数据分析与算法设计
第3章 蛮力法
（Brute Force）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

蛮力法

- 蛮力法：基于问题的描述和所涉及的定义直接求解
  - 计算a^n：连乘n-1次
  - 基于定义的矩阵乘法算法
  - 求最大公约数：连续整数检测算法

- 明显的缺陷：效率不高

- 存在的意义：
  - 简单性和广泛的适用性
  - 基线算法，可作为基础衍生更高效的算法
  - 仍可解决规模小的问题（“杀鸡焉用牛刀”）

目录

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对和凸包问题的蛮力算法
- 穷举查找
- 深度优先和广度优先查找

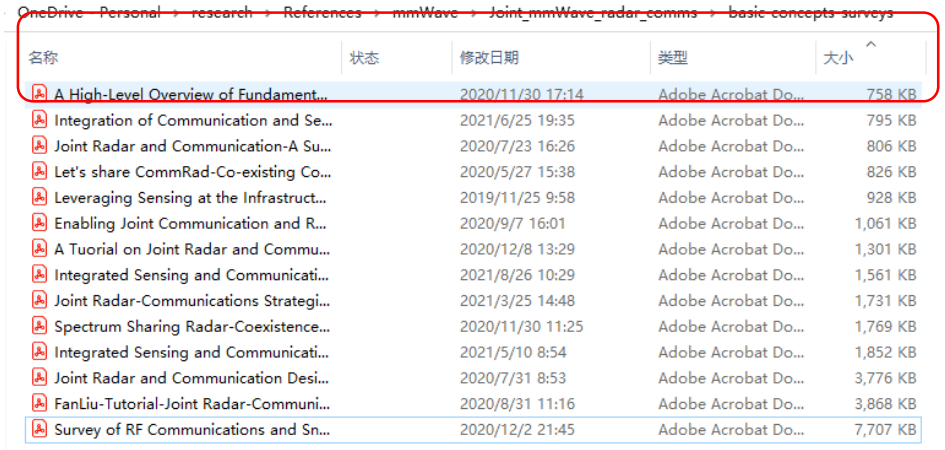
排序

- 问题描述：
  - 给定任意一个可排序的包含n个元素的列表，将元素按照非降序方式重新排列

- 应用举例

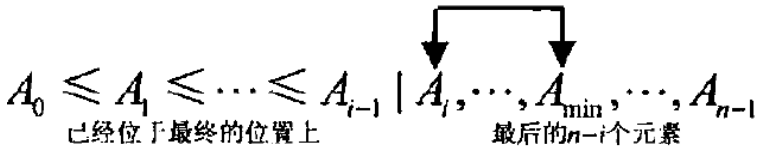
商品推荐：
按综合评分、价格等排序

电脑文件：
按不同属性排序



选择排序

- 算法思想
  - 扫描整个列表，找出最小元素，然后将最小元素与第一个元素交换位置
  - 从第二个元素开始再次扫描列表，找出n-1个元素中的最小元素，和第二个元素交换位置
  - 如此类推，做n-1遍后排序结束

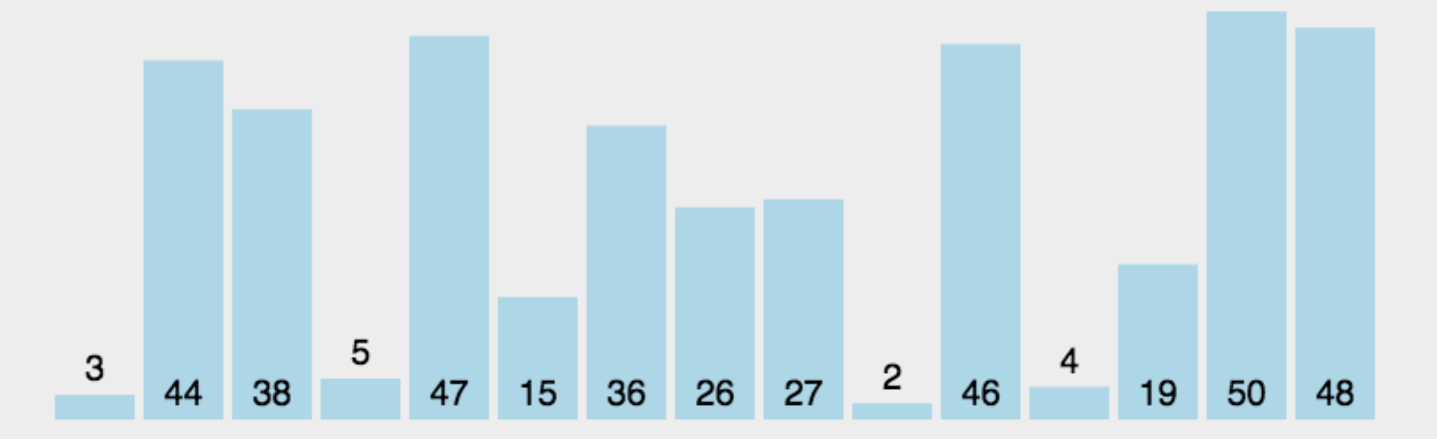


- 算法伪代码

```
算法  SelectionSort(A[0..n − 1])
      //该算法用选择排序对给定的数组排序
      //输入：一个可排序数组 A[0..n − 1]
      //输出：升序排列的数组 A[0..n − 1]
      for i ← 0  to n − 2  do
          min ← i
          for  j ← i+1  to  n − 1  do
              if A[j] < A[min]  min ← j
          swap A[i] and A[min]
```

选择排序

- 动态示例



- 时间效率
  - 基本操作：比较
  - 执行次数：$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = n(n-1)/2 \in \Theta(n^2)$

选择排序:不稳定性

ChatGPT

当排序对象是记录（record）时，我们可以举一个例子来说明选择排序的不稳定性。

假设我们有如下记录列表：

```
css                                                    Copy code

[(5, "apple"), (3, "banana"), (5, "cherry"), (2, "banana")]
```

其中，每个记录包含两个部分：一个整数和一个字符串。我们想按照整数的升序对记录进行排序。如果选择排序是稳定的，那么在排序之后，具有相同整数值的字符串应该保持其相对顺序不变。

使用选择排序算法，可能会导致不稳定性。在第一次选择最小元素并交换之后，我们可能会得到以下结果：

```
css                                                    Copy code

[(2, "banana"), (3, "banana"), (5, "cherry"), (5, "apple")]
```

字符串关键字的相对顺序改变

冒泡排序

- 算法思想
  - 比较相邻元素，如果是逆序的话就交换它们的位置
  - 第一遍逐对比较后，最大的元素就“沉到”列表的最后一个位置
  - 第二遍操作将第二大元素沉下去
  - n-1遍操作后，完成排序

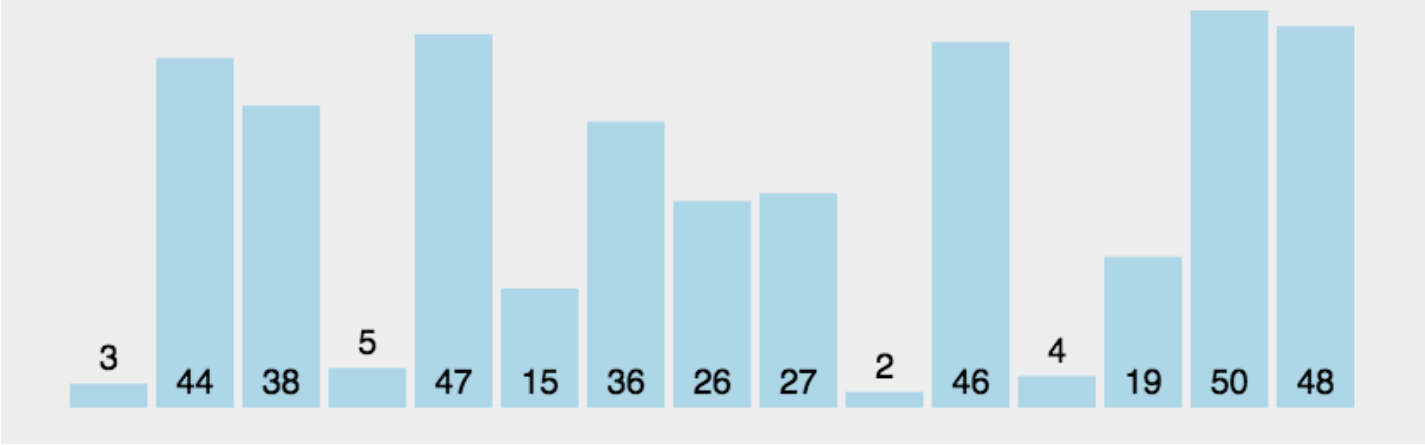
$$A_0, \cdots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \cdots, A_{n-i-1} \mid \underbrace{A_{n-i} \leqslant \cdots \leqslant A_{n-1}}_{\text{已经位于最终的位置上}}$$

- 算法伪代码

算法 BubbleSort($A[0..n-1]$)
//该算法用冒泡排序对数组 $A[0..n-1]$进行排序
//输入：一个可排序数组 $A[0..n-1]$
//输出：非降序排列的数组 $A[0..n-1]$
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow 0$ **to** $n-2-i$ **do**
    if $A[j+1] < A[j]$ swap $A[j]$ and $A[j+1]$

冒泡排序

- 动态示例



- 时间效率
  - 基本操作：比较
  - 执行次数：$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = n(n-1)/2 \in \Theta(n^2)$

改进冒泡排序

改进的冒泡算法如下：

```
ALGORITHM BubbleSortImproved( A[0,...,n - 1] )
// 冒泡排序算法的改进
// 输入：数组A，数组中的元素属于某偏序集
// 输出：按升序排列的数组A
for i ← 0 to n - 2 do
    flag ← True
    for j ← 0 to n - 2 - i do
        if A[j+1] < A[j]
            swap(A[j], A[j+1])
            flag ← False
    // 如果在某一轮的比较中没有交换，则flag为True，算法结束
    if flag = True return
```

未完待续…

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

目录

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对和凸包问题的蛮力算法
- 穷举查找
- 深度优先和广度优先查找

顺序查找

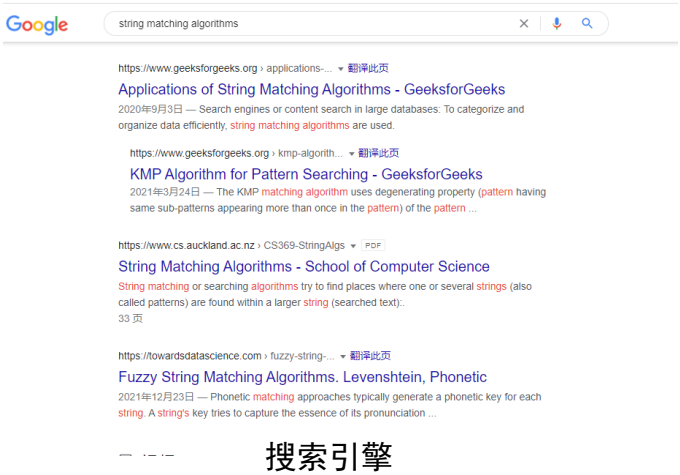
算法 *SequentialSearch2*$(A[0..n],K)$

```
//顺序查找的算法实现，它用了查找键来做限位器
//输入：一个 n 个元素的数组 A 和一个查找键 K
//输出：第一个值等于 K 的元素的位置，如果找不到这样的元素，返回-1
A[n] ← K
i ← 0
while A[i] ≠ K do
    i ← i + 1
if i < n return i
else return -1
```

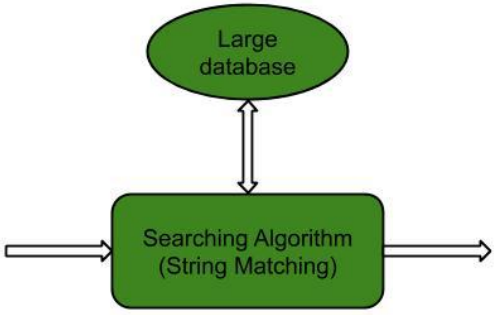
注：把查找键添加到列表的末尾，那么查找就一定会成功，不必在算法的每次循环时都检查是否已经到了表的末尾

字符串匹配

- 问题描述
  - 从文本中寻找匹配模式的子串，即求出第一个匹配模式的子串在文本中的开始位置
    - 文本：给定的由n个字符组成的串
    - 模式：给定的由m个字符组成的串
- 应用举例



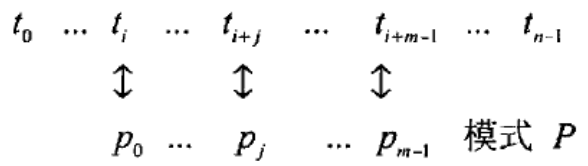
搜索引擎



数据库查找

蛮力匹配

- 算法思想
  - 将模式对准文本的前m个字符从左往右进行比对。如果其中有一个字符不匹配，模式往右移动一位，继续下一组m个字符的比对

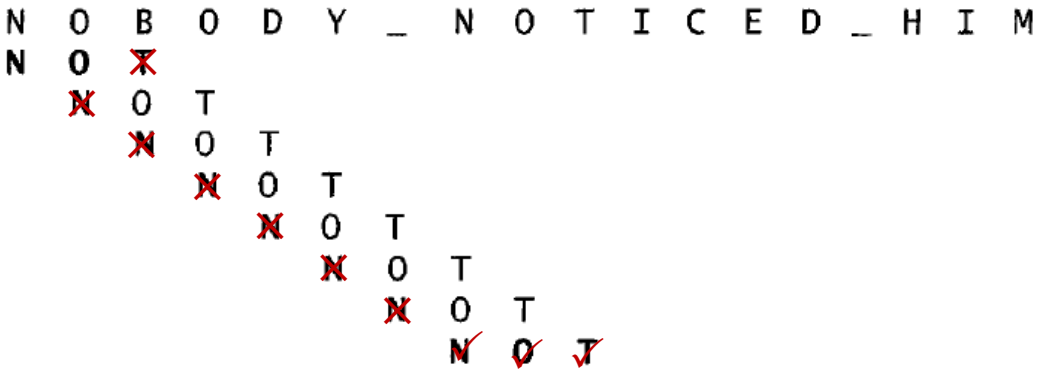


- 算法伪代码

```
算法  BruteForceStringMatch(T[0..n-1], P[0..m-1])
//该算法实现了蛮力字符串匹配
//输入：一个 n 个字符的数组 T[0..n-1]，代表一段文本
//      一个 m 个字符的数组 P[0..m-1]，代表一个模式
//输出：如果查找成功，返回文本的第一个匹配子串中第一个字符的位置，
//      否则返回-1
for i←0 to n-m do
    j←0
    while j<m and P[j]=T[i+j] do
        j←j+1
        if j=m return i
return -1
```

蛮力匹配

- 实例
  - 文本：NOBODY_NOTICED_HIM
  - 模式：NOT



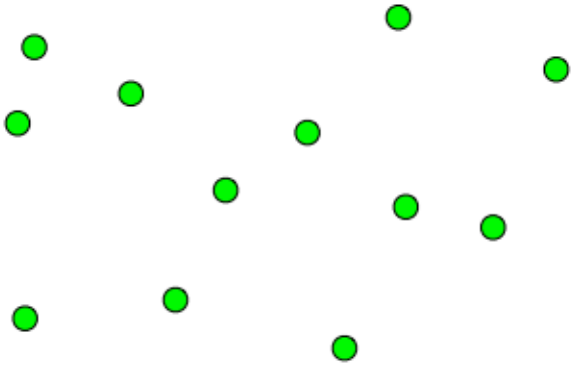
- 时间效率
  - 最差效率：$\Theta(nm)$
  - 平均效率：$\Theta(n)$ （为什么？）

目录

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对和凸包问题的蛮力算法
- 穷举查找
- 深度优先和广度优先查找

最近对问题

- 问题描述
  - 在一个包含n个点的集合中，找出距离最近的两个点



- 应用举例
  - 航空管制：判定某区域内最有可能碰撞的两架飞机
  - 邮政管理：寻找地理位置最近的邮局
  - 通信网络：最佳链路(最短距离)调度
  - 机器学习：基于相似度（如欧式距离）的数据聚类

蛮力求解最近对

- 算法思想
  - 分别计算每一对点之间的距离，然后找出距离最小的那一对

- 算法伪代码

算法 BruteForceClosestPoints(p)

//使用蛮力算法求平面中距离最近的两点

//输入：一个 $n(n\geq 2)$个点的列表 p，$p_1=(x_1,y_1),\cdots,p_n=(x_n,y_n)$

//输出：两个最近点的距离

$d \leftarrow \infty$

for $i \leftarrow 1$ **to** $n-1$ **do**

  for $j \leftarrow i+1$ **to** n **do**

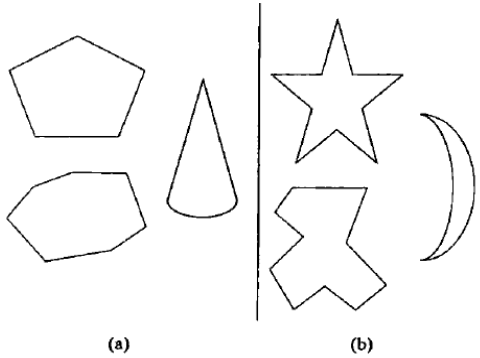
    $d \leftarrow \min(d, \text{sqrt}((x_i-x_j)^2+(y_i-y_j)^2))$ //sqrt 是平方根函数

return d

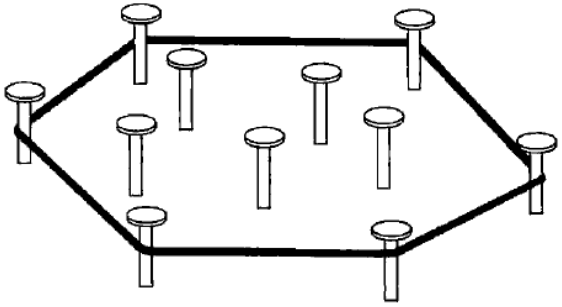
时间效率：$\Theta(n^2)$

凸包问题

- 相关定义
  - 凸集(Convex Set)
    - 设S是平面点集，若S中任意两点的连线都属于该集合，则称S是凸集
  - 凸包(Convex Hull)
    - 一个点集S的凸包是指包含S的最小凸集
    - 若S是凸集，则S的凸包就是本身



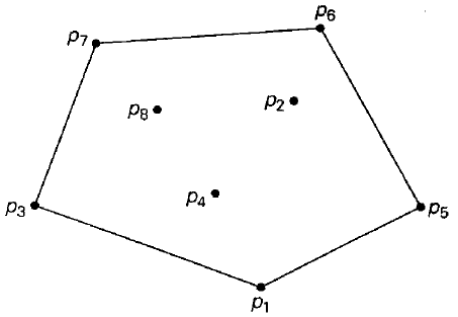
(a)凸集合，(b)非凸集合



用橡皮筋圈来解释凸包

凸包问题

- 问题描述
  - 给定一个n个点的平面点集S，求S的凸包
- 解决原理
  - 定理：任意包含n>2个点（不共线的点）的集合S的凸包是以S中的某些点为顶点的凸多边形（如是所有点都位于一条直线上，多边形退化为一条线，但它两个端点仍包含在S中）
  - 极点：凸集的极点是指不出现在集合中任何线段中间的点。凸多边形的顶点就是极点。



8 个点的集合的凸包是以 p_1，p_5，p_6，p_7 和 p_3 为顶点的凸多边形

凸包问题

- 蛮力算法思想
  - 设点集大小为n，首先将其中的点两两配对，得到直线段条。然后对每一条直线段，检查其余的n-2个点是否位于该直线段的同一边。若是，则该直线段属于凸多边形的边界，其顶点是极点

  - 判断是否同一边的方法：设直线方程为

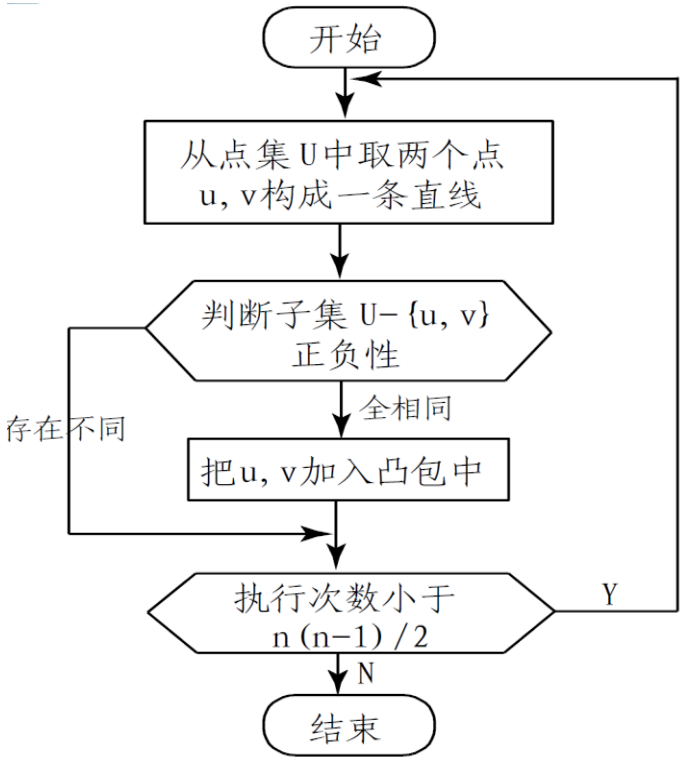
$$ax + by = c$$

$$a = y_2 - y_1, b = x_1 - x_2, c = x_1y_2 - y_1x_2$$

把其余n-2个点代入，判断ax+by-c的正负性，如果具有相同的正负性，则位于同一边

凸包问题

- 算法流程图



- 算法效率：$O(n^3)$

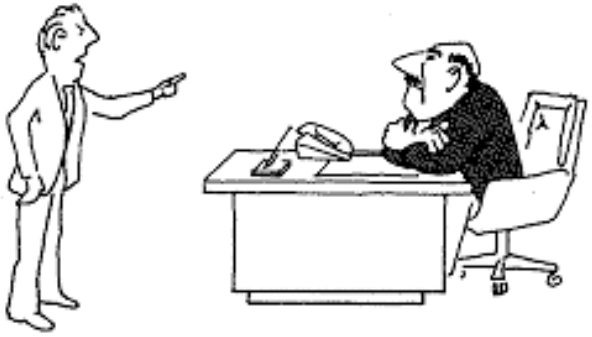
目录

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对和凸包问题的蛮力算法
- 穷举查找
- 深度优先和广度优先查找

穷举查找

- 穷举法：简单的蛮力法，它按要求生成所有符合约束的可行解，再从中找出最优解

- 原理简单，但大多数情况下不可行的，因为复杂度过大。只适合一些规模小的问题，或者是确实无法找到更有效的精确求解方法的组合问题



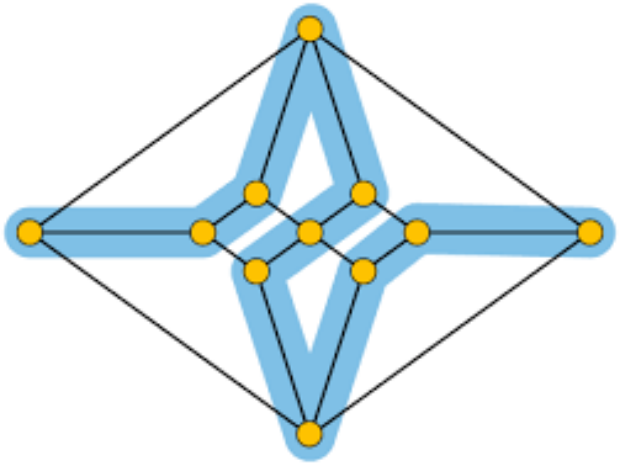
"I can't find an efficient algorithm, because no such algorithm is possible!"

旅行商问题

- 问题描述（TSP-Traveling Salesman Problem）
  - 给定n个城市相互之间的距离，求一条能走遍n个城市的最短路径，要求从任一城市出发，每个城市只访问一次，最后回到起点
  - 等价于图论中的“最短Hamilton回路”问题

Sir William Rowan Hamilton

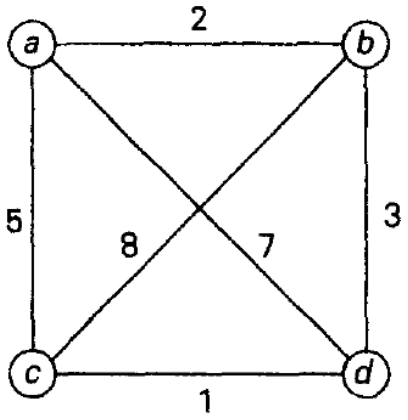
- **Sir William Rowan Hamilton** (August 4, 1805 – September 2, 1865) was an Irish mathematicianSir, physicist, and astronomer who made important contributions to the development of optics, dynamics, and algebra. His discovery of quaternions is perhaps his best known investigation. Hamilton's work was also significant in the later development of quantum mechanics.



Hamilton回路示意

旅行商问题

- 实例（小规模问题的穷举查找）

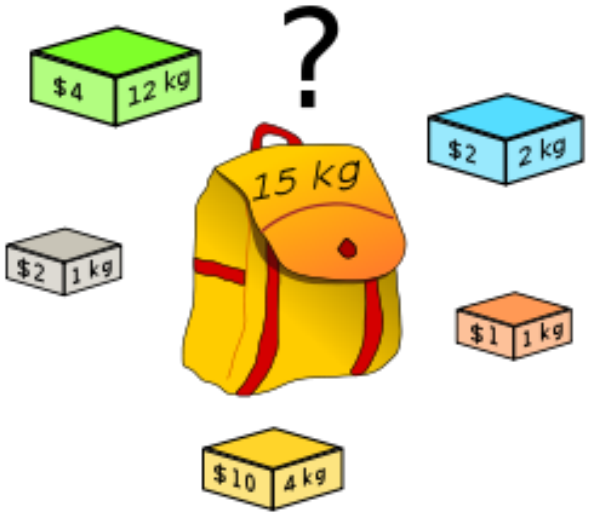


路线	旅程	
a —> b —> c —> d —> a	l = 2 + 8 + 1 + 7 = 18	
a —> b —> d —> c —> a	l = 2 + 3 + 1 + 5 = 11	最佳
a —> c —> b —> d —> a	l = 5 + 8 + 3 + 7 = 23	
a —> c —> d —> b —> a	l = 5 + 1 + 3 + 2 = 11	最佳
a —> d —> b —> c —> a	l = 7 + 3 + 8 + 5 = 23	
a —> d —> c —> b —> a	l = 7 + 1 + 8 + 2 = 18	

$(n-1)!/2$ 对路线

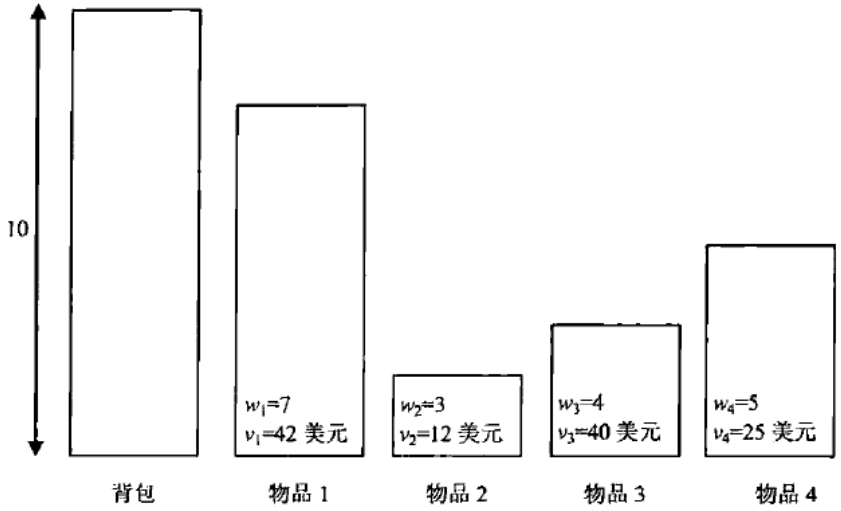
背包问题

- 问题描述（Knapsack Problem）
  - 给定n个重量为W_1、W_2、…W_n，价值为V_1、V_2、…V_n的物品和一个承重为W的背包，要求选择一些物品装入背包，使得背包内物品的价值之和达到最大



背包问题

- 实例（小规模问题的穷举查找）



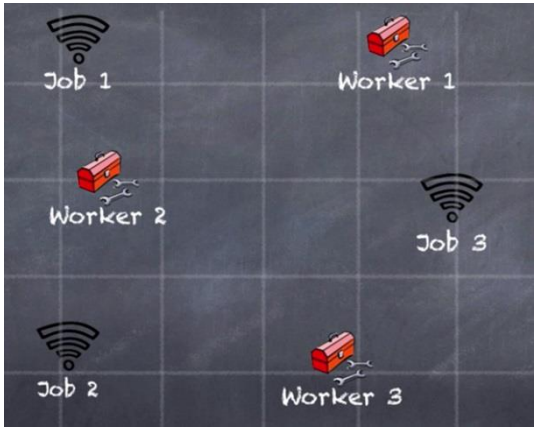
子 集	总 重 量	总价值/美元
∅	0	0
{1}	7	42
{2}	3	12
{3}	4	40
{4}	5	25
{1, 2}	10	54
{1, 3}	11	不可行
{1, 4}	12	不可行
{2, 3}	7	52
{2, 4}	8	37
{3, 4}	**9**	**65**
{1, 2, 3}	14	不可行
{1, 2, 4}	15	不可行
{1, 3, 4}	16	不可行
{2, 3, 4}	12	不可行
{1, 2, 3, 4}	19	不可行

子集总数：$C_0^n + C_1^n + C_2^n + \ldots + C_n^n = 2^n$

时间效率：$\Omega(2^n)$

分配问题

- 问题描述（Distribution Problem）
  - 有n个任务需要分配给n个人执行，一人一个任务。将第j个任务分配给第i个人的成本是C[i,j]，求总成本最小的分配方案



网络公司派单给用户装wi-fi
- Job：装wi-fi
- Worker：工人

- 一种分配方案，对应一个排列
  - 穷举法要列出所有可能的n!种分配方案进行比较

分配问题

- 实例（小规模问题的穷举查找）

人　员	任务 1	任务 2	任务 3	任务 4
人员 1	9	2	7	8
人员 2	6	4	3	7
人员 3	5	8	1	8
人员 4	7	6	9	4

$$C=\begin{bmatrix}9 & 2 & 7 & 8\\ 6 & 4 & 3 & 7\\ 5 & 8 & 1 & 8\\ 7 & 6 & 9 & 4\end{bmatrix}$$

$\langle 1,2,3,4\rangle$　成本 $=9+4+1+4=18$

$\langle 1,2,4,3\rangle$　成本 $=9+4+8+9=30$

$\langle 1,3,2,4\rangle$　成本 $=9+3+8+4=24$

$\langle 1,3,4,2\rangle$　成本 $=9+3+8+6=26$

$\langle 1,4,2,3\rangle$　成本 $=9+7+8+9=33$

$\langle 1,4,3,2\rangle$　成本 $=9+7+1+6=23$

分配问题

- 更高效的算法：匈牙利算法(Hungarian Method)

$$\min \quad z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$s.t. \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \cdots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \cdots, n$$

$$x_{ij} = 0\text{或}1, \quad i, j = 1, 2, \cdots, n$$

分配问题：匈牙利算法

定理1：设分配问题的成本矩阵为C，若将该矩阵的某一行或者列的各元素都减去同一个常数t，得到新的成本矩阵C'，则以C'为成本矩阵的新分配问题与原分配问题的最优解相同，但其新最优值比原最优值小t。

证明：

$$z' = \sum_{i=1}^{n}\sum_{j=1}^{n} c'_{ij}x_{ij} = \sum_{i=1,i\neq k}^{n}\sum_{j=1}^{n} c'_{ij}x_{ij} + \sum_{j=1}^{n} c'_{kj}x_{kj}$$

$$= \sum_{i=1,i\neq k}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n}(c_{kj}-t)x_{kj}$$

$$= \sum_{i=1,i\neq k}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} c_{kj}x_{kj} - t\sum_{j=1}^{n} x_{kj} = z - t.$$

分配问题：匈牙利算法

定理2：若将分配问题的成本矩阵每一行与每一列分别减去各行和各列的最小元素，则得到的新分配问题与原分配问题有相同的最优解。

注：对应的新成本矩阵C’必然会出现一些零元素，若元素C’[i,j]=0，则表示第j项工作分配给第i个人的成本最低。

分配问题：匈牙利算法

- 实例1

$$C = \begin{bmatrix} 40 & 60 & 15 \\ 25 & 30 & 45 \\ 55 & 30 & 25 \end{bmatrix}$$

  - Step1：row reduction（行化简）
    - 每一行减去这一行的最小元素

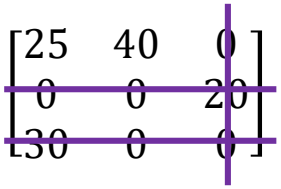
$$\begin{bmatrix} 40 & 60 & 15 \\ 25 & 30 & 45 \\ 55 & 30 & 25 \end{bmatrix} \Longrightarrow \begin{bmatrix} 25 & 45 & 0 \\ 0 & 5 & 20 \\ 30 & 5 & 0 \end{bmatrix}$$

  - Step2：column reduction（列化简）
    - 每一列减去这一列的最小元素

$$\begin{bmatrix} 25 & 45 & 0 \\ 0 & 5 & 20 \\ 30 & 5 & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} 25 & 40 & 0 \\ 0 & 0 & 20 \\ 30 & 0 & 0 \end{bmatrix}$$

分配问题：匈牙利算法

- 实例1
  - Step3：Test for an optimal assignment（测试最优性）
    - 用最少的直线把矩阵中的零元素都覆盖



直线数=3=n

  - Final step: Making the final assignment (完成最优分配)
    - 选择n个不在同一行、同一列的零元素代表最终的分配方案

$$\begin{bmatrix} 25 & 40 & 0^* \\ 0^* & 0 & 20 \\ 30 & 0^* & 0 \end{bmatrix} \xrightarrow{\text{成本}} \begin{bmatrix} 40 & 60 & 15 \\ 25 & 30 & 45 \\ 55 & 30 & 25 \end{bmatrix}$$

如果在步骤3中，直线数不等于n呢？

分配问题：匈牙利算法

- 实例2

$$C = \begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix}$$

  - Step1：row reduction（行化简）
    - 每一行减去这一行的最小元素

$$\begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix} \Longrightarrow \begin{bmatrix} 20 & 15 & 0 \\ 5 & 0 & 10 \\ 10 & 5 & 0 \end{bmatrix}$$

  - Step2：column reduction（列化简）
    - 每一列减去这一列的最小元素

$$\begin{bmatrix} 20 & 15 & 0 \\ 5 & 0 & 10 \\ 10 & 5 & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

分配问题：匈牙利算法

- 实例2
  - Step3：Test for an optimal assignment（测试最优性）
    - 用最少的直线把矩阵中的零元素都覆盖

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

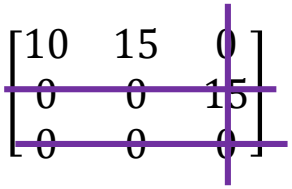
直线数= 2 < n

  - Step4: Shift zeros（转移零元素）
    - 转移至少1个0到直线未覆盖的位置：找出没被直线覆盖的值中的最小值；每个未被直线覆盖的值都减去这个最小值，然后在直线交叉处加上该最小值；把直线移除，回到第三步

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 10 & 15 & 0 \\ 0 & 0 & 15 \\ 0 & 0 & 0 \end{bmatrix}$$

分配问题：匈牙利算法

- 实例2
  - Repeat-Step3：Test for an optimal assignment（测试最优性）
    - 用最少的直线把矩阵中的零元素都覆盖



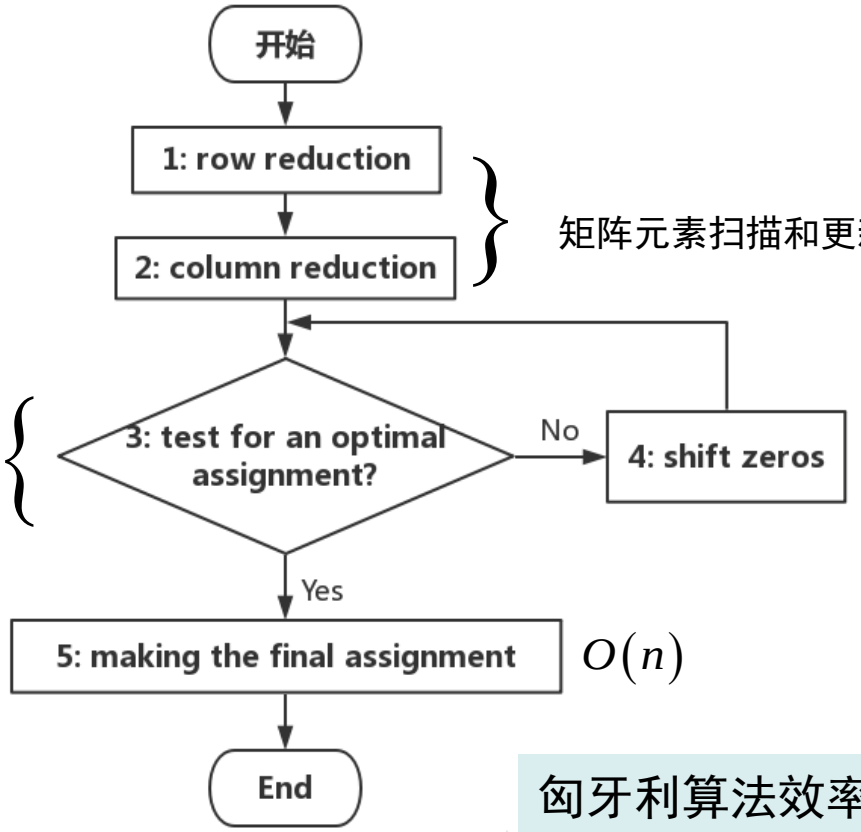
直线数= 3 = n

  - Final step: Making the final assignment（完成最优分配）
    - 选择n个不在同一行、同一列的零元素代表最终的分配方案

$$\begin{bmatrix} 10 & 15 & 0^* \\ 0^* & 0 & 15 \\ 0 & 0^* & 0 \end{bmatrix} \xrightarrow{\text{成本}} \begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix}$$

分配问题：匈牙利算法

- 算法流程图和效率分析



矩阵元素扫描和更新 $O(n^2)$

如果最优性测试不满足，则迭代调整，每次调整需扫描和更新矩阵元素一轮；最多可能有n迭代

$O(n*n^2)$

$O(n)$

匈牙利算法效率 $O(n^3) < O(n!)$

永不停歇地探索组合问题的高效求解

- 案例：小量变引起大质变，多项式几何助力旅行商问题研究取得突破性进展

两年前，当 Nathan Klein 刚进入华盛顿大学研究生院时，他的导师提出了一个谦逊的培养计划：一起研究理论计算机科学领域一个最有名的待解决问题。

他们当时的想法是，就算最终没能解决它，Klein 也能在过程中学到很多。Klein 同意了这一想法。「我当时还不知道害怕，」他说，「我那时还是个一年级新生，根本搞不清状况。」

而现在，在今年 7 月发布的一篇 arXiv 论文中，华盛顿大学在读博士 Nathan Klein 及其导师 Anna Karlin 和 Shayan Oveis Gharan 终于实现了一个令计算机科学家追逐了半个世纪的目标：寻找旅行商问题近似解的更优方法。

永不停歇地探索组合问题的高效求解

旅行商问题是一个优化问题，其目标是寻找走完一组城市的最短路径（或成本最低的路径）。这个问题的解决方案可用于 DNA 测序和共享物流规划等许多应用。几十年来，这一问题激发了计算机科学领域多项根本性的进步，帮助展现了线性规划等技术的力量。不过，其潜在的可能性还未得到完全探索，不过研究者为此付出的努力并不少。

正如计算复杂性领域著名专家 Christos Papadimitriou 说的那样：旅行商问题「不是一个问题，而是一个会让人上瘾的东西」。

大多数计算机科学家认为：并不存在一种有效解决各种城市组合可能性的最优解。但在 1976 年，Nicos Christofides 提出了一种能有效找到近似解的算法——往返旅程最多比最佳往返旅程长 50%。那时，计算机科学家预计很快就有人能在 Christofides 的简单算法上实现提升，进一步接近真实解。但预期的进展并未到来。

永不停歇地探索组合问题的高效求解

现在，Karlin、Klein 和 Oveis Gharan 已经证明：十年前设计的一种算法优于 Christofides 算法的 50% 标准，虽然它也只是将这个百分数减少了非常小的数字（10^-36，0.2 billionth of a trillionth of a trillionth of a percent）。尽管如此，进步终究是进步，这一微小进步能够突破理论上的僵局以及心理上的门槛。研究者希望这能带来契机，实现进一步的改进。

华盛顿大学研究生 Nathan Klein（左）及其导师 Anna Karlin 和 Shayan Oveis Gharan

康奈尔大学教授 David Williamson 表示：「这是我一生事业所追求的目标。」他自 1980 年代以来一直在研究旅行商问题。

旅行商问题是理论计算机科学家试图解决的基础性问题之一，旨在探索高效计算（efficient computation）的极限。Williamson 说：「这个新结果是向人们展示高效计算的发展前沿事实上比我们预想的更好的第一步。」

A (Slightly) Improved Approximation Algorithm for Metric TSP

Anna R. Karlin*, Nathan Klein†, and Shayan Oveis Gharan‡

University of Washington

September 1, 2020

Abstract

For some $\epsilon > 10^{-36}$ we give a $3/2 - \epsilon$ approximation algorithm for metric TSP.

目录

- 选择排序和冒泡排序
- 顺序查找和蛮力字符串匹配
- 最近对和凸包问题的蛮力算法
- 穷举查找
- 深度优先和广度优先查找

深度/广度优先查找

- 目的：遍历图的顶点
- 深度优先查找（DFS-Depth First Search）
  - 从任意顶点开始访问，将该顶点标记为已访问。每次迭代时，紧接着处理与当前顶点邻接的未访问顶点（顺藤摸瓜，沿着一个分支越走越深）
  - 遇到死端时，沿着来路回退一条边，尝试继续从那里访问未访问的顶点 （回溯到上一个分岔口，选择下一条分支）
  - 当图中不存在未访问顶点时，算法结束

  - 可用栈来跟踪DFS的操作
    - 顶点在第一次访问时，入栈
    - 顶点成为死端时，出栈

DFS实例

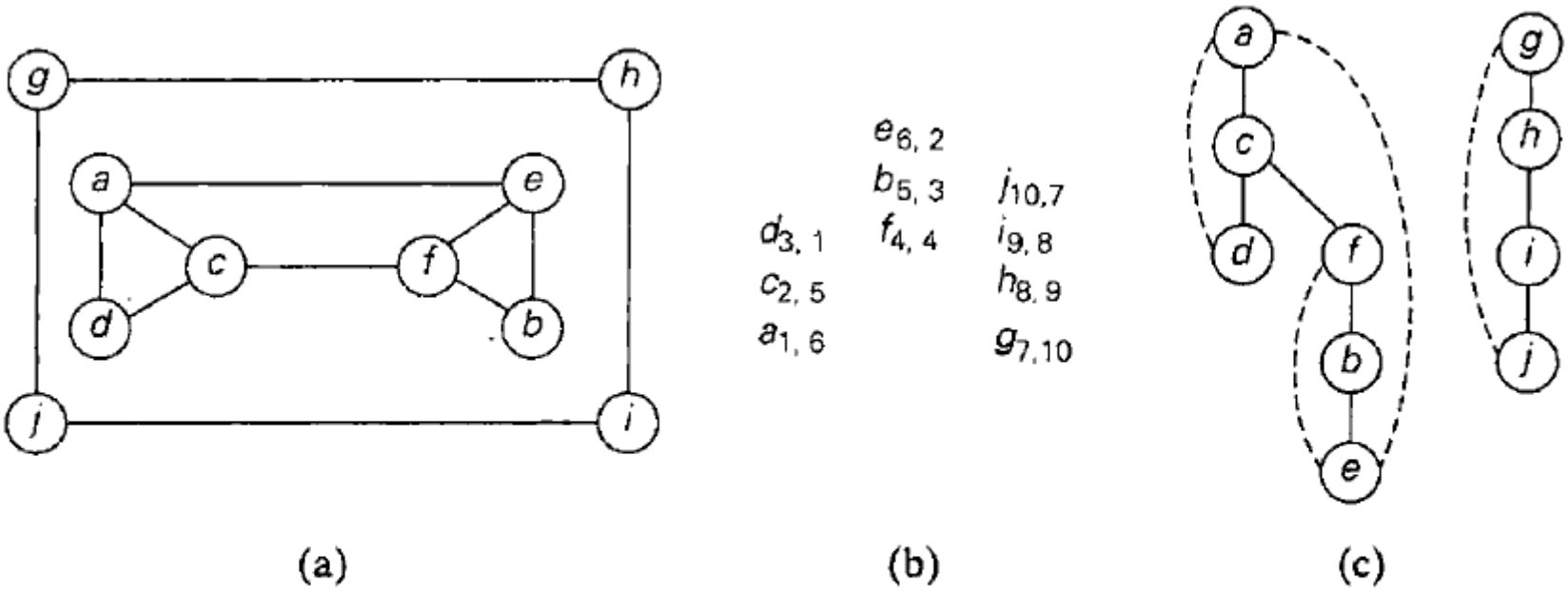


图 3.10　DFS 遍历的例子。(a)图：(b)遍历栈(第一个下标数字指出某个顶点被访问到的顺序，也就是入栈顺序，第二个下标数字指出该顶点变成终点的顺序，也就是出栈顺序)；(c)DFS 森林(树向边用实线表示，回边用虚线表示)

DFS算法伪代码

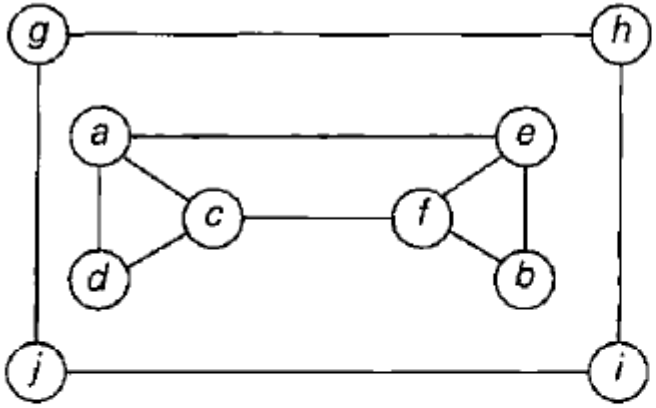
```
算法  DFS(G)
    //实现给定图的深度优先查找遍历
    //输入：图 G=<V, E>
    //输出：图 G 的顶点，按照被 DFS 遍历第一次访问到的先后次序，用连续的整数标记
    将 V 中的每个顶点标记为 0，表示还“未访问”
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            dfs (v)
dfs (v)
    //递归访问所有和 v 相连接的未访问顶点，然后按照全局变量 count 的值
    //根据遇到它们的先后顺序，给它们赋上相应的数字
    count ← count + 1; mark v with count
    for each vertex w in V adjacent to v do
        if w is marked with 0
            dfs(w)
```

广度优先查找

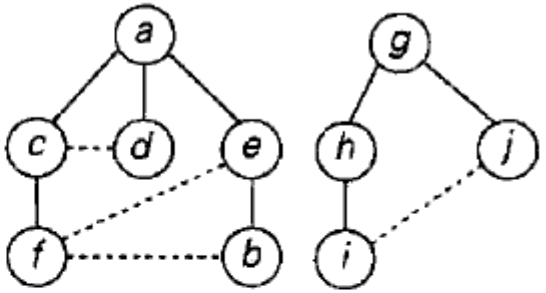
- 算法思想（BFS-Breath First Search）
  - 按照一种同心圆的方式，首先访问所有和初始顶点邻接的顶点，然后是离它两条边的所有未访问顶点，以此类推，直到所有与初始顶点同在一个连通分量中的顶点都已访问完
  - 如果仍存在未被访问的顶点，则从图的其它连通分量中的任意顶点重新开始

- 可用队列来跟踪BFS的操作
  - 找出所有和队头顶点邻接的未访问顶点，把它们标记为已访问，再把它们入队
  - 将队头顶点从队列中移除，出队

BFS实例



$a_1\ c_2\ d_3\ e_4\ f_5\ b_6$
$g_7\ h_8\ j_9\ i_{10}$



(a) (b) (c)

图 3.11 BFS 遍历的例子。(a)图；(b)遍历队列，其中的数字指出某个节点被访问到的顺序，也就是入队(或出队)顺序；(c)BFS 森林(树向边用实线表示，交叉边用虚线表示)

BFS算法伪代码

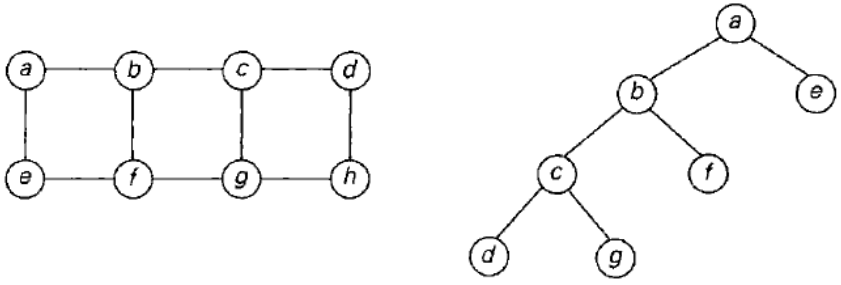
```
算法  BFS(G)
    //实现给定图的广度优先查找遍历
    //输入：图 G = <V, E>
    //输出：图 G 的顶点，按照被 BFS 遍历访问到的先后次序，用连续的整数标记
    将 V 中的每个顶点标记为 0，表示还“未访问”
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            bfs(v)
    bfs(v)
    //访问所有和 v 相连接的未访问顶点，然后按照全局变量 count 的值
    //根据访问它们的先后顺序，给它们赋上相应的数字
    count ← count + 1; mark v with count and initialize a queue with v
    while the queue is not empty do
        for each vertex w in V adjacent to the front vertex do
            if w is marked with 0
                count ← count + 1; mark w with count
                add w to the queue
        remove the front vertex from the queue
```

DFS versus BFS

	DFS	BFS
数据结构	栈(stack)	队列(queue)
边类型	树与回边(back edges)	树与交叉边(cross edges)
邻接链表的效率	$\Theta(\|V\|+\|E\|)$	$\Theta(\|V\|+\|E\|)$
邻接矩阵的效率	$\Theta(\|V\|^2)$	$\Theta(\|V\|^2)$
应用	判断是否有环 判断是否连通 求关节点 (articulation points)	判断是否有环 判断是否连通 求最短路径 (minimum-edge paths)

DFS & BFS 应用

- DFS求关节点
  - 特性：从图中移走关节点和所有它邻接的边之后，图被分为若干个不相交的部分
    - 对根节点u, 若它有两棵或两棵以上的子树，则该根节点u 为关节点
    - 对于非叶子节点&非根节点u, 若其子树的节点均没有指向u的祖先节点的回边，说明删除u之后，根节点与u的子树的节点不再连通；则节点u为关节点
- BFS求最短路径

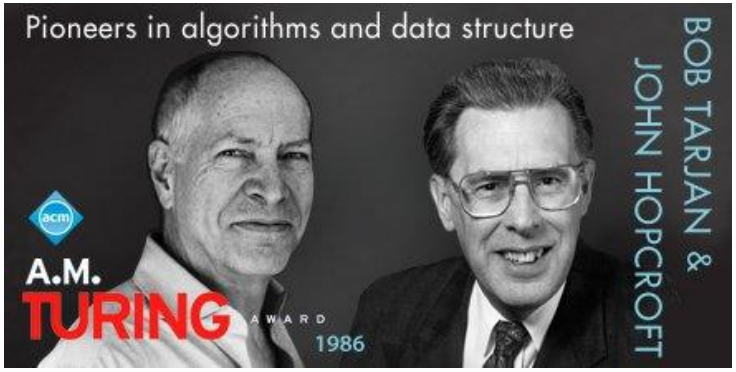


基于 BFS 求最少边路径的算法图示。(a)图；(b)BFS 树的一部分，确定了从 *a* 到 *g* 的最少边路径

John Hopcroft & Robert Tarjan

Turing Award in 1986 "for fundamental achievements in the design and analysis of algorithms and data structures"

One of their fundamental achievements was a linear-time algorithm for determining whether a graph is planar.



In general, graph algorithms require a systematic way of exploring a graph. The most important and common technique used by the planarity testing algorithms is the *Depth First Search*.

Depth first search (DFS) is a method for visiting all the vertices of a graph *G* in a specific way. It starts from an arbitrarily chosen vertex of *G* as a root node, and continues moving from the current vertex to an unexplored adjacent neighbor. When the current vertex has no unexplored adjacent vertices, the traversal backtracks to the first vertex with unexplored adjacent vertices. Look at the figures below for an example of *DFS*.

课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
3	3.1	9, 12b, 13	6, 7, 14
	3.2	6	3, 10, 11
	3.3	9	6
	3.4		9, 10, 11
	3.5	1, 4	10, 11

注：只需上交“课后作业题”；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年3月19日