西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计
Programming in C

1011014

主讲：姜学锋，计算机学院

图搜索与回溯算法

1、图搜索算法

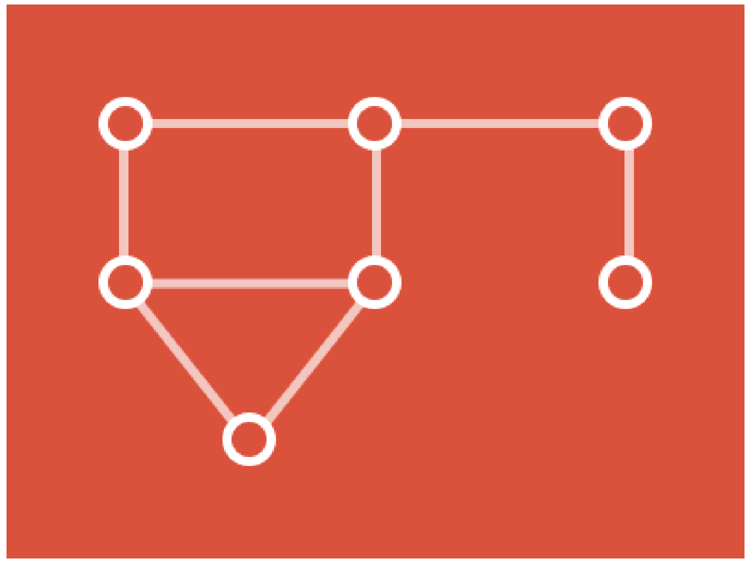
2、广度优先搜索BFS

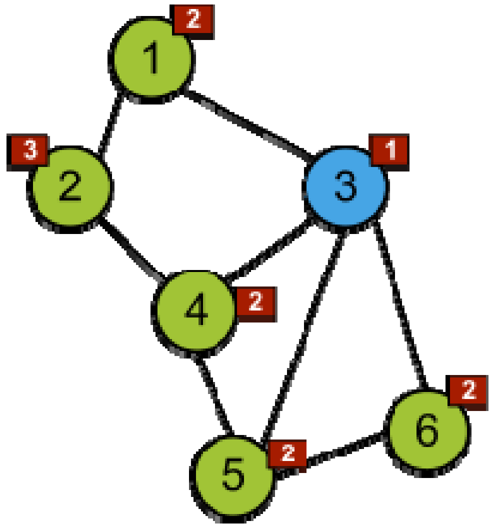
3、深度优先搜索DFS

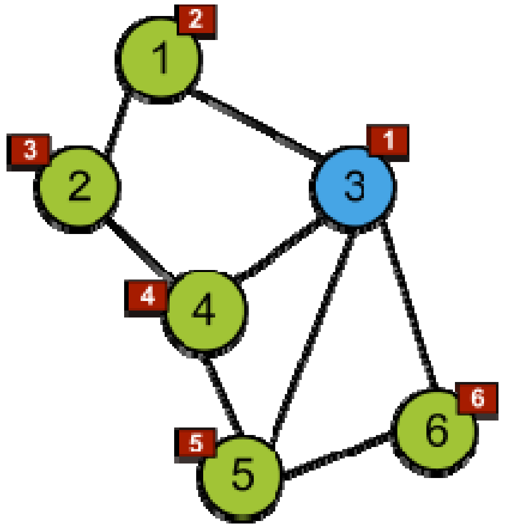
4、回溯算法

11.3.6 图搜索算法

- 搜索算法(Search Algorithm)
- 使用计算机求解的问题时，有许多问题是无法用数学公式进行计算或采用模拟方法来找出答案的。这样的问题往往需要根据问题所给定的一些条件，在问题的所有可能解中用某种方式找出问题的解来，这就是所谓的搜索算法。







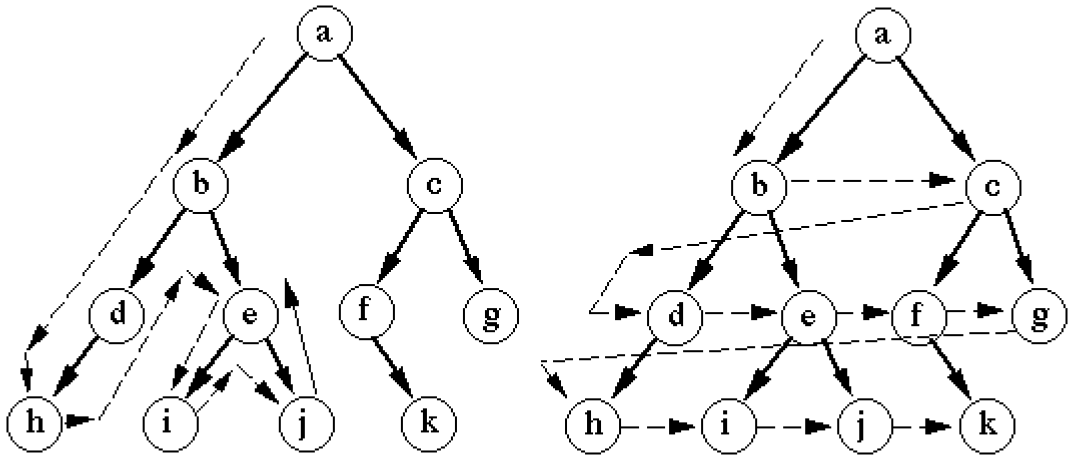
11.3.6 图搜索算法

- 搜索算法(Search Algorithm)
- 通常用搜索算法解决的问题可以分成两类：一类问题是给定初始结点，要求找出符合约束条件的目标结点；另一类问题是给出初始结点和目标结点，找出一条从初始结点到达目标结点的路径。

- 常见的搜索算法有枚举法、广度优先搜索法、深度优先搜索法、双向广度优先搜索法、A*算法、回溯法、分支限界法等。

11.3.6 图搜索算法

- 图搜索算法(Graph Search Algorithm)
- 搜索一个图指是有序地沿着图的边访问所有顶点。搜索过程实际上是根据初始条件和扩展规则构造一棵解答树并寻找符合目标状态的节点的过程。

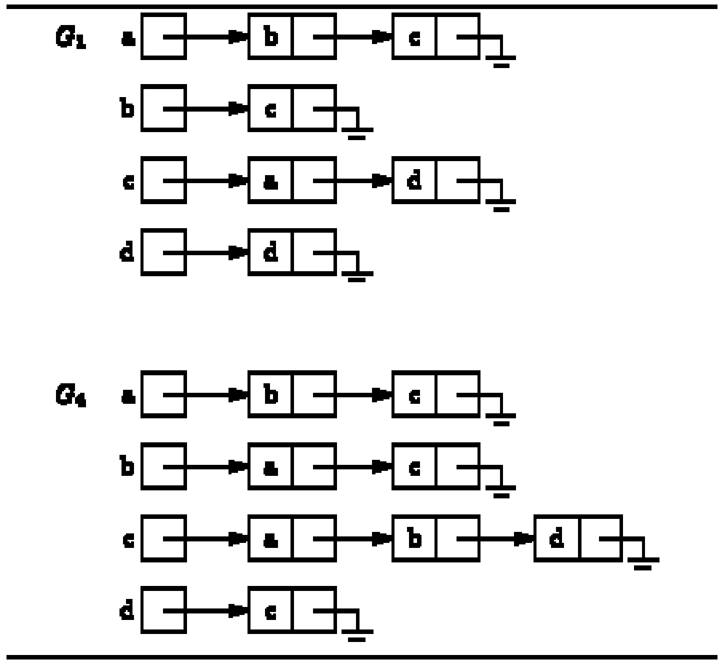


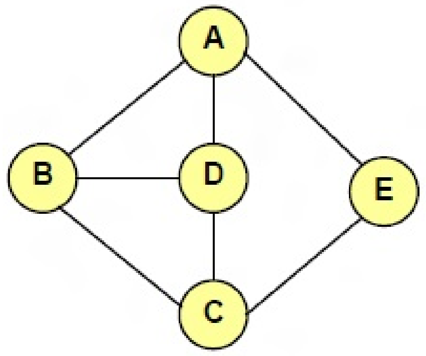
Depth-first search

Breadth-first search

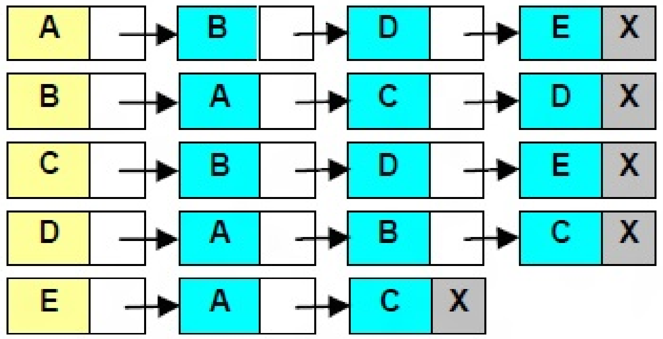
11.3.6 图搜索算法

- 最常用的图表示方法有两种：邻接表（Adjacency list），邻接矩阵（Adjacency matrix）。还有十字链表、邻接多重表等。邻接表适用于稀疏图，通常使用的是邻接表，因为大部分图都不会过于稠密。





Undirected Graph



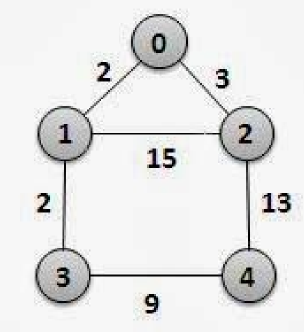
Adjacency list

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	1	0
C	0	1	0	1	1
D	1	1	1	0	0
E	1	0	1	0	0

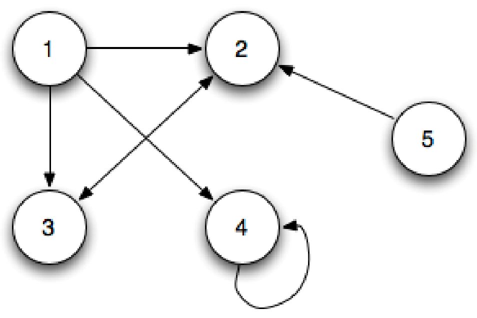
Adjacency matrix

11.3.6 图搜索算法

- 邻接矩阵适用于稠密图。由于邻接矩阵简单明了，当图较小时，更多地采用邻接矩阵。如果一个图不是加权的，采用邻接矩阵还有一个好处：在储存邻接矩阵的每个元素时，可以只使用一个二进位，而不必用一个整型的空间。



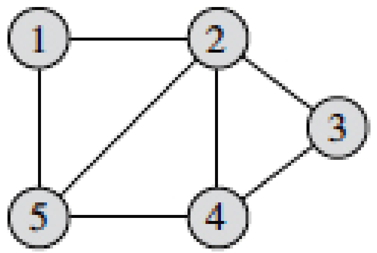
	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0



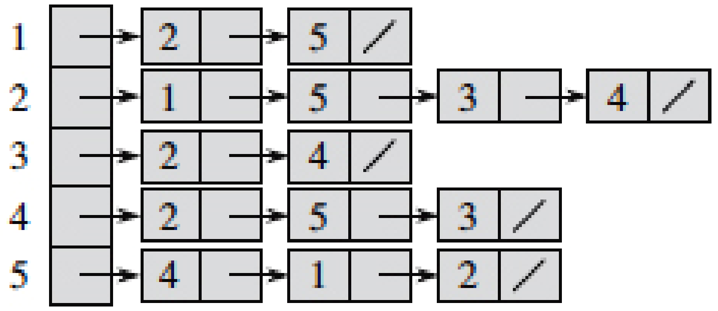
	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	1	0	0	0

11.3.6 图搜索算法

- 图示(a)是图的邻接表表示，一个有5个顶点和7条边的无向图G，(b)是图的邻接表表示，(c)G的邻接矩阵表示。



(a)



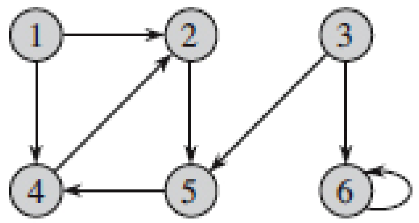
(b)

$$\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

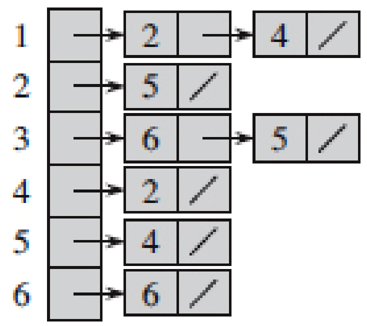
(c)

11.3.6 图搜索算法

- 图示(a)有6个顶点和8条边的有向图G，(b)G的邻接表表示，(c)G的邻接矩阵表示。



(a)



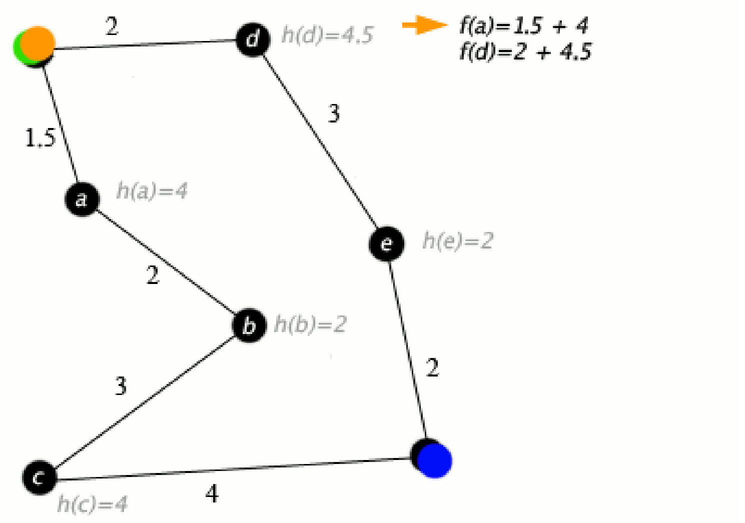
(b)

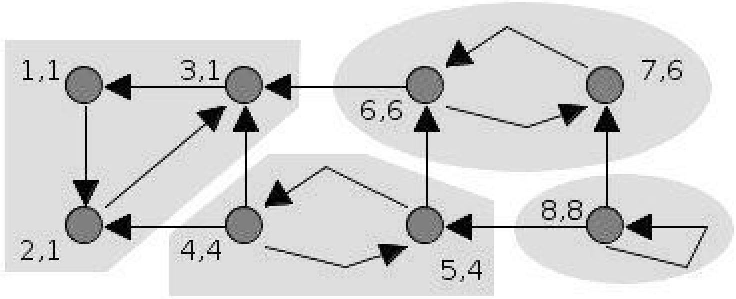
$$
\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
$$

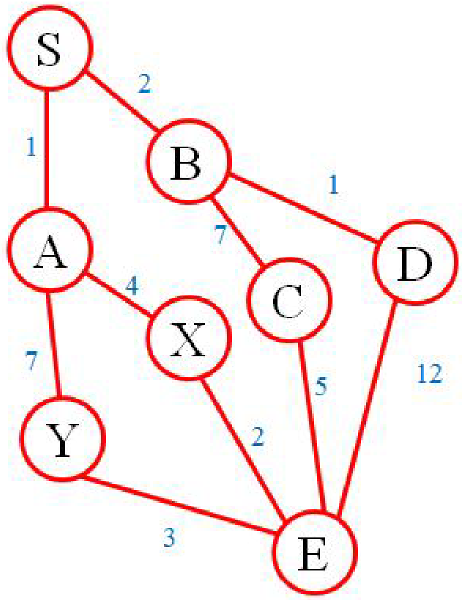
(c)

11.3.6 图搜索算法

- 基本的图搜索算法有两种：深度优先搜索（Depth-first Search）、广度优先搜索（Breadth-first Search）。
- 很多高级图算法的核心过程都用到了这两种搜索思想，比如A*，IDA*，双向广搜，计算有向图强连通分量（Kosaraju算法和Tarjan算法）等等。

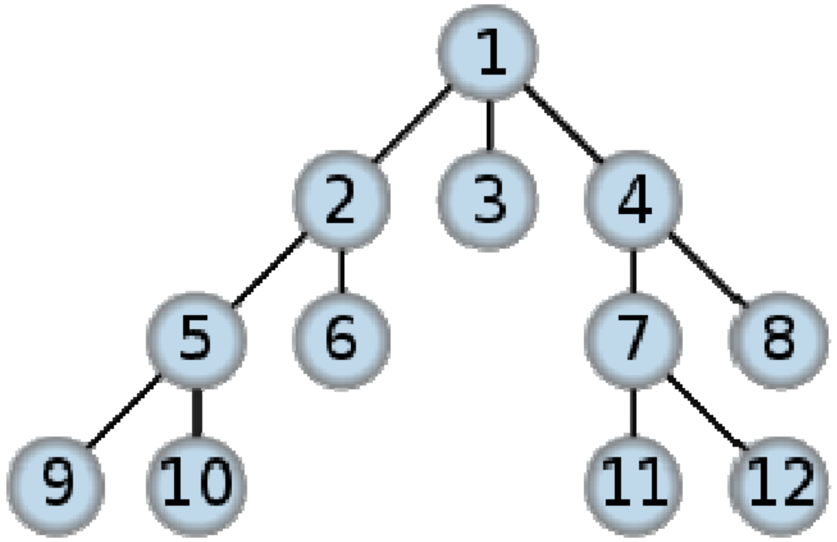


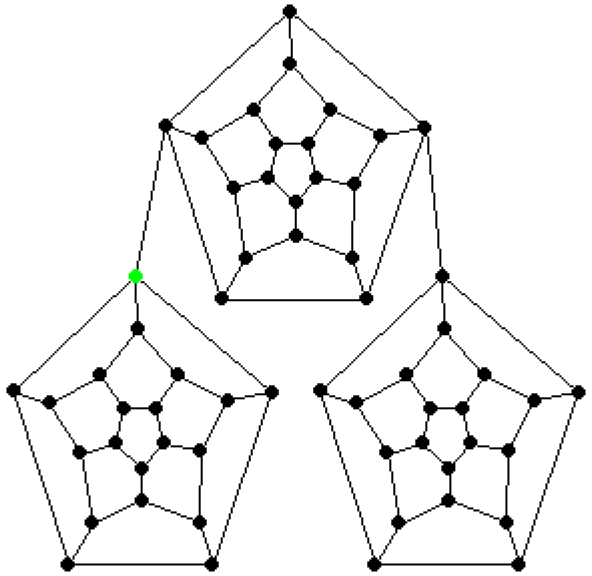


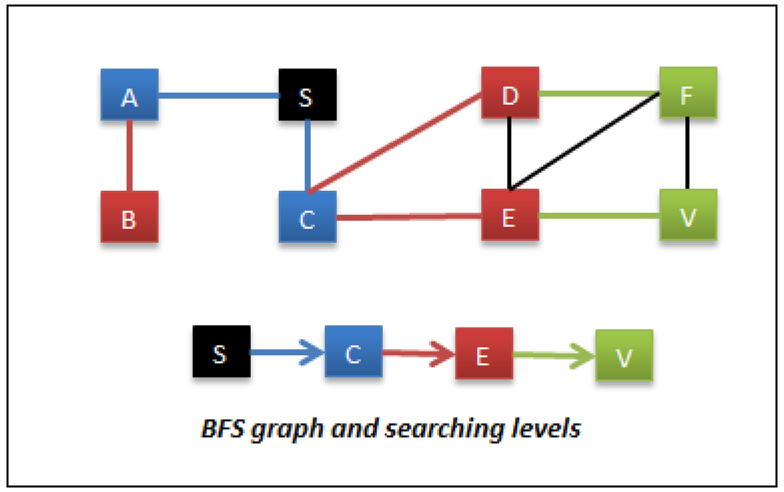


11.3.7 广度优先搜索BFS

- 一、广度优先搜索（广搜，BFS，Breadth-first Search）
- 在给定图G=(V,E)和一个源顶点s的情况下，广度优先搜索系统地探索G中的边，以发现可从s到达的所有顶点，并计算s到所有这些可达顶点之间的距离（即最少的边数）。



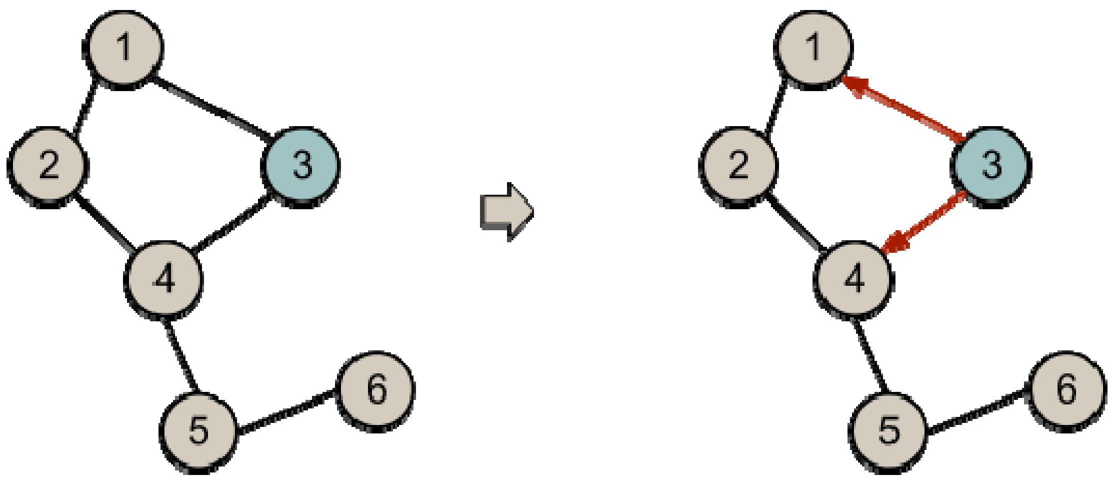


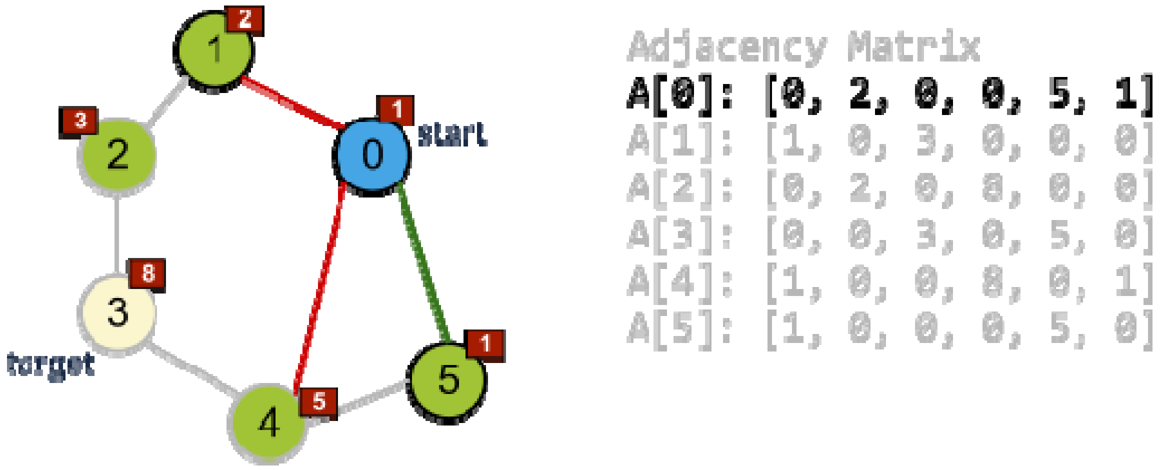


BFS graph and searching levels

11.3.7 广度优先搜索BFS

- BFS之所以称之为广度优先，是因为它自始至终都是从一个结点沿其广度方向向外扩展，通过已找到和未找到顶点之间的边界向外扩展。就是说，算法首先搜索和s距离为k的所有顶点，然后再去搜索和s距离为 k+1 的其他顶点。

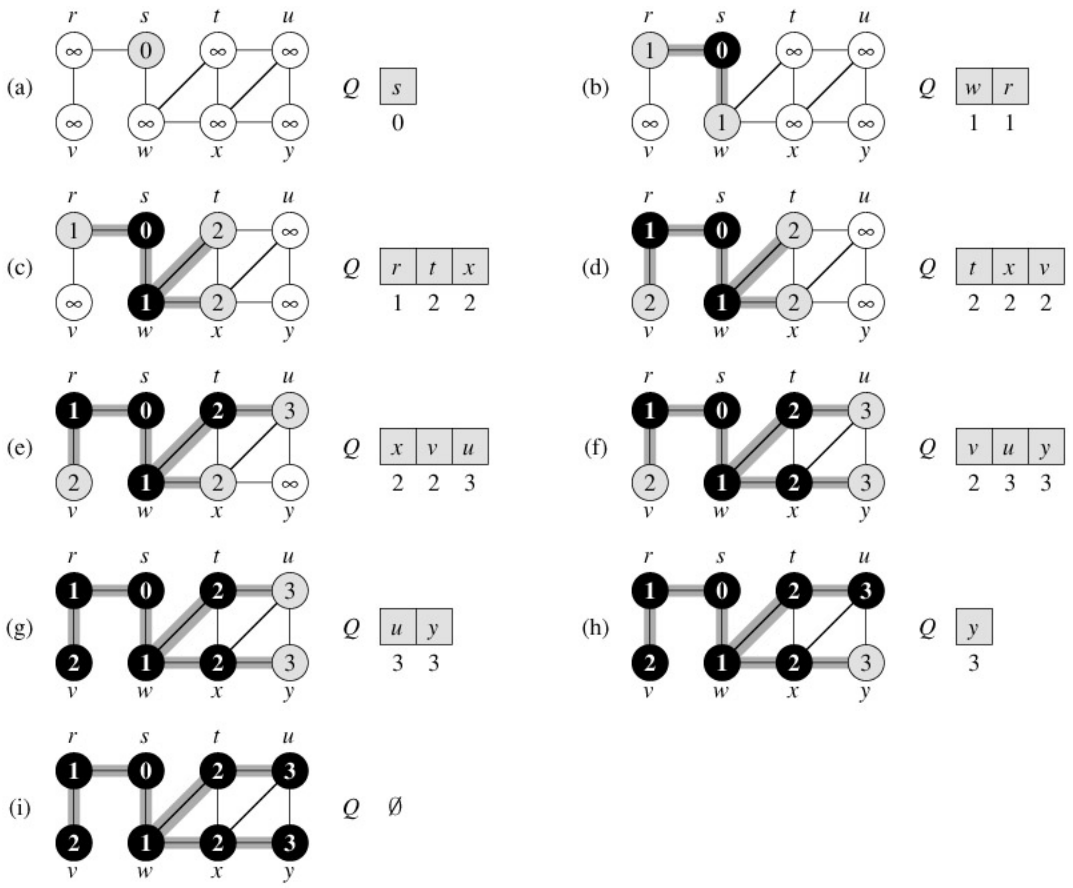


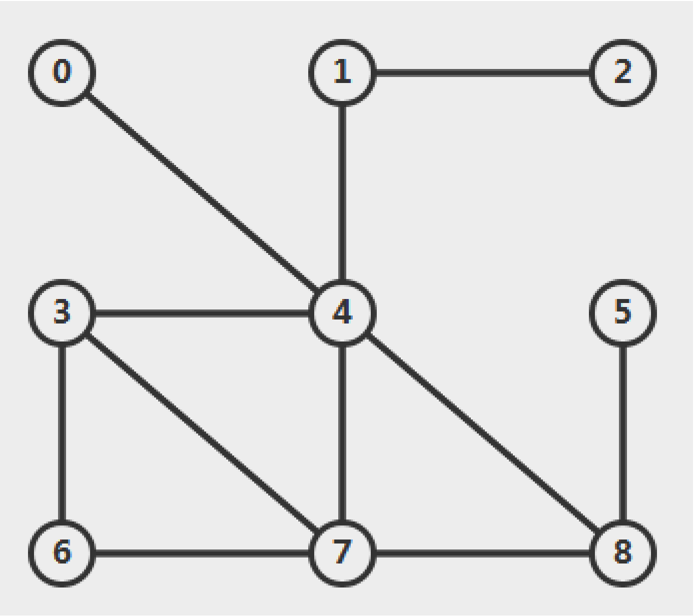


```
A[0]: [0, 2, 0, 0, 5, 1]
A[1]: [1, 0, 3, 0, 0, 0]
A[2]: [0, 2, 0, 8, 0, 0]
A[3]: [0, 0, 3, 0, 5, 0]
A[4]: [1, 0, 0, 8, 0, 1]
A[5]: [1, 0, 0, 0, 5, 0]
```

11.3.7 广度优先搜索BFS

▸1. BFS算法原理





11.3.7 广度优先搜索BFS

- 为了保持搜索的轨迹，广度优先搜索为每个顶点着色：白色、灰色或黑色。算法开始前所有顶点都是白色（表示未访问过），随着搜索的进行，各顶点会逐渐变成灰色（表示已访问过），然后成为黑色（表示已访问过，且所有邻接的顶点均已访问过）。在搜索中第一次碰到一顶点时，我们说该顶点被发现，此时该顶点变为灰色。所有和黑色顶点邻接的顶点都已被发现。灰色顶点可以与一些白色顶点相邻接，它们代表着已找到和未找到顶点之间的边界。广度优先搜索算法使用先进先出队列以保证搜索以广度优先的方式执行。

- 2．BFS问题的特征
- 可以采用BFS搜索算法解决的问题的特点是：
  - （1）有一组具体的状态，状态是问题可能出现的每一种情况。全体状态所构成的状态空间是有限的，问题规模较小。
  - （2）在问题的求解过程中，可以从一个状态按照问题给定的条件，转变为另外的一个或几个状态。
  - （3）可以判断一个状态的合法性，并且有明确的一个或多个目标状态。
  - （4）所要解决的问题是：根据给定的初始状态找出目标状态，或根据给定的初始状态和结束状态，找出一条从初始状态到结束状态的路径。

11.3.7 广度优先搜索BFS

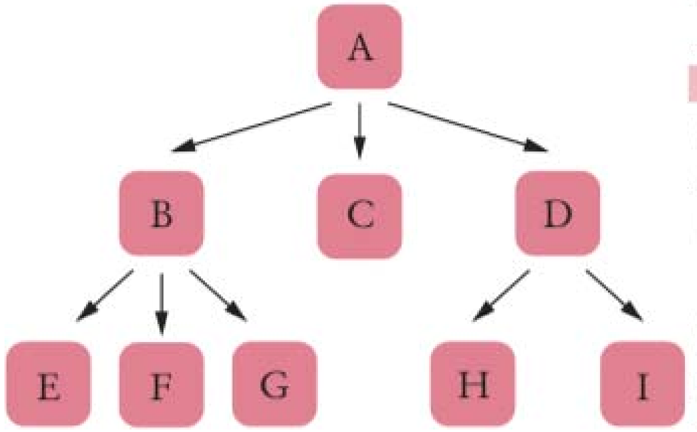
- 3．BFS求解问题的步骤
- （1）定义一个状态结点
- 使用BFS时，需要构造一个表明状态特征和不同状态之间关系的数据结构，称之为结点。不同的问题需要用不同的数据结构描述。

11.3.7 广度优先搜索BFS

- 3．BFS求解问题的步骤
- （2）确定结点的扩展规则
- 根据问题所给定的条件，从一个结点出发，可以生成一个或多个新的结点，这个过程通常称为扩展。结点之间的关系一般可以表示成一棵树，它被称为解答树。搜索算法的搜索过程实际上就是根据初始条件和扩展规则构造一棵解答树并寻找符合目标状态的结点的过程。

11.3.7 广度优先搜索BFS

- 3．BFS求解问题的步骤
- （3）搜索策略
- 为了便于进行搜索，要设置一个表存储所有的结点。因为在广度优先搜索算法中，要满足先生成的结点先扩展的原则，所以存储结点的表一般设计成先进先出的队列。



```
Queue
A
B C D        AddchildrenofA
C D E F G    AddchildrenofB
D E F G
E F G H I    AddchildrenofD
F G H I
G H I
H I
I
```

11.3.7 广度优先搜索BFS

- 搜索的步骤一般是：
- ①从队列头取出一个结点，检查它按照扩展规则是否能够扩展，如果能则产生一个新结点。
- ②检查新生成的结点，看它是否已在队列中存在，如果新结点已经在队列中出现过，就放弃这个结点，然后回到①。否则，如果新结点未曾在队列中出现过，则将它加入到队列尾。
- ③检查新结点是否目标结点。如果新结点是目标结点，则搜索成功，程序结束；若新结点不是目标结点，则回到①，再从队列头取出结点进行扩展。

11.3.7 广度优先搜索BFS

- 3．BFS求解问题的步骤
- 最终可能产生两种结果：找到目标结点，或扩展完所有结点而没有找到目标结点。
- 如果目标结点存在于解答树的有限层上，广度优先搜索算法一定能保证找到一条通向它的最佳路径，因此广度优先搜索算法特别适用于只需求出最优解的问题。当问题需要给出解的路径，则要保存每个结点的来源，也就是它是从哪一个节点扩展来的。

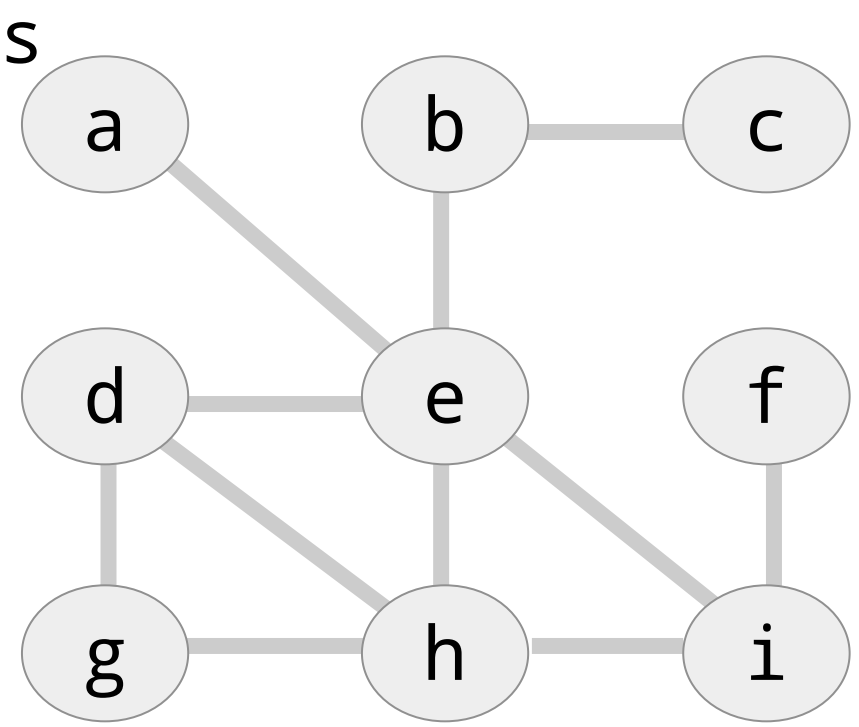
- 4．BFS算法性能分析
- （1）空间复杂度
- 因为所有节点都必须被储存，因此BFS的空间复杂度为 O(|V| + |E|)，其中|V|是节点数，|E|是边数。也可以为O(B^M)，其中B是最大分支系数，M是树的最长路径长度。由于对空间的大量需求，因此BFS并不适合解非常大的问题。
- （2）时间复杂度
- 最差情形下，BFS必须寻找所有到可能节点的所有路径，因此时间复杂度为O(|V| + |E|)。

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



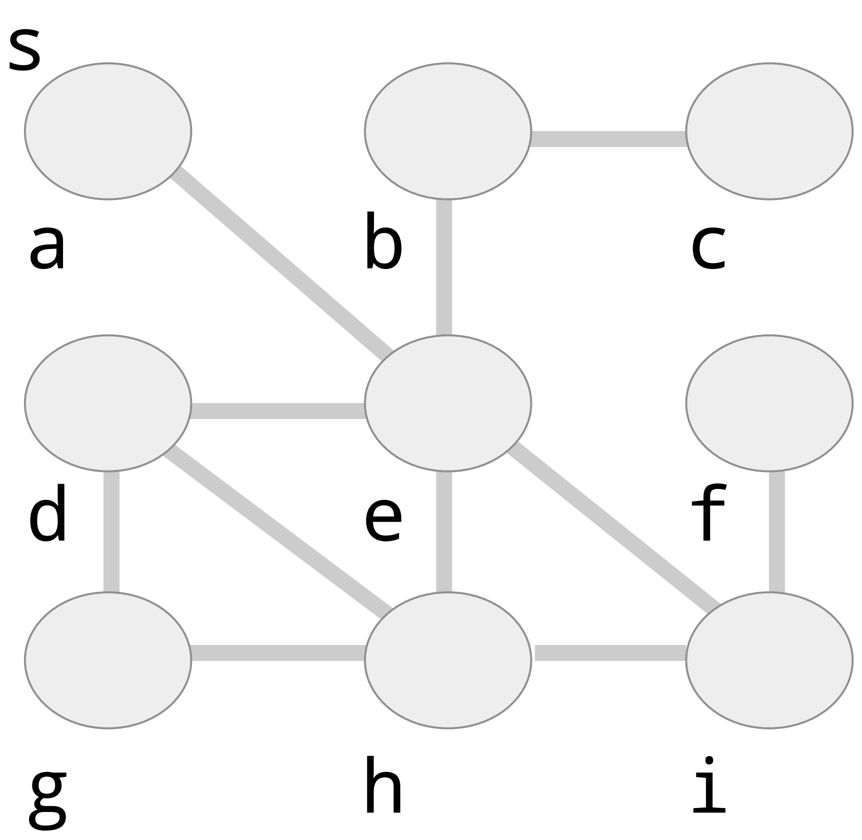
队列Q:

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

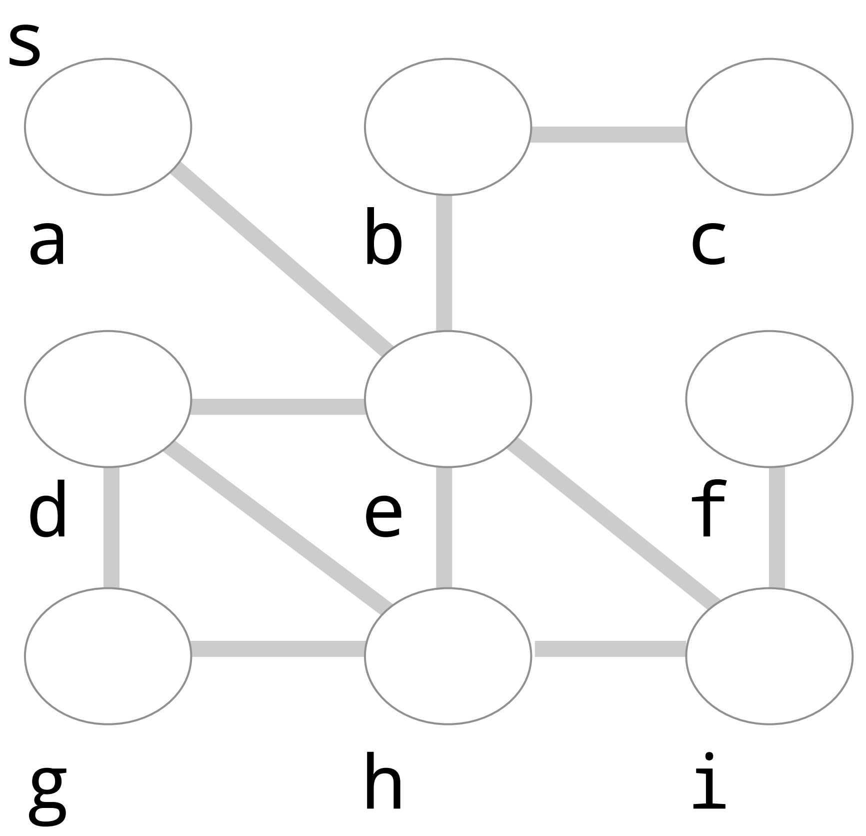
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

除源顶点s外，置每个顶点为白色

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

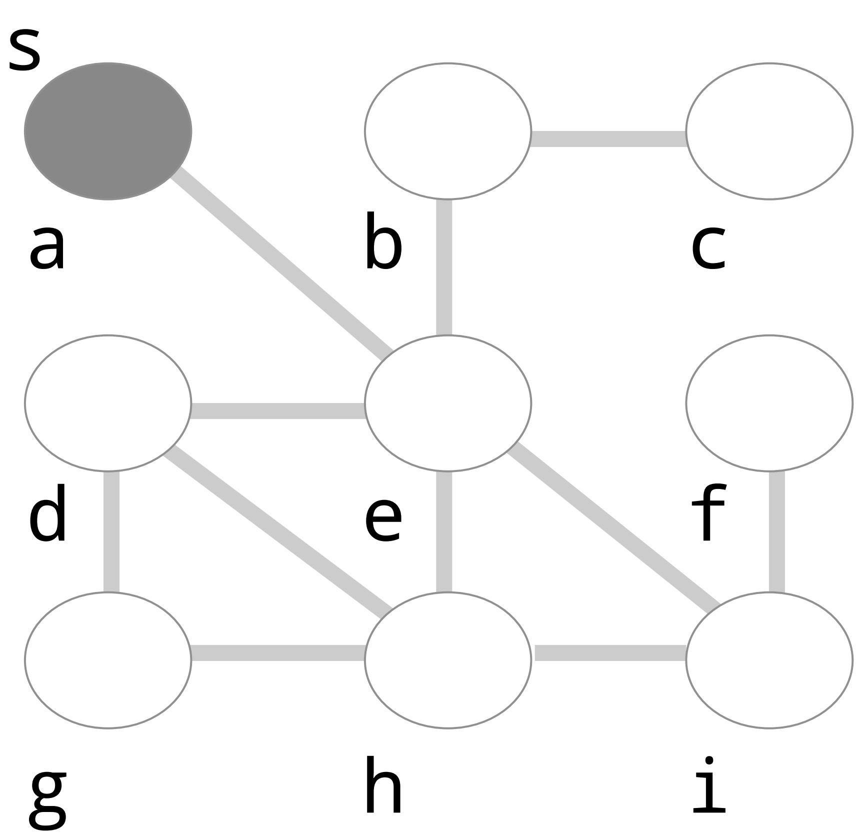
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

开始时，源顶点s已被发现，置为灰色

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

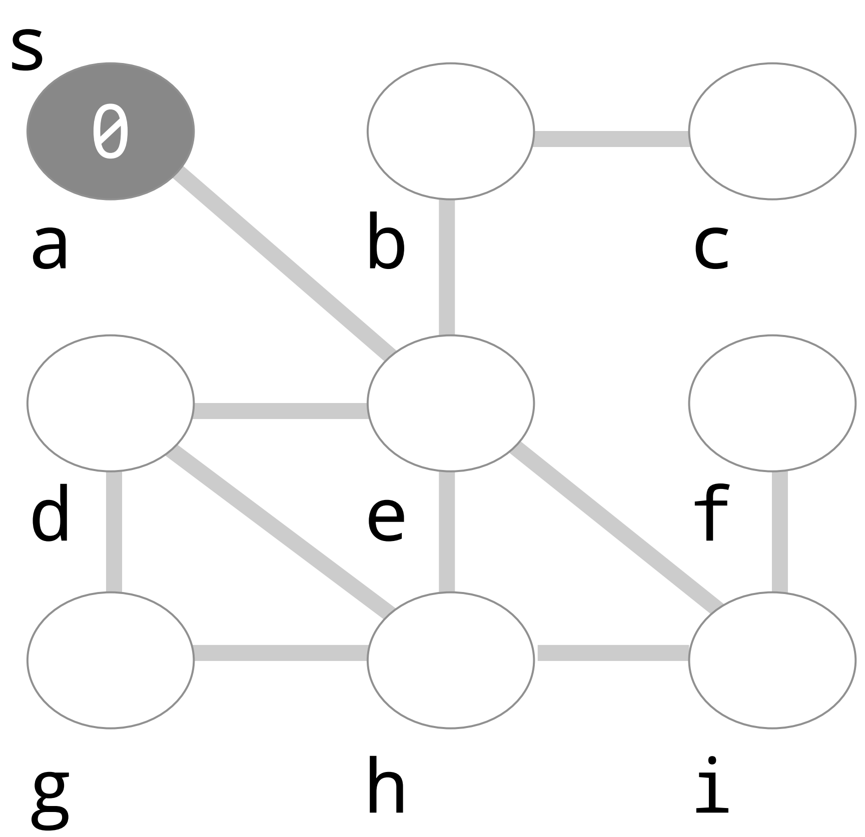
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

将s.d距离初始化为0

```
BFS(G,s):
  for each u ∈ G.V - {s}
    u.color = white
  s.color = gray
  s.d = 0
  Q = ∅
  Q.enqueue(s)
  while Q ≠ ∅
    u = Q.dequeue()
    for each v ∈ G.Adj[u]
      if v.color == white
        v.color = gray
        v.d = u.d + 1
        v.pred = u
        Q.enqueue(v)
    u.color = black
```

图G:



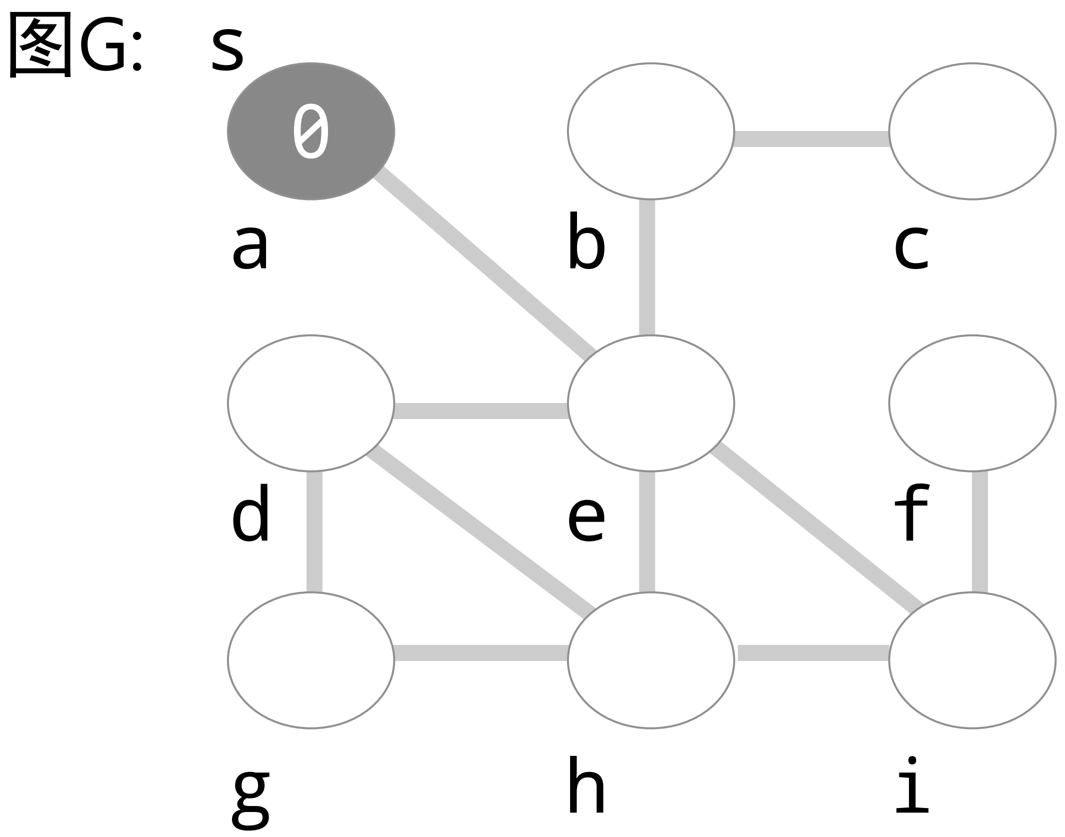
队列Q:

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

初始化队列Q，使其仅含源顶点s(a)

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: a

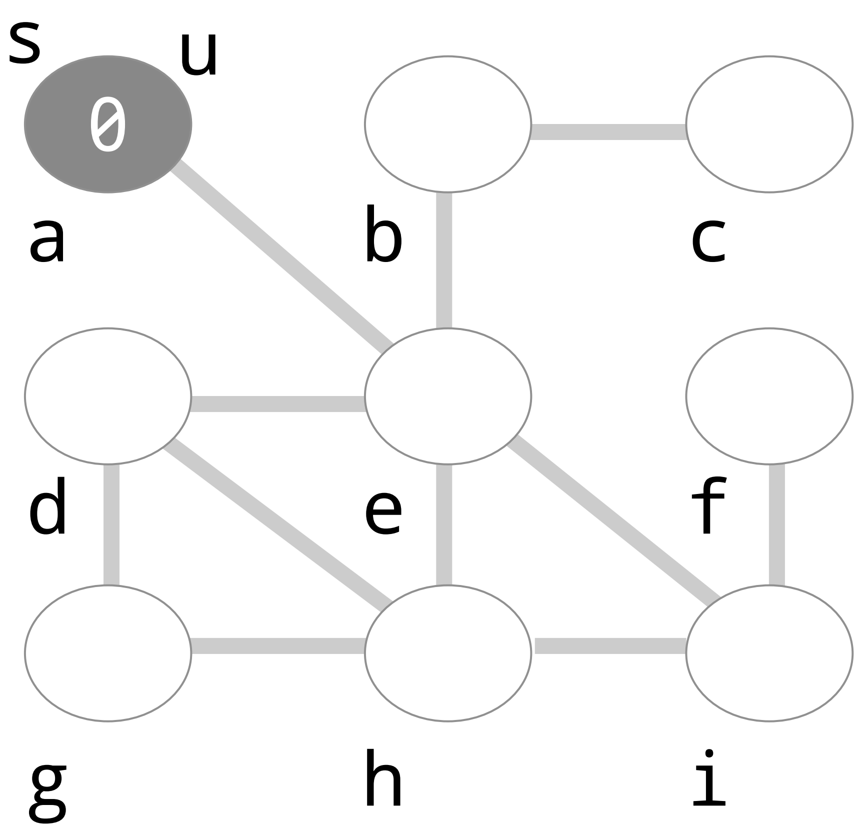
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(a)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

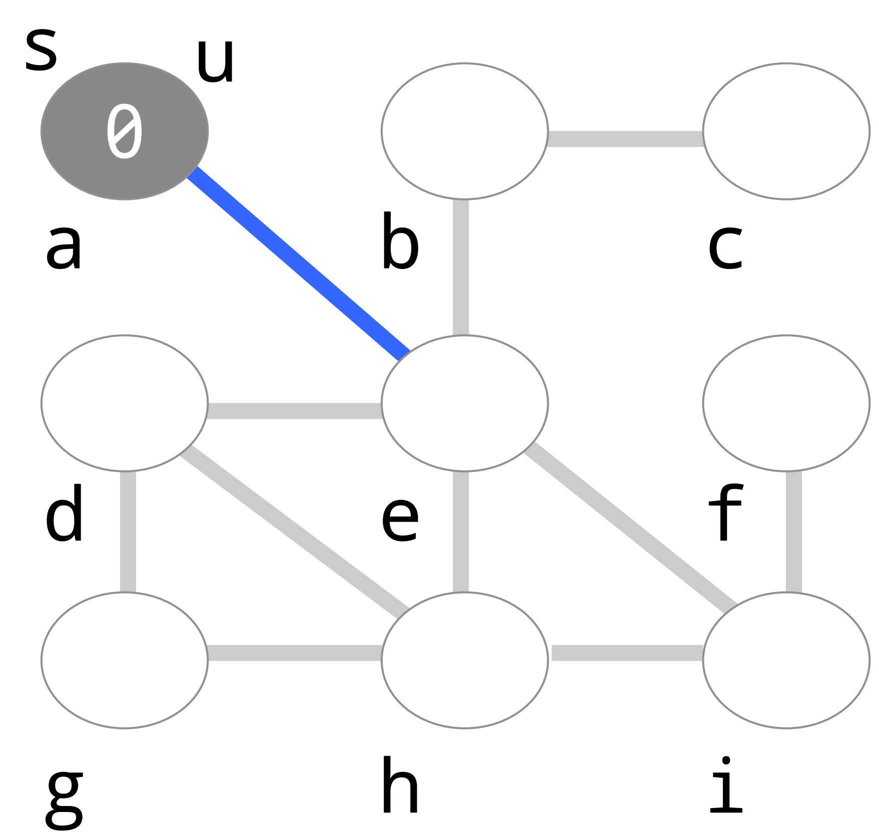
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
  for each u ∈ G.V - {s}
    u.color = white
  s.color = gray
  s.d = 0
  Q = ∅
  Q.enqueue(s)
  while Q ≠ ∅
    u = Q.dequeue()
    for each v ∈ G.Adj[u]
      if v.color == white
        v.color = gray
        v.d = u.d + 1
        v.pred = u
        Q.enqueue(v)
    u.color = black
```

图G:



队列Q:

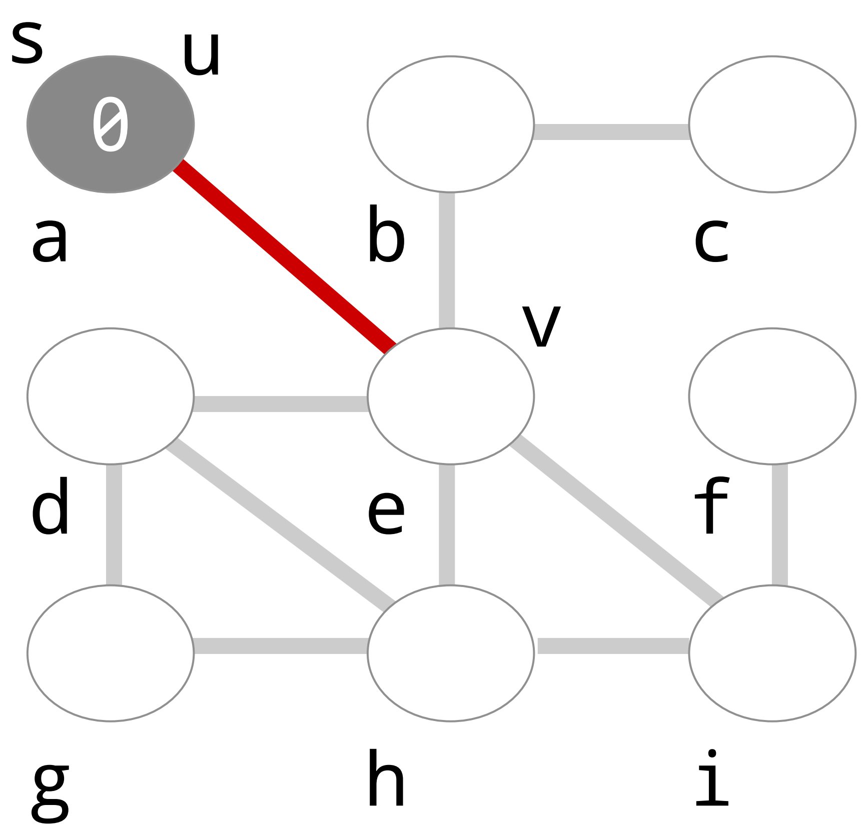
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

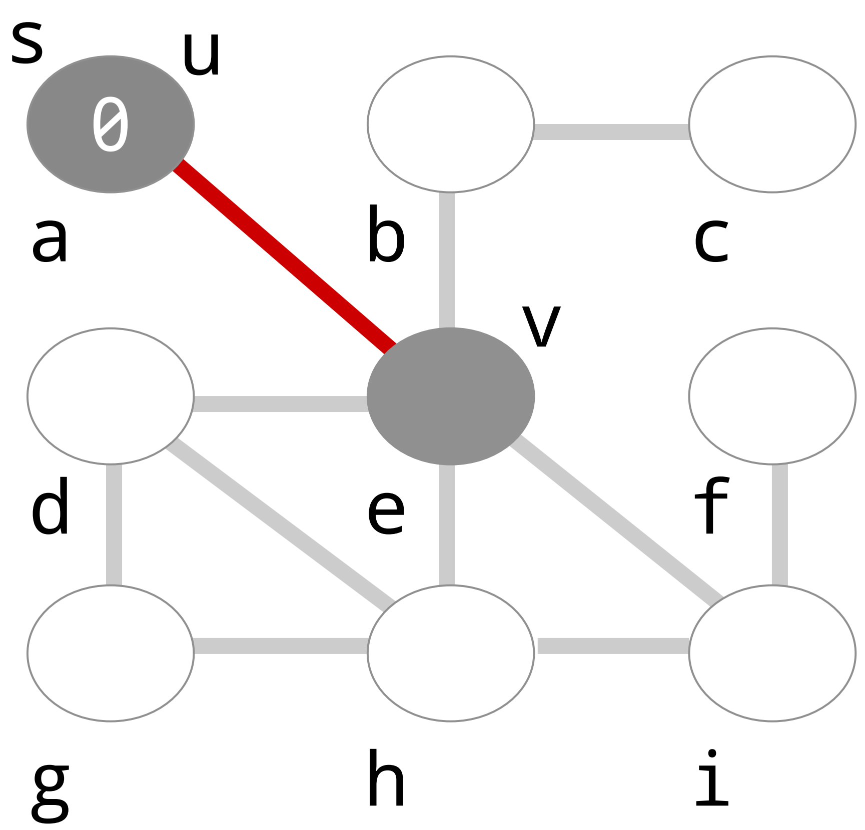
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

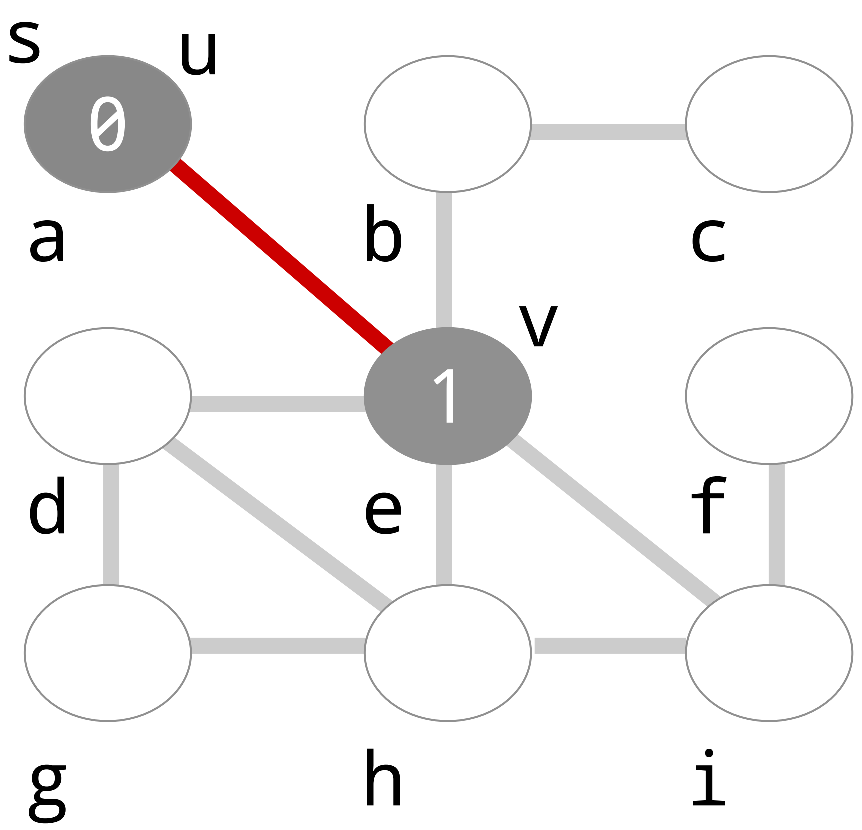
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

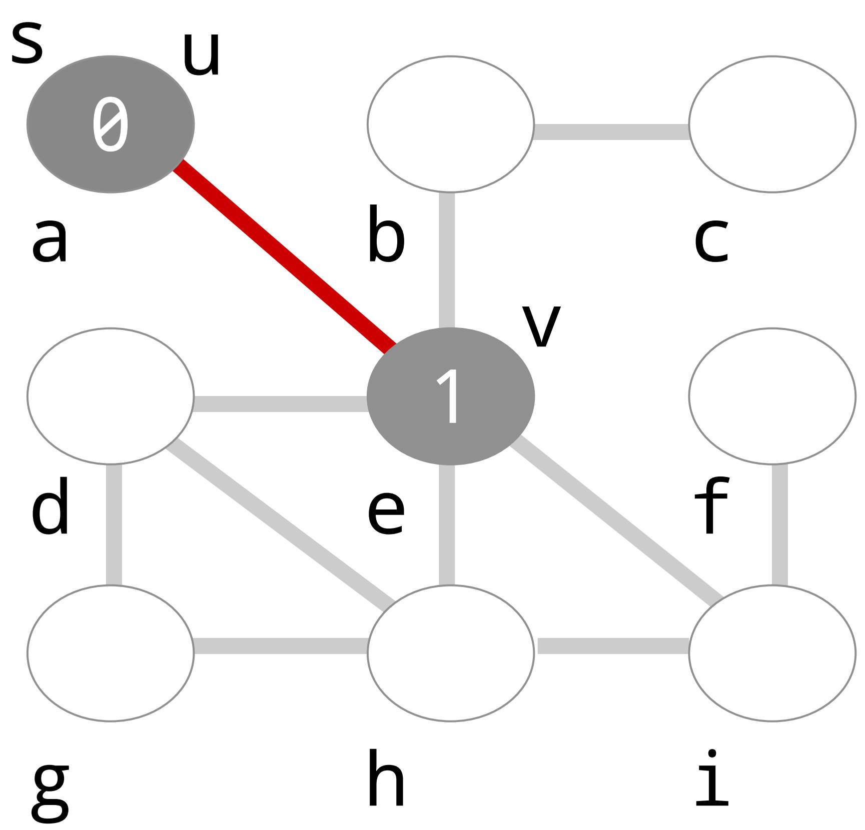
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

记录v的前向结点是u

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



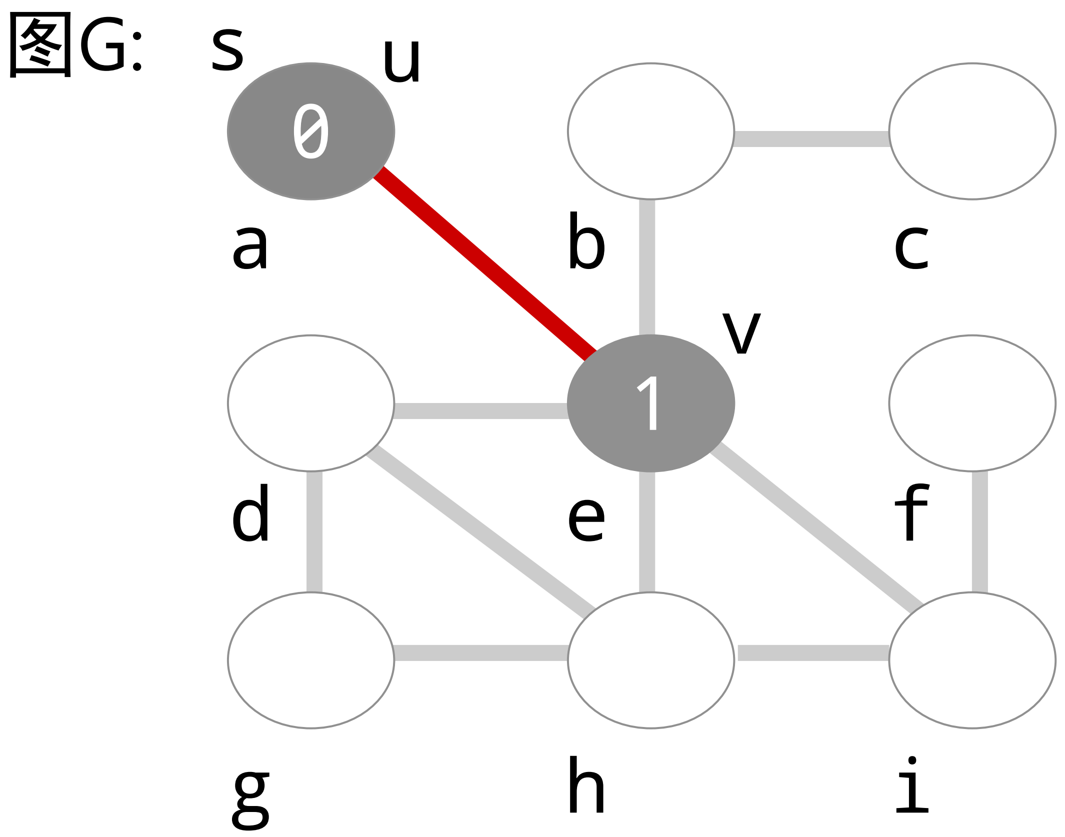
队列Q:

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为顶点v的父母，v(e)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



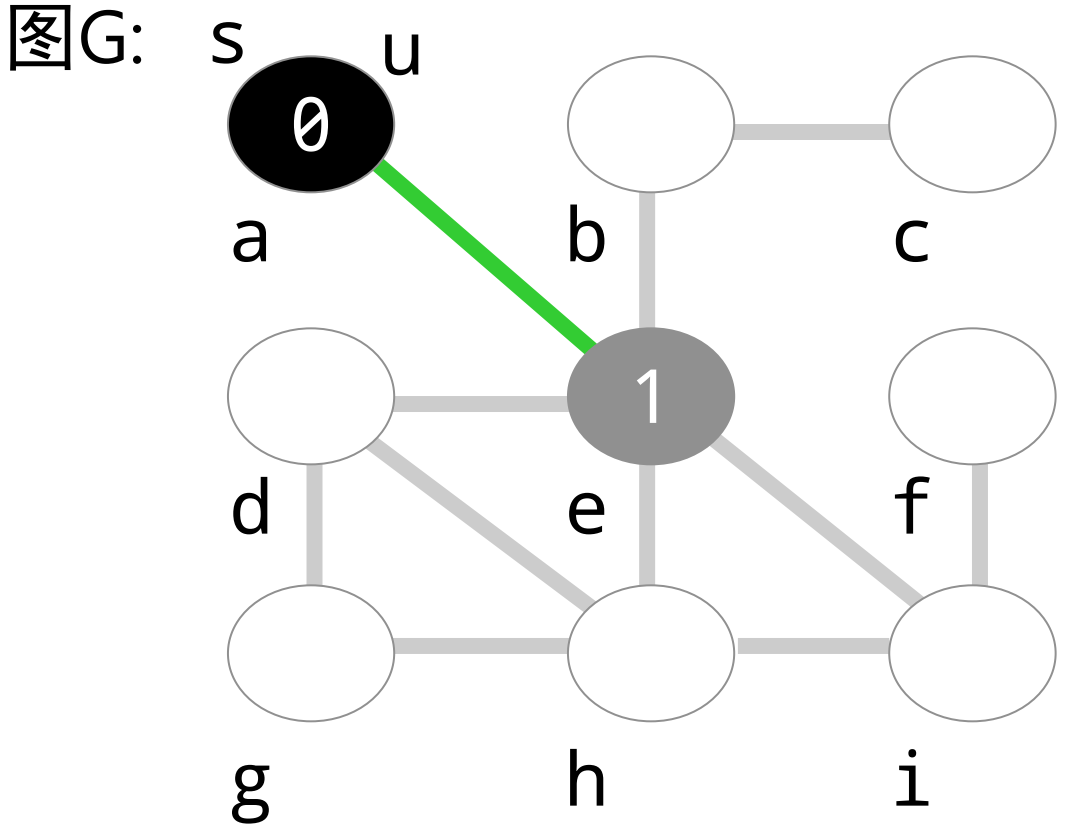
队列Q: e

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: e

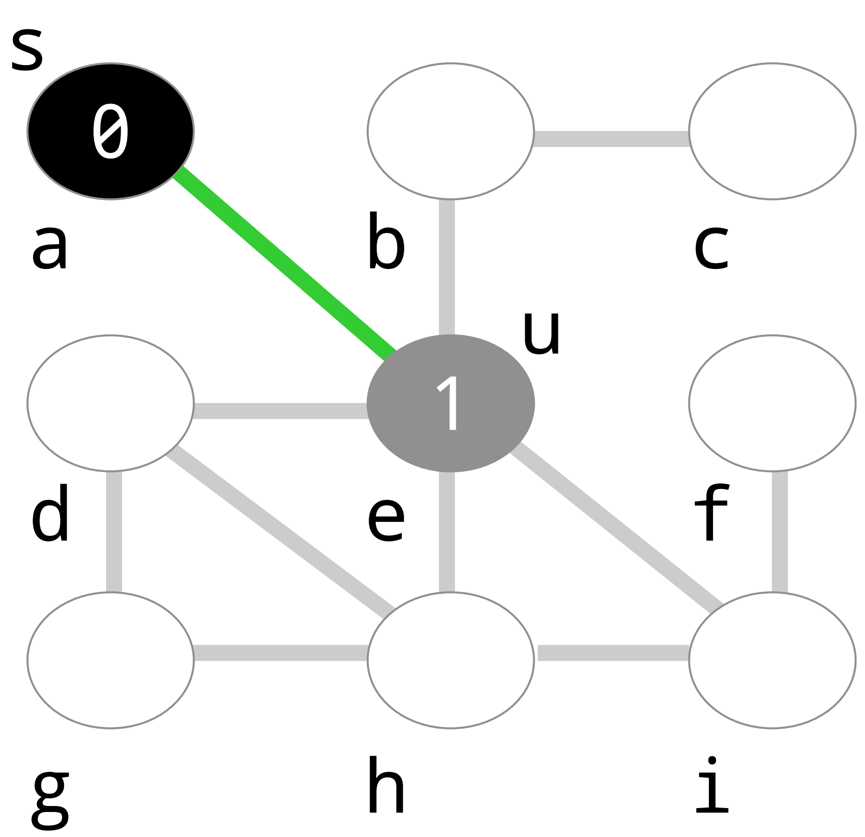
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(e)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

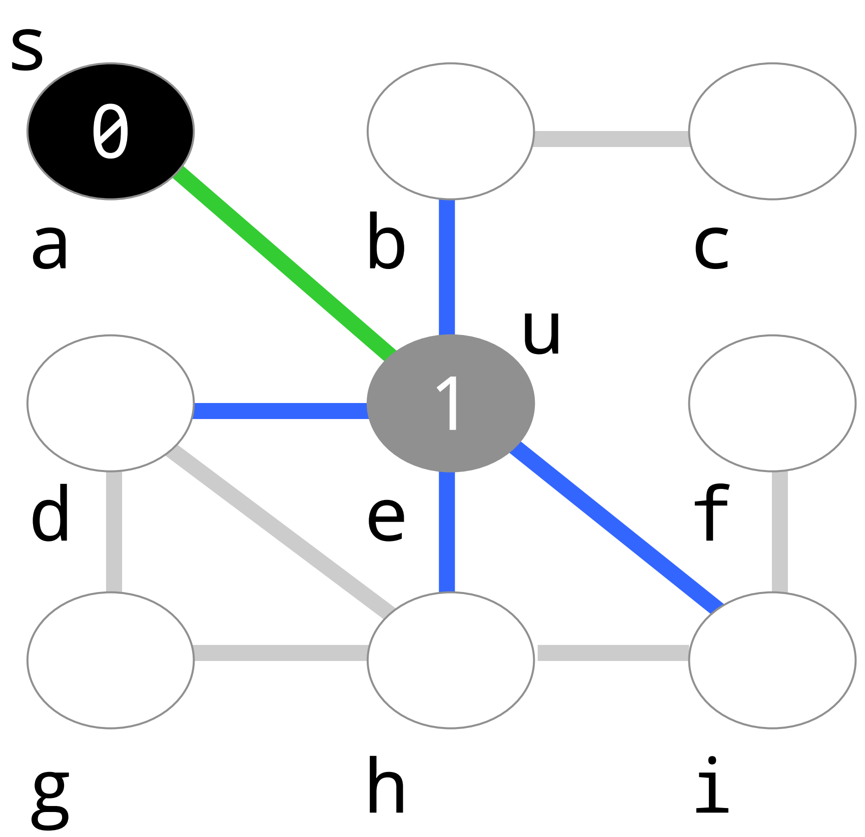
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

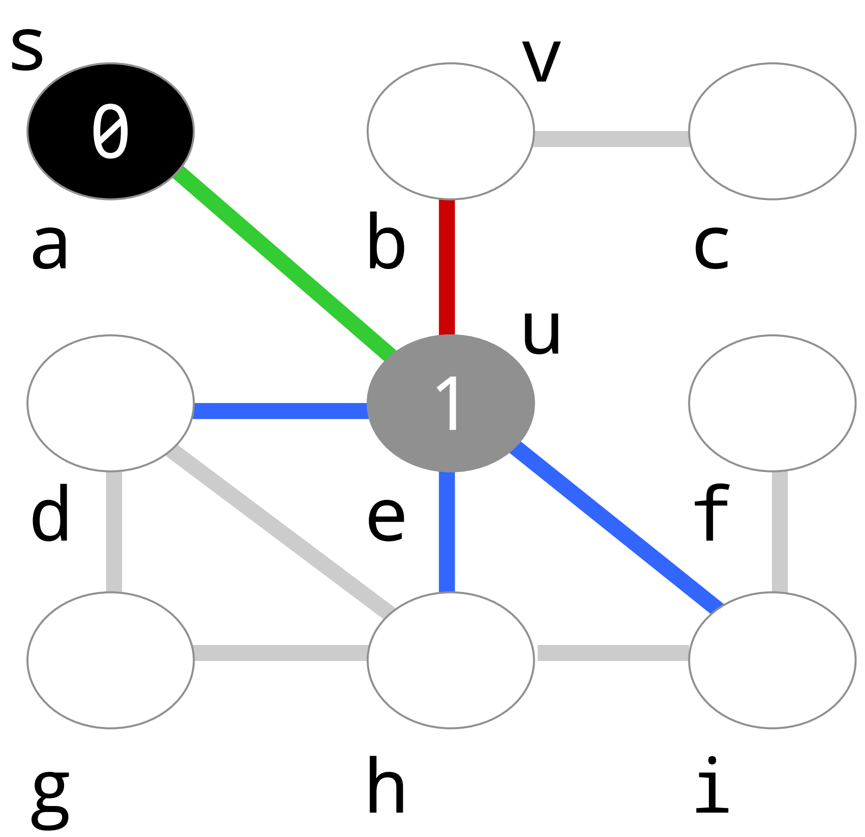
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

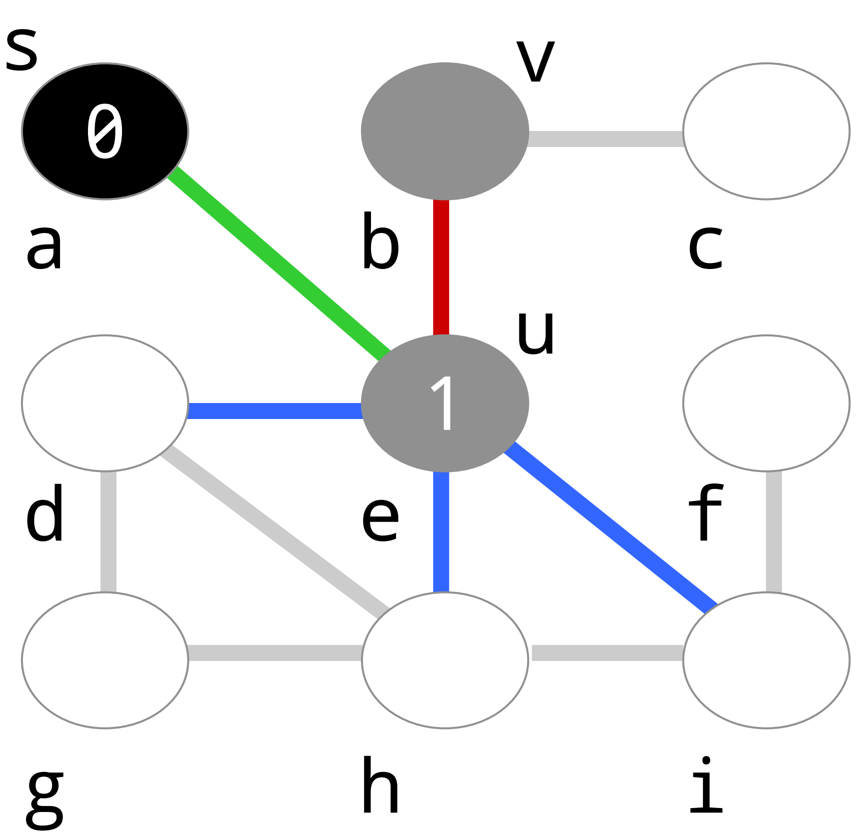
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

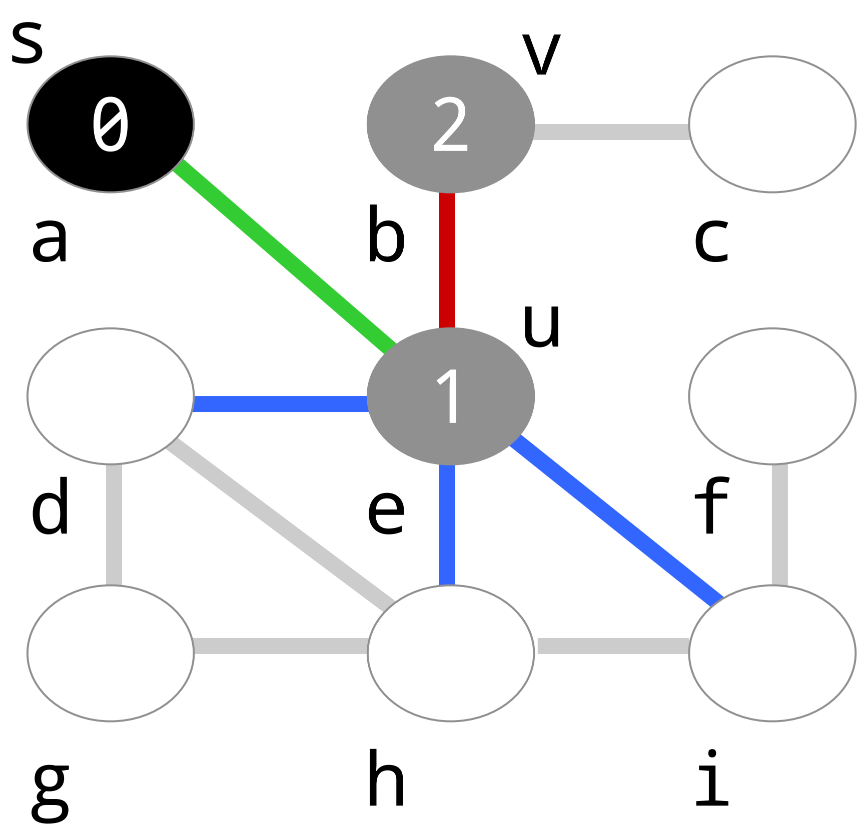
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

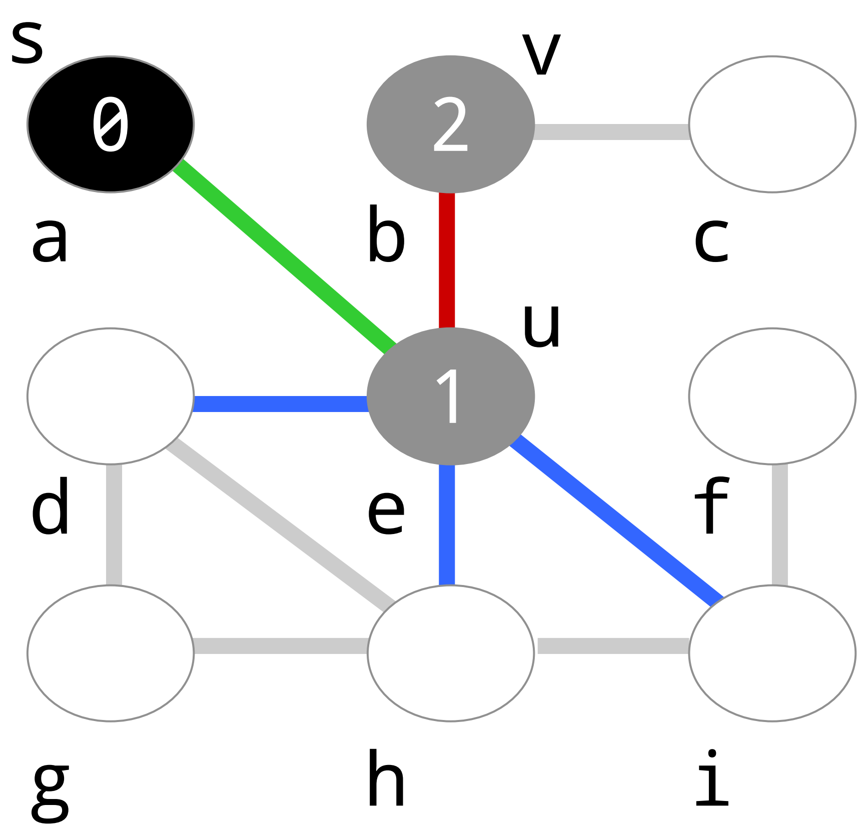
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(b)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



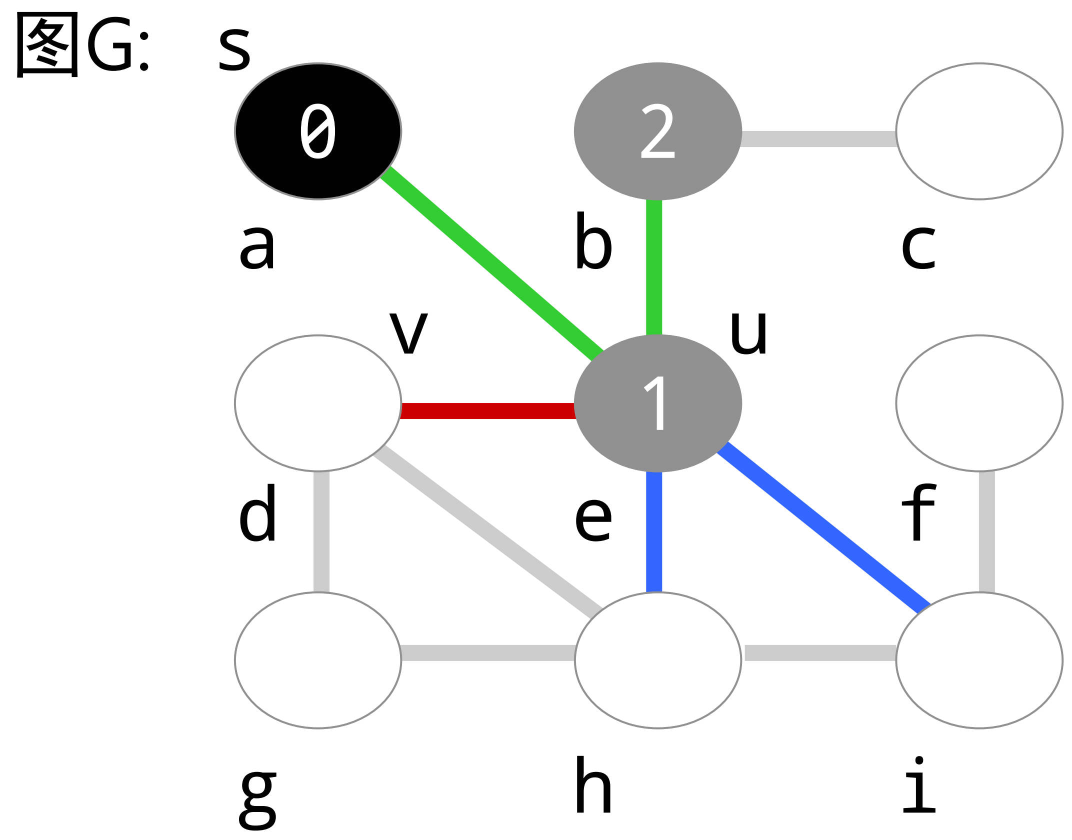
队列Q: b

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



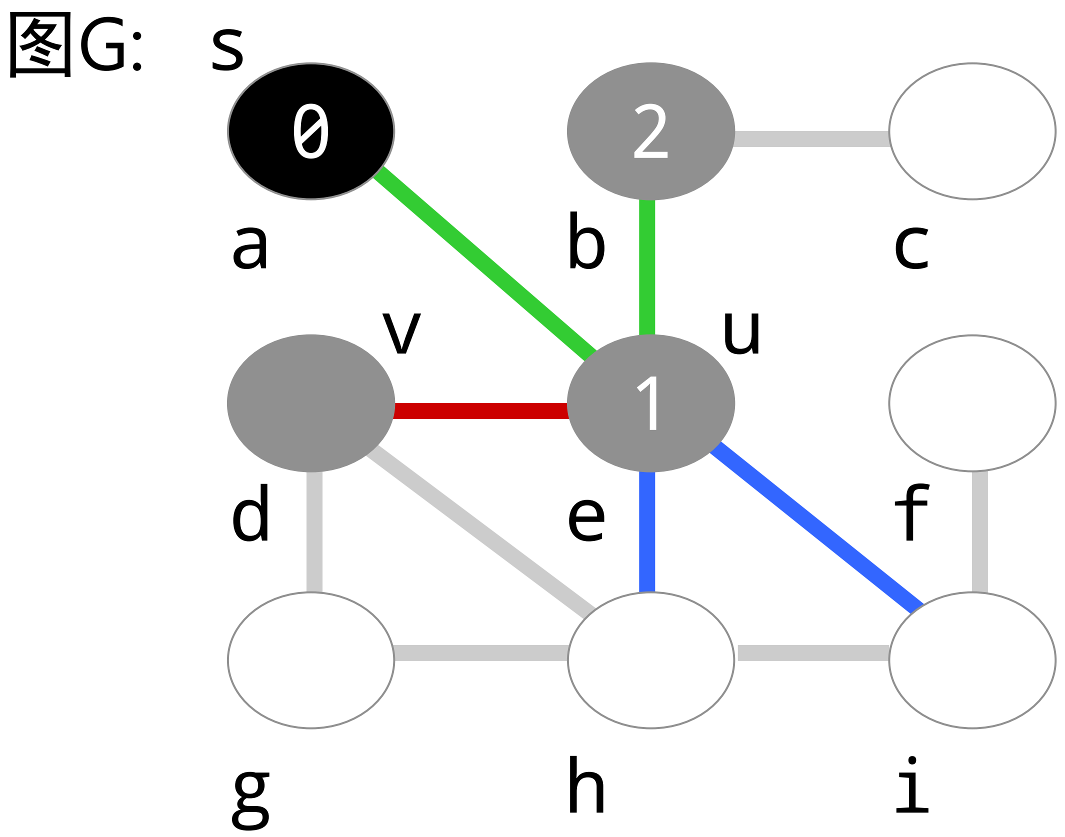
队列Q: b

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



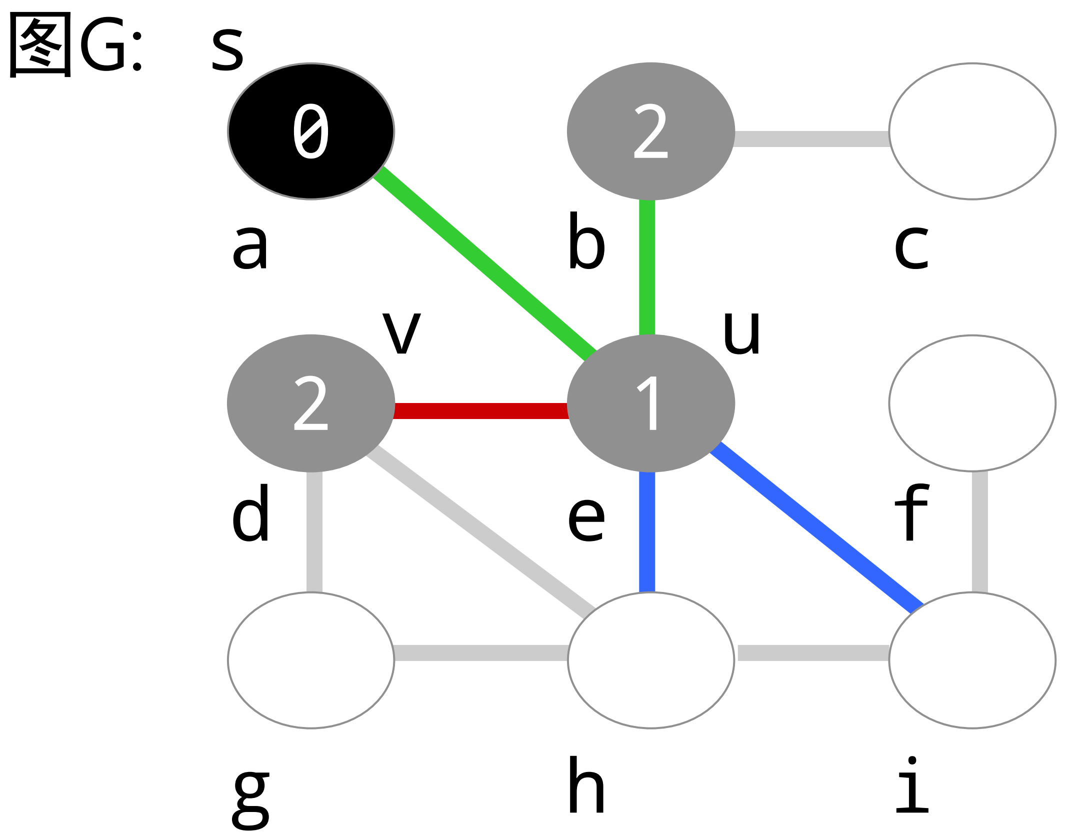
队列Q: b

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: b

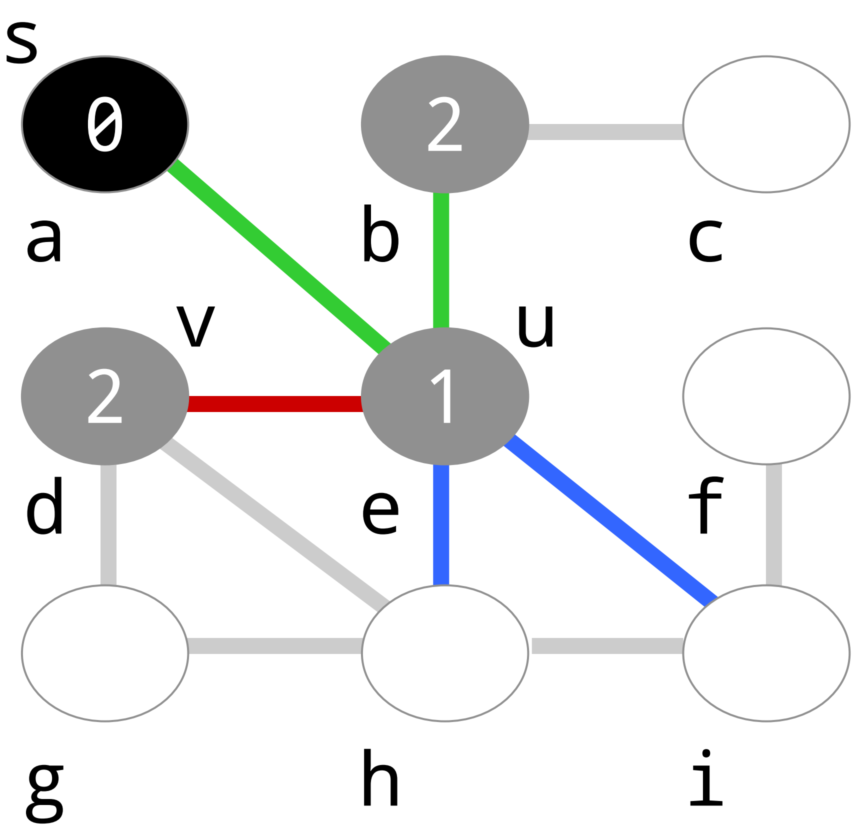
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(d)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d |

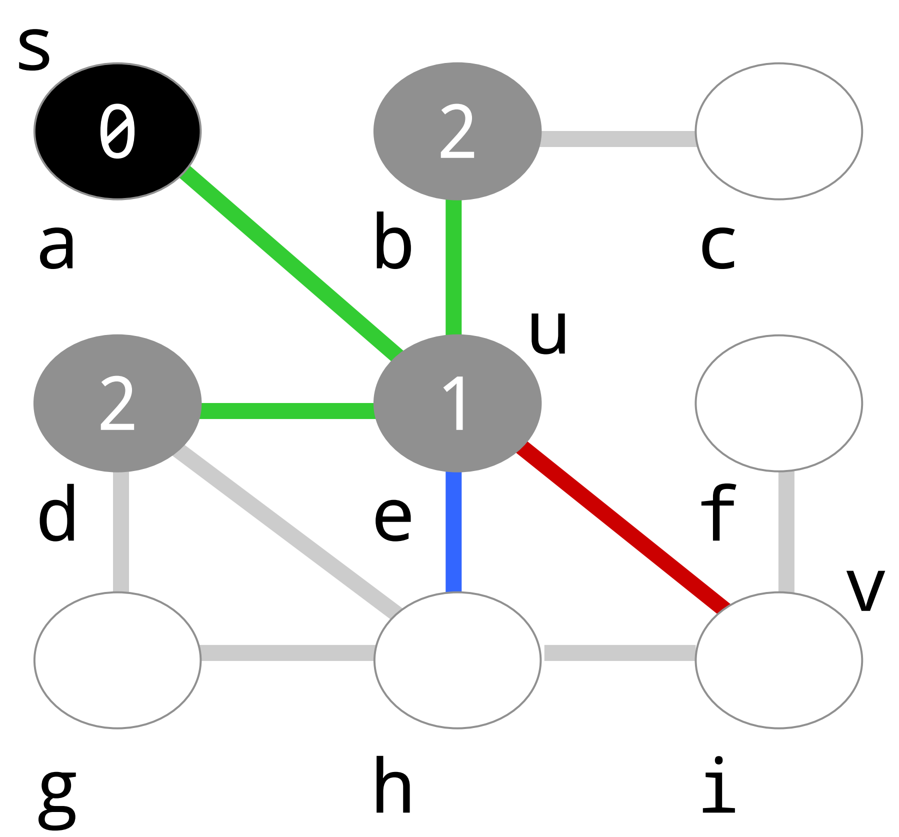
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d |

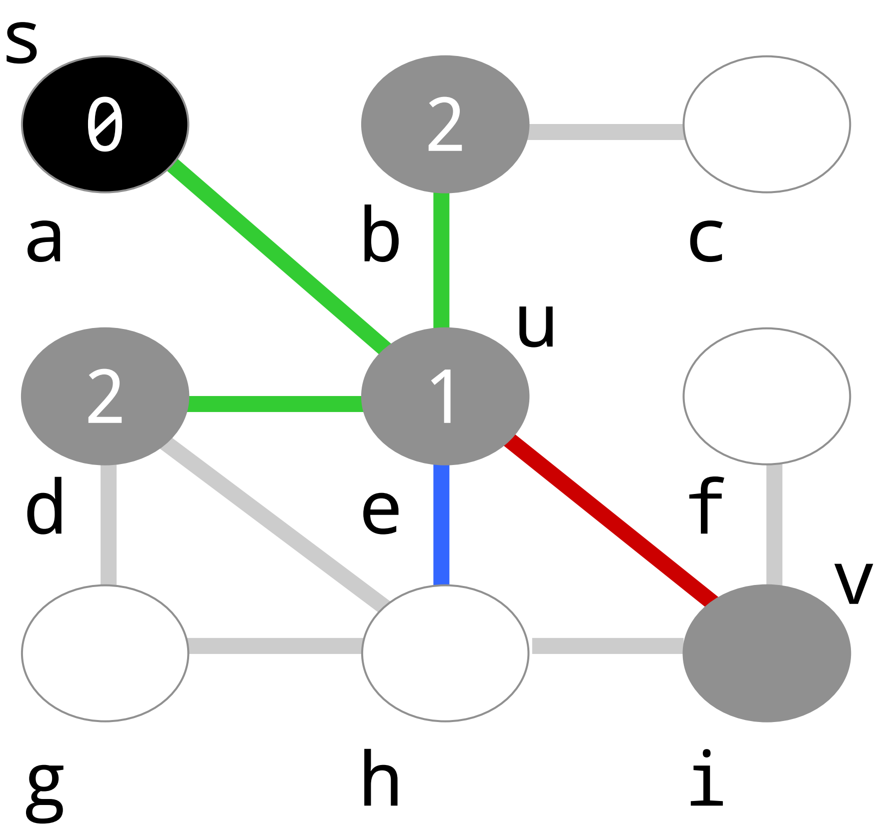
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d |

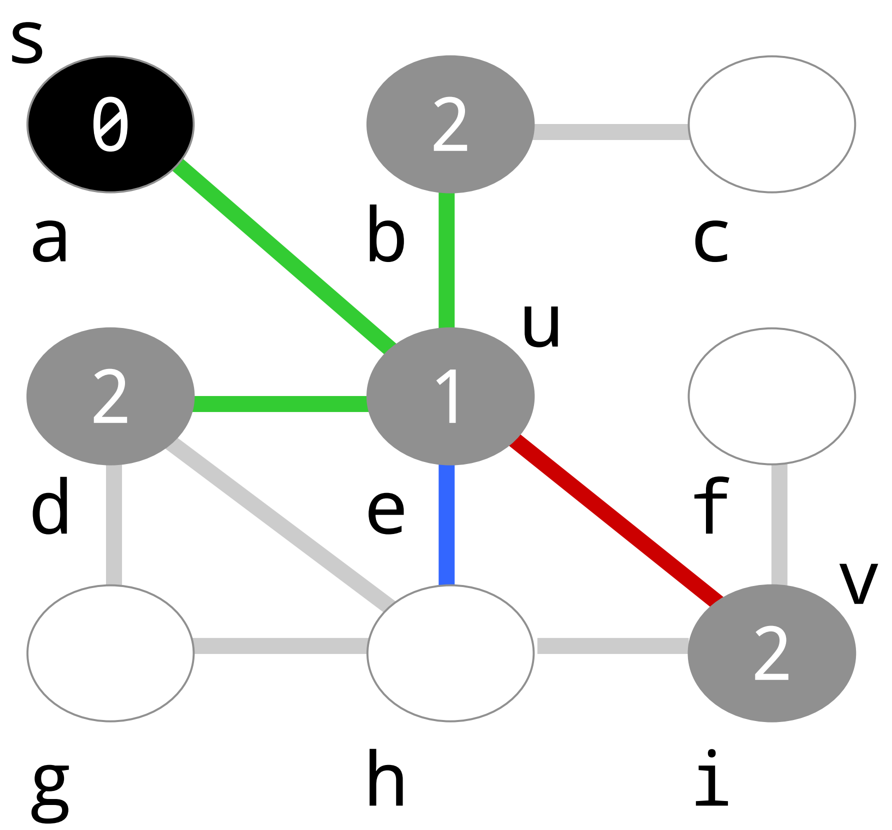
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d |

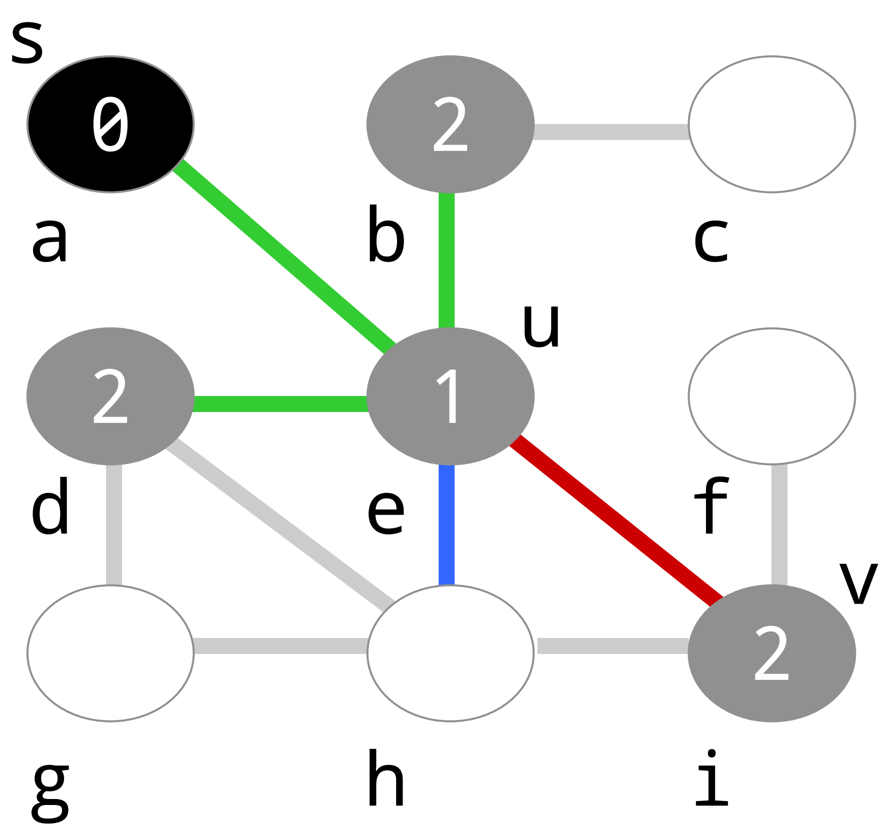
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(i)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: b d i

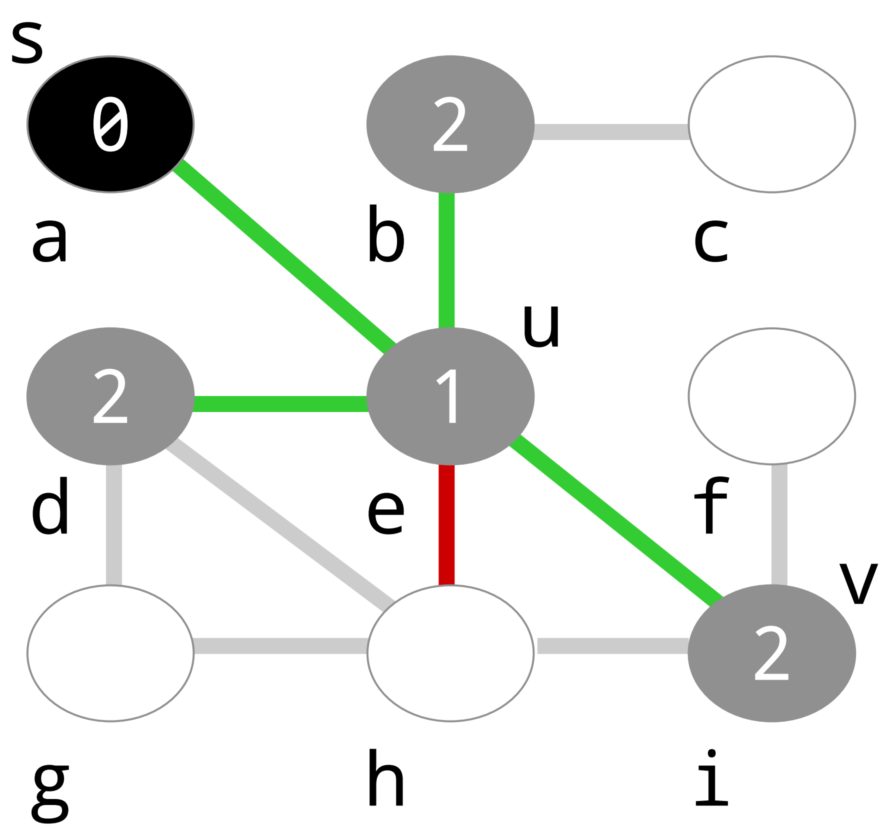
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



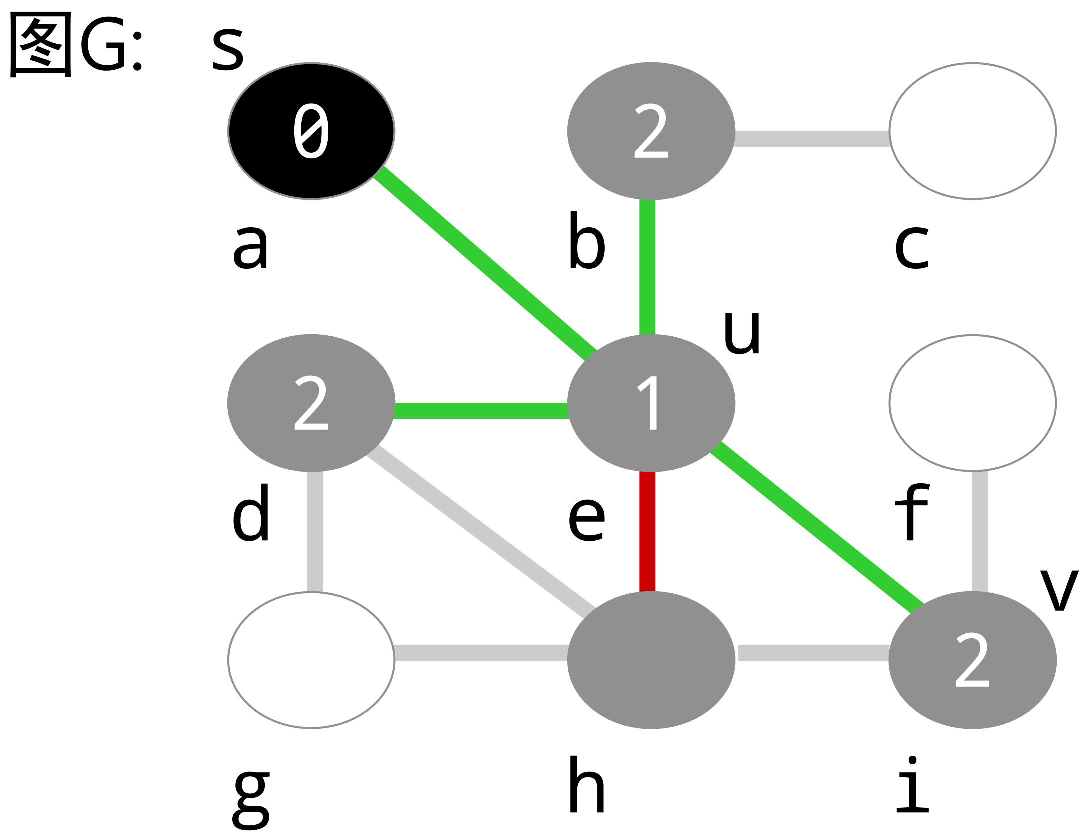
队列Q: b | d | i

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: | b | d | i |

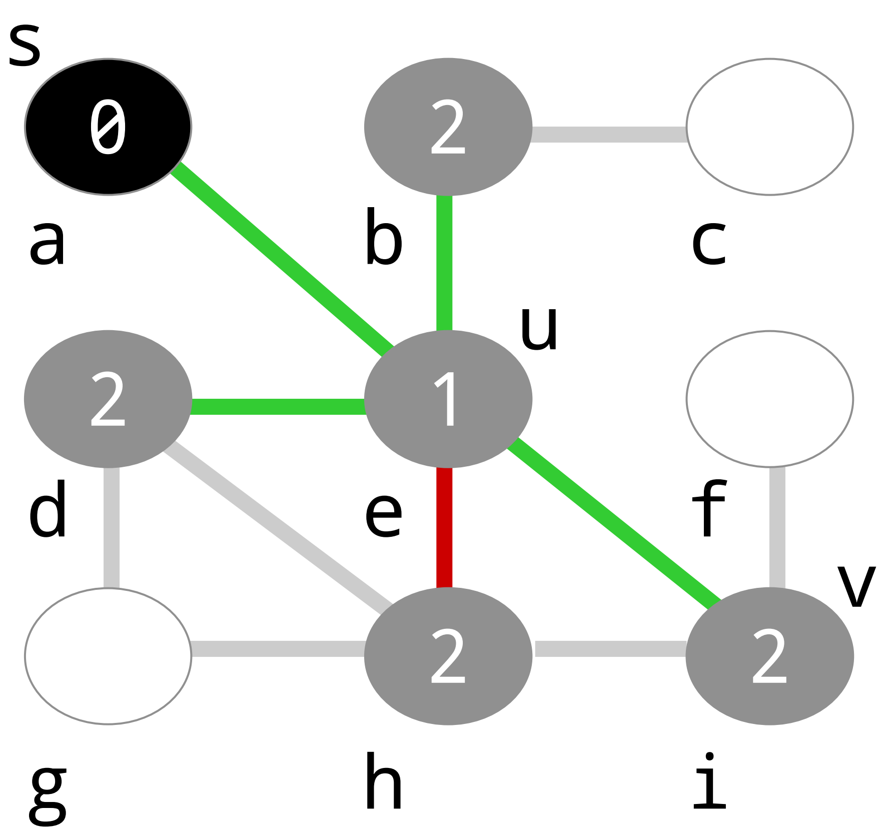
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d | i |

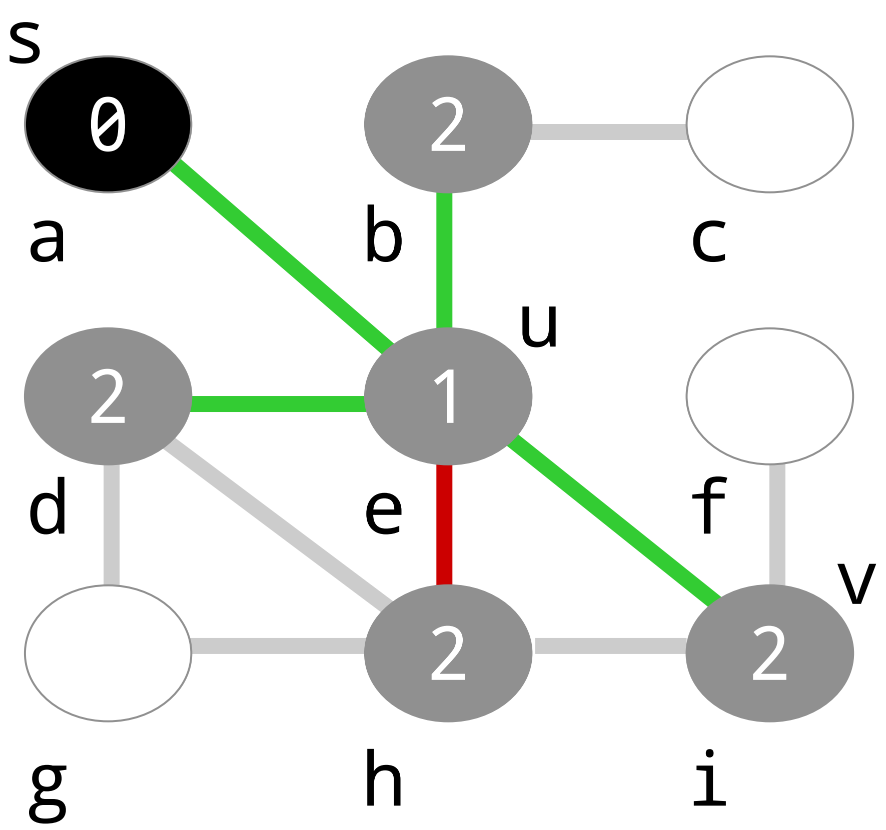
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(h)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: b | d | i | h

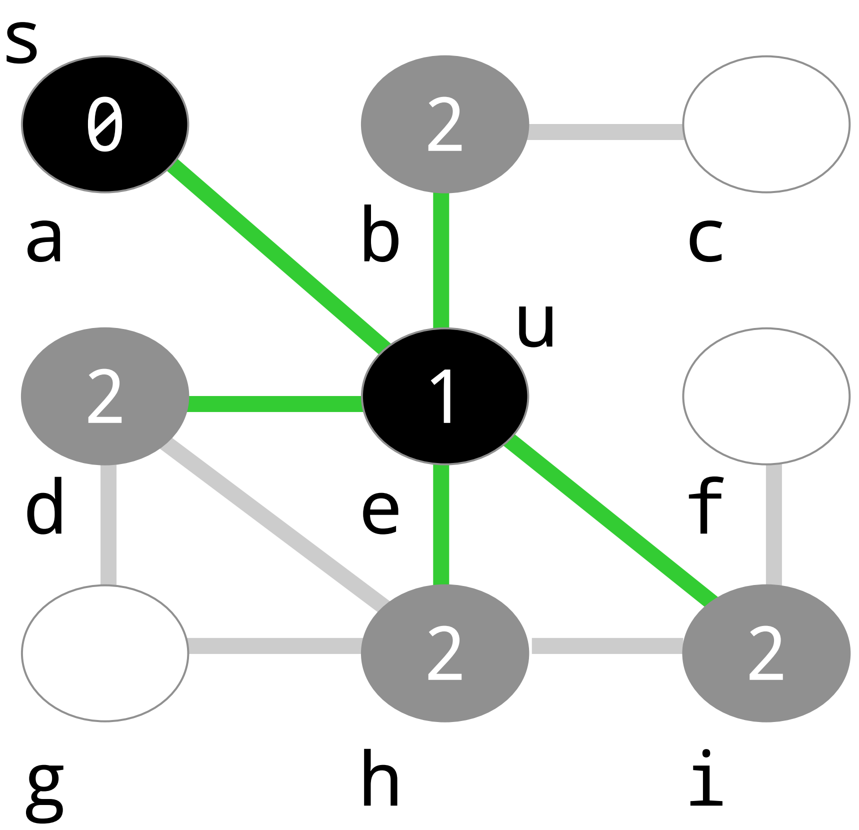
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | b | d | i | h |

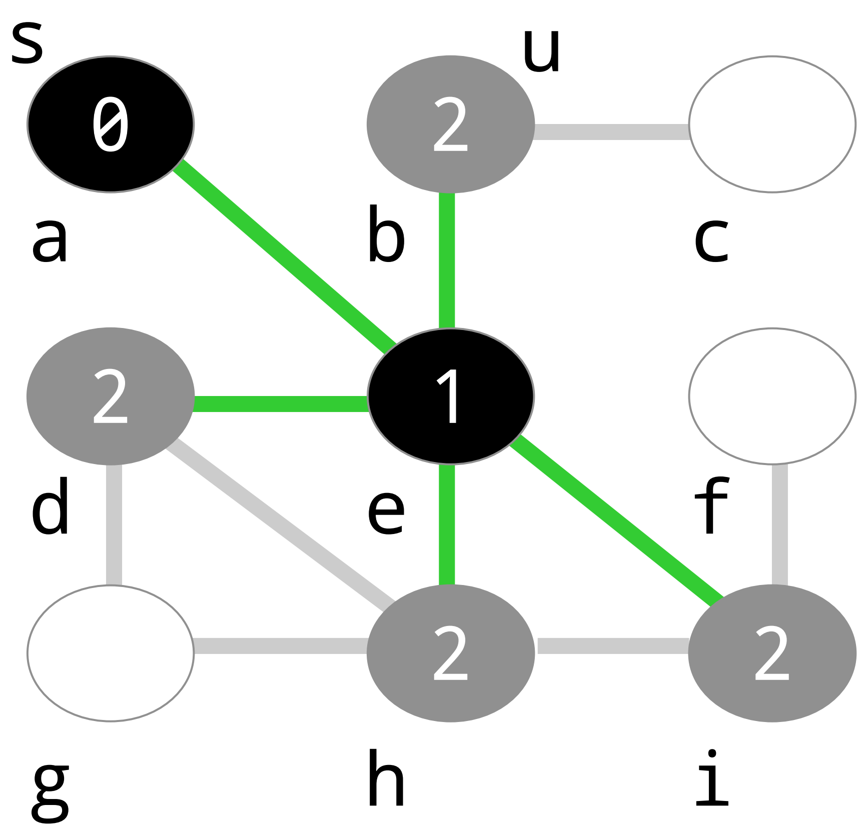
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(b)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | d | i | h |

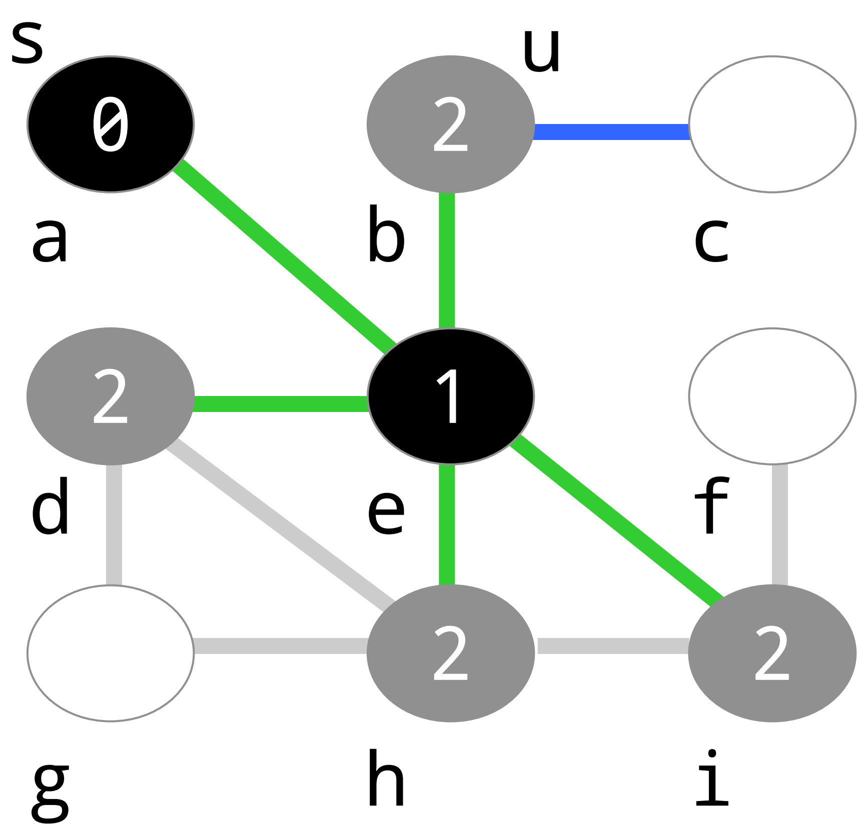
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: d i h

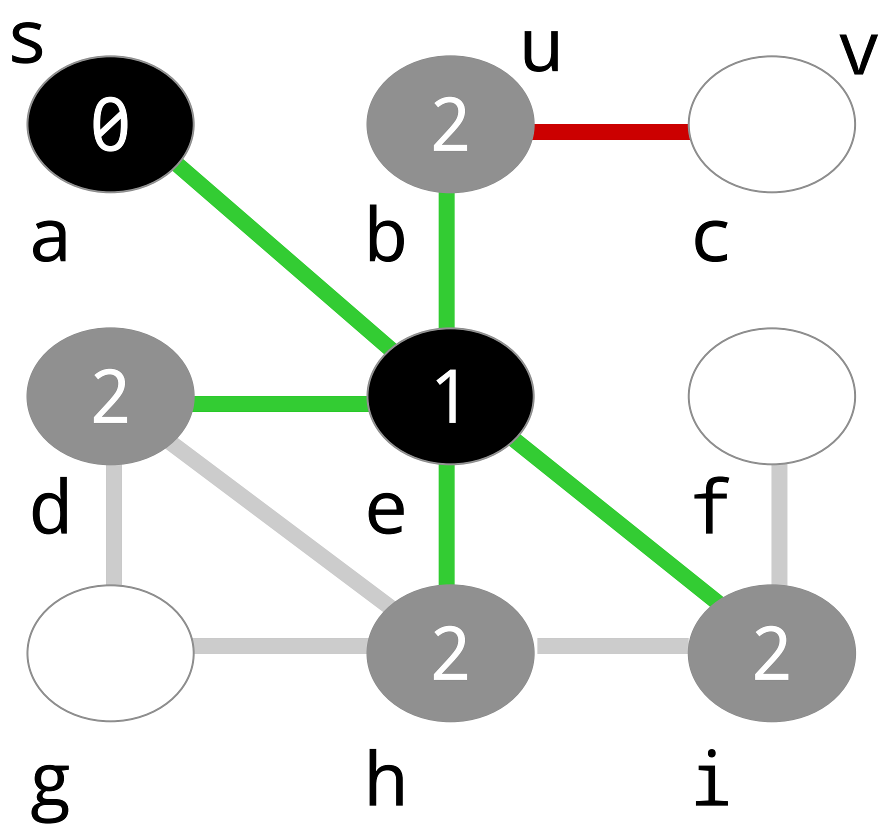
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



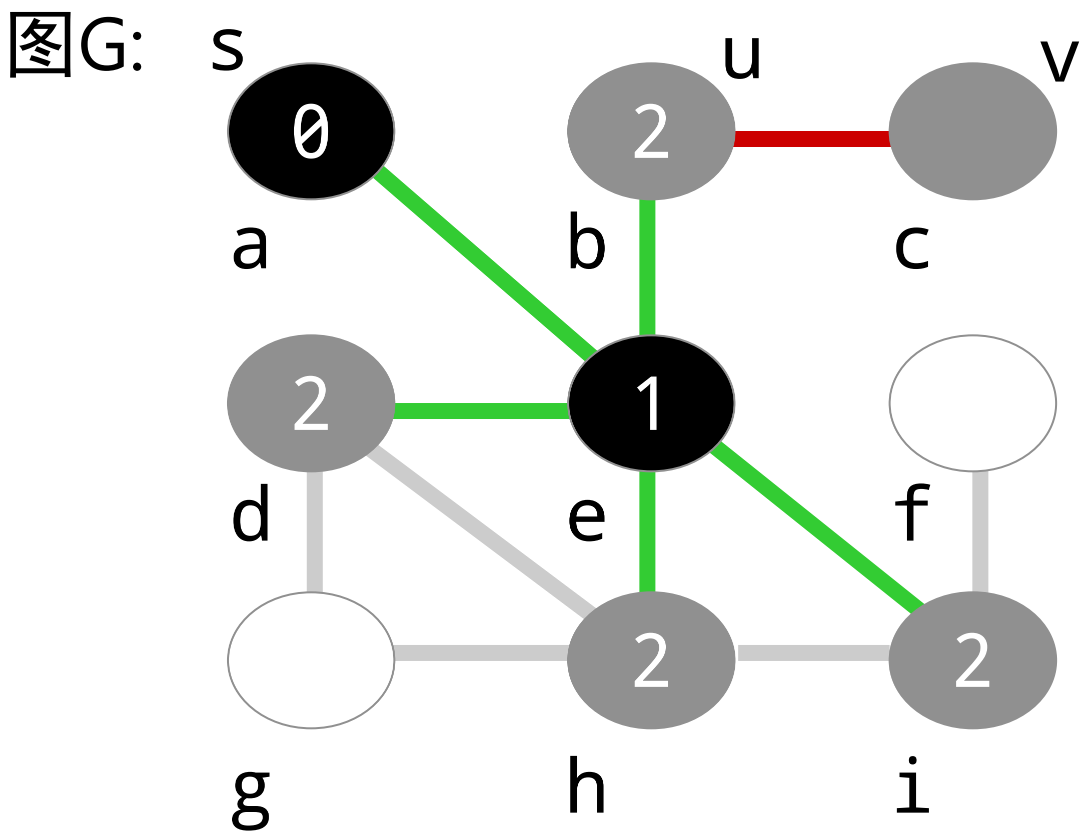
队列Q: | d | i | h |

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: d i h

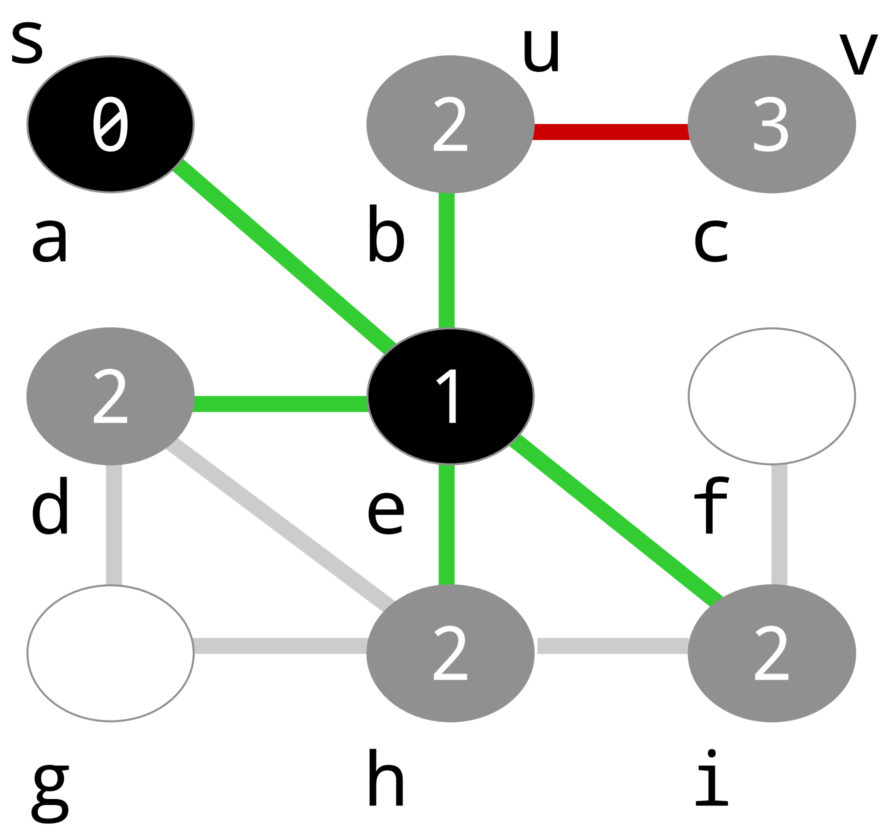
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

d	i	h

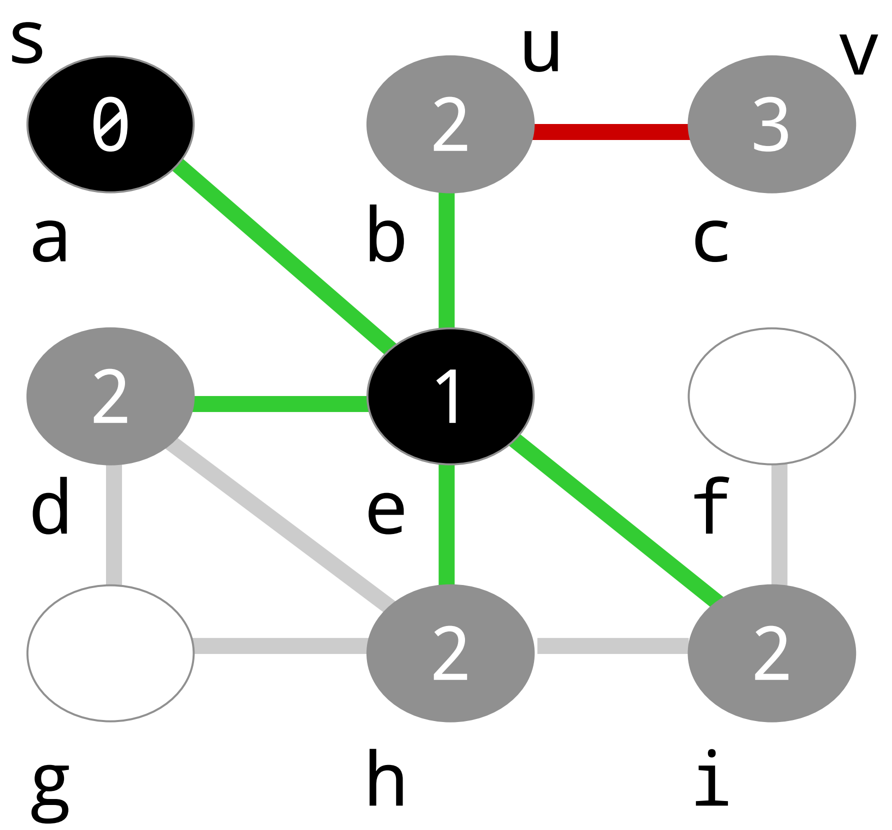
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(c)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | d | i | h | c |

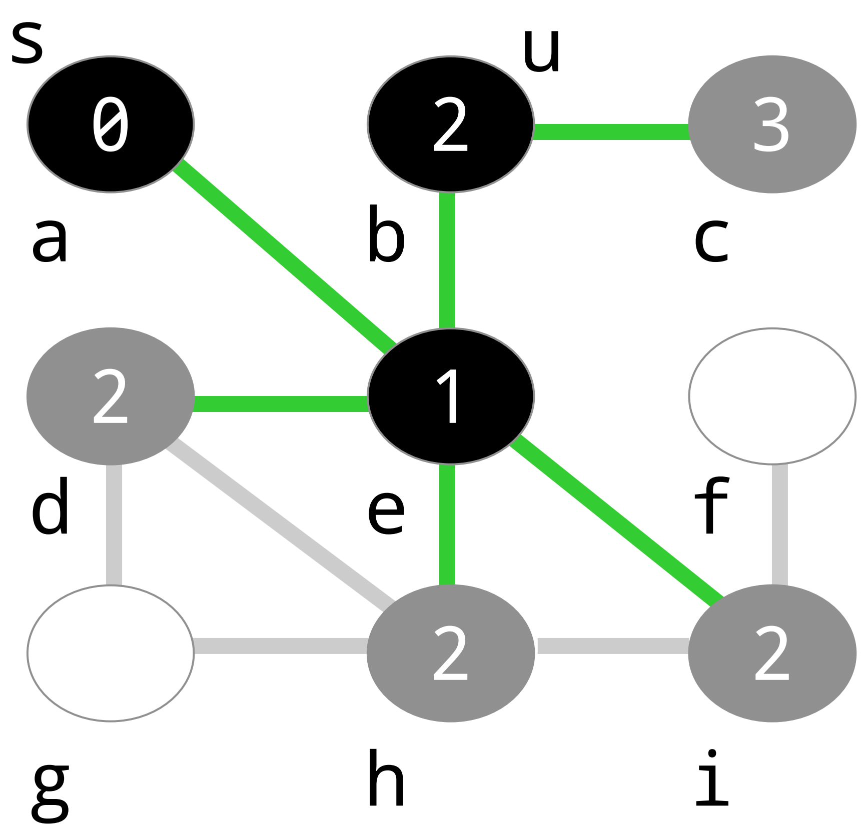
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

d	i	h	c

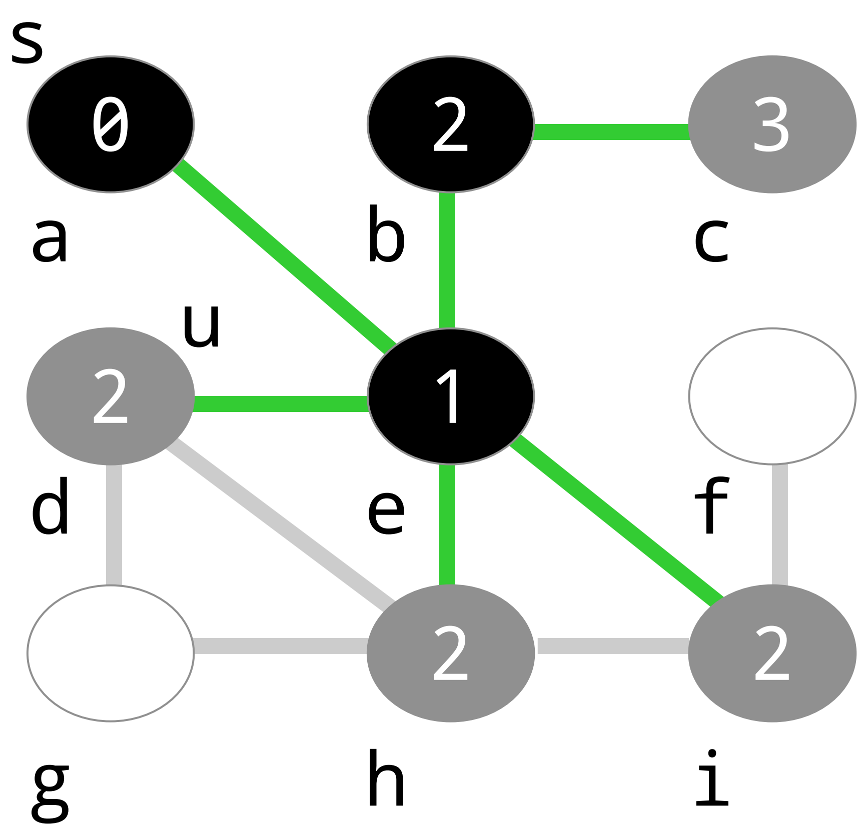
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(d)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | i | h | c |

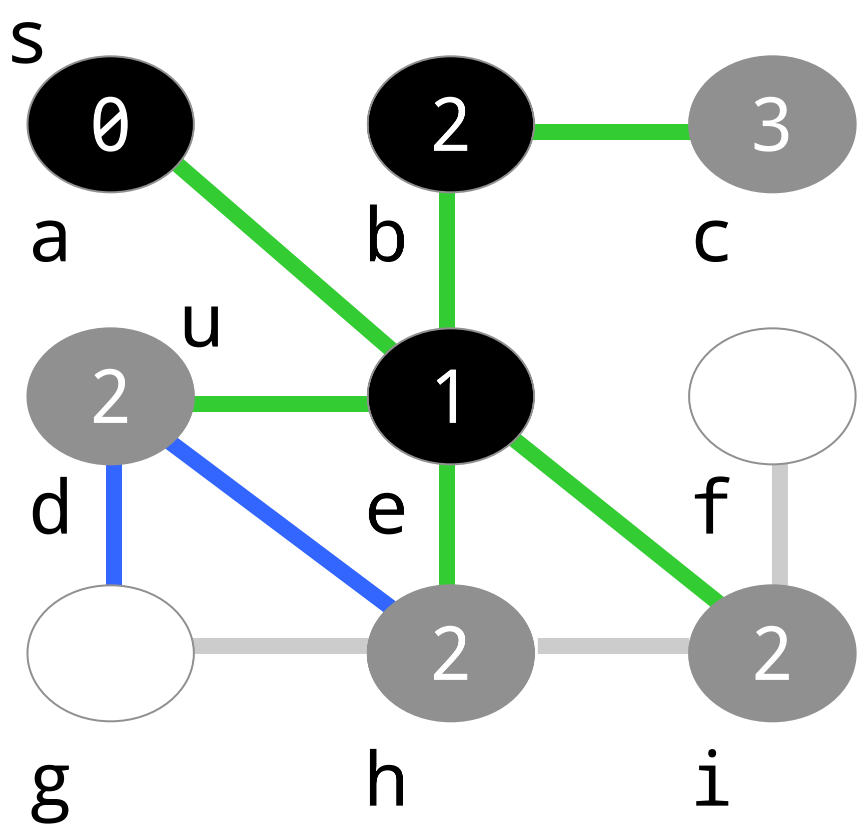
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:			
	i	h	c

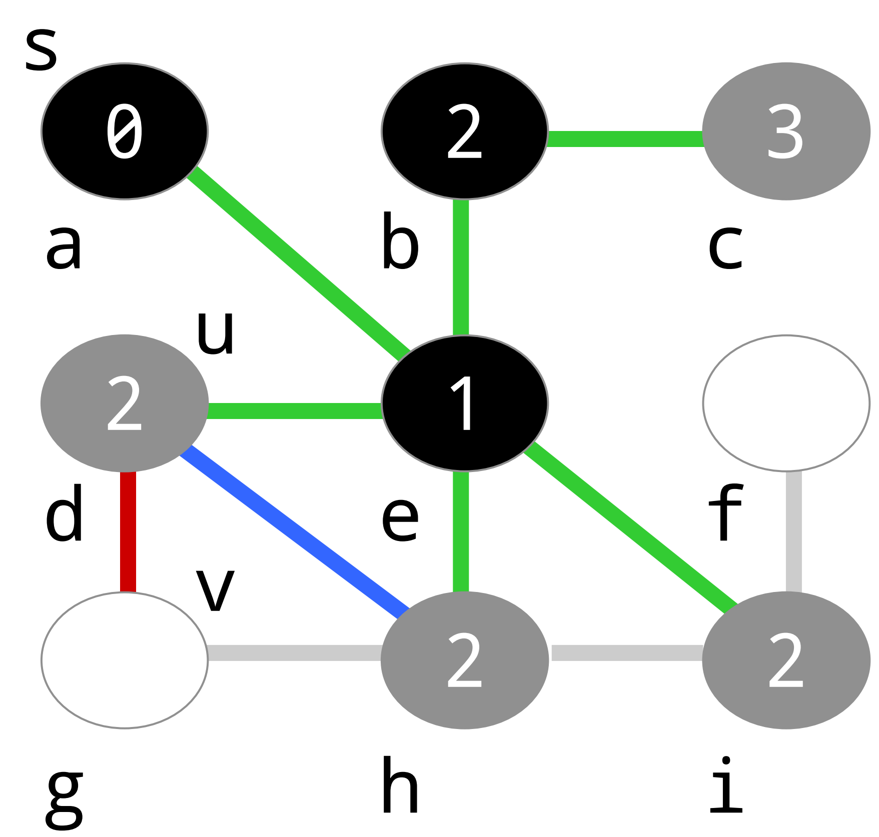
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



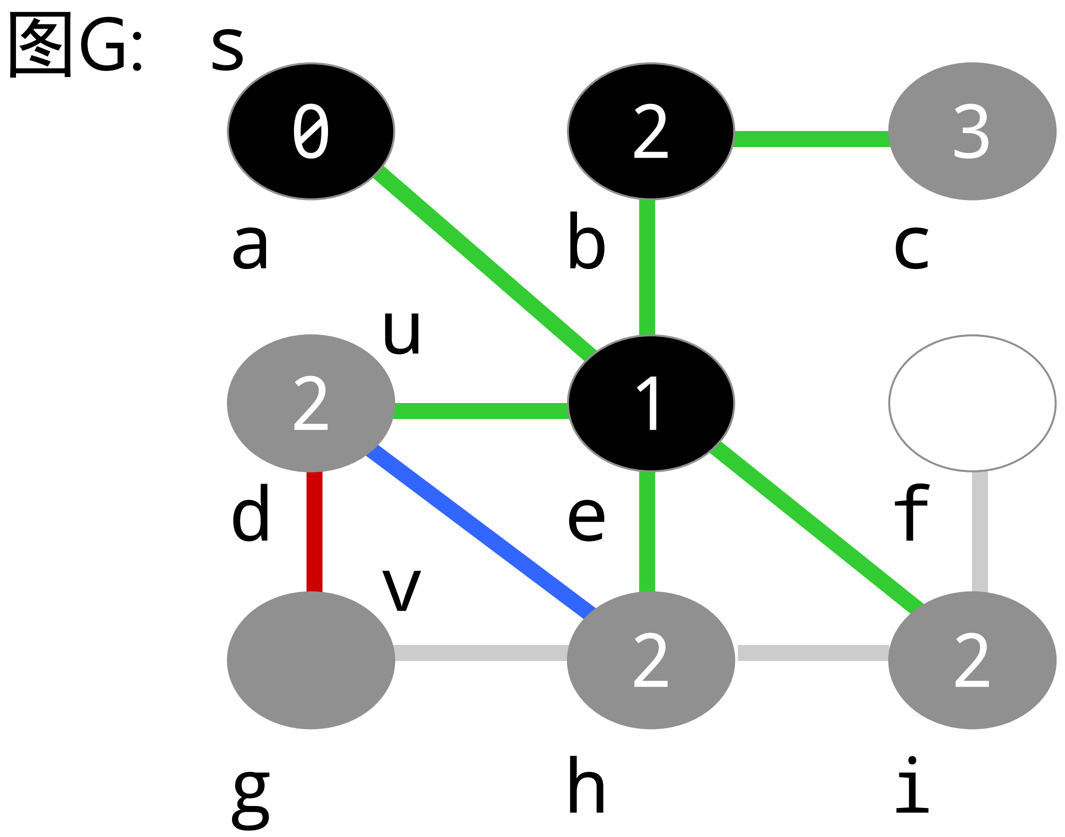
队列Q: | i | h | c |

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: | i | h | c |

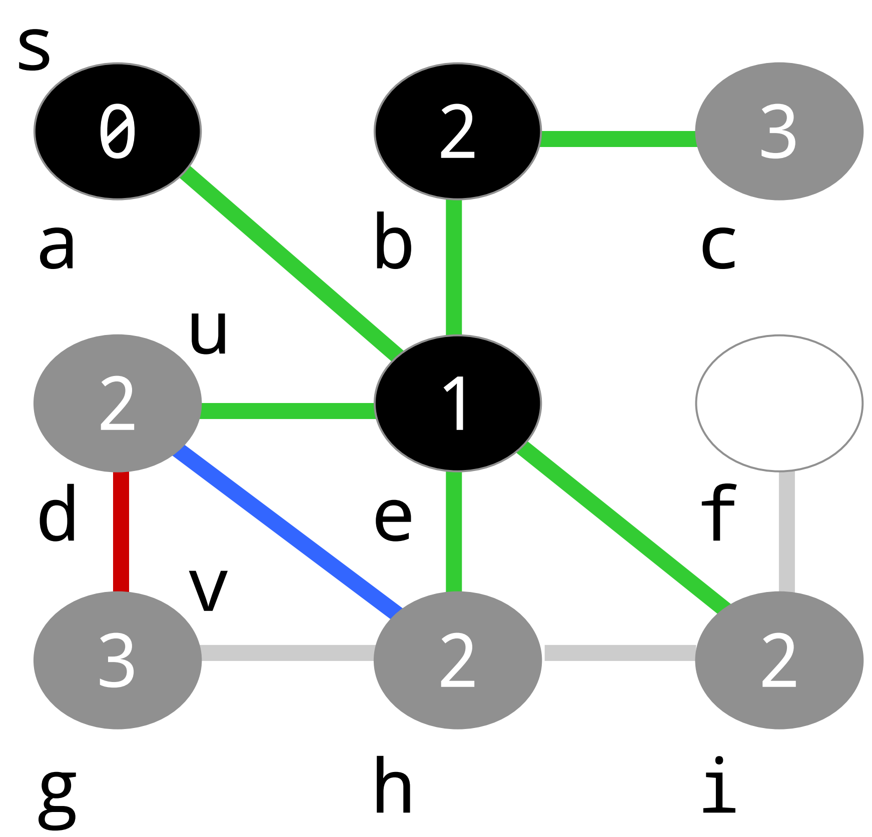
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | i | h | c |

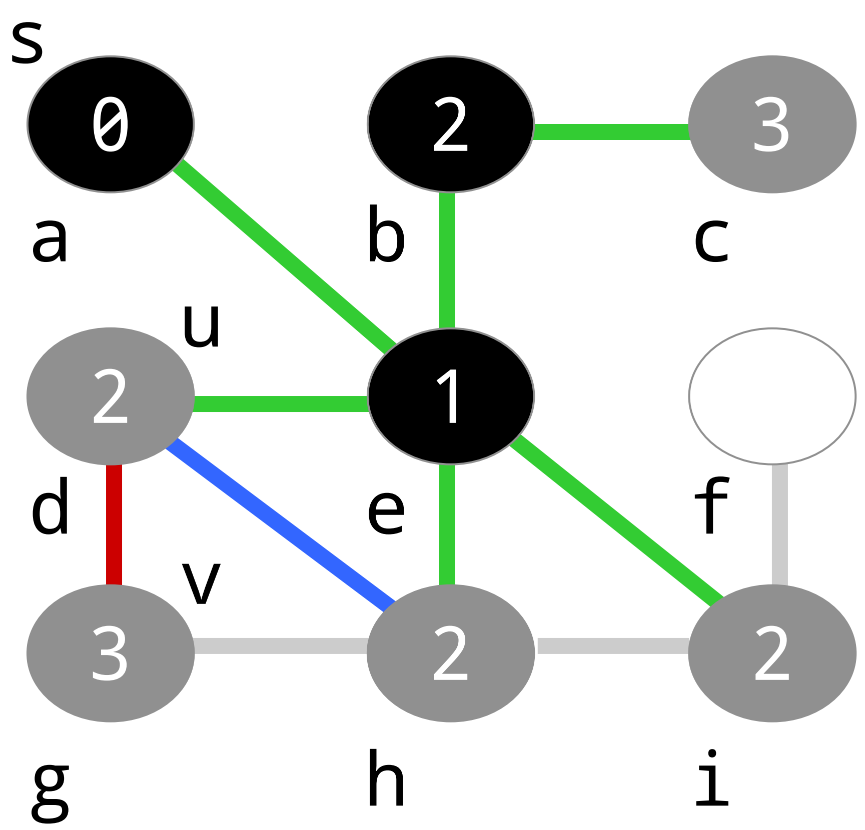
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(g)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: i h c g

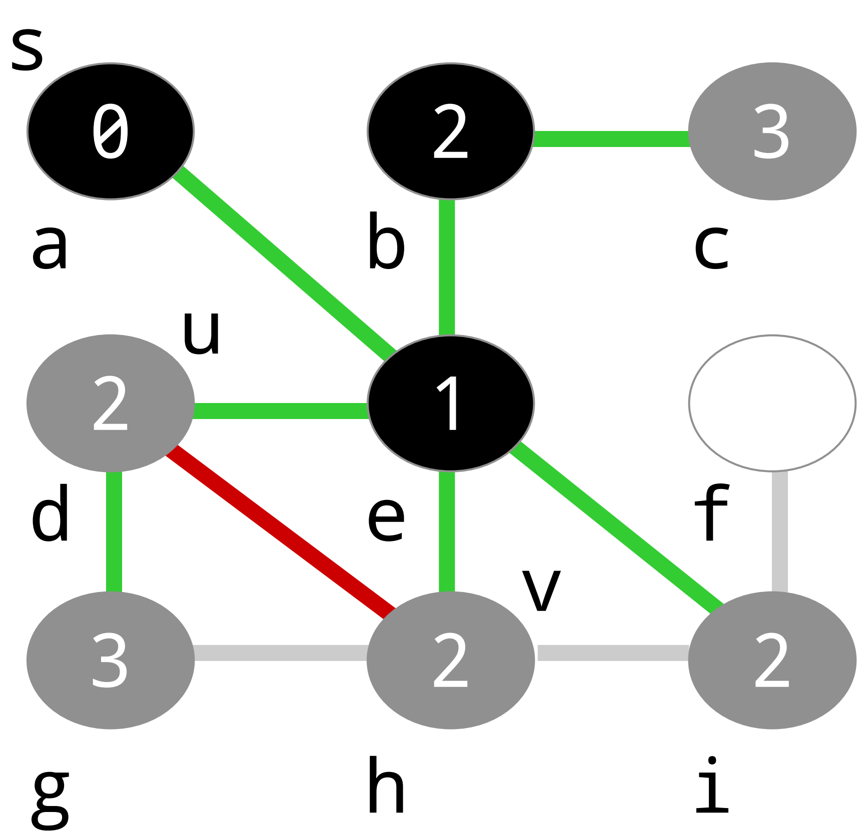
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: i h c g

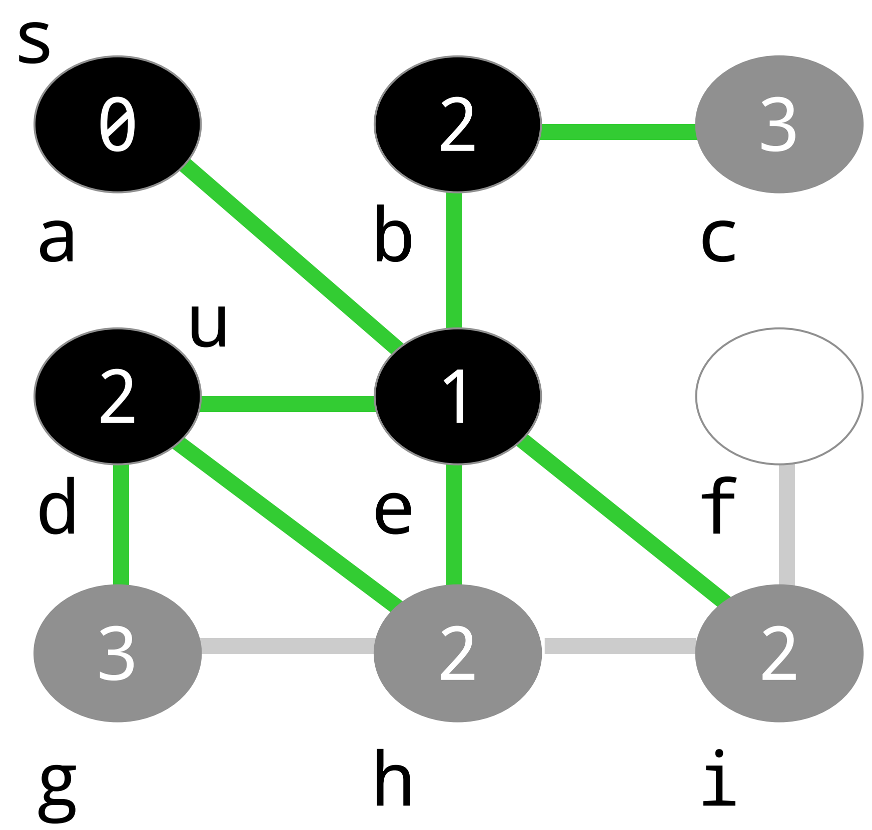
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



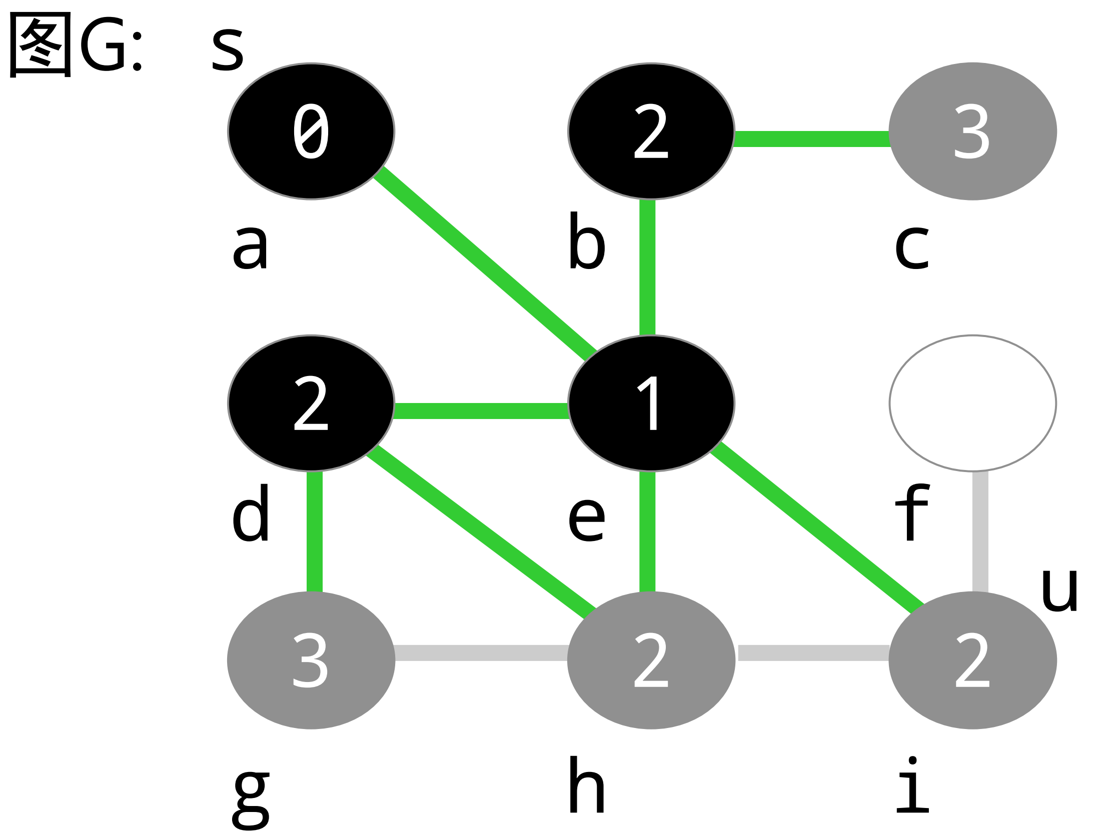
队列Q: | i | h | c | g |

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(i)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



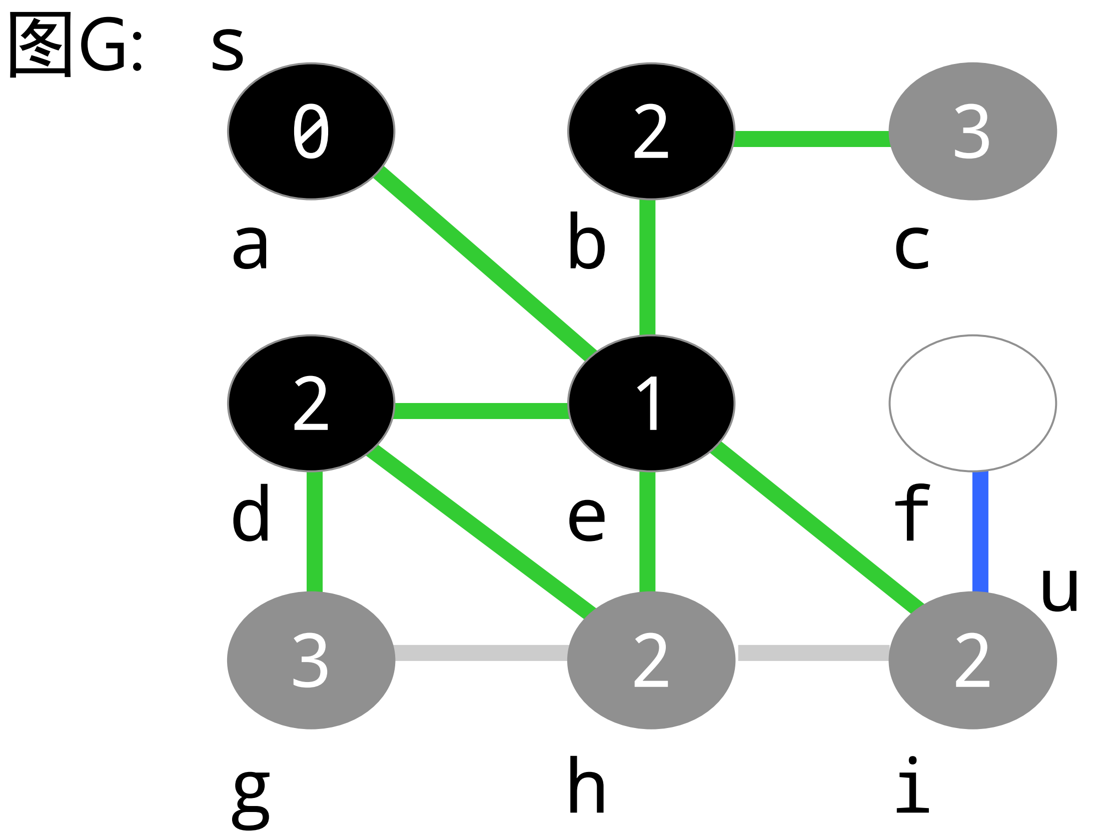
队列Q: | h | c | g |

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q:

h	c	g

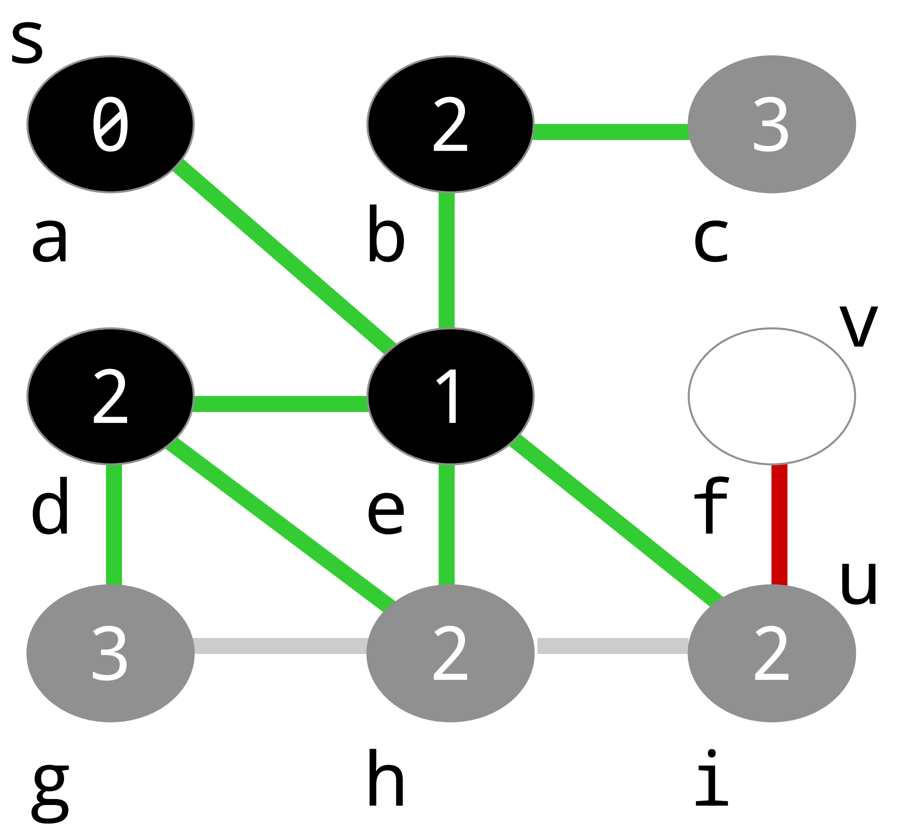
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

判断v是否为白色：未访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | h | c | g |

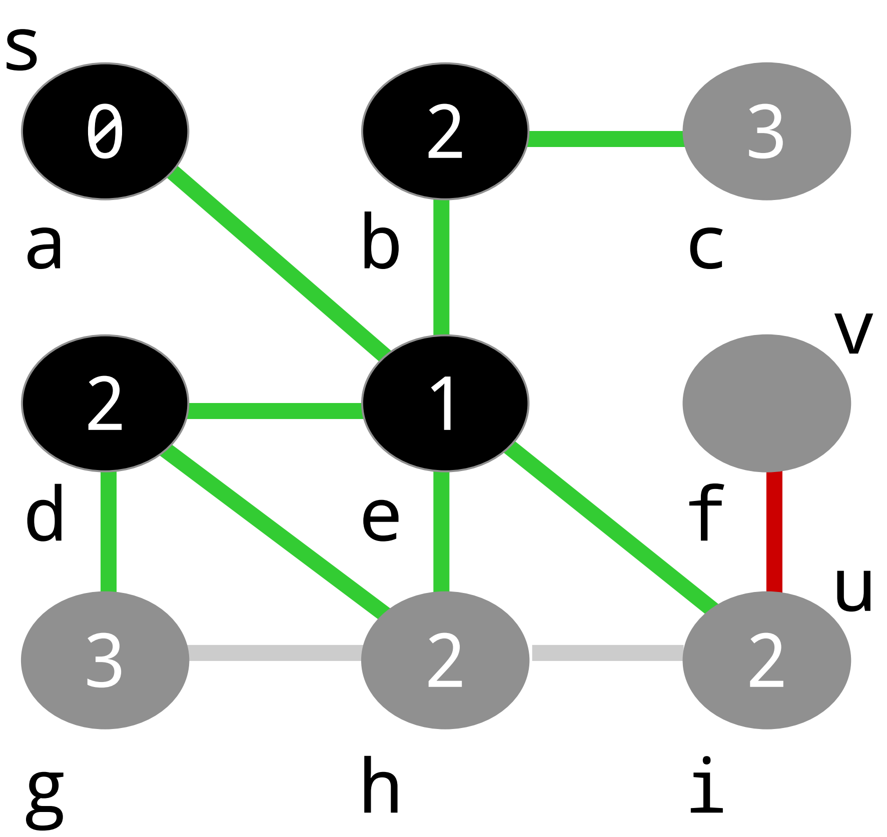
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

v置为灰色：已访问过的

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



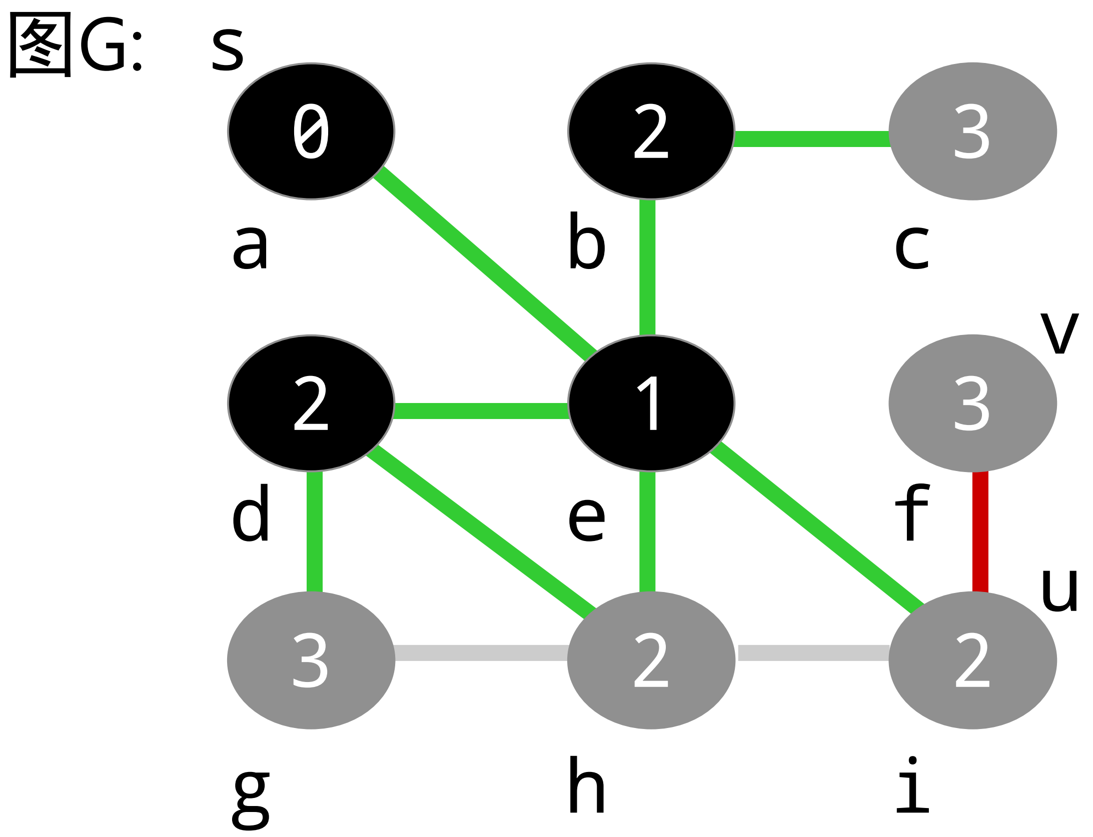
队列Q: h | c | g

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

设置距离为u.d+1

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: | h | c | g |

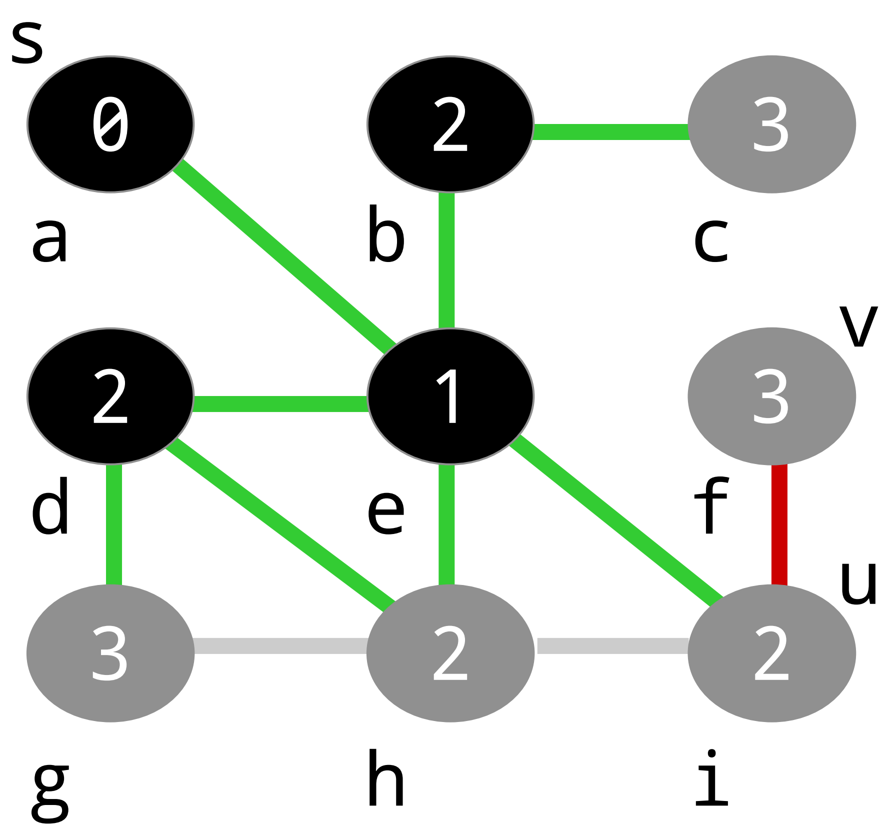
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u记为该顶点v的父母，v(f)插入队列中

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | h | c | g | f |

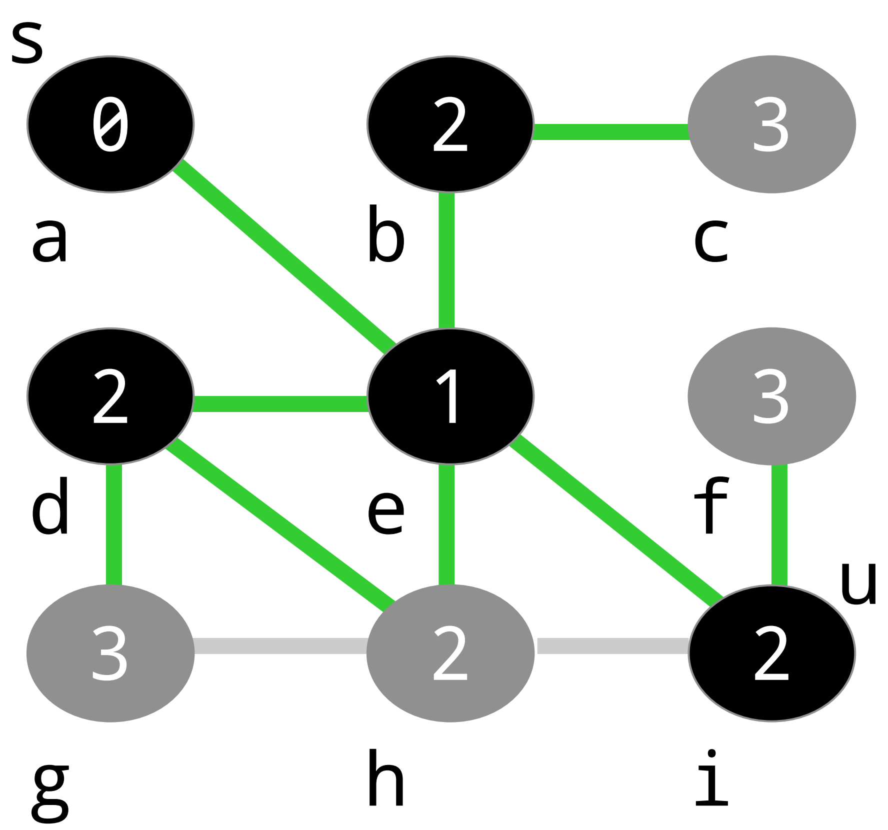
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | h | c | g | f |

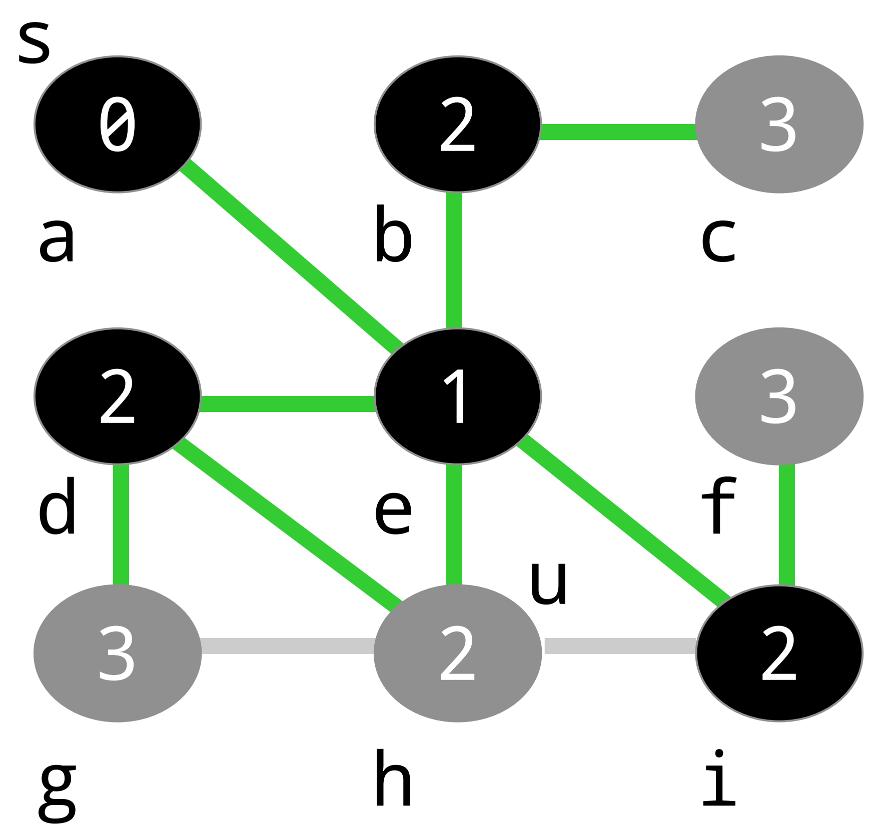
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(h)，并将其从Q中去掉

```
BFS(G,s):
  for each u ∈ G.V - {s}
    u.color = white
  s.color = gray
  s.d = 0
  Q = ∅
  Q.enqueue(s)
  while Q ≠ ∅
    u = Q.dequeue()
    for each v ∈ G.Adj[u]
      if v.color == white
        v.color = gray
        v.d = u.d + 1
        v.pred = u
        Q.enqueue(v)
    u.color = black
```

图G:



队列Q: c g f

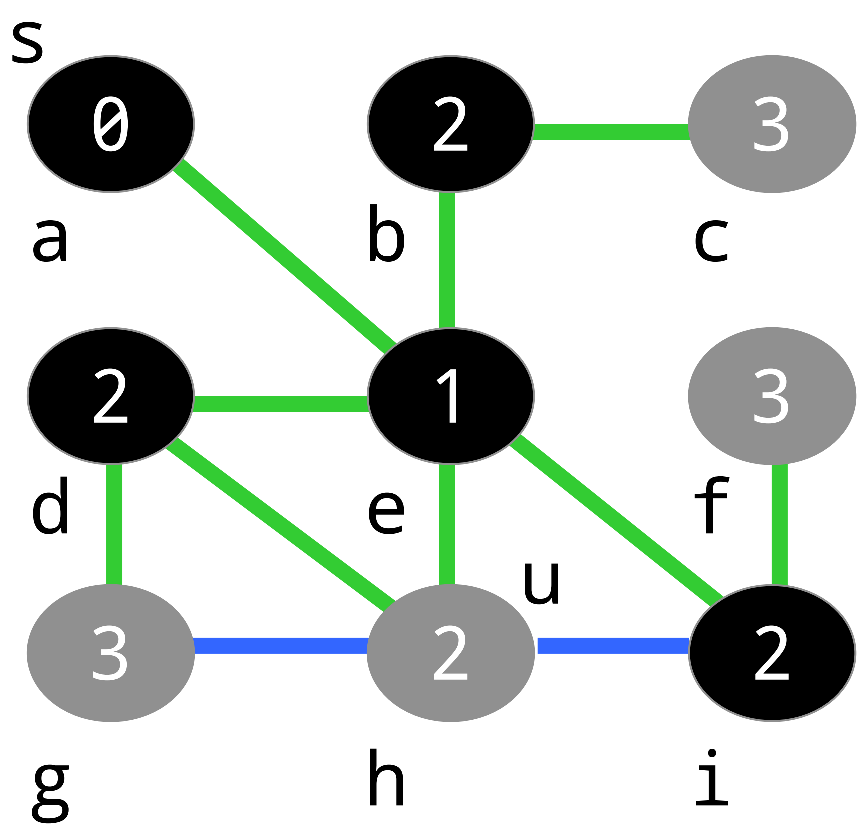
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

循环检查u的邻接表中的每个顶点v

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: c | g | f

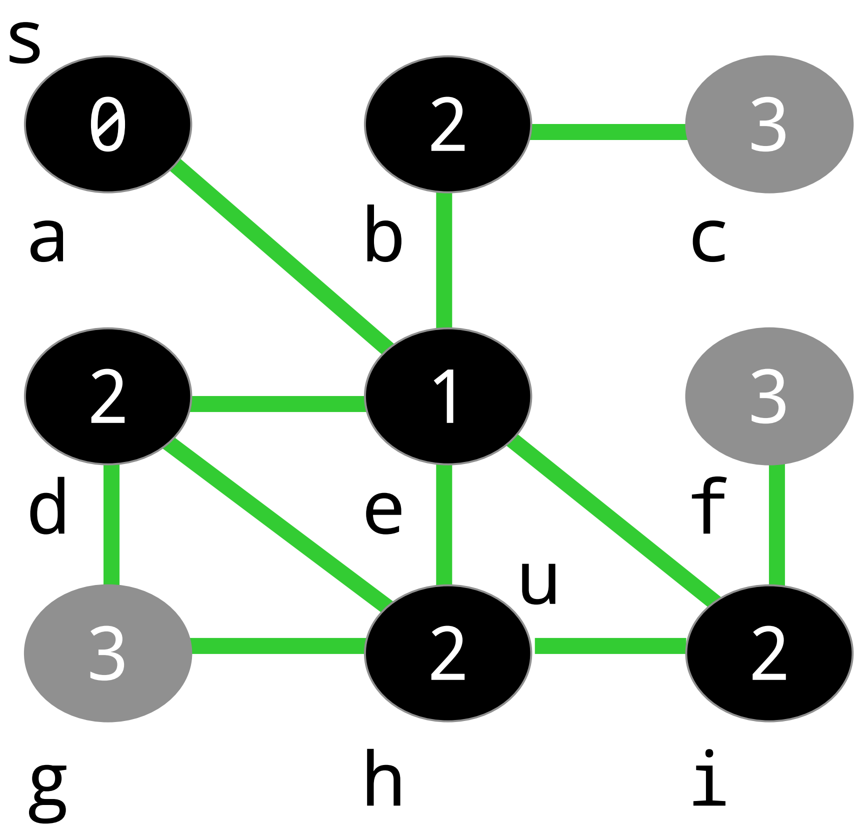
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



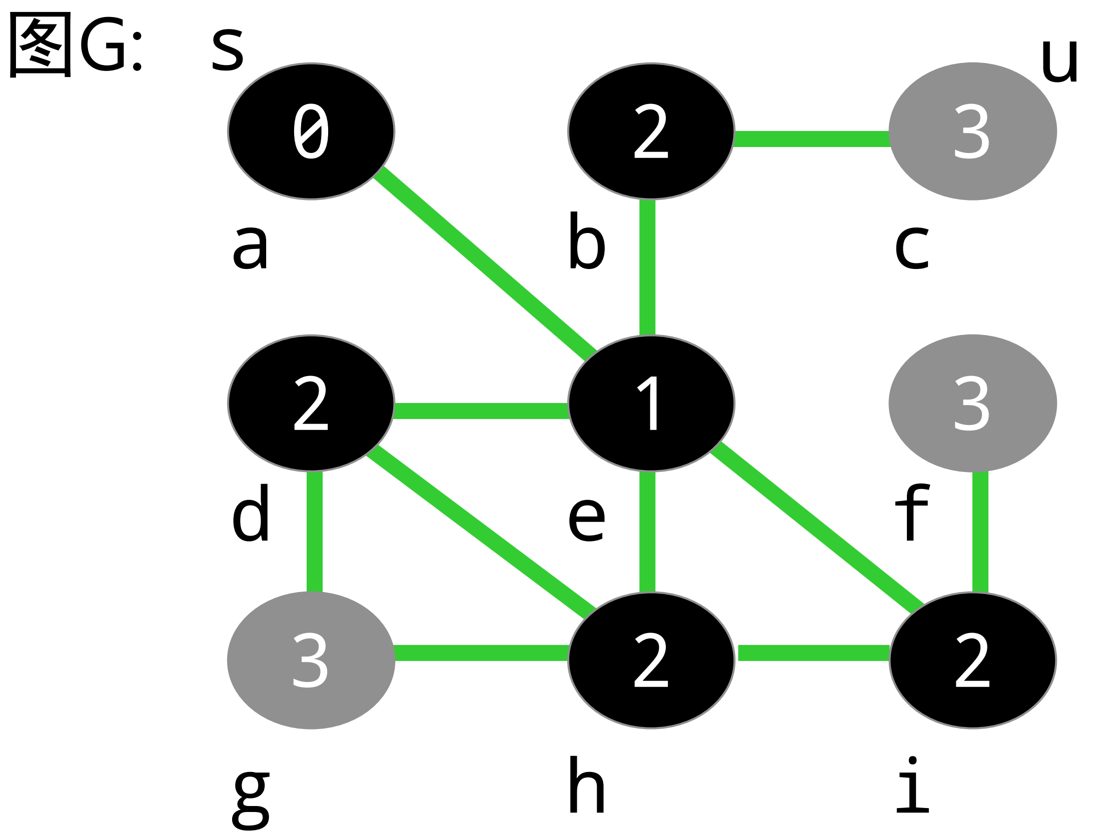
队列Q: | c | g | f |

11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(c)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```



队列Q: g f

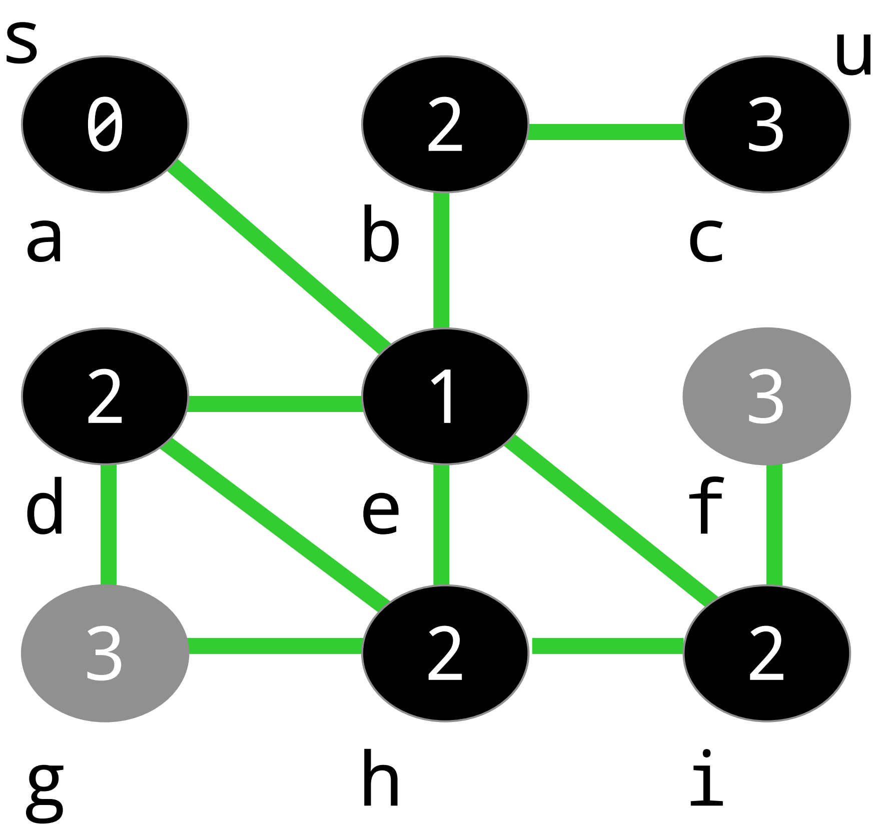
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: | g | f |

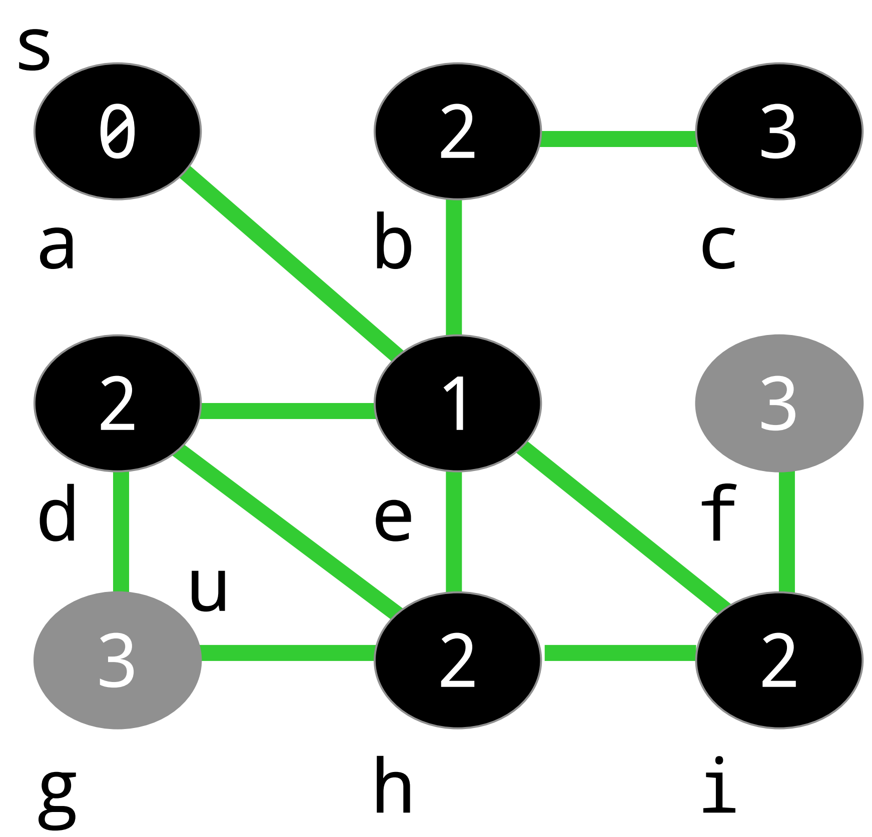
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(g)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: f

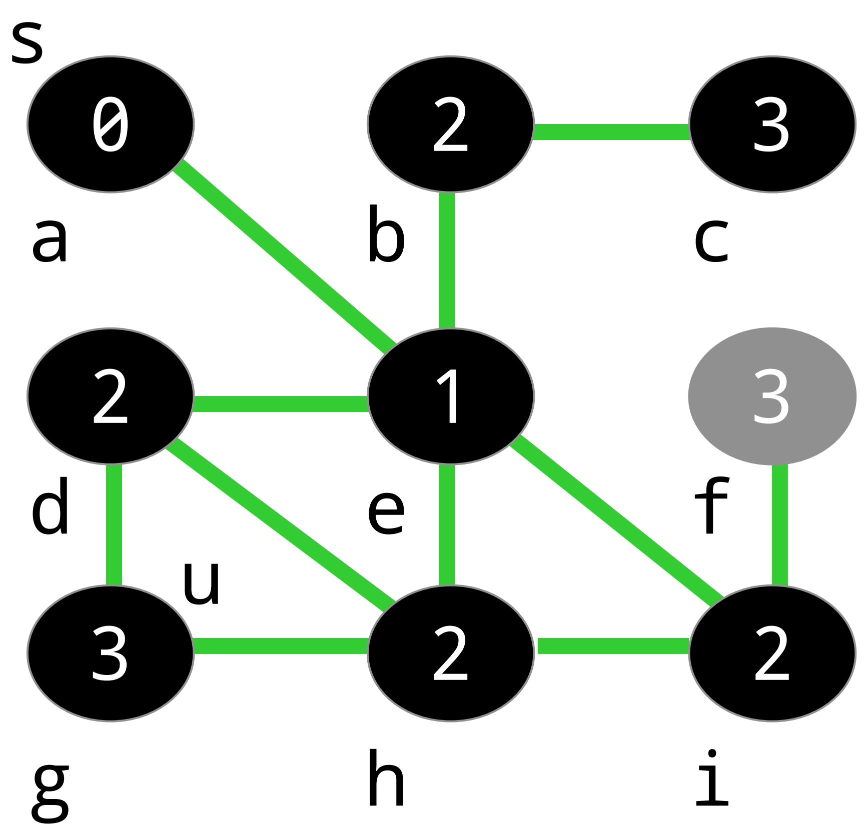
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q: f

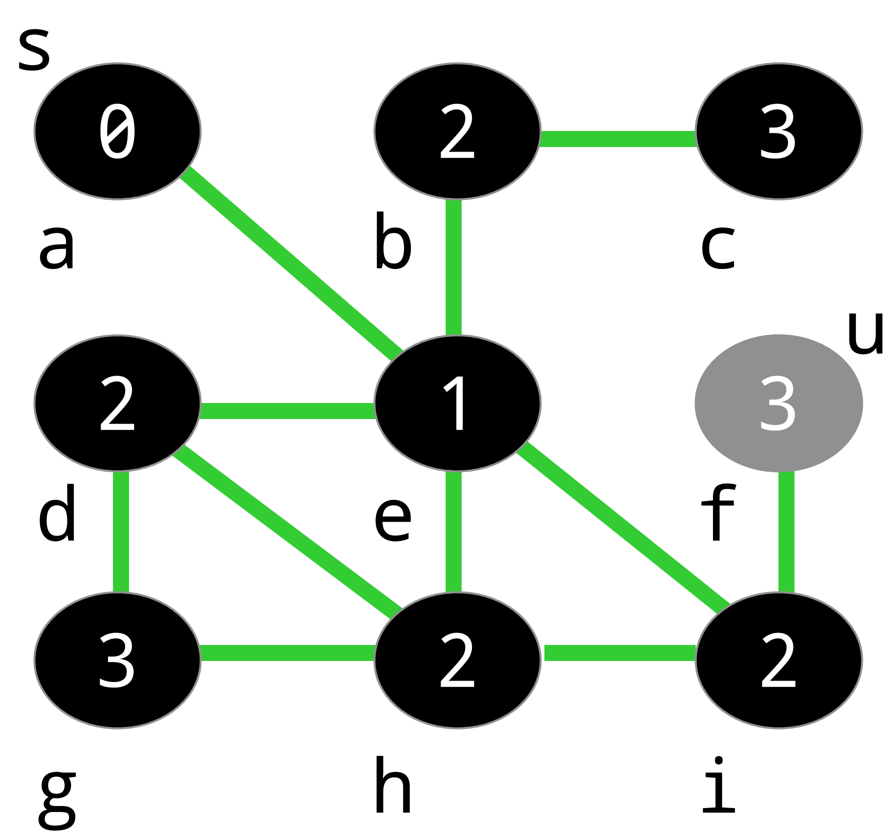
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

确定队列Q头部的灰色顶点u(f)，并将其从Q中去掉

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

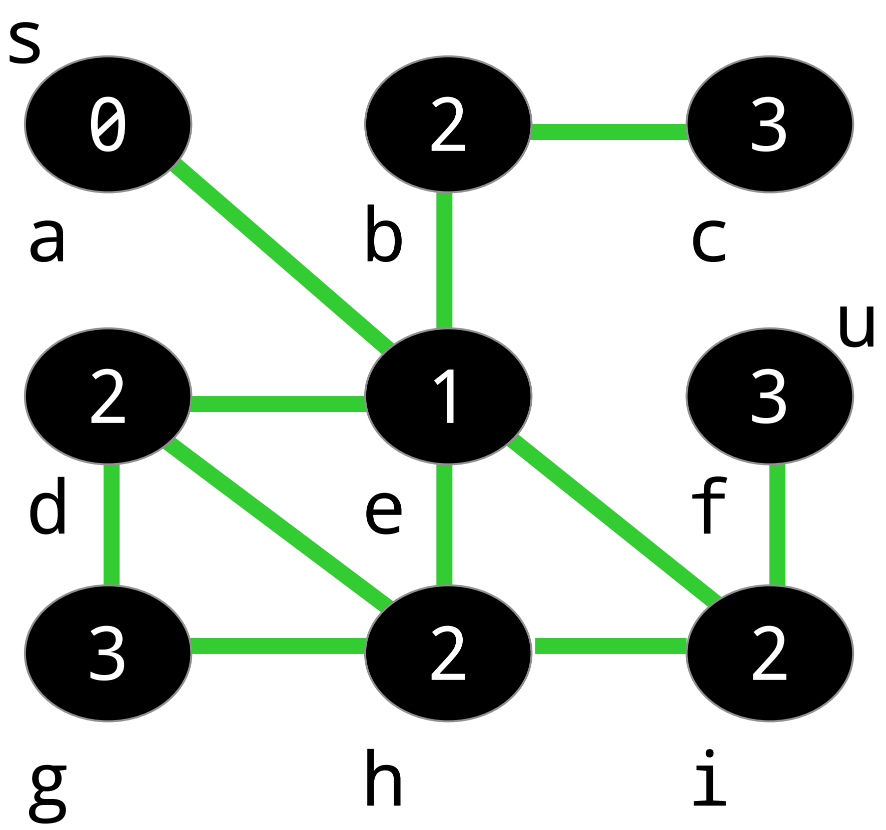
11.3.7 广度优先搜索BFS

BFS广度优先搜索（Breadth-first Search）算法示意

u置为黑色：已访问过的，且邻接顶点均访问过

```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white
s.color = gray
s.d = 0
Q = ∅
Q.enqueue(s)
while Q ≠ ∅
  u = Q.dequeue()
  for each v ∈ G.Adj[u]
    if v.color == white
      v.color = gray
      v.d = u.d + 1
      v.pred = u
      Q.enqueue(v)
  u.color = black
```

图G:



队列Q:

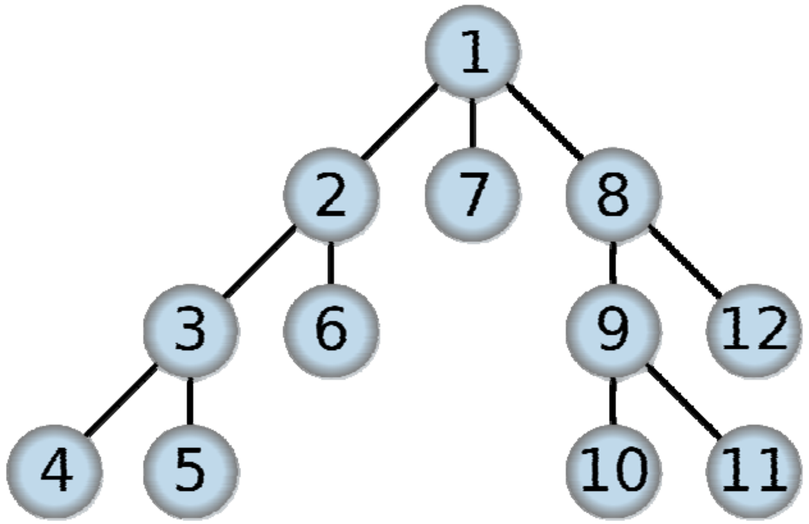
11.3.7 广度优先搜索BFS

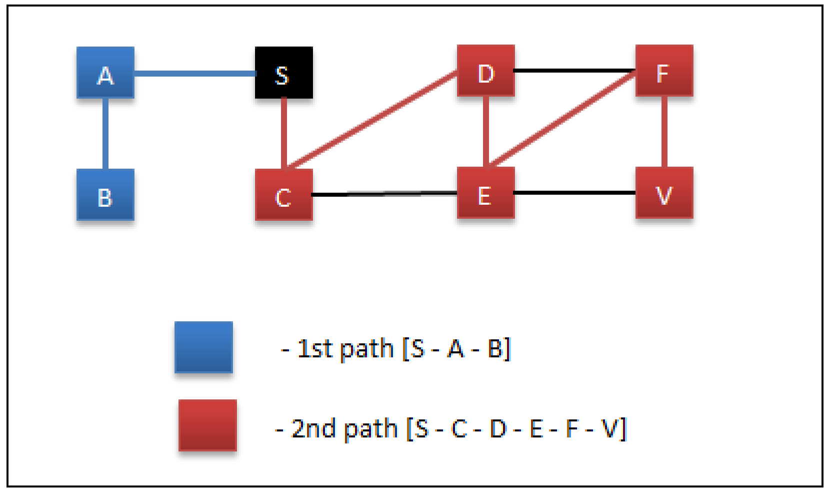
BFS广度优先搜索（Breadth-first Search）算法

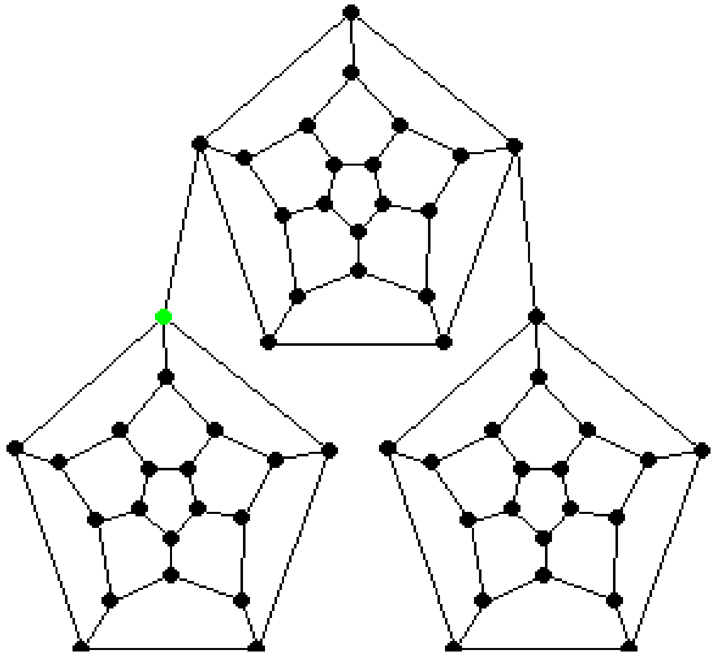
```
BFS(G,s):
for each u ∈ G.V - {s}
  u.color = white //除源顶点s外，置每个顶点为白色
s.color = gray //开始时，源顶点s已被发现，置为灰色
s.d = 0 //将s.d距离初始化为0
Q = 1 //队列Q置空
Q.enqueue(s) //入队，初始化队列Q，使其仅含源顶点s
while Q ≠ 1
  u = Q.dequeue() //出队，确定队列Q头部的灰色顶点u，并将其从Q中去掉
  for each v ∈ G.Adj[u] //循环检查u的邻接表中的每个顶点v
    if v.color == white //判断v是否为白色：未访问过的
      v.color = gray //v置为灰色：已访问过的
      v.d = u.d + 1 //设置距离为u.d+1
      v.pred = u //u记为顶点v的父母
      Q.enqueue(v) //v插入队列中
  u.color = black //u置为黑色：已访问过的，且邻接顶点均访问过
```

11.3.8 深度优先搜索DFS

- 二、深度优先搜索（DFS，Depth First Search）
- 深度优先搜索所遵循的搜索策略是尽可能“深”地搜索一个图，对于新发现的顶点，如果它还有以此为起点而未探测到的边，就沿此边继续探测下去。

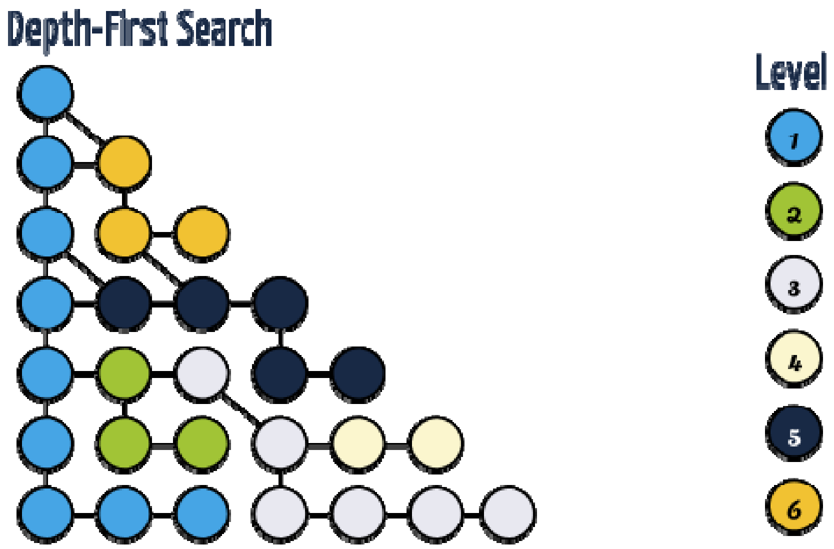


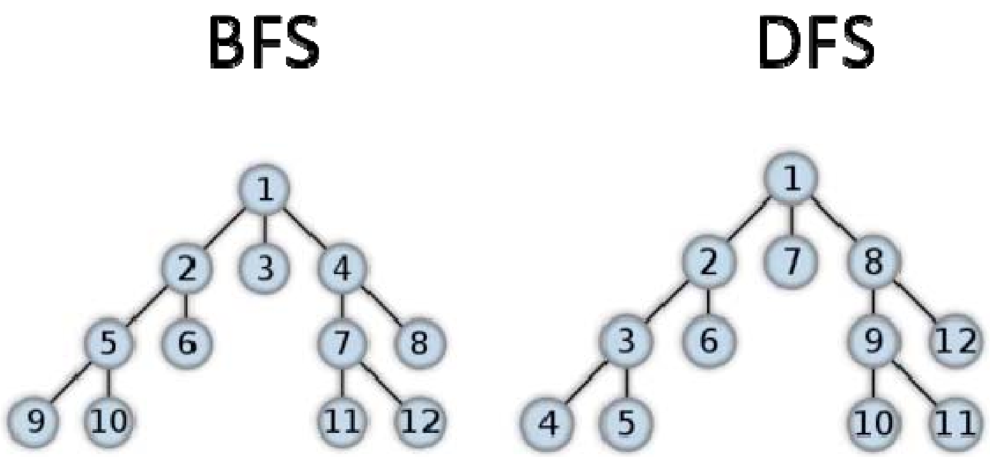




11.3.8 深度优先搜索DFS

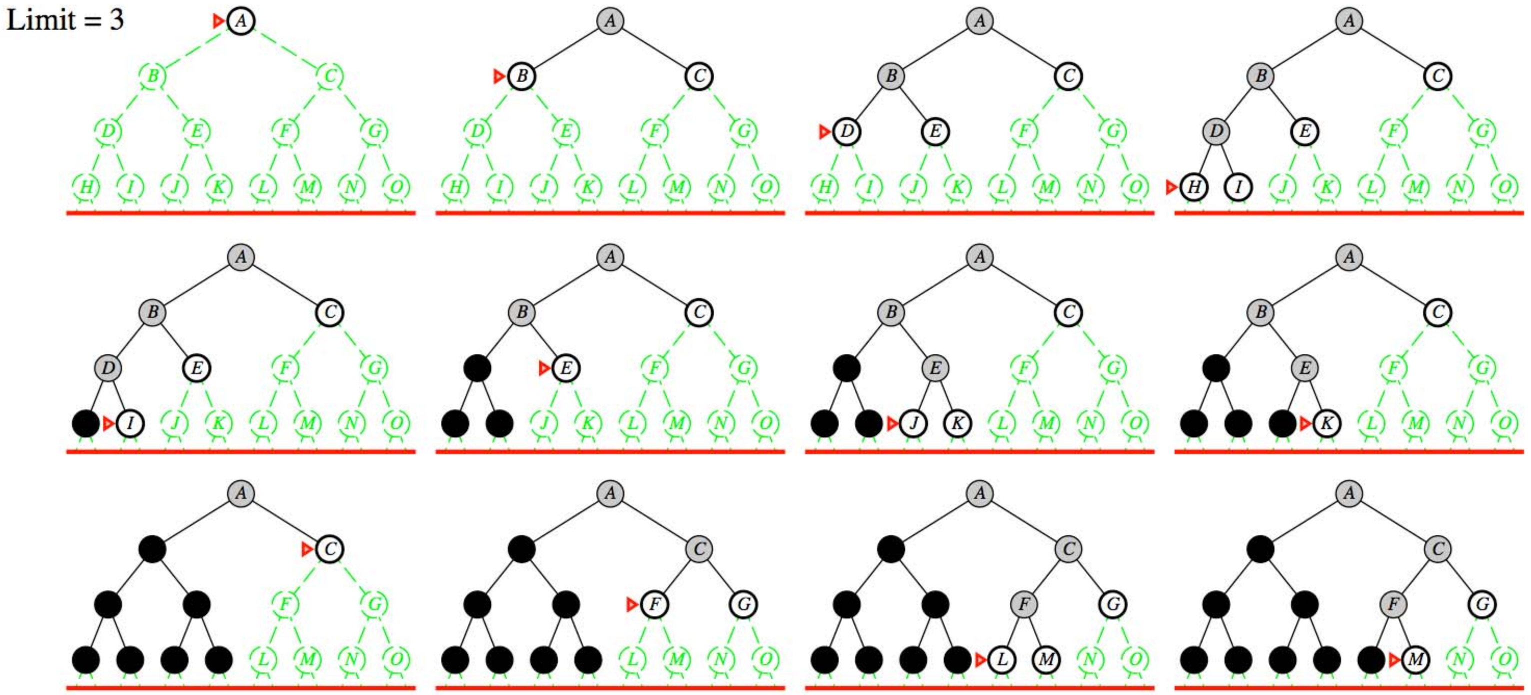
- 当顶点v的所有边都已被探寻过后，搜索将回溯到发现顶点v有起始点的那些边。这一过程一直进行到已发现从源顶点可达的所有顶点时为止。如果还存在未被发现的顶点，则选择其中一个作为源顶点，并重复以上过程。整个过程反复进行，直到所有的顶点都被已发现为止。

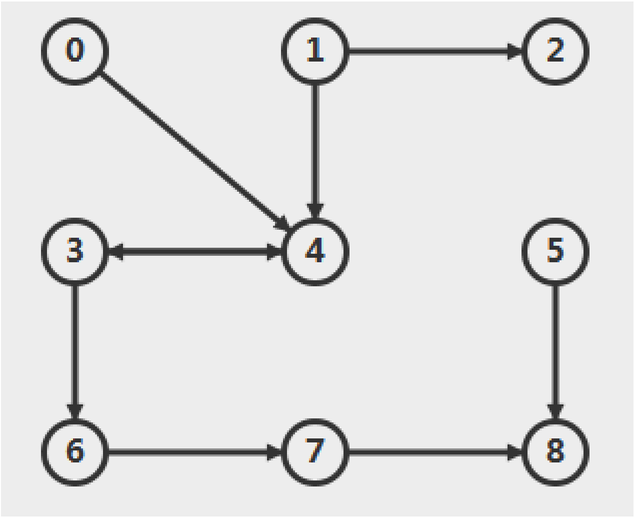




11.3.8 深度优先搜索DFS

▶1. DFS算法原理



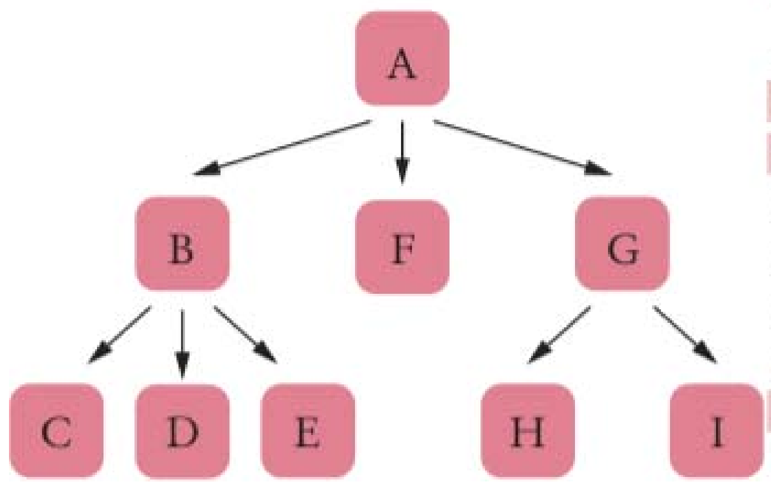


11.3.8 深度优先搜索DFS

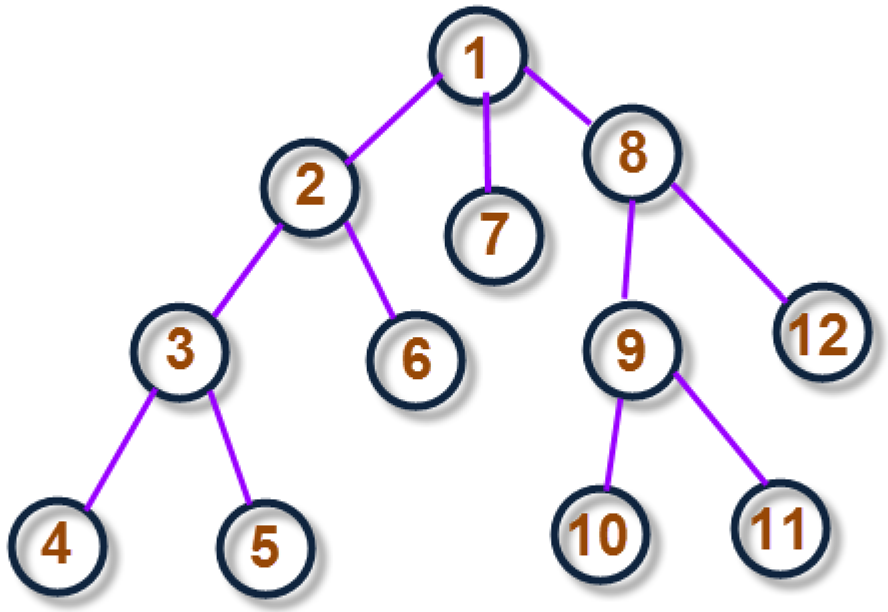
- 1. DFS算法原理
- 每当扫描已发现结点u的邻接表从而发现新结点v时，深度优先搜索将置v的先辈为u。其先辈子图形成一个由数个深度优先树组成的深度优先森林。
- 深度优先在搜索过程中也为结点着色以表示结点的状态。每个顶点开始均为白色，搜索中被发现时置为灰色，结束时又被置成黑色(即当其邻接表被完全检索之后)。这样可以保证每一顶点搜索结束时只存在于一棵深度优先树上，因此这些树都是分离的。

11.3.8 深度优先搜索DFS

- 1. DFS算法原理
- 除了创建一个深度优先森林外，深度优先搜索同时为每个结点加盖时间戳。每个结点u有两个时间戳：当结点u第一次被发现时记录下发现时刻u.d，当结束检查u的邻接表时(并置u为黑色)记录下完成时刻u.f。



Stack
A
G F B PushchildrenofA
G F E DC PushchildrenofB
G F E D
G F E
G F
G
I H PushchildrenofG
I



11.3.8 深度优先搜索DFS

- 2．DFS算法性能分析
- （1）空间复杂度
- DFS算法是一个递归算法，需要借助一个递归工作栈，故它的空间复杂度为O(|V|)。
- （2）时间复杂度
- ①当以邻接表存储时，时间复杂度为O(|V|+|E|)。
- ②当以邻接矩阵存储时，时间复杂度为O(|V|^2)。

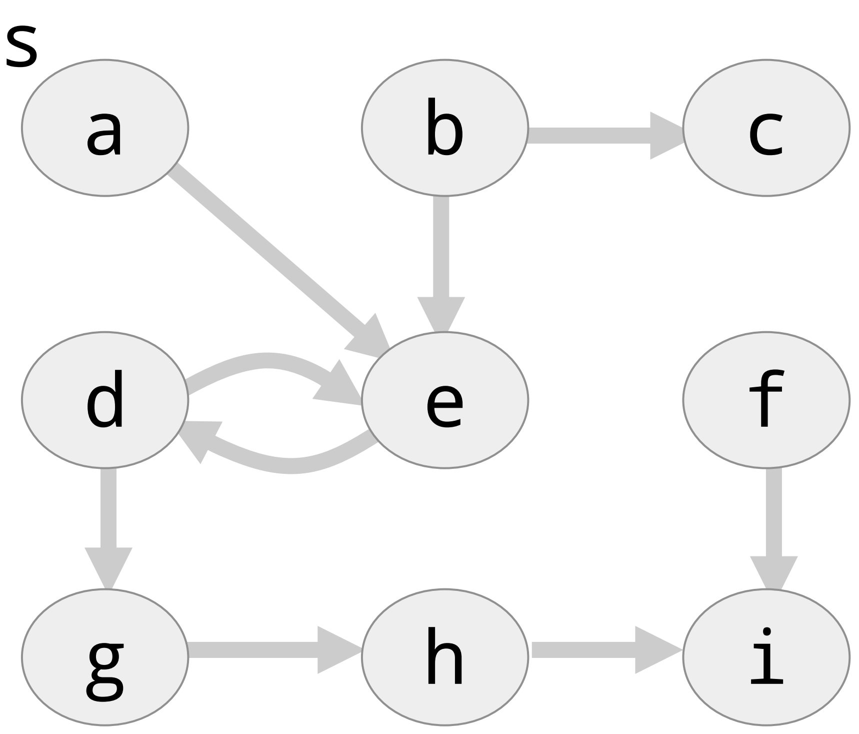
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:



调用栈:

time:

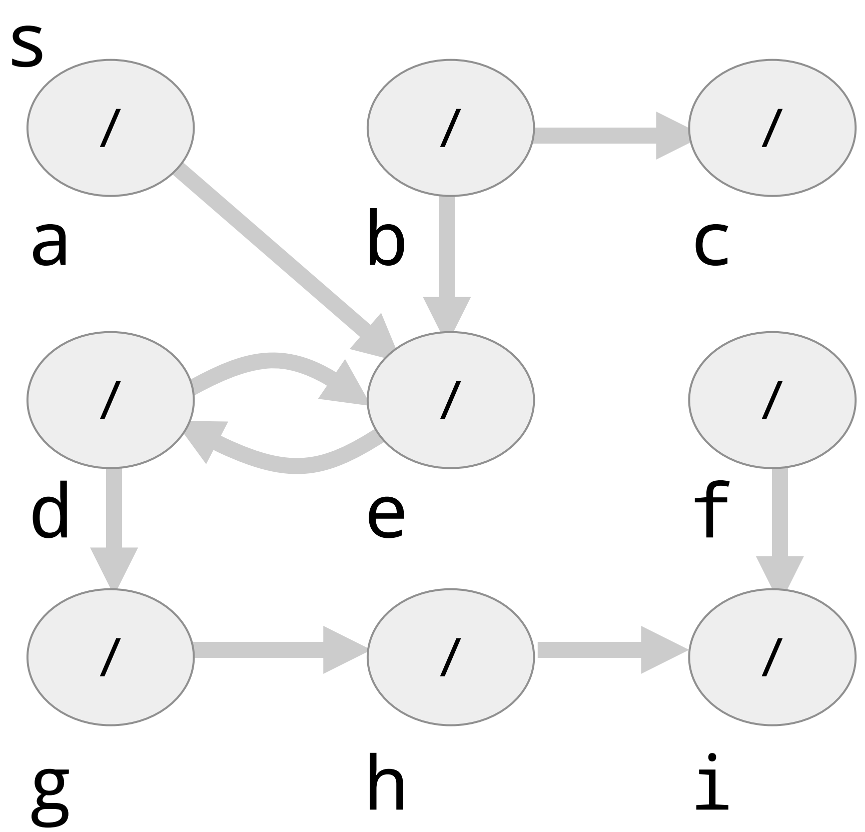
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:



调用栈:
↓
DFS(G)

time: -

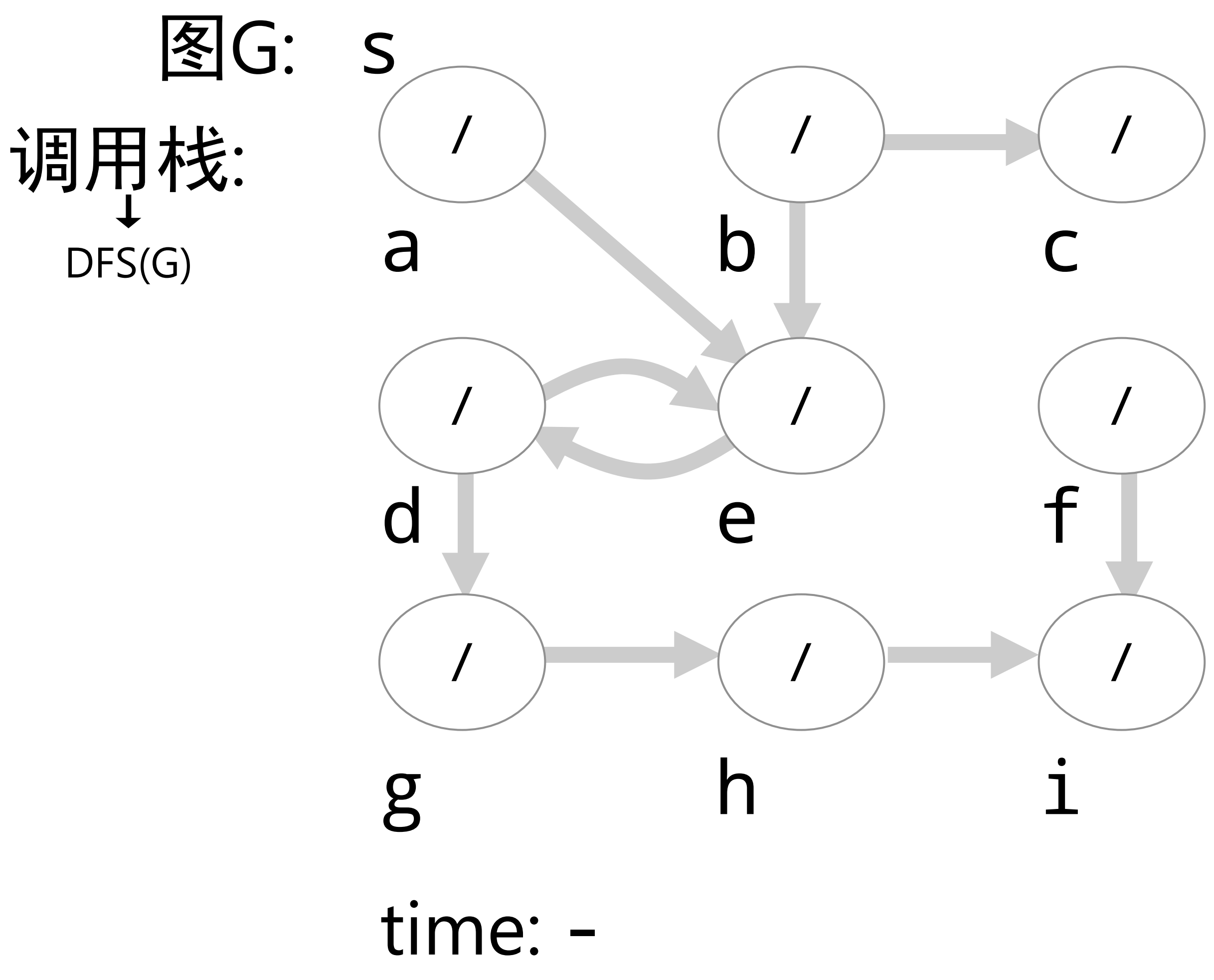
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

所有顶点置为白色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



调用栈:

DFS(G)

time: -

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

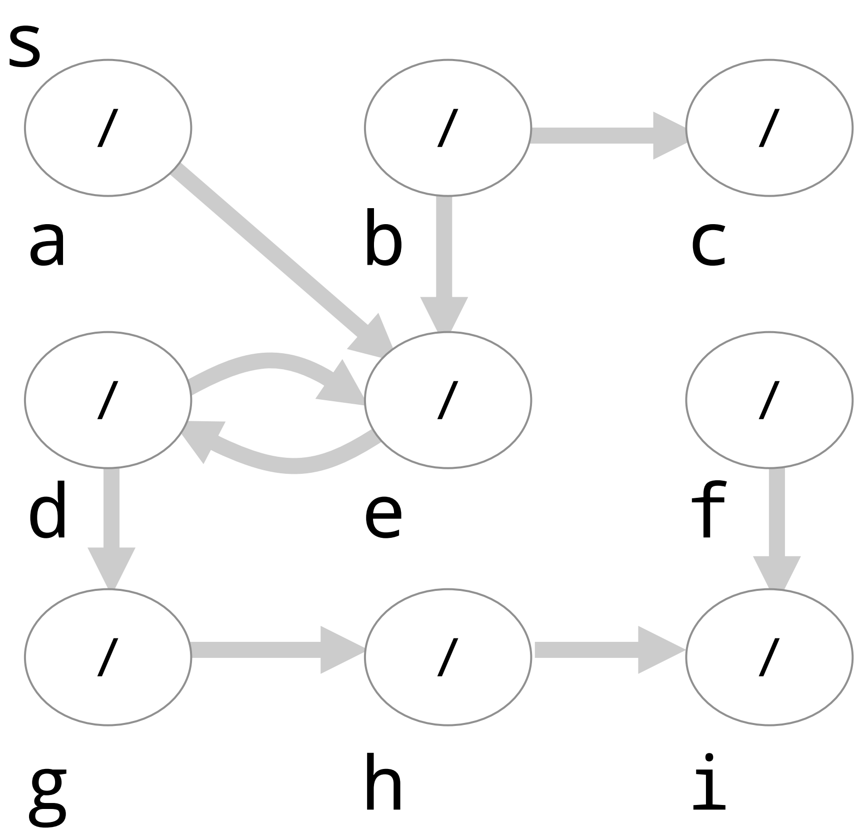
复位时间戳

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)

图G:



time: 0

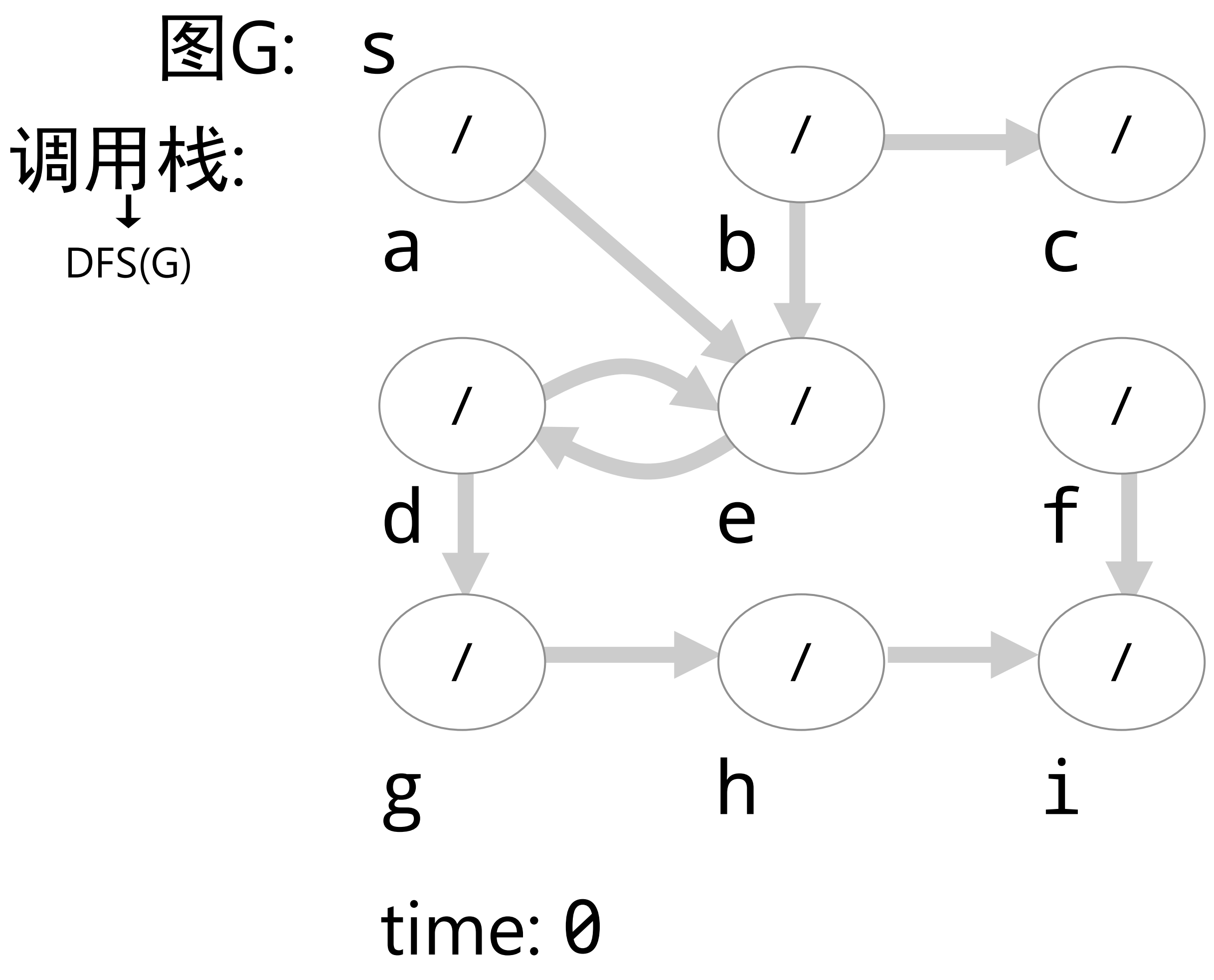
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

依次检索V中的顶点

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

发现白色顶点时，调用DFS-VISIT访问该顶点u(a)

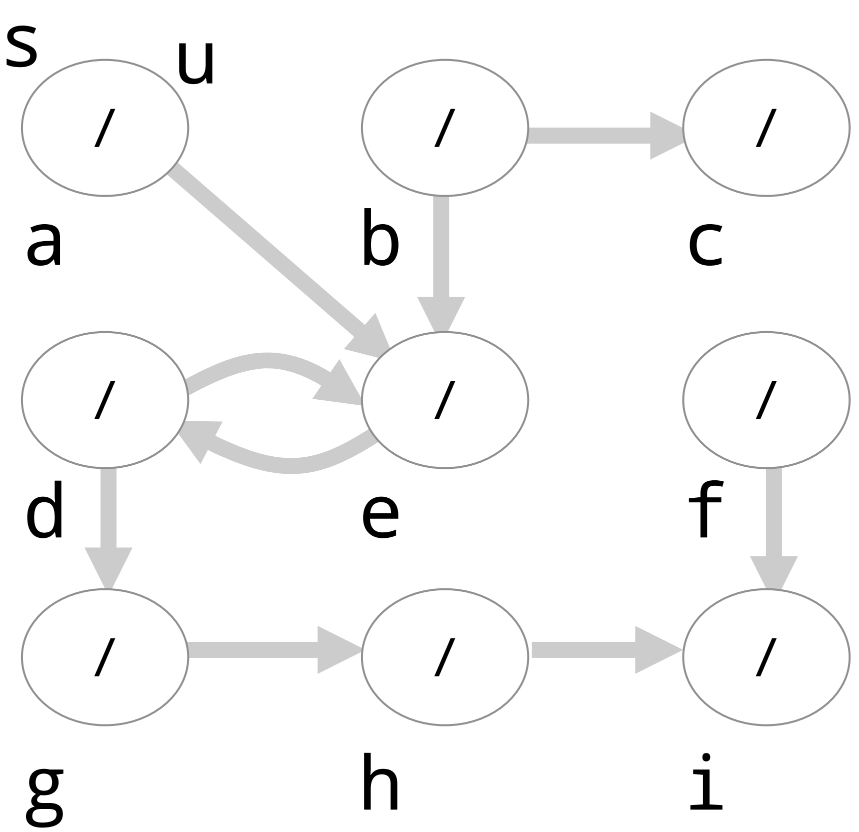
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

DFS(G)



time: 0

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

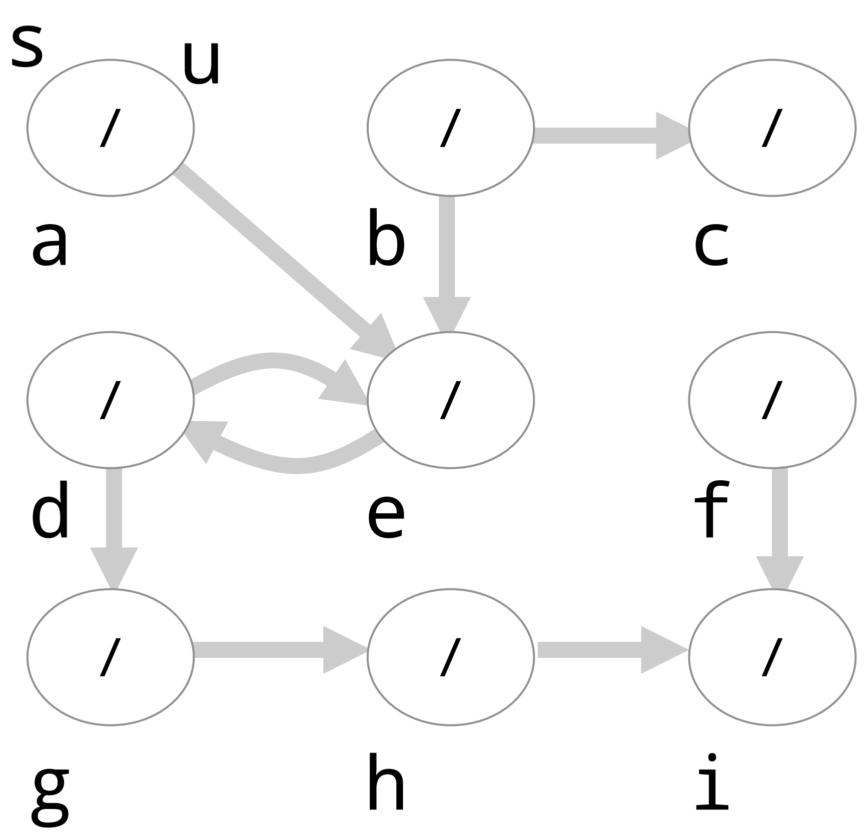
u(a)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)



time: 0

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

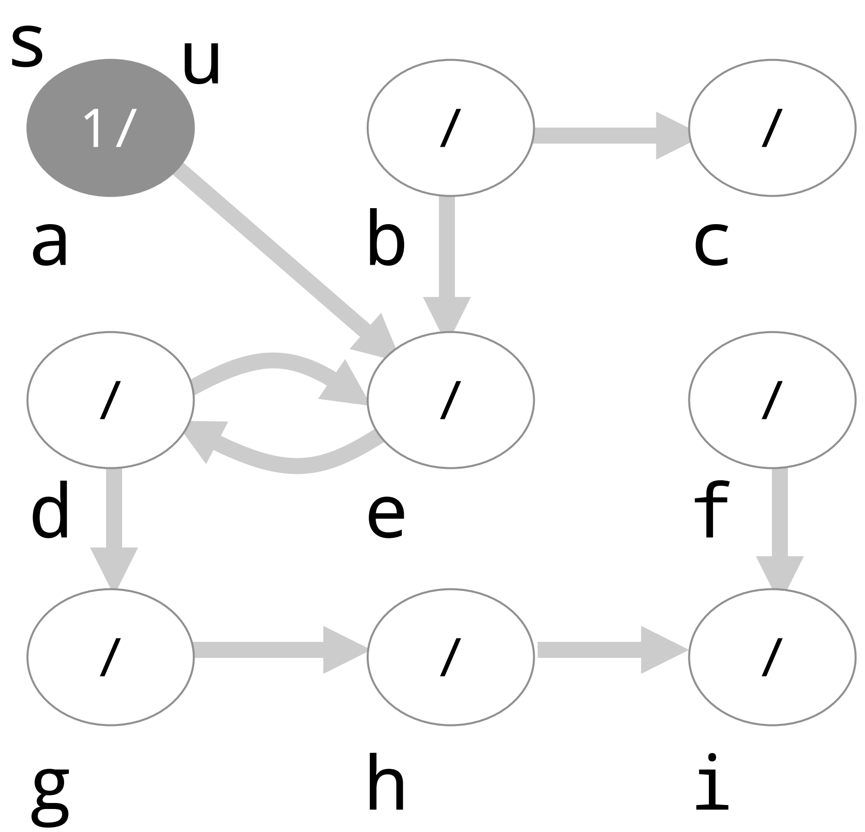
记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)

图G:



time: 1

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

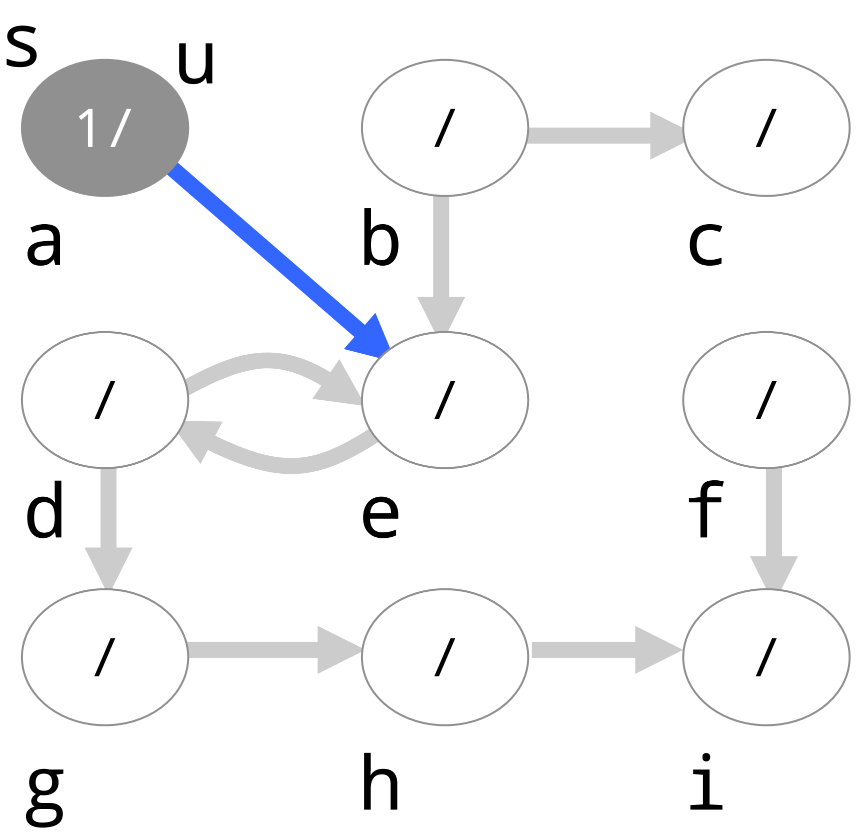
检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)



time: 1

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

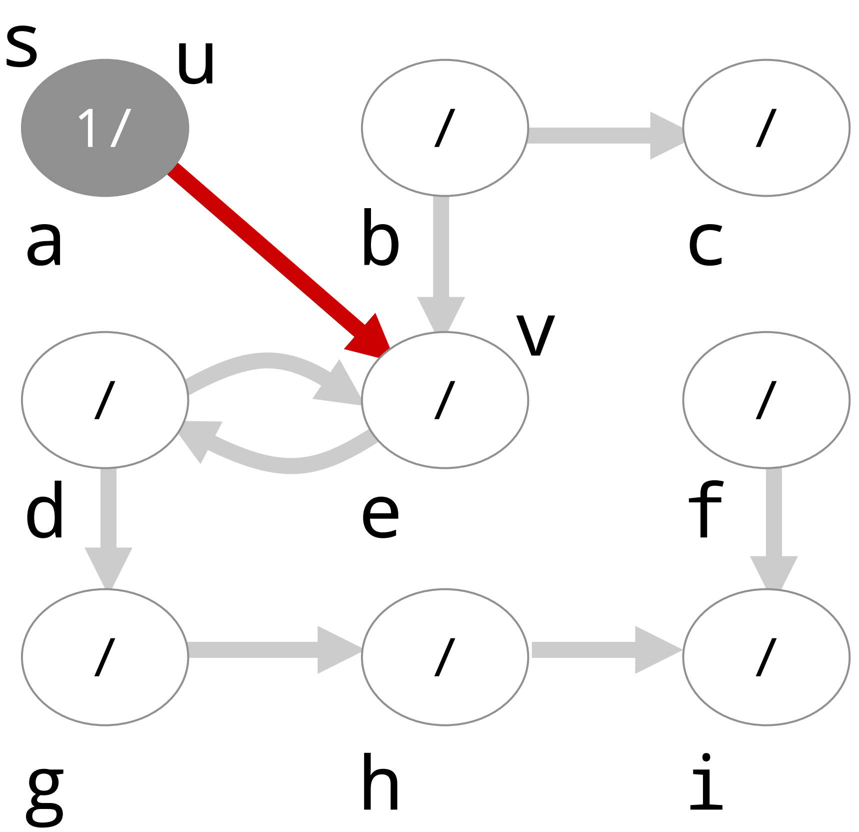
若v为白色，则递归访问v(e)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)

图G:



time: 1

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

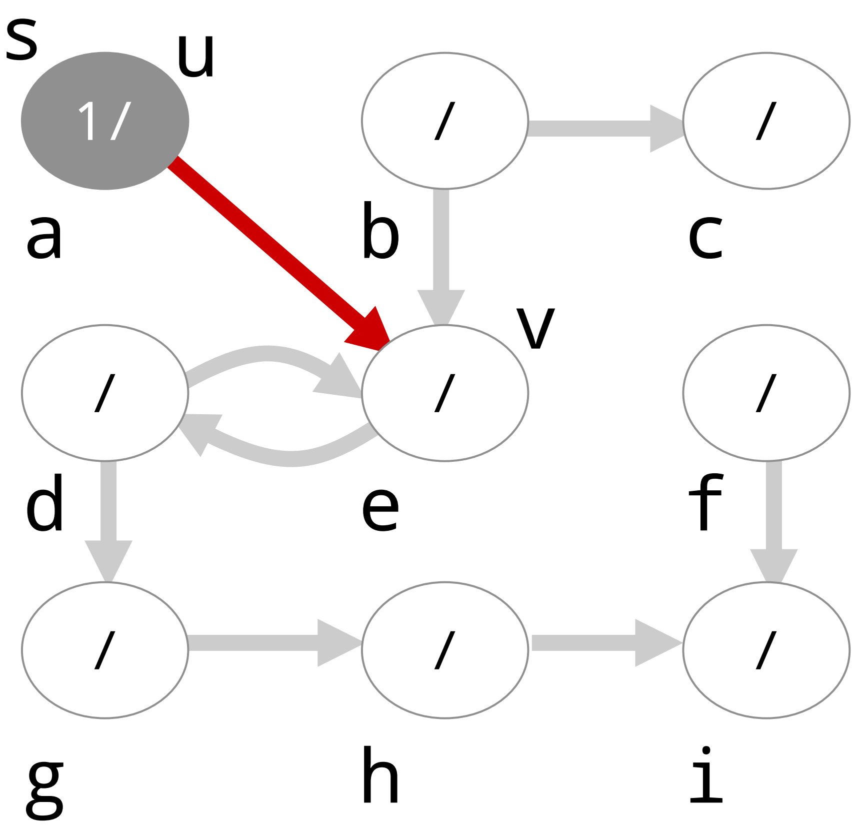
递归深度，访问邻结点v(e)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)

图G:



time: 1

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u(e)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

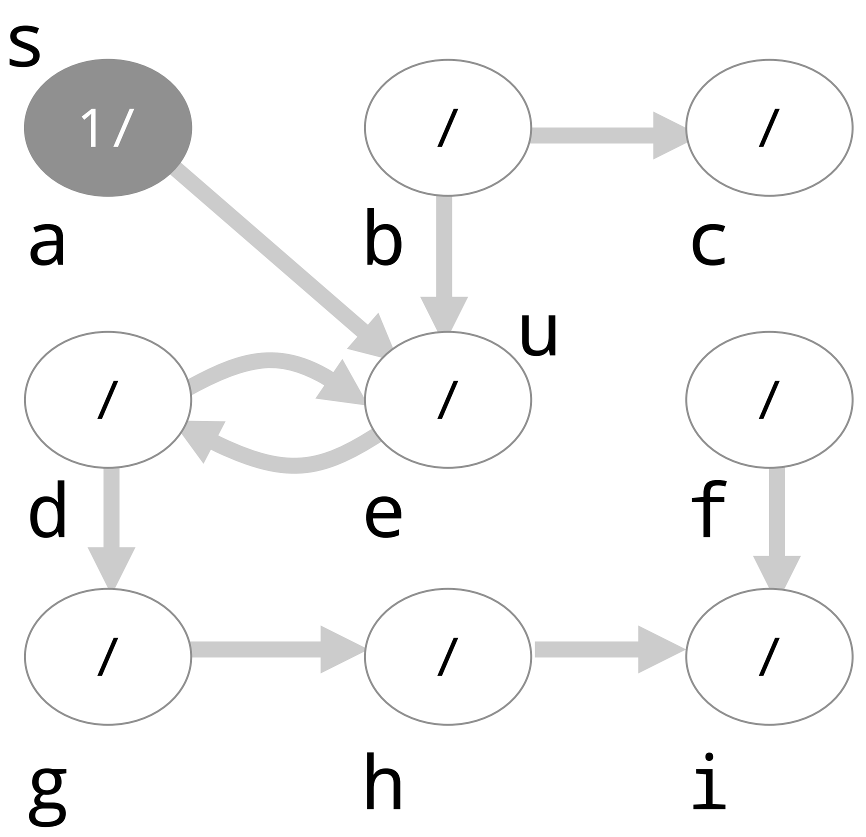
DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)



time: 1

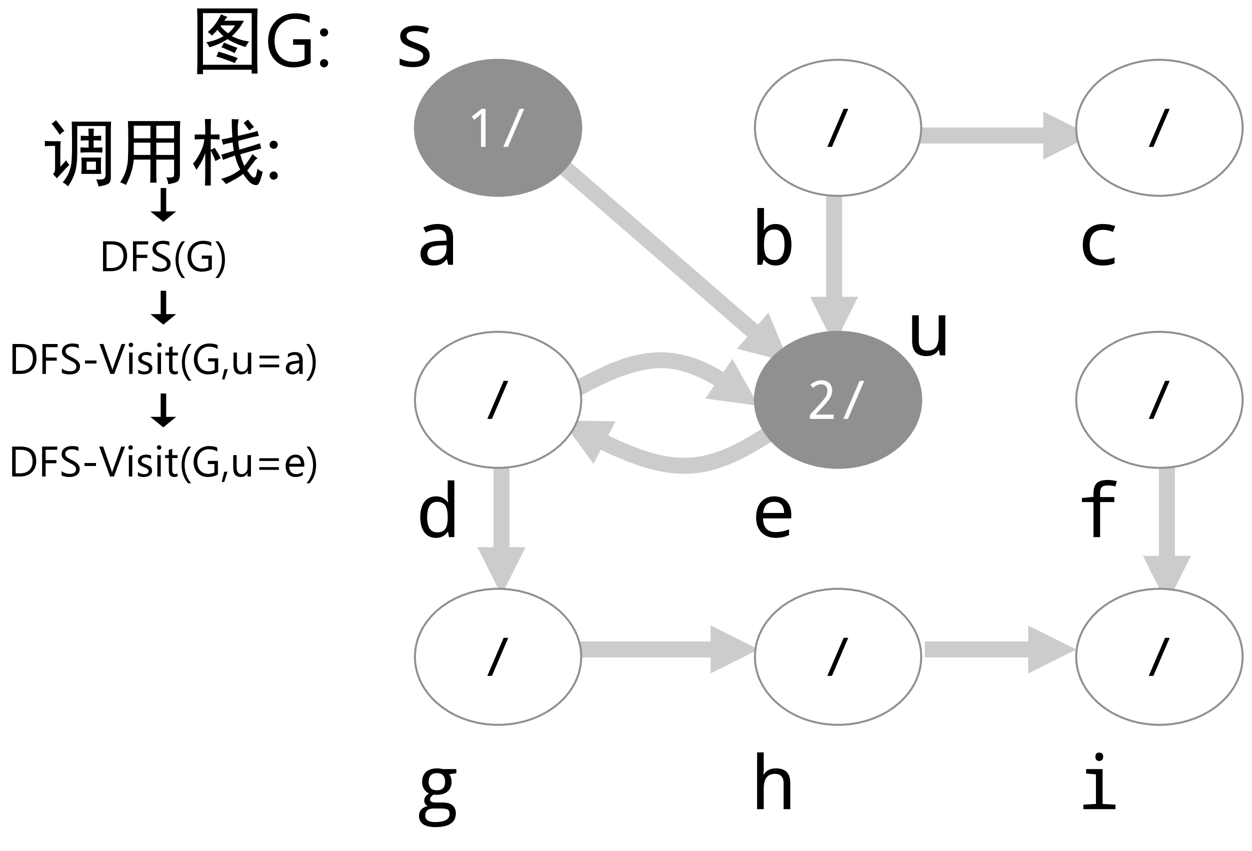
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```


图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

time: 2

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

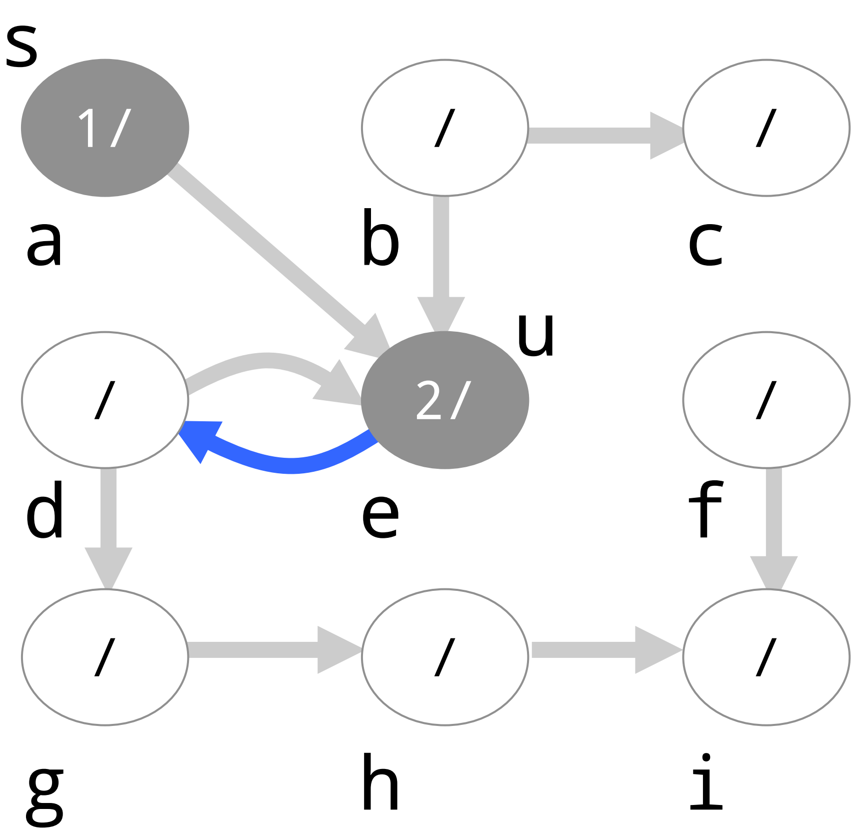
检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)

图G:



time: 2

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(d)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

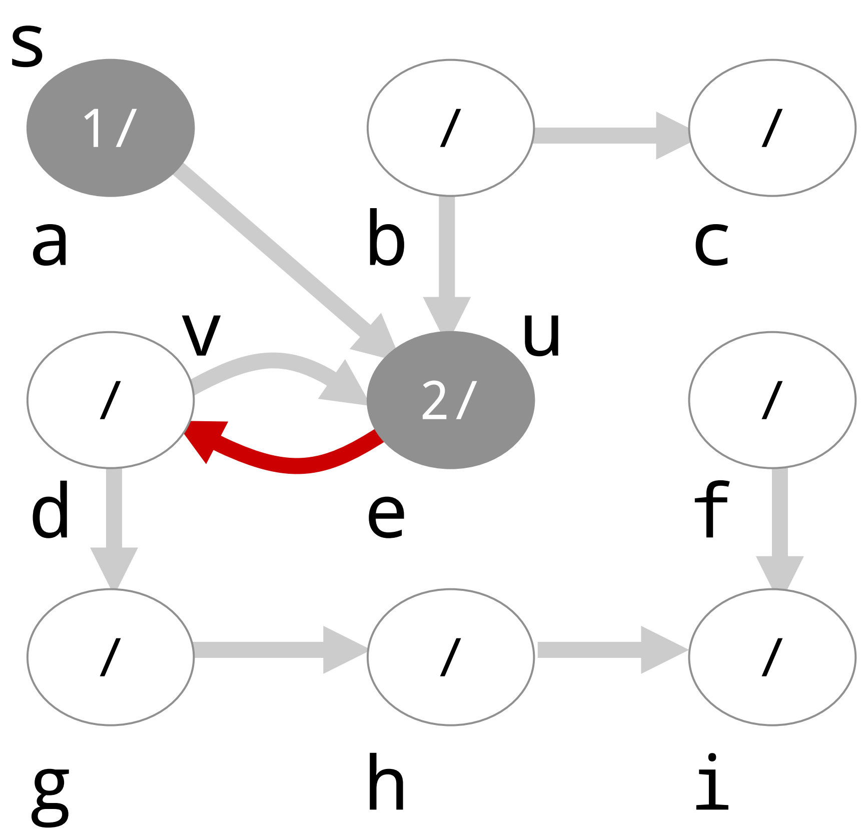
DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)



time: 2

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u(d)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

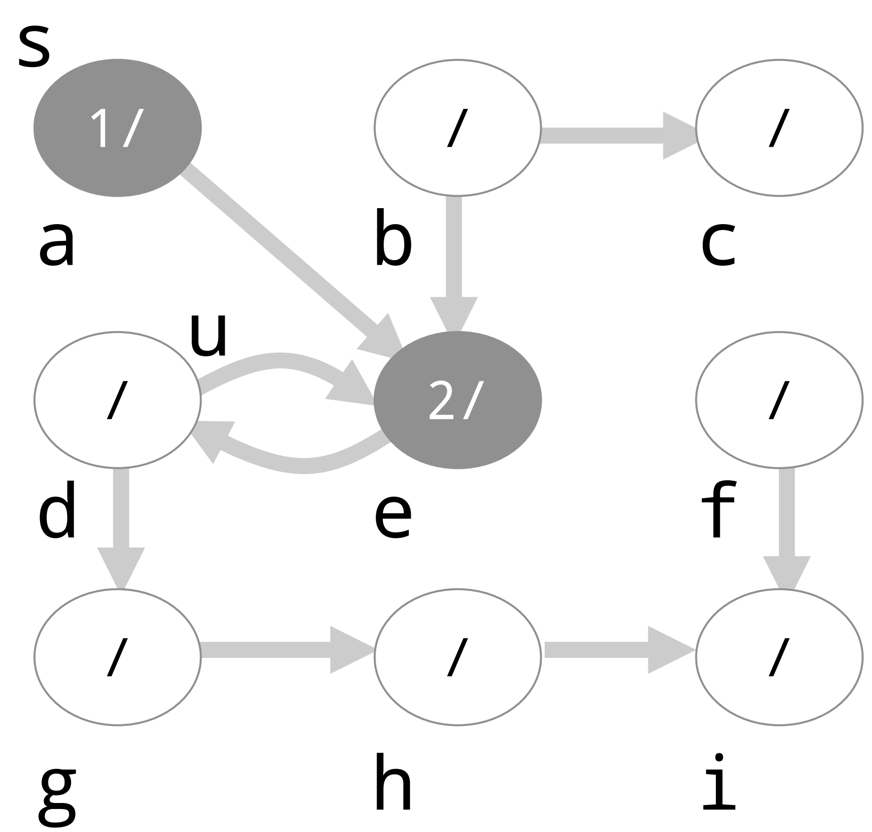
↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

图G:



time: 2

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

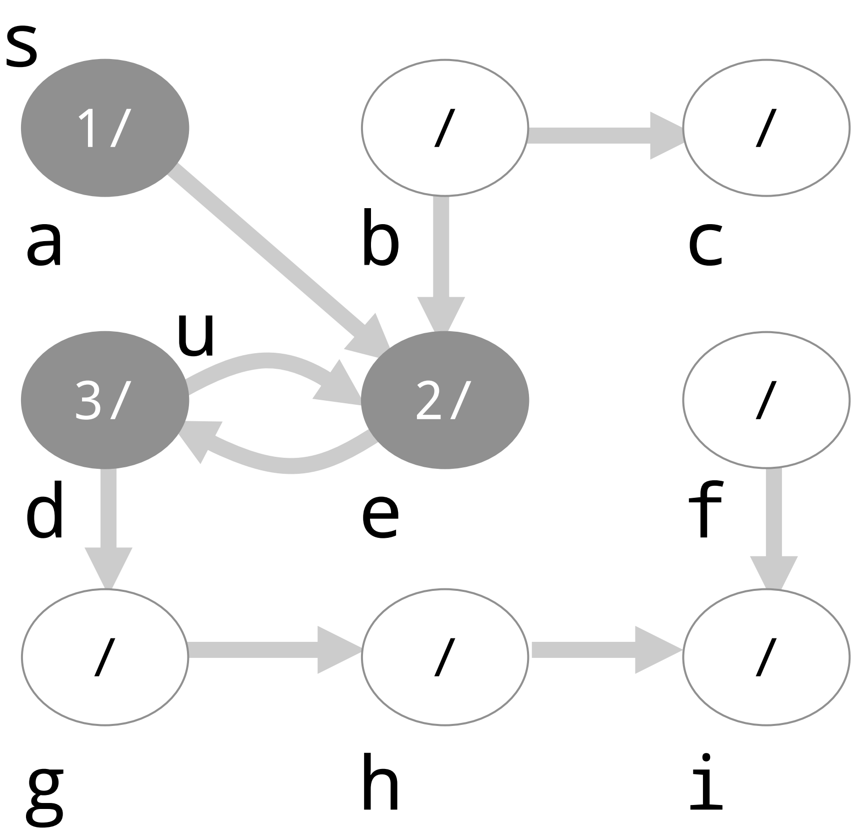
DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)



time: 3

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

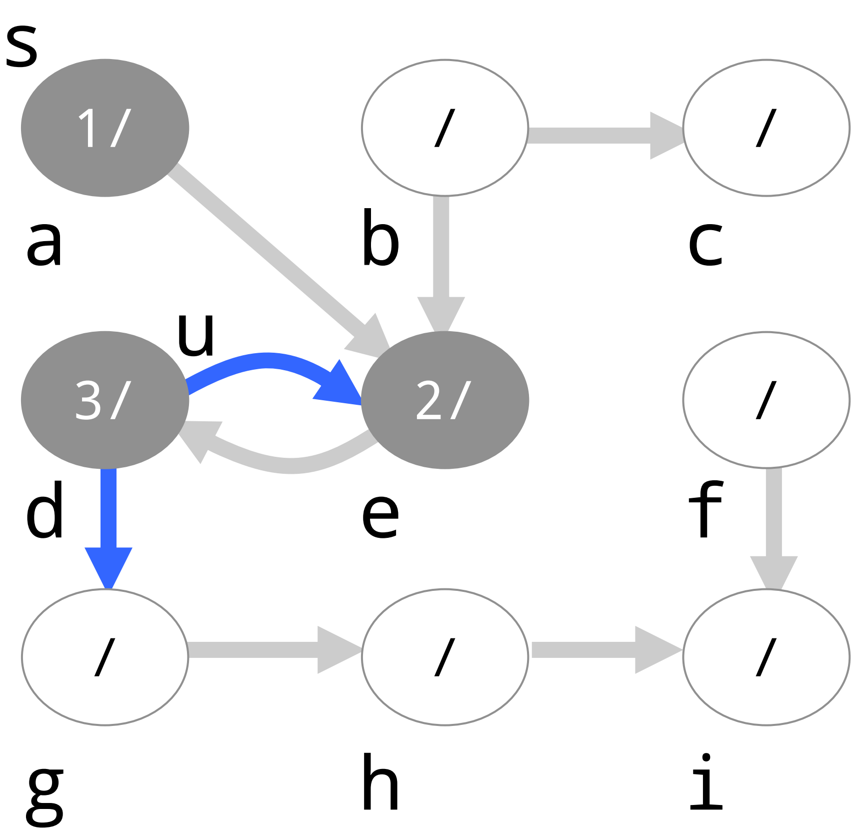
↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

图G:



time: 3

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

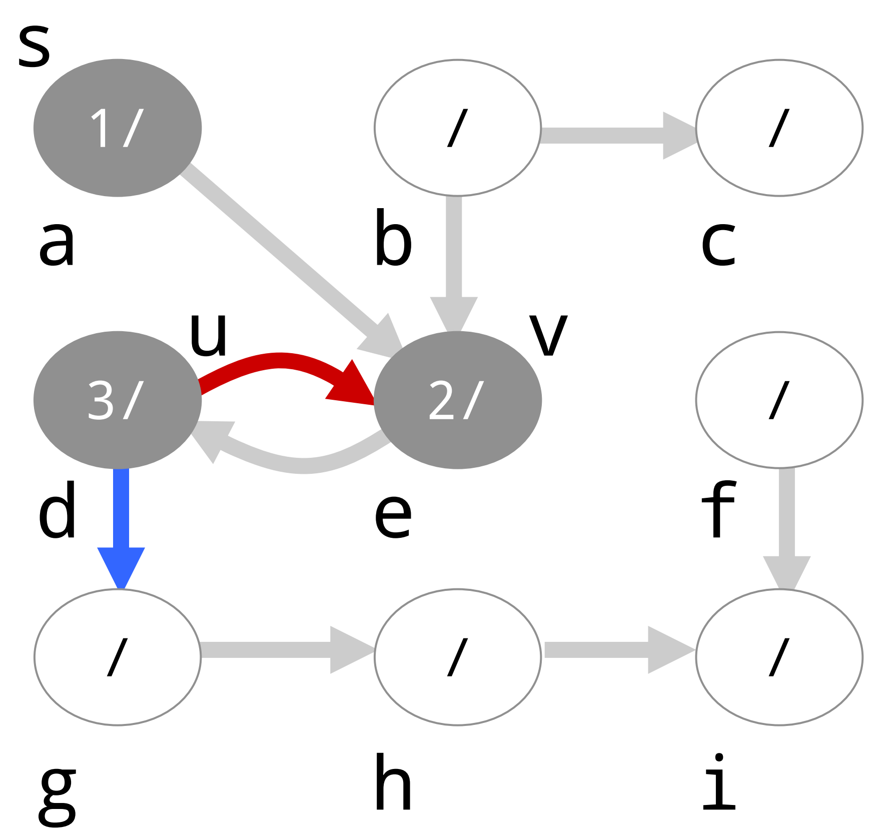
若v为白色，则递归访问v(e)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)



time: 3

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

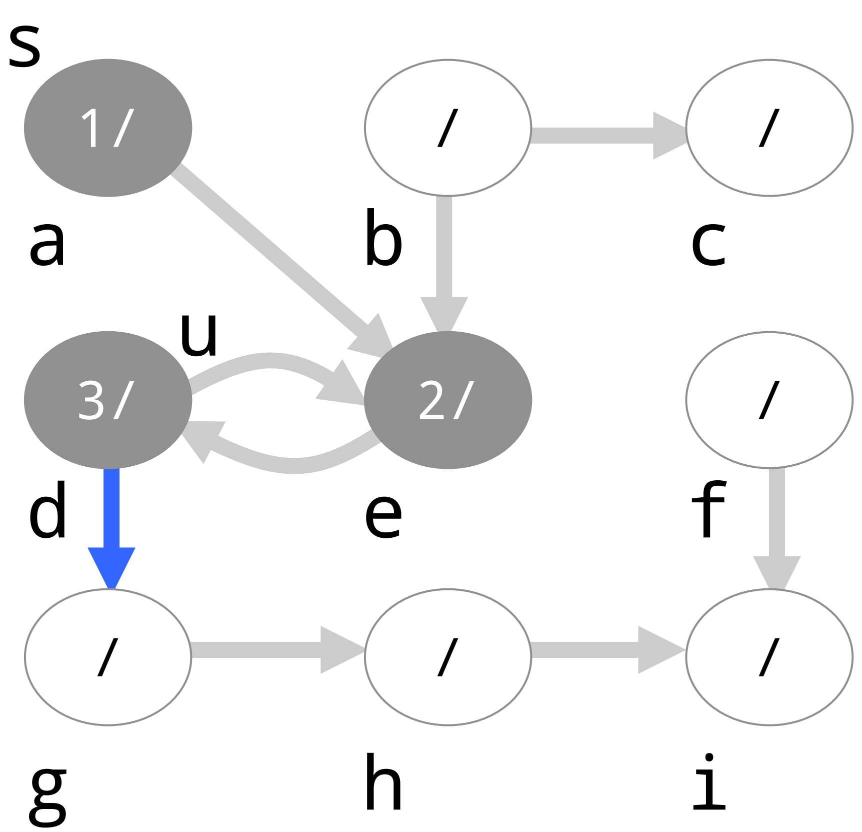
↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

图G:



time: 3

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(g)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

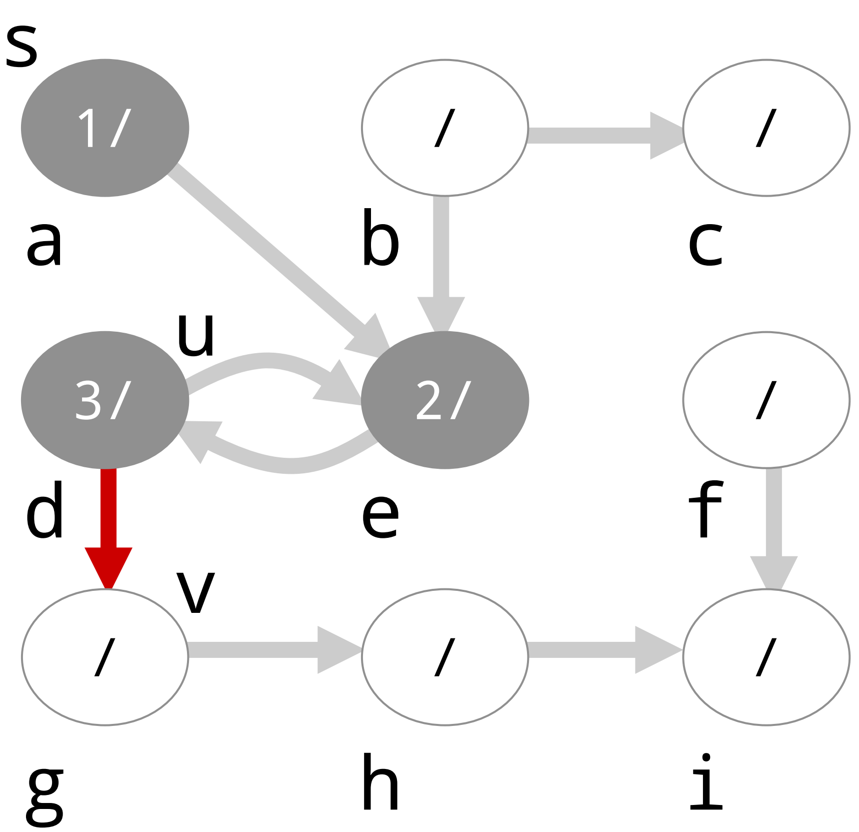
↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

图G:



time: 3

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

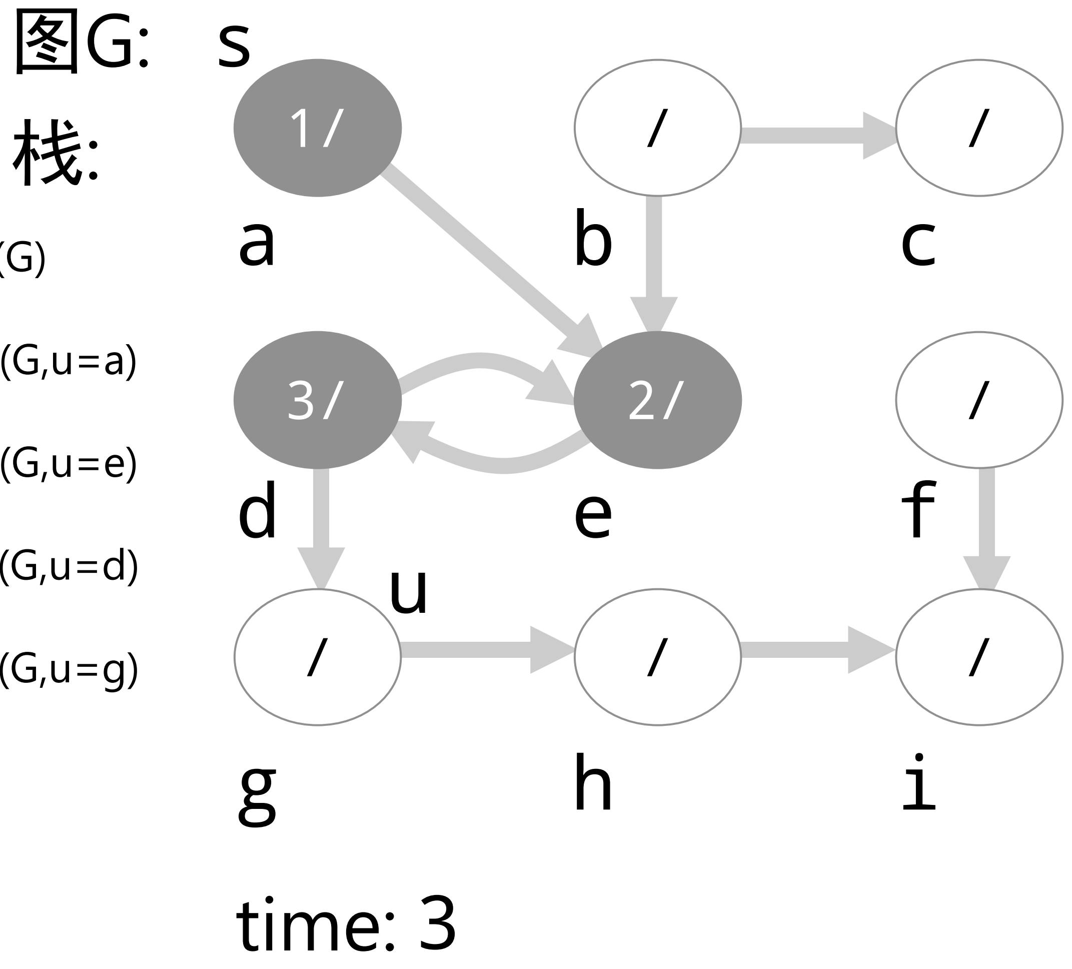
u(g)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)



11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

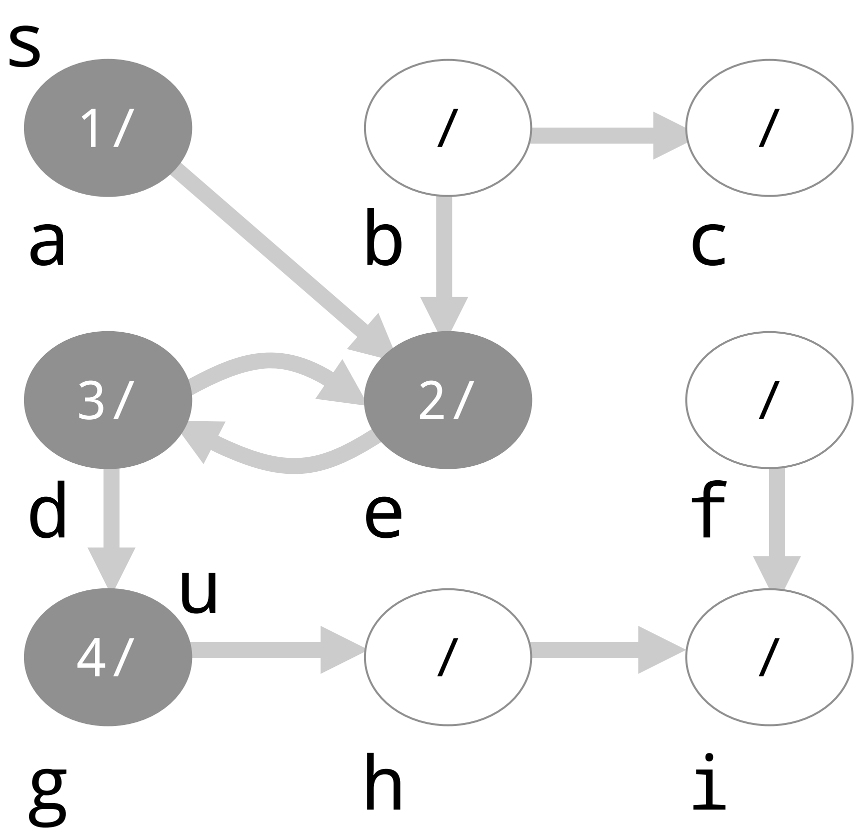
记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)



time: 4

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

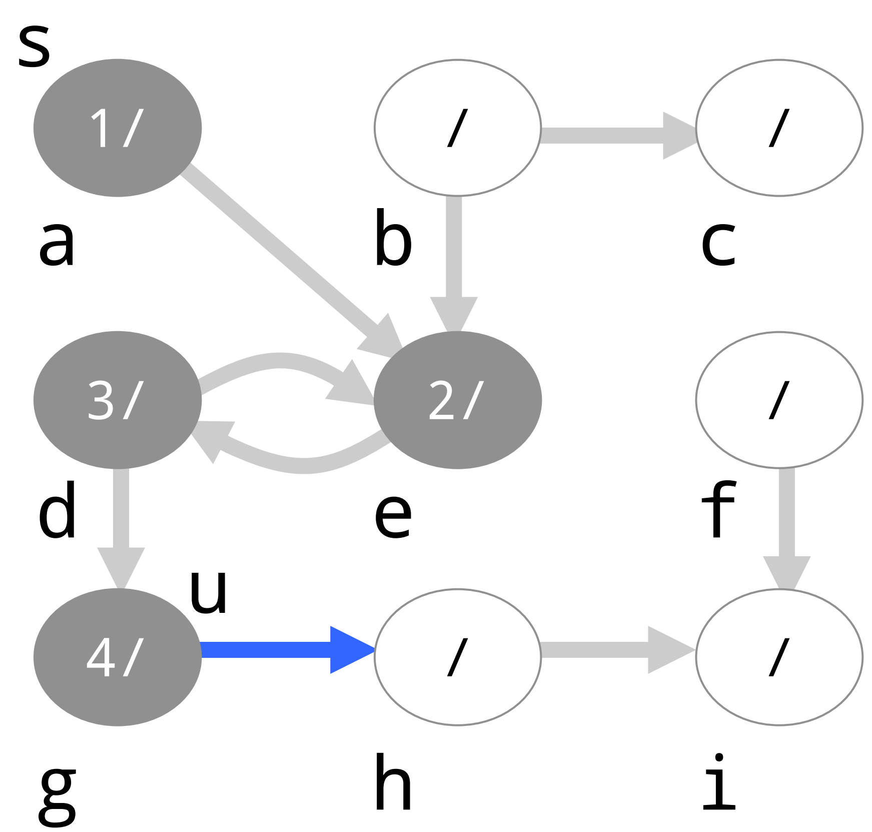
↓ DFS(G)

↓ DFS-Visit(G,u=a)

↓ DFS-Visit(G,u=e)

↓ DFS-Visit(G,u=d)

↓ DFS-Visit(G,u=g)



time: 4

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(h)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

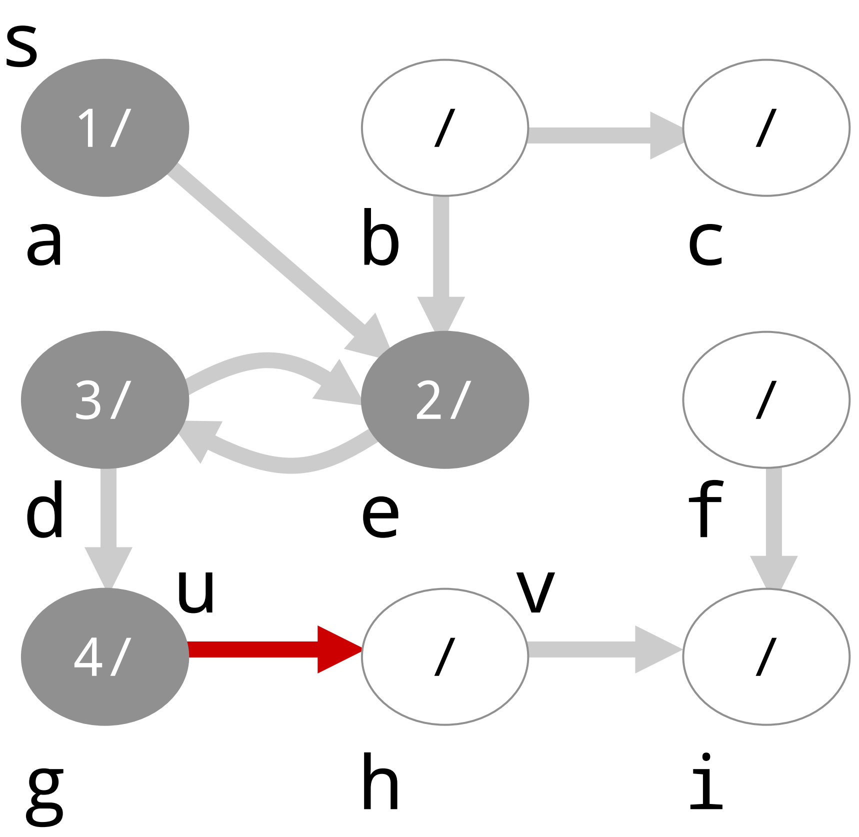
DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)



time: 4

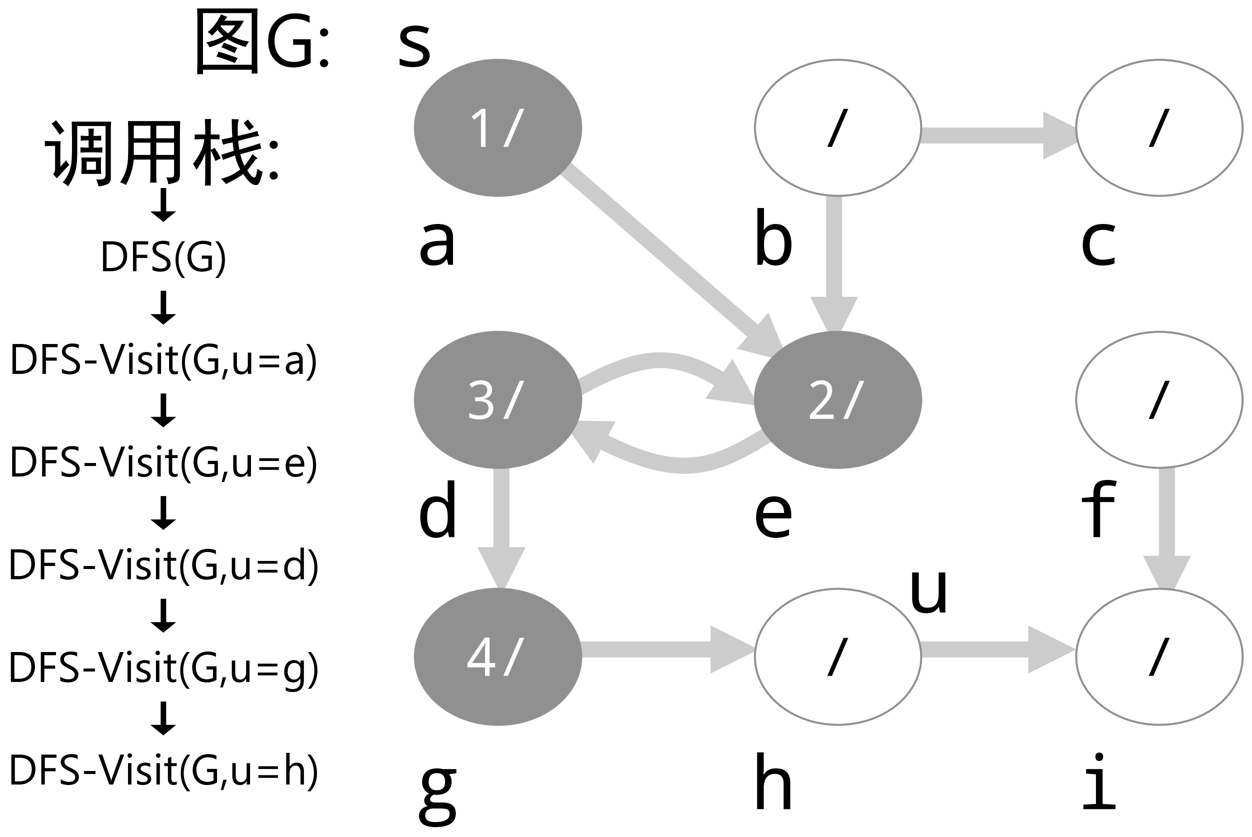
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u(h)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



time: 4

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

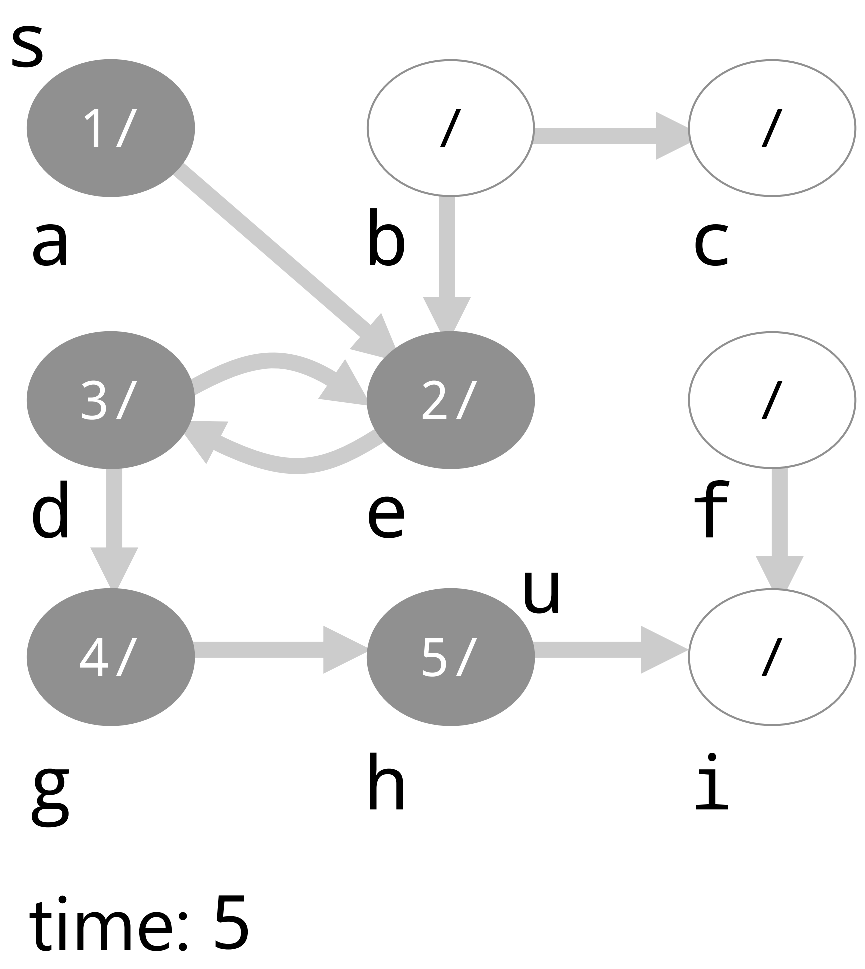
记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)
↓
DFS-Visit(G,u=h)

图G:



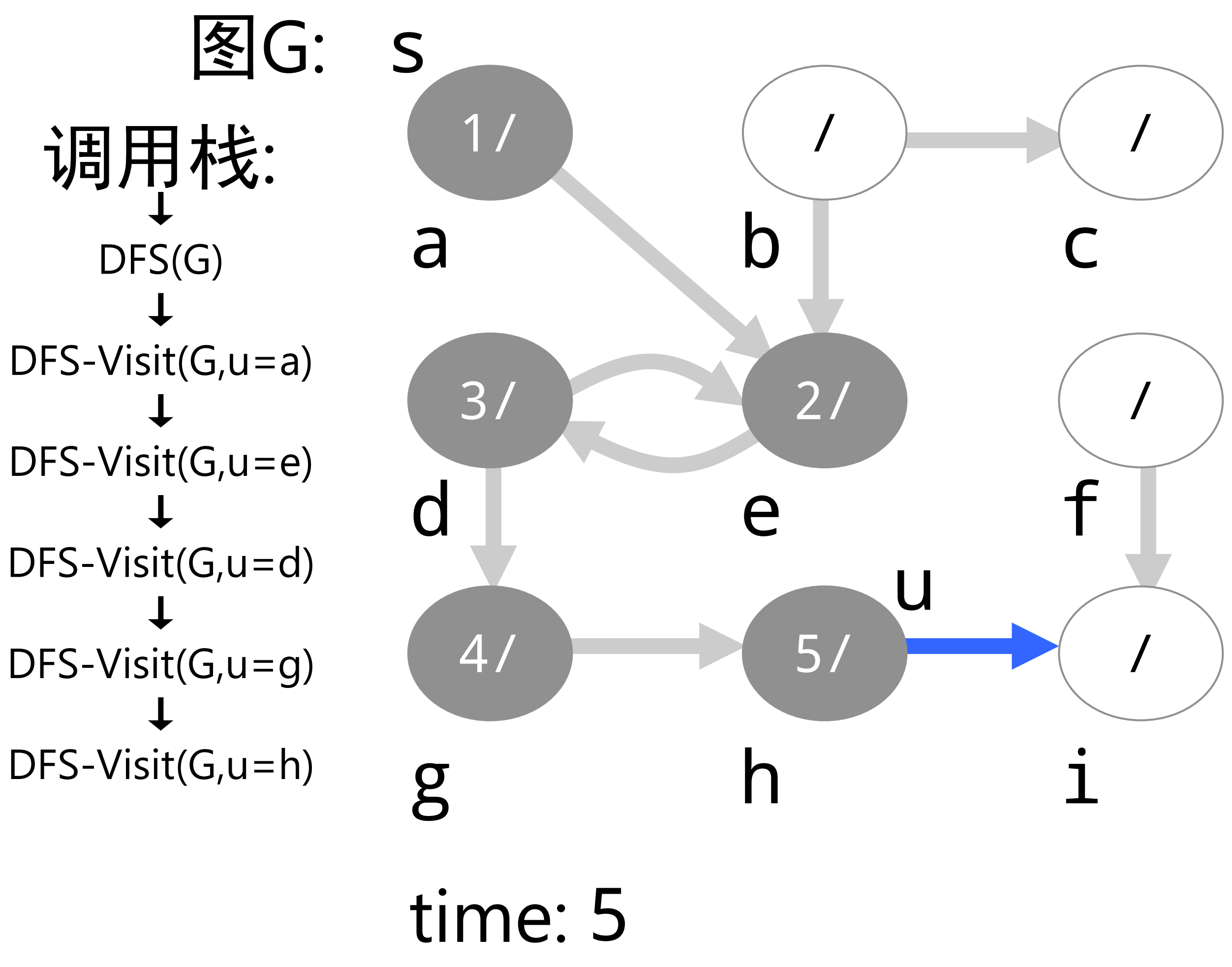
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



图G:

调用栈:

↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)
↓
DFS-Visit(G,u=h)

time: 5

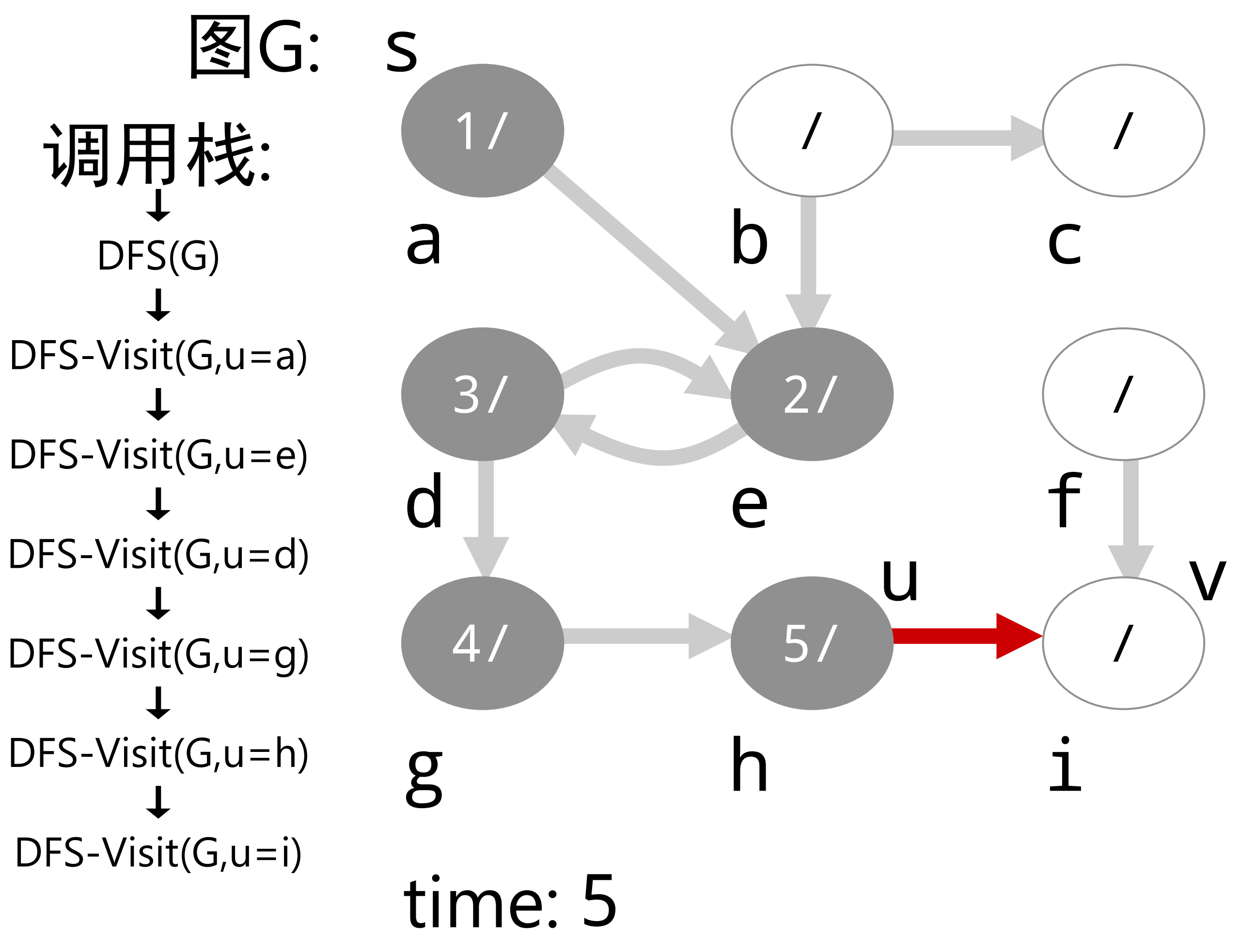
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(i)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



图G:

调用栈:

DFS(G)

DFS-Visit(G,u=a)

DFS-Visit(G,u=e)

DFS-Visit(G,u=d)

DFS-Visit(G,u=g)

DFS-Visit(G,u=h)

DFS-Visit(G,u=i)

time: 5

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u(i)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

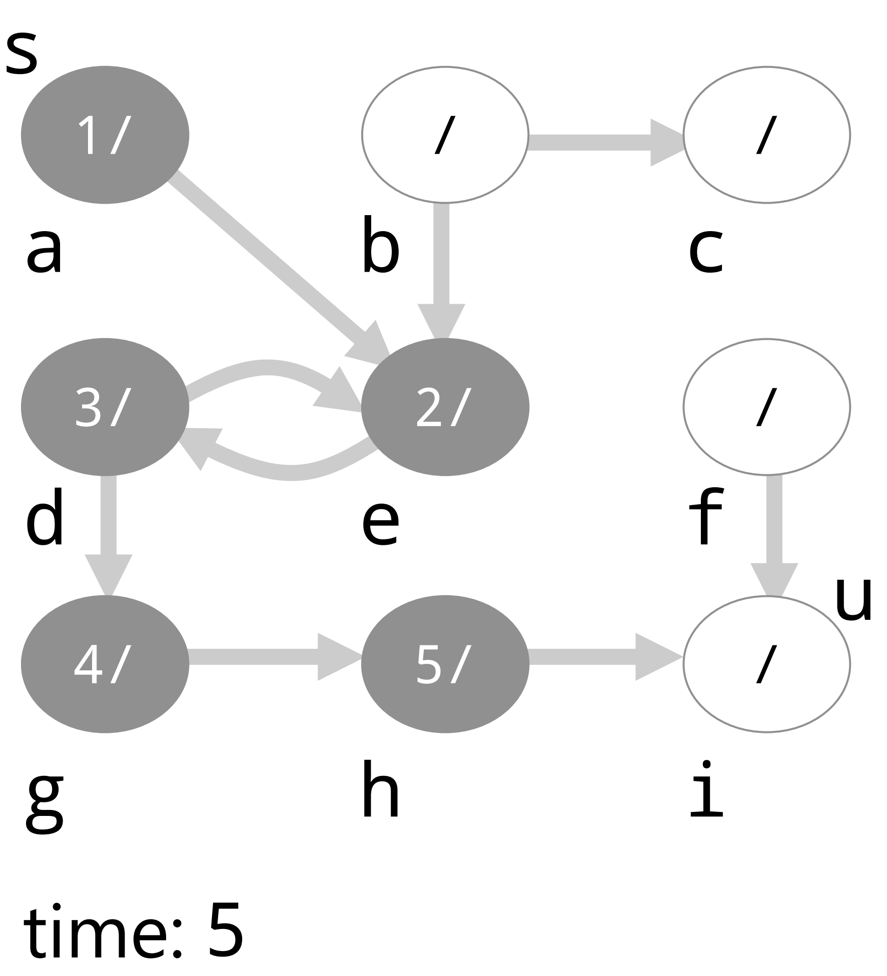
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)
↓
DFS-Visit(G,u=h)
↓
DFS-Visit(G,u=i)

图G:



11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)

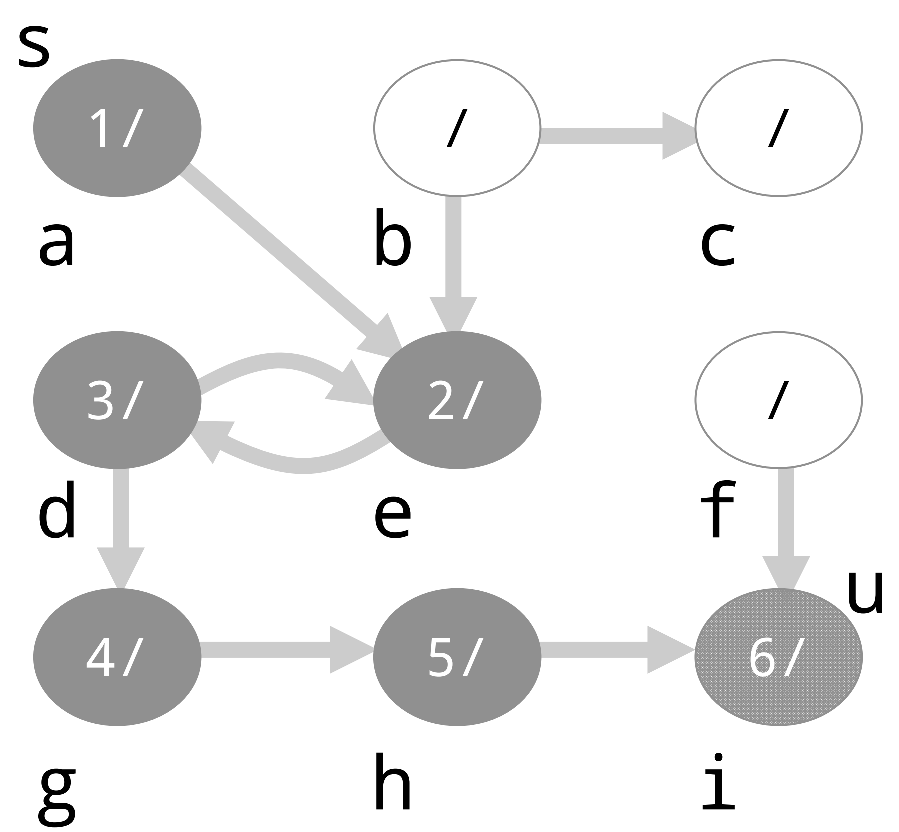
↓

DFS-Visit(G,u=h)

↓

DFS-Visit(G,u=i)

图G:



time: 6

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)

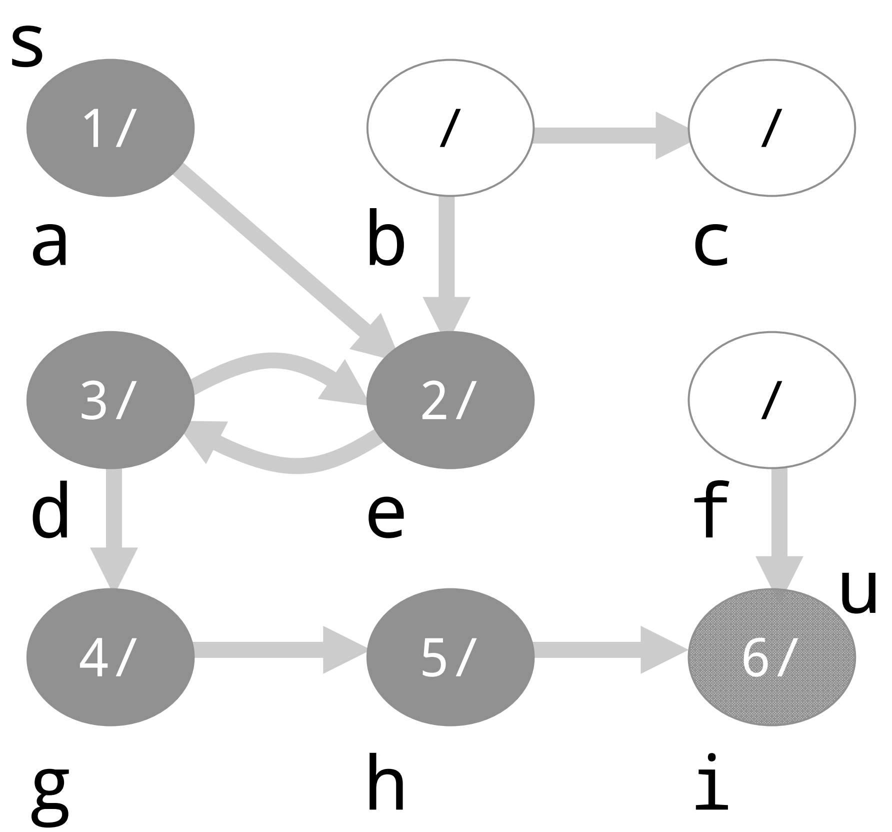
↓

DFS-Visit(G,u=h)

↓

DFS-Visit(G,u=i)

图G:



time: 6

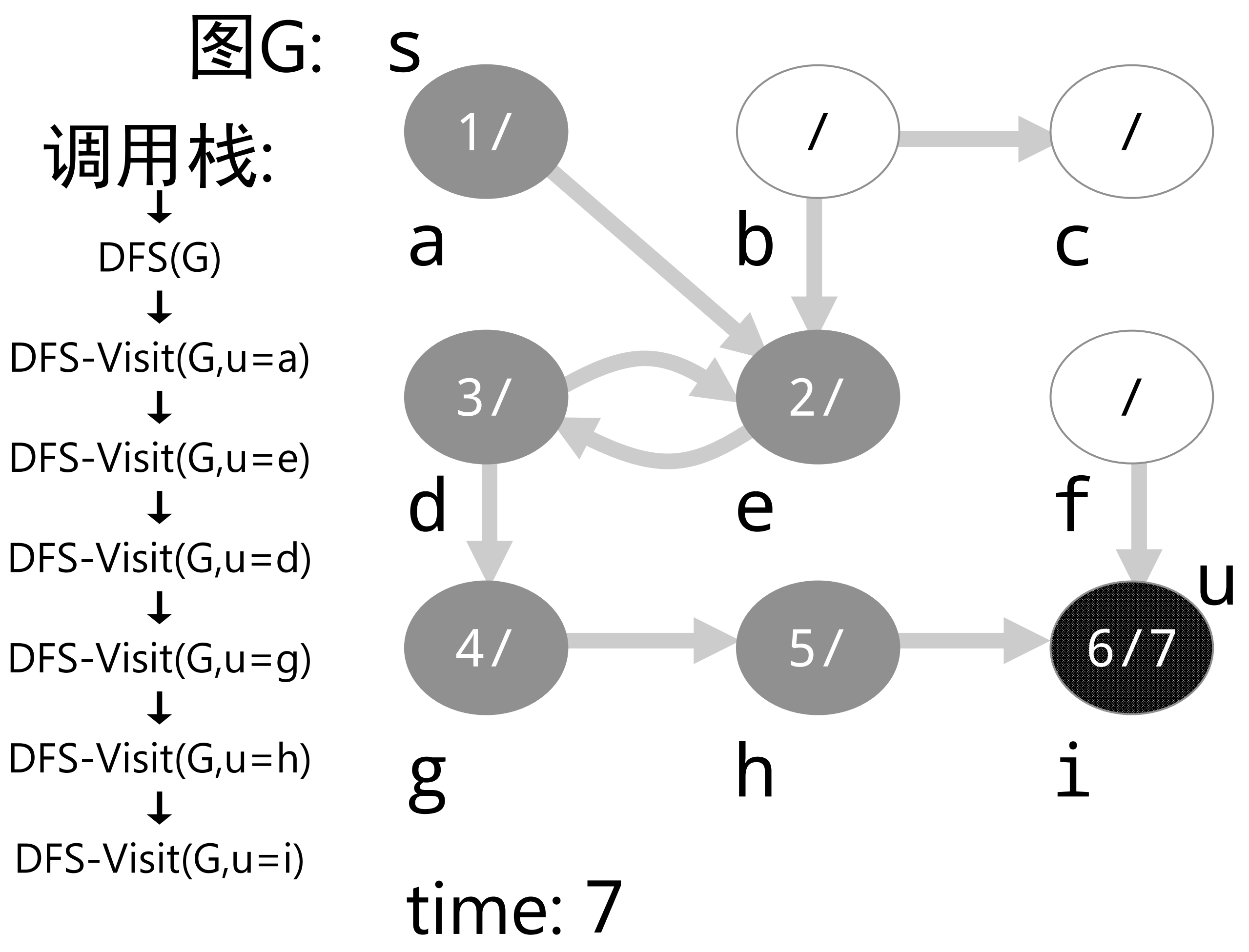
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



图G:

调用栈:

DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)
↓
DFS-Visit(G,u=g)
↓
DFS-Visit(G,u=h)
↓
DFS-Visit(G,u=i)

time: 7

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束邻结点v(i)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

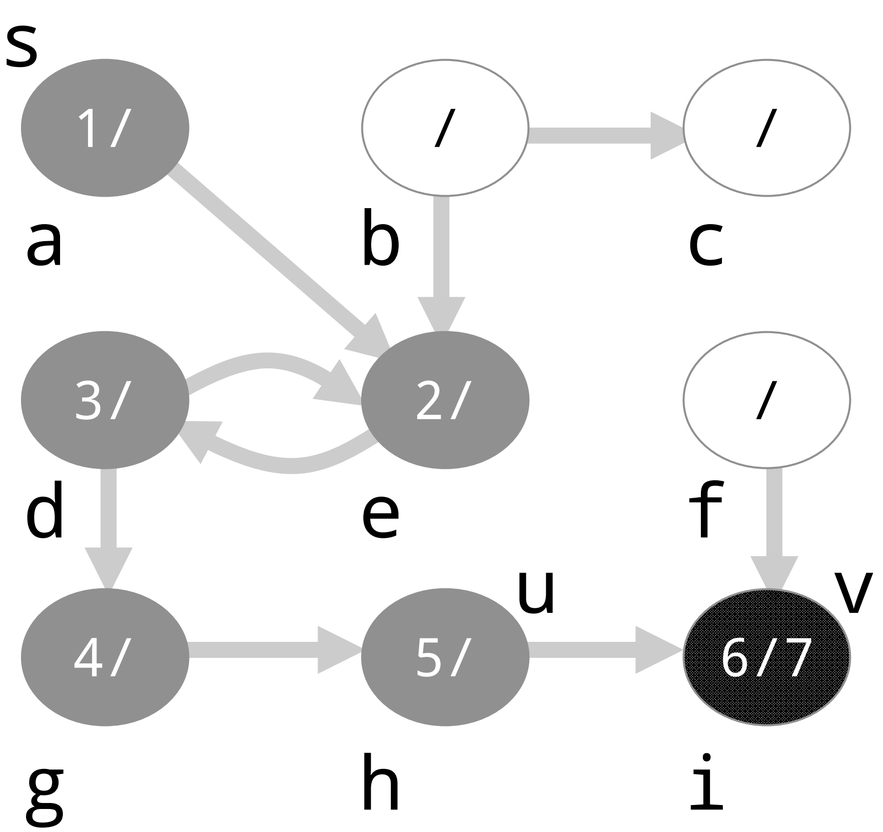
DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)

↓

DFS-Visit(G,u=h)



time: 7

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

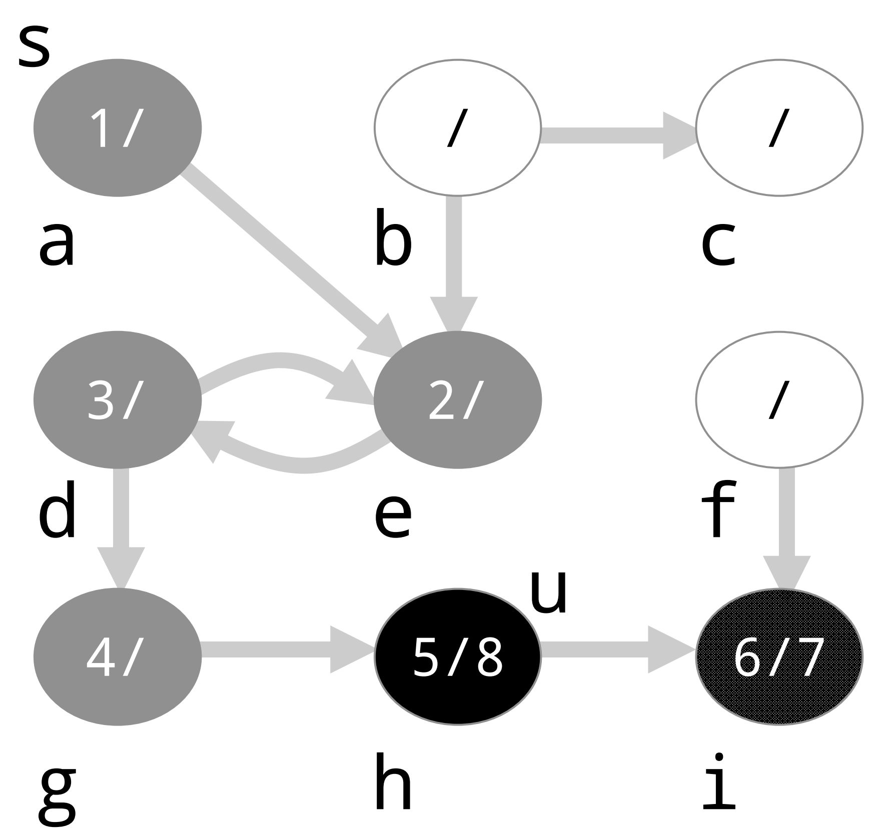
DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)

↓

DFS-Visit(G,u=h)



time: 8

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束邻结点v(h)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

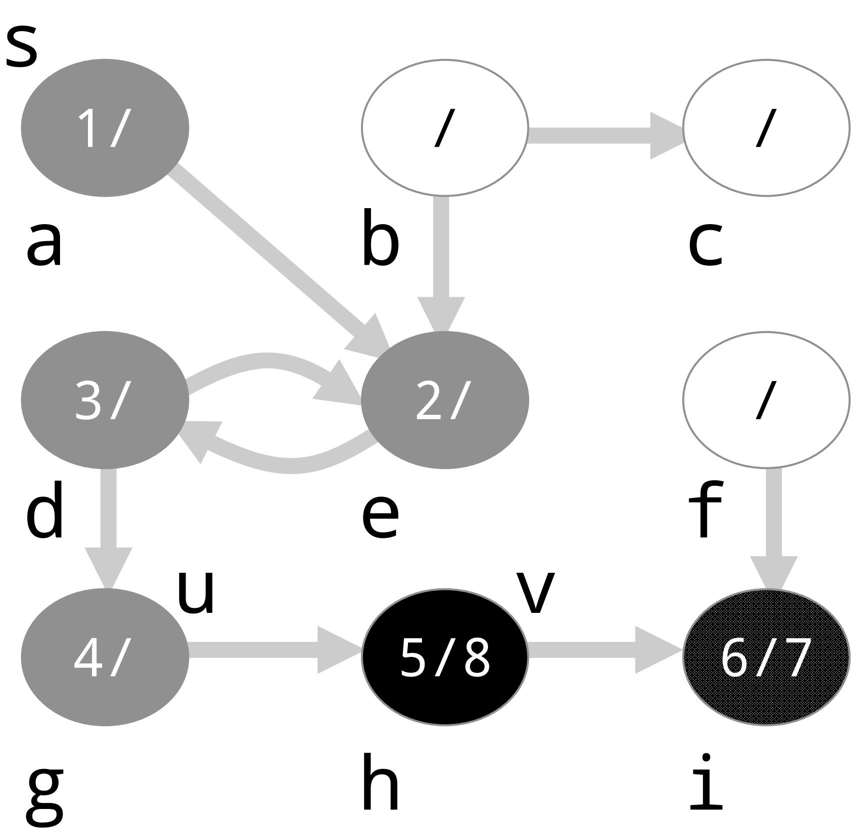
DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)



time: 8

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

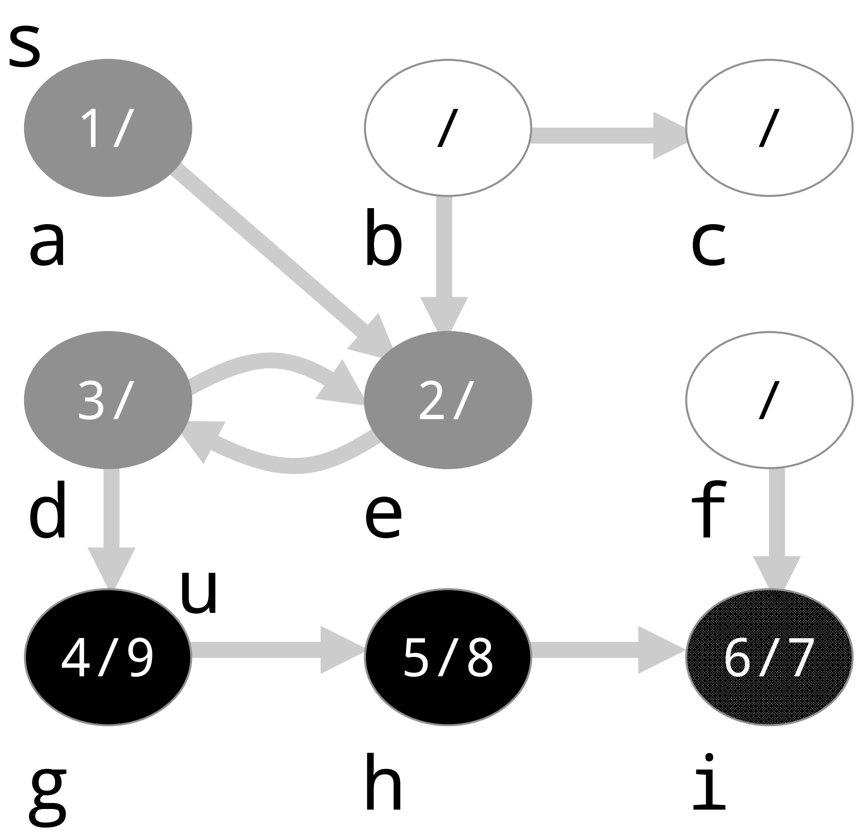
DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)

↓

DFS-Visit(G,u=g)



time: 9

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

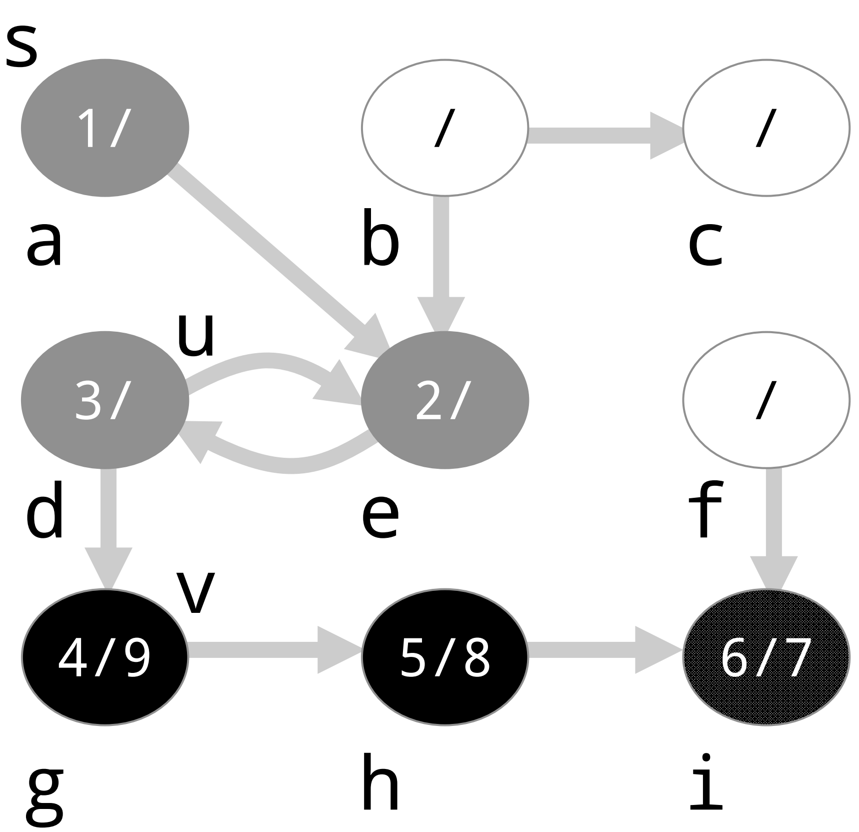
递归回退，结束邻结点v(g)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)
↓
DFS-Visit(G,u=d)



time: 9

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)

↓

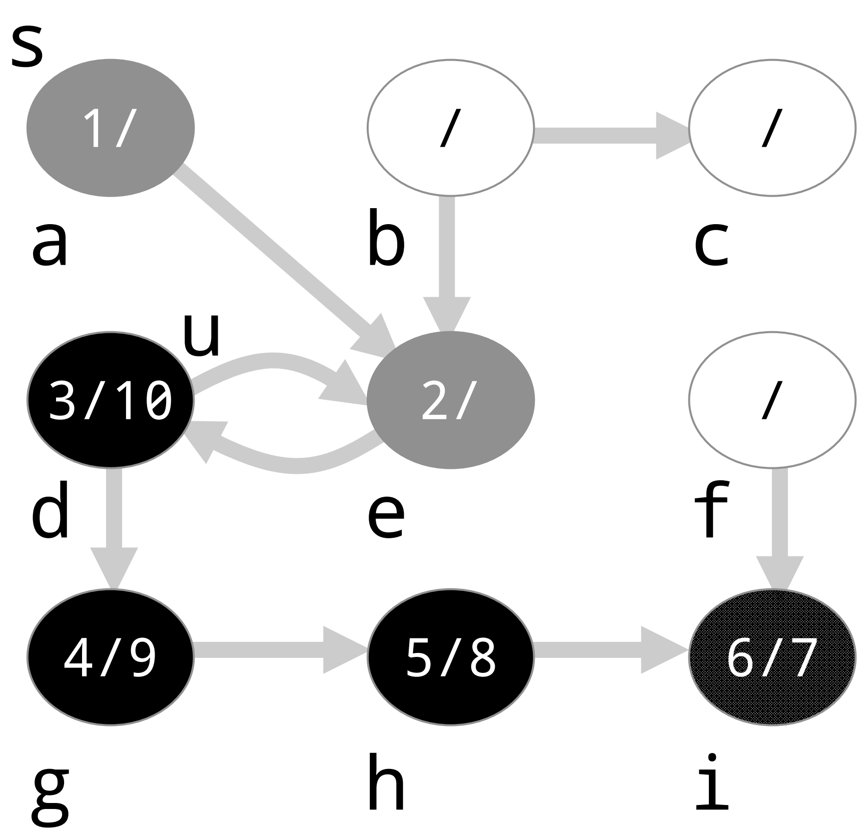
DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

↓

DFS-Visit(G,u=d)



time: 10

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束邻结点v(d)访问

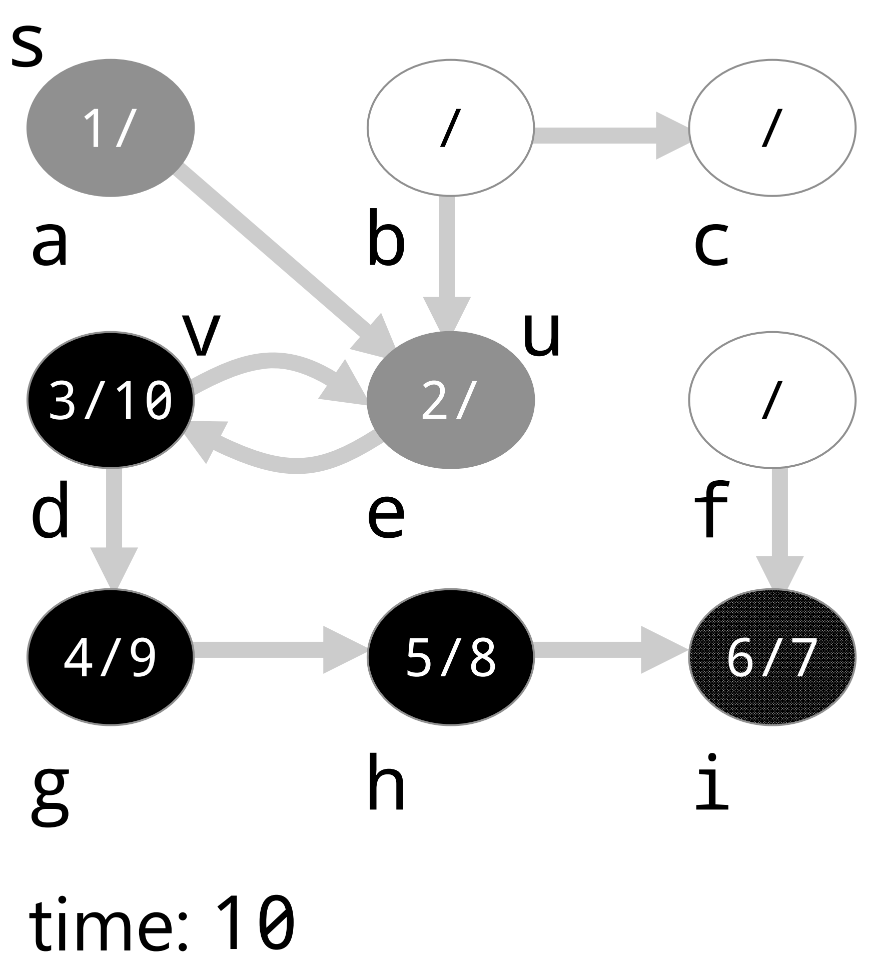
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

↓
DFS(G)
↓
DFS-Visit(G,u=a)
↓
DFS-Visit(G,u=e)

图G:



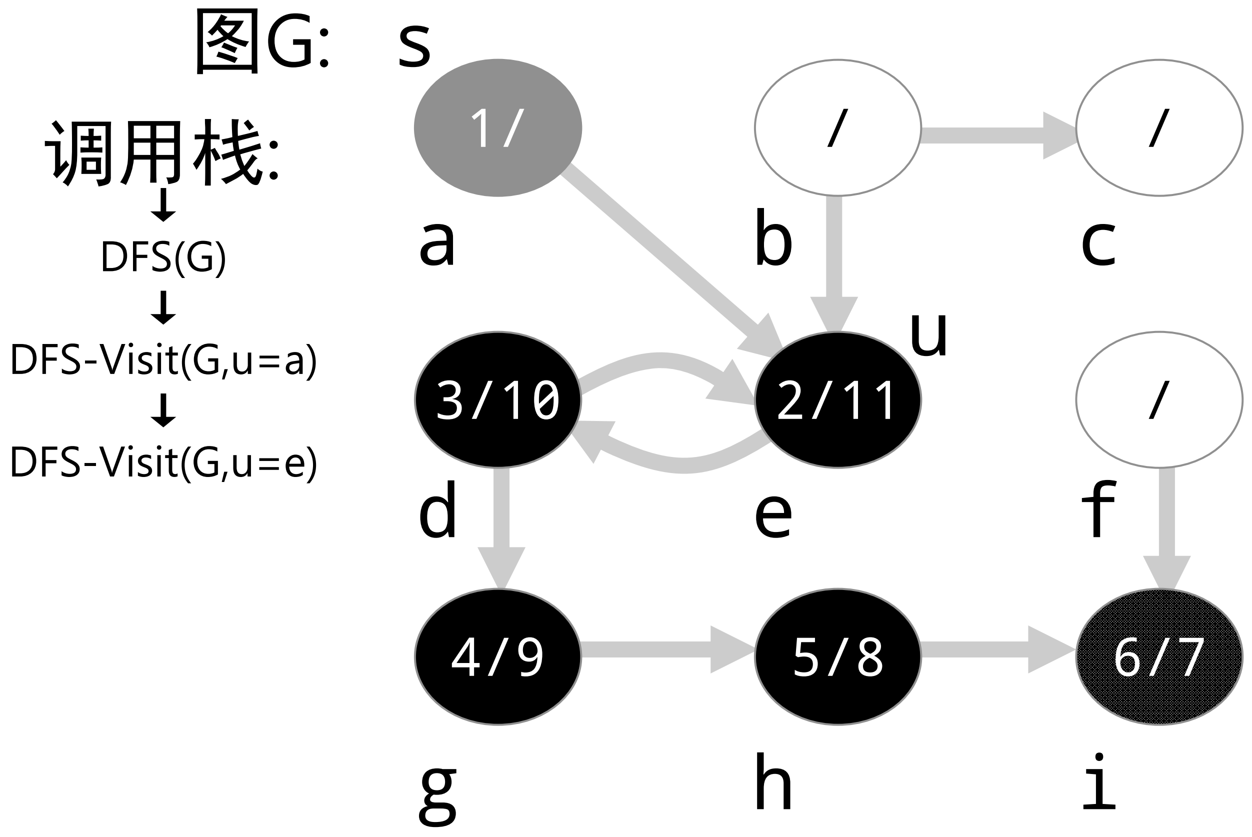
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



调用栈:

↓

DFS(G)

↓

DFS-Visit(G,u=a)

↓

DFS-Visit(G,u=e)

time: 11

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束邻结点v(e)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

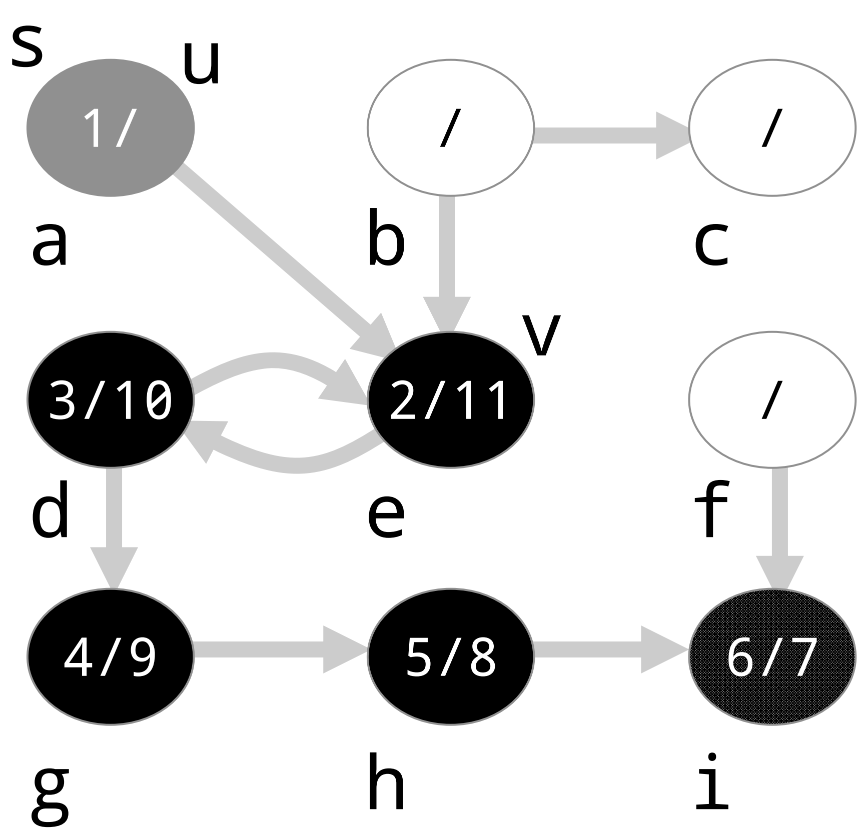
↓

DFS(G)

↓

DFS-Visit(G,u=a)

图G:



time: 11

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

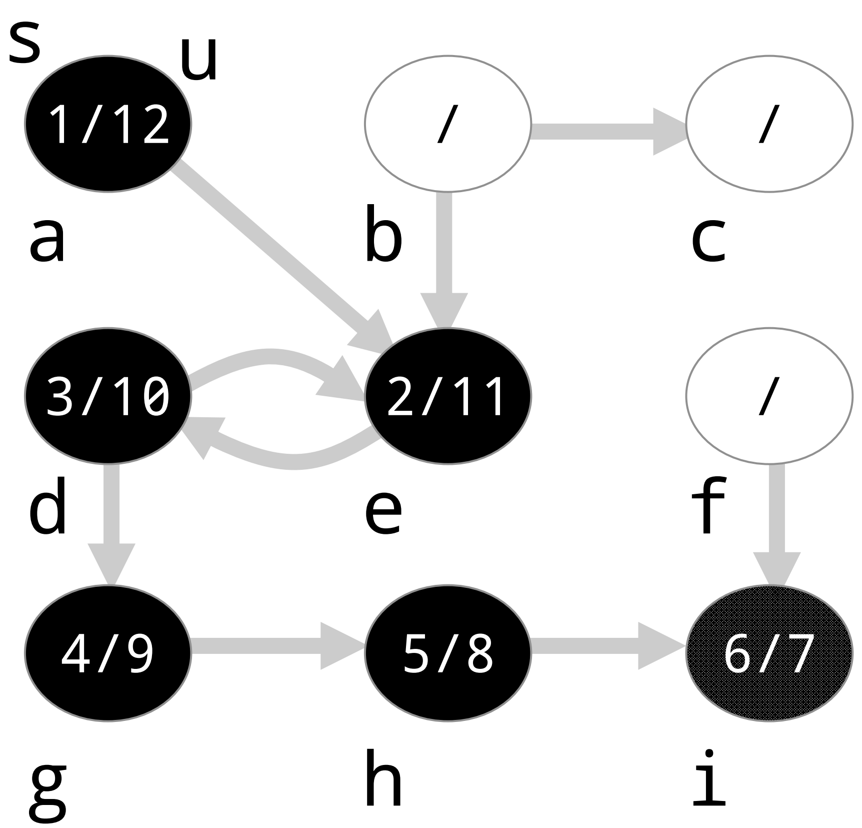
u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=a)

图G:



time: 12

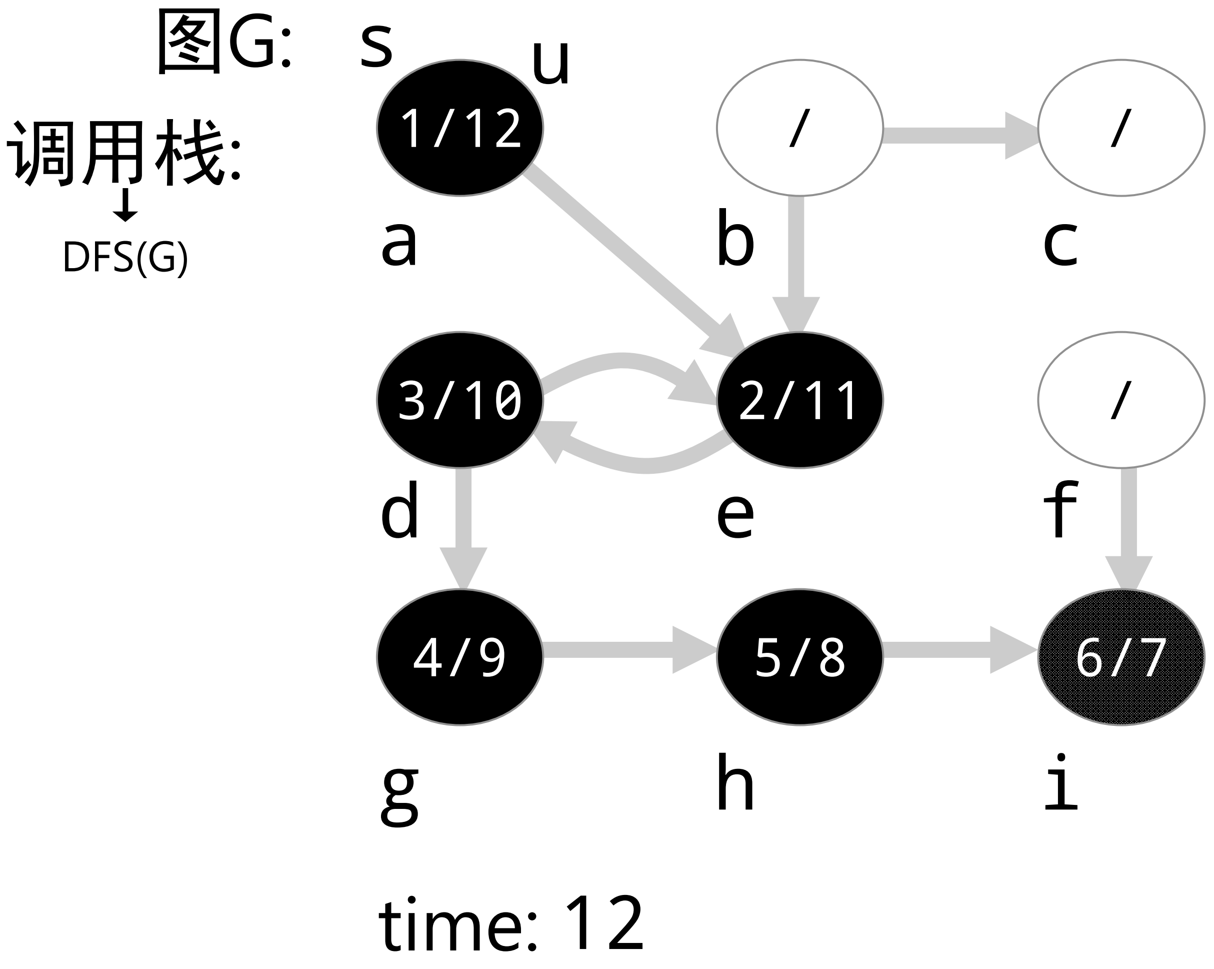
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束u(a)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

发现白色顶点时，调用DFS-VISIT访问该顶点u(b)

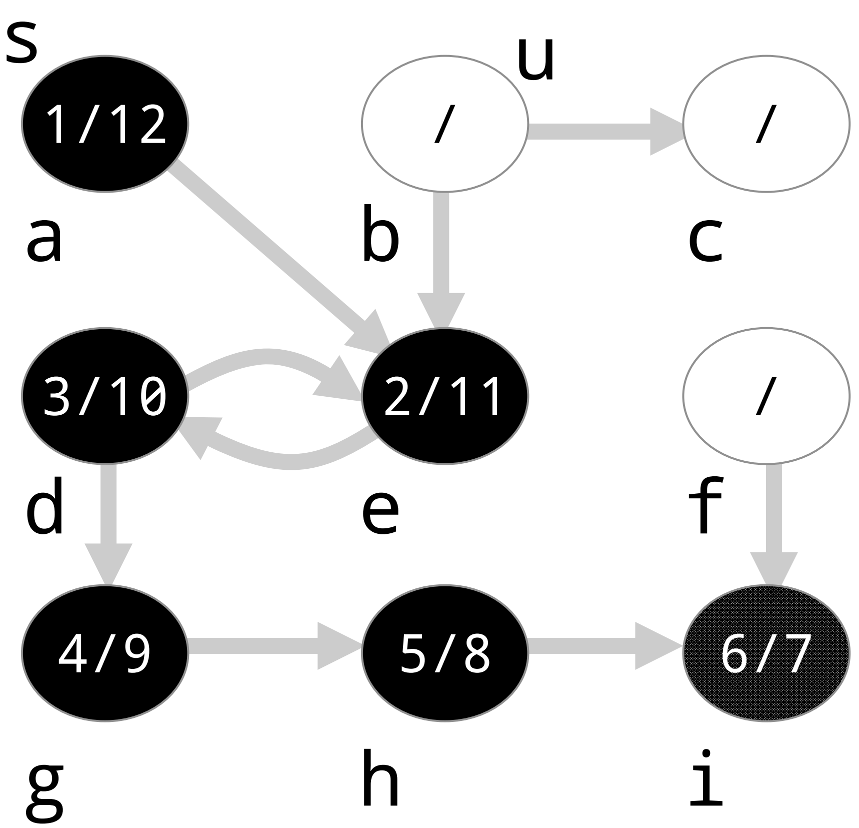
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

DFS(G)



time: 12

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

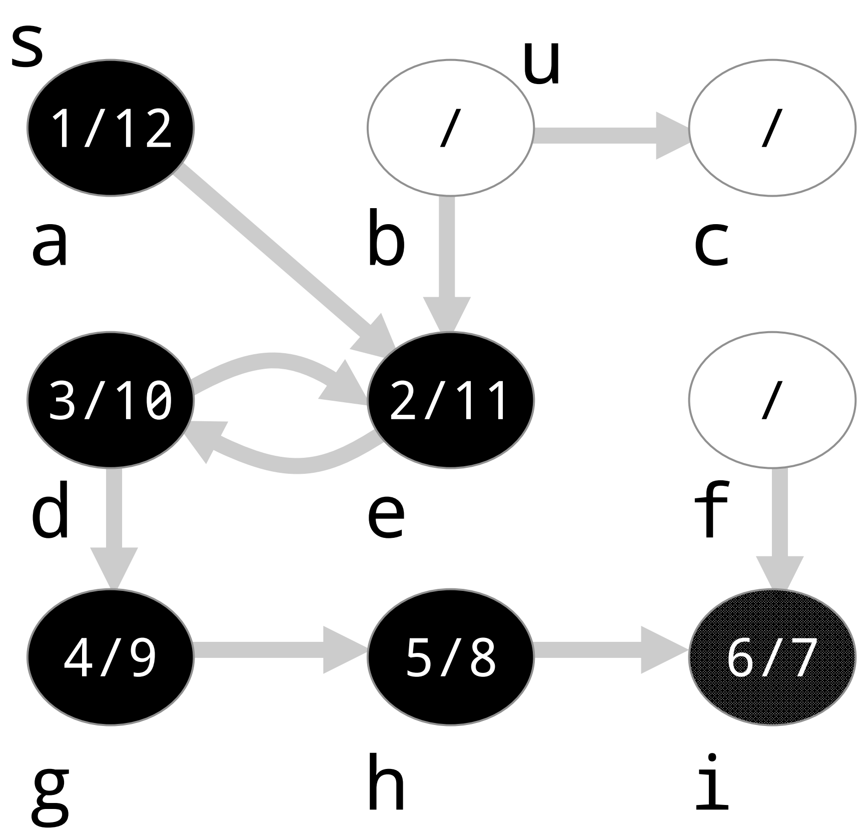
u(b)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)



time: 12

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

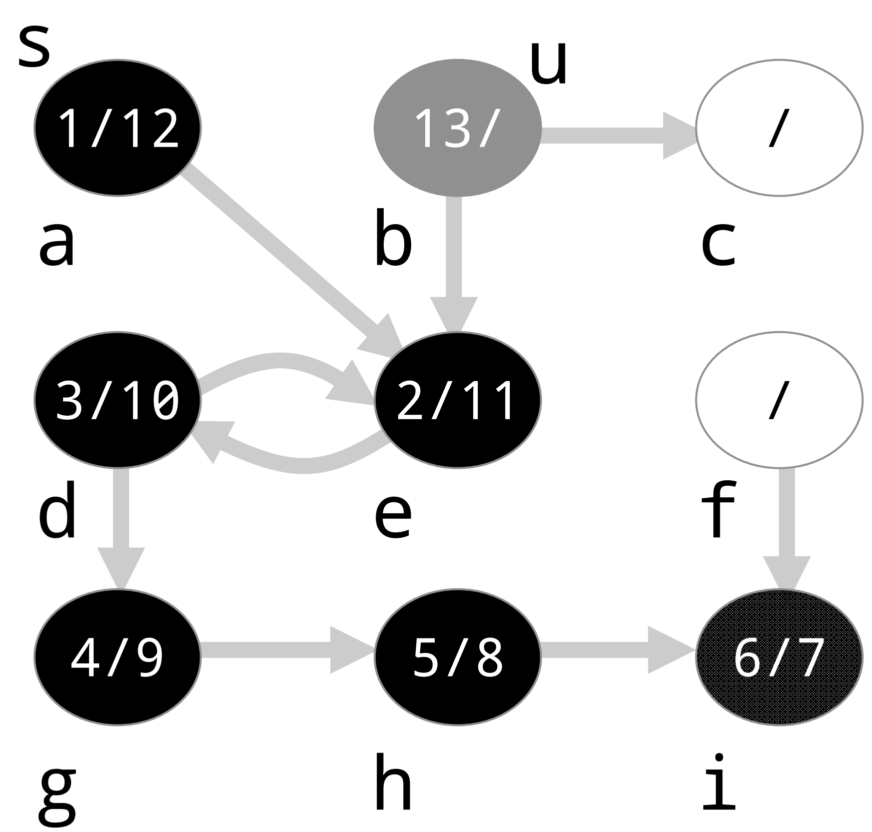
↓

DFS(G)

↓

DFS-Visit(G,u=b)

图G:



time: 13

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

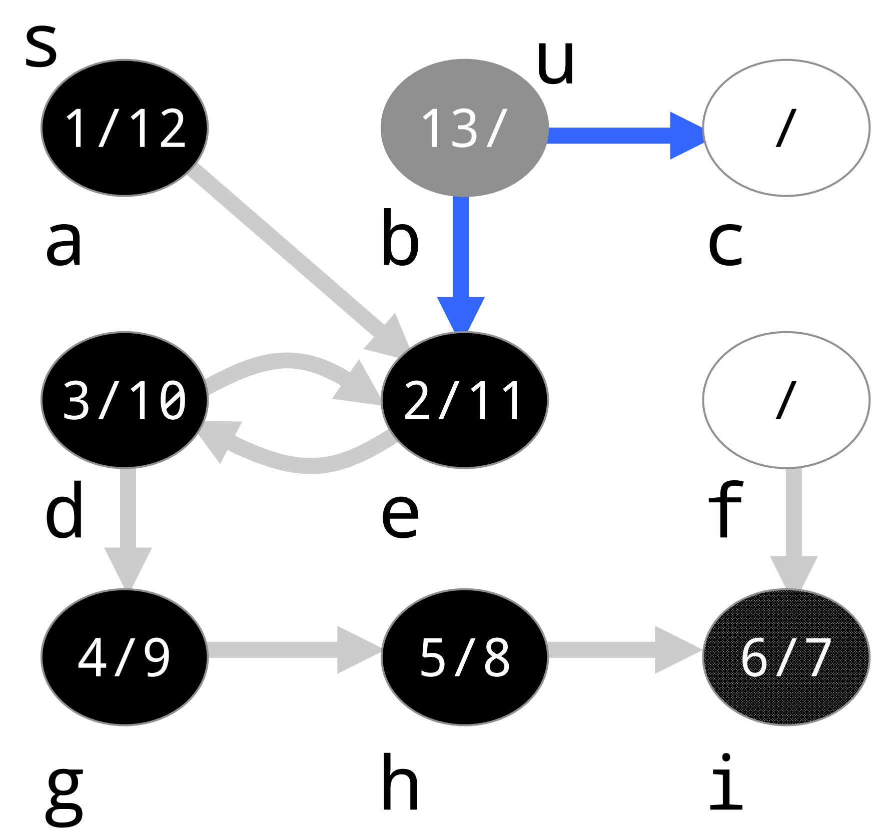
检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)



time: 13

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(c)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

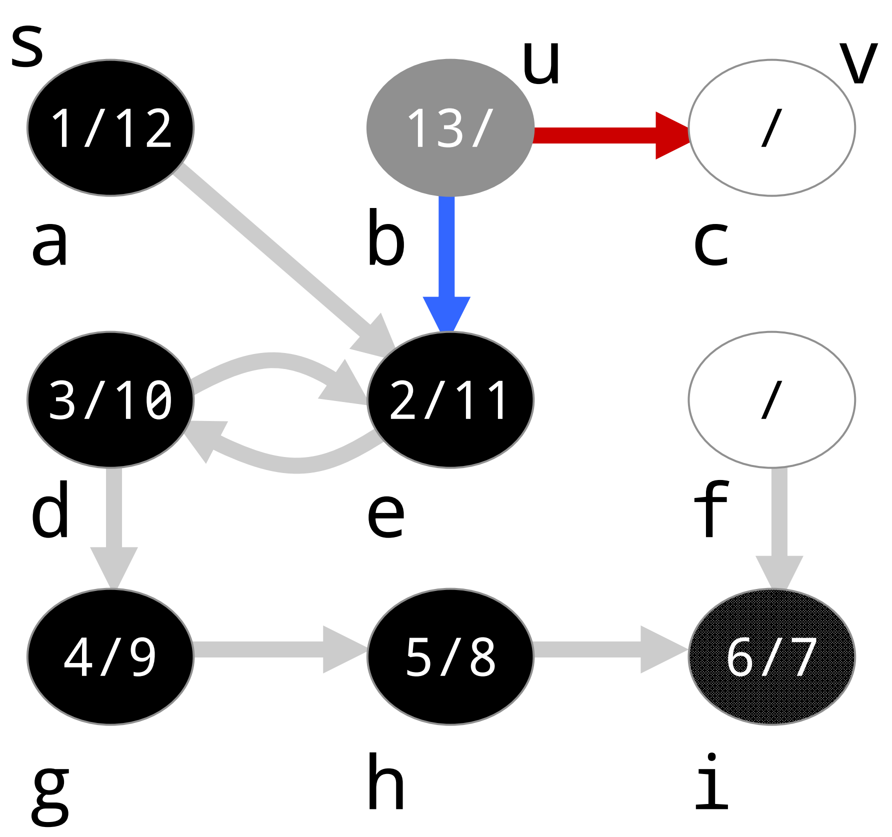
↓

DFS(G)

↓

DFS-Visit(G,u=b)

图G:



time: 13

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

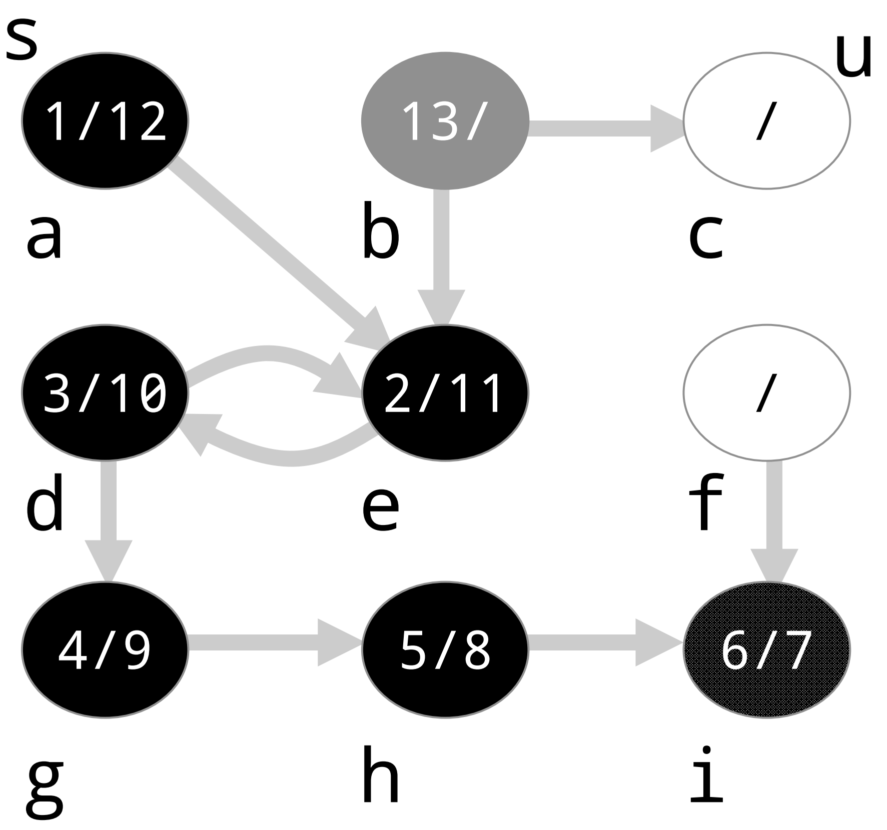
u(c)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)

DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)
↓
DFS-Visit(G,u=c)

图G:



time: 13

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

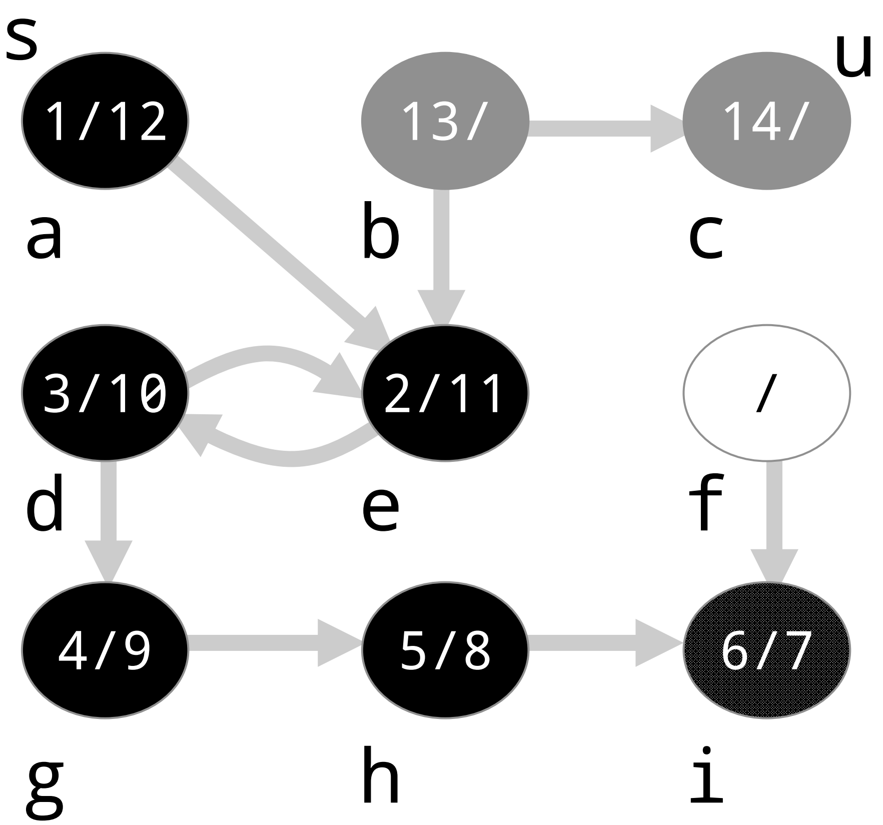
记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)
↓
DFS-Visit(G,u=c)



time: 14

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

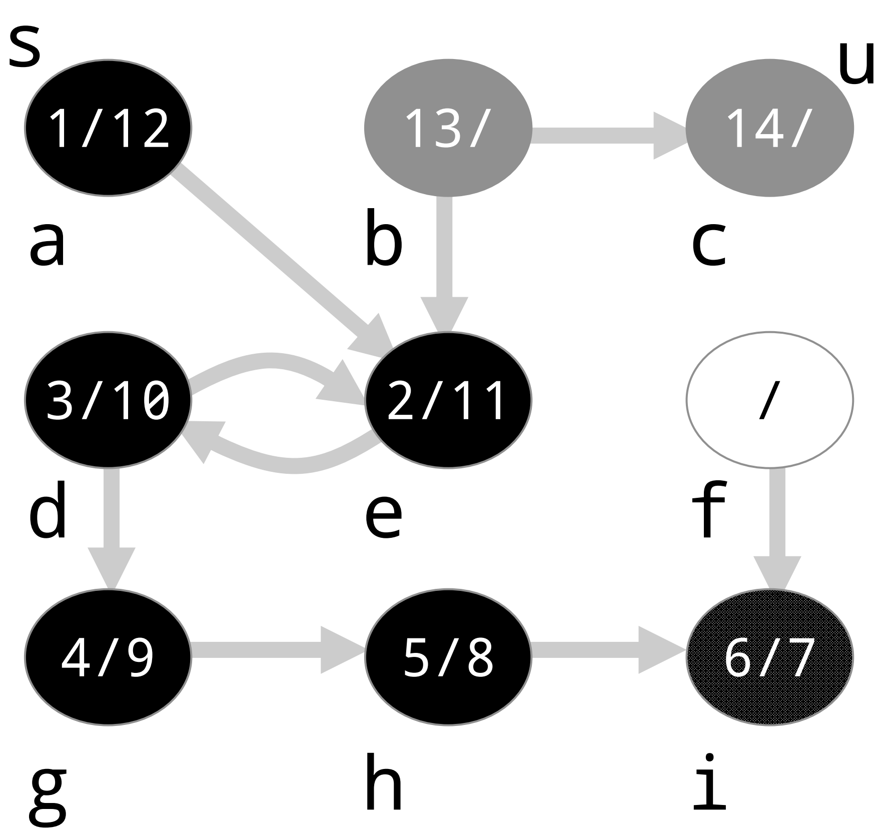
DFS(G)

↓

DFS-Visit(G,u=b)

↓

DFS-Visit(G,u=c)



time: 14

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

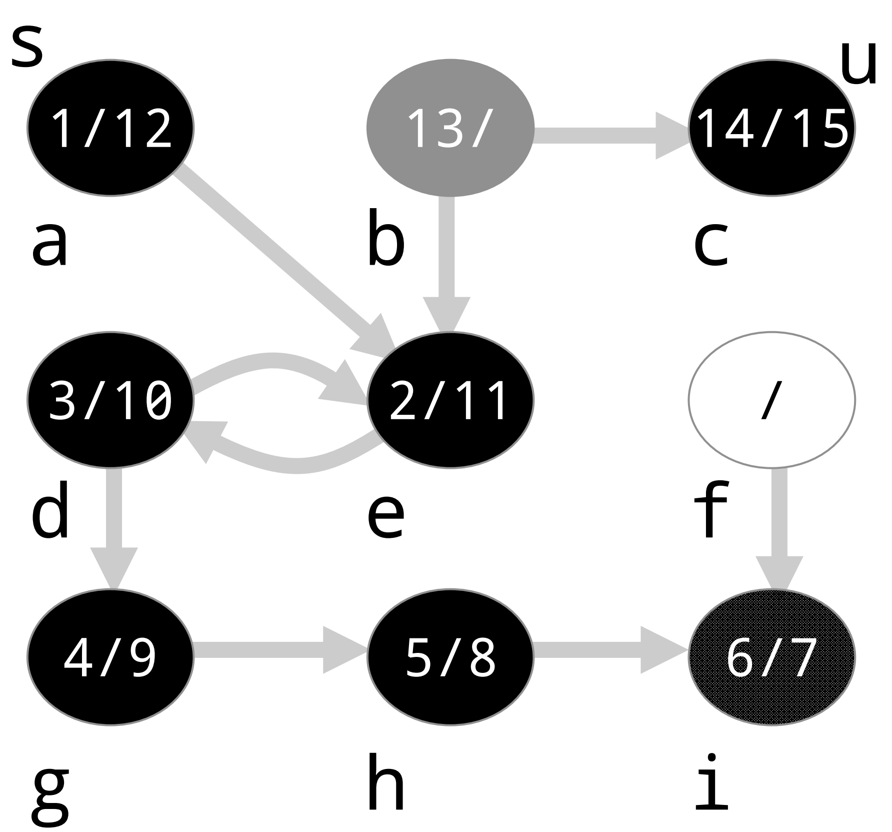
DFS(G)

↓

DFS-Visit(G,u=b)

↓

DFS-Visit(G,u=c)



time: 15

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

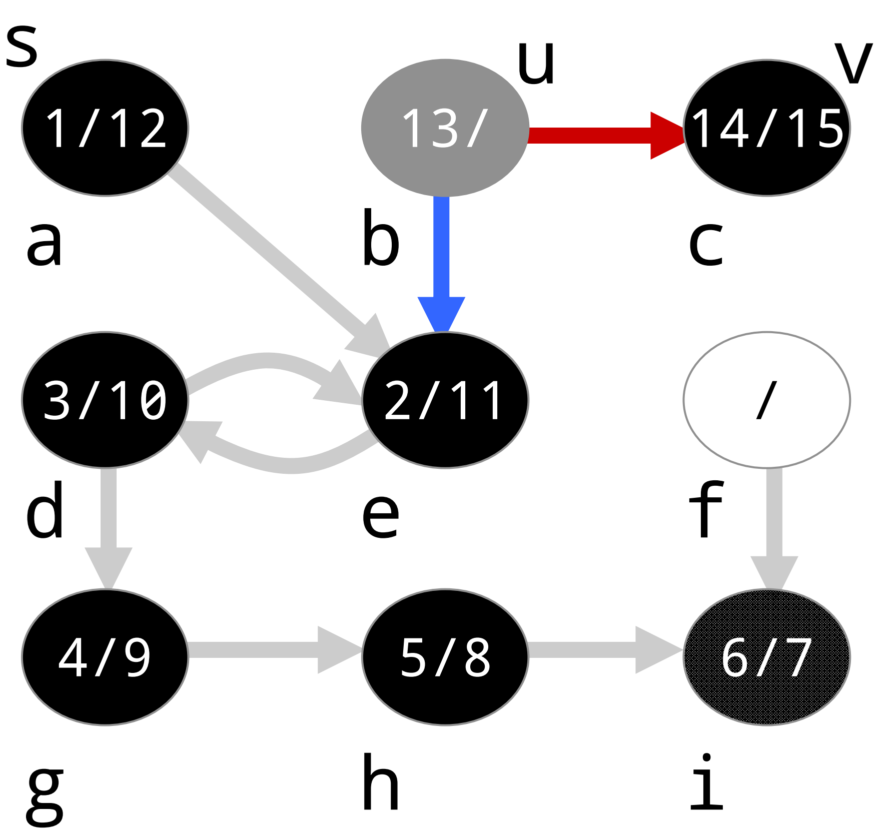
递归回退，结束邻结点v(c)访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)

图G:



time: 15

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

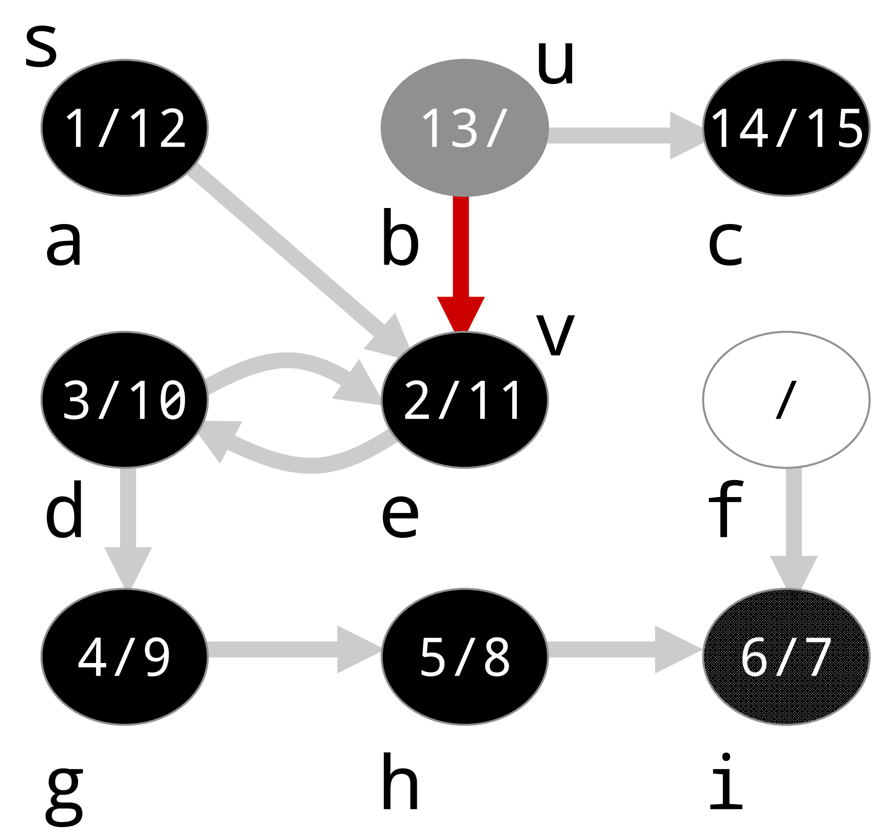
若v为白色，则递归访问v(e)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=b)



time: 15

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

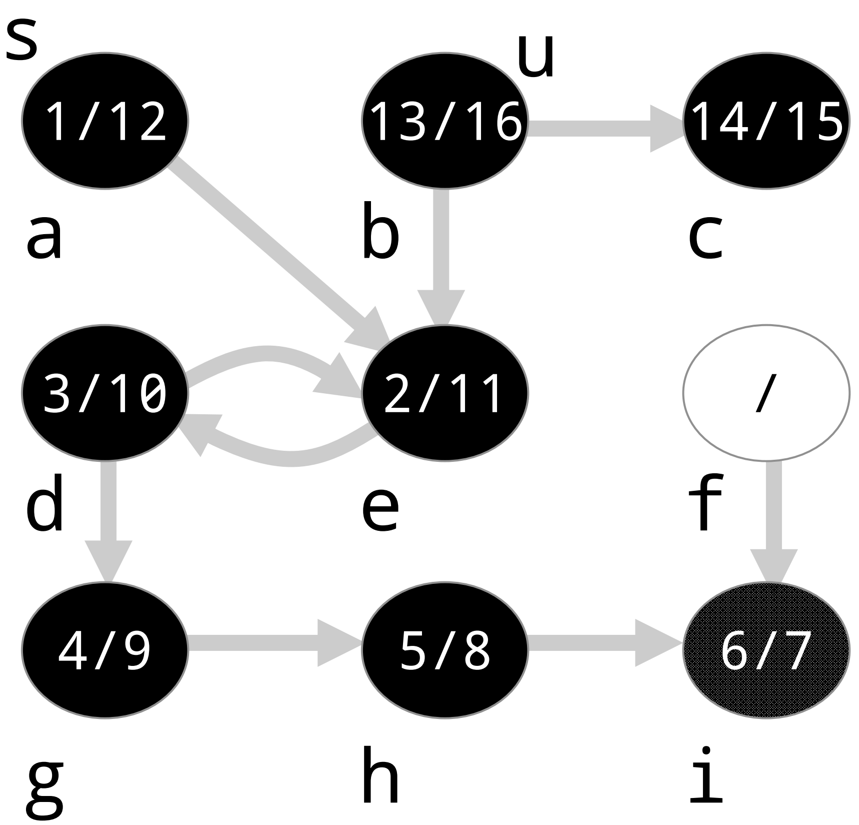
↓

DFS(G)

↓

DFS-Visit(G,u=b)

图G:



time: 16

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束u访问

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

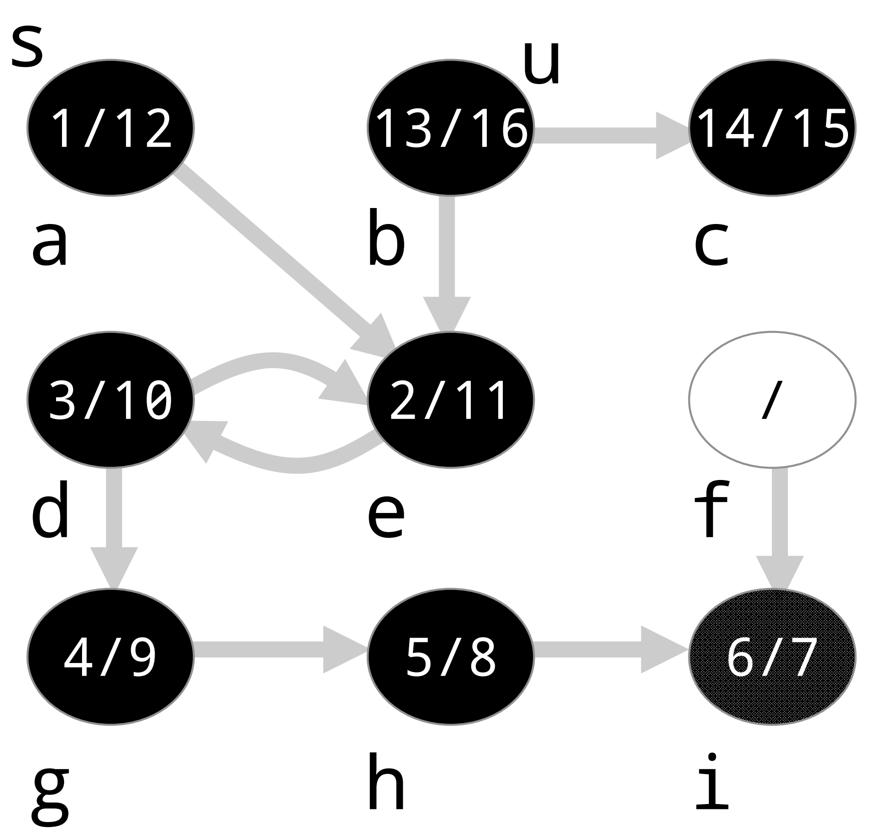
```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

↓

DFS(G)



time: 16

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

发现白色顶点时，调用DFS-VISIT访问该顶点u(f)

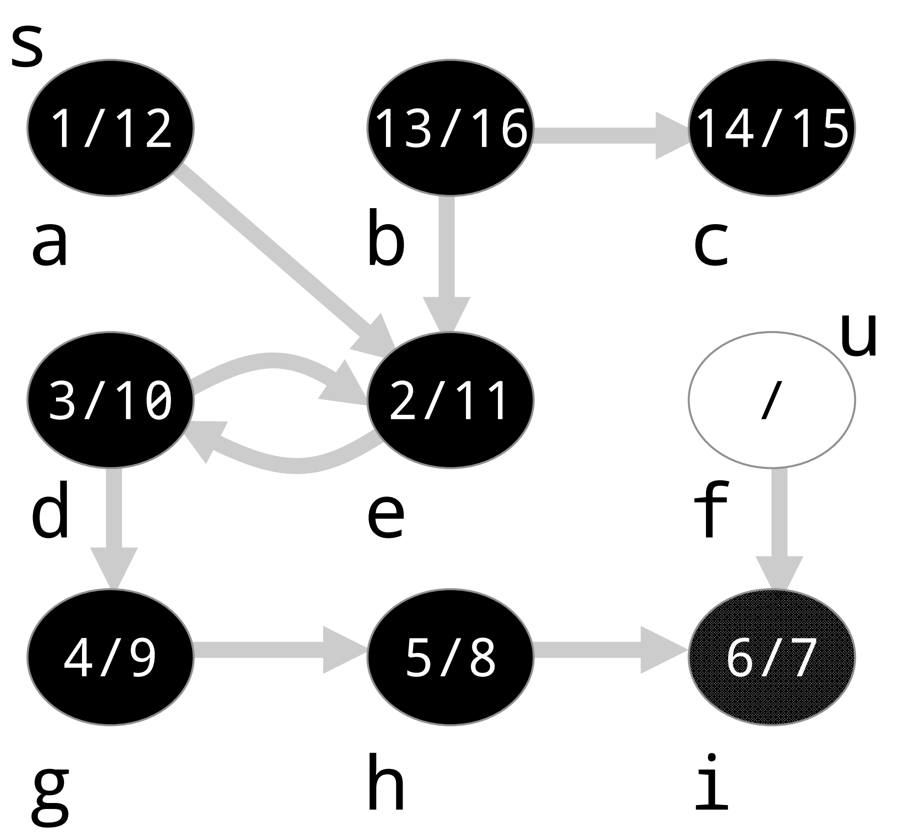
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:

DFS(G)



time: 16

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

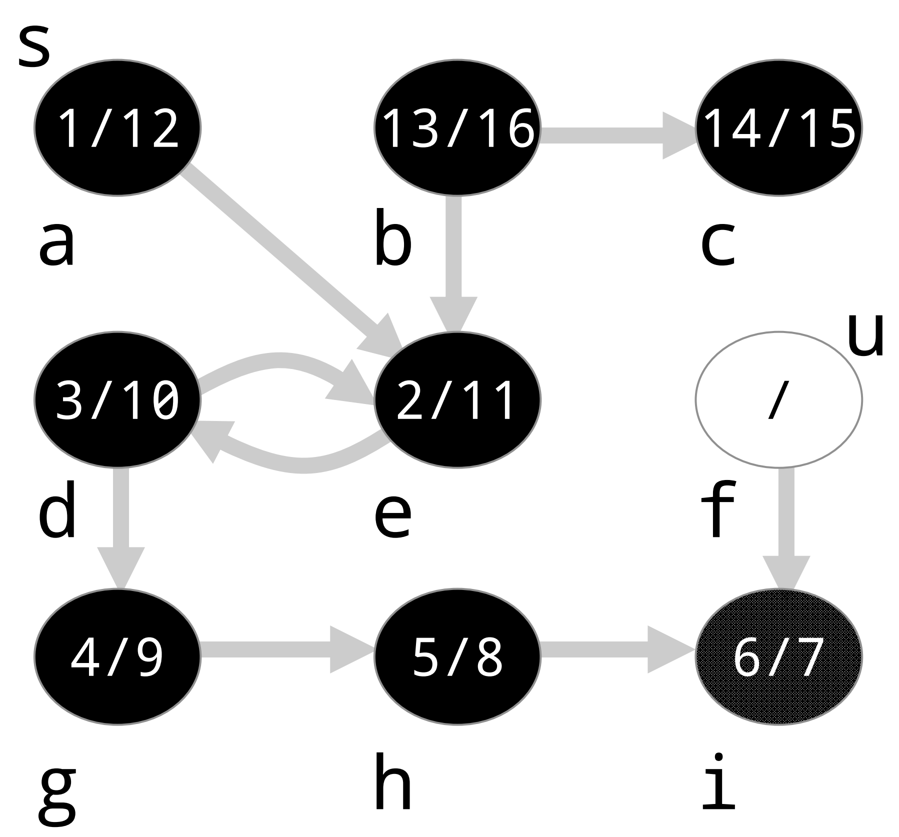
u(f)成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=f)

图G:



time: 16

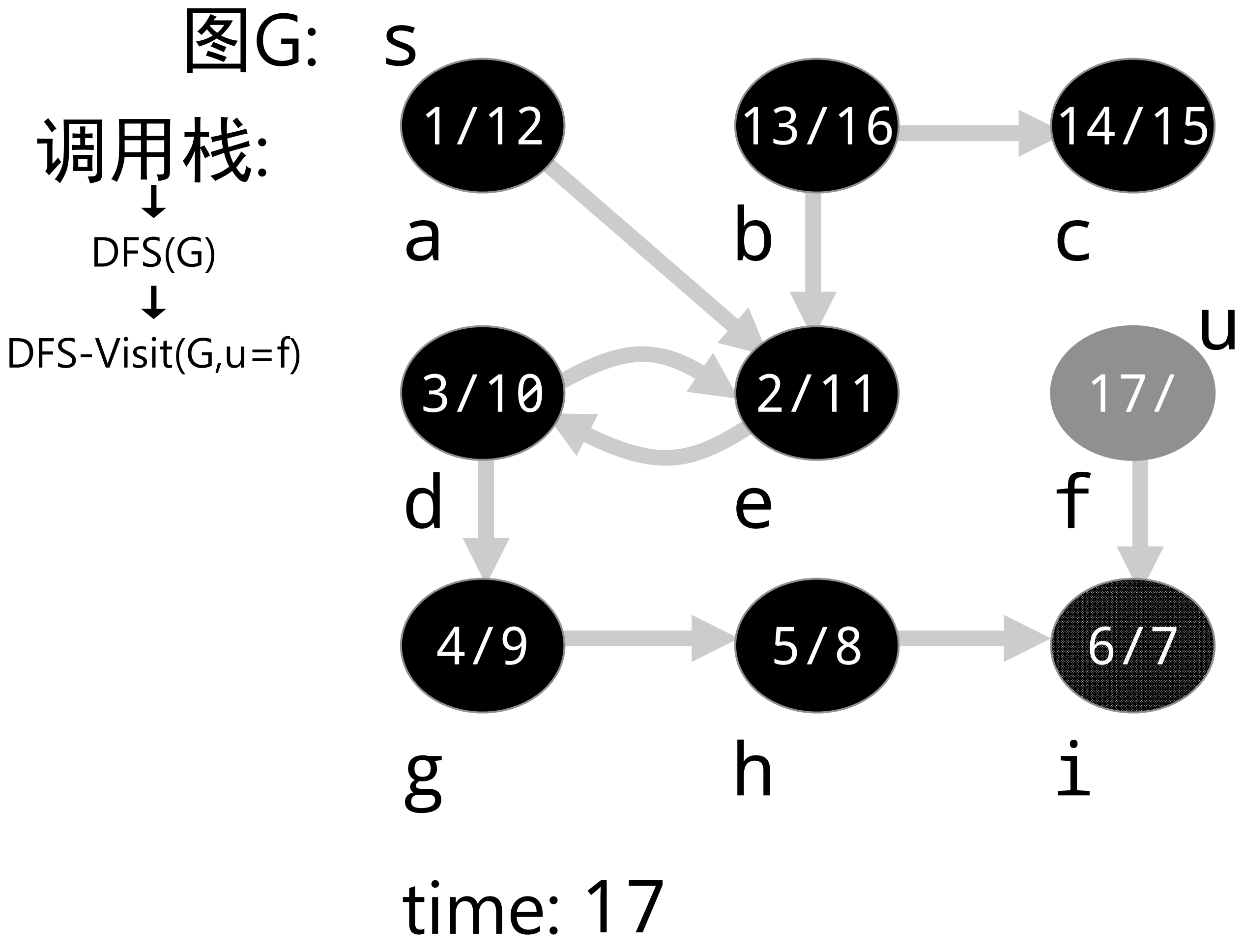
11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

记录u发现时刻，且置为灰色

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```



11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

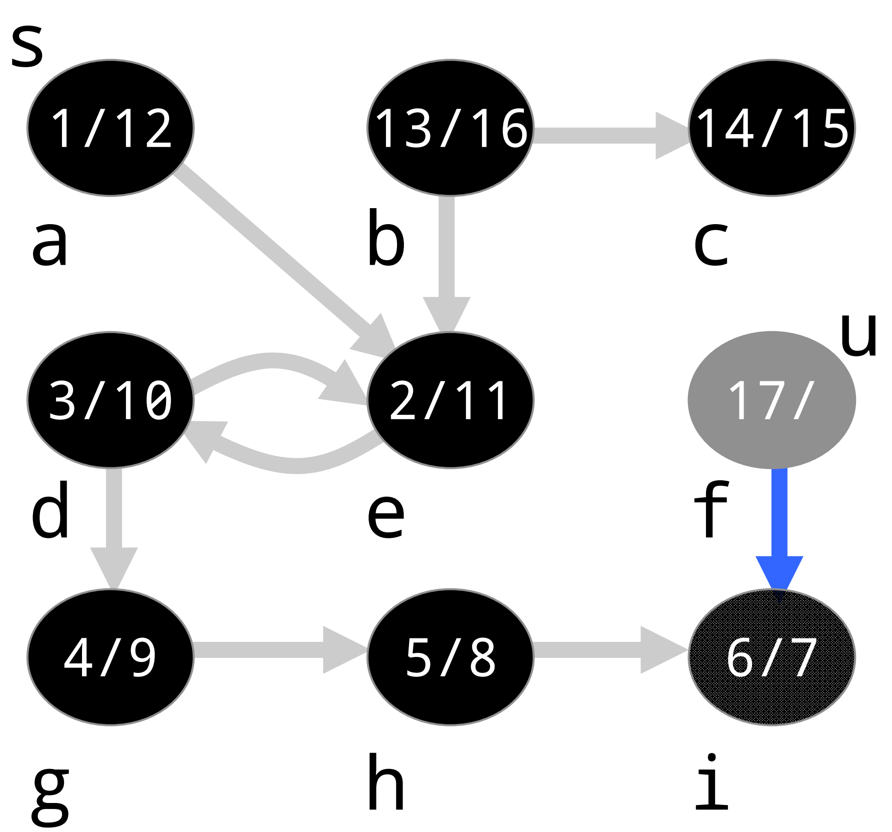
检查并访问u的每一个邻接点v

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

图G:

调用栈:
↓
DFS(G)
↓
DFS-Visit(G,u=f)



time: 17

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

若v为白色，则递归访问v(i)

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

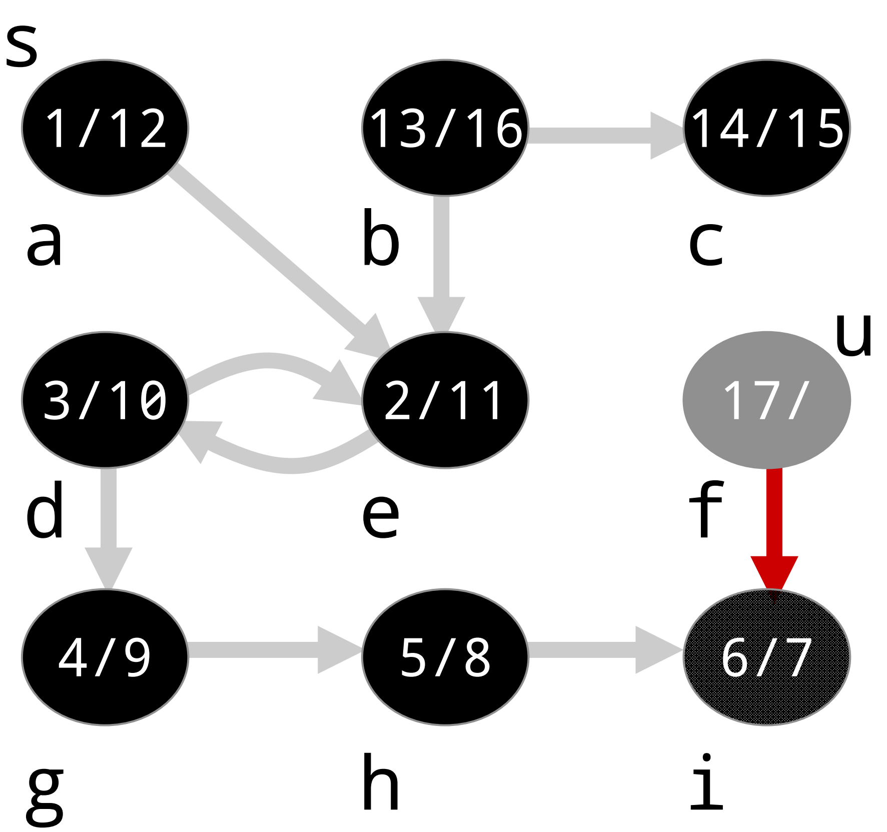
↓

DFS(G)

↓

DFS-Visit(G,u=f)

图G:



time: 17

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

u置为黑色，所有邻接点均已访问完成，记录完成时刻

```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

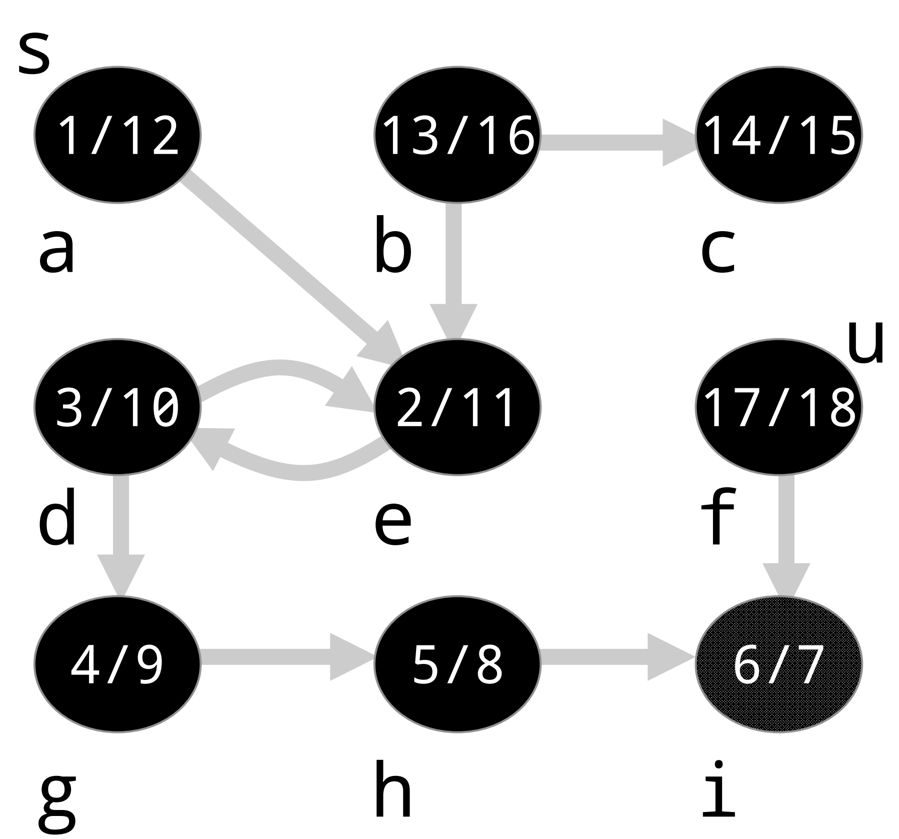
↓

DFS(G)

↓

DFS-Visit(G,u=f)

图G:



time: 18

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法示意

递归回退，结束u访问

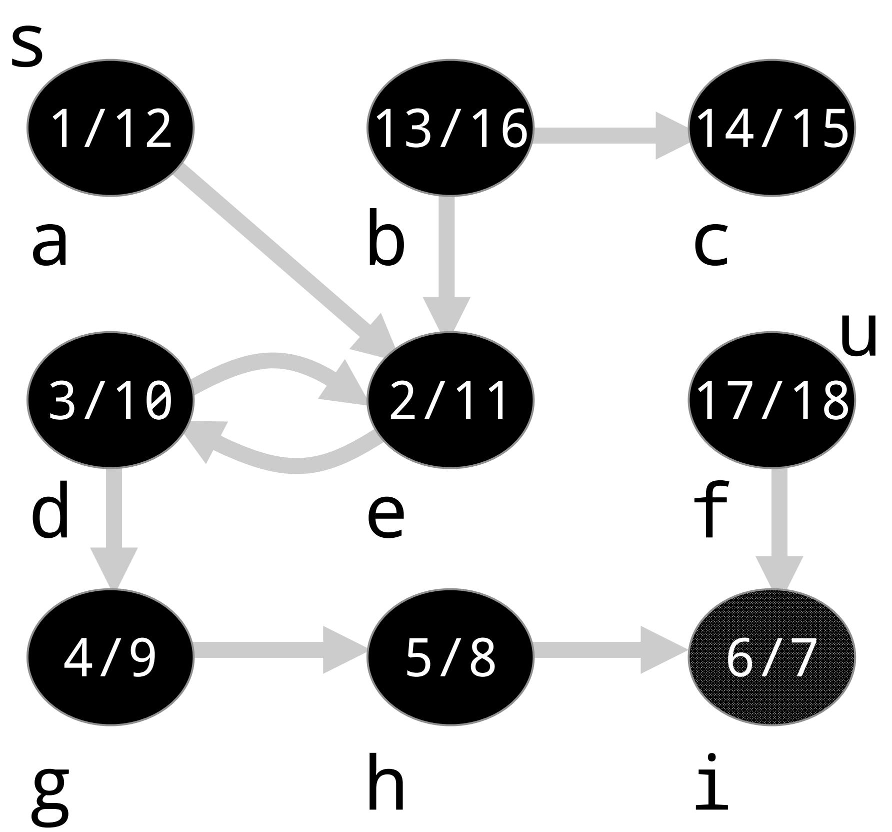
```
DFS(G):
for each u ∈ G.V
  u.color = white
time = 0
for each u ∈ G.V
  if u.color == white
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u):
u.d = ++time
u.color = gray
for each v ∈ G.Adj[u]
  if v.color == white
    DFS-Visit(G,v)
u.color = black
u.f = ++time
```

调用栈:

DFS(G)

图G:



time: 18

11.3.8 深度优先搜索DFS

DFS深度优先搜索（Depth First Search）算法

```
DFS(G):
for each u ∈ G.V
  u.color = white //所有顶点置为白色
time = 0 //复位时间戳
for each u ∈ G.V //依次检索V中的顶点
  if u.color == white //发现白色顶点时，调用DFS-VISIT访问该顶点u
    DFS-Visit(G,u)
```

```
DFS-Visit(G,u): //u成为深度优先森林中一棵新的树，对应一个发现时刻u.d和一个完成时刻u.f
u.d = ++time //记录u发现时刻
u.color = gray //u置为灰色
for each v ∈ G.Adj[u] //检查并访问u的每一个邻接点v
  if v.color == white //若v为白色，则递归访问v
    DFS-Visit(G,v) //递归深度，访问邻结点v
u.color = black //u置为黑色，所有邻接点均已访问完成
u.f = ++time //记录u完成时刻
```

11.3.9 搜索应用

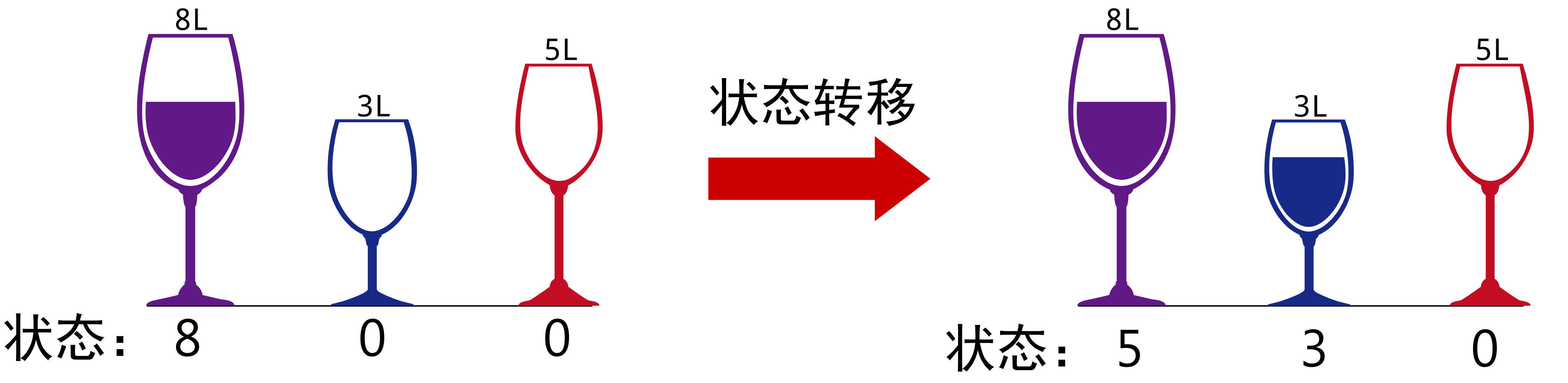
【BFS举例】

一个8L的杯子装满了酒，有一个3L的空杯子和一个5L的空杯子，问怎样才能用最少的次数倒出4L的酒。（酒不能倒掉）

11.3.9 搜索应用

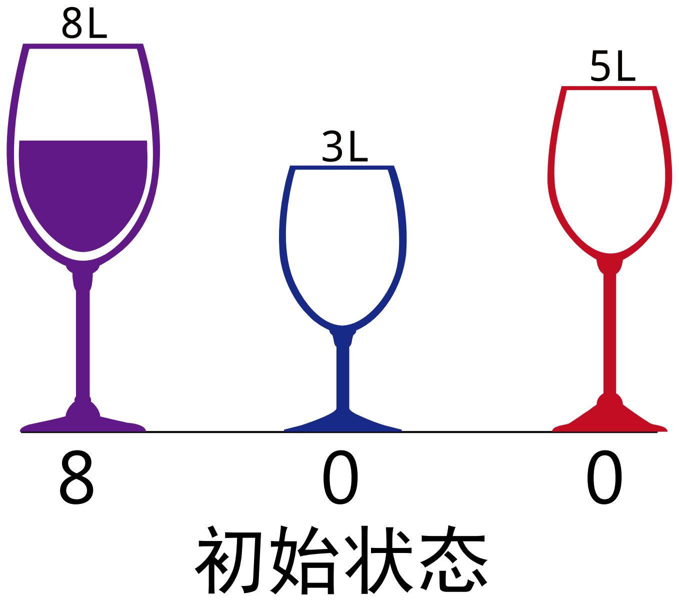
例题分析

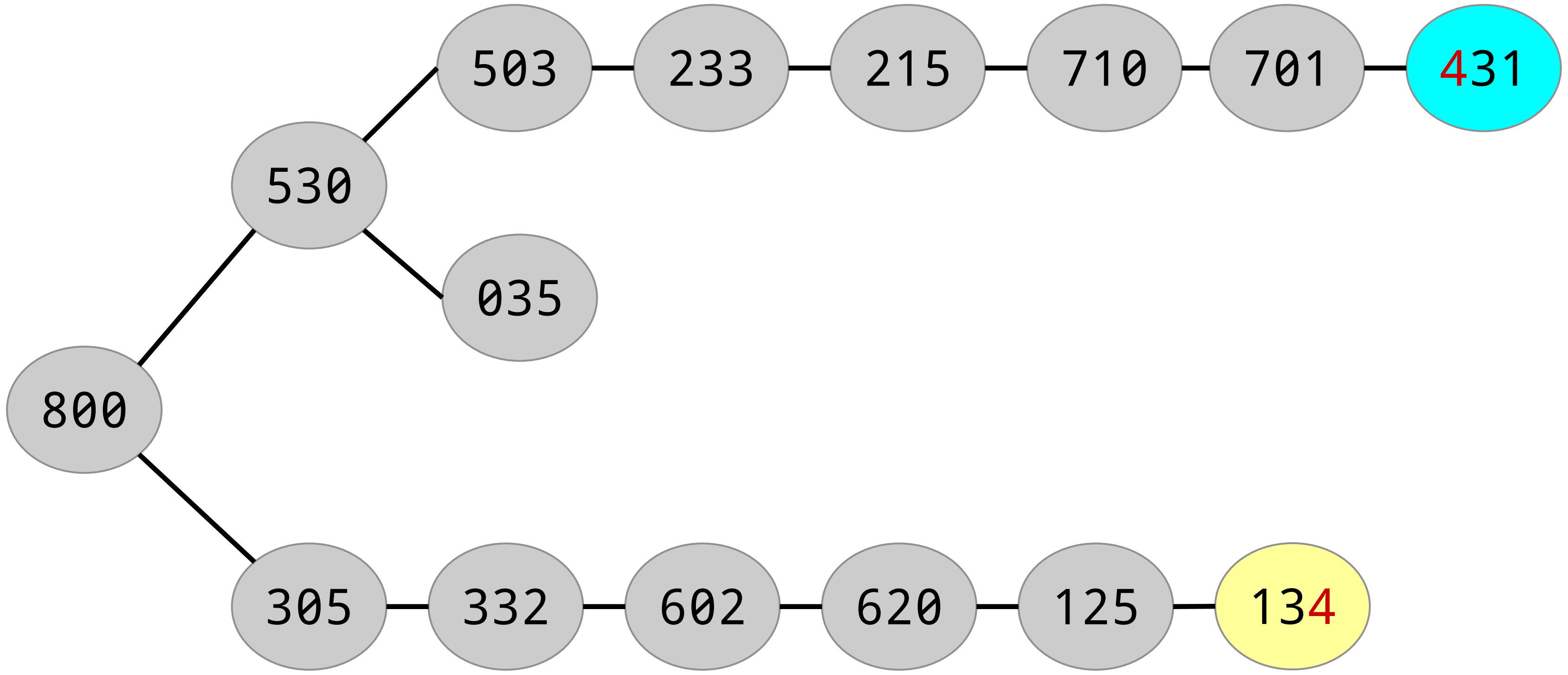
把每次倒完后的3个杯子中酒的数量视为一个状态，则倒酒的过程就是一个状态转移的过程。初始状态为800，状态转移则是把杯子A的酒倒入杯子B，结果是要么A为空，要么B为满。



例题分析

于是有：





得：800➞305➞332➞602➞620➞125➞134

例题分析

求解过程

（1）采用广度优先的状态转移，也就是（8L、3L、5L）从左到右，然后在每一列上从上到下，从而保证找到最少的步数。

（2）对于已经出现过的状态，不再处理（如从800到530后，还可以倒回去成为800，但因为800已出现过，不再处理）

（3）图的设计：

①顶点：某个时刻3个杯子中酒的状态

②边：可以通过一次倒酒实现的从一个状态到另外一个状态的转移

【DFS举例】

给出4个整数，要求用加、减、乘、除四个运算使其运算结果等于24（4个数不重复使用，且数字之间的除法中不得出现小数）。例如给出4个数：1、2、3、4。则以下运算得到24：

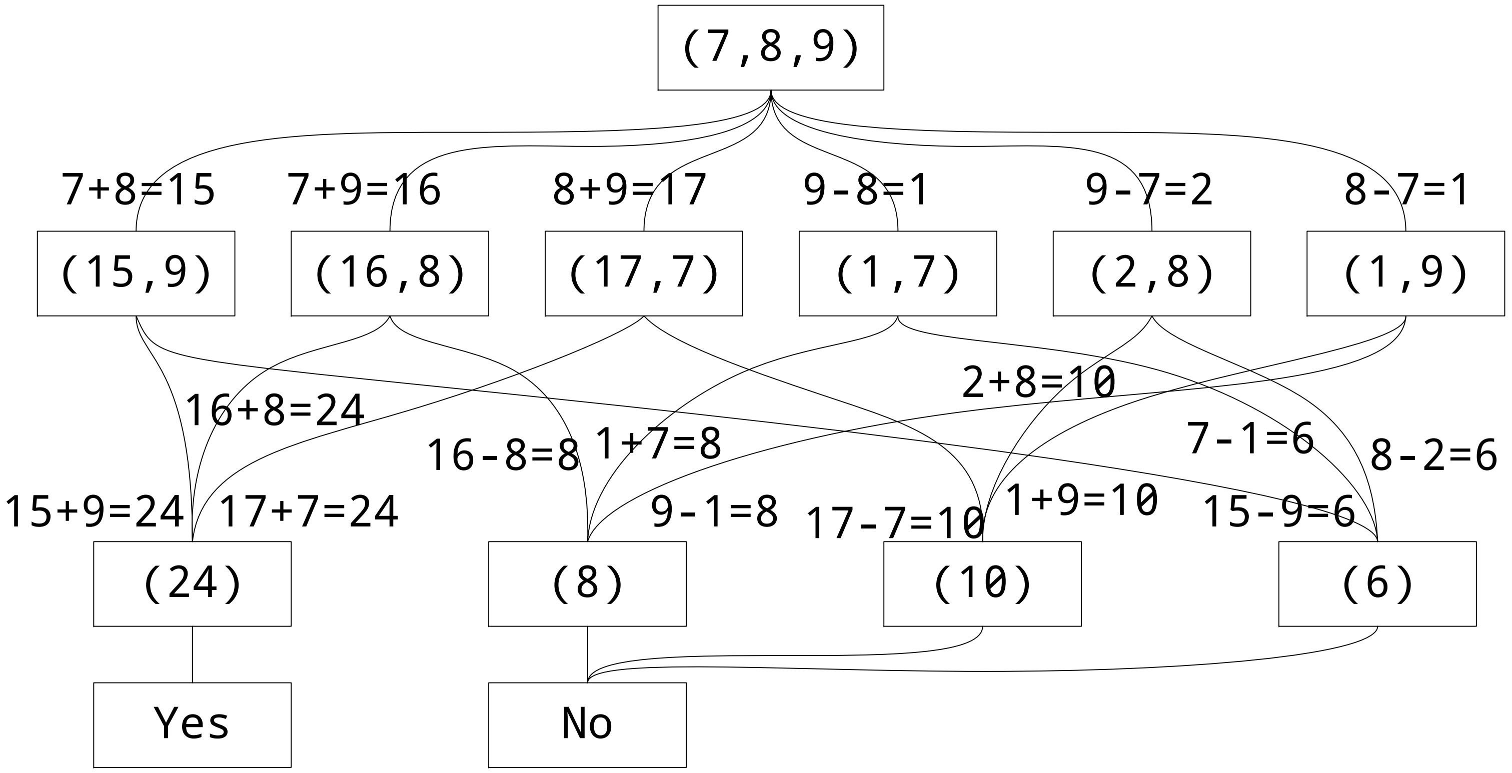
1*2*3*4=24、2*3*4/1=24、(1+2+3)*4=24、……

如上，是有很多种组合方式使得结果为24的，当然也有无法得到结果的4个数，例如：1、1、1、1。

现在给出4个数，判断它们能够通过一定的运算组合之后等于24吗？

例题分析

先把问题简化为：有7、8、9三个数，只有加减运算情形。得到如下搜索树：



(7,8,9)

7+8=15
7+9=16
8+9=17
9-8=1
9-7=2
8-7=1

(15,9)
(16,8)
(17,7)
(1,7)
(2,8)
(1,9)

2+8=10
16+8=24
7-1=6
16-8=8
1+7=8
8-2=6
15+9=24
17+7=24
9-1=8
17-7=10
1+9=10
15-9=6

(24)
(8)
(10)
(6)

Yes
No

例题分析

此处只有3个数且只有加减法，所以第二层节点最多6个，当延伸到第三层时节点数就比较多了，若使用BFS，缺点就明显了，需要很大的空间去维护那个队列。若使用DFS，第一层是3个数，第二层2个数，即递归深度不会超过3层，因此更合理。

11.3.9 搜索应用

【BFS举例】

八数码问题(Eight-puzzle)

在3X3的棋盘上，摆有8个棋子，在每个棋子上标有1～8中的某一数字。棋盘中留有一个空格。空格周围的棋子可以移到空格中。要求解的问题是，给出一种初始布局和目标布局，找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。初始状态和目标状态如下：

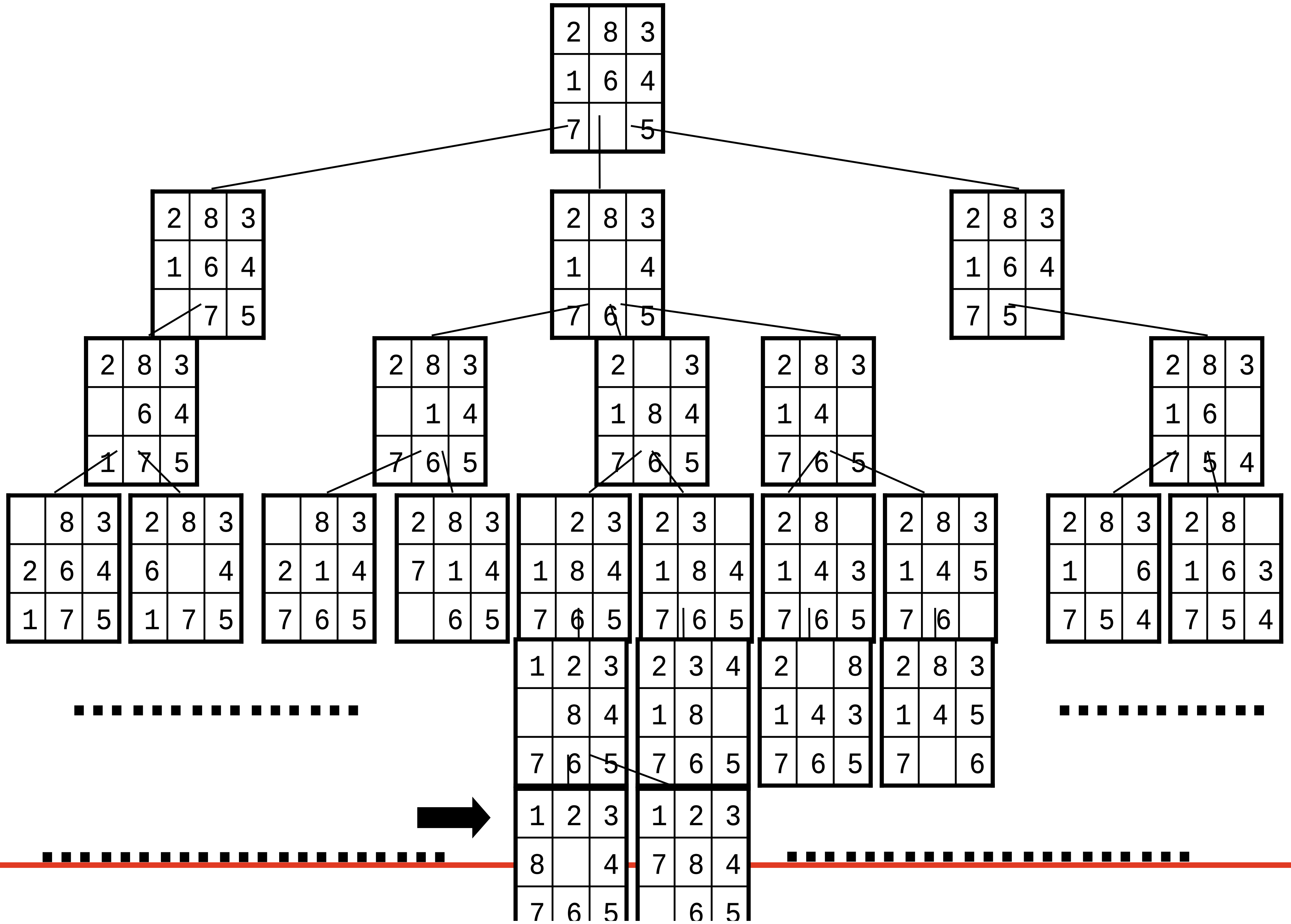
初始状态

2	8	3
1	6	4
7		5

目标状态

1	2	3
8		4
7	6	5

11.3.9 搜索应用

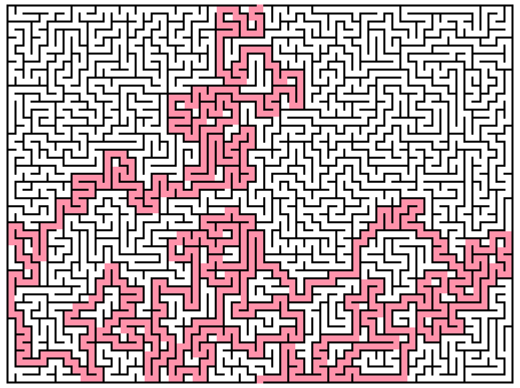

2 8 3 / 1 6 4 / 7 5
2 8 3 / 1 6 4 / 7 5
2 8 3 / 1 4 / 7 6 5
2 8 3 / 1 6 4 / 7 5
2 8 3 / 6 4 / 1 7 5
2 8 3 / 1 4 / 7 6 5
2 3 / 1 8 4 / 7 6 5
2 8 3 / 1 4 / 7 6 5
2 8 3 / 1 6 / 7 5 4
8 3 / 2 6 4 / 1 7 5
2 8 3 / 6 4 / 1 7 5
8 3 / 2 1 4 / 7 6 5
2 8 3 / 7 1 4 / 6 5
2 3 / 1 8 4 / 7 6 5
2 3 / 1 8 4 / 7 6 5
2 8 / 1 4 3 / 7 6 5
2 8 3 / 1 4 5 / 7 6
2 8 3 / 1 6 / 7 5 4
2 8 / 1 6 3 / 7 5 4
1 2 3 / 8 4 / 7 6 5
2 3 4 / 1 8 / 7 6 5
2 8 / 1 4 3 / 7 6 5
2 8 3 / 1 4 5 / 7 6
1 2 3 / 8 4 / 7 6 5
1 2 3 / 7 8 4 / 6 5

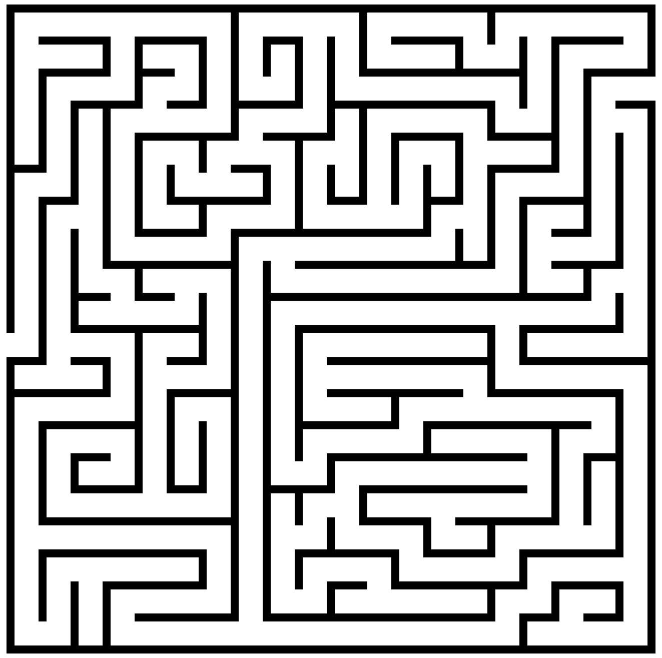
【DFS和BFS举例】

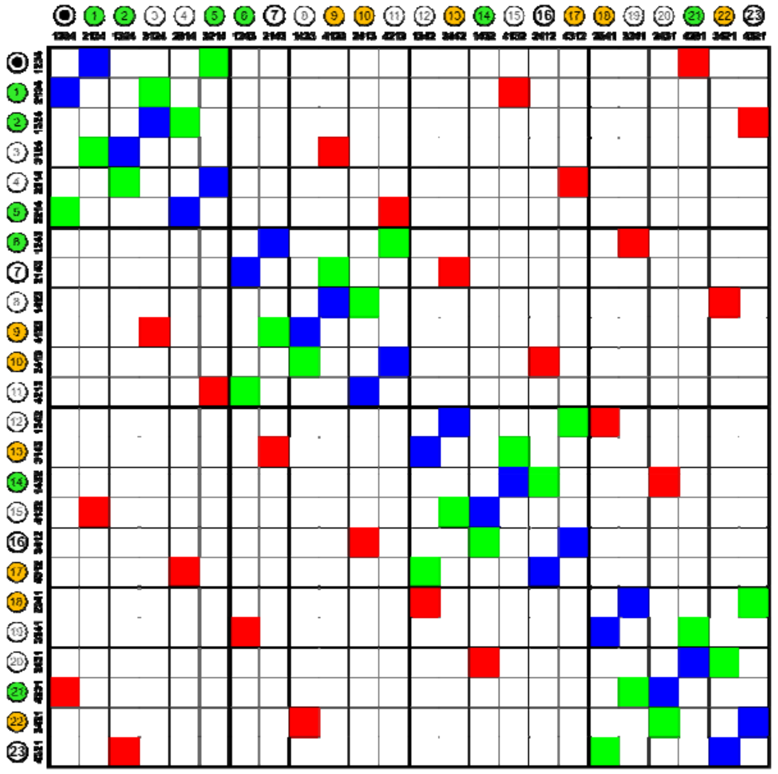
迷宫问题

（1）迷宫生成

（2）机器人走迷宫

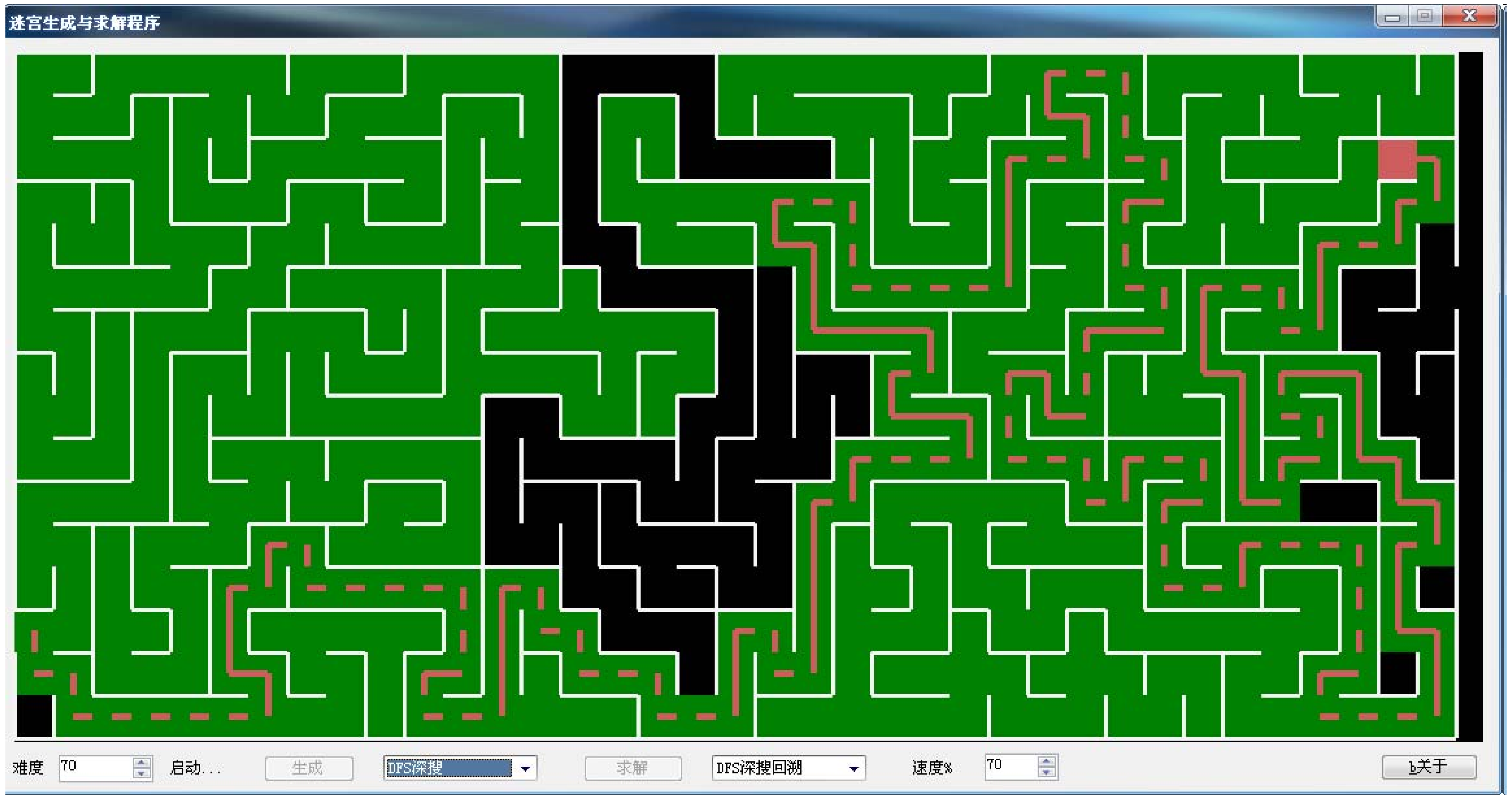






11.3.9 搜索应用

- 迷宫生成及求解程序



迷宫生成与求解程序

难度 70

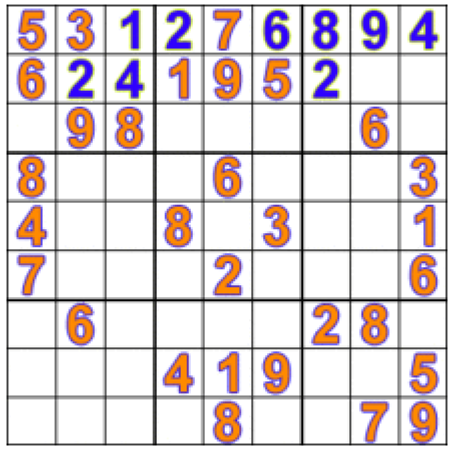
DFS深搜

DFS深搜回溯

速度% 70

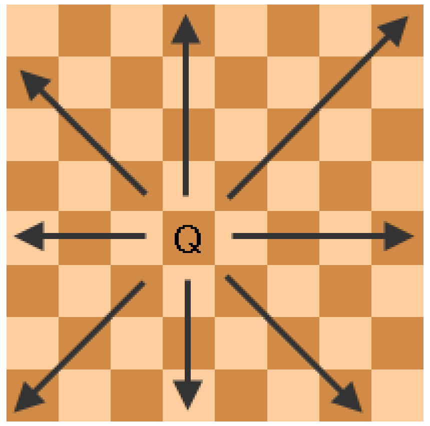
11.3.4 回溯法

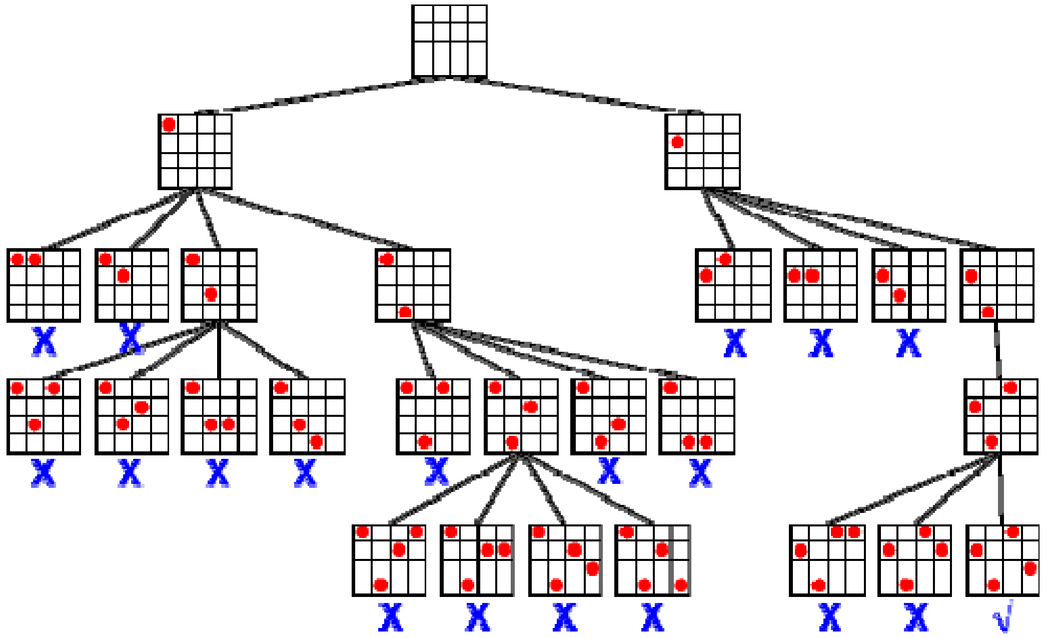
- 回溯法（backtracking）
- 回溯法是一个既带有系统性又带有跳跃性的搜索算法。



5	3	1	2	7	6	8	9	4
6	2	4	1	9	5	2		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

回溯法求解数独（Sudoku）问题

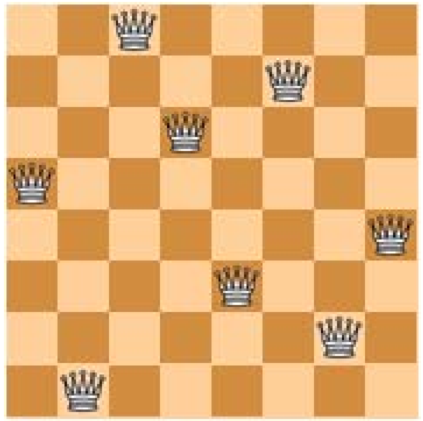


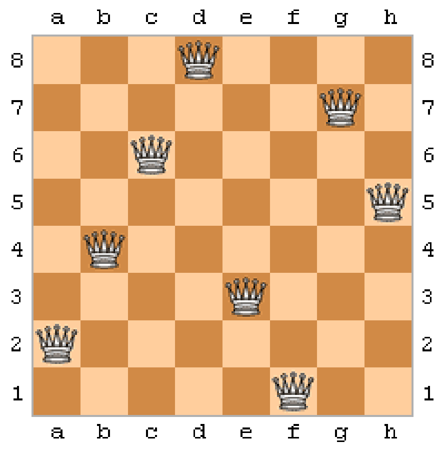


回溯法求解4皇后问题

11.3.4 回溯法

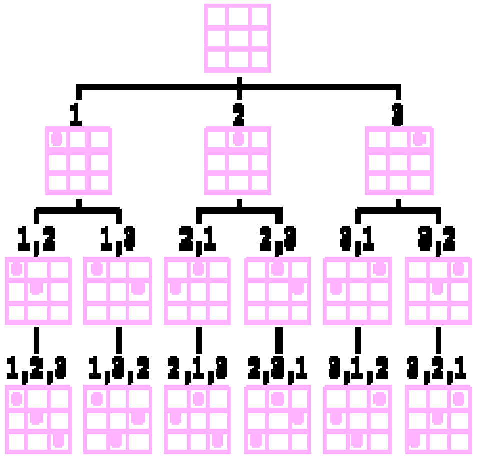
- 1．基本思想
- 回溯法在包含问题的所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树。算法搜索至解空间树的任一结点时，总是先判断该结点是否肯定不包含问题的解。如果肯定不包含，则跳过对以该结点为根的子树的系统搜索，逐层向其祖先结点回溯。否则，进入该子树，继续按深度优先的策略进行搜索。

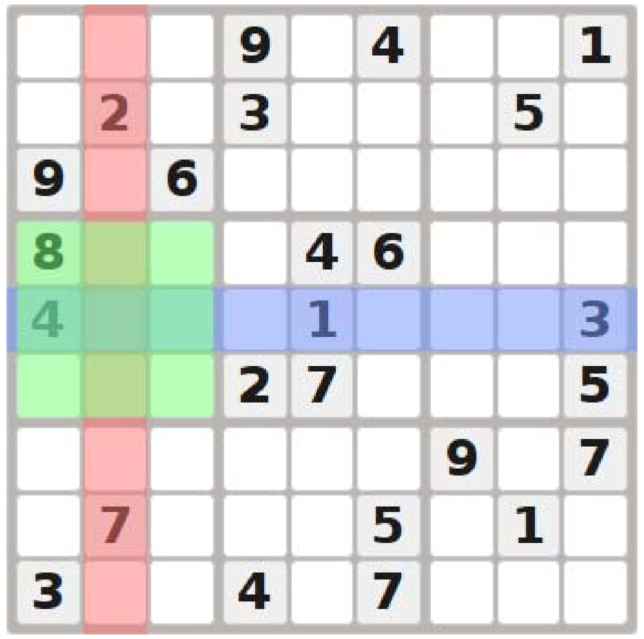




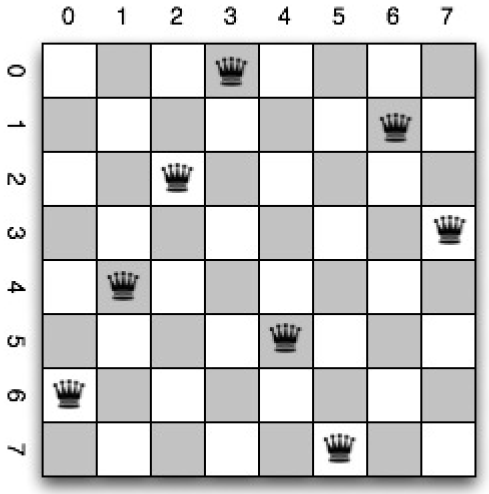
11.3.4 回溯法

- 回溯法求问题的所有解时，要回溯到根，且根结点的所有子树都已被搜索遍才结束。回溯法求问题的任一解时，只要搜索到问题的一个解就可以结束。适用于求解组合数较大的问题。





			9		4			1
	2		3				5	
9		6						
8				4	6			
4				1				3
			2	7				5
						9		7
	7				5		1	
3			4		7			



11.3.4 回溯法

- 应用回溯法解问题时，首先应明确定义问题的解空间，问题的解空间至少应包含问题的一个（最优）解。

- 确定了解空间的组织结构后，回溯法就从开始结点（根结点）出发，以深度优先的方式搜索整个解空间。

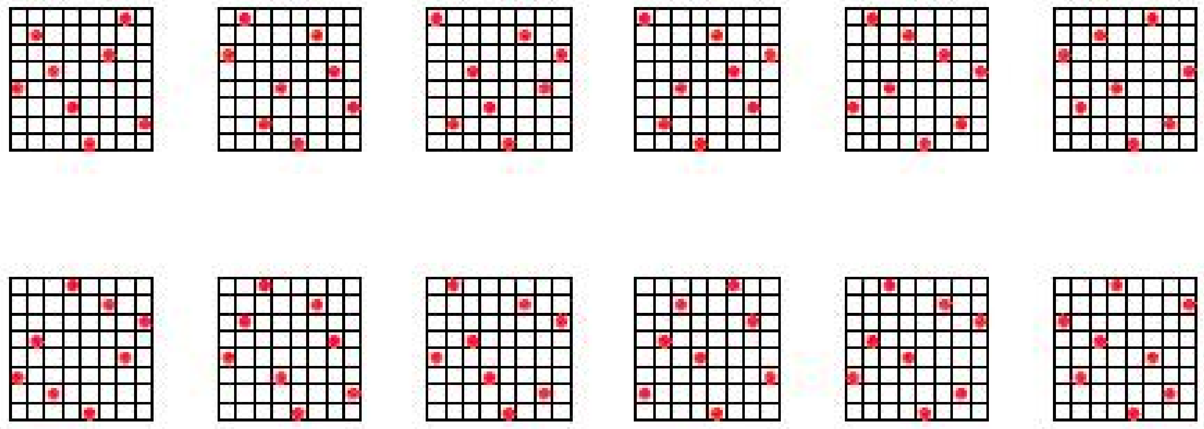
11.3.4 回溯法

- 2．基本步骤
- ①针对所给问题，定义问题的解空间；
- ②确定易于搜索的解空间结构；
- ③以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。
- 由于回溯法是对解空间的深度优先搜索，因此可用递归函数来实现回溯法。

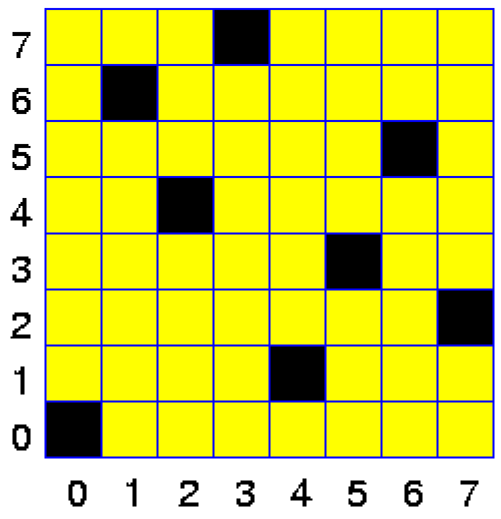
11.3.4 回溯法

【例11.9】

设计一个算法求n皇后问题。在nxn格棋盘上放置n个皇后，使其不能互相攻击，即任意两个皇后不放在同一行、同一列或同一斜线上，问有多少种摆法。



8–QUEENS PROBLEM



(0, 4, 7, 5, 2, 6, 1, 3): lexicographically first solution, when placing queens row–by–row, found after placing 114 queens

11.3.4 回溯法

例题分析

分析

（1）回溯算法设计

用n元组x[1:n]表示n皇后问题的解，其中x[i]表示皇后i放在棋盘的第i行的第x[i]列。由于不允许将两个皇后放在同一列上，所以解向量中的x[i]互不相同。将nxn格棋盘看作是二维方阵，其行号从上到下，列号从左到右依次为1,2,...,n。从棋盘左上角到右下角的主对角线及其平行线（即斜率为－1的各斜线）上，两个下标值的差（行号减去列号）值相等。同理，斜率为+1的每条斜线上，两个下标值的和（行号加上列号）值相等。

例题分析

因此，若两个皇后放置的位置分别是(i,j)和(k,l)，且i-j=k-l或i+j=k+l，则说明这两个皇后处于同一条斜线上。前面两个方程等价于i-k=j-l和i-k=l-j 。由此可知，只要|i-k|=|j-l|成立，就表明两个皇后位于同一条斜线上。

用回溯法解n皇后问题时，用完全n叉树表示解空间，用可行性约束剪去不满足行、列和斜线约束的子树即可求解。

例题分析

（2）算法设计步骤

解n皇后问题的回溯法中，用递归方法backtrack(1)实现对整个解空间的回溯搜索，backtrack(i)搜索解空间中第i层子树，sum记录当前已找到的可行方案数目。

在算法backtrack中，当i>n时算法搜索至叶结点，得到一个新的n皇后放置方案，则当前已找到的可行方案数sum加1。当i≤n时，当前扩展结点K是解空间中的内部结点。该结点有x[i]=1,2,...,n，共n个儿子结点。对当前扩展结点K的每个儿子结点，由place约束检查可行性，并以深度优先的方式递归对可行子树搜索，或剪去不可行子树。

11.3.4 回溯法

例11.9

```
#include <stdio.h>
#include <math.h>
int place(int k,int x[])
{ //检测可行性
  int i;
  for(i=1; i<k; i++)
    if (abs(k-i)==abs(x[i]-x[k]) || x[i]==x[k]) return 0;
  return 1;
}
void backtrack(int t,int n,int x[],int *sum)
{ //回朔法求n皇后问题,x排列方法，sum解总数
  int i;
  if (t>n) (*sum)++; //得到新的解，可以在此输出x得到排列方法
  else
    for(i=1; i<=n; i++) {
```

11.3.4 回溯法

例11.9

```c
      x[t]=i;
      if (place(t,x)) backtrack(t+1,n,x,sum);
    }
}
int main()
{
  int i,sum=0,x[100],n=8; //8皇后
  for(i=0; i<=n; i++) x[i]=0;
  backtrack(1,n,x,&sum); //计算8皇后问题解数目
  printf("%d\n",sum);
  return 0;
}
```

CP程序设计