程序设计方法与实践

——蛮力法

蛮力法

- 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

- 1、选项排序和冒泡排序
- 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

1、选择排序和冒泡排序

- 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1

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

$for\ i \leftarrow 0\ to\ n-2\ do$ **//最小元素最终放置的位置**

$\quad min \leftarrow i$

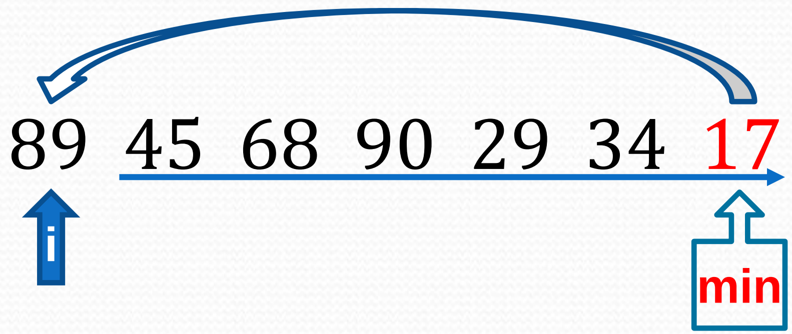
$\quad for\ j \leftarrow i+1\ to\ n-1\ do$

$\quad\quad if\ A[j] < A[min]\quad min \leftarrow j$

$\quad swap\ A[i]\ and\ A[min]$

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

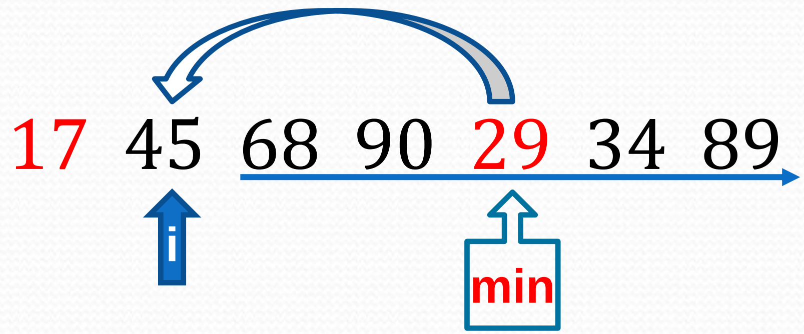
- 选择排序
  - 举例：89,45,68,90,29,34,17

17 45 68 90 29 34 89

i

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

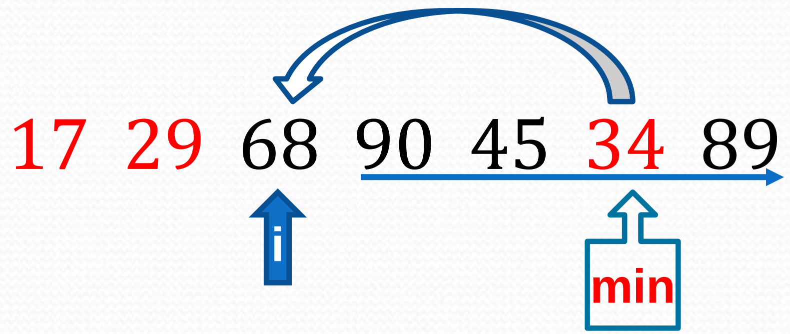
- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 68 90 45 34 89

i

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17



1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 90 45 68 89

i

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 90 45 68 89

i min

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 45 90 68 89

i

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 45 90 68 89

i

min

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 45 68 90 89

i

1、选择排序和冒泡排序

- 选择排序
  - 举例：89,45,68,90,29,34,17

17 29 34 45 68 90 89

i min

1、选择排序和冒泡排序

- 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

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

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

- 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

- 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
$for\ i \leftarrow 0\ to\ n-2\ do$ //每轮沉底最大元素，要执行n-1轮
  $for\ j \leftarrow 0\ to\ n-2-i\ do$ //从头比较，最后i+1个不用比
    $if\ A[j+1] < A[j]$
      $swap\ A[j]\ and\ A[j+1]$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

89 45 68 90 29 34 17

$\boldsymbol{i = 0}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 89 68 90 29 34 17

$\boldsymbol{i=0}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

$\boldsymbol{i=0}$

$$\begin{array}{ccccccc} 45 & 89 & 68 & 90 & 29 & 34 & 17 \\ & j & & & & & \end{array}$$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 90 29 34 17

$\boxed{i=0}$ $[\quad j \quad]$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

$\boldsymbol{i = 0}$

45 68 89 90 29 34 17

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 90 34 17

$\boldsymbol{i=0}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

$\boldsymbol{i = 0}$

45 68 89 29 90 34 17

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 34 90 17

j

$\boldsymbol{i = 0}$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 34 90 17

$\boldsymbol{i = 0}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 34 17 90

$\boldsymbol{i = 0}$

$\left[\quad \overset{\Uparrow}{j} \quad \right]$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 34 17 90

$\boldsymbol{i = 1}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 89 29 34 17 90

j

$\boldsymbol{i=1}$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 89 34 17 90

$\boldsymbol{i = 1}$ [j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 89 34 17 90

$\boldsymbol{i=1}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 34 89 17 90

$\boldsymbol{i=1}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 34 89 17 90

$\boldsymbol{i=1}$ [j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 34 17 89 90

$\boldsymbol{i = 1}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 68 29 34 17 89 90

$\boldsymbol{i=2}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

$\boldsymbol{i=2}$

45 68 29 34 17 89 90

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 29 68 34 17 89 90

$$[\quad j \quad]$$

$\boldsymbol{i=2}$

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 29 68 34 17 89 90

$\boxed{i = 2}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 29 34 68 17 89 90

$i = 2$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

$\boldsymbol{i = 2}$

45 29 34 68 17 89 90

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 29 34 17 68 89 90

$\boldsymbol{i = 2}$ [j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

45 29 34 17 68 89 90

$\boldsymbol{i=3}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 45 34 17 68 89 90

$\boldsymbol{i = 3}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 45 34 17 68 89 90

$\boldsymbol{i = 3}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 34 45 17 68 89 90

$\boldsymbol{i = 3}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 34 45 17 68 89 90

$\boldsymbol{i = 3}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 34 17 45 68 89 90

$\boldsymbol{i = 3}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 34 17 45 68 89 90

$\boldsymbol{i = 4}$

j

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 34 17 45 68 89 90

$\boldsymbol{i = 4}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 17 34 45 68 89 90

$\boldsymbol{i=4}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

29 17 34 45 68 89 90

$\boldsymbol{i = 5}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

17 29 34 45 68 89 90

$\boldsymbol{i = 5}$

[j]

1、选择排序和冒泡排序

- 冒泡排序
  - 举例：89,45,68,90,29,34,17

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

$$=\sum_{i=0}^{n-2}(n-1-i)=1+2+3\ldots+(n-1)$$

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

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

蛮力法

- 1、选项排序和冒泡排序
- 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

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

$A[n] \leftarrow K$
$i \leftarrow 0$
$while\ A[i] \neq K\ do$
  $i \leftarrow i+1$
$if\ i < n \quad return\ i$
$else\ return\ -1$

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≤n)
  - 从文本中寻找匹配模式的子串。

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

$$p_0 \ldots p_j \quad \ldots p_{m-1} \quad \text{模式P}$$

2、顺序查找和蛮力字符串匹配

- 蛮力字符串匹配

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

//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

$for\ i \leftarrow 0\ to\ n-m\ do$ //最后的m-1个串不用比，长度不够

$\quad j \leftarrow 0$

$\quad while\ j < m\ and\ P[j] = T[i+j]\ do$

$\quad\quad j \leftarrow j+1$

$\quad\quad if\ j = m\quad return\ i$

$return\ -1$

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

i

$NOBODY\_NOTICED\_HIM$

$NO\color{red}{T}$

比较3次

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

i

$$NOBODY\_NOTICED\_HIM$$

$$\color{red}{N}OT$$

比较1次

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

i

$NOBODY\_NOTICED\_HIM$

NOT

比较1次

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

i

$$NOBODY\_NOTICED\_HIM$$

$$NOT$$

比较1次

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

i

$$NOBODY\_NOTICED\_HIM$$

$$NOT$$

比较1次

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

i

$NOBODY\_NOTICED\_HIM$

NOT

比较1次

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

i

$$NOBODY\_NOTICED\_HIM$$

$$NOT$$

比较1次

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

- 蛮力字符串匹配
  - 举例：NOBODY_NOTICED_HIM，匹配NOT

i

$$NOBODY\_NOTICED\_HIM$$

$$NOT$$

比较3次

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

- **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：$\Theta(n*m)$

文本**000 000 000 000** 模式**001** 比较几次？ **(12-3+1)*3**

如果模式是**100**，比较几次？ **(12-3+1)*1**

  - 平均效率：$\Theta(n+m)$

假设子串匹配效率最佳(子串只比较一次即发现不同跳出)，若第i个位置匹配成功，比较了$(i+m)$次，平均比较次数：

- $\sum_{i=0}^{n-m} p(\mathrm{i}+\mathrm{m}) = \frac{1}{n-m+1}\sum_{i=0}^{n-m}(i+m) = \frac{n+m}{2} = \theta(n+m)$

蛮力法

- 1、选项排序和冒泡排序
- 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

3、最近对和凸包问题的蛮力算法

- 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
  - 可对应聚类分析等问题

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

$d \leftarrow \infty$

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

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

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

$return\ d$

3、最近对和凸包问题的蛮力算法

- 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
    基本操作：

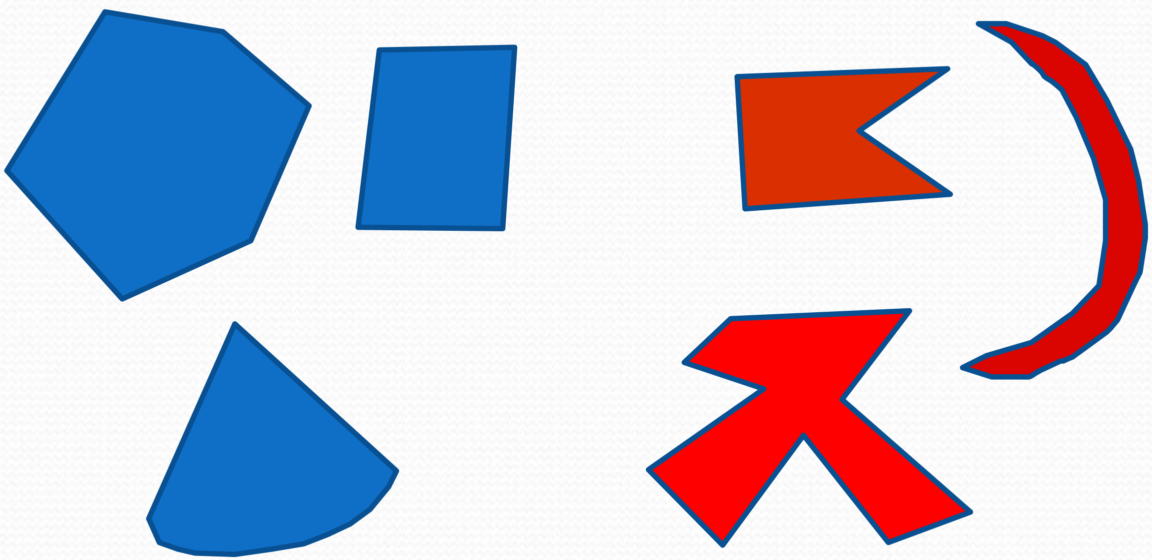
$$distance(P_i, P_j) = sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$$

可忽略开方操作，则基本操作为2次平方操作。

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

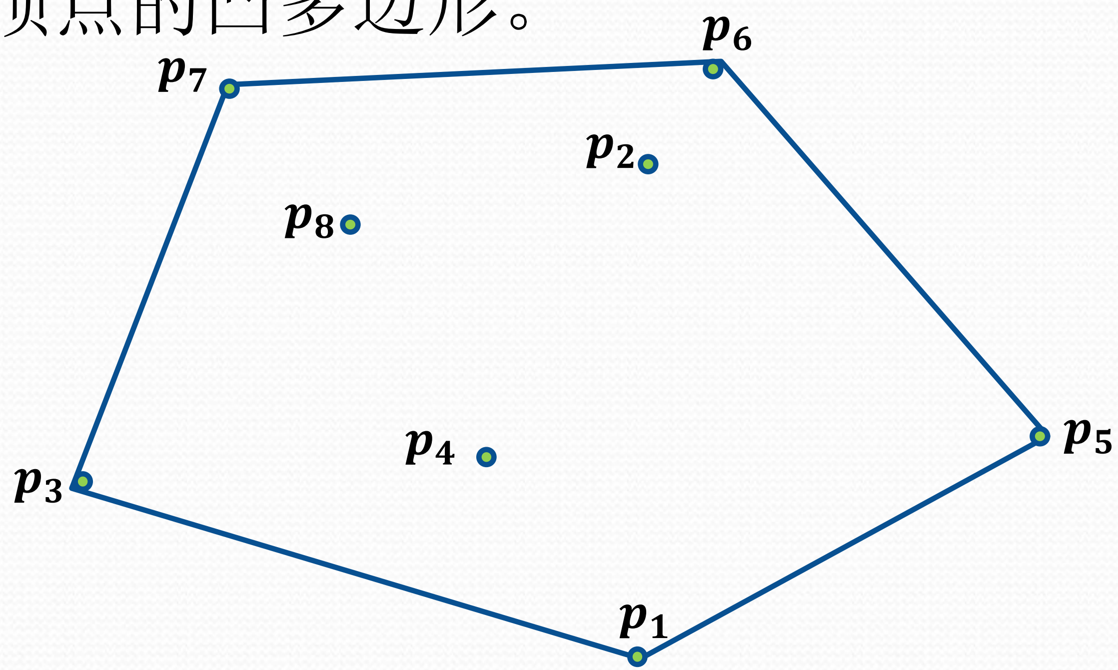
3、最近对和凸包问题的蛮力算法

- 凸包问题
  - 凸包可以为给定集合提供近似。
  - 凸包可以代替点集进行快速碰撞检测。
- 凸集合：平面上点的集合，若其中任意两点为端点的线段都属于该集合，则此集合是凸的



3、最近对和凸包问题的蛮力算法

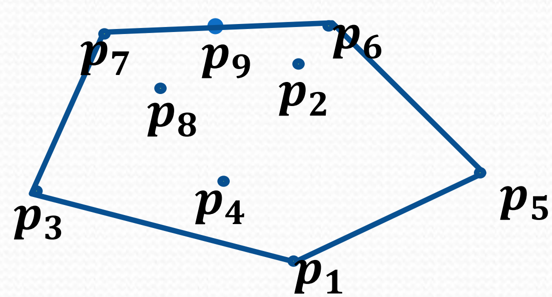
- 凸包问题
- 凸包：点集S的凸包，是包含S的最小凸集合。
  - 任意包含n>2个点（不共线）的集合S的凸包是以S中某些点为顶点的凸多边形。



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、最近对和凸包问题的蛮力算法

- 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;三个共线的点的极点是最远的两个点；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、最近对和凸包问题的蛮力算法

- step1：对于一个n个点集合中的两个点P_i,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，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、最近对和凸包问题的蛮力算法

- 凸包问题

  - 1、取集合n中的两个点i,j点。
  - 2、构造方程：$\boldsymbol{ax+by=c}$，其中

  $\boldsymbol{a} = \mathrm{y}_2 - \boldsymbol{y_1}, b = \boldsymbol{x_1} - \boldsymbol{x_2}, \mathrm{c} = \boldsymbol{x_1 y_2 - y_1 x_2}$

  - 3、若n中所有其他点带入2式左均≥c或均≤c，

  说明其他点都在i,j点一侧。i,j边为凸包一个边。

  $$\boxed{\boldsymbol{C(n) \in \Theta(n^3)}}$$

蛮力法

- 1、选项排序和冒泡排序
- 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

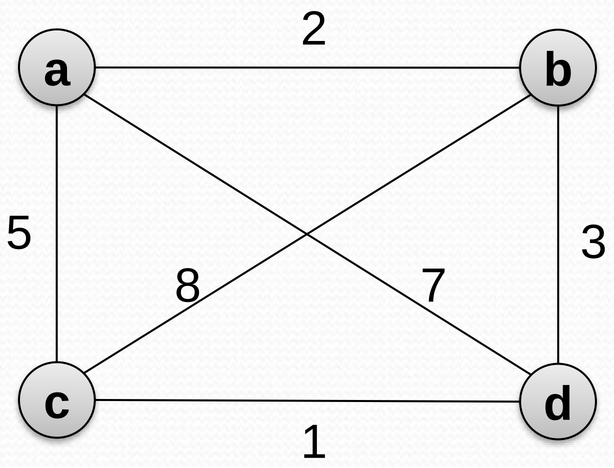
- 要求生成问题域中每个元素，选出满足问题约束的元素，然后找出一个期望元素（最优化元素）
  - 旅行商问题
  - 背包问题
  - 分配问题

4、穷举查找

- 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 $V_{i_0}, V_{i_1}, \ldots, V_{i_{n-1}}, V_{i_0}$
  - 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。

4、穷举查找

- 旅行商问题



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

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

4、穷举查找

- 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

3.6E12	**8.64E13**	**3.15E16**	**3.15E18**

15!	**16!**	**17!**	**18!**	**19!**	**20!**
1.3E12	**2.1E13**	**3.6E14**	**6.4E15**	**1.2E17**	**2.4E18**

4、穷举查找

- 背包问题
  - 给定重量为$w_1, w_2, \dots, w_n$，价值为$v_1, v_2, \dots, v_n$的物品和一个承重为W的背包，求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

- 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

- 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

- 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困难问题，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

- 分配问题
  - n个任务分配给n个人执行，每个任务一个人。对$i,j \in 1,2,...,n$来说，j任务分配给i人的成本为$C[i,j]$。问最小成本的分配方案。
  - 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、选项排序和冒泡排序
- 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、深度优先查找和广度优先查找

- 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，深度优先使用堆栈数据结构

5、深度优先查找和广度优先查找

- 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”

$count \leftarrow 0$

$for\ each\ vertex\ v\ in\ V\ do$

$\quad if\ v\ is\ marked\ with\ 0$

$\quad\quad dfs(v)$

算法$dfs(v)$

//递归访问和v相连的未访问顶点，赋值访问顺序。

$count \leftarrow count + 1; mark\ v\ with\ count$

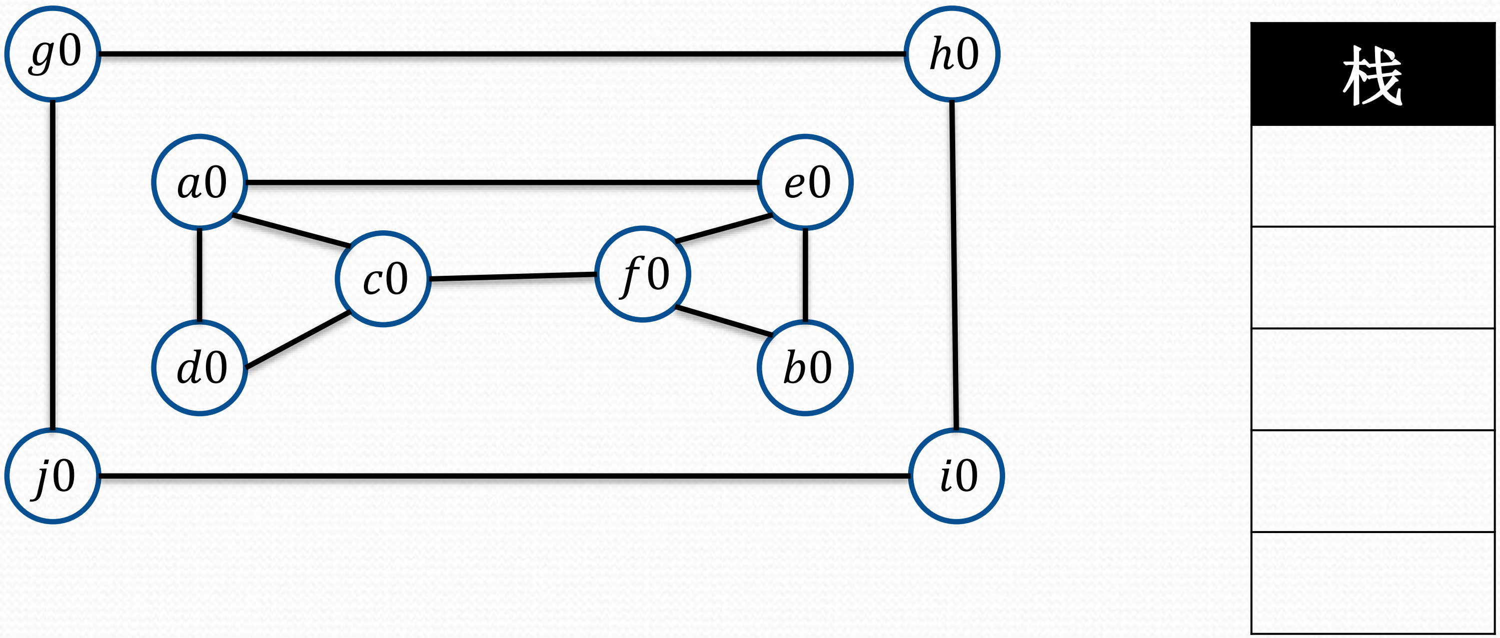
$for\ each\ vertex\ w\ in\ V\ adjacent\ to\ v\ do$

$\quad if\ w\ is\ marked\ with\ 0$

$\quad\quad dfs(\mathrm{w})$

5、深度优先查找和广度优先查找

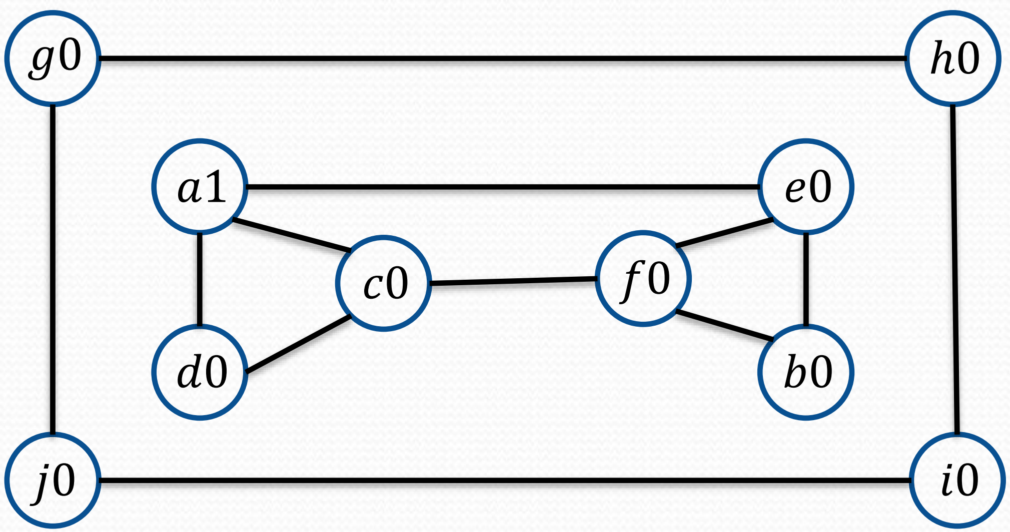
- 深度优先查找DFS

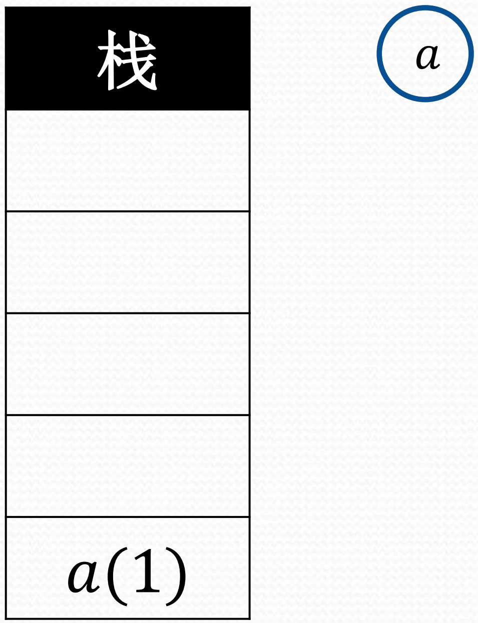


$$count = 0$$

5、深度优先查找和广度优先查找

- 深度优先查找DFS

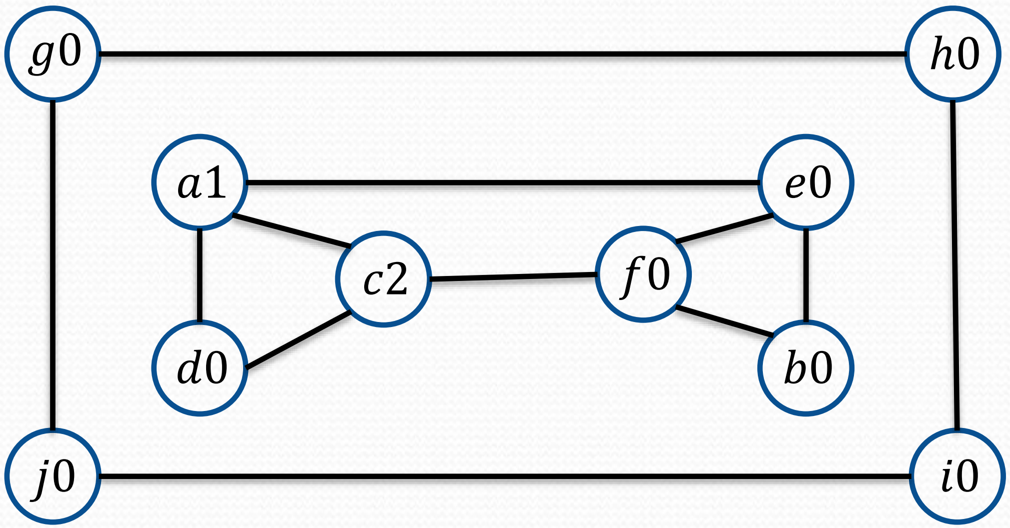




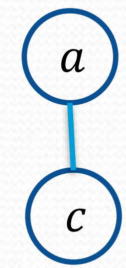
$count = 1$

5、深度优先查找和广度优先查找

- 深度优先查找DFS



栈
$c(2)$
$a(1)$

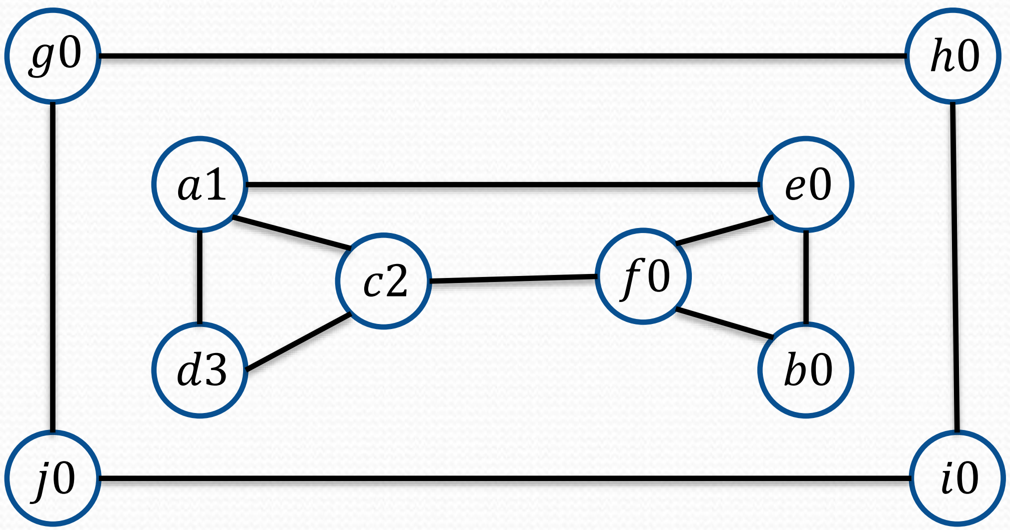


$count = 2$

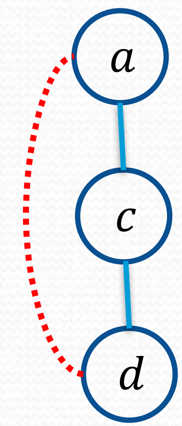
实线 树向边，是连接顶点的边。

5、深度优先查找和广度优先查找

- 深度优先查找DFS



栈
$d(3)$
$c(2)$
$a(1)$

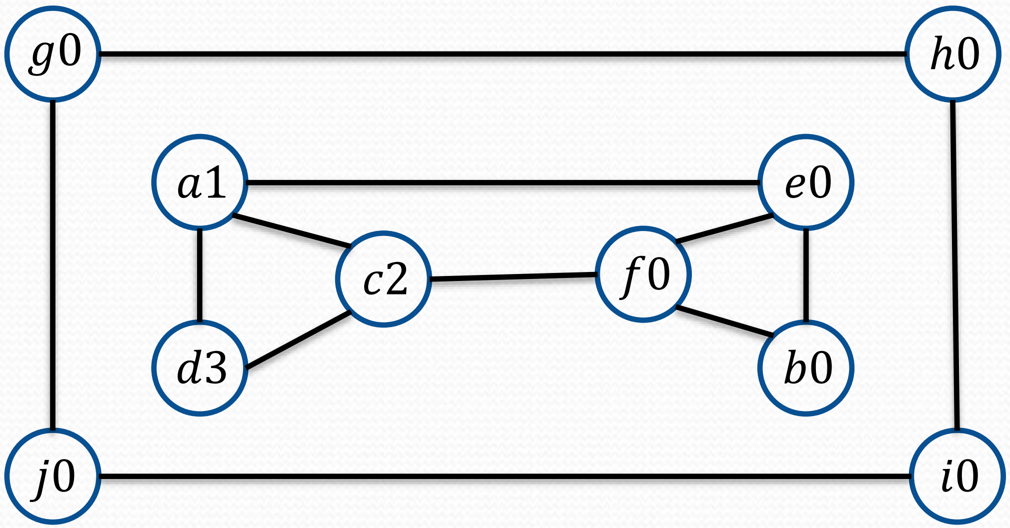


虚线 回边，是指向已访问的节点，且非直接前驱的边

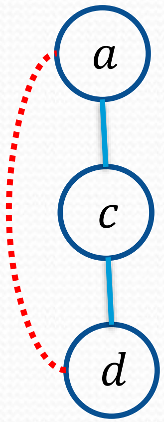
$count = 3$

5、深度优先查找和广度优先查找

- 深度优先查找DFS



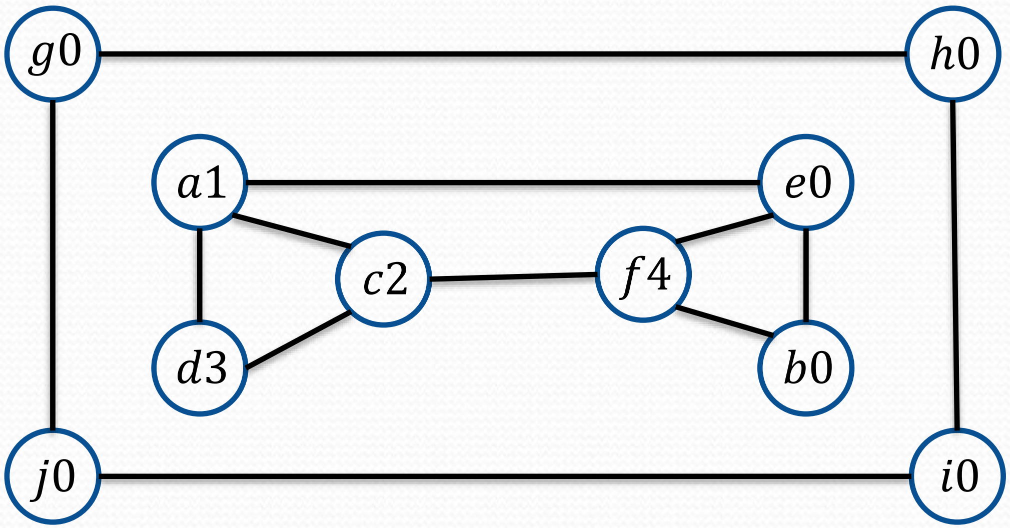
栈
$c(2)$
$a(1)$



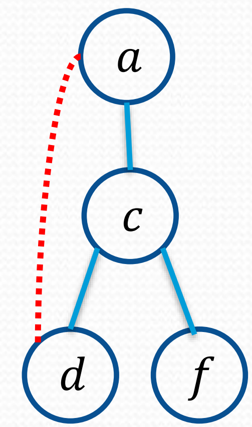
$$count = 3$$

5、深度优先查找和广度优先查找

- 深度优先查找DFS



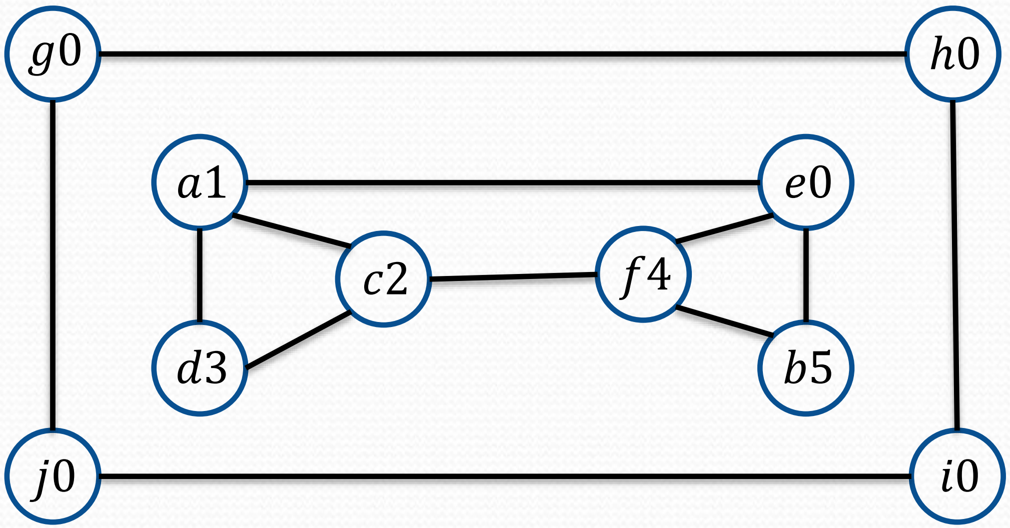
栈
$f(4)$
$c(2)$
$a(1)$



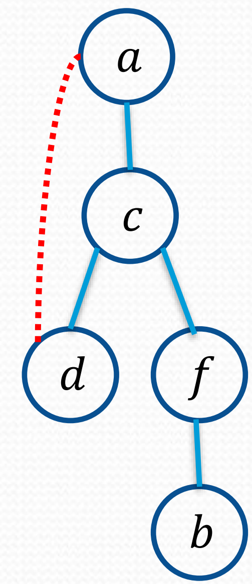
$count = 4$

5、深度优先查找和广度优先查找

- 深度优先查找DFS



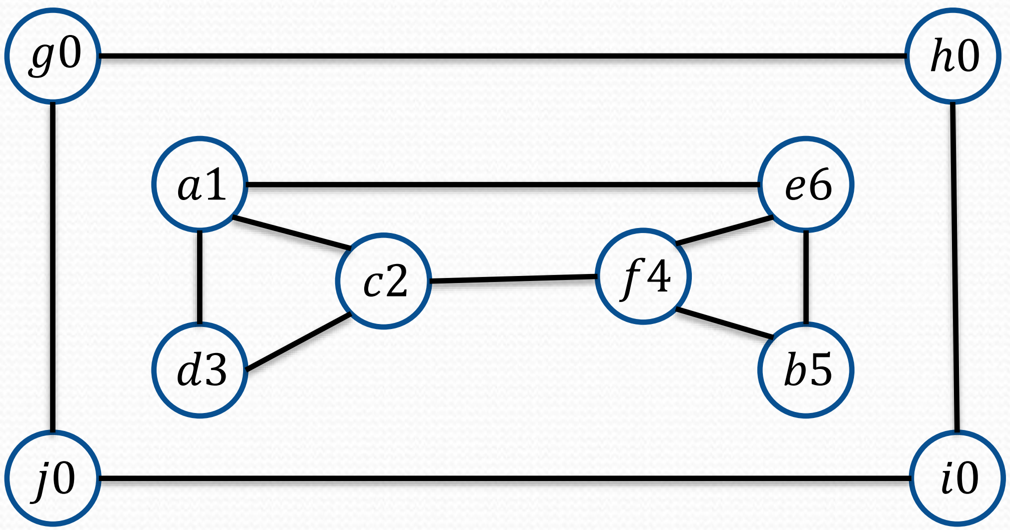
栈
$b(5)$
$f(4)$
$c(2)$
$a(1)$



$count = 5$

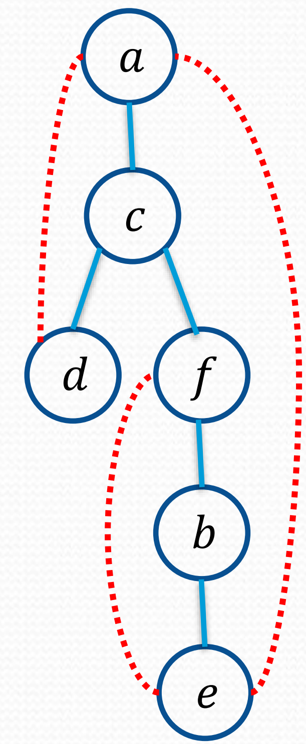
5、深度优先查找和广度优先查找

- 深度优先查找DFS



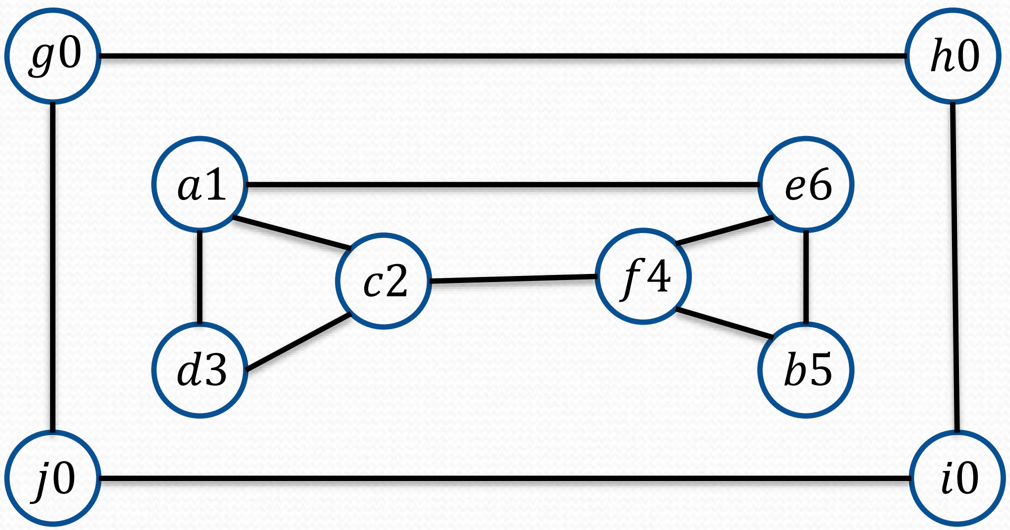
栈
$e(6)$
$b(5)$
$f(4)$
$c(2)$
$a(1)$

$count = 6$



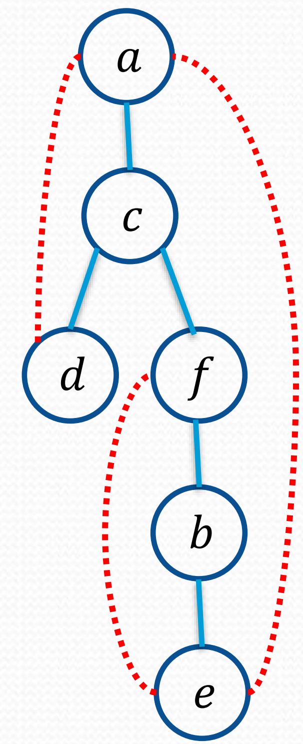
5、深度优先查找和广度优先查找

- 深度优先查找DFS



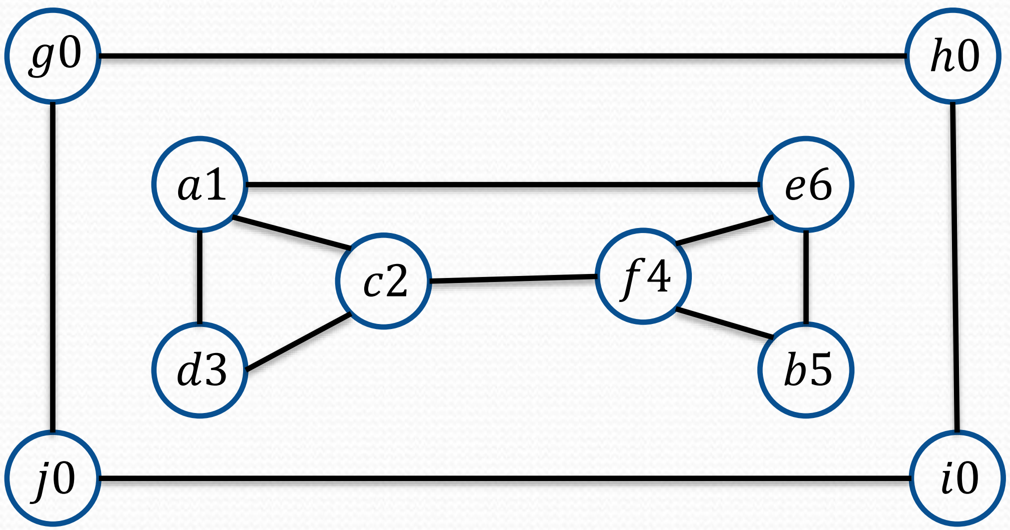
栈
$b(5)$
$f(4)$
$c(2)$
$a(1)$

$count = 6$



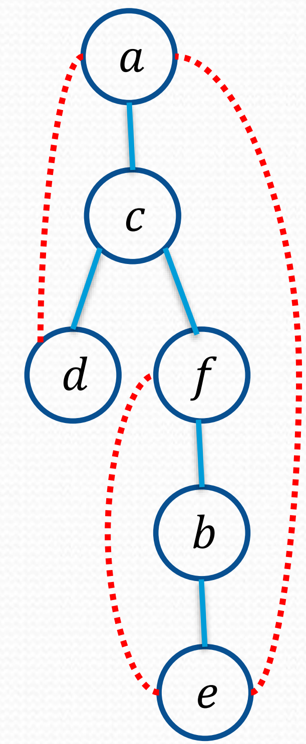
5、深度优先查找和广度优先查找

- 深度优先查找DFS



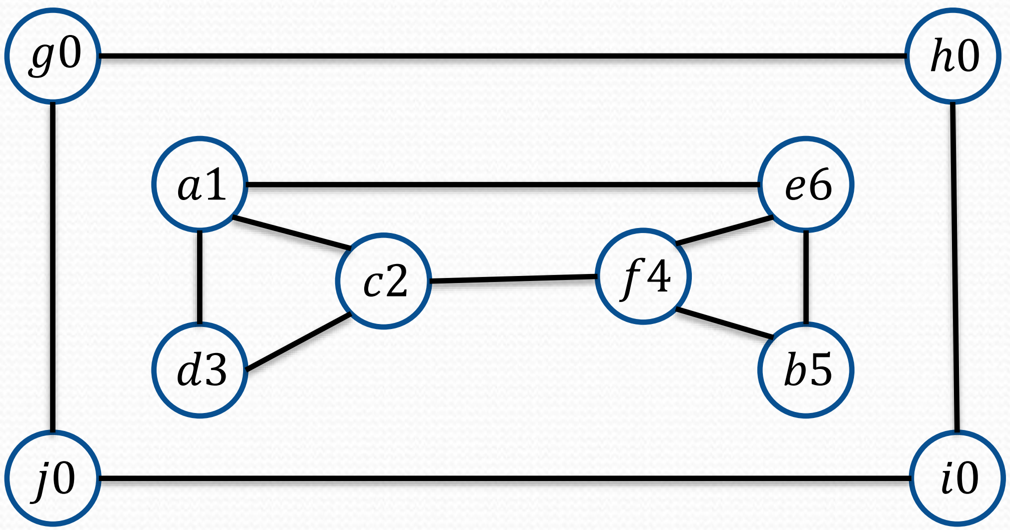
栈
$f(4)$
$c(2)$
$a(1)$

$count = 6$



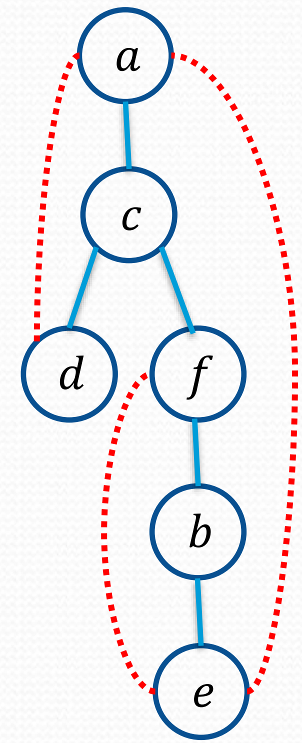
5、深度优先查找和广度优先查找

- 深度优先查找DFS



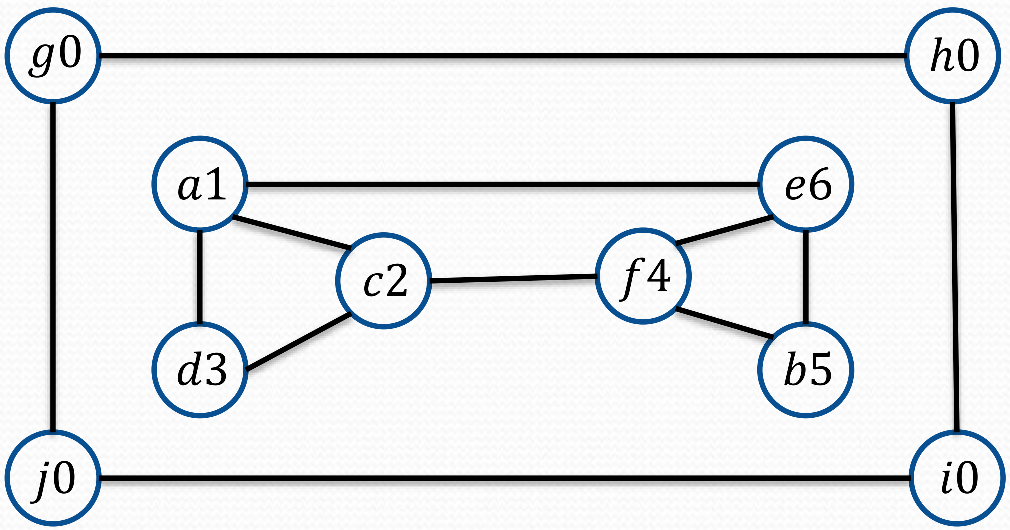
栈
$c(2)$
$a(1)$

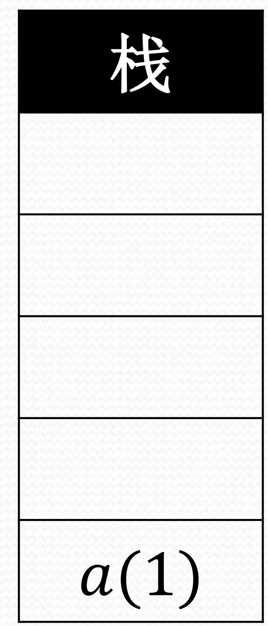
$count = 6$



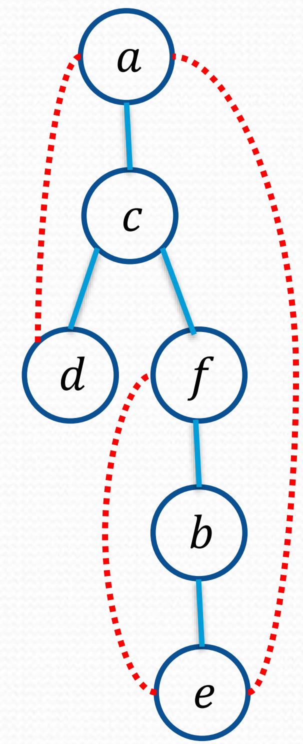
5、深度优先查找和广度优先查找

- 深度优先查找DFS



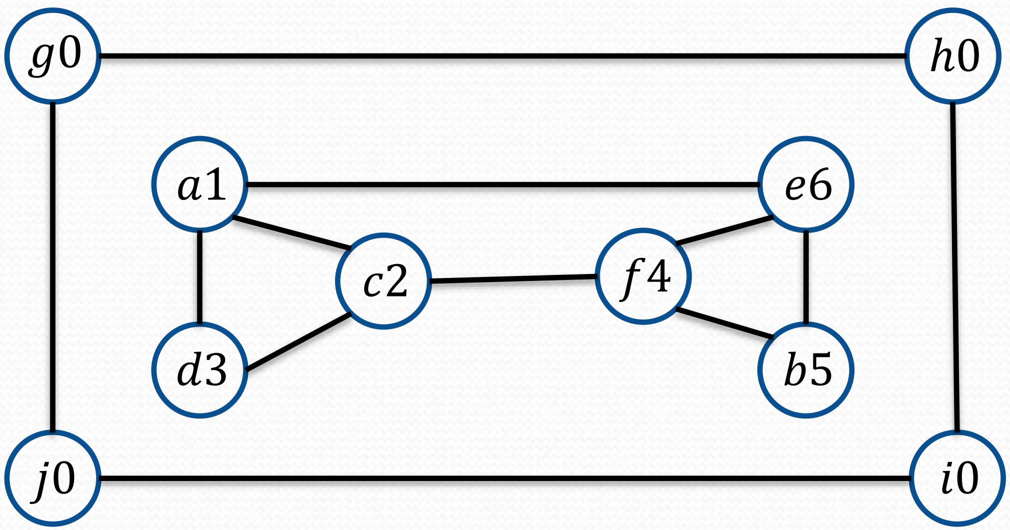


$count = 6$



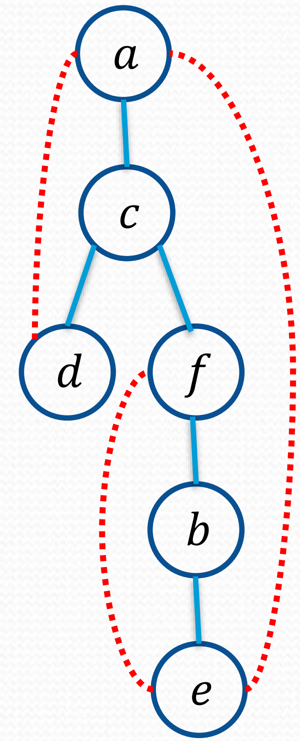
5、深度优先查找和广度优先查找

- 深度优先查找DFS



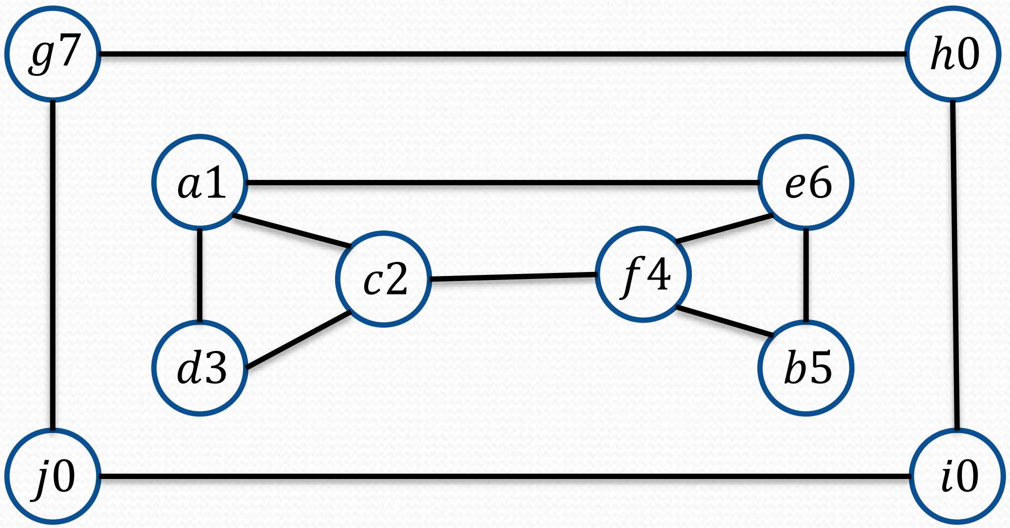
栈

$$count = 6$$



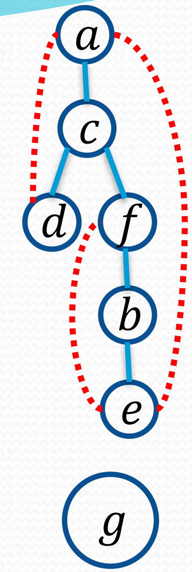
5、深度优先查找和广度优先查找

- 深度优先查找DFS



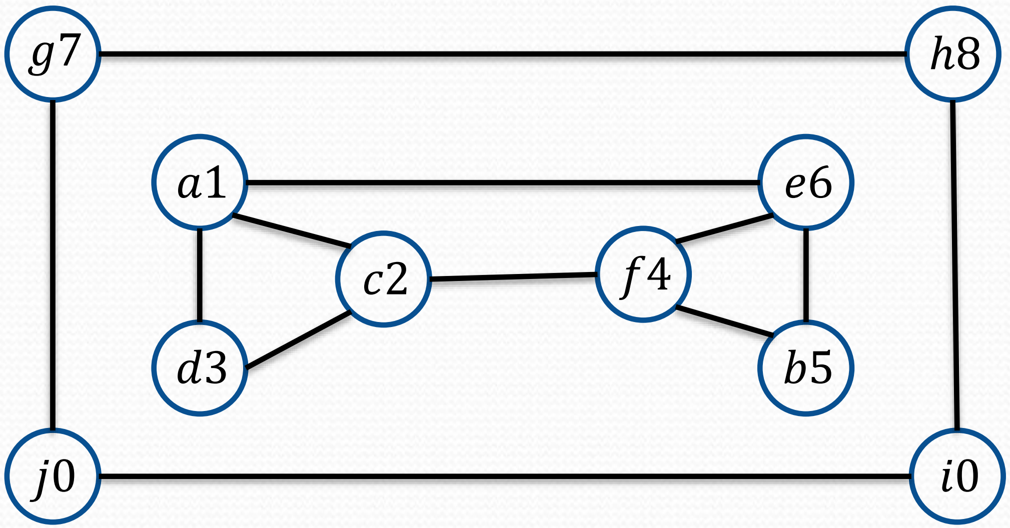
栈
$g(7)$

$count = 7$



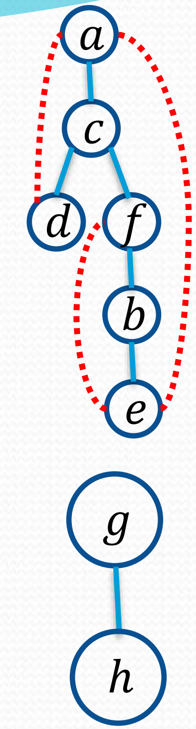
5、深度优先查找和广度优先查找

- 深度优先查找DFS



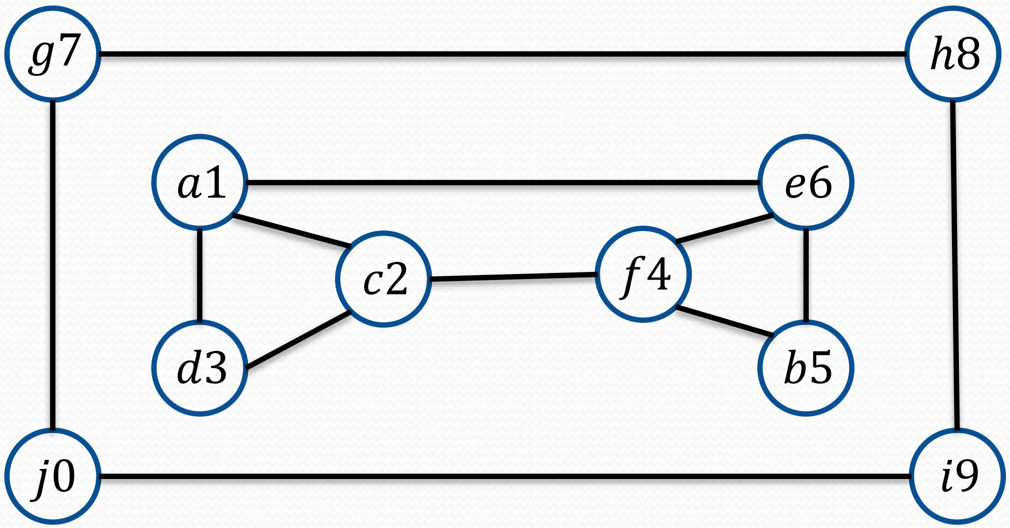
栈
$h(8)$
$g(7)$

$count = 8$



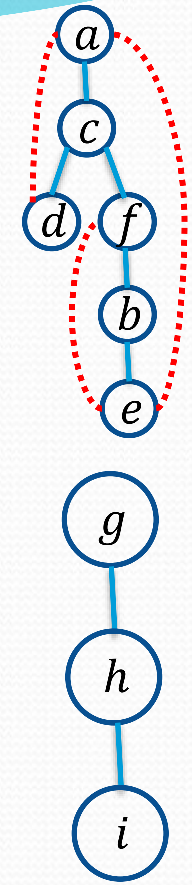
5、深度优先查找和广度优先查找

- 深度优先查找DFS



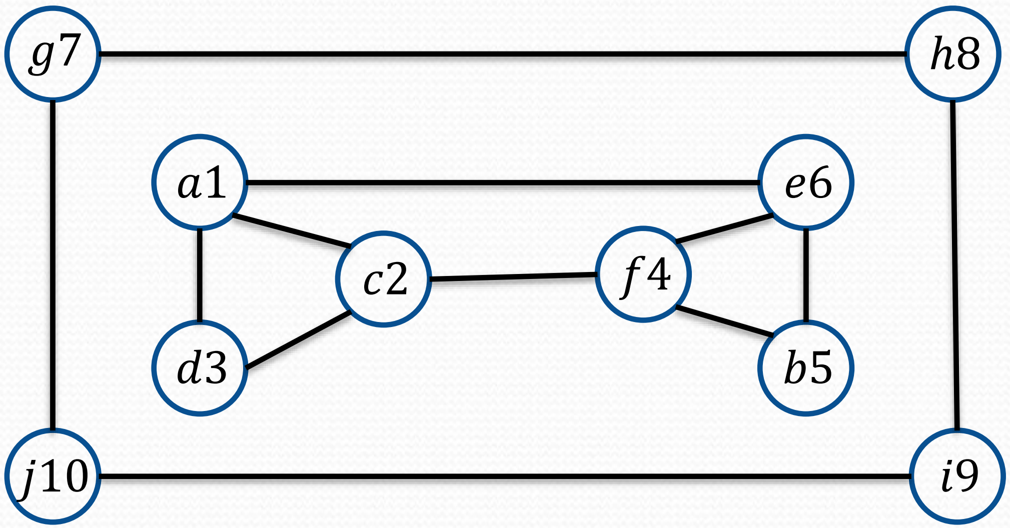
栈
$i(9)$
$h(8)$
$g(7)$

$$count = 9$$



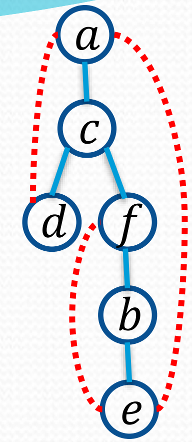
5、深度优先查找和广度优先查找

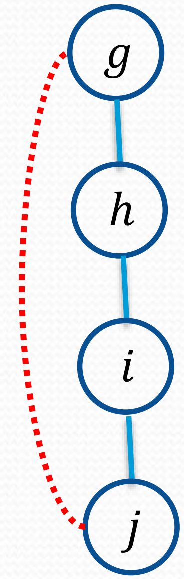
- 深度优先查找DFS



栈
$j(10)$
$i(9)$
$h(8)$
$g(7)$

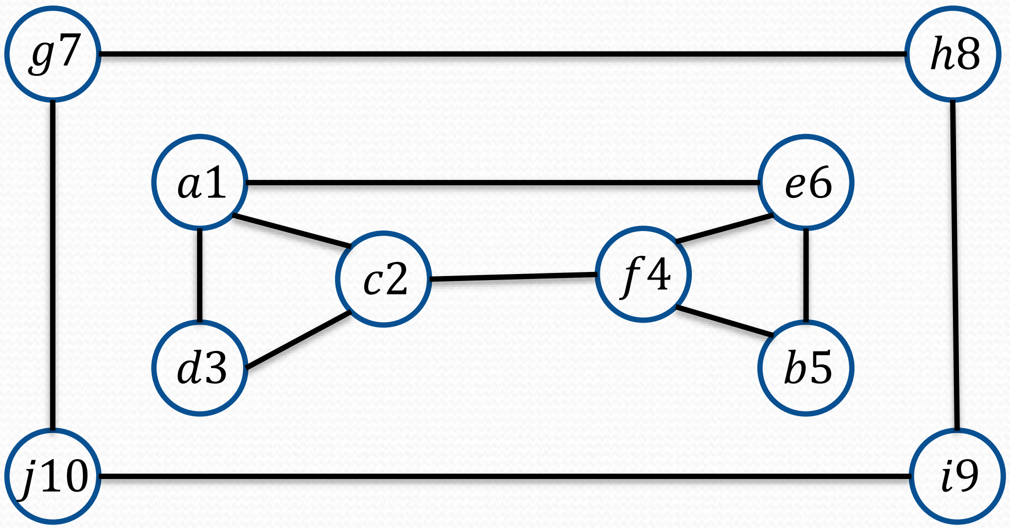
$count = 10$





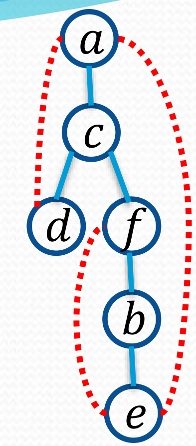
5、深度优先查找和广度优先查找

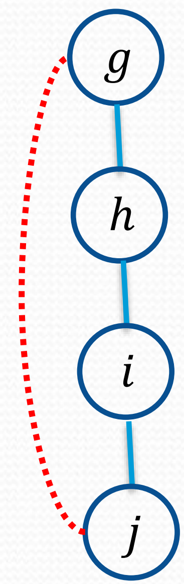
- 深度优先查找DFS



栈
$i(9)$
$h(8)$
$g(7)$

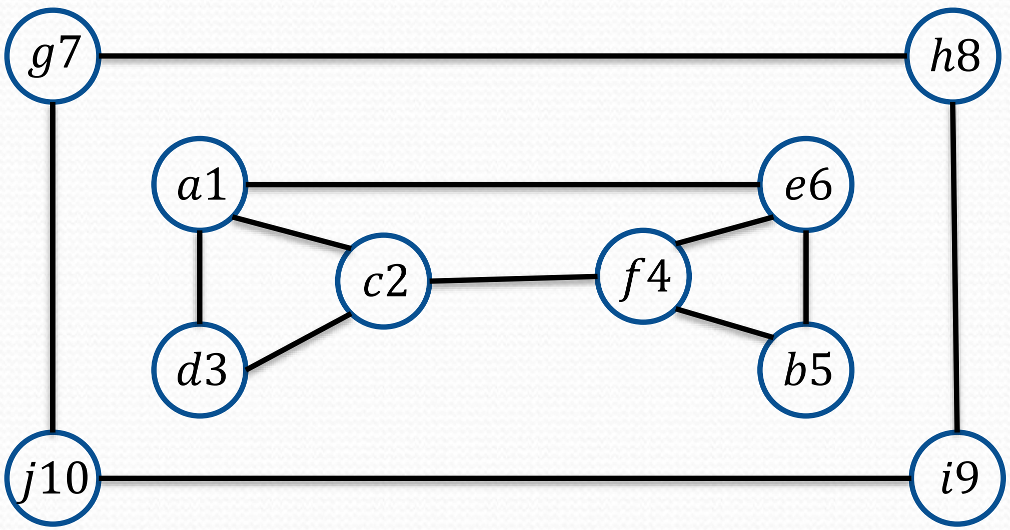
$count = 10$



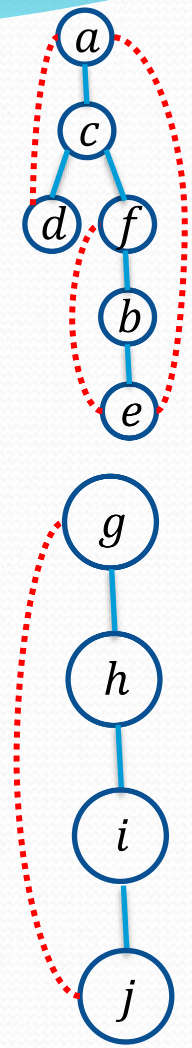


5、深度优先查找和广度优先查找

- 深度优先查找DFS

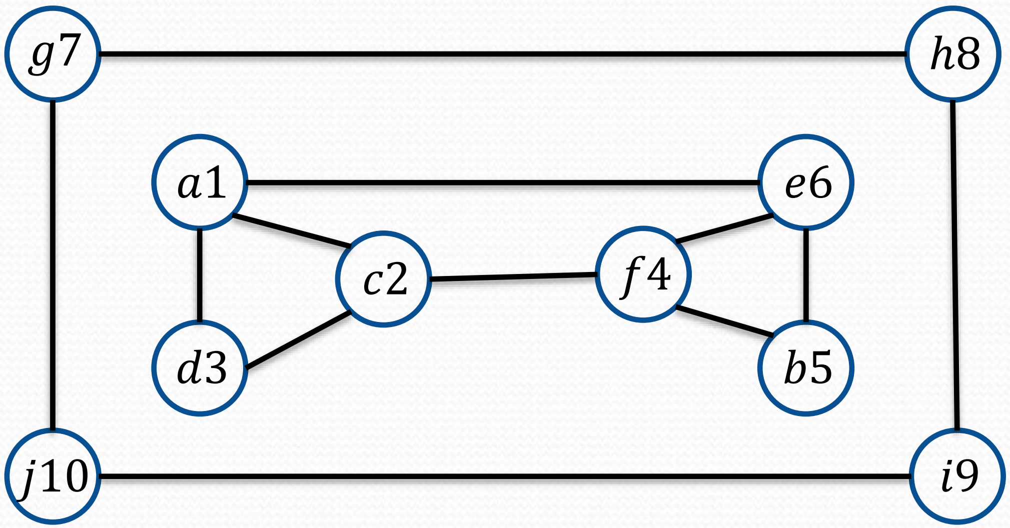


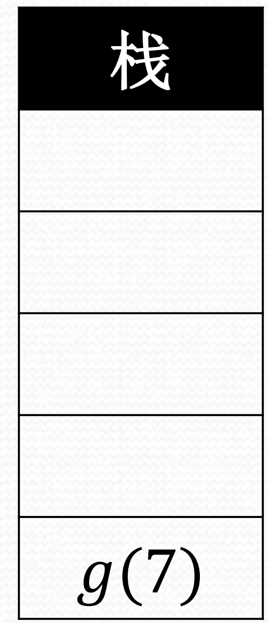
$count = 10$



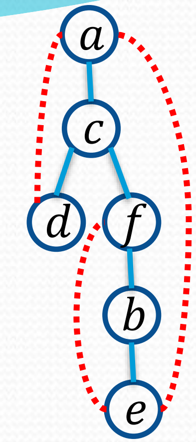
5、深度优先查找和广度优先查找

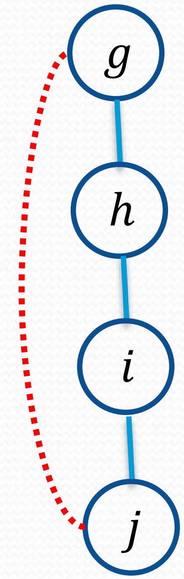
- 深度优先查找DFS





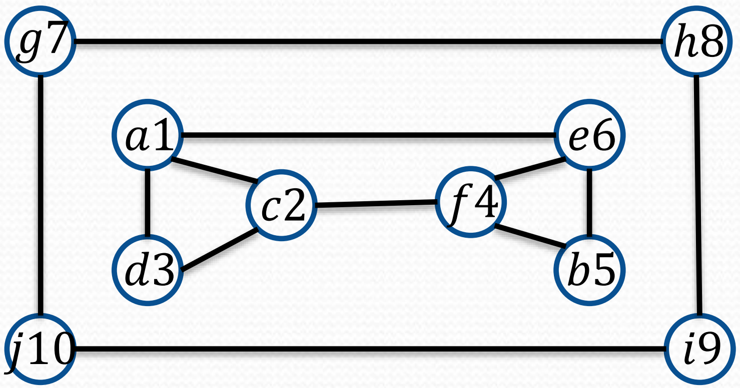
$count = 10$

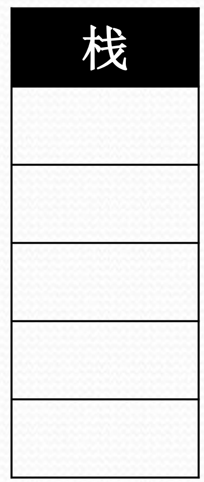




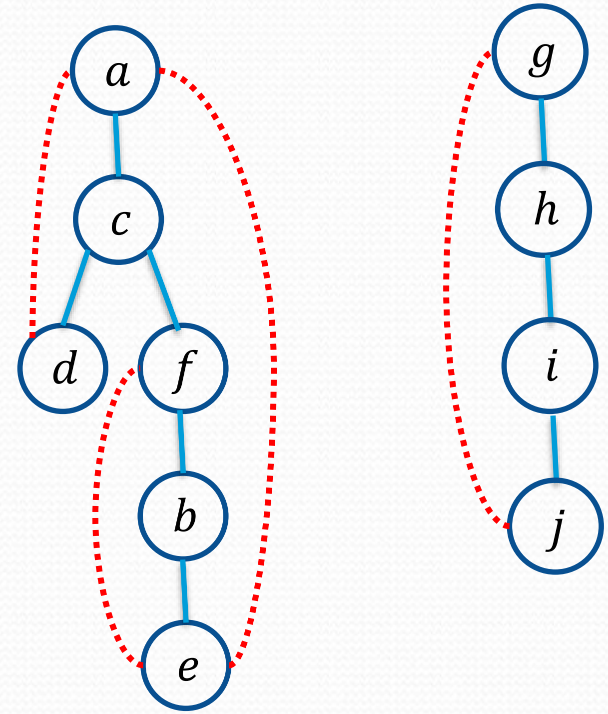
5、深度优先查找和广度优先查找

- 深度优先查找DFS





$count = 10$



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、深度优先查找和广度优先查找

- 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。
  - 具有两种顶点访问顺序：先序访问（顶点入栈的时候访问）和后序访问（顶点出栈的时候访问）。

5、深度优先查找和广度优先查找

- 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。
  - 利用回边查找无环性。

5、深度优先查找和广度优先查找

- 广度优先查找BFS
  - 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，循环
  - 直到所有与初始点联通的顶点访问完毕。
- 可根据广度优先遍历图，构造出广度优先查找森林，广度优先使用队列数据结构

5、深度优先查找和广度优先查找

- 广度优先查找BFS

算法 BFS(G) //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BFS访问顺序，用连续整数标记

V中每个顶点标记为0, 表示未访问

$count \leftarrow 0$

$for\ each\ vertex\ v\ in\ V\ do$

$if\ \ v\ is\ marked\ with\ 0$

$bfs(v)$

算法 $bfs(v)$

//访问所有和v相连接的未访问节点, 根据访问顺序,给他们赋值。

$count \leftarrow count + 1; mark\ v\ with\ count\ and\ initialize\ a\ queue\ with\ v;$

$while\ the\ queue\ is\ not\ empty\ do$

$for\ each\ vertex\ w\ in\ V\ adjacent\ to\ the\ front\ vertex\ do$

$if\ w\ is\ marked\ with\ 0$

$count = count + 1;$

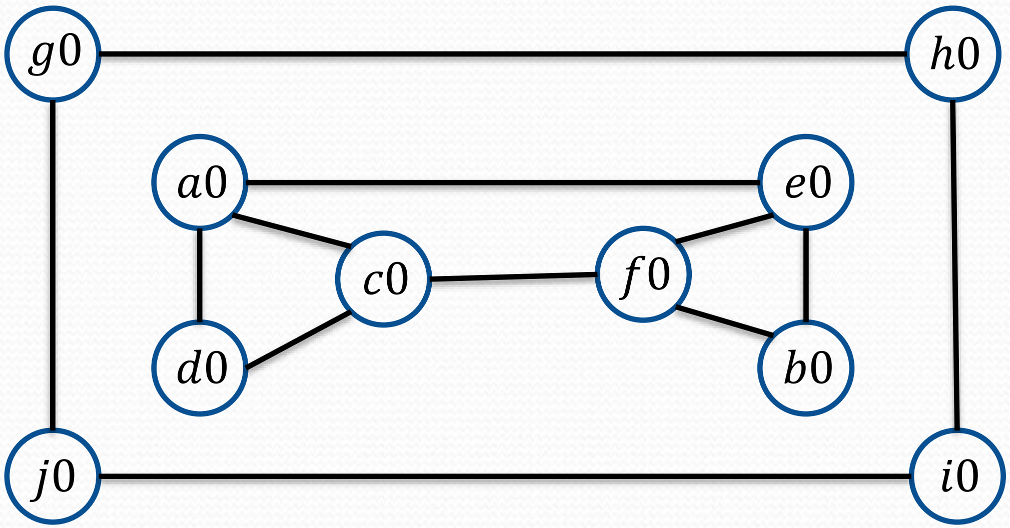
$mark\ w\ with\ count$

$add\ w\ to\ the\ queue$

$remove\ the\ front\ vertex\ from\ the\ queue$

5、深度优先查找和广度优先查找

- 广度优先查找BFS

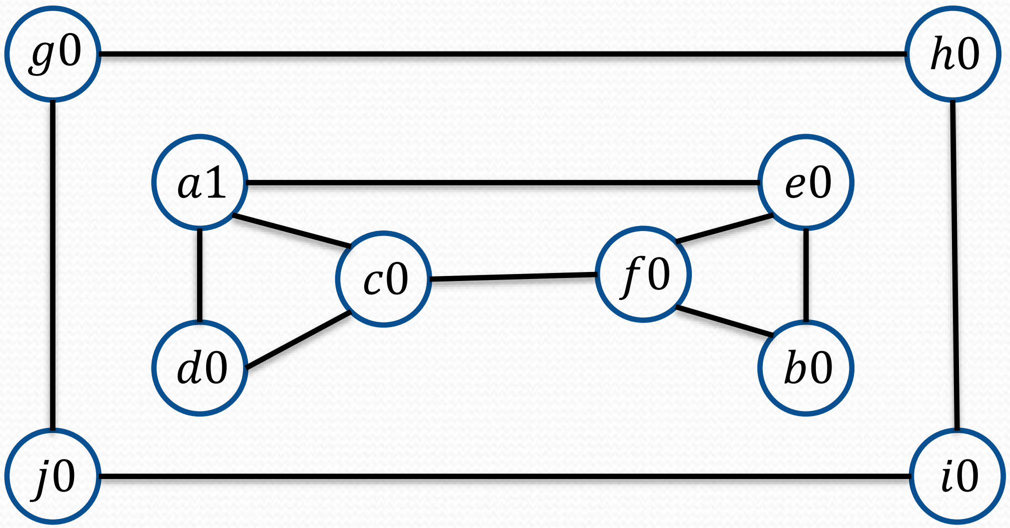


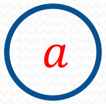
$$count = 0$$

队列								

5、深度优先查找和广度优先查找

- 广度优先查找BFS



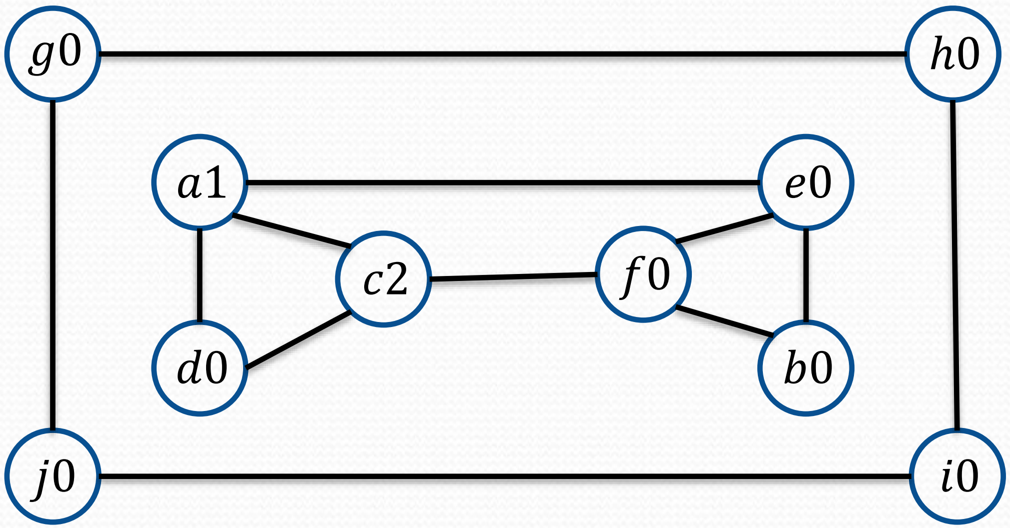


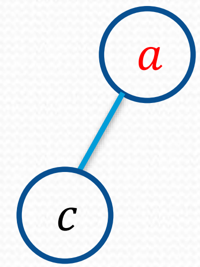
$$count = 1$$

队列								
a								

5、深度优先查找和广度优先查找

- 广度优先查找BFS





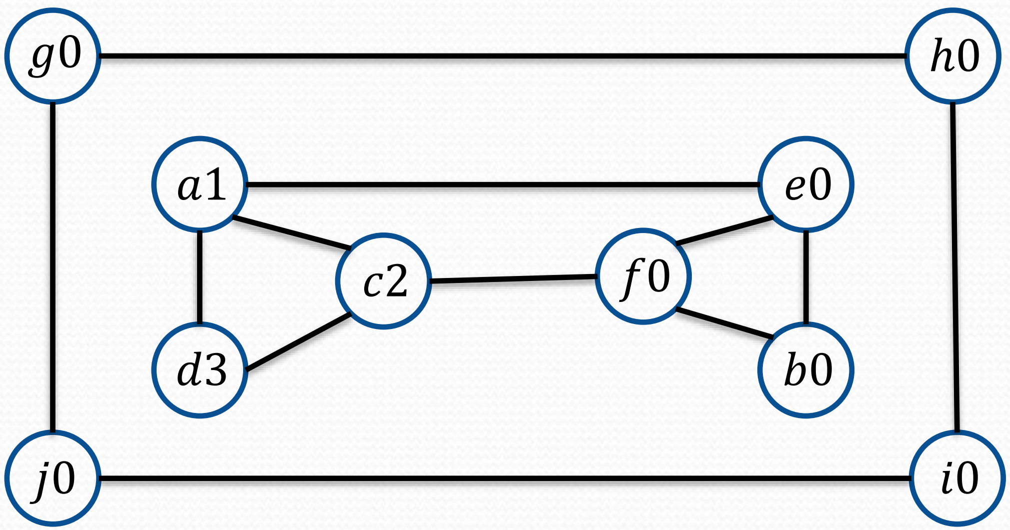
$$count = 2$$

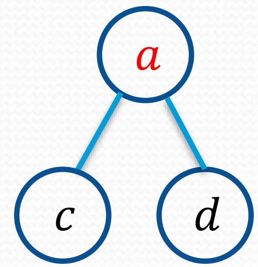
队列								
a	c							

实线 树向边，是连接顶点的边。

5、深度优先查找和广度优先查找

- 广度优先查找BFS



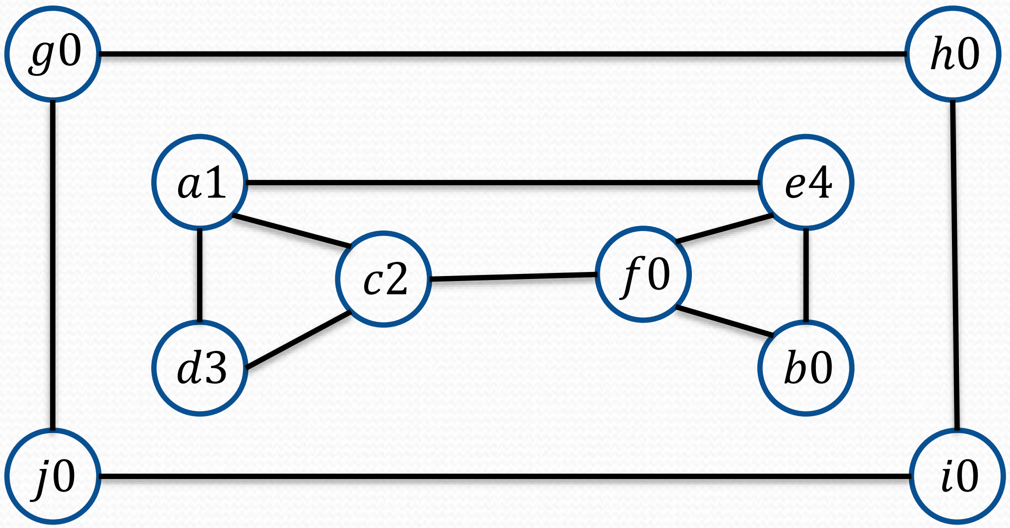


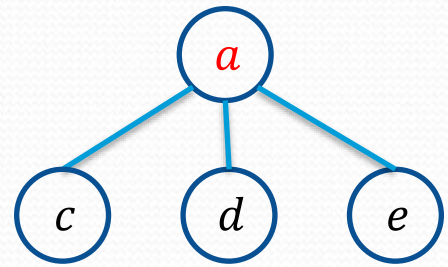
$count = 3$

队列								
a	c	d						

5、深度优先查找和广度优先查找

- 广度优先查找BFS



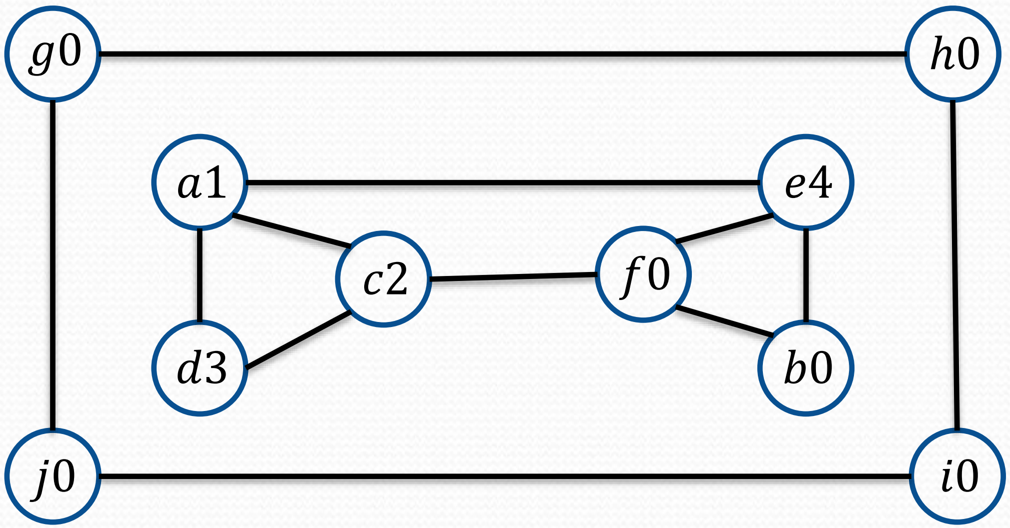


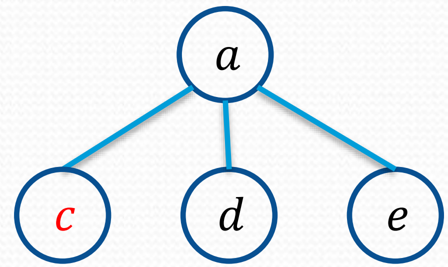
$$count = 4$$

队列								
a	c	d	e					

5、深度优先查找和广度优先查找

- 广度优先查找BFS



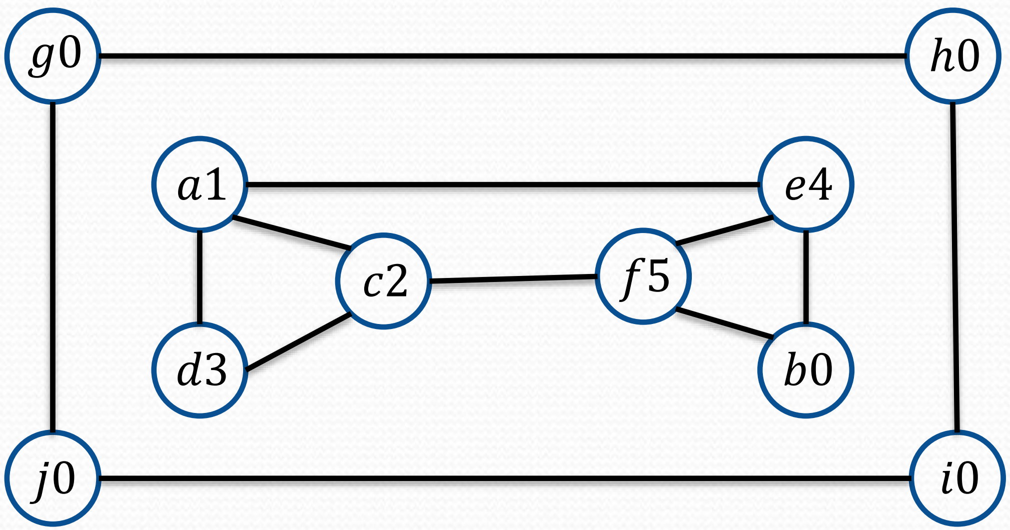


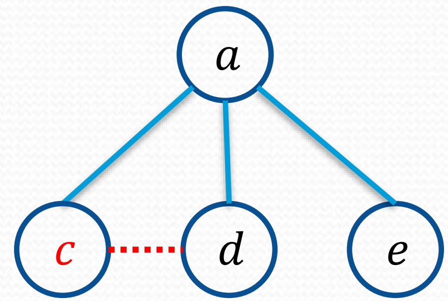
$$count = 4$$

队列								
	c	d	e					

5、深度优先查找和广度优先查找

- 广度优先查找BFS





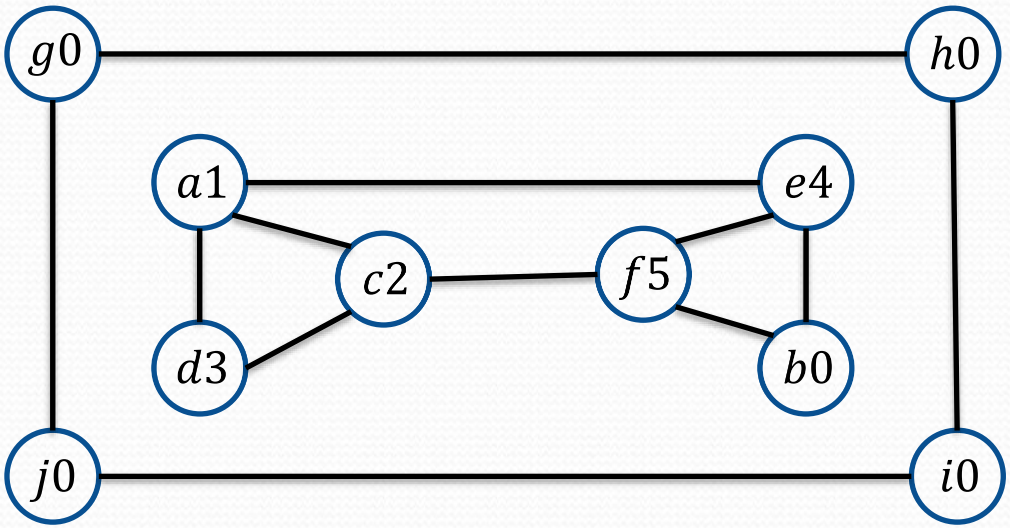
$count = 5$

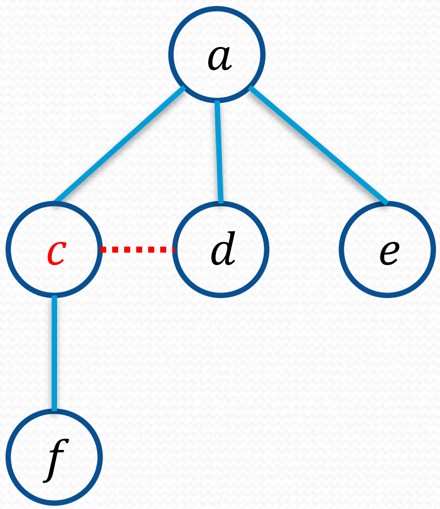
队列								
	c	d	e	f				

虚线 交叉边，是指向已访问的节点，且非直接前驱的边

5、深度优先查找和广度优先查找

- 广度优先查找BFS



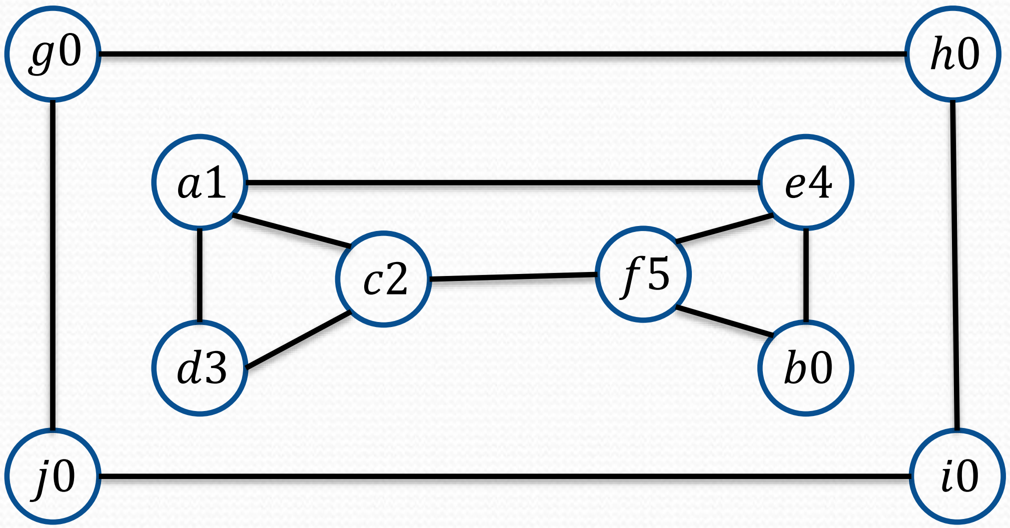


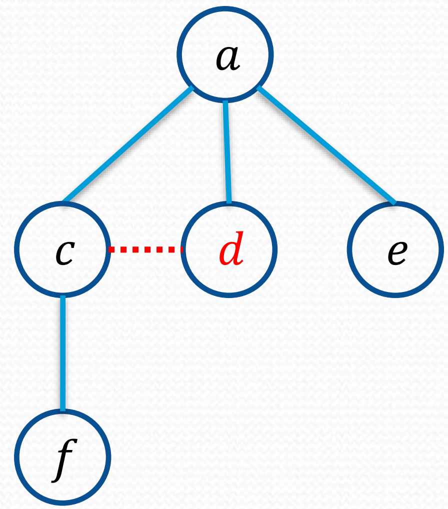
$$count = 5$$

队列								
	c	d	e	f				

5、深度优先查找和广度优先查找

- 广度优先查找BFS



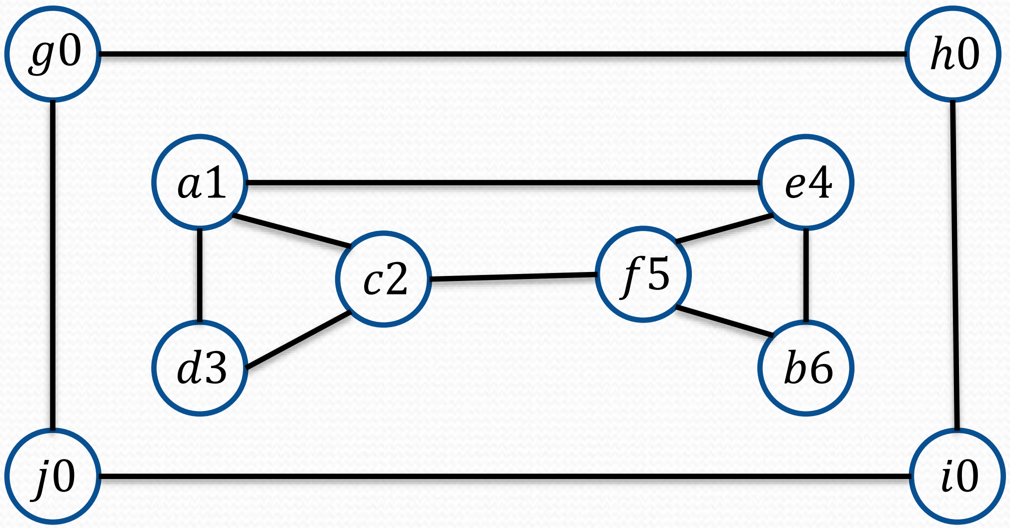


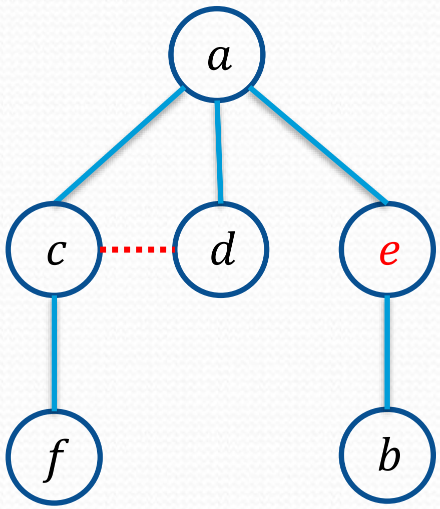
$$count = 5$$

队列								
		d	e	f				

5、深度优先查找和广度优先查找

- 广度优先查找BFS



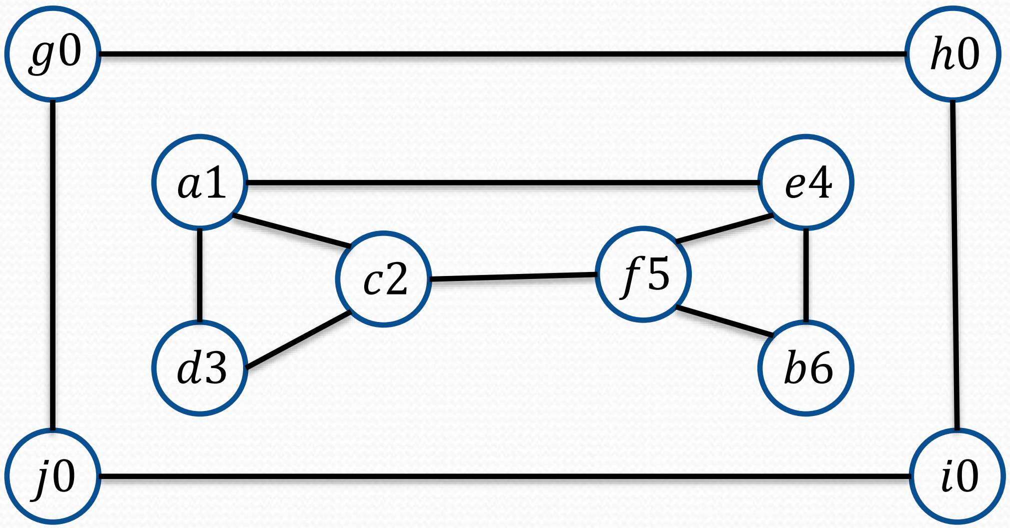


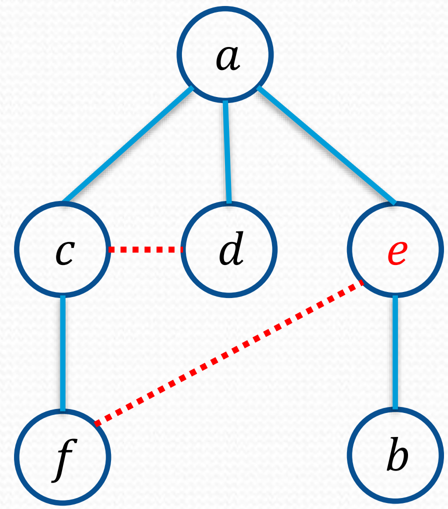
$$count = 6$$

队列								
			e	f	b			

5、深度优先查找和广度优先查找

- 广度优先查找BFS



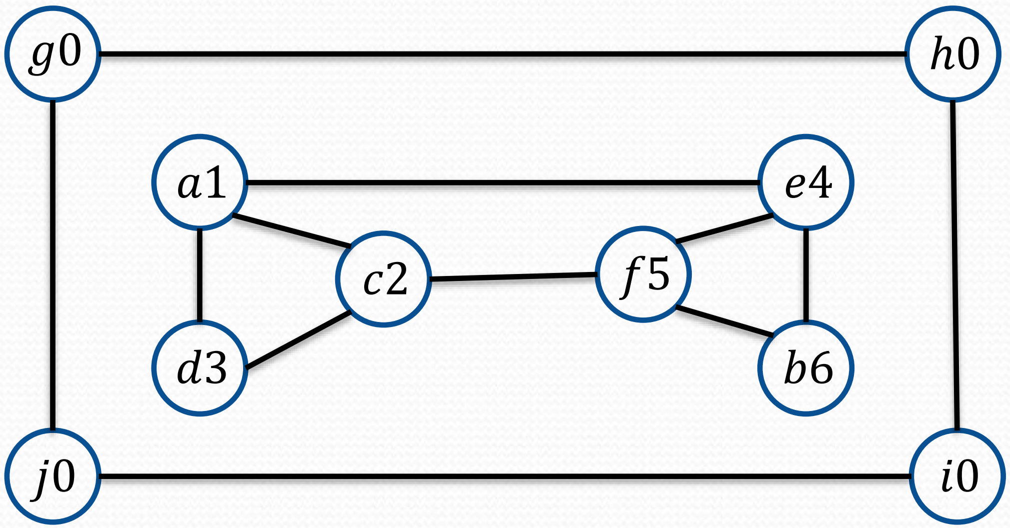


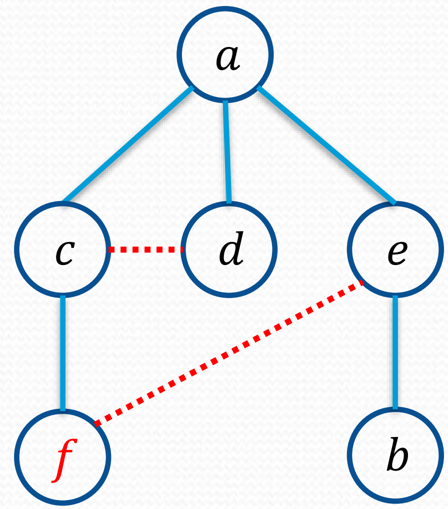
$$count = 6$$

队列								
			e	f	b			

5、深度优先查找和广度优先查找

- 广度优先查找BFS



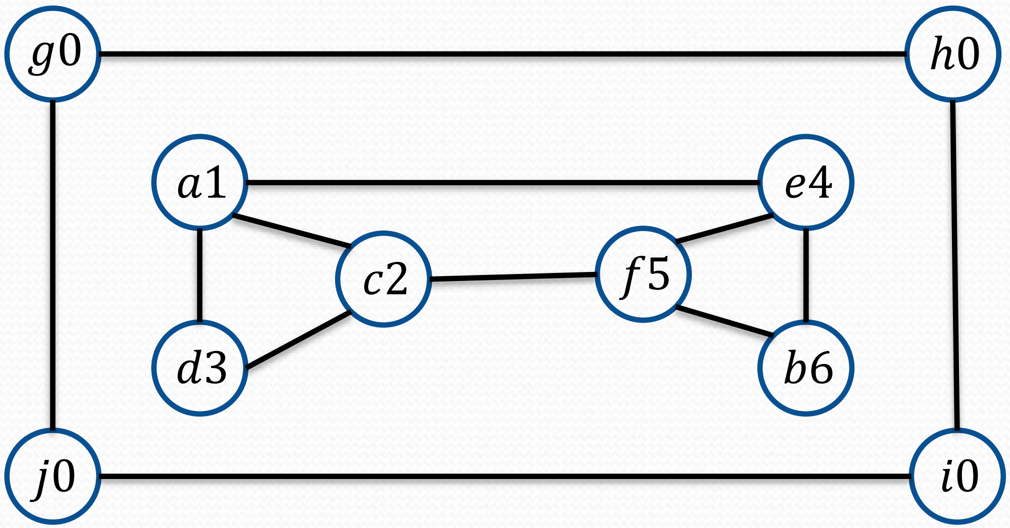


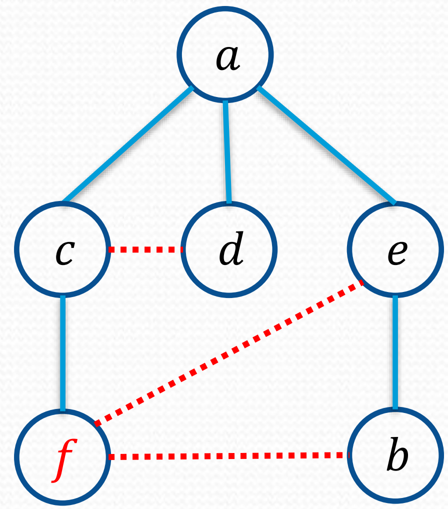
$count = 6$

队列								
				f	b			

5、深度优先查找和广度优先查找

- 广度优先查找BFS



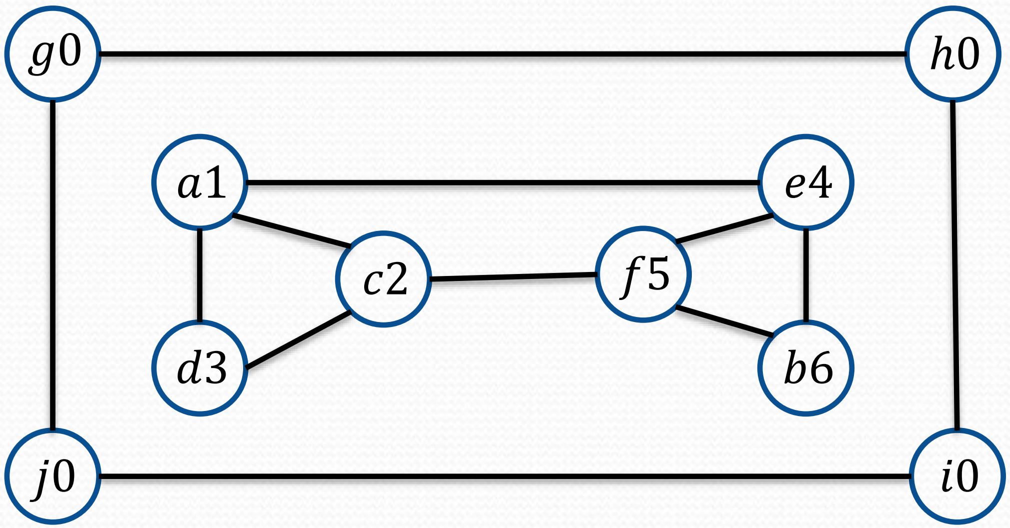


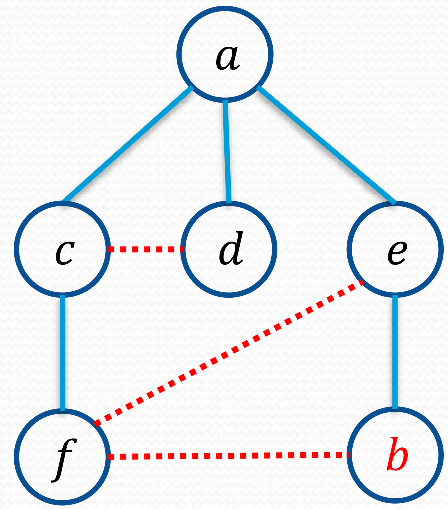
$$count = 6$$

队列								
				f	b			

5、深度优先查找和广度优先查找

- 广度优先查找BFS



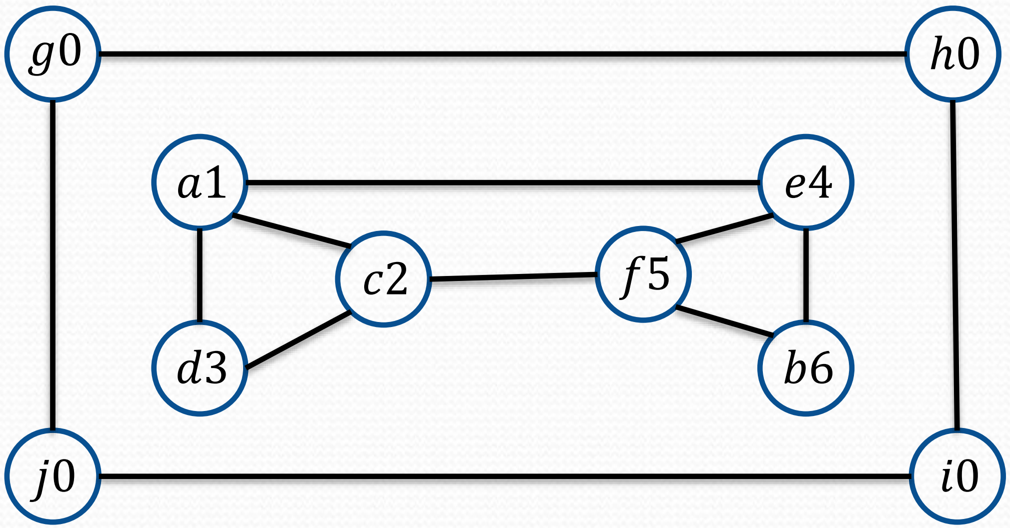


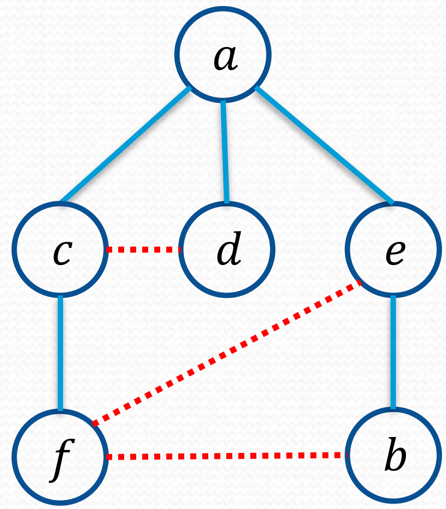
$$count = 6$$

队列								
					b			

5、深度优先查找和广度优先查找

- 广度优先查找BFS



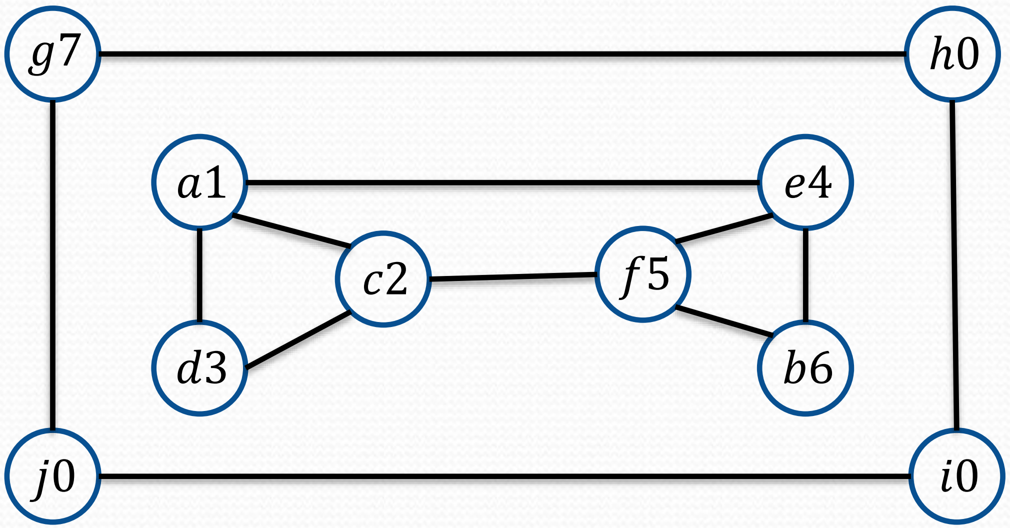


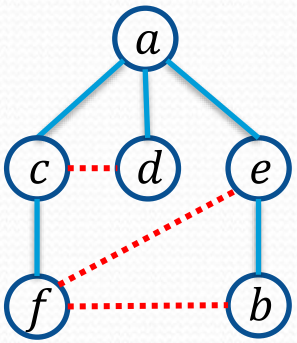
$count = 6$

队列								

5、深度优先查找和广度优先查找

- 广度优先查找BFS



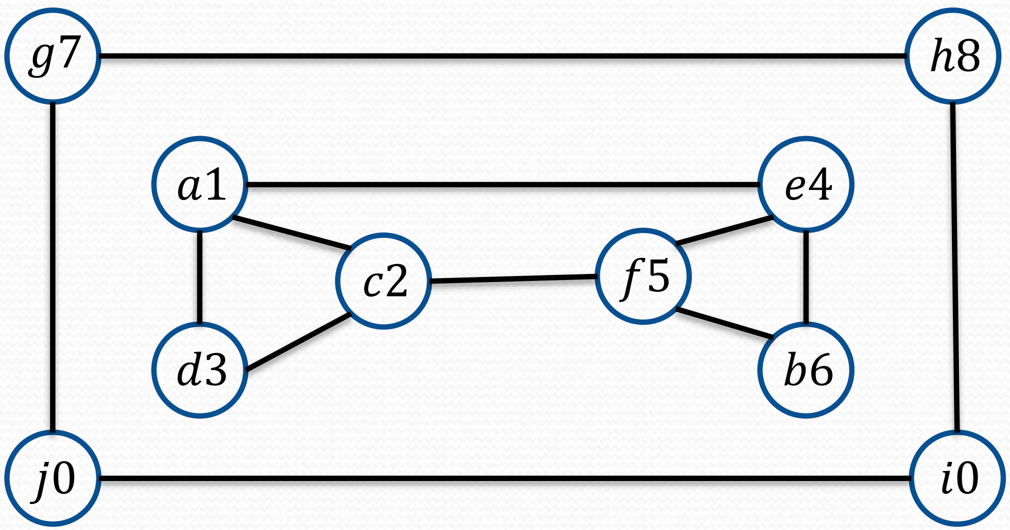


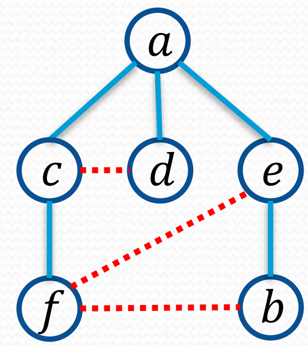
$count = 7$

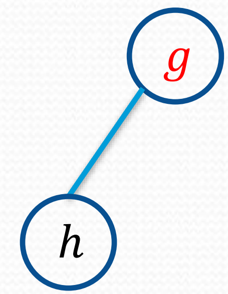
队列								
g								

5、深度优先查找和广度优先查找

- 广度优先查找BFS





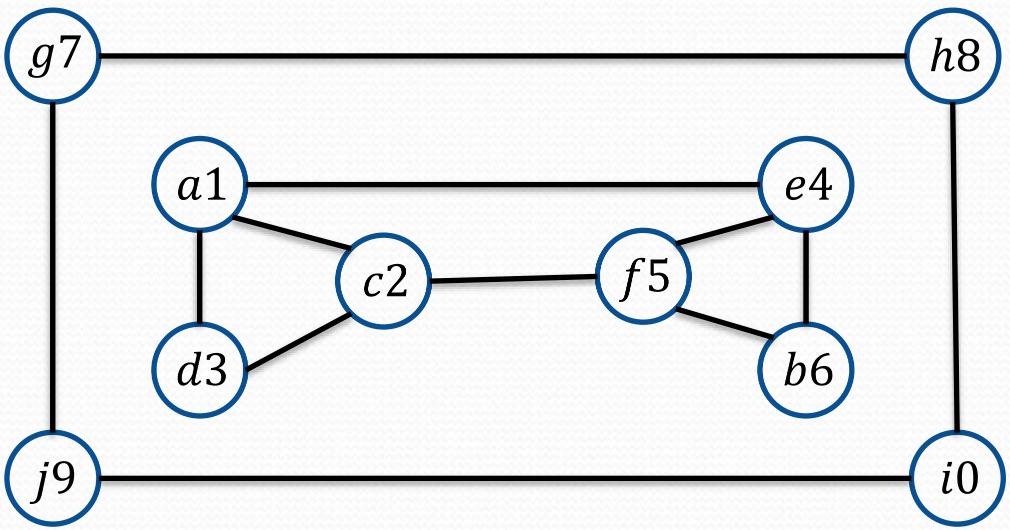


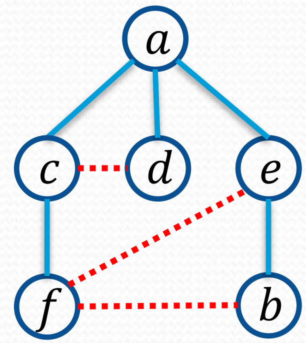
$$count = 8$$

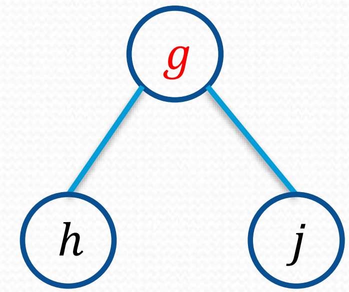
队列								
g	h							

5、深度优先查找和广度优先查找

- 广度优先查找BFS





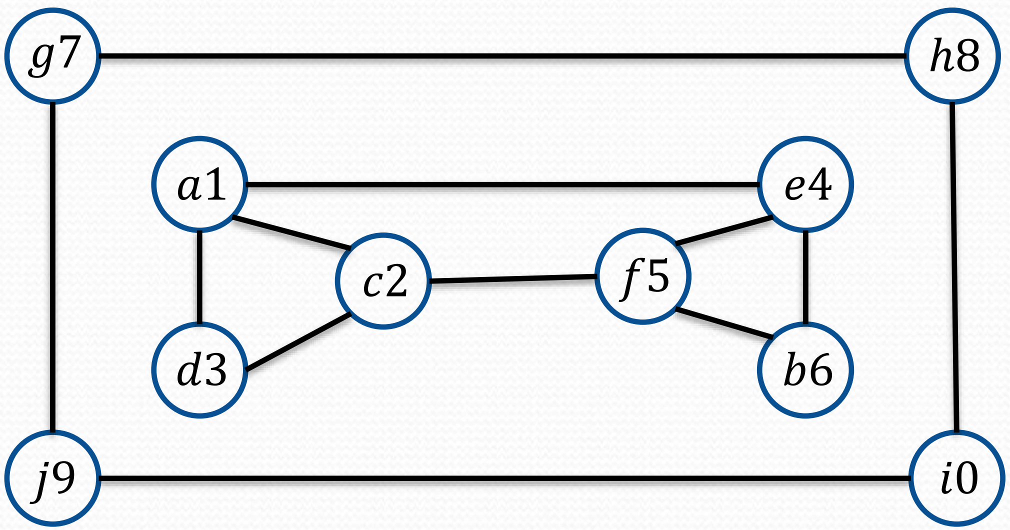


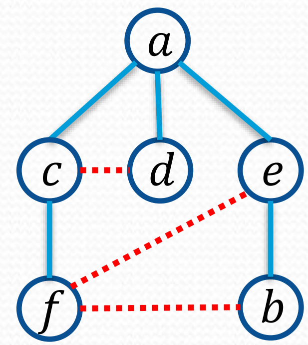
$$count = 9$$

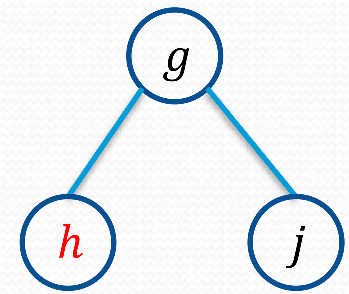
队列								
g	h	j						

5、深度优先查找和广度优先查找

- 广度优先查找BFS





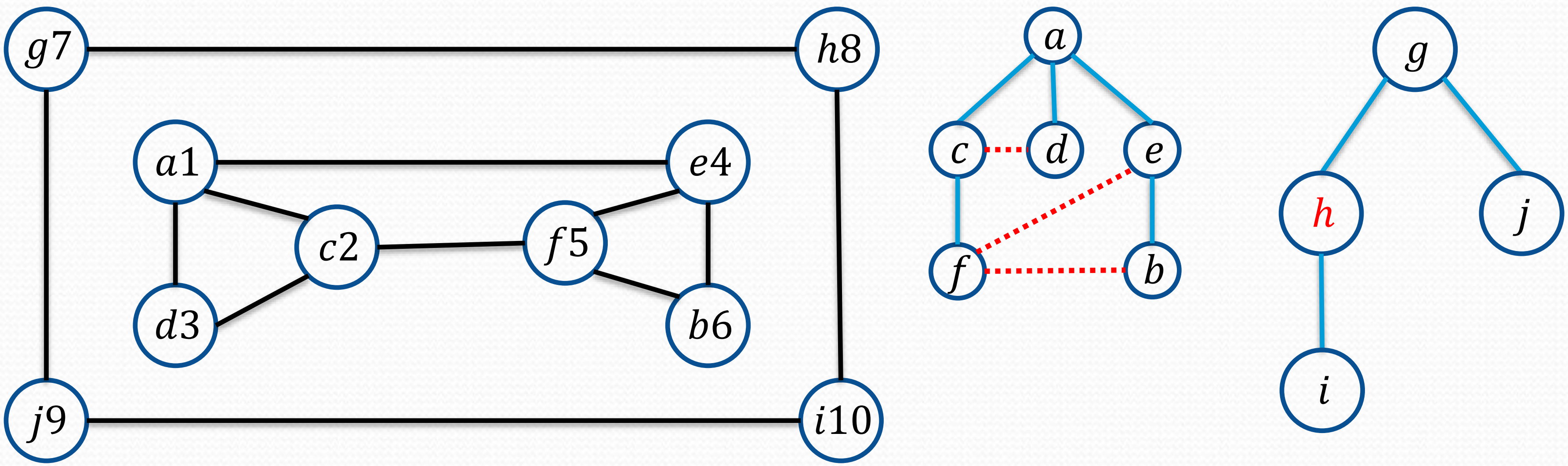


$$count = 9$$

队列								
	h	j						

5、深度优先查找和广度优先查找

- 广度优先查找BFS

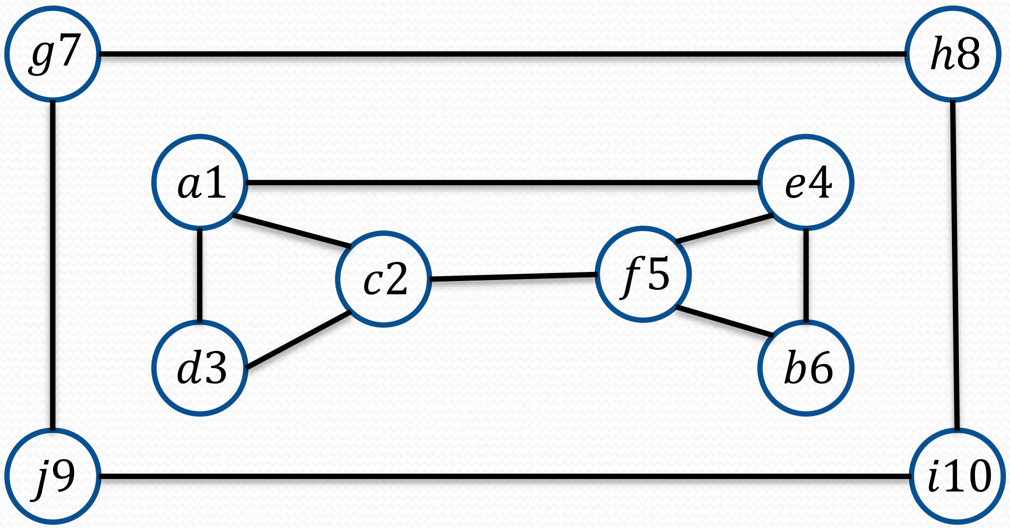


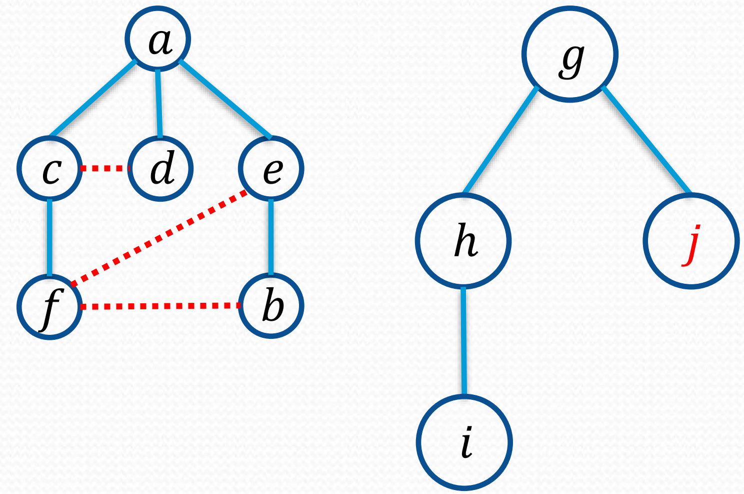
$$count = 10$$

队列								
	h	j	i					

5、深度优先查找和广度优先查找

- 广度优先查找BFS



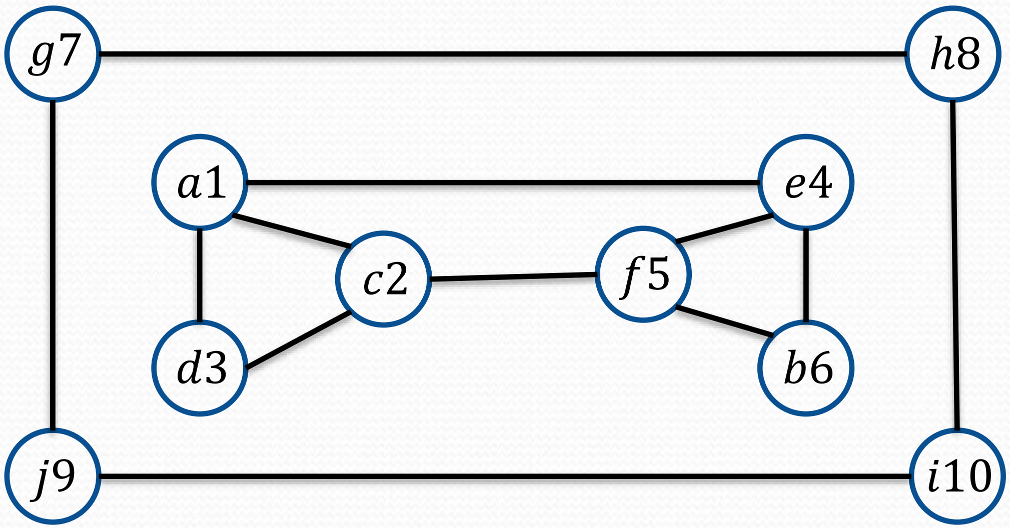


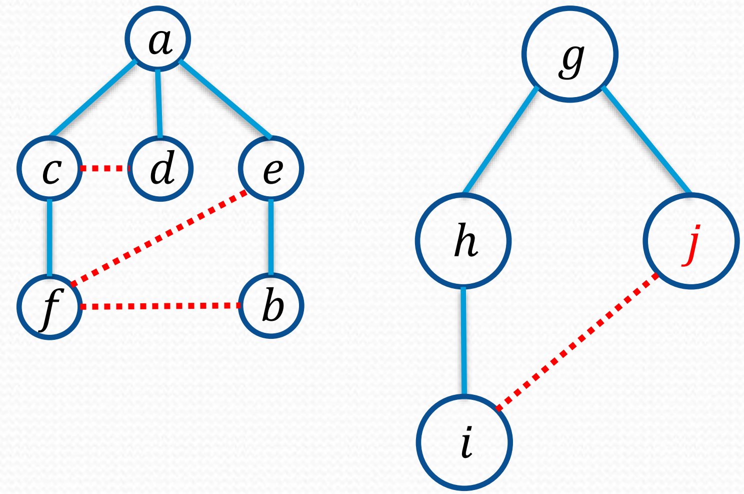
$count = 10$

队列								
		j	i					

5、深度优先查找和广度优先查找

- 广度优先查找BFS



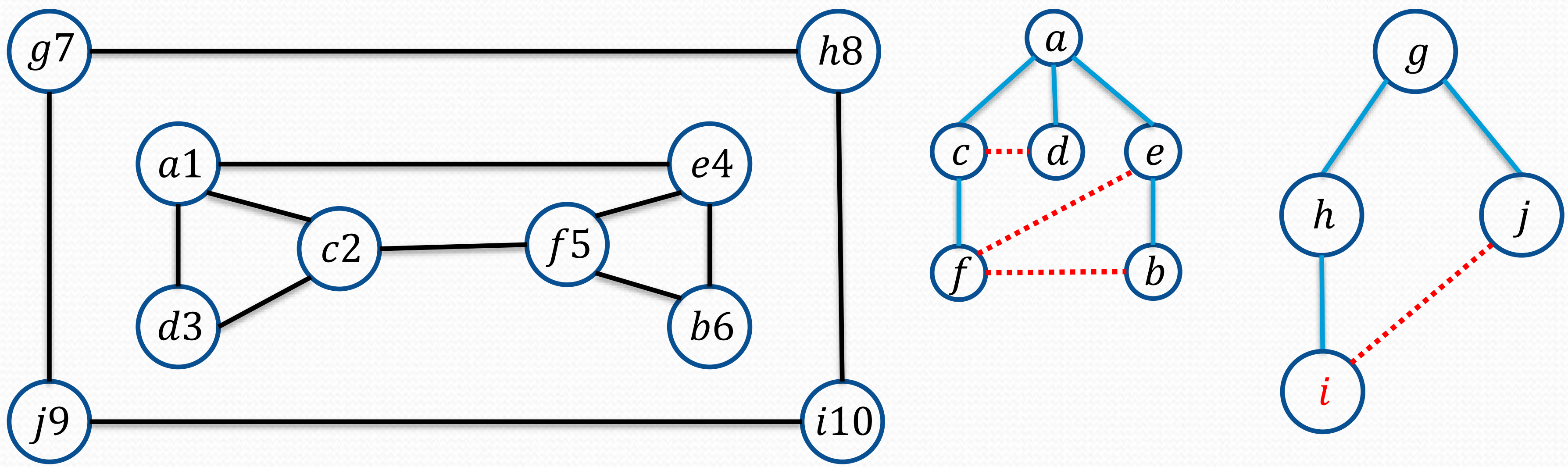


$$count = 10$$

队列								
		j	i					

5、深度优先查找和广度优先查找

- 广度优先查找BFS

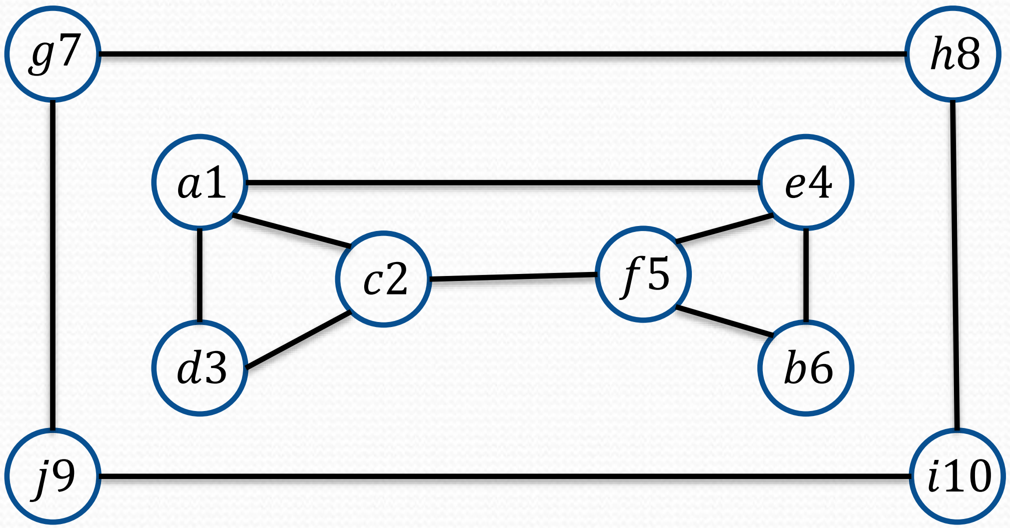


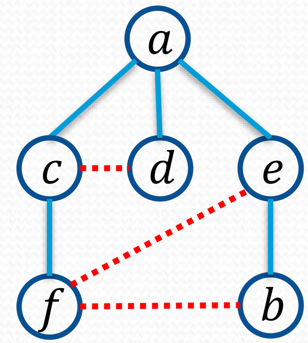
$count = 10$

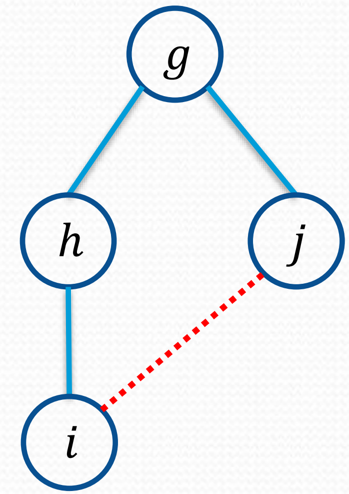
队列								
			i					

5、深度优先查找和广度优先查找

- 广度优先查找BFS







$$count = 10$$

队列								

5、深度优先查找和广度优先查找

- 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、深度优先查找和广度优先查找

- 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、深度优先查找和广度优先查找

- 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	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert^2)$
邻接链表的效率	$\Theta(\lvert V\rvert+\lvert E\rvert)$	$\Theta(\lvert V\rvert+\lvert E\rvert)$

蛮力法总结

- 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。