北京理工大學
BEIJING INSTITUTEOF TECHNOLOGY

程序设计方法与实践
——蛮力法

高广宇

北京理工大学,计算机学院

蛮力法

蛮力法一种简单直接地解决问题的方法，常常直接基于问题的描述和所涉及的概念定义。

虽然巧妙和高效的算法很少来自于蛮力法，但我们不应该忽略它作为一种重要的算法设计策略的地位。

蛮力法

北京理工大学

- **简单直接地解决问题的方法，也就暴力法、枚举法、穷举法。——基本适合所有问题**
- **解题步骤——依靠循环**
  - （1）确定扫描和枚举变量。
  - （2）确定枚举变量的范围，设置相应的循环。
  - （3）根据问题的描述确定约束的条件，以便找到合理的解。

蛮力法的特点

北京理工大学

- 第一，和其他某些策略不同，我们可以应用蛮力法来解决广阔领域的各种问题。
- 第二，对于一些重要的问题来说（比如：排序、查找、矩阵乘法和字符串匹配），蛮力法可以产生一些合理的算法，它们多少具备上些实用价值，而且并不限制实例的规模。
- 第三，如果要解决的问题实例不多，而且蛮力法可以用一种能够接受速度对实例求解，那么，设计一个更高效算法所花费的代价很可能是不值得的。
- 第四，即使效率通常很低，仍然可以用蛮力法解决一些小规模的问题实例。
- 第五，一个蛮力算法可以为研究或教学目的的服务。

蛮力法

北京理工大学

- 1、选择排序和冒泡排序
- 2、顺序查找和蛮力字符串匹配
- 3、最近对和凸包问题蛮力法
- 4、穷举查找
- 5、深度优先和广度优先查找

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 1）从$i=0$开始扫描列表从i到末尾，找到最小元素
- 2）最小元素与i元素交换位置。
- 3）$i++$返回

算法 SelectionSort ($A[0 \ldots n-1]$)

//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
//输出：升序排列的数组A

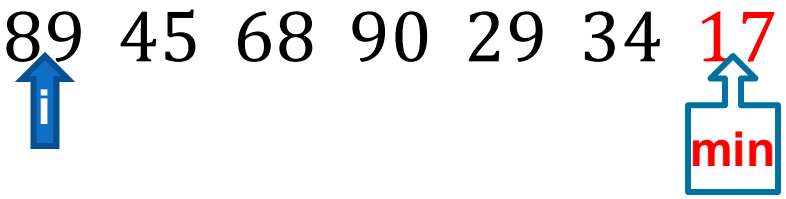
$for\ i \leftarrow 0\ to\ n-2\ do$
    $min \leftarrow i$
    $for\ j \leftarrow i+1\ to\ n-1\ do$
        $if\ A[j] < A[min] \quad min \leftarrow j$
    $swap\ A[i]\ and\ A[min]$

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

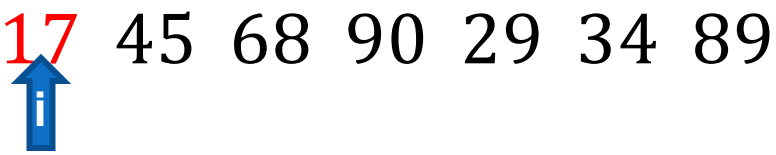


1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

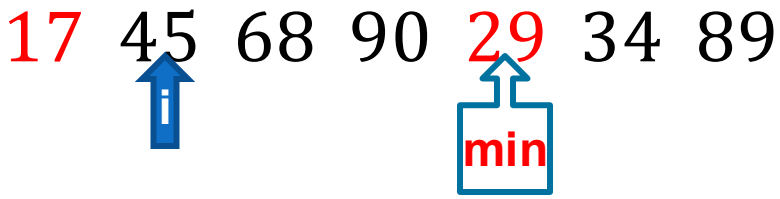


1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

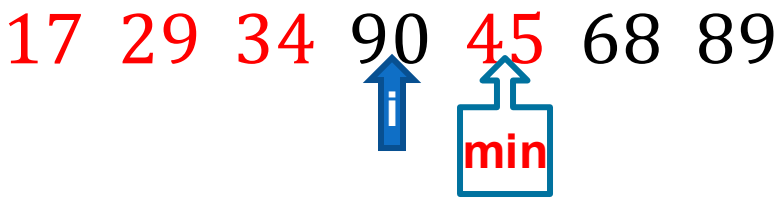
- 举例：89,45,68,90,29,34,17

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

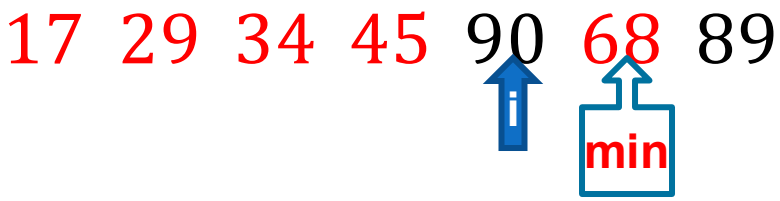
17 29 34 45 90 68 89

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

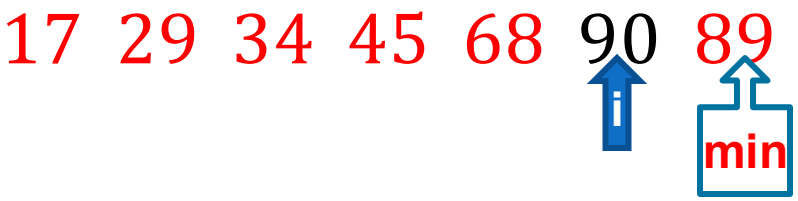
- 举例：89,45,68,90,29,34,17

1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 选择排序

- 举例：89,45,68,90,29,34,17

89	45	68	90	29	34	**17**
17 |	45	68	90	**29**	34	89
17	29 |	68	90	45	**34**	89
17	29	34 |	90	**45**	68	89
17	29	34	45 |	90	**68**	89
17	29	34	45	68 |	90	**89**
17	29	34	45	68	89 |	90

17 29 34 45 68 89 90

i

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

- 算法比较次数为$\Theta(n^2)$，交换次数为$\Theta(n)$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 比较相邻元素，逆序则交换，直到最大元素沉底。则完成一个最大元素的排序。循环执行n次。

算法 BubbleSort $(A[0 \ldots n-1])$

//该算法用冒泡排序对给定的数组排序
//输入：一个可排序数组A
//输出：升序排列的数组A

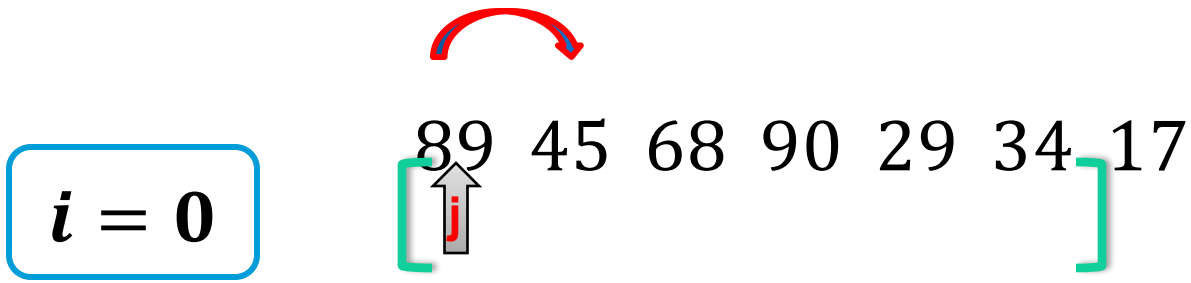
$for\ i \leftarrow 0\ to\ n-2\ do$
  $for\ j \leftarrow 0\ to\ n-2-i\ do$
    $if\ A[j+1] < A[j] \quad swap\ A[j]\ and\ A[j+1]$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

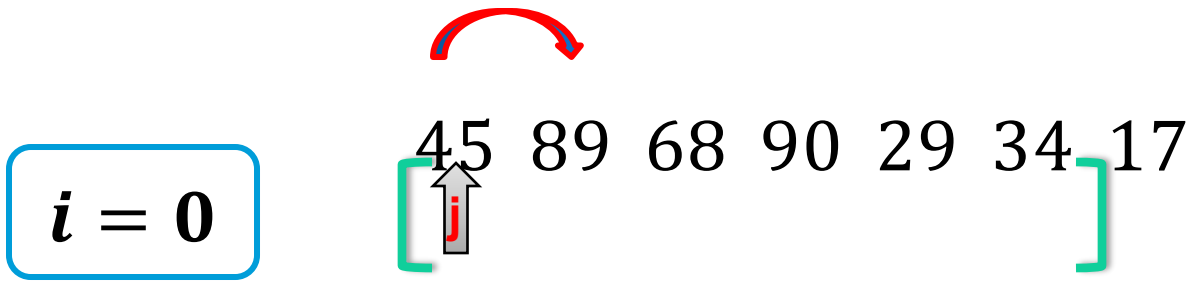


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



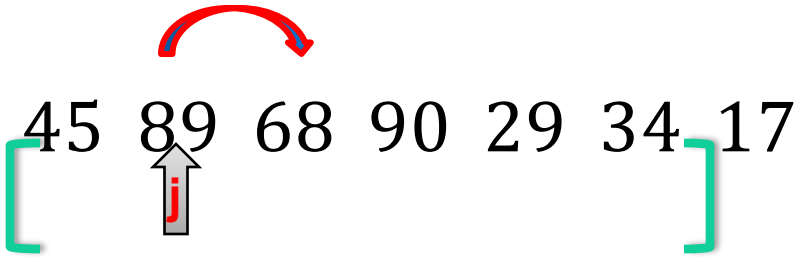
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$

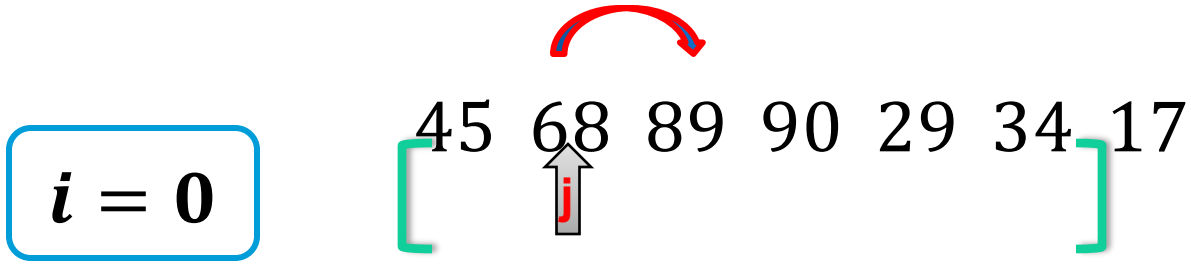


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

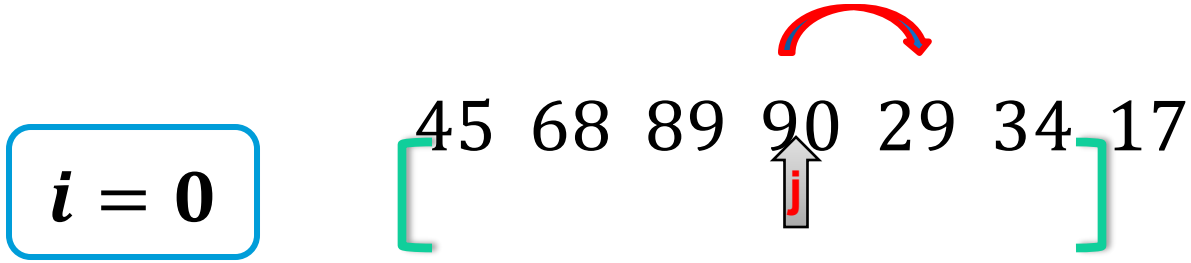


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



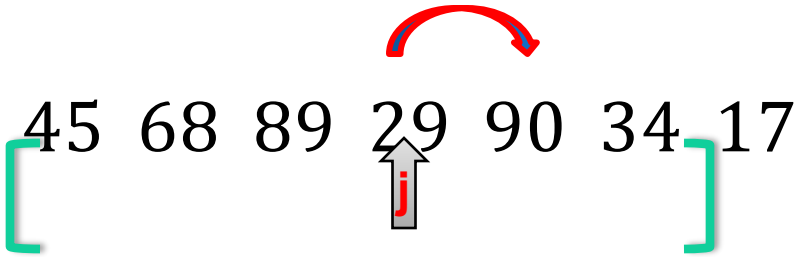
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$



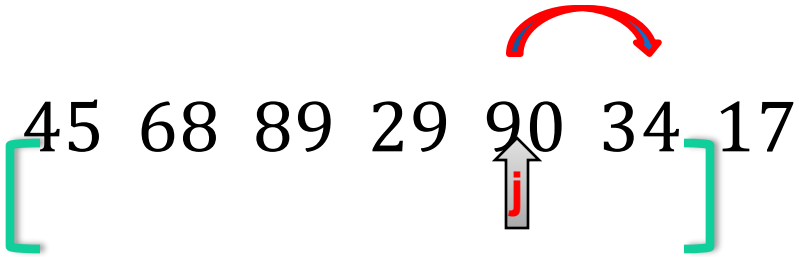
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$



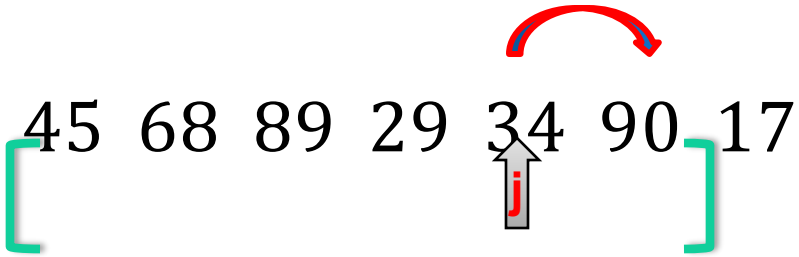
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$



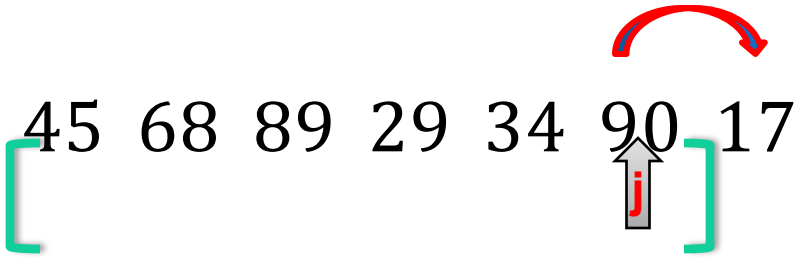
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$



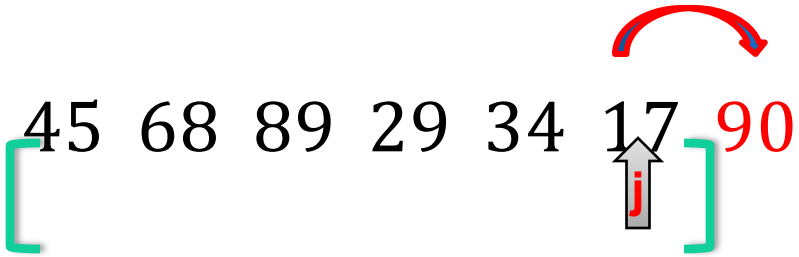
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 0$



1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$

45 68 89 29 34 17 90

j

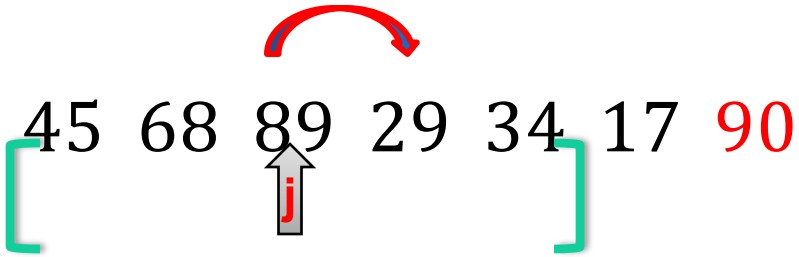
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



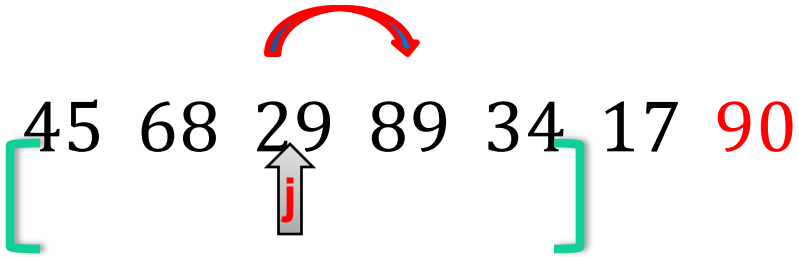
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



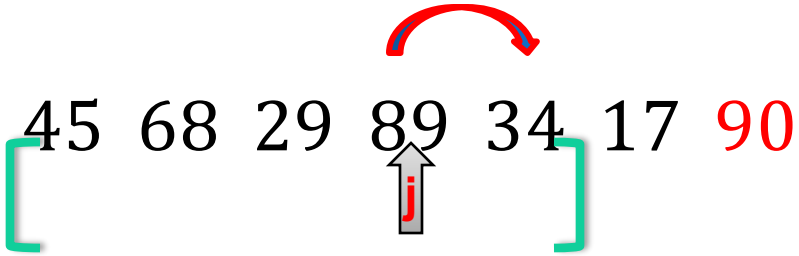
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



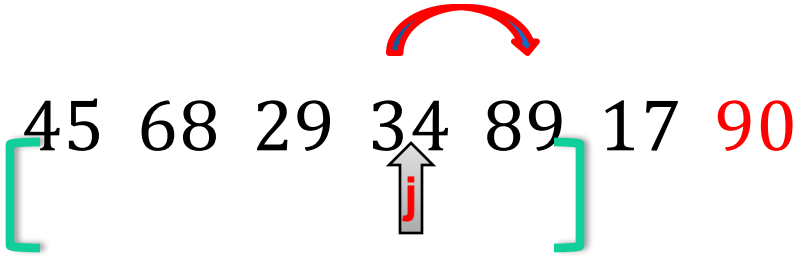
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



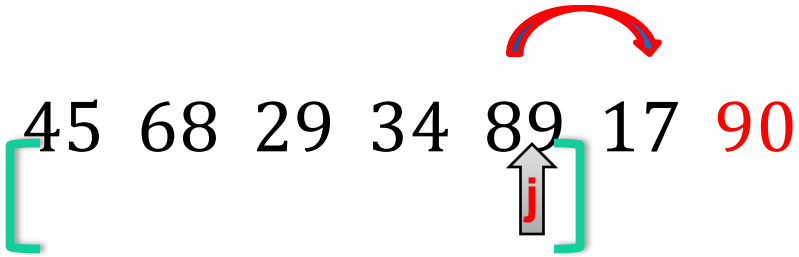
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



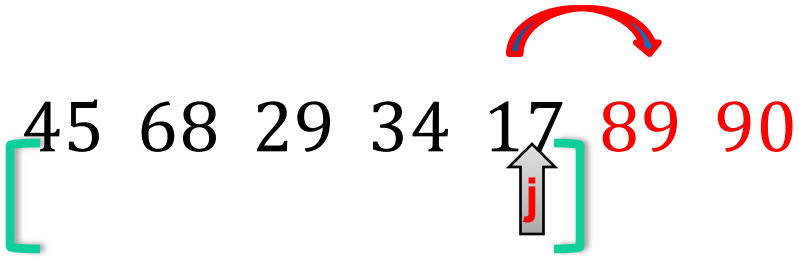
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 1$



1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 2$

45 68 29 34 17 89 90

[j]

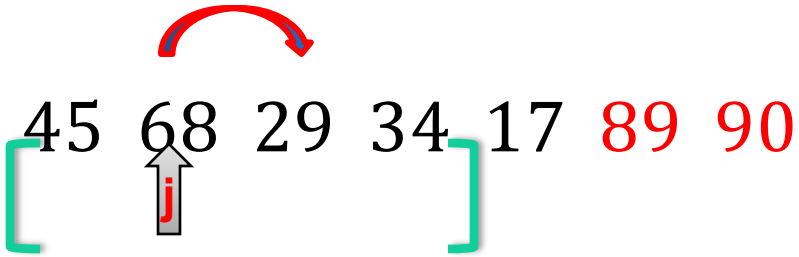
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 2$

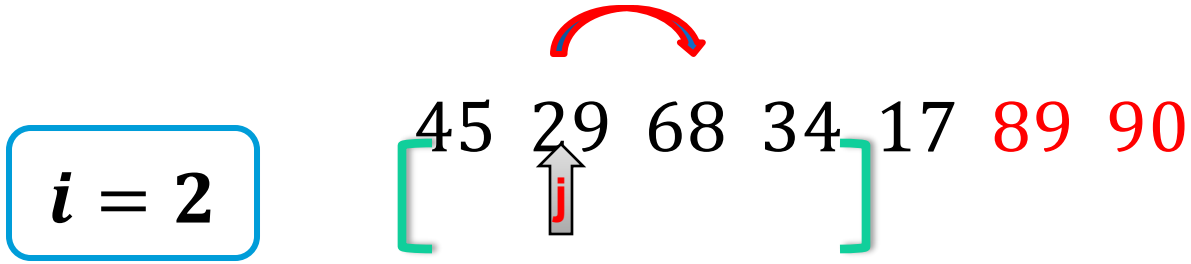


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

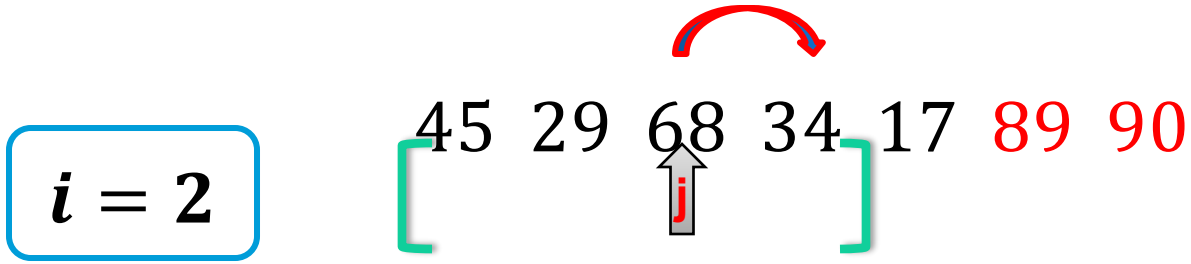


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



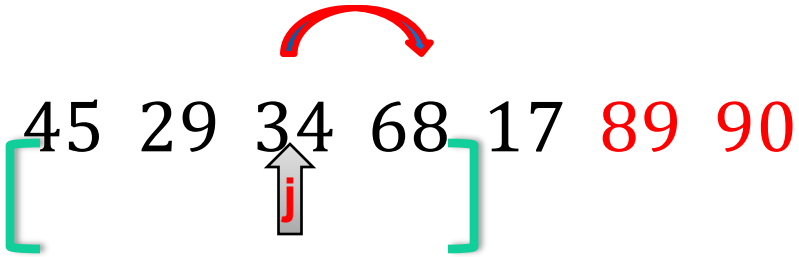
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 2$



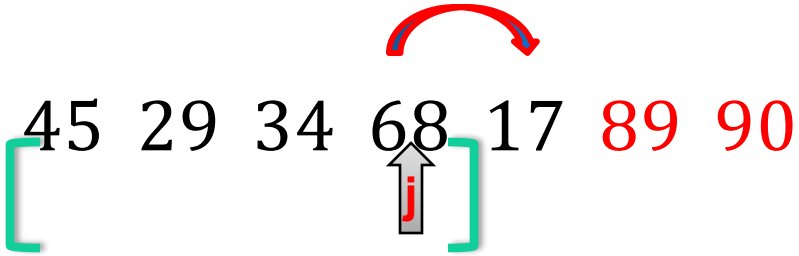
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 2$



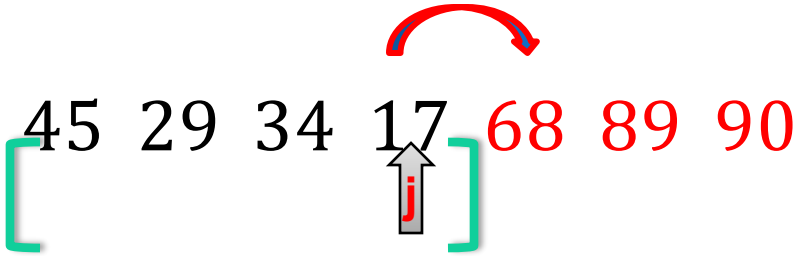
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 2$

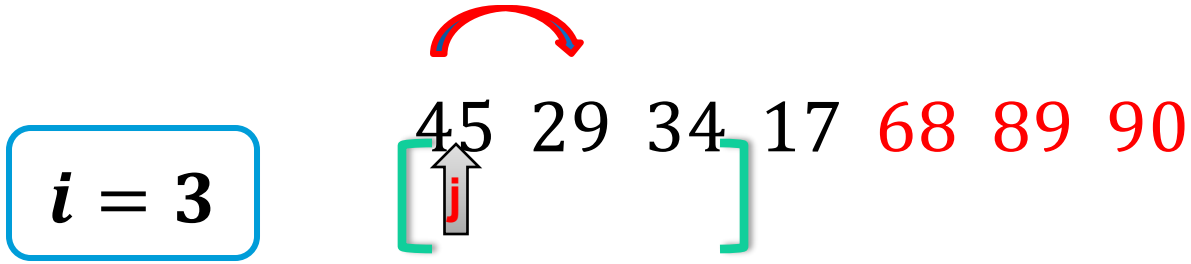


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



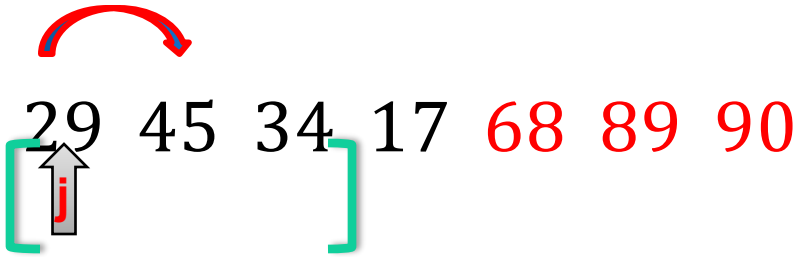
1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 3$

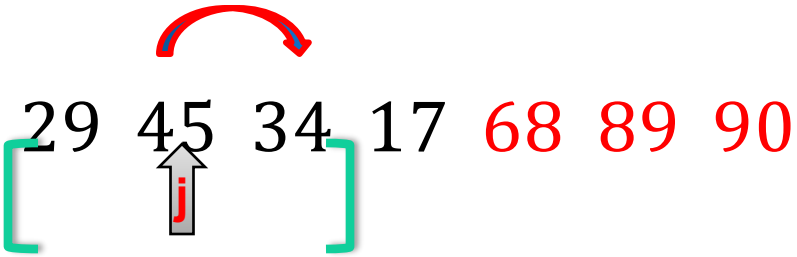


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



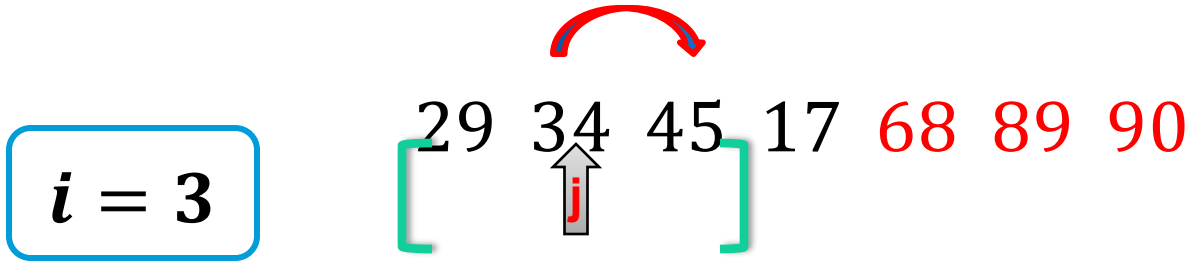
$i = 3$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

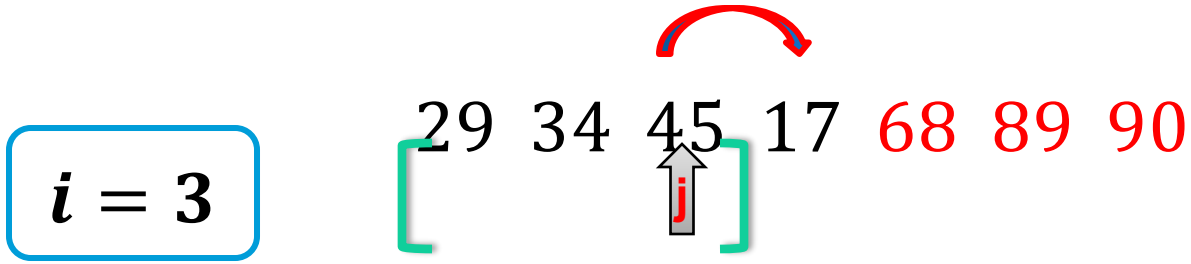


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

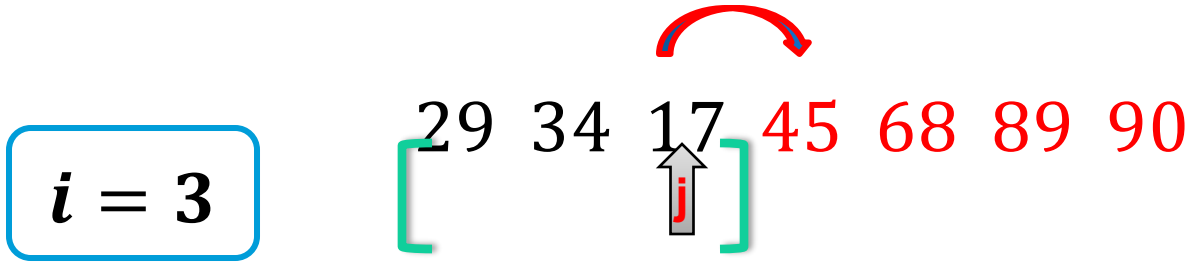


1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 4$

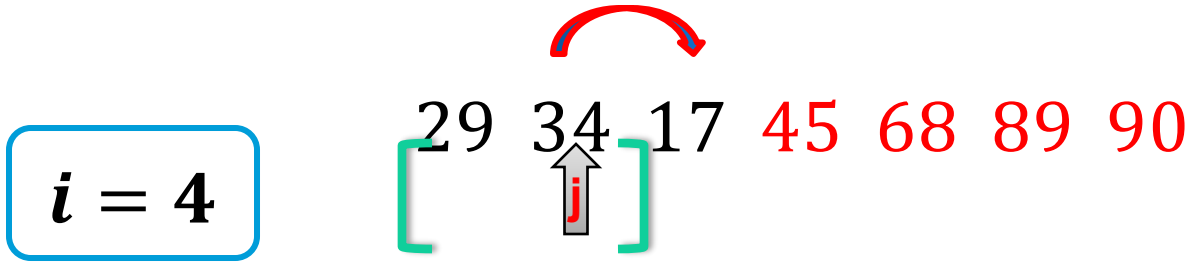
29 34 17 45 68 89 90

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 4$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

29 17 34 45 68 89 90

$i = 5$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$i = 5$

1、选择排序和冒泡排序

北京理工大学

- 对于所有规模为n的数组来说,该冒泡排序版本的键值比较次数都是相同的,我们可以用下面这个求和表达式来表示。它和选择排序的表达式几乎是完全相同的:

$$C(n)=\sum_{i=0}^{n-2}\sum_{j=0}^{n-2-i}1=\sum_{i=0}^{n-2}[(n-2-i)-0+1]$$

$$=\sum_{i=0}^{n-2}(n-1-i)=\frac{(n-1)n}{2}\in\Theta(n^2)$$

- 但它的键交换次数取决于特定输入。最坏的情况下就是遇到降序的数组，这时，键交换次数和键比较次数是相同的:

$$S_{\text{worst}}(n)=C(n)=\frac{(n-1)n}{2}\in\Theta(n^2)$$

1、选择排序和冒泡排序

北京理工大学

➢ 冒泡排序

- 举例：89,45,68,90,29,34,17

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$
$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

算法交换次数　$S_{worst}(n) = C(n) \in \Theta(n^2)$

蛮力法

北京理工大学

- 1、选择排序和冒泡排序
- 2、顺序查找和蛮力字符串匹配
- 3、最近对和凸包问题蛮力法
- 4、穷举查找
- 5、深度优先和广度优先查找

2、顺序查找和蛮力字符串匹配

➢ 顺序查找

- 使用限位器，节省循环内的范围比较操作

算法 SequentialSearch2 ($A[0 \ldots n]$,K)

//顺序查找算法，使用查找键做限位器
//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的元素位置，找不到返回-1
$A[n] \leftarrow K$
$i \leftarrow 0$
$while\ A[i] \neq K\ do$
    $i \leftarrow i + 1$
$if\ i < n\ \ return\ i$
$else\ return\ -1$

顺序查找是阐释蛮力法的很好的工具，有着蛮力法典型的优点（简单）和缺点（效率低）。

2、顺序查找和蛮力字符串匹配

蛮力字符串匹配

- 给定一个n个字符的串（称为文本）
- 给定一个m个字符的串（称为模式，$m \leq n$)
- 从文本中寻找匹配模式的子串。

$$t_0 \cdots t_i \cdots t_{i+j} \cdots t_{i+m-1} \cdots t_{n-1}$$ 文本T

$$p_0 \cdots p_j \cdots p_{m-1}$$ 模式P

2、顺序查找和蛮力字符串匹配

➢ 蛮力字符串匹配

算法 BruteForceStringMatch ($T[0 \ldots n-1], P[0 \ldots m-1]$)

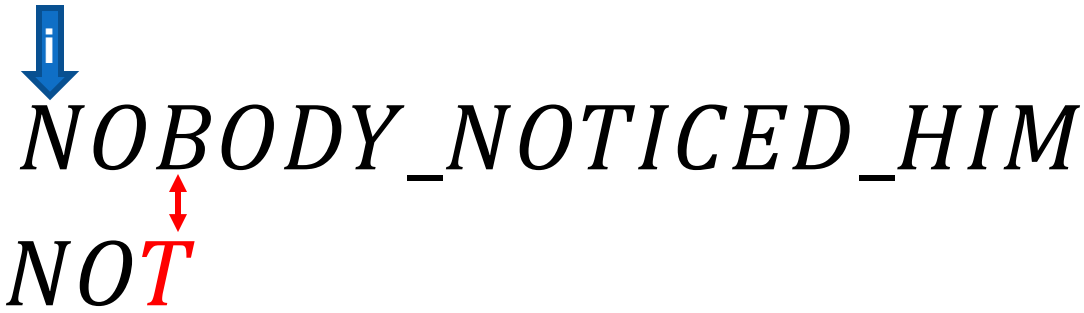
```
//蛮力字符串匹配
//输入：  一个n个字符的数组T代表文本
//        一个m个字符的数组P代表模式
//输出：  文本第一个匹配子串中第一个字符位置
//        找不到返回-1
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
        if j = m  return i
return −1
```

2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

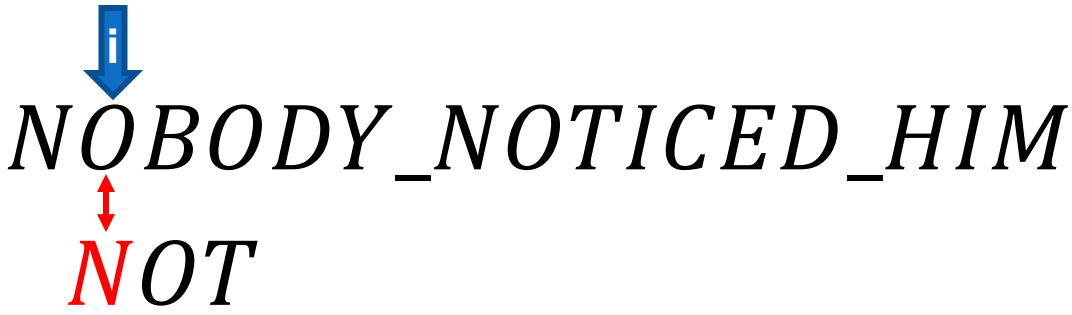


2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

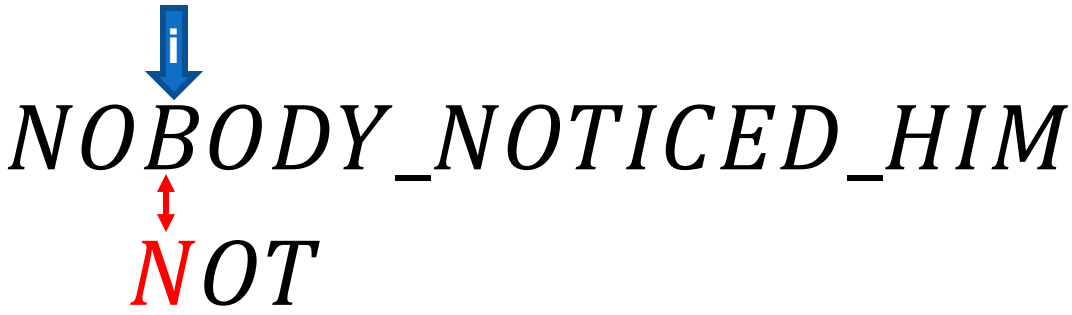


2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT



2、顺序查找和蛮力字符串匹配

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

2、顺序查找和蛮力字符串匹配

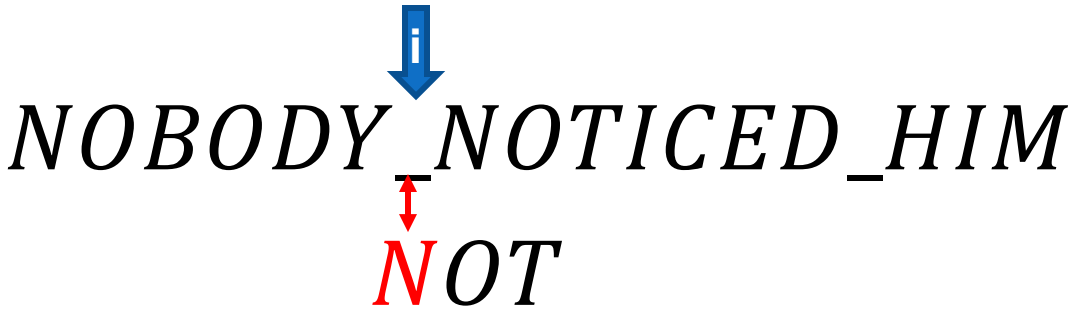
➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

2、顺序查找和蛮力字符串匹配

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT

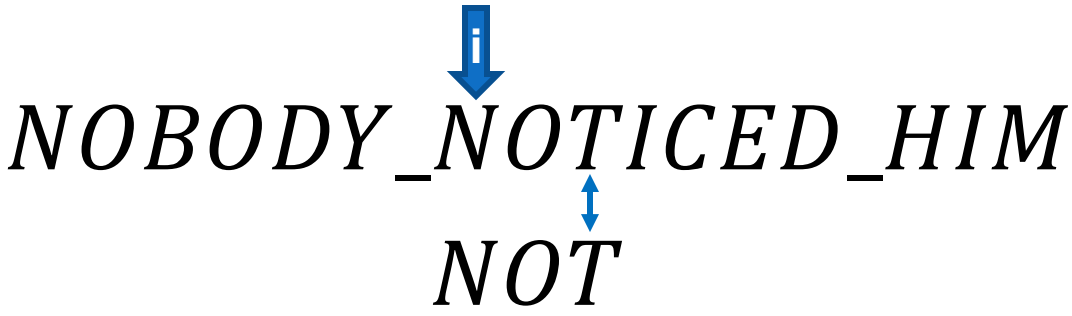


2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 举例：NOBODY_NOTICED_HIM，匹配NOT



算法最差比较次数 $C_{worst}(n,m) \in O(nm)$

2、顺序查找和蛮力字符串匹配

北京理工大学

➢ 蛮力字符串匹配

- 比较次数: 待检索文本长度 n, Pattern长度m
- 最差效率：$\Theta(n*m)$　　000 000 000 000
- 平均效率：$\Theta(n+m)$　　001

- 最好情况：第 i 个位置匹配成功，比较了 $(i-1+m)$ 次，平均比较次数：

$$\sum_{i=0}^{n-m} p(i-1+m) = \frac{1}{n-m+1}\sum_{i=0}^{n-m}(i-1+m)$$

$$= \frac{n+m}{2} - 1 = \Theta(n+m)$$

蛮力法

- 1、选择排序和冒泡排序
- 2、顺序查找和蛮力字符串匹配
- 3、最近对和凸包问题蛮力法
- 4、穷举查找
- 5、深度优先和广度优先查找

3、最近对和凸包问题蛮力法

最近对问题

- 在包含 n 个点的集合中，找出距离最近的两个点。

3、最近对和凸包问题蛮力法

最近对问题

- 在包含n个点的集合中，找出距离最近的两个点。
- 问题中的点可以代表飞机、邮局、数据库记录等。

3、最近对和凸包问题蛮力法

最近对问题

- 在包含n个点的集合中，找出距离最近的两个点。
- 问题中的点可以代表飞机、邮局、数据库记录等。
- 统计学中的聚类分析。

求解该问题的蛮力法：分别计算每一对点之间的距离，然后找出距离最小的一对。

3、最近对和凸包问题蛮力法

北京理工大学

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。

算法 BruteForceClosestPoints($p[0 \ldots n-1]$)

//蛮力算法求平面中距离最近的两个点
//输入：包含m个点的列表p
//输出：两个最近点的距离

$d \leftarrow \infty$

$for\ i \leftarrow 0\ to\ n-2\ do$

$\quad for\ j \leftarrow i+1\ to\ n-1\ do$

$\quad\quad d \leftarrow \min(d, distance(P_i, P_j))$

$return\ d$

3、最近对和凸包问题蛮力法

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。
- 基本操作：

3、最近对和凸包问题蛮力法

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。
- 基本操作：**计算平方根**。

3、最近对和凸包问题蛮力法

北京理工大学

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。
- 基本操作：**计算平方根**。

  计算平方根像加法或者乘法一样简单?

3、最近对和凸包问题蛮力法

北京理工大学

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。
- 基本操作：**计算平方根**。

  计算平方根像加法或者乘法一样简单？×

  整数的平方根也多是无理数，需要近似解，非常麻烦。

3、最近对和凸包问题蛮力法

北京理工大学

最近对问题

- 问题：在包含n个点的集合中，找出距离最近的两个点
- 蛮力法求解：分别计算每一对点之间的距离，然后找出距离最小的一对。
- 基本操作：**计算平方根**。

  计算平方根像加法或者乘法一样简单？×

  整数的平方根也多是无理数，需要近似解，非常麻烦。

  避免求平方根！窍门是忽略平方根函数。

3、最近对和凸包问题蛮力法

北京理工大学

➢ 最近对问题

- 在包含n个点的集合中，找出距离最近的两个点

基本操作：

$$distance(P_i, P_j) = sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$$

3、最近对和凸包问题蛮力法

北京理工大学

➢ 最近对问题

- 在包含n个点的集合中，找出距离最近的两个点

基本操作：

$$distance(P_i, P_j) = sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$$

可忽略开方操作，则基本操作为2次平方操作。

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 2 = 2 \sum_{i=0}^{n-2} [n-1-i] = \frac{(n-1)n}{2} \in \Theta(n^2)$$

3、最近对和凸包问题蛮力法

凸包问题

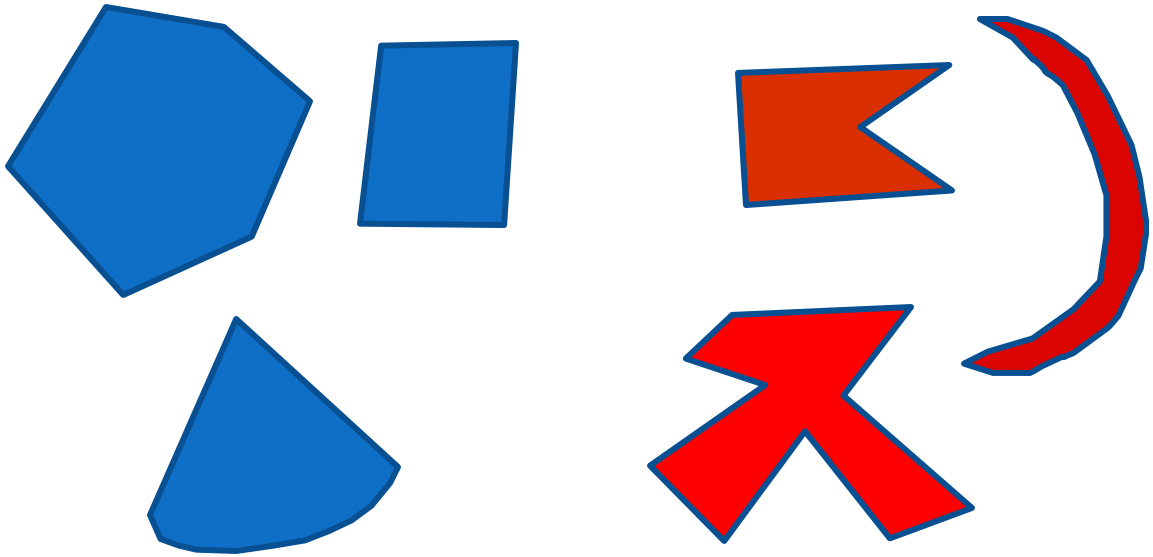
- 在平面或高维空间给定集合中寻找凸包被视为计算集合中最重要的问题之一。
- 凸包能方便地提供目标形状或给定数据集的近似。
- 例如，计算机动画中用物体的凸包替换物体本身，加速碰撞检测；
- 地理信息系统中根据凸包问题根据卫星图像计算可通达地形图（accessibility map）；
- 数理统计方法利用该技术进行异常检测；

3、最近对和凸包问题蛮力法

北京理工大学

凸包问题

- 凸集合：对于平面上的一个点集合（有限的或无限的），如果以集合中任意两点p和q为端点的线段都属于该集合，则此集合是凸的



3、最近对和凸包问题蛮力法

北京理工大学

凸包问题

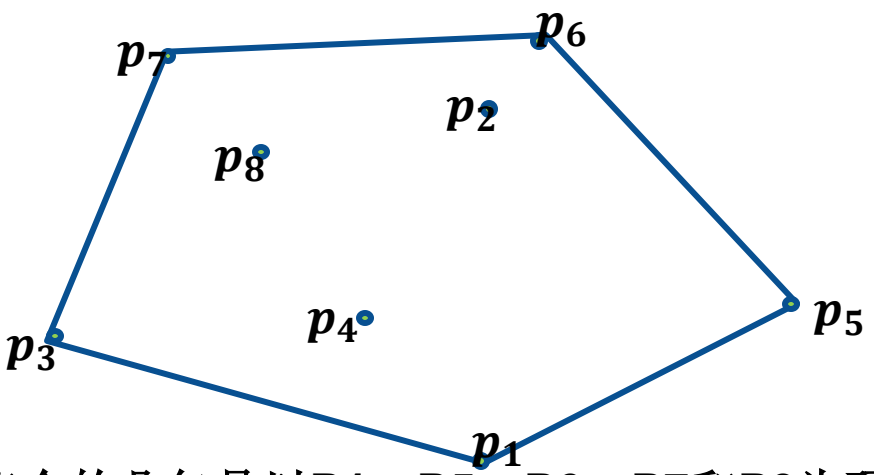
- 凸包：一个点集S的凸包(convex hull) 是包含S的最小凸集合(“最小”是S的凸包一定是所有包含S的凸集合的子集)。

3、最近对和凸包问题蛮力法

北京理工大学

凸包问题

- 定义：一个点集S的凸包(convex hull) 是包含S的最小凸集合(“最小”是S的凸包一定是所有包含S的凸集合的子集)。
- 定理：任意包含$n > 2$个点（不共线）的集合S的凸包是以S中某些点为顶点的凸多边形。

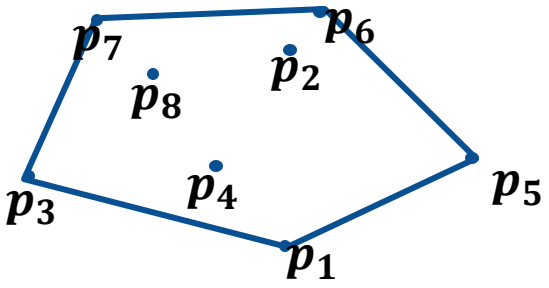


图中8个点的集合的凸包是以P1，P5，P6，P7和P3为顶点的凸多边形

3、最近对和凸包问题蛮力法

北京理工大学

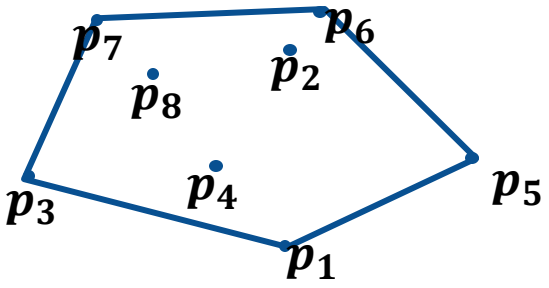
- 凸包问题是为一个n个点的集合构造凸包的问题。为了解决该问题，**需要找出某些点，它们将作为这个集合的凸多边形的顶点**。数学家将这种多边形的顶点称为“极点”。
- 一个凸集合的极点是这个集合中这样的点：**对于任何以集合中的点为端点的线段来说，它们不是这种线段的中点**。例如，一个三角形的极点是它的3个顶点，一个圆形的极点是它圆周上的所有点，对于图中8个点的集合来说，它的凸包的极点是P_1，P_5，P_6，P_7和P_3。



3、最近对和凸包问题蛮力法

北京理工大学

- 凸包问题的简单但直接的算法：对于一个n个点集合中的两个点p_i和p_j，当且仅当该集合中的其他点都位于穿过这两点的直线的同一边时，他们的连线是该集合凸包边界的一部分。
- 对每一对点都做一遍检验之后，满足条件的线段构成了该凸包的边界。



蛮力法解决凸包问题

北京理工大学

- step1：对于一个n个点集合中的两个点P_i, Pj，当且仅当该集合中的其他点都位于穿过这两点的直线的同一边时，它们的连线是该集合凸包边界的一部分。

- step2：对每一对点都做一遍检查之后，满足条件的线段构成该凸包的边界。

3、最近对和凸包问题蛮力法

北京理工大学

➢凸包问题

- 1、取集合n中的两个点i,j点。
- 2、在坐标平面上穿过两点的直线的构造方程:

$$ax + by = c,$$

其中，

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

- 3、若n中所有其他点代入式中均$\geq c$或均$\leq c$，说明其他点都在i,j点一侧。i,j边为凸包一个边。

$$C(n) \in \Theta(n^3)$$

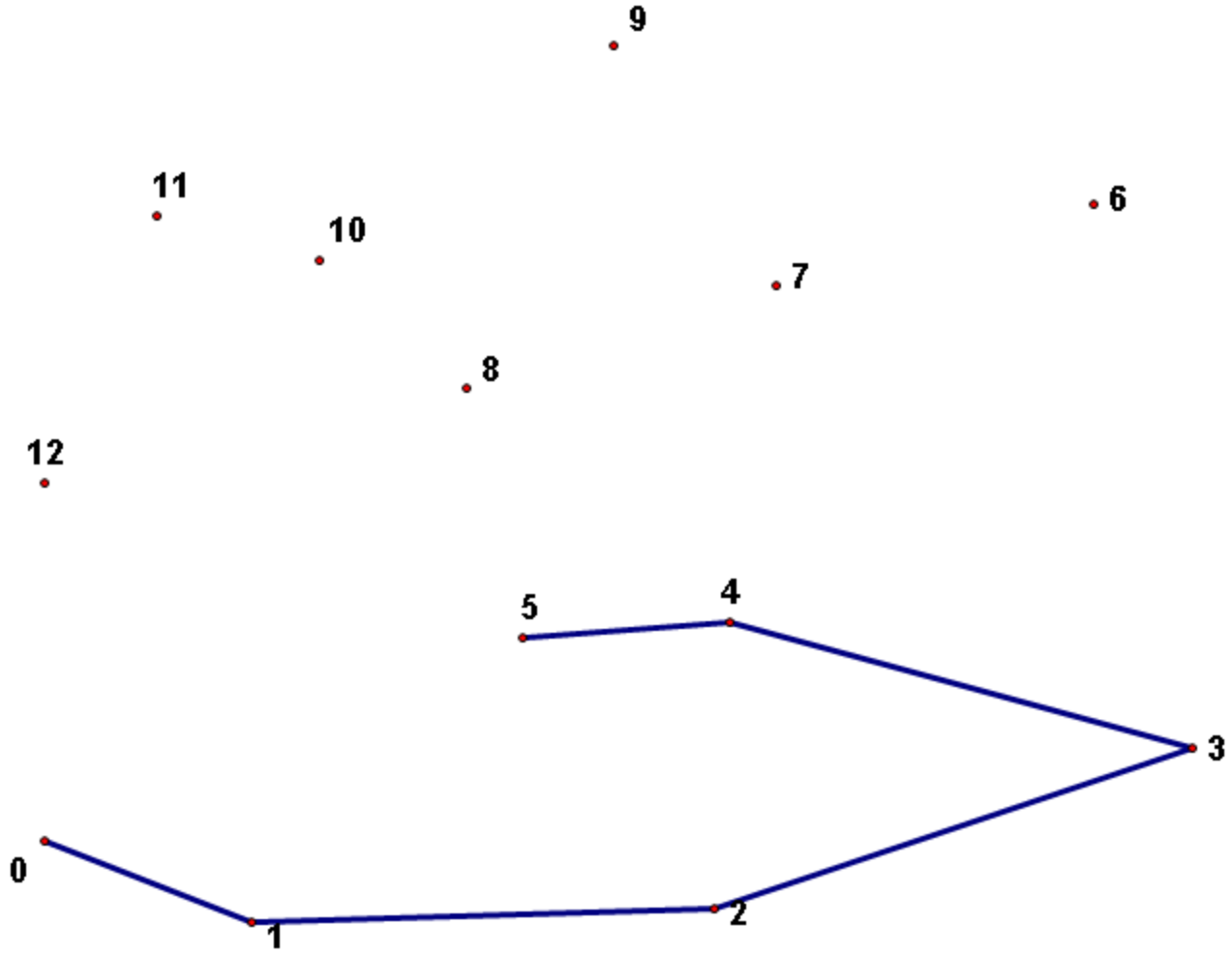
凸包问题的Graham-Scan算法

- Graham的扫描是一个很优美的过程，用到的数据结构也很简单，仅仅是一个栈而已；
- 核心思想是按照排好的序，依次加入新点得到新的边；
- 如果和上一条边成左转关系就压栈继续；如果右转就弹栈直到和栈顶两点的边成左转关系，压栈继续；
- 实现的时候我们不用存边，只需要含顺序在栈里存点，相邻两点就是一条边；
- 由于我们时时刻刻都保证栈内是一个凸壳，所以最后扫描完毕，就得到了一个凸包。

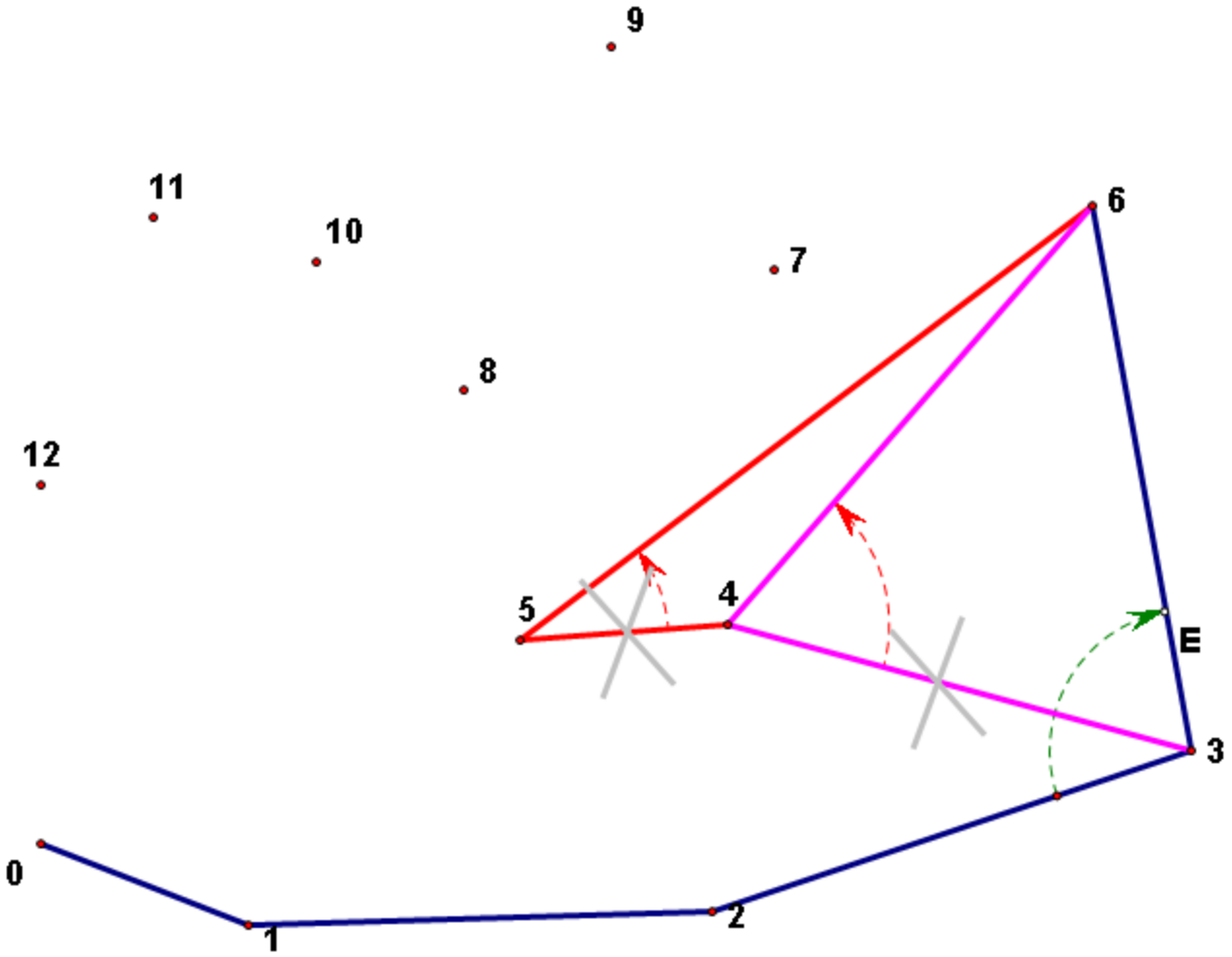
凸包问题的Graham-Scan算法

北京理工大学

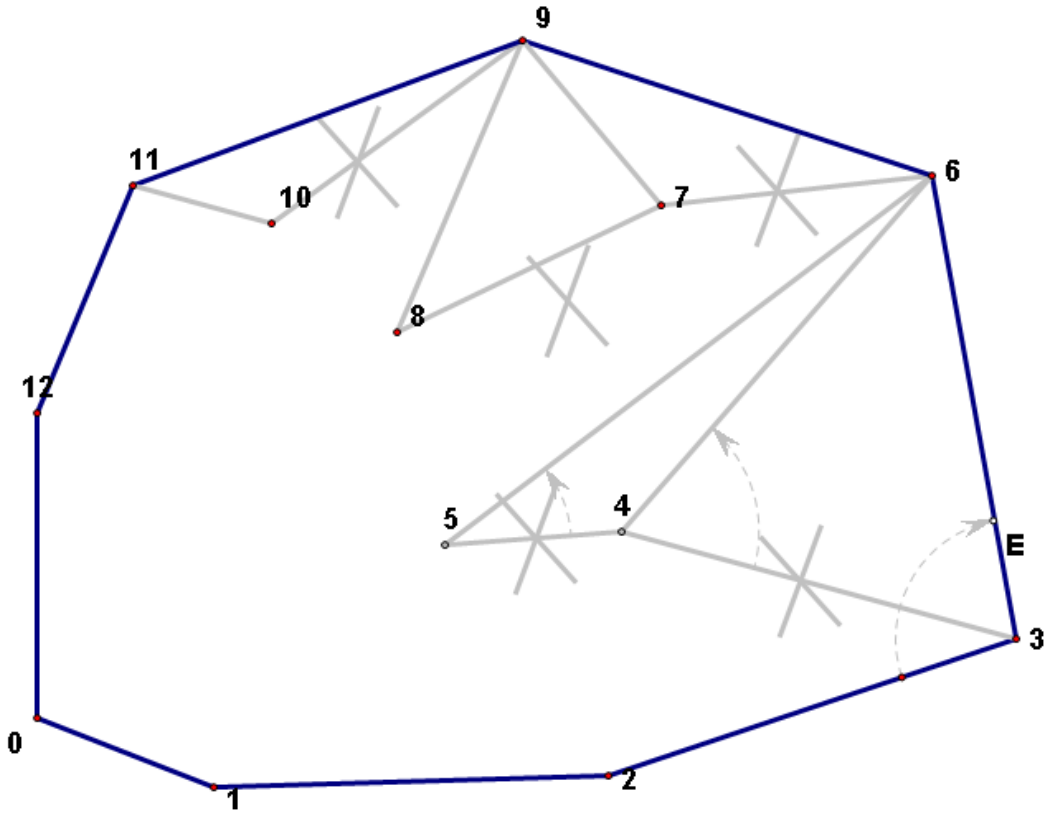
- 下面演示一下栈扫描的过程



凸包问题的Graham-Scan算法



凸包问题的Graham-Scan算法



这样Graham扫描算法基本完成
复杂度是排序$O(N\log_2 N)$，扫描$O(N)$ {每个点仅仅出入栈一次}，合起来是$O(N\log_2 N)$的算法。

凸包问题的Graham-Scan算法

北京理工大学

```
GRAHAM-SCAN(Q)
1  let p0 be the point in Q with the minimum y-coordinate,
          or the leftmost such point in case of a tie
2  let⟨p1, p2, ···, pm⟩be the remaining points in Q,
          sorted by polar angle in counterclockwise order around p0
          (if more than one point has the same angle, remove all but
          the one that is farthest from p0)
3  PUSH(p0, S)
4  PUSH(p1, S)
5  PUSH(p2, S)
6  for i ← 3 to m
7       do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                    and pi makes a nonleft turn
8             do POP(S)
9          PUSH(pi, S)
10 return S
```

蛮力法

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

4、穷举查找

北京理工大学

- 要求生成问题域中每个元素，选出满足问题约束的元素，然后找出一个期望元素（最优化元素）
  - 旅行商问题
  - 背包问题
  - 分配问题

4、穷举查找

对于组合问题来说，穷举查找是一种简单的蛮力方法。它要求生成问题域中的每一个元素，选出其中满足问题约束的元素，然后再找出一个期望元素（例如，使目标函数达到最值的元素）。

4.1 旅行商问题（TSP）

按照非专业的说法，找出一条n个给定城市间的最短路径，使我们在回到出发城市前，对每个城市都只访问一次，即最短哈密顿回路问题。

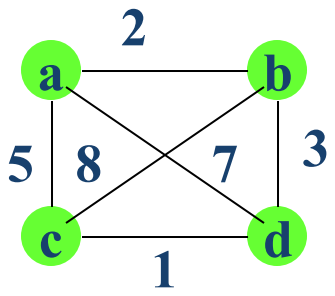
哈密顿回路：图中的每个顶点只穿越一次的回路。

4、穷举查找

北京理工大学

➢旅行商问题

- 找出n个给定城市间的最短路径，回到出发的城市前，每个城市只访问一次。
- 求图的最短哈密顿回路(Hamiltonian circuit)
- 哈密顿回路为$n+1$个相邻顶点构成的序列 $V_{i_0}, V_{i_1}, \cdots, V_{i_{n-1}}, V_{i_0}$
- 开始和结束设定为V_{i_0}，则需要生成$n-1$个中间城市的组合得到所有路线，从中选择最优路线。



路线	旅程	
a→b→c →d → a	I=2+8+1+7=18	
a→b→d →c →a	I=2+3+1+5=11	最佳
a→c→b →d →a	I=5+8+3+7=23	
a→c→d →b →a	I=5+1+3+2=11	最佳
a→d→b →c →a	I=7+3+8+5=23	
a→d→c →b →a	I=7+1+8+2=18	

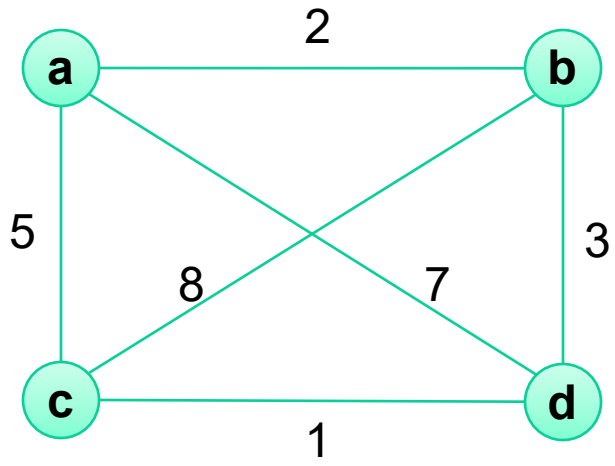
用穷举查找对一个小规模旅行商问题的求解过程

- 对于TSP问题，可通过生成$n-1$个中间城市（节点）组合来得到所有的旅行路径，计算这些路径的长度，然后求得最短的路径。
- 对无向图，同一节点序列的路径方向是双向的，对回路来说代价一样，因此只需考虑一个方向，即节点排列总次数是$(n-1)!/2$

4、穷举查找

北京理工大学

旅行商问题



路线	旅程
$a \to b \to c \to d \to a$	$l = 2 + 8 + 1 + 7 = 18$
$a \to b \to d \to c \to a$	$l = 2 + 3 + 1 + 5 = 11$
$a \to c \to b \to d \to a$	$l = 5 + 8 + 3 + 7 = 23$
$a \to c \to d \to b \to a$	$l = 5 + 1 + 3 + 2 = 11$
$a \to d \to b \to c \to a$	$l = 7 + 3 + 8 + 5 = 23$
$a \to d \to c \to b \to a$	$l = 7 + 1 + 8 + 2 = 18$

- $S(n) = (n-1)!$
- 考虑到反向路径长度相同，可以减少一半
- $S(n) = (n-1)!/2$

4、穷举查找

北京理工大学

➢ 旅行商问题

1. **a.** 假设每一条旅行路线都能够在固定的时间内生成出来，对于书中描述的旅行商问题的穷举查找算法来说，它的效率类型是怎样的？

b. 如果该算法的实现运行在一台每秒能做 10 亿次加法的计算机上，请估计在下述时间中，该算法能够处理的城市个数。

i. 1 小时　　**ii.** 24 小时　　**iii.** 1 年　　**iv.** 100 年

4、穷举查找

北京理工大学

背包问题

- 给定n个重量为$w_1, w_2, \ldots, w_n$,价值为$v_1, v_2, \ldots, v_n$的物品和一个承重为W的背包，求这些物品中最有价值的一个子集，也就是最多装多少价值物品?
  - 穷举查找n个物品的所有子集
  - 总重量不超过承重能力
  - 找出价值最大的

4、穷举查找

北京理工大学

背包问题

- 例子：背包体积W=10。物品1(体积7,价值42),物品2(3,12),物品3(4,40),物品4(5,25)

子集	总重量	总价值
∅	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

4、穷举查找

北京理工大学

背包问题

- 例子：背包体积W=10。物品1(体积7,价值42),物品2(3,12),物品3(4,40),物品4(5,25)

子集	总重量	总价值
{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

4、穷举查找

北京理工大学

- **背包问题**
  - 例子：背包体积W=10。物品1(体积7,价值42),物品2(3,12),物品3(4,40),物品4(5,25)
  - n个元素的子集个数为2^n,以查找子集为基准的算法复杂度$\in \Omega(2^n)$
- **旅行商问题和背包问题都是NP(非多项式)问题，目前没有已知的效率可以用多项式来表示的算法。**

4、穷举查找

北京理工大学

分配问题

- n 个任务分配给 n 个人执行，每个任务一个人。对每一对$i, j \in 1,2,\ldots,n$来说，将j个任务分配给第i人的成本为$C[i,j]$。找出最小成本分配方案。
- 例子，成本矩阵：

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

4、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

分配问题

- 4个任务全排列分配给1-4号人员执行,复杂度$n!$

分配方案	成本
<1,2,3,4>	9+4+1+4=18
<1,2,4,3>	9+4+8+9=30
<1,3,2,4>	9+3+8+4=24
<1,3,4,2>	9+3+8+6=26
<1,4,2,3>	9+7+8+9=33
<1,4,3,2>	9+7+1+6=23
<2,1,3,4>	2+6+1+4=13
…	…

蛮力法

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

5、深度优先和广度优先查找

- **深度优先查找DFS**
  - 1）从任意顶点开始访问图的顶点，标记为已访问
  - 2）访问当前节点邻接的一个未访问节点，直到遇到终点（所有临边都被访问过）
  - 3）沿着来路后退一条边。若还有未访问的点则转2，否则结束
- **可根据深度优先遍历图，构造出深度优先查找森林**

5、深度优先和广度优先查找

北京理工大学

➢ 深度优先查找DFS

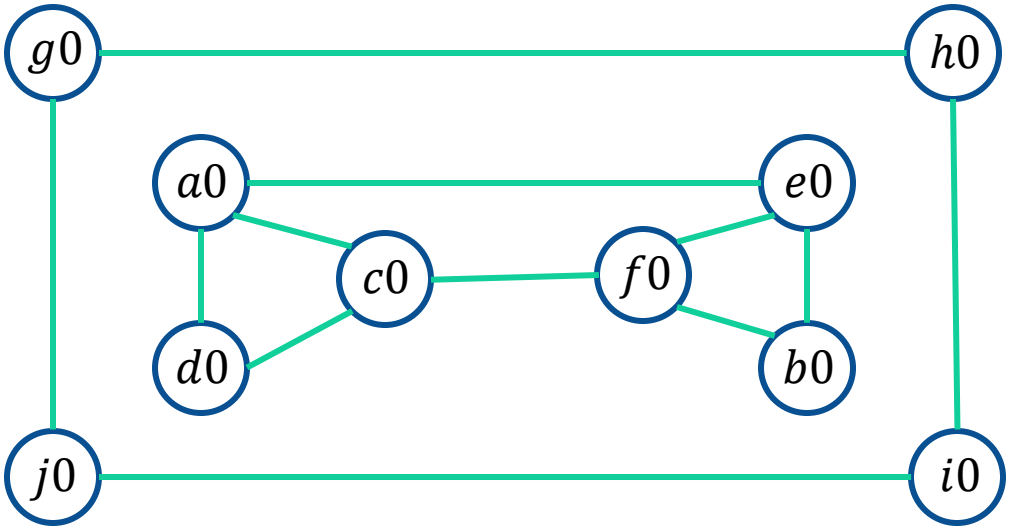
算法 DFS(G)

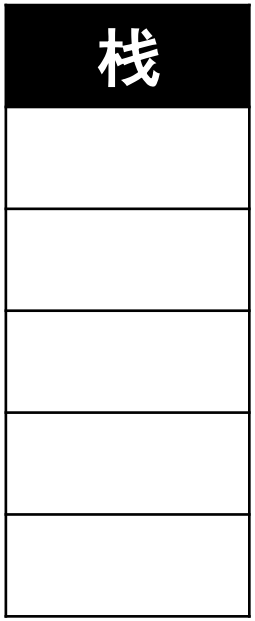
//实现给定图的深度优先查找遍历
//输入：图G=<V,E>
//输出：图G的顶点，按DFS首次访问顺序，用连续整数标记
将V中的每个顶点标记为0，表示“未访问”
$count \leftarrow 0$
$for\ each\ vertex\ v\ in\ V\ do$
    $if\ v\ is\ marked\ with\ 0$
        $dfs(v)$
$dfs(v)$
//递归访问和v相连的未访问顶点，赋值访问顺序。
$count \leftarrow count + 1; mark\ v\ with\ count$
$for\ each\ vertex\ w\ in\ V\ adjacent\ to\ v\ do$
    $if\ w\ is\ marked\ with\ 0$
        $dfs(v)$

5、深度优先和广度优先查找

北京理工大学

- 深度优先查找DFS

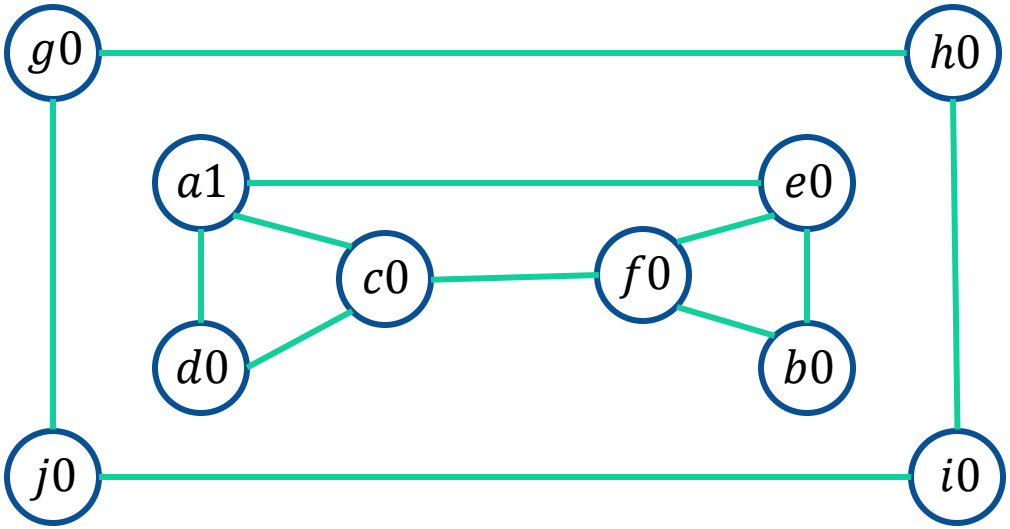




$count = 0$

5、深度优先和广度优先查找

- 深度优先查找DFS



栈
$a(1)$

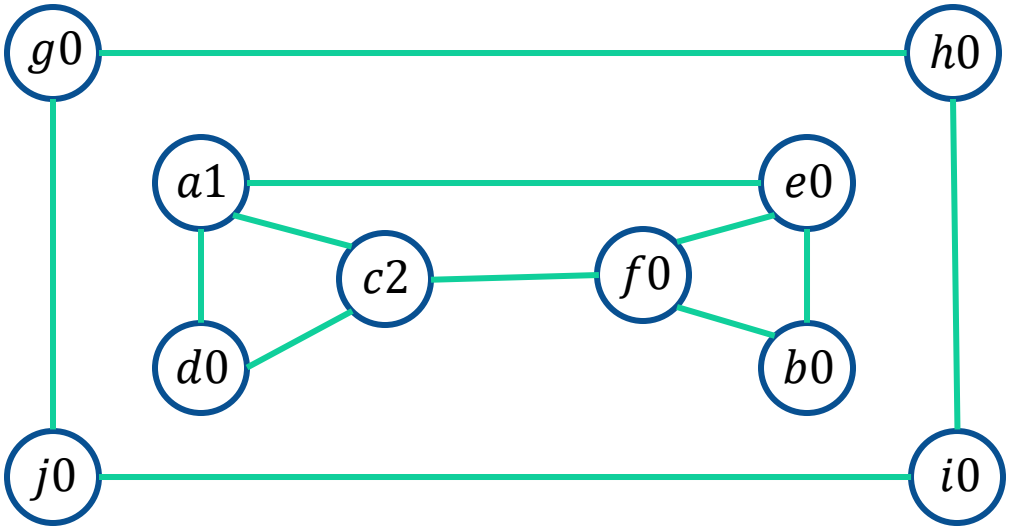
a

$count = 1$

5、深度优先和广度优先查找

北京理工大学

- 深度优先查找DFS



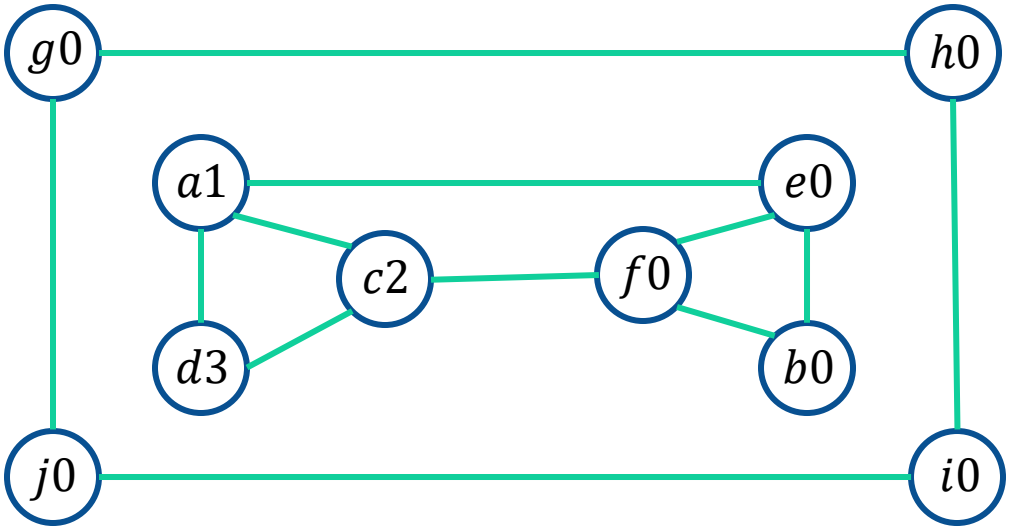
栈
c(2)
a(1)

$count = 2$

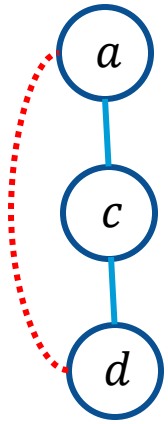
5、深度优先和广度优先查找

北京理工大学

- 深度优先查找DFS



栈
$d(3)$
$c(2)$
$a(1)$

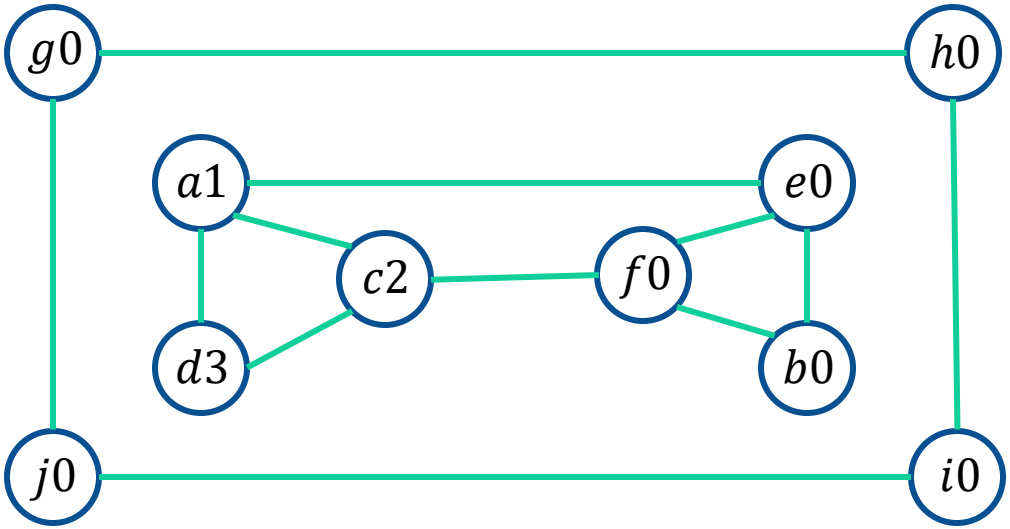


$count = 3$

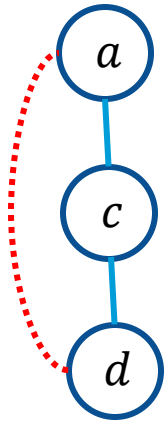
5、深度优先和广度优先查找

北京理工大学

- 深度优先查找DFS



栈
c(2)
a(1)

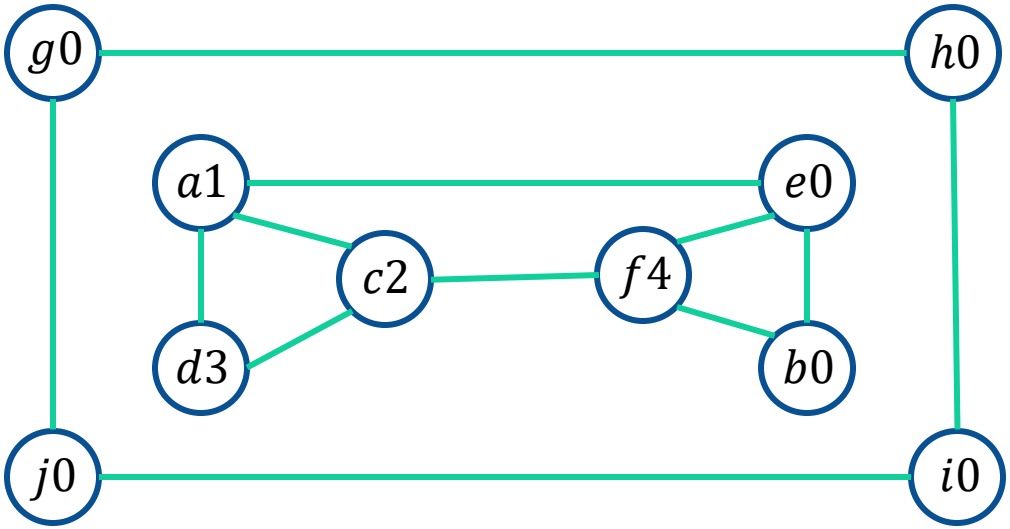


$count = 3$

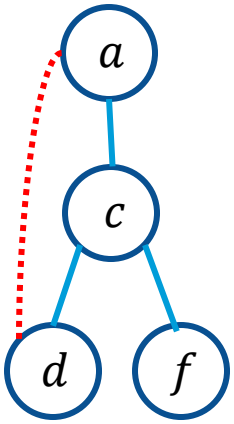
5、深度优先和广度优先查找

北京理工大学

- 深度优先查找DFS



栈
$f(4)$
$c(2)$
$a(1)$

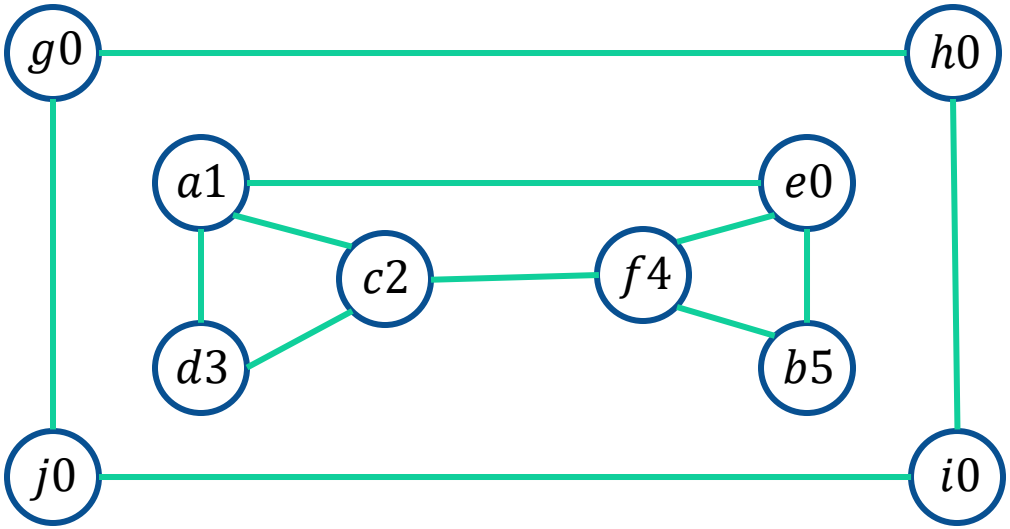


$count = 4$

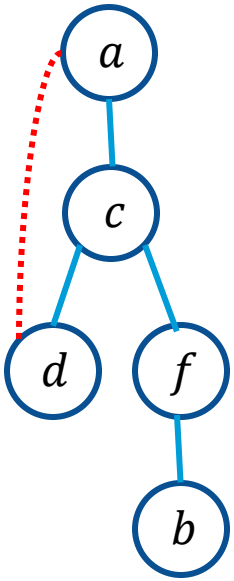
5、深度优先和广度优先查找

北京理工大学

➢ 深度优先查找DFS



栈
$b(5)$
$f(4)$
$c(2)$
$a(1)$

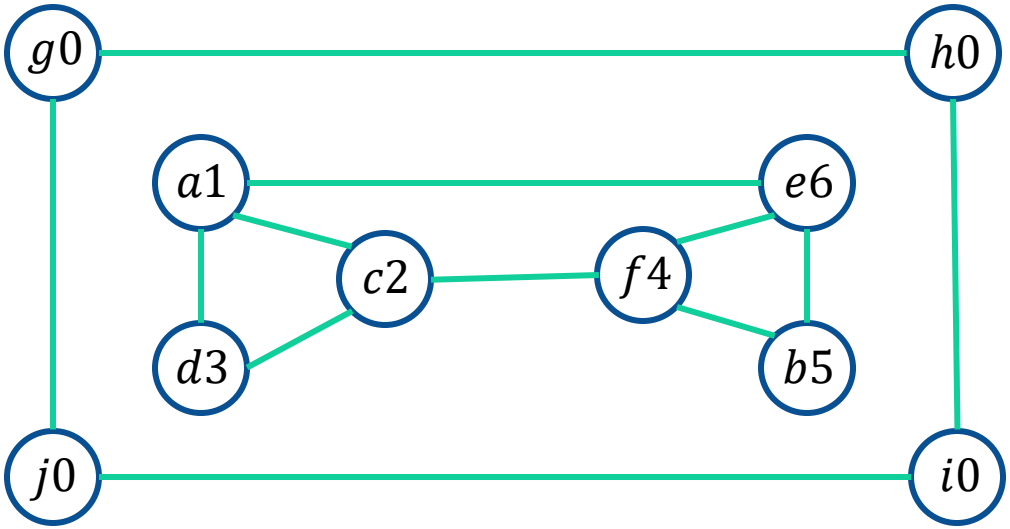


$count = 5$

5、深度优先和广度优先查找

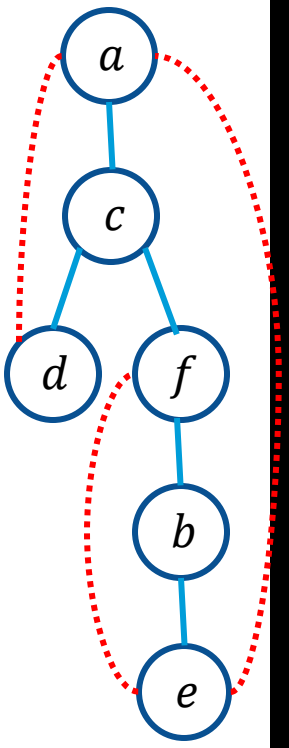
北京理工大学

➢ 深度优先查找DFS



栈
$e(6)$
$b(5)$
$f(4)$
$c(2)$
$a(1)$

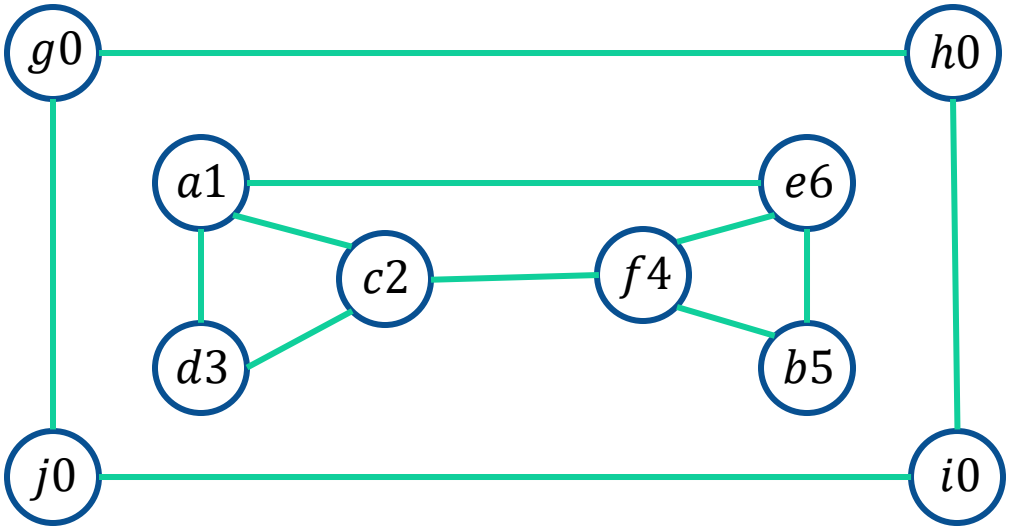
$count = 6$



5、深度优先和广度优先查找

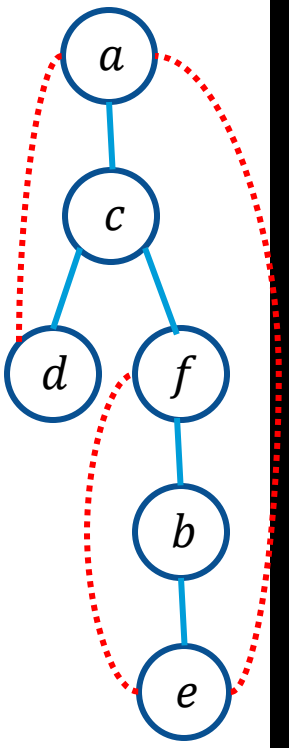
北京理工大学

➢ 深度优先查找DFS



栈
$b(5)$
$f(4)$
$c(2)$
$a(1)$

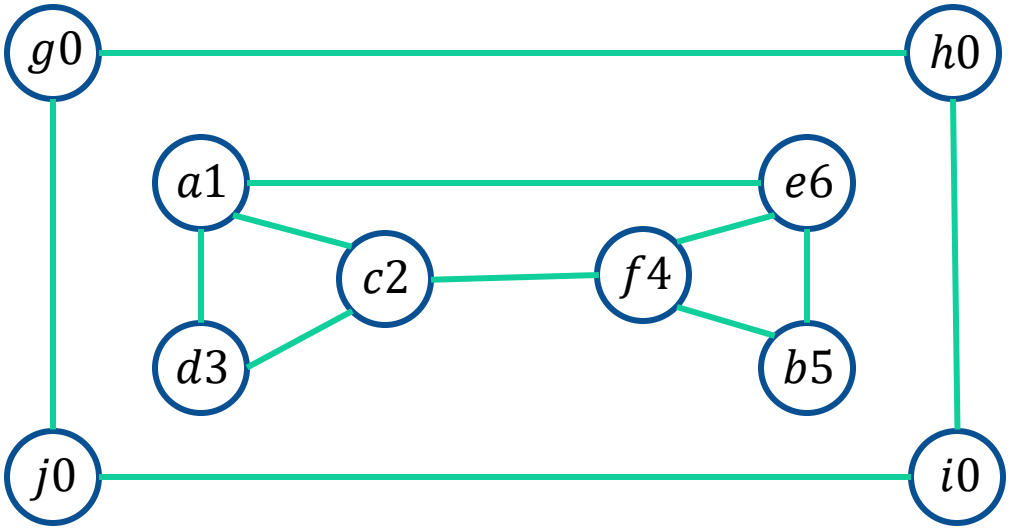
$count = 6$



5、深度优先和广度优先查找

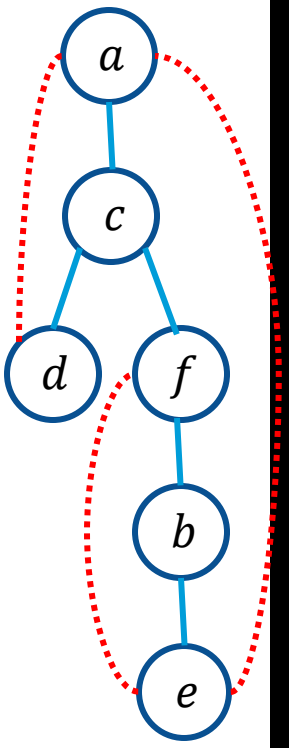
北京理工大学

➢ 深度优先查找DFS



栈
$f(4)$
$c(2)$
$a(1)$

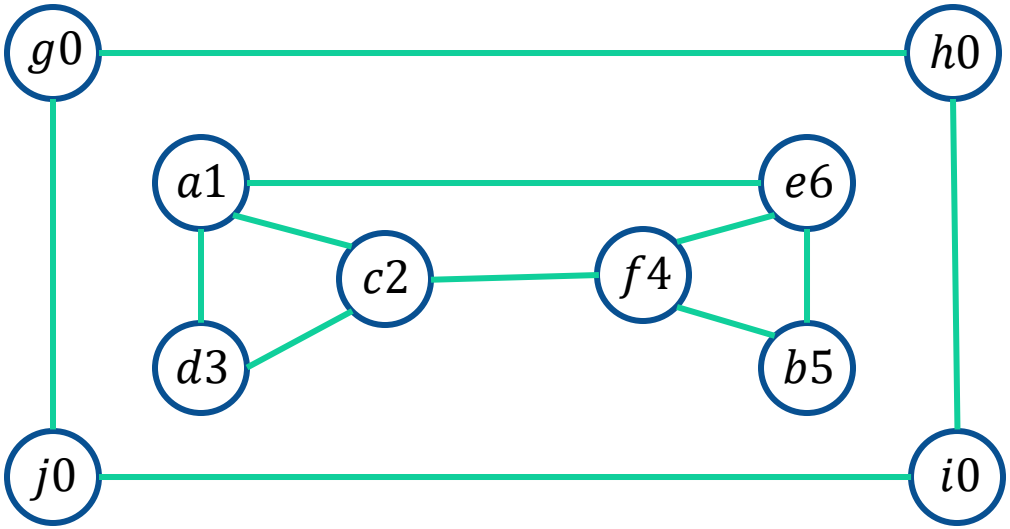
$count = 6$



5、深度优先和广度优先查找

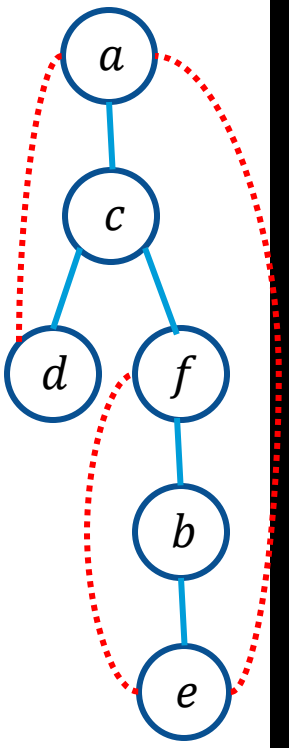
北京理工大学

- 深度优先查找DFS



栈
c(2)
a(1)

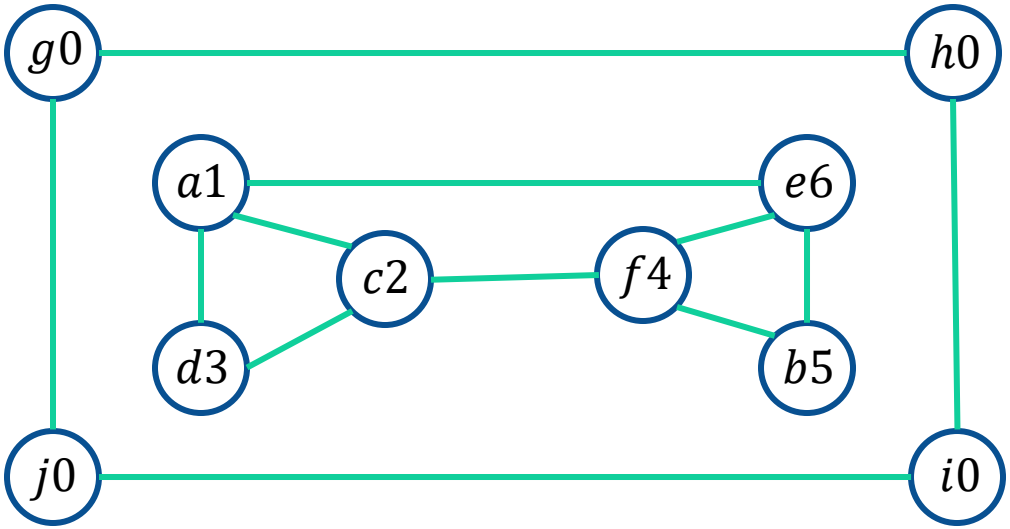
$count = 6$



5、深度优先和广度优先查找

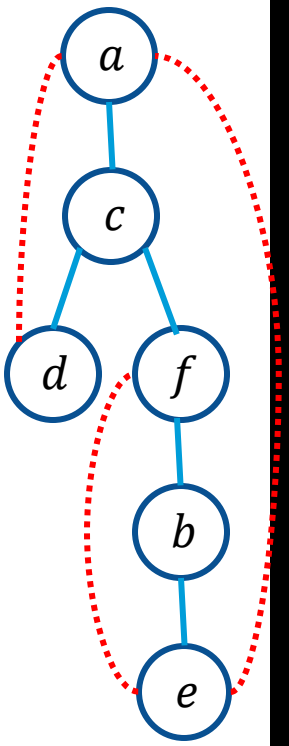
北京理工大学

➢ 深度优先查找DFS



栈
a(1)

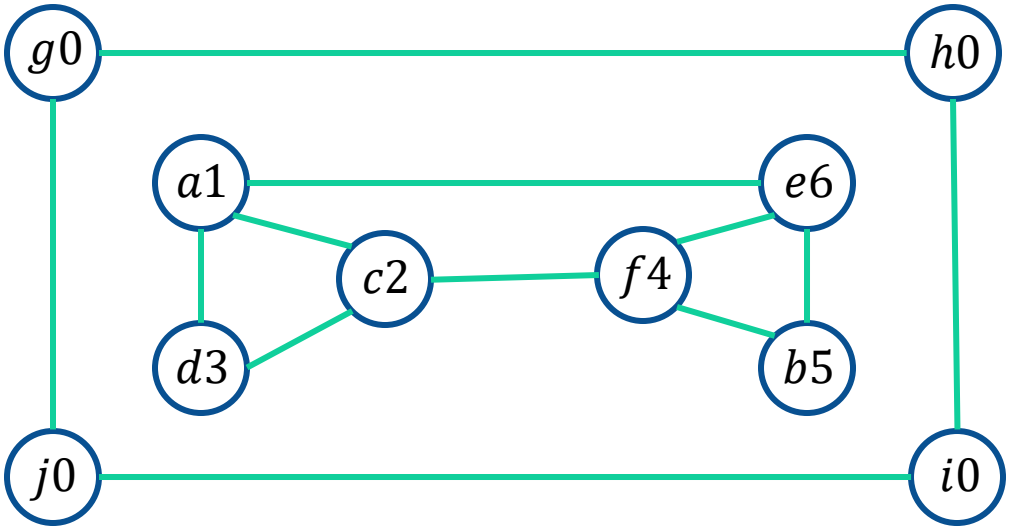
$count = 6$

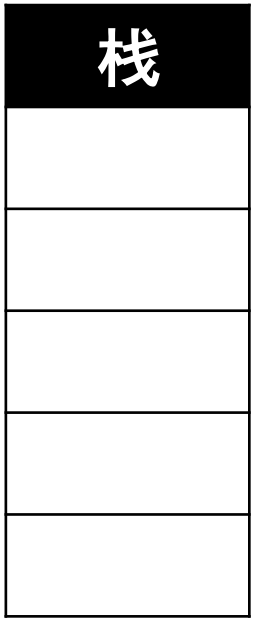


5、深度优先和广度优先查找

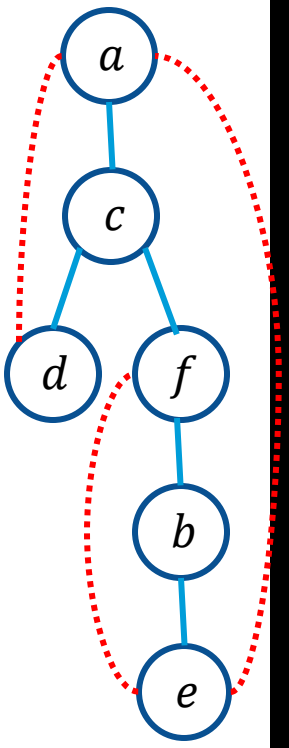
北京理工大学

- 深度优先查找DFS

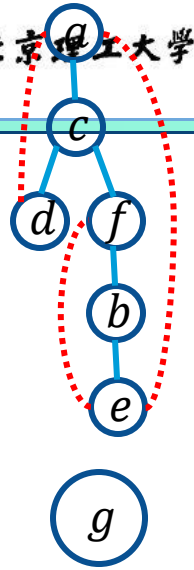




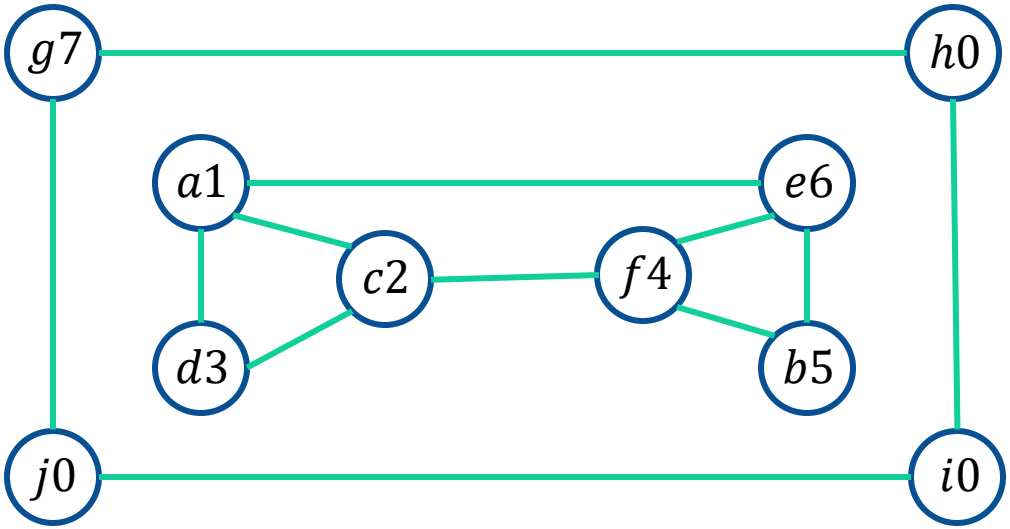
$count = 6$



5、深度优先和广度优先查找



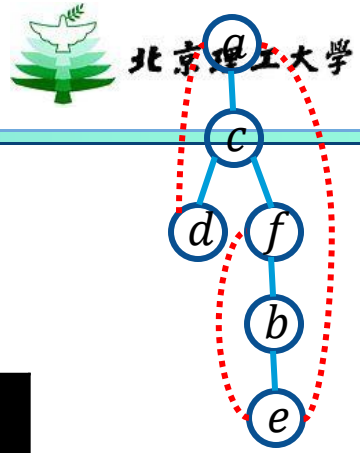
- 深度优先查找DFS



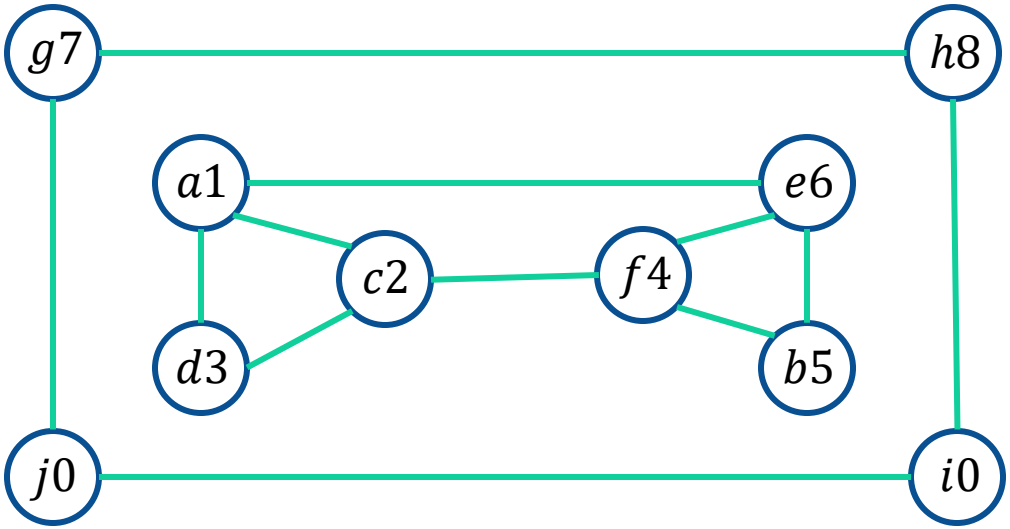
栈
g(7)

$count = 7$

5、深度优先和广度优先查找



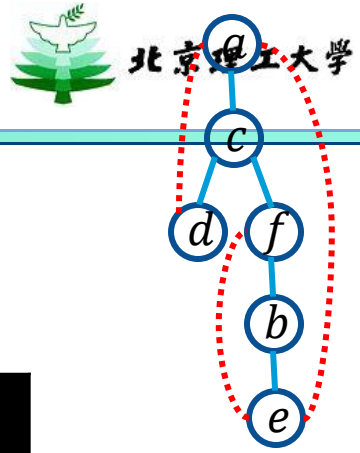
- 深度优先查找DFS



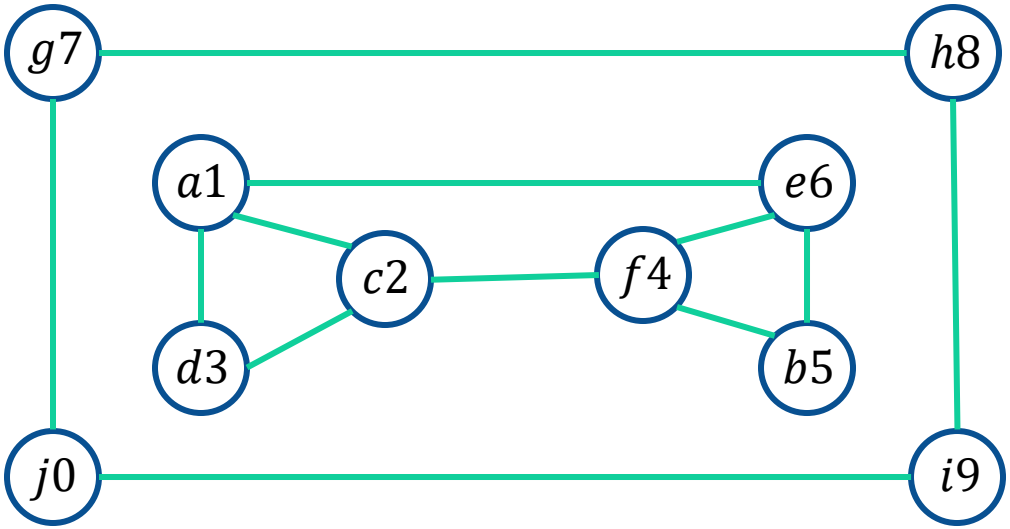
栈
h(8)
g(7)

$count = 8$

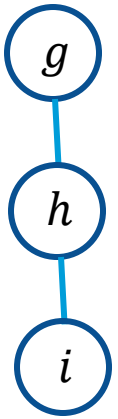
5、深度优先和广度优先查找



- 深度优先查找DFS

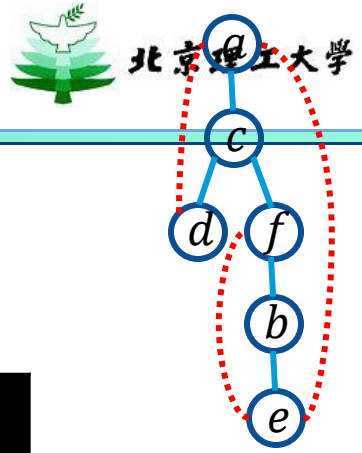


栈
$i(9)$
$h(8)$
$g(7)$

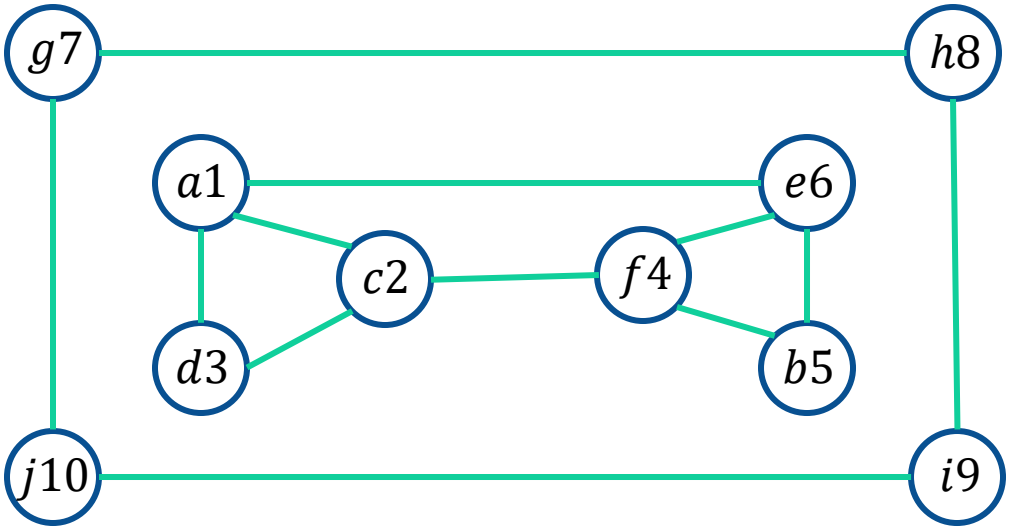


$count = 9$

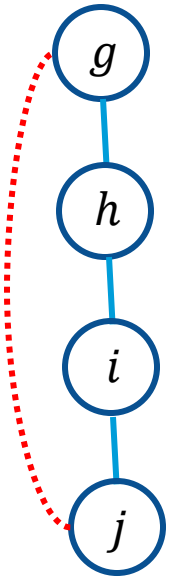
5、深度优先和广度优先查找



- 深度优先查找DFS

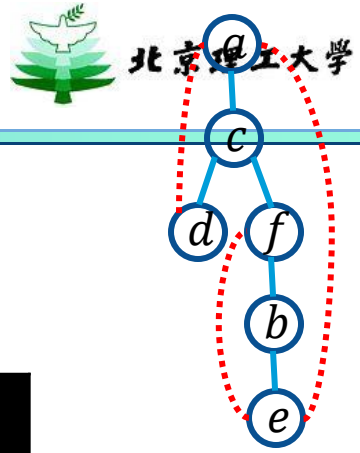


栈
$j(10)$
$i(9)$
$h(8)$
$g(7)$

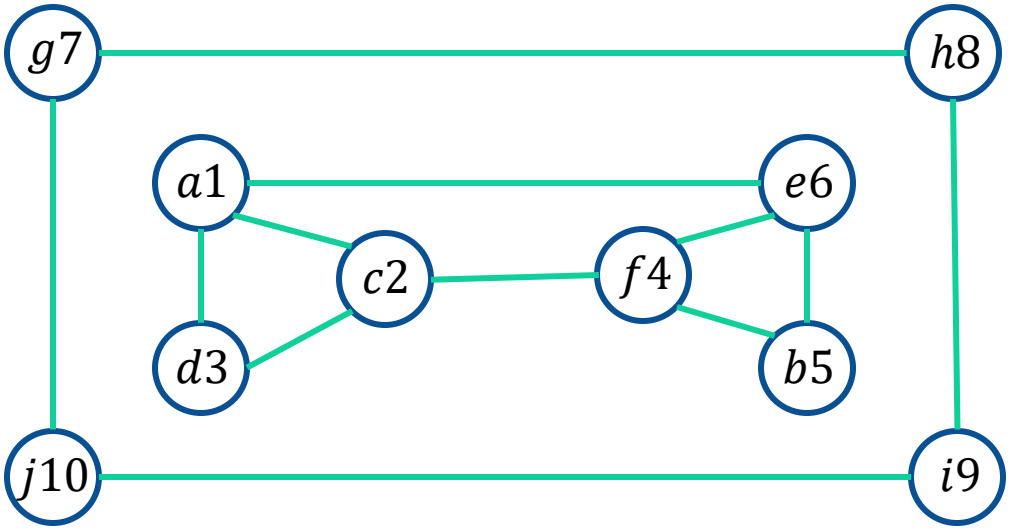


$count = 10$

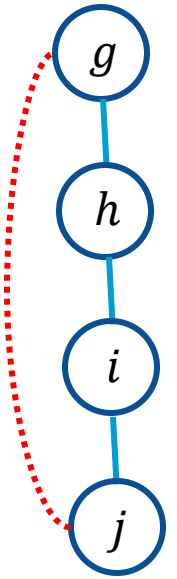
5、深度优先和广度优先查找



- 深度优先查找DFS



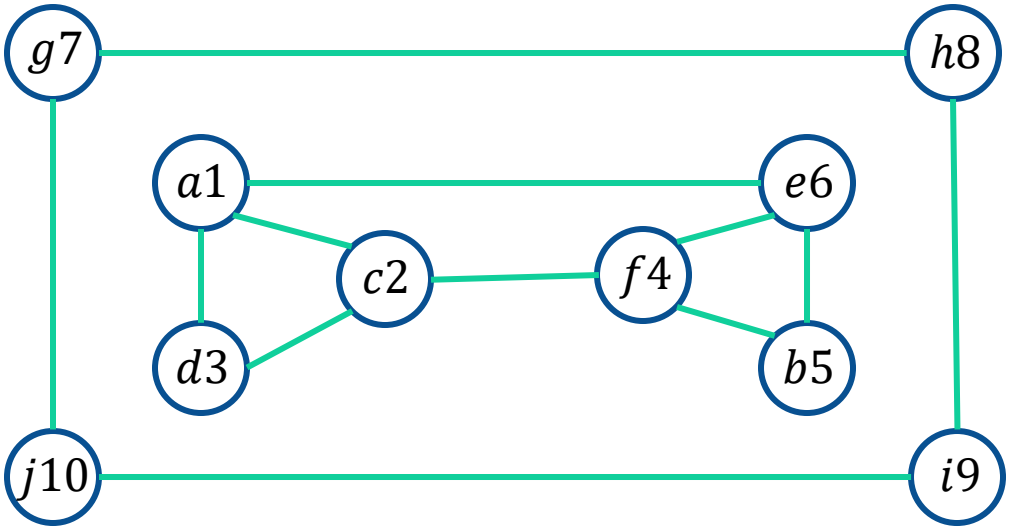
栈
$i(9)$
$h(8)$
$g(7)$



$count = 10$

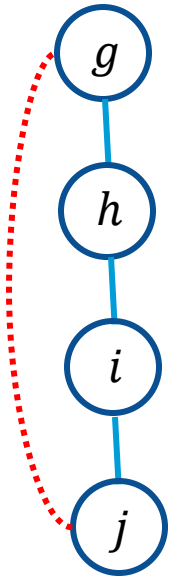
5、深度优先和广度优先查找

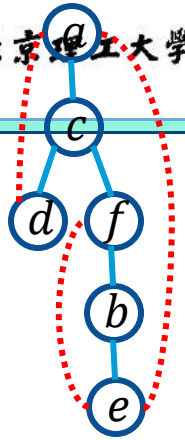
- 深度优先查找DFS



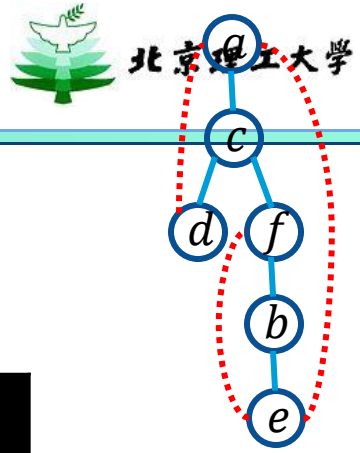
栈
$h(8)$
$g(7)$

$count = 10$

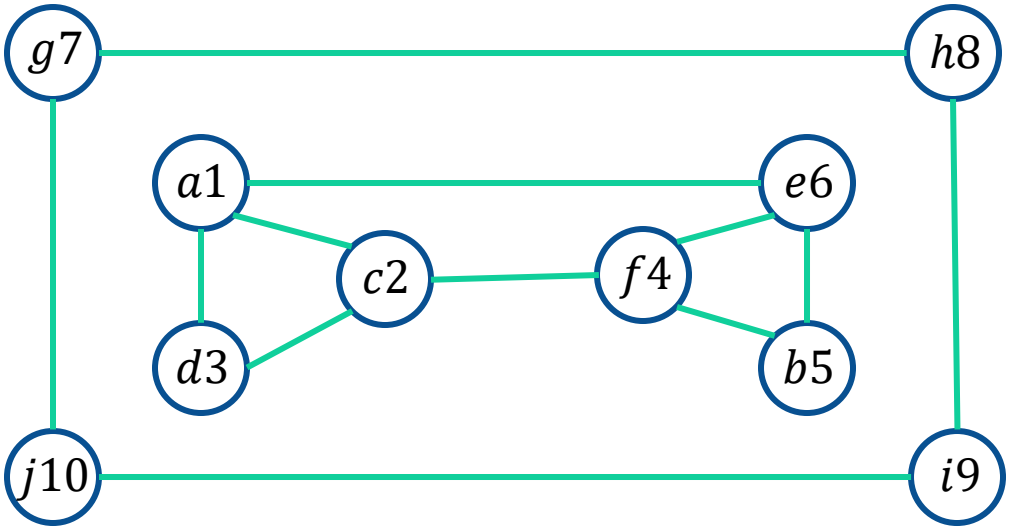




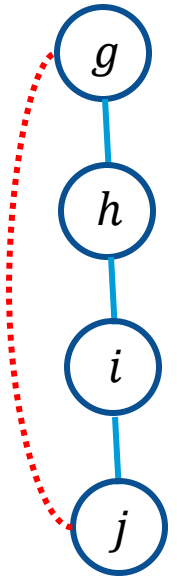
5、深度优先和广度优先查找



- 深度优先查找DFS



栈
g(7)

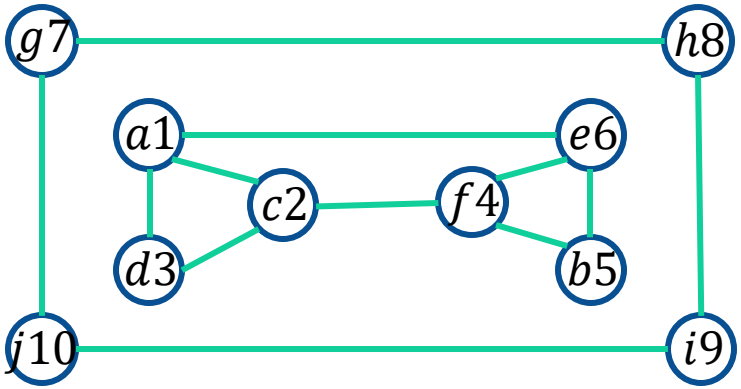


$count = 10$

5、深度优先和广度优先查找

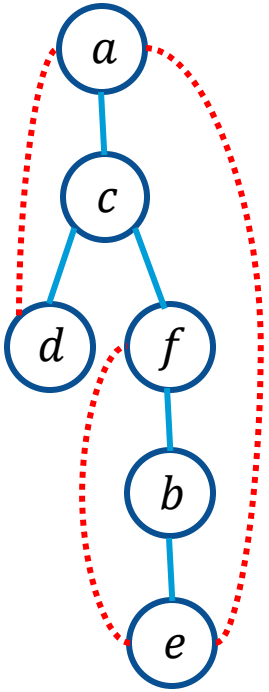
北京理工大学

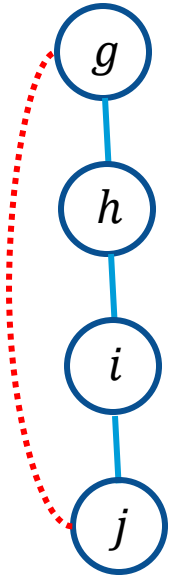
➢ 深度优先查找DFS



栈

$count = 10$





实线 树向边，是连接顶点的边。

虚线 回边，是指向已访问的节点，且非直接前驱的边

5、深度优先和广度优先查找

北京理工大学

➢ 深度优先查找DFS

- 深度优先查找的效率: 消耗的时间和用来表示图的数据结构的规模是成正比的。因此，对于邻接矩阵表示法，该遍历的时间效率属于$\Theta(|V|^2)$; 而对于邻接链表表示法，它属于$\Theta(|V|+|E|)$，其中|V|和|E|分别是图的顶点和边的数量。

5、深度优先和广度优先查找

深度优先查找DFS

- DFS重要的基本应用包括检查图的连通性和无环性。因为DFS在访问了所有和初始顶点有路径相连的顶点之后就会停下来，所以我们可以这样检查一个图的连通性：

从任意一个节点开始DFS遍历，算法停下来时，检查是否所有的顶点都被访问过。

5、深度优先和广度优先查找

北京理工大学

广度优先查找BFS

- 按照同心圆的方式，首先访问所有和初始顶点邻接的顶点，然后是离他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访问过了为止。

- 通常使用队列来跟踪广度优先搜索的操作是比较方便的。

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”

$count \leftarrow 0$

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

$count \leftarrow count+1$; 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

      $count \leftarrow count+1$; mark w with *count*

      add w to the queue

  remove the front vertex from the queue

5、深度优先和广度优先查找

北京理工大学

➢ 广度优先查找BFS

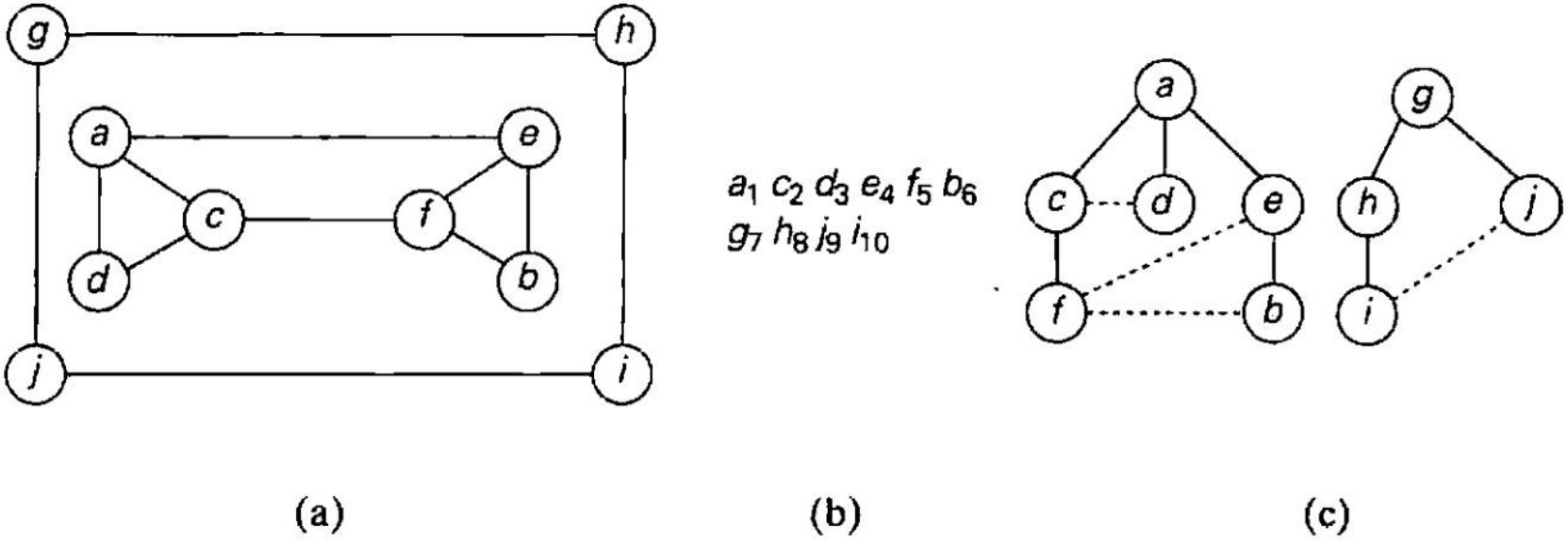


图 3.11　BFS 遍历的例子。(a)图；(b)遍历队列，其中的数字指出某个节点被访问到的顺序，也就是入队(或出队)顺序；(c)BFS 森林(树向边用实线表示，交叉边用虚线表示)

5、深度优先和广度优先查找

北京理工大学

广度优先查找BFS

- 广度优先查找的效率: 和深度优先查找一样，消耗的时间和用来表示图的数据结构的规模是成正比的。因此，对于邻接矩阵表示法，该遍历的时间效率属于$\Theta(|V|^2)$; 而对于邻接链表表示法，它属于$\Theta(|V|+|E|)$，其中|V|和|E|分别是图的顶点和边的数量。

5、深度优先和广度优先查找

广度优先查找BFS

- 广度优先查找的效率:
- 不同的是：广度优先查找只产生顶点的一种排序。因为队列是FIFO(先进先出)的结构，所以顶点入队的次序和他们出队的次序是相同的。
- 也可以通过广度优先查找验证图的连通性和无环性。

5、深度优先和广度优先查找

北京理工大学

➢ 主要性质对比

表 3.1　深度优先查找(DFS)和广度优先查找(BFS)的主要性质

项　目	DFS	BFS
数据结构	栈	队列
顶点顺序的种类	两种顺序	一种顺序
边的类型(无向图)	树向边和回边	树向边和交叉边
应用	连通性、无环性、关节点	连通性、无环性、最少边路径
邻接矩阵的效率	$\Theta(\|V\|^2)$	$\Theta(\|V\|^2)$
邻接链表的效率	$\Theta(\|V\|+\|E\|)$	$\Theta(\|V\|+\|E\|)$

蛮力法总结

北京理工大学

- 蛮力法是一种简单直接解决问题的方法。
- 具有广泛的适用性和简单性，但效率低下。
- 著名的蛮力法算法包括：
  - 基于定义的矩阵乘法
  - 选择排序
  - 顺序查找
  - 简单的字符串匹配
- 穷举查找是解组合问题的蛮力法。
- 旅行商问题，背包问题和分配问题是典型的能够用穷举查找求解的问题。