d(c_j,c_i)$
- **求解：** $1,2,\ldots,n$的排列$k_1,k_2,\ldots,k_n$使得

$$\min\left\{\sum_{i=1}^{n-1} d(c_{k_i},c_{k_{i+1}})+d(c_{k_n},c_{k_1})\right\}$$

实例

输入：

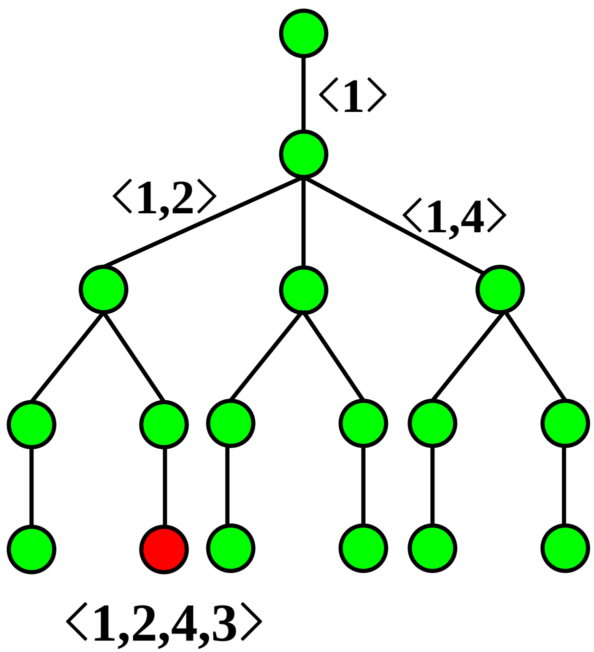
- C={1,2,3,4}
- $d(1,2)=5$, $d(1,3)=9$
- $d(1,4)=4$, $d(2,3)=13$
- $d(2,4)=2$, $d(3,4)=7$

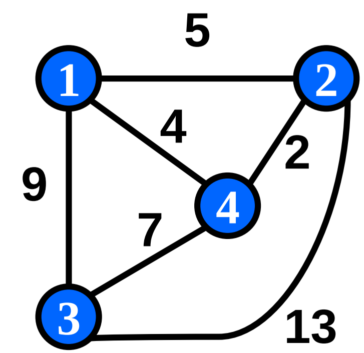


解：〈1,2,4,3〉，长度=5+2+7+9=23

搜索空间

- 排列树，有$(n-1)!$片树叶



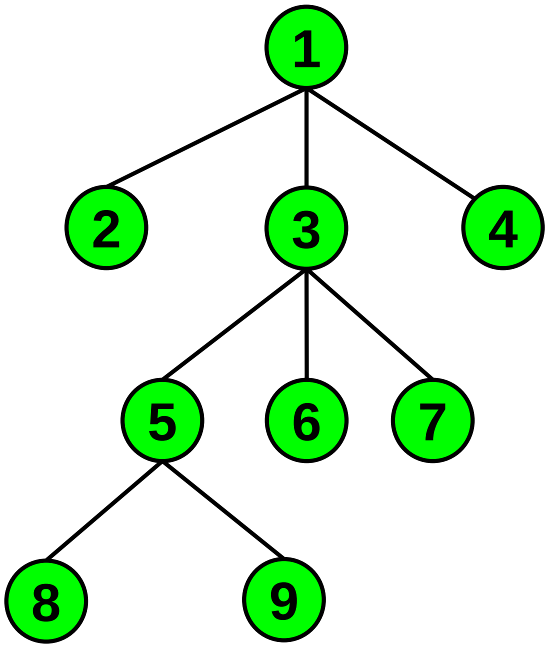


小结

问题	解性质	解向量	搜索空间	搜索方式	约束条件
n后问题	可行解	$\langle x_1, x_2, ..., x_n\rangle$ x_i:第i行列号	n叉树	深度/宽度优先	彼此不攻击
0-1背包问题	最优解	$\langle x_1, x_2, ..., x_n\rangle$ x_i=0/1 $x_i=1\Leftrightarrow$选i	子集树	深度/宽度优先	不超过背包重量
TSP问题	最优解	$\langle k_1, k_2, ..., k_n\rangle$ 1,2,...,n的排列	排列树	深度/宽度优先	选未经过的城市
特点	搜索解	扩张部分向量	树	跳跃式遍历	约束条件回溯判定

回顾一下

- 深度与宽度优先搜索



深度优先访问顺序：

1→2→3→5→8→9→6→7→4

宽度优先访问顺序：

1→2→3→4→5→6→7→8→9

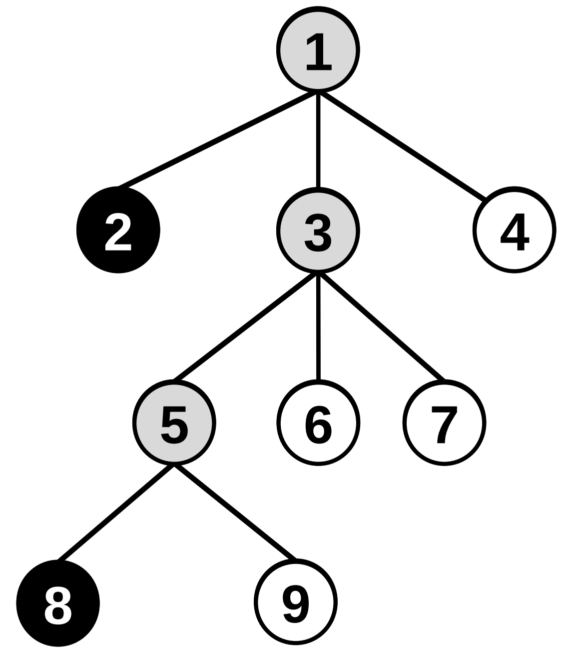
回溯法基本设计思想

- **适用对象：** 求解搜索问题和优化问题
- **搜索空间：** 树，结点对应部分解向量，可行解在树叶上
- **搜索过程：** 采用系统的方法隐含遍历搜索树
- **搜索策略：** 深度/宽度优先、函数优先、宽深结合
- **结点分支判定条件：**
  - 满足约束条件—分支扩张解向量
  - 不满足约束条件—回溯到该结点的父结点
- **结点状态：** 动态生成
  - 可以约定：白色结点（尚未访问）、灰色结点（访问过）、黑色结点（该结点为根的子树遍历完成---访问过且不再访问）
- **存储：** 当前路径

结点状态例子

- 策略：深度优先
- 访问次序：

  1→2→3→5→8



- 已完成访问：2,8
- 已访问但未结束：1,3,5
- 尚未访问：9,6,7,4

回溯法适用条件

- 在结点$\langle x_1, x_2, ..., x_k\rangle$处

  $P(x_1, x_2, ..., x_k)$为真

  $\Leftrightarrow$向量$\langle x_1, x_2, ..., x_k\rangle$满足某个性质（约束条件）

(例如：n后中k个皇后放在彼此不攻击的位置)

- 多米诺性质

  $$P(x_1, x_2, ..., x_{k+1}) \rightarrow P(x_1, x_2, ..., x_k) \quad 0 < k < n$$

- 逆否命题：

  $$\neg P(x_1, x_2, ..., x_k) \rightarrow \neg P(x_1, x_2, ..., x_{k+1}) \quad 0 < k < n$$

  作用：k维向量不满足约束条件，扩张向量到k+1维仍旧不满足，可以回溯！

一个实例

例：求不等式的整数解

$$5x_1+4x_2-x_3\leqslant 10,\ 1\leqslant x_k\leqslant 3,\ k=1,2,3$$

$P(x_1, x_2, ..., x_k)$：将$x_1, x_2, ..., x_k$代入原不等式的相应部分，部分和小于等于10

➔不满足多米诺性质：

$$5x_1+4x_2-x_3\leqslant 10 \nRightarrow 5x_1+4x_2\leqslant 10$$

变换使得问题满足多米诺性质：

令 $x_3=3-x_3^*$，那么原不等式变换为

$$5x_1+4x_2+x_3^*\leqslant 13$$

$$1\leqslant x_1, x_2\leqslant 3,\ 0\leqslant x_3^*\leqslant 2$$

小结

- 回溯法适用条件：多米诺性质
- 回溯法设计步骤
  1. 定义解向量和每个分量的取值范围
     - 解向量为$\langle x_1, x_2, ..., x_n \rangle$
     - 确定x_i的取值集合为X_i
  2. 由$\langle x_1, x_2, ..., x_{k-1} \rangle$确定如何计算$x_k$取值集合$S_k$, $S_k \subseteq X_k$
  3. 确定结点儿子的排列规则
  4. 判断是否满足多米诺性质
  5. 确定每个结点分支的约束条件
  6. 确定搜索策略：深度优先/宽度优先，...
  7. 确定存储搜索路径的数据结构

算法分析与设计

Analysis and Design of Algorithm

Lesson 13

要点回顾

- 回溯法设计要素
  - 适用对象：求解搜索问题和优化问题，通用算法的美称
  - 搜索空间：树--n叉树、子集树、排列树
  - 搜索过程：采用系统的方法（深度优先）隐含（带有跳跃性）遍历搜索树
  - 结点分支判定条件：
    - 满足约束条件——分支扩张解向量
    - 不满足约束条件——回溯到该结点的父结点
  - 回溯法的适用条件：
    - 满足多米诺性质

回溯法的实现及实例

回溯法两种实现

- **迭代实现**
- 算法Backtrack(n)
- 输入：n
- 输出：所有解

1. 对于i=1,2,...,n确定X_i
2. $k \leftarrow 1$
3. 计算S_k
4. **while** $S_k \neq \emptyset$ **do**
5. $\quad x_k \leftarrow S_k$中最值；$S_k \leftarrow S_k - \{x_k\}$
6. $\quad$**if** $k<n$ **then**
7. $\quad\quad k \leftarrow k+1$; 计算$S_k$
8. $\quad$**else** $\langle x_1, x_2, ..., x_n \rangle$是解
9. **if** $k>1$ **then** $k \leftarrow k-1$; **goto** 4

确定初始取值

满足约束分支搜索

回溯

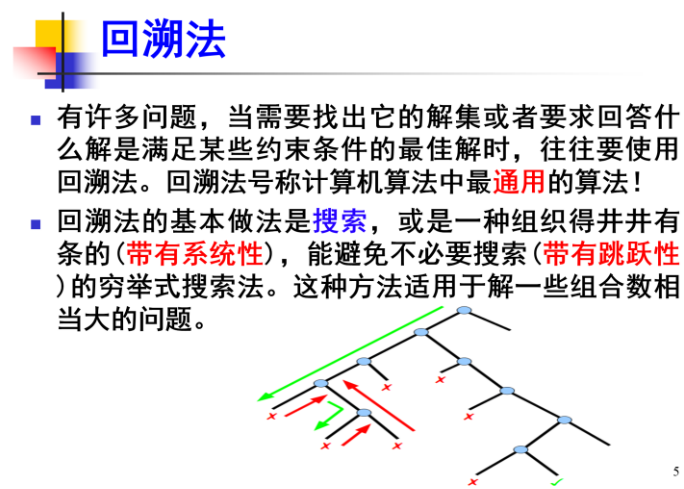
回溯法两种实现

- 递归实现
- 算法ReBack(k)
  1. if $k>n$ then $\langle x_1, x_2, ..., x_n\rangle$是解
  2. else while $S_k \neq \emptyset$ do
  3. $x_k \leftarrow S_k$中最值
  4. $S_k \leftarrow S_k$-$\{x_k\}$
  5. 计算S_{k+1}
  6. ReBack(k+1)

满足约束分支搜索

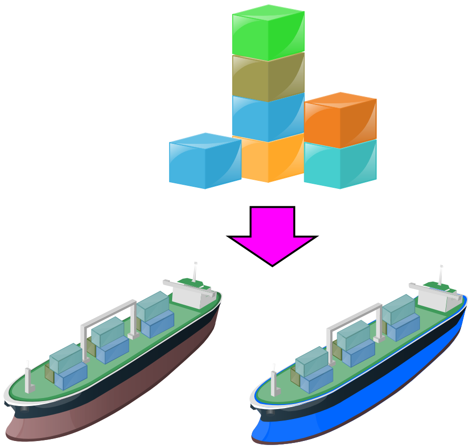
未搜索过的分支

- 算法ReBacktrack(n)
- 输入：n
- 输出：所有解
  1. for k←1 to n 计算X_k且$S_k \leftarrow X_k$
  2. ReBack(1)


回溯法
- 有许多问题，当需要找出它的解集或者要求回答什么解是满足某些约束条件的最佳解时，往往要使用回溯法。回溯法号称计算机算法中最通用的算法！
- 回溯法的基本做法是搜索，或是一种组织得井井有条的(带有系统性)，能避免不必要搜索(带有跳跃性)的穷举式搜索法。这种方法适用于解一些组合数相当大的问题。

装载问题

问题： 有n个集装箱，需要装上两艘载重分别为c_1和c_2的轮船。w_i为第i个集装箱的重量，且$w_1+w_2+\ldots+w_n \leqslant c_1+c_2$。问是否存在一种合理的装载方案把这$n$个集装箱装上船？



实例：

$W=\langle 90, 80, 40, 30, 20, 12, 10 \rangle$

$c_1=152$, $c_2=130$

解： 1,3,6,7装第一艘船，其余第2艘

求解思路

输入： W=⟨w_1, w_2, …, w_n⟩为集装箱重量，c_1和c_2为船的最大载重

算法思想： 令第一艘船的载入量为W_1

1. 用回溯法求使得c_1-W_1达到最小的装载方案
2. 若满足

$$w_1+w_2+\ldots+w_n-W_1 \leqslant c_2$$

则回答“YES”，否则回答“NO”

算法伪码

- 算法Loading(W,c_1)
  1. Sort(W)
  2. $B \leftarrow c_1$; $best \leftarrow c_1$; $i \leftarrow 1$
  3. while $i \leqslant n$ do
  4. if 装入i后重量不超过c_1 ⟶ 此处隐含回溯
  5. then $B \leftarrow B - w_i$; $x[i] \leftarrow 1$; $i \leftarrow i+1$
  6. else $x[i] \leftarrow 0$; $i \leftarrow i+1$
  7. if $B < best$ then 记录解; $best \leftarrow B$
  8. Backtrack(i) ⟶ 此处找到了一个可行解，回溯
  9. if i=1 then return 最优解
  10. else goto 3

B为当前空隙
$best$为最小空隙

子过程 Backtrack

- 算法**Backtrack**(i)

while $i > 1$ **and** $x[i]=0$ **do**
    $i \leftarrow i-1$
if $x[i]=1$
then $x[i] \leftarrow 0$
        $B \leftarrow B+w_i$
        $i \leftarrow i+1$

沿右分支一直回
溯发现左分支边
或到根为止

实例

W=〈90, 80, 40, 30, 20, 12, 10〉

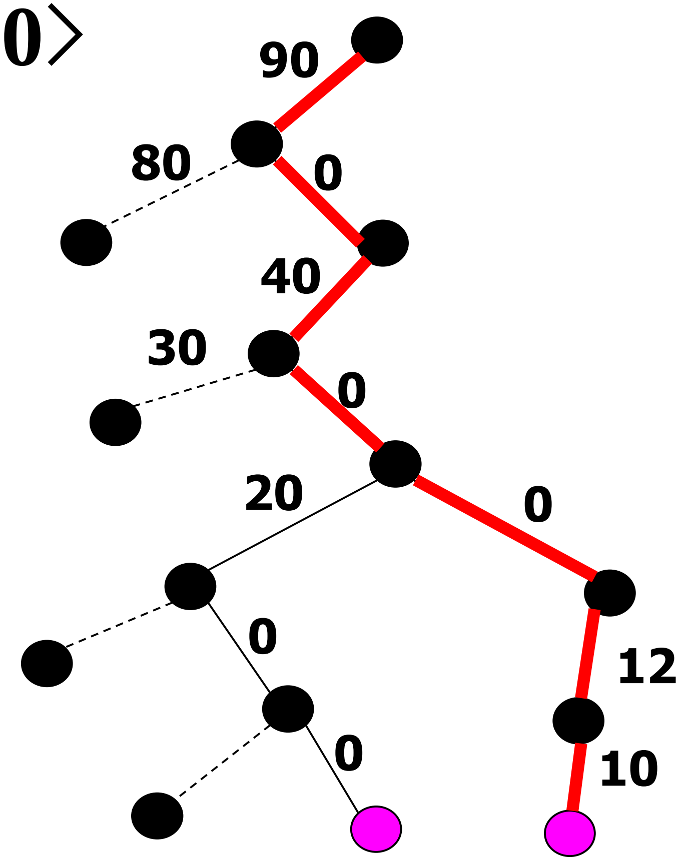
c_1 =152, c_2 =130

解：

可以装，方案如下：

1,3,6,7装第一艘船

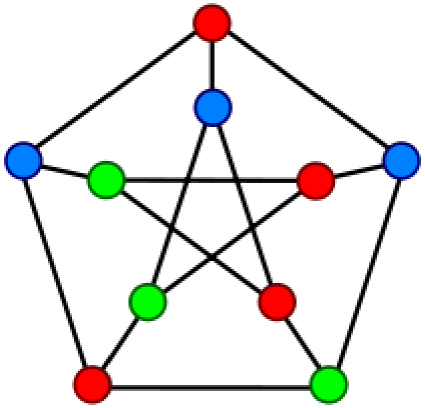
2,4,5装第二艘船



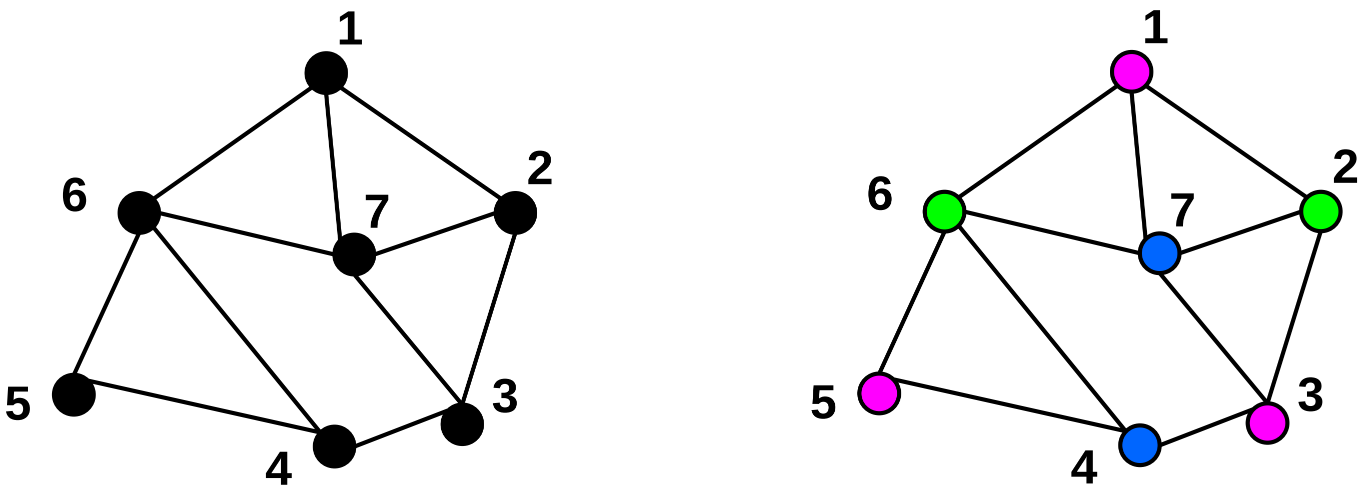
时间复杂度：O(2^n)

图的着色问题

- **输入：** 无向连通图G和*m*种颜色的集合，用这些颜色给图的顶点着色，每个顶点一种颜色。要求是：G的每条边的两个顶点着不同颜色。
- **输出：** 所有可能的着色方案。如果不存在着色方案，回答“NO”



实例



n=7, m=3

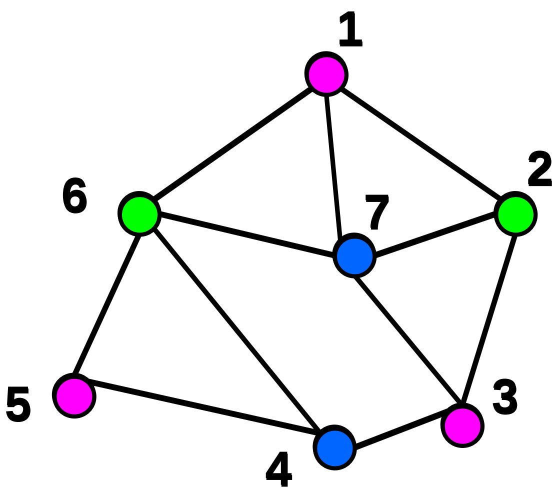
解向量

- 设G=(V, E), V={1,2,...,n}
- 颜色编号：1,2, ... , m
- 解向量：$\langle x_1, x_2, ..., x_n \rangle$
  - $x_i \in \{1,2,...,m\}$
- 结点的部分向量$\langle x_1, x_2, ..., x_k \rangle$, $1 \leqslant k \leqslant n$
  - 表示只给顶点1,2,...,k着色的部分方案

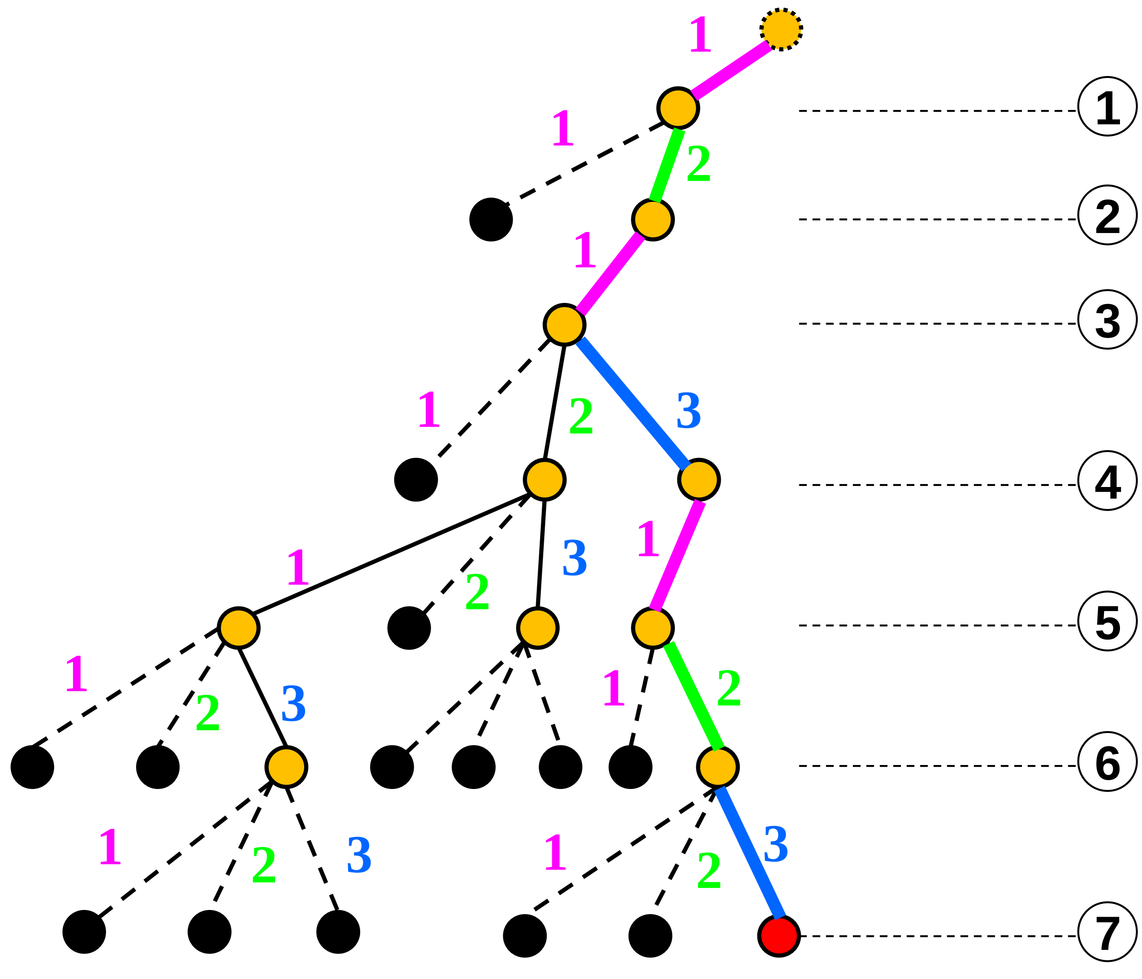
算法设计

- **搜索树：** m叉树
- **约束条件：** 在结点⟨x_1,…, x_k⟩处，顶点k+1的邻接表中结点已用过的颜色不能再用。如果邻接表种结点已用过m种颜色，则结点k+1没法着色，从该结点回溯到其父节点。➔满足多米诺性质
- **搜索策略：** 深度优先
- **时间复杂度：** $O(n \times m^n)$

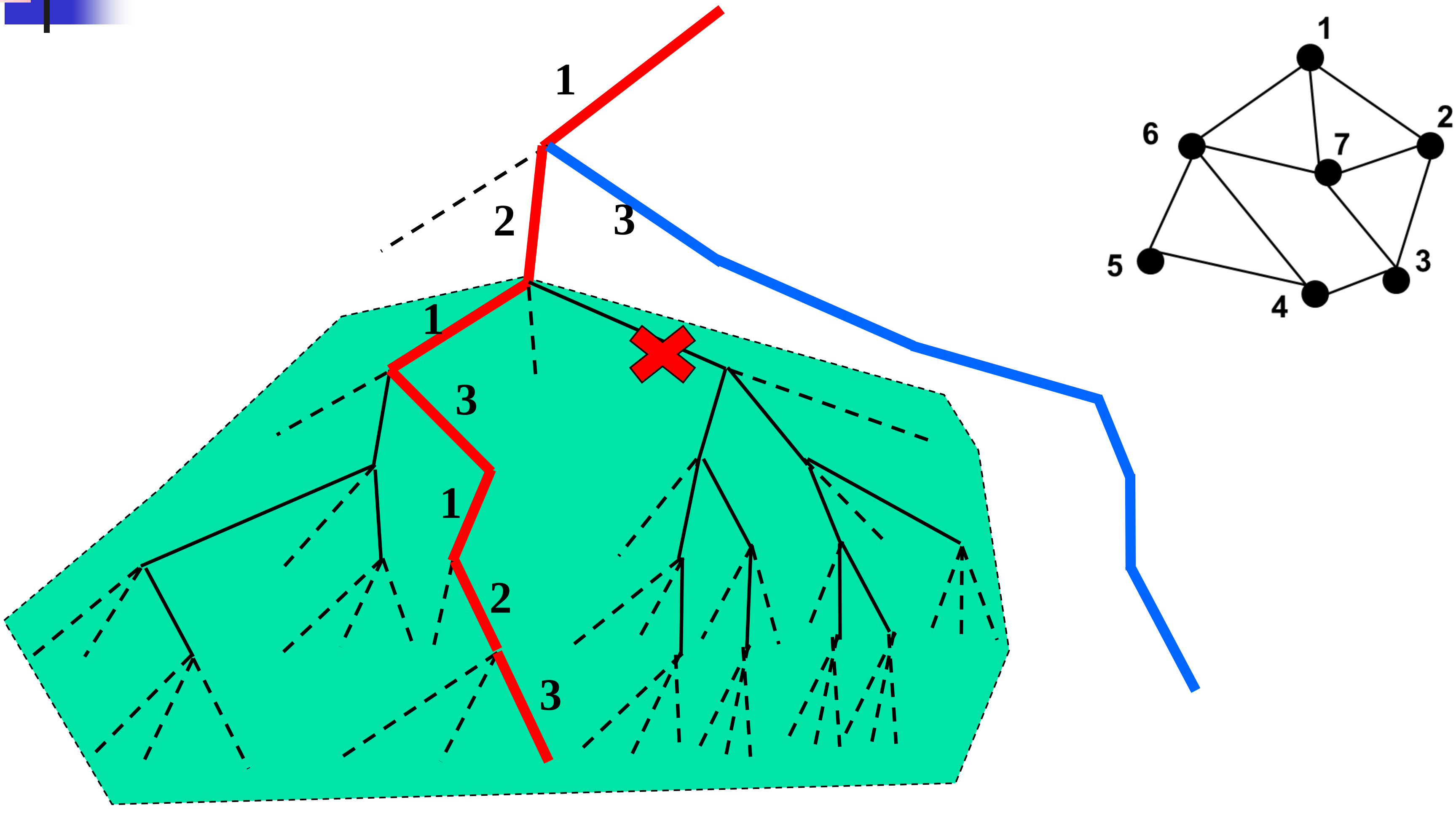
运行实例



n=7, m={1、2、3}



搜索树



第一个解向量：<1, 2, 1, 3, 1, 2, 3>

时间复杂度与改进途径

- 时间复杂度：O($n \times m^n$)
- 根据**对称性**，只需搜索1/6的解空间。当1和2确定，即<1,2>以后，只有1个解，在<1,3>为根的子树中，也只有一个解。由于3个子树的对称性，总共有6个解。
- 在取定<1,2>后，不可扩张成<1,2,3>，因为7和1,2,3都相邻。7没法着色。可以从打叉的结点回溯，而不必搜索其子树。

着色问题的应用

- **会场分配问题：**
  - 有n项活动需要安排，对于活动i,j，如果i,j时间冲突，就说i,j不相容。如何分配这些活动，使得每个会场的活动相容且占用会场数最少？
- **建模：**
  - 活动作为图的顶点，如果i,j不相容，则在i和j之间加一条边，会场标号作为颜色标号。求图的一种着色方案，使得使用的颜色数最少。

第五章小结

- 理解回溯法的解空间构造方法
  - M叉树、子集树、排列树
- 理解回溯法的深度优先搜索策略
- 掌握用回溯法解题的算法框架
  - 递归回溯
  - 迭代回溯
- 通过应用范例学习回溯法的设计策略
  - *n*后问题、0-1背包问题 、旅行售货员问题
  - 装载问题
  - 图的着色问题