



算法分析与设计

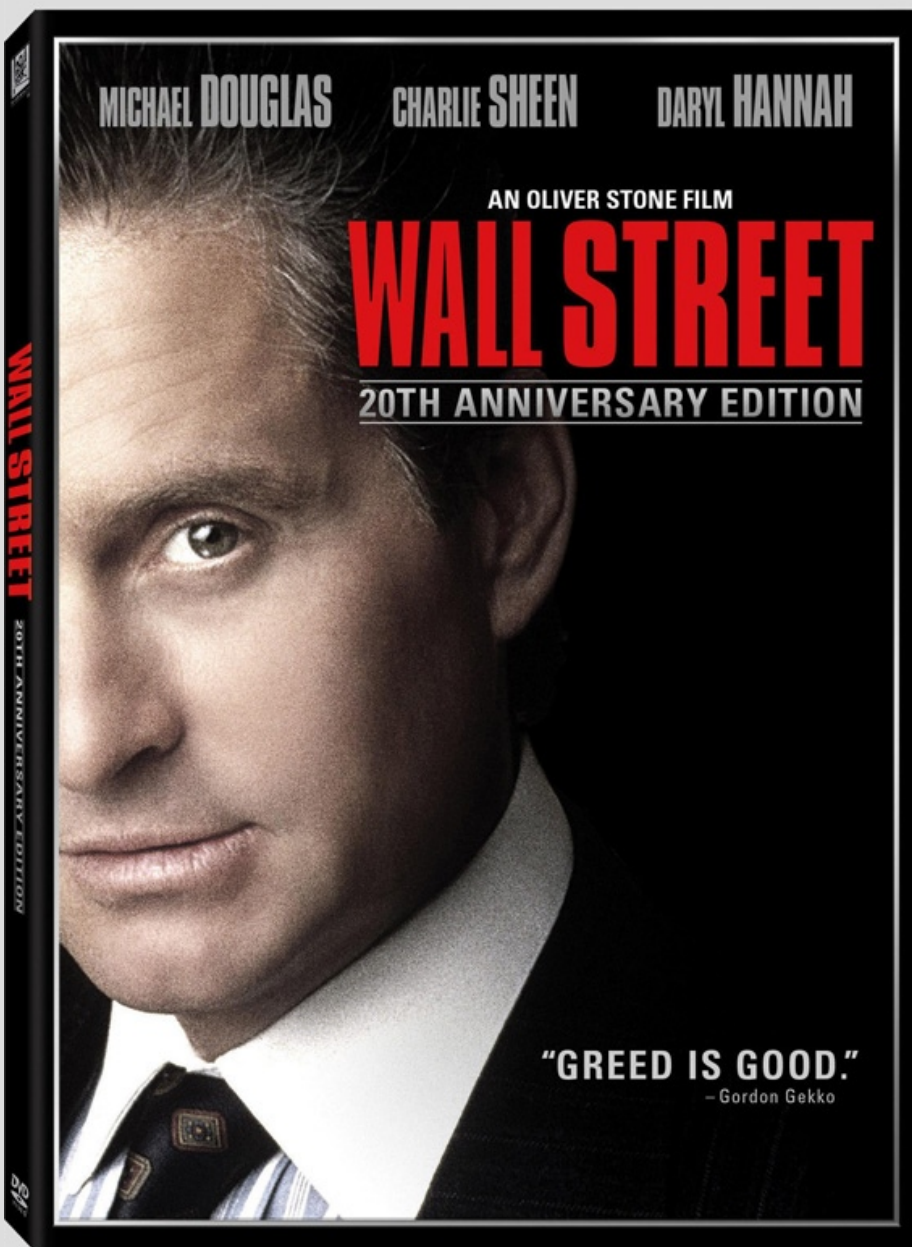
Analysis and Design of Algorithm

Lesson 12



要点回顾

- 贪心算法正确性归纳证明
 - 叙述一个有关自然数 n 的命题，该命题断定该贪心策略的执行最终将导致最优解。其中自然数 n 可以代表步数或问题规模
 - 证明命题对所有的自然数为真
 - 归纳基础（从最小实例规模开始）
 - 归纳步骤（第一或第二数学归纳法）
- 几个实例：
 - 最优前缀码（Huffman树）
 - 最小生成树（Prim和Kruskal算法）
 - 单源最短路径（Dijkstra算法）



贪婪，
我找不到一个更好的
词汇来描述它，
它就是好！
它就是对！
它就是有效！

——**Gordon Gekko**

贪心算法求解NP完全问题



回顾：NP完全问题

- P类问题

- 所有可以用多项式时间的确定性算法来进行判定或求解的问题。

- NP类问题

- 可用多项式时间的非确定性算法来进行判定或求解的问题。

- NP完全问题

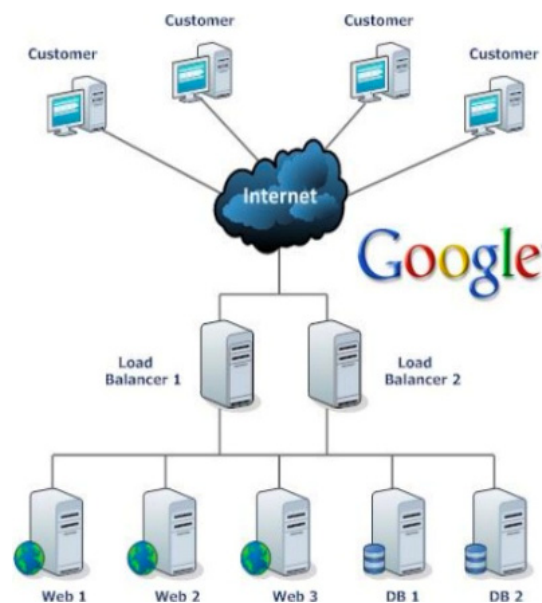
- NP中某些问题不确定能否在多项式时间内求解。
- 这些问题中，任何一个如果存在多项式时间的算法，那么所有NP问题都是多项式时间可解。

多机调度问题及其应用

- **多机调度问题**要求给出一种作业调度方案，使所给的 n 个作业(每个作业运行时间为 t_i)，在尽可能短的时间内由 m 台相同的机器加工处理完成。



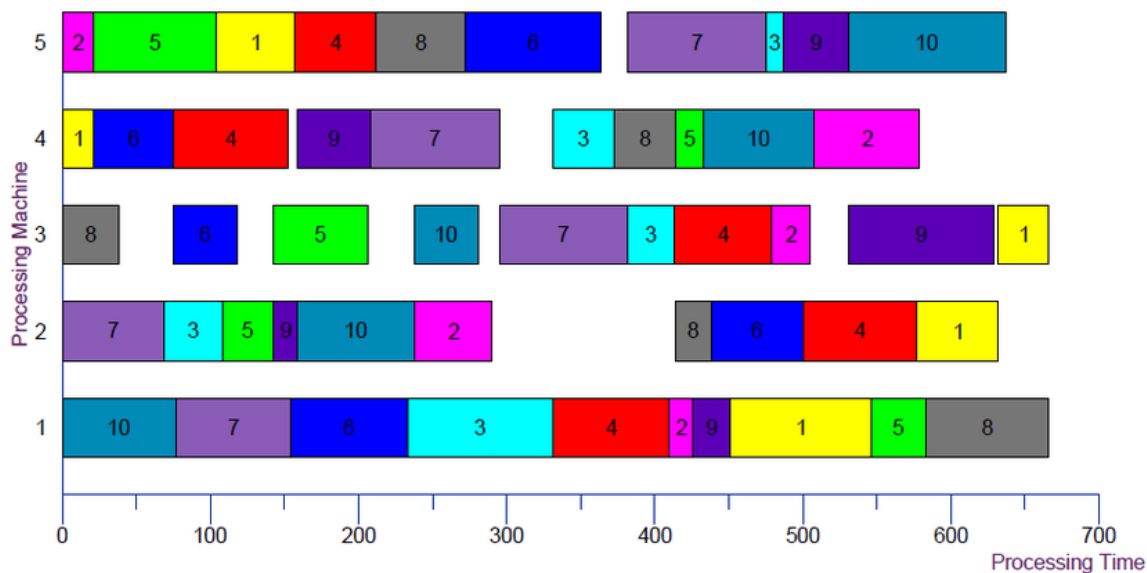
工厂机床调度



计算机集群调度

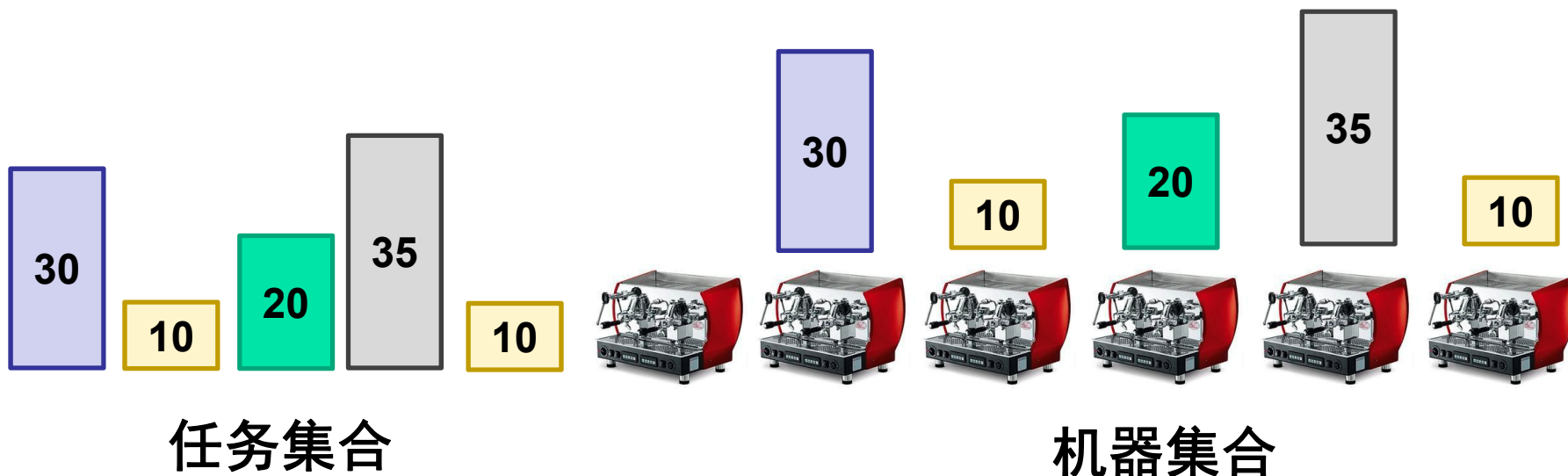
多机调度问题及其应用

- 多机调度问题是**NP完全问题**，到目前为止还没有有效的解法。
- 对于这一类问题，用**贪心选择策略**有时可以设计出较好的**近似算法**。



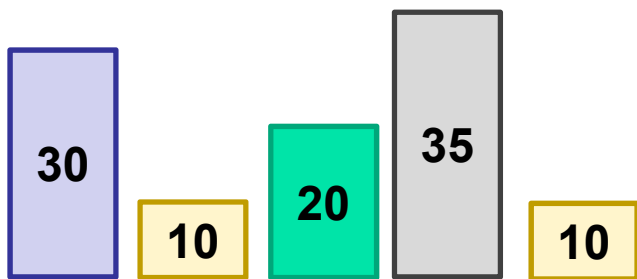
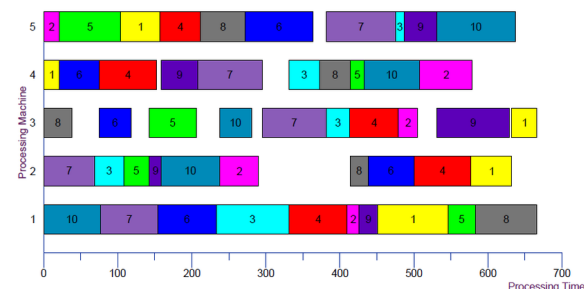
多机调度问题—贪心策略

- 当机器数 $m \geq$ 任务数 n 时
 - 每台机器放一个任务

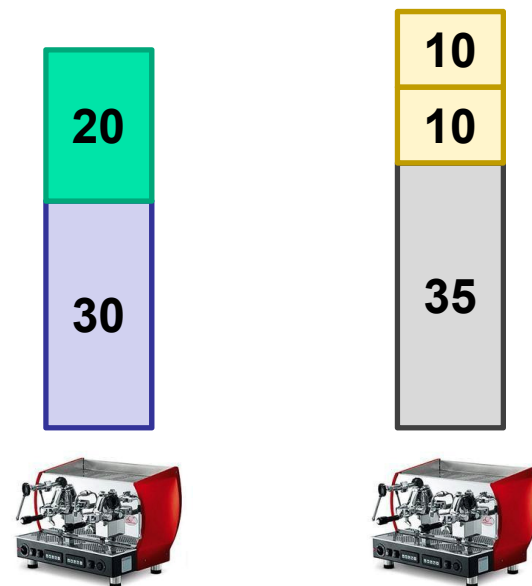


多机调度问题—贪心策略

- 当机器数 $m <$ 任务数 n 时
 - 优先放长任务
 - 优先选择负载最轻机器



任务集合



机器集合



第四章小结

- 贪心法适用于组合优化问题
- 求解过程是多步判断过程，最终的判断序列对应于问题的最优解
- 判断依据某种“短视”贪心选择性质，性质的好坏决定了算法的成败。贪心性质往往依赖于直觉或者经验
- 贪心法**正确性证明**
 - 直接计算优化函数，贪心法解恰好取得最优值
 - 数学归纳法（对算法步骤或者问题规模归纳）



贪心法小结(cont.)

- 对于某些不能保证对所有的实例都得到最优解的贪心算法，可做参数化分析或者误差分析。
- 贪心算法的优势：
 - 算法简单
 - 时空复杂度低
- 几个著名的贪心算法
 - 最小生成树的Prim算法和Kruskal算法
 - 单源最短路径的Dijkstra算法



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

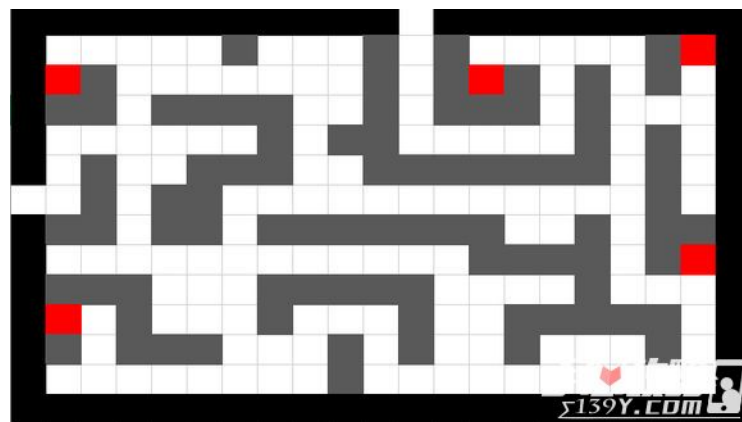
- **理解**回溯法的深度优先搜索策略
- **掌握**用回溯法解题的算法框架
 - **递归**回溯最优子结构性质
 - **迭代**回溯贪心选择性质
 - 子集树算法框架
 - 排列树算法框架
- 通过应用**范例学习**回溯法的设计策略
 - n 后问题、0-1背包问题、旅行售货员问题
 - 装载问题
 - 图的着色问题

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。

走迷宫游戏





回顾：最优化问题(第3章)

- 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，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**。

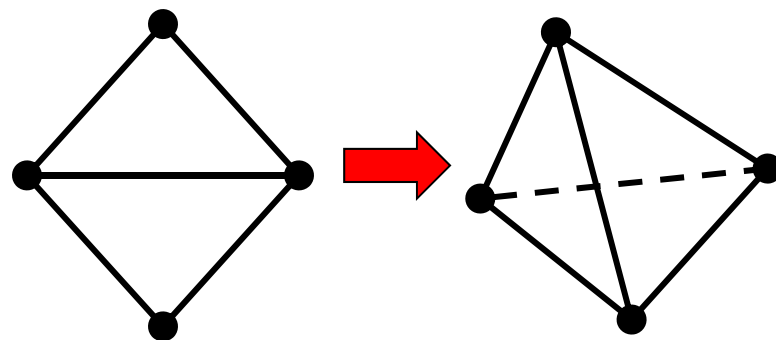
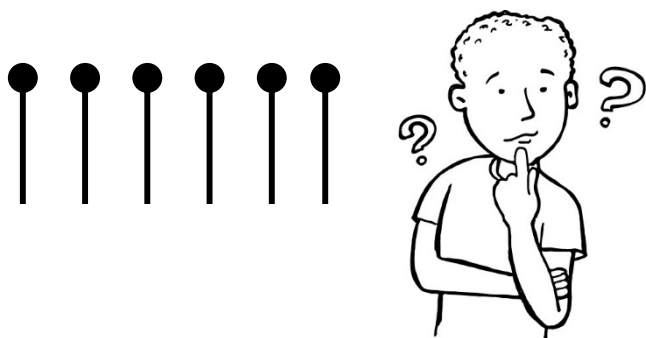


回溯问题的解空间

- **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, \dots, 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)

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？

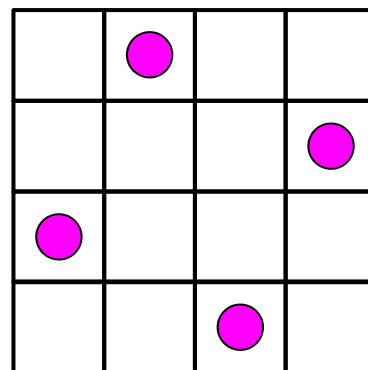
解是4维向量： $\langle x_1, x_2, x_3, x_4 \rangle$

解： $\langle 2, 4, 1, 3 \rangle$, $\langle 3, 1, 4, 2 \rangle$

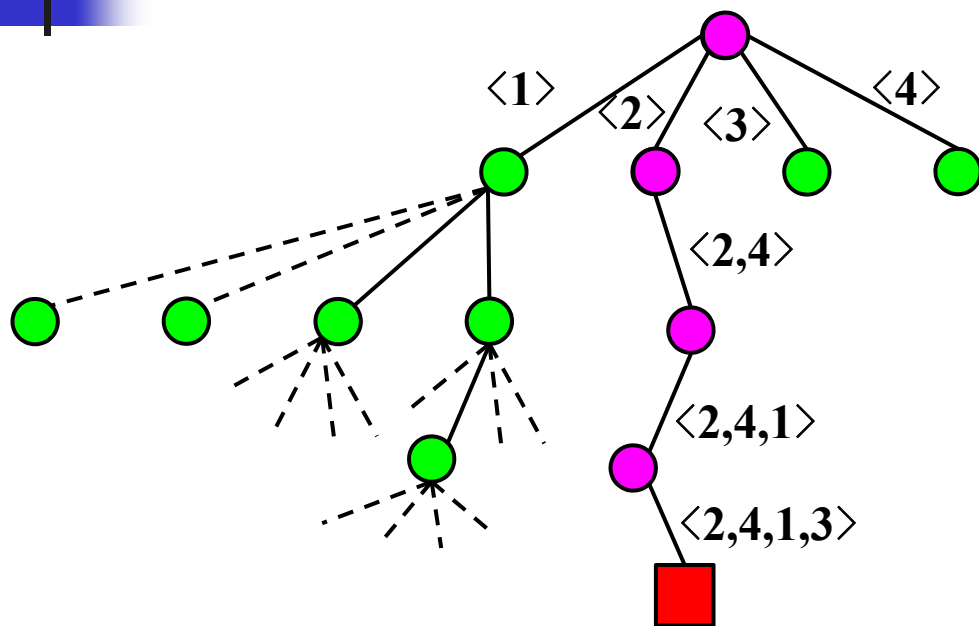
推广到8后问题

解： 8维向量，有92个。


例如： $\langle 1, 5, 8, 6, 3, 7, 2, 4 \rangle$ 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： $\langle 0,1,1,1 \rangle$ ，价值：28，重量：13



算法设计

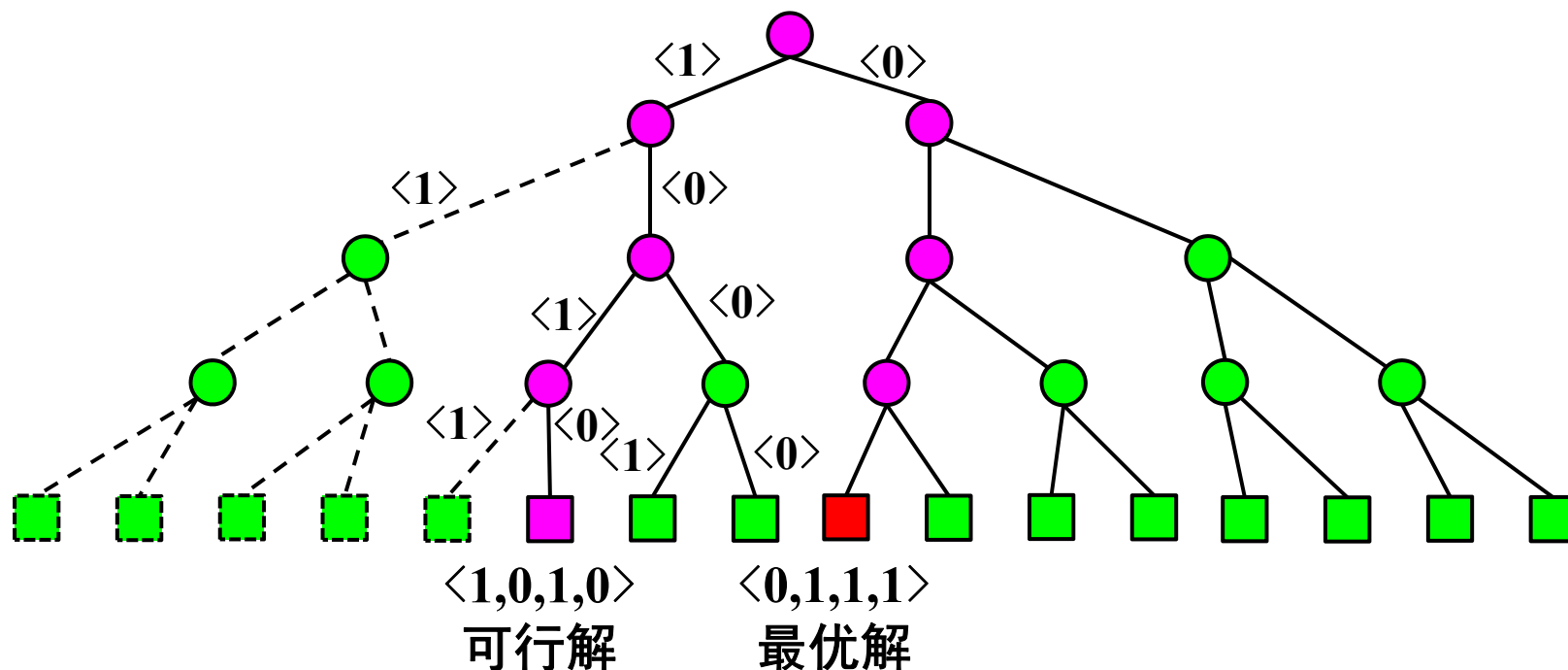
- **解：** n 维0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$
 $x_i = 1 \Leftrightarrow$ 物品 i 选入背包
- **结点：** $\langle x_1, x_2, \dots, x_k \rangle$ （部分向量）
- **搜索空间：** 一棵0-1取值的二叉树，称为**子集树**，有 2^n 片树叶
- **可行解：** 满足约束条件(不超重)的解
- **最优解：** 可行解中价值达到最大的解

实例

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



- **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

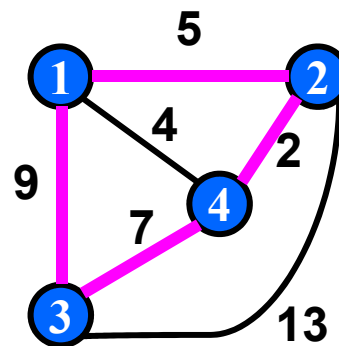
- **问题：** 一个售货员需要在 n 个城市销售商品，已知任两个城市之间的距离，求一条每个城市恰好经过一次的回路，使得总长度最小。
- **建模：** 城市集 $C=\{c_1, c_2, \dots, c_n\}$, 距离 $d(c_i, c_j)=d(c_j, c_i)$
- **求解：** $1, 2, \dots, n$ 的排列 k_1, k_2, \dots, k_n 使得

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

实例

输入：

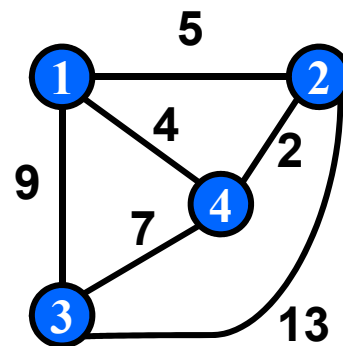
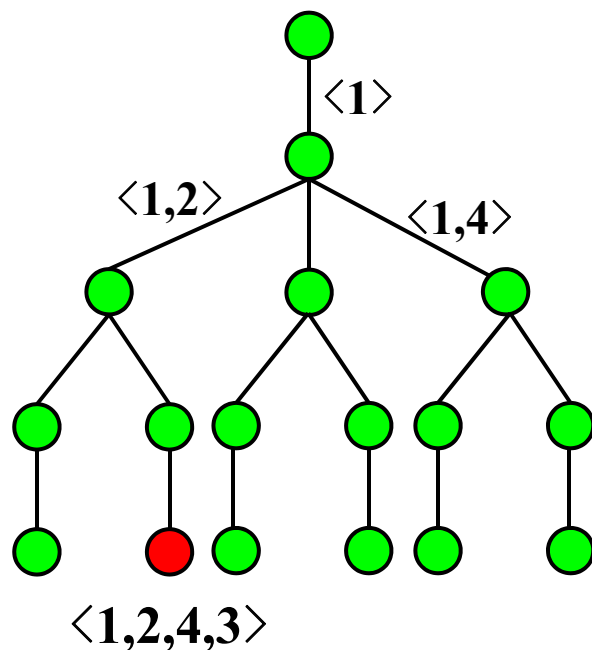
- $C=\{1,2,3,4\}$
- $d(1,2)=5, d(1,3)=9$
- $d(1,4)=4, d(2,3)=13$
- $d(2,4)=2, d(3,4)=7$



解： $\langle 1,2,4,3 \rangle$, 长度 $=5+2+7+9=23$

搜索空间

- 排列树，有 $(n-1)!$ 片树叶



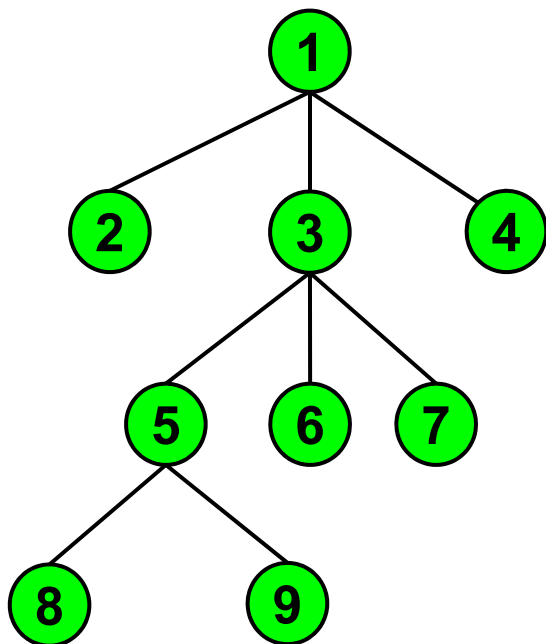
小结

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	$\langle x_1, x_2, \dots, x_n \rangle$ x_i :第 i 行列号	n 叉树	深度优先	彼此 不攻击
0-1背包 问题	最优解	$\langle x_1, x_2, \dots, x_n \rangle$ $x_i=0/1$ $x_i=1 \Leftrightarrow$ 选 i	子集树	深度优先	不超过 背包重量
TSP问题	最优解	$\langle k_1, k_2, \dots, k_n \rangle$ $1, 2, \dots, n$ 的排列	排列树	深度优先	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

回溯法设计思想和适用条件

回顾一下

■ 深度与宽度优先搜索



深度优先访问顺序:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow 4$

宽度优先访问顺序:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$



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（正在访问）、黑色结点（该结点为根的子树遍历完成）
- **存储：**当前路径

结点状态例子

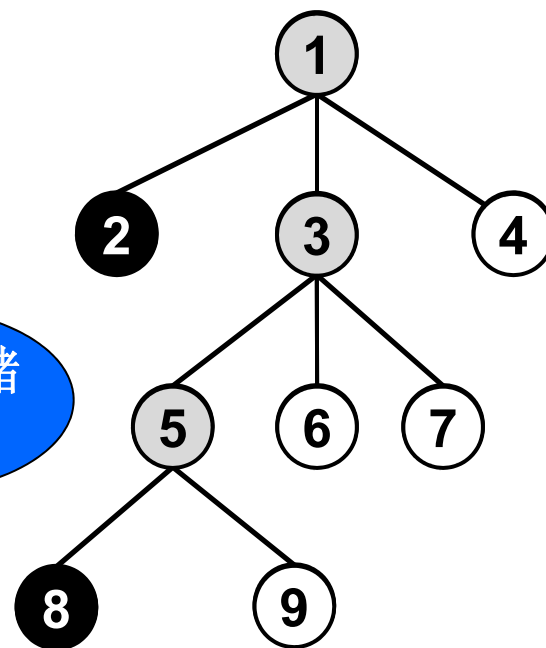
- 策略：深度优先

- 访问次序：

1→2→3→5→8

当前存储
的路径

- 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





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
- **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
- **存储：** 当前路径