Premaster Course Algorithms 1

Chapter 5: Basic Graph Algorithms

Christian Scheideler

SS 2018

Basic Graph Algorithms

Overview:

- Basic notation
- Graph representations
- Breadth-First-Search
- Depth-First-Search
- Minimum spanning trees

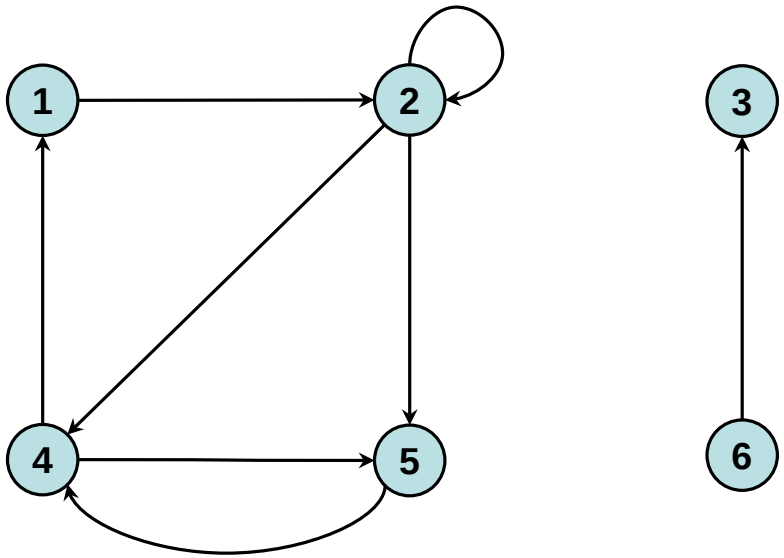
Basic Notation

- A directed graph G is a pair (V,E), where V is a finite set and $E \subseteq V \times V$.
- Elements in V are called nodes or vertices, elements in E are called edges. Correspondingly, V is the node set and E the edge set of G.
- Edges are ordered pairs of nodes. Edges of the form (u,u), $u \in V$, are allowed and called loops.
- If $(u,v) \in E$, we say that the edge leads from u to v. We also say that u and v are adjacent.
- The degree of a node v is the number of edges (v,w) in E.

Directed Graph

$$V = \{1,2,3,4,5,6\}$$

$$E = \{(1,2),(2,2),(2,4),(2,5),(4,1),(4,5),(5,4),(6,3)\}$$



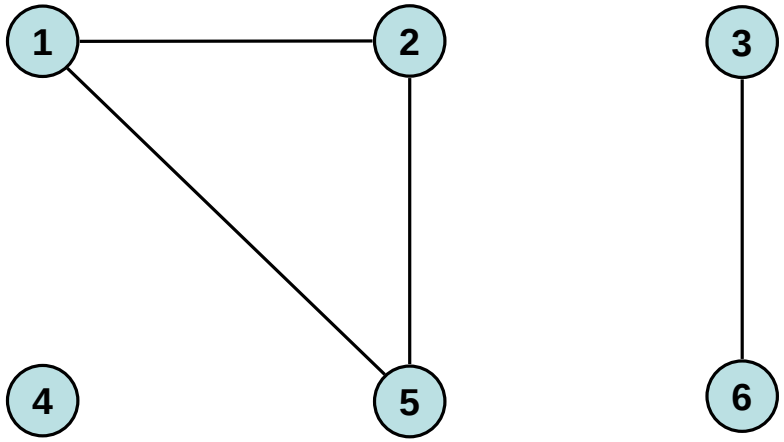
Basic Notation

- An undirected graph G is a pair (V,E), where V is a finite set and E is a set of subsets of V of size two.
- Elements in V are called nodes or vertices, elements in E are called edges. Correspondingly, V is the node set and E the edge set of G.
- Edges have the form $\{u,v\}$. Edges of the form $\{u,u\}$ are usually not allowed.
- If $\{u,v\} \in E$, then we say that u and v are adjacent.
- The degree of u is the number of edges $\{u,v\}$ in E.

Undirected Graph

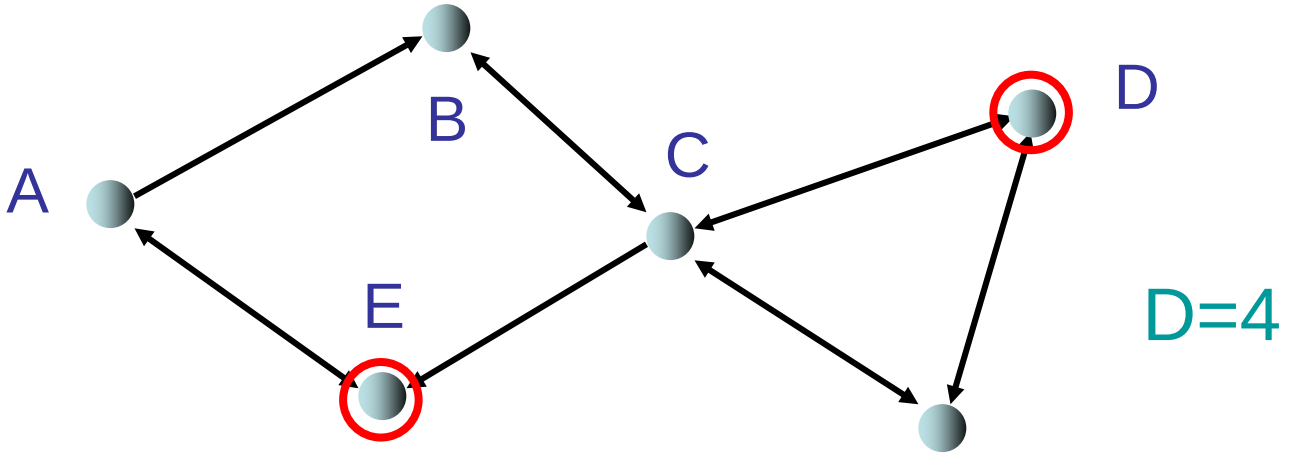
V={1,2,3,4,5,6}

E={ {1,2}, {1,5}, {2,5}, {3,6} }



Graph Theory

- n: number of nodes, m: number of edges
- $\delta(v,w)$: distance of w to v in G
  - directed graph: number of edges of a shortest directed path from v to w
  - undirected draph: number of edges of a shortest path from v to w
- $D=\max_{v,w} \delta(v,w)$: diameter of G

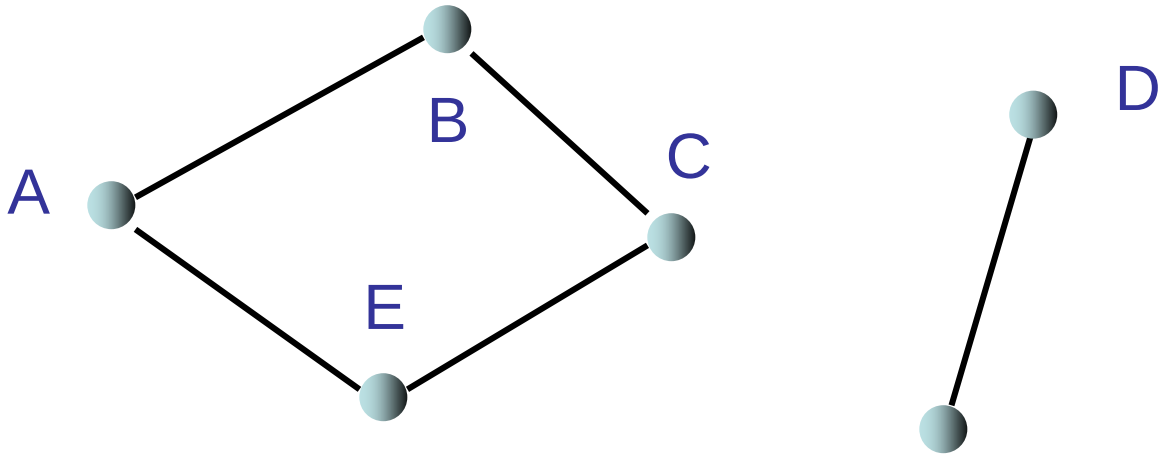


Graph Theory

G undirected:

- G is connected: diameter D is finite (i.e., there is a path from every node to every other node in G)

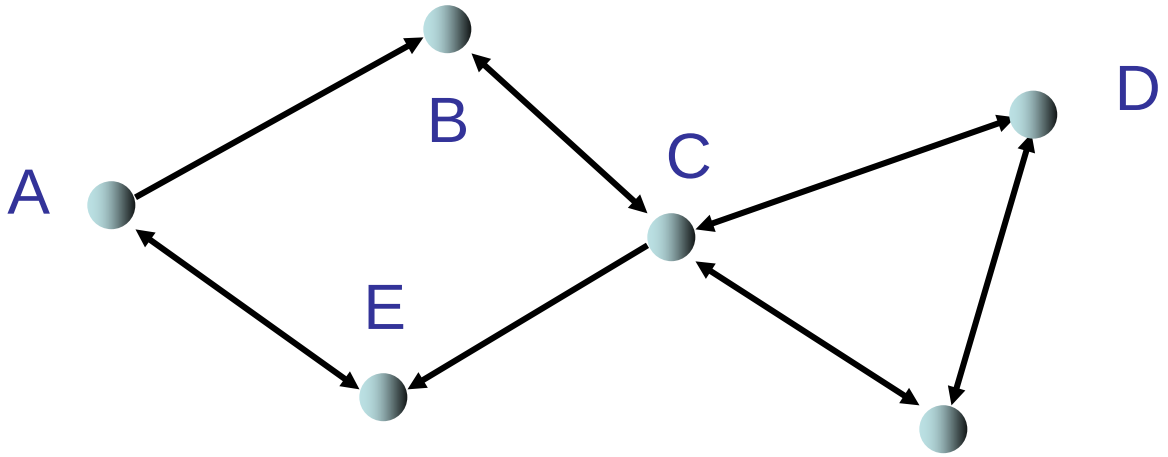
Example: graph that is not connected



Graph Theory

G directed:

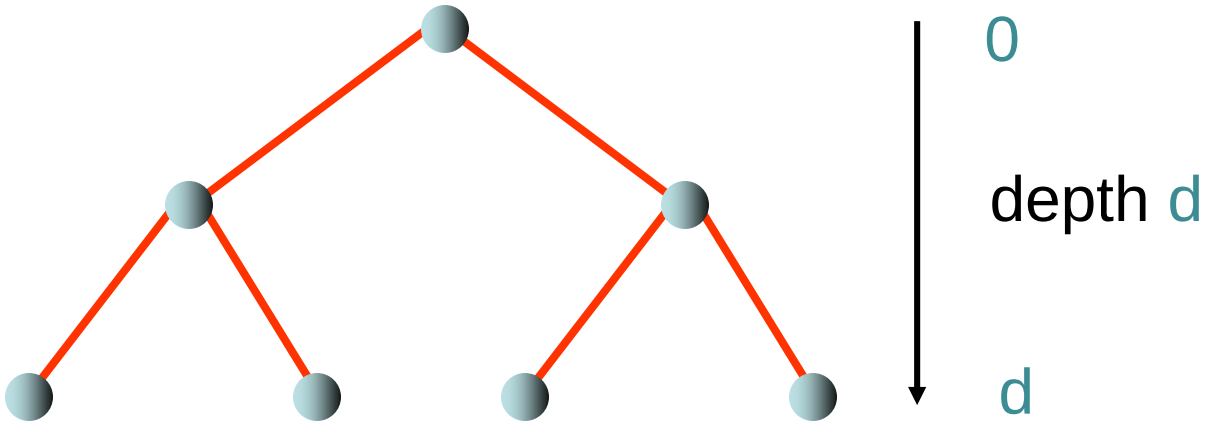
- G is weakly connected: diameter D is finite, if all edges are seen as undirected
- G is strongly connected: D is finite



Linear list:

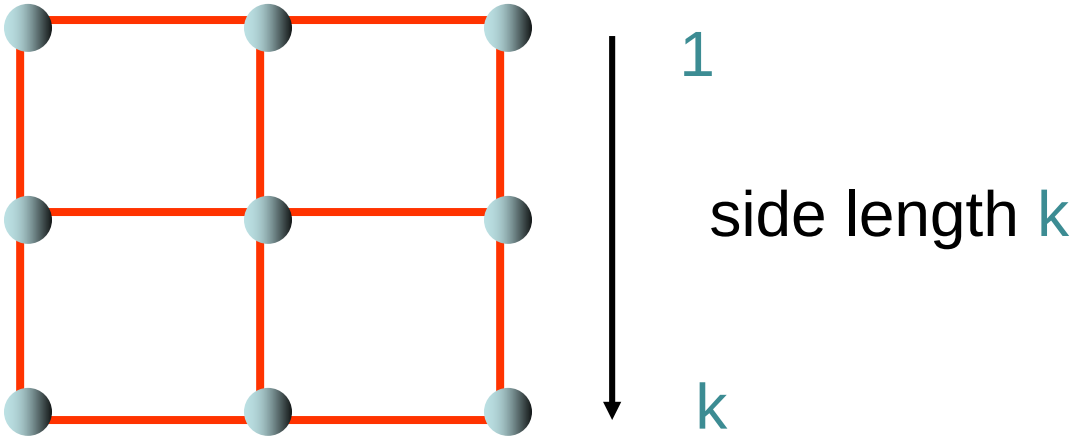
- degree 2
- large diameter (n-1 for n nodes)
  $\rightarrow$ bad for data structures

Complete binary tree:



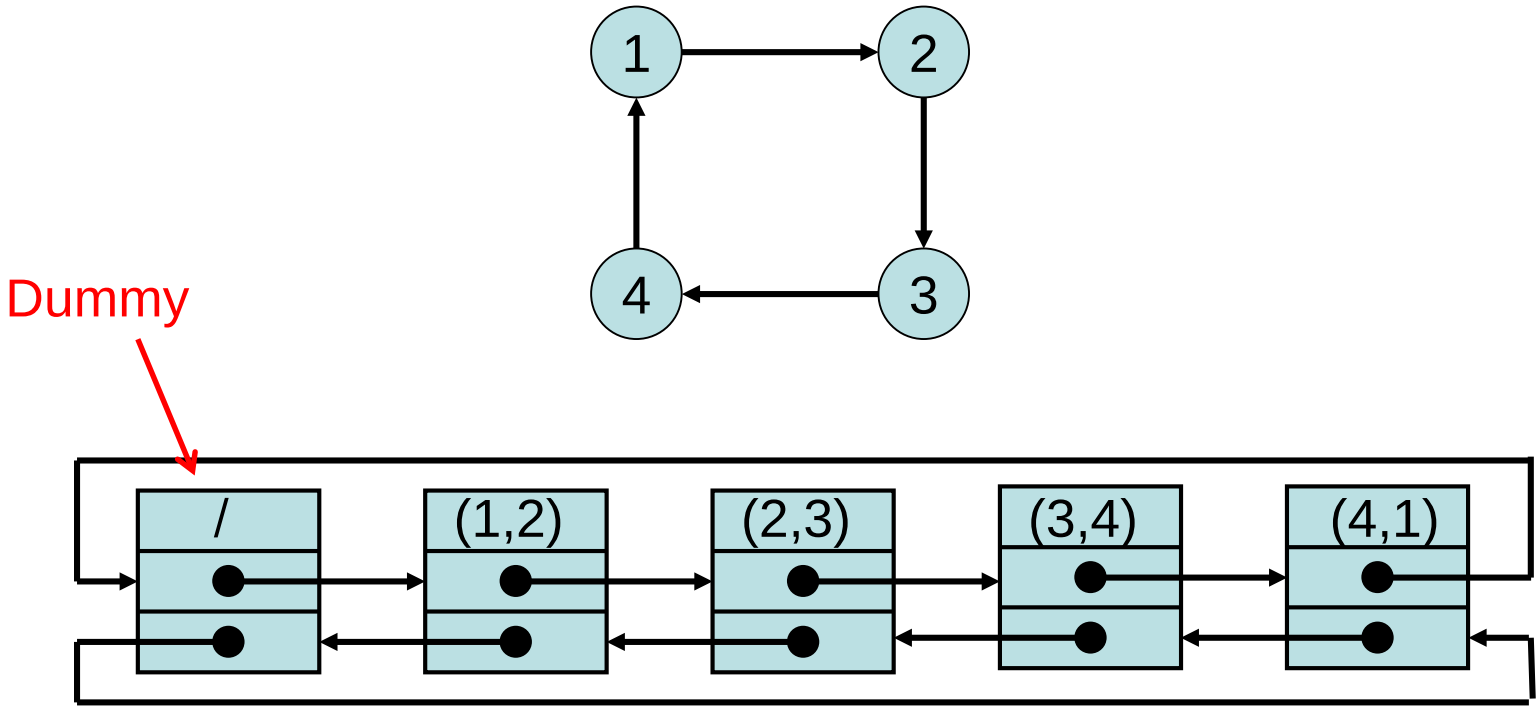
- $n=2^{d+1}-1$ nodes, degree 3
- Diameter is $2d \sim 2 \log_2 n$
  $\rightarrow$ good for data structures

2-dimensional grid:

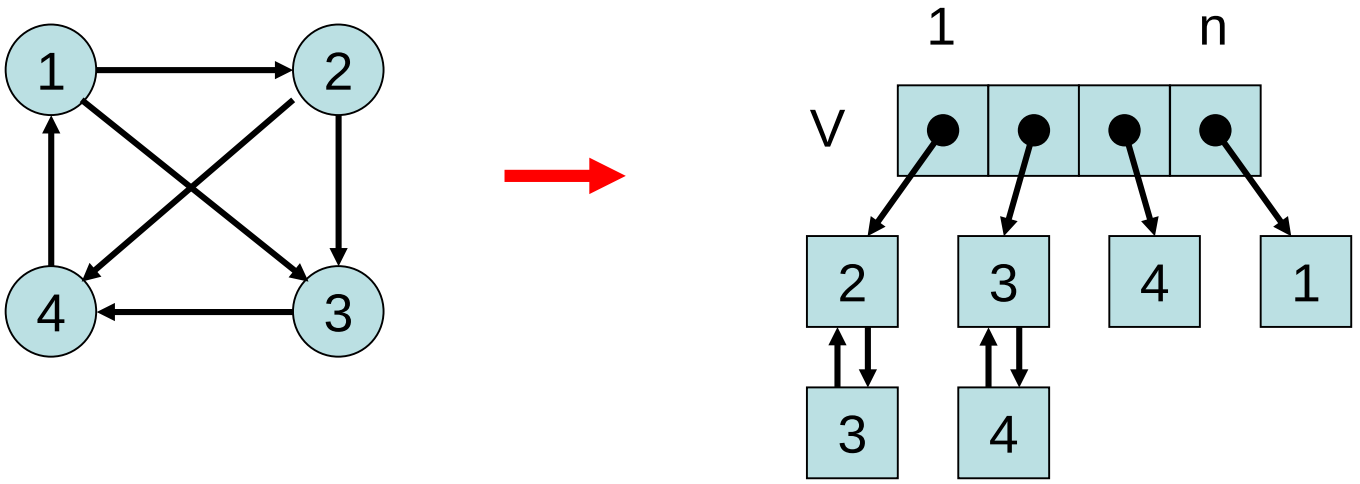


- $n = k^2$ nodes, maximum degree 4
- diameter is $2(k-1) \sim 2\sqrt{n}$

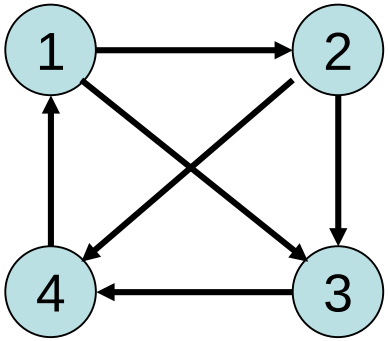
1: Sequence of edges



2: Adjacency list



3: Adjacency matrix

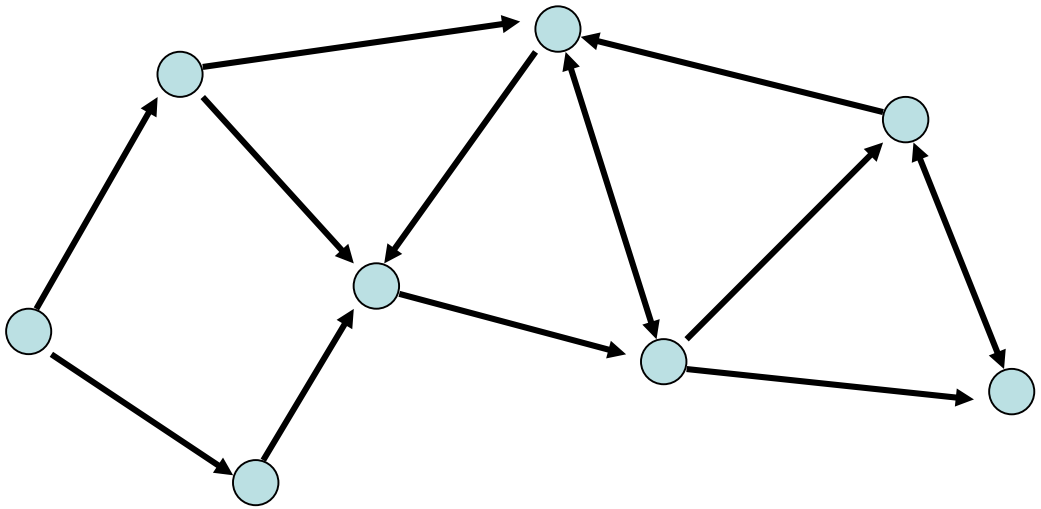


$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

- $A[i,j] \in \{0,1\}$
- $A[i,j]=1$ if and only if $(i,j) \in E$

Graph Traversal

Central question: How can we traverse the nodes of the graph so that every node is visited at least once?



Graph Traversal

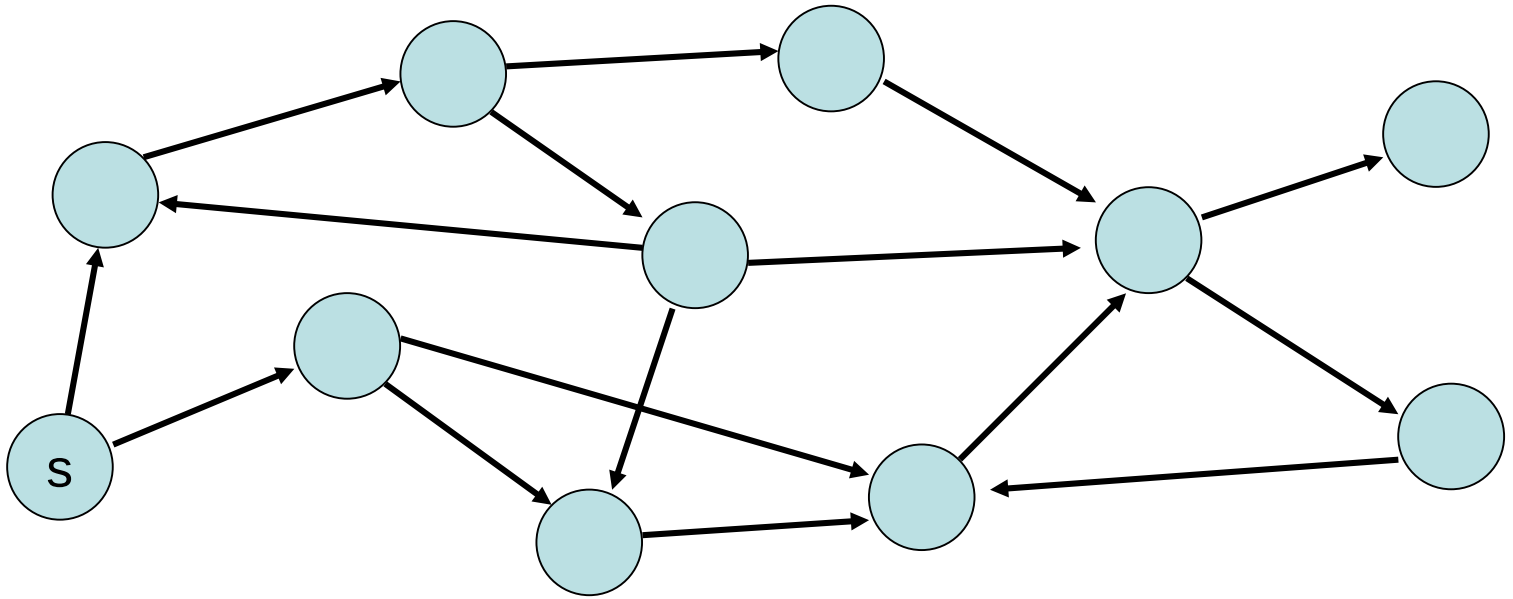
Central question: How can we traverse the nodes of the graph so that every node is visited at least once?

Basic strategies:

- Breadth-first search
- Depth-first search

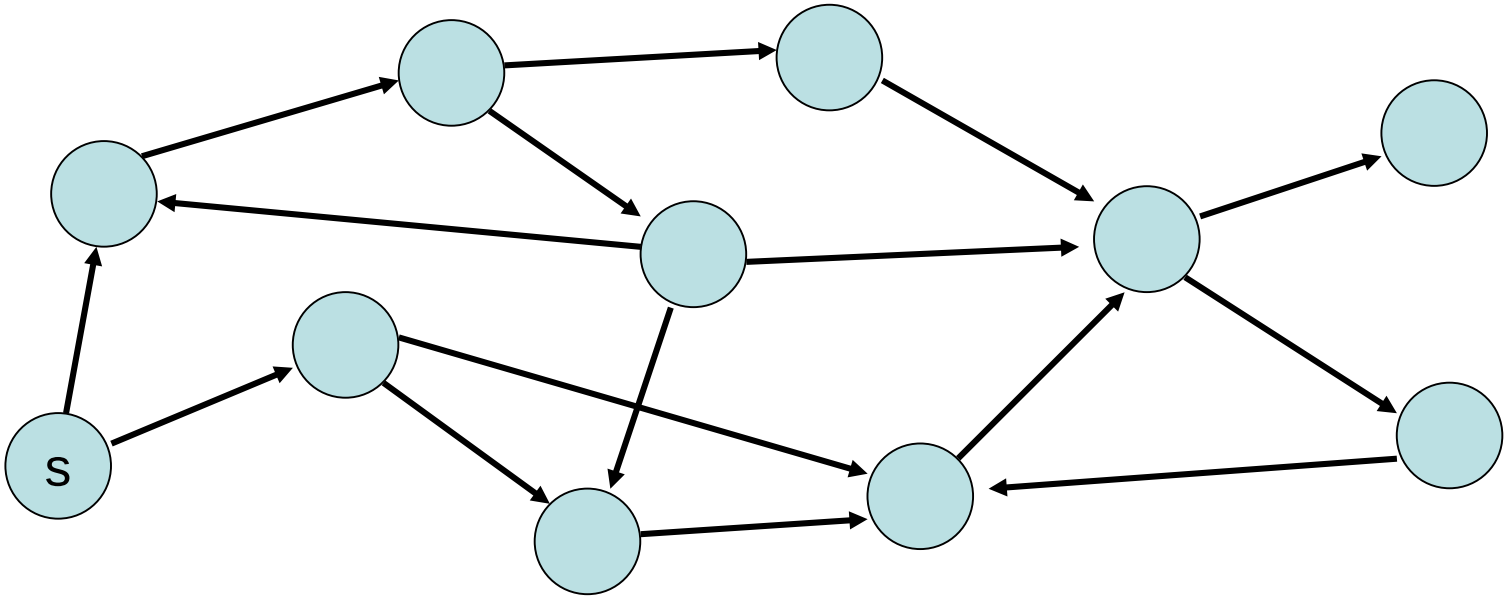
Breadth-First Search

- Start at some node s
- Explore graph distance by distance from s



Depth-First Search

- Start from some node s
- Explorate graph in depth-first manner
  (: current, : still active, : done)



Breadth-First Search

- Breadth-first search algorithm forms the base of many other graph algorithms.
- Given a graph $G=(V,E)$ and a source $s \in V$, the goal of breadth-first search is to find all nodes $v \in V$ that are reachable from s. A node v is reachable from s if there is a (directed) path in G from s to v.
- The breadth-first search algorithm can also be used to compute the distance $\delta(s,v)$ of v from s.

Breadth-First Search

- If a new node v is discovered while searching for new nodes in $Adj[u]$, then u is called the predecessor of v.
  ($Adj[u]$: set of neighbors or adjacency list of u, i.e., the nodes $v \in V$ with $\{u,v\} \in E$ resp. $(u,v) \in E$)
- In the algorithm: nodes are white, gray, or black.
- White nodes are nodes that have not been discovered yet.
- Gray nodes are nodes that have already been discovered but whose adjacency list has not completely been explored yet.
- Black nodes are nodes that have already been discovered and fully processed.

Pseudo-code for Breadth-First Search

```
BFS(G,S)
1    for each node u∈V\{s} do
2        color[u]←WHITE
3        d[u] ← ∞
4        p[u] ← NIL
5    color[s] ← GRAY
6    d[s] ← 0
7    π[s] ← NIL
8    Q ← { }
9    Enqueue(Q,s)
10   while Q ≠ { } do
11       u ← Dequeue(Q)
12       for each v∈Adj[u] do
13           if color[v]=WHITE then
14               color[v] ← GRAY
15               d[v] ← d[u]+1
16               π[v] ← u
17               Enqueue(Q,v)
18       color[u] ← BLACK
```

- BFS uses queue for gray nodes.
- color[u]:=stores color of v. Initially WHITE.
- d[u]:=field for distance to s. Initially ∞.
- π[u]:=field for predecessor. Initially NIL.

Pseudo-code for Breadth-First Search

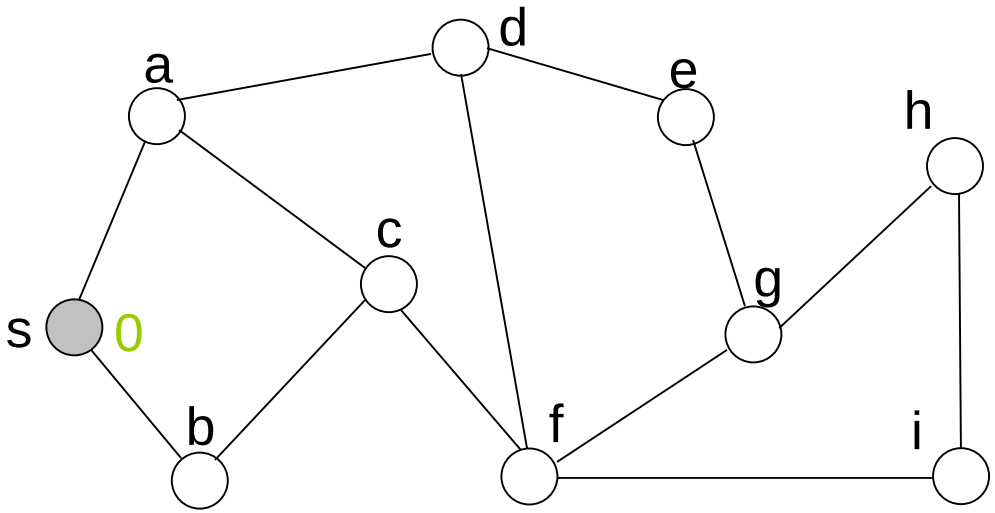
```
BFS(G,S)
1    for each node u∈V\{s} do
2        color[u]←WHITE
3        d[u] ← ∞
4        p[u] ← NIL
5    color[s] ← GRAY
6    d[s] ← 0
7    π[s] ← NIL
8    Q ← { }
9    Enqueue(Q,s)
10   while Q ≠ { } do
11       u ← Dequeue(Q)
12       for each v∈Adj[u] do
13           if color[v]=WHITE then
14               color[v] ← GRAY
15               d[v] ← d[u]+1
16               π[v] ← u
17               Enqueue(Q,v)
18       color[u] ← BLACK
```

- BFS uses queue for gray nodes.
- color[u]:=stores color of v. Initially WHITE.
- d[u]:=field for distance to s. Initially ∞.
- π[u]:=field for predecessor. Initially NIL.

initialize BFS

Breadth-First Search Example

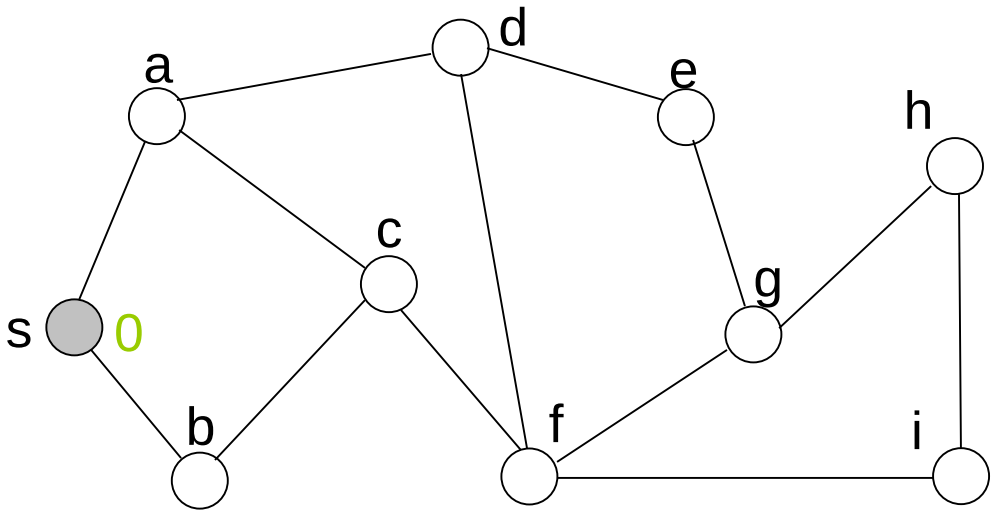
```
BFS(G,s)
1.  „initialize BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: s

Breadth-First Search Example

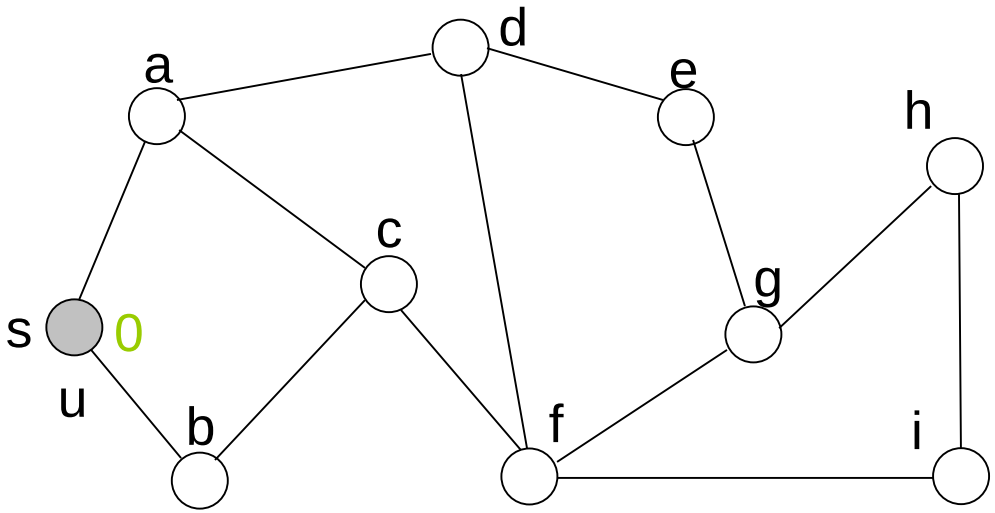
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: s

Breadth-First Search Example

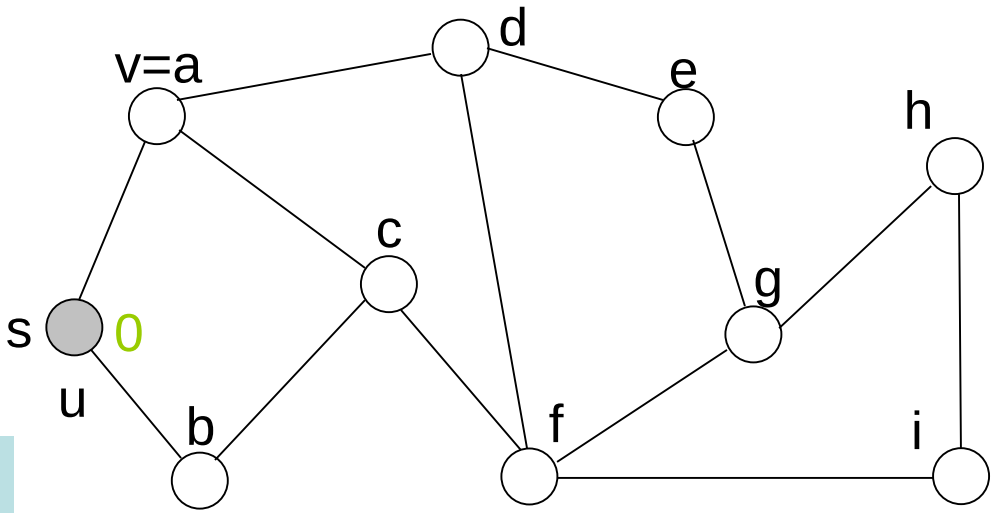
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

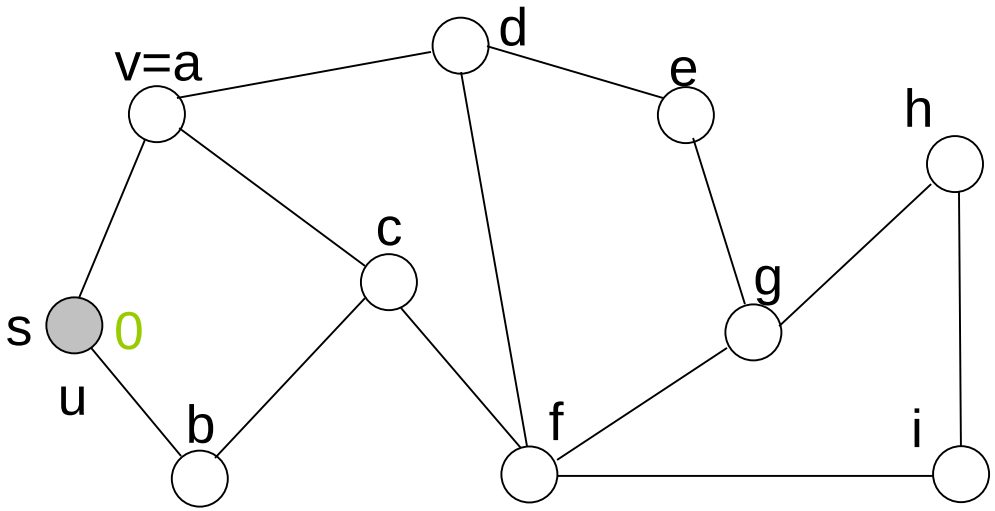
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

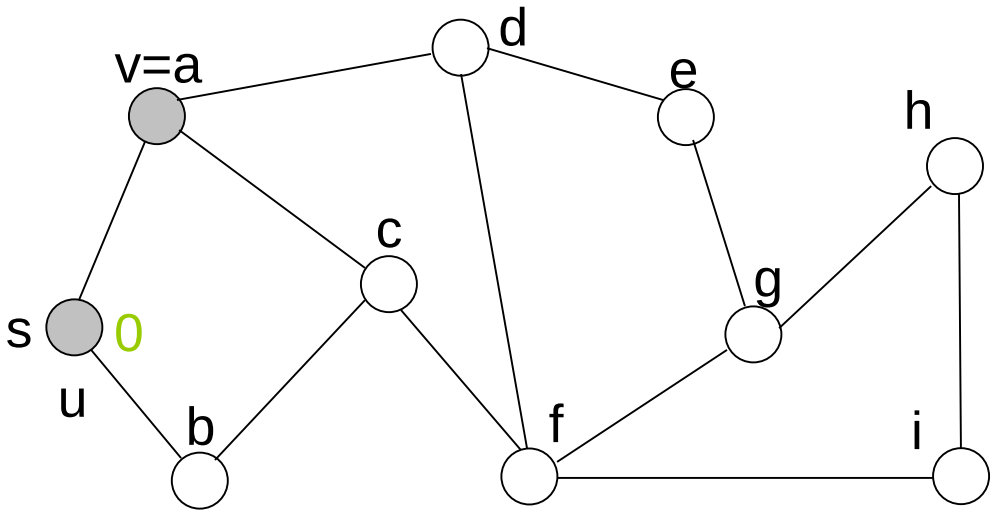
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

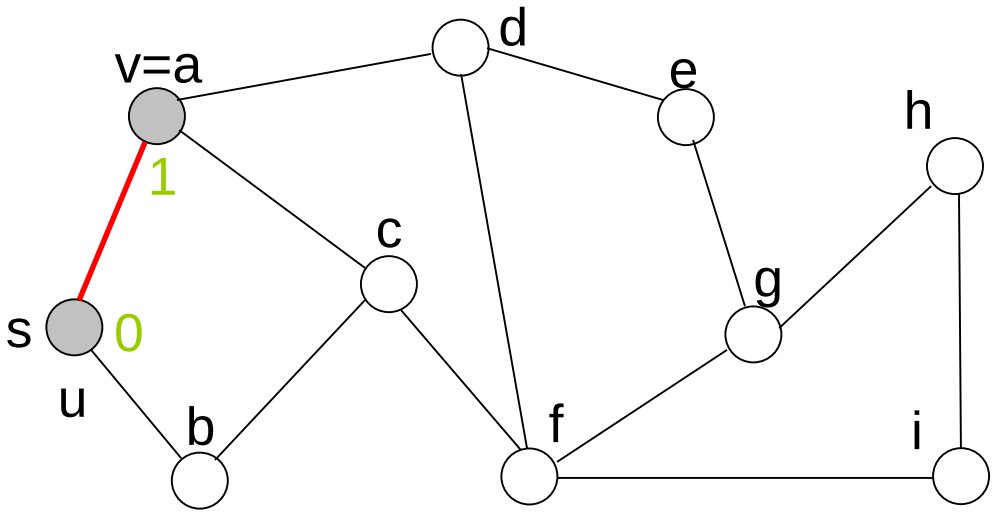
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

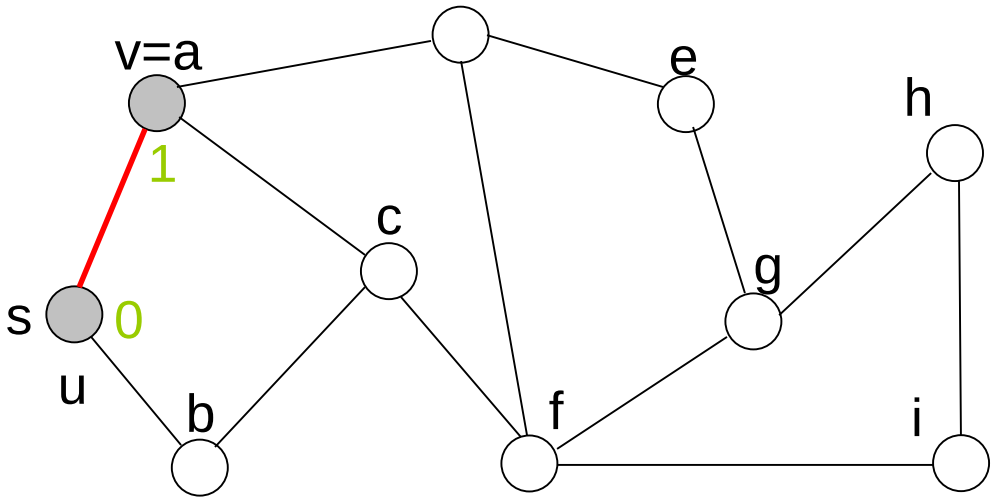
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

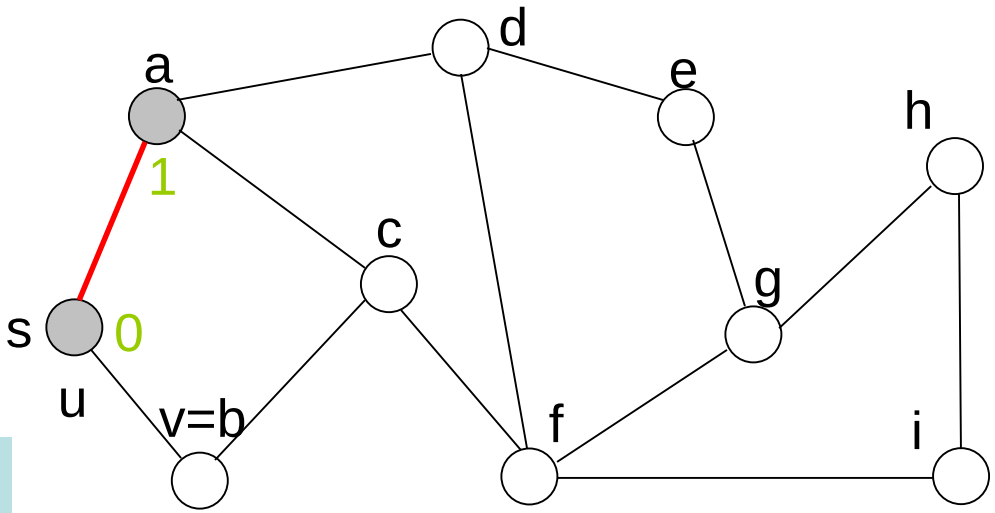
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breadth-First Search Example

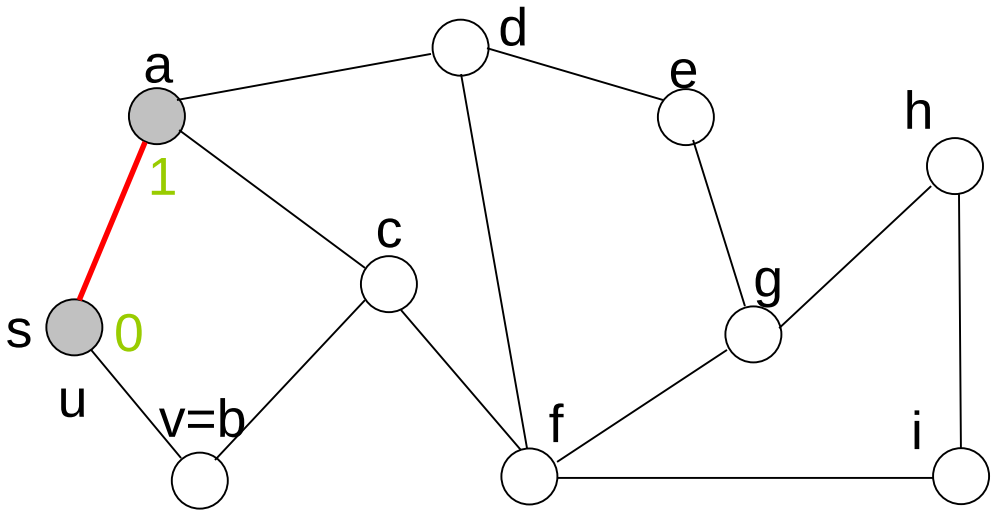
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breadth-First Search Example

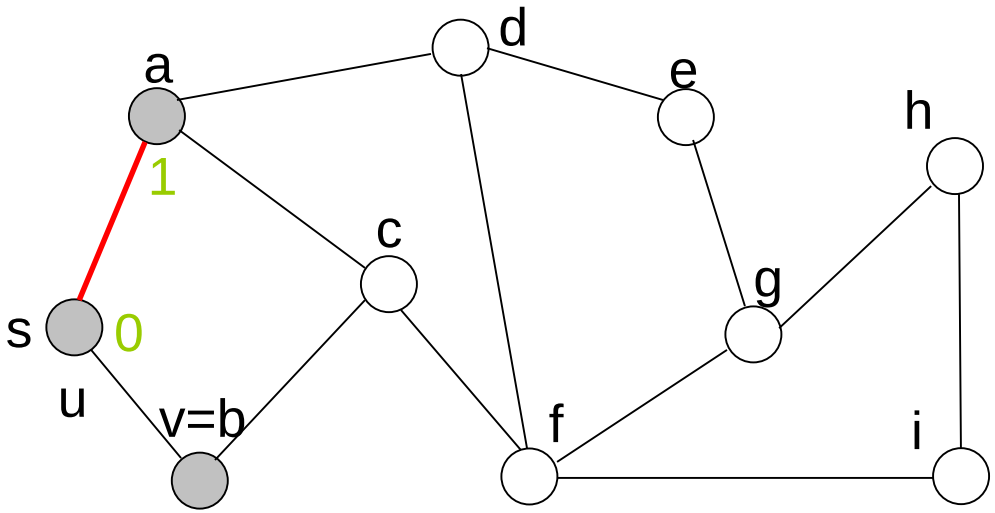
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breadth-First Search Example

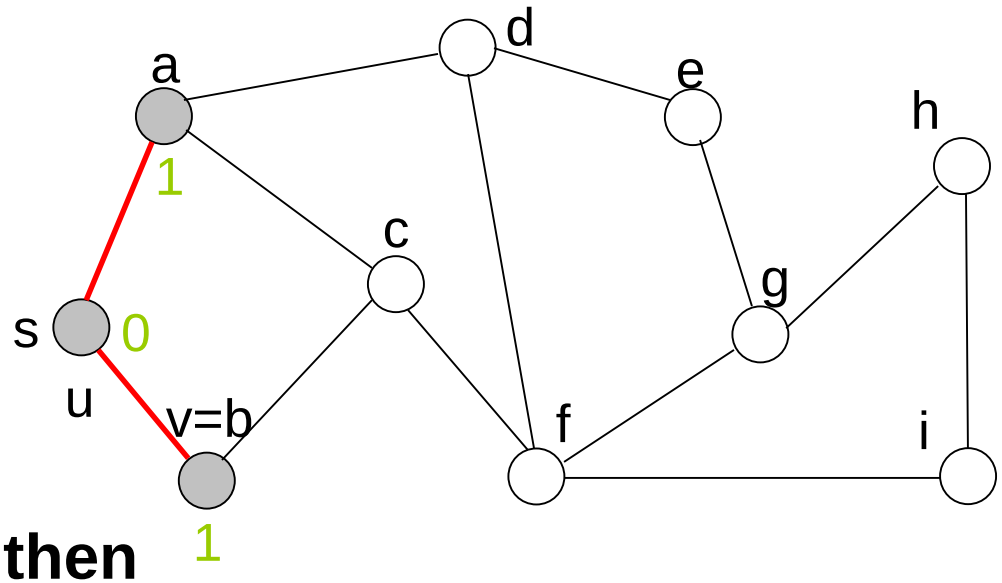
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breadth-First Search Example

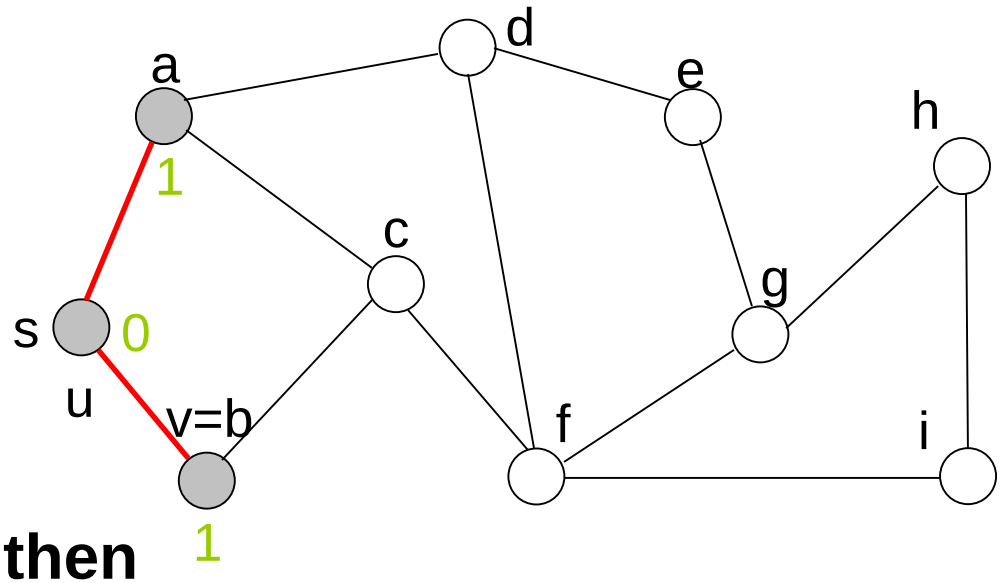
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breadth-First Search Example

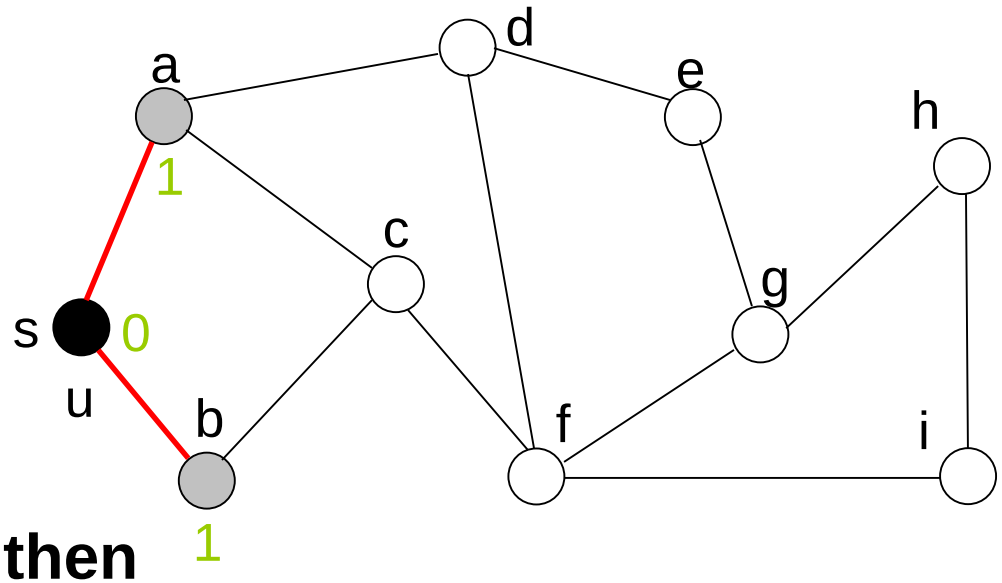
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breadth-First Search Example

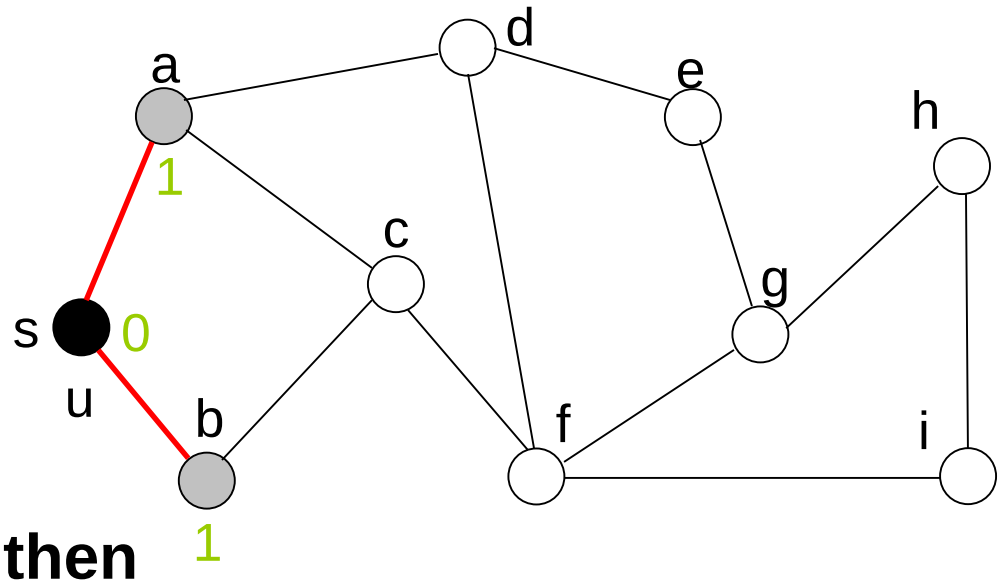
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breadth-First Search Example

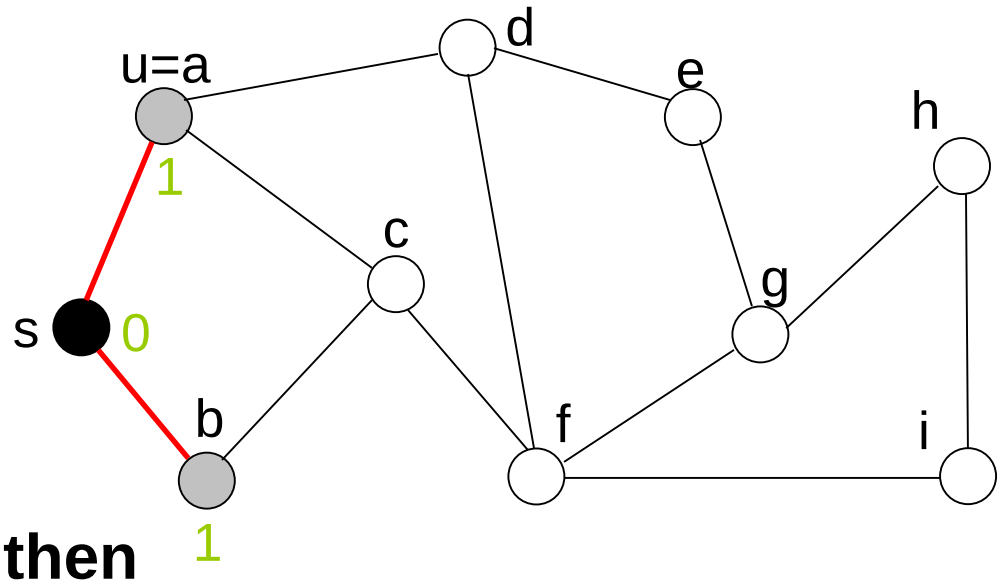
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breadth-First Search Example

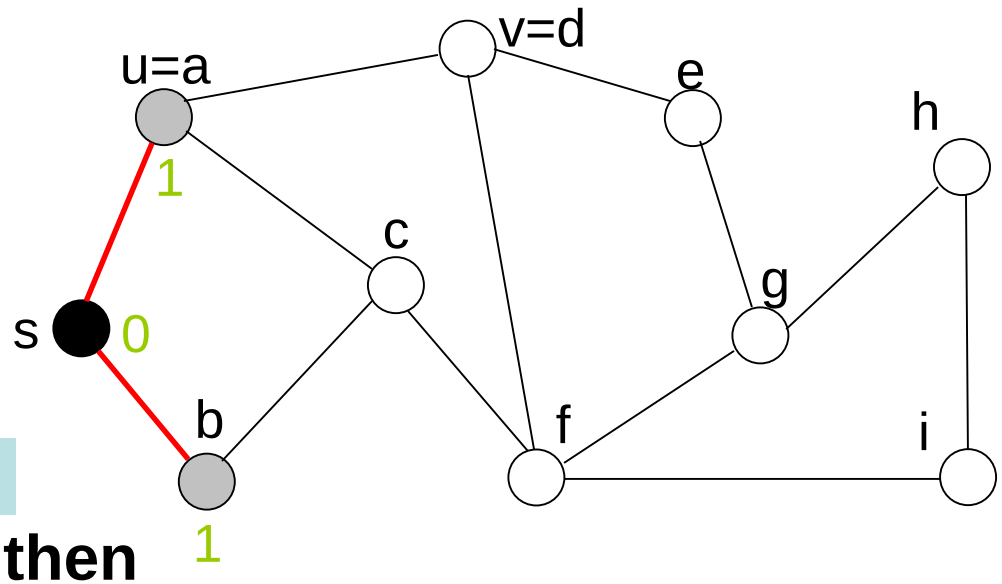
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breadth-First Search Example

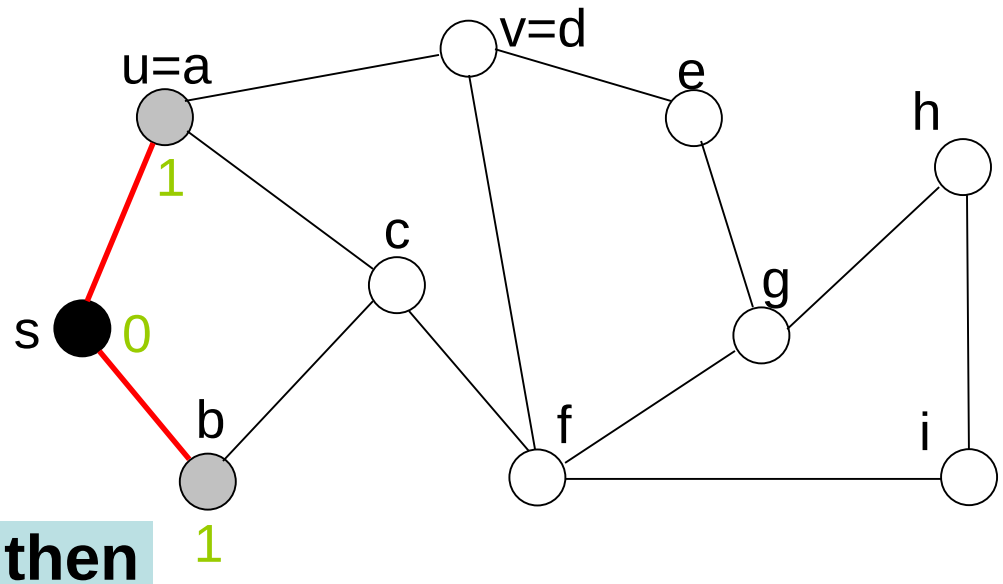
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breadth-First Search Example

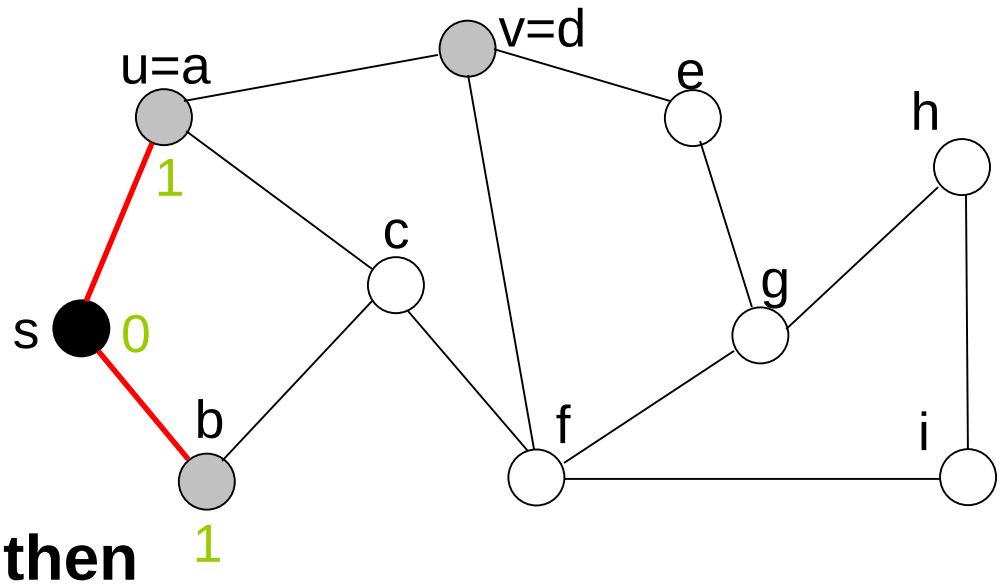
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breadth-First Search Example

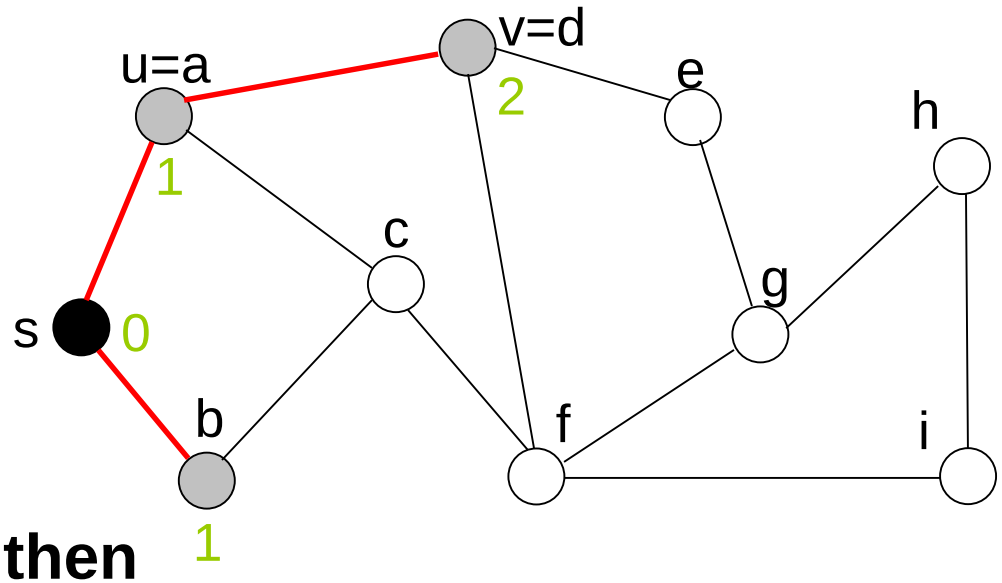
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breadth-First Search Example

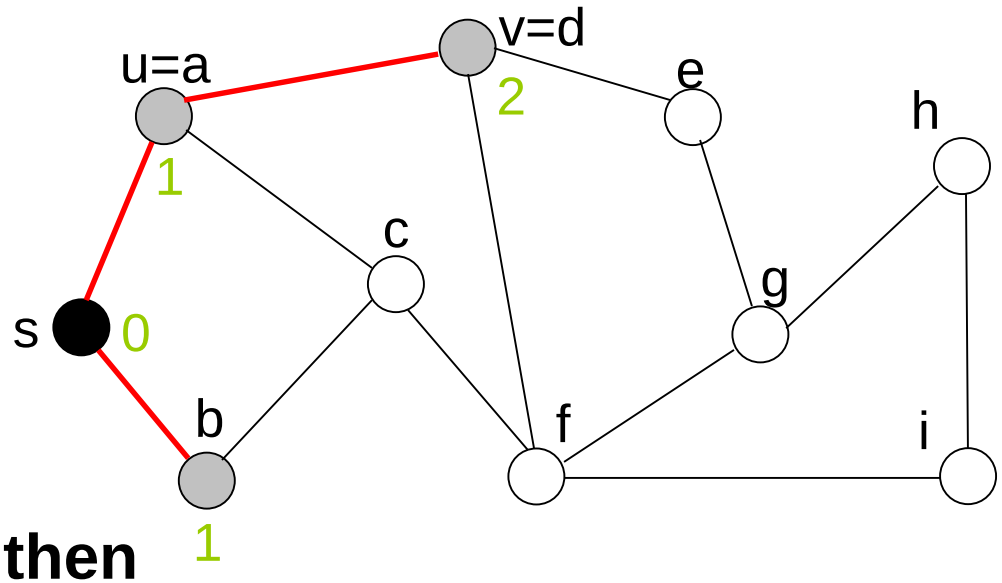
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breadth-First Search Example

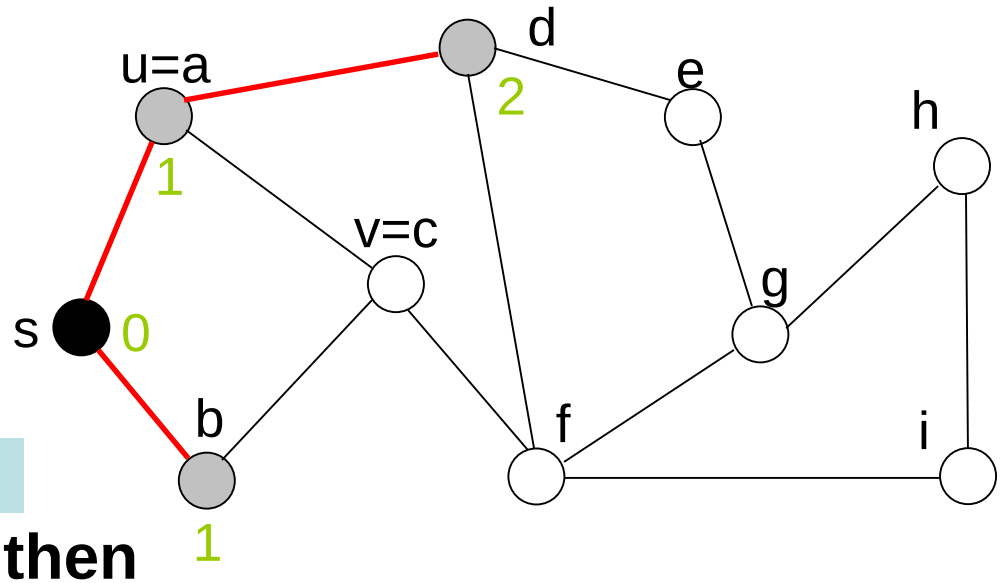
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breadth-First Search Example

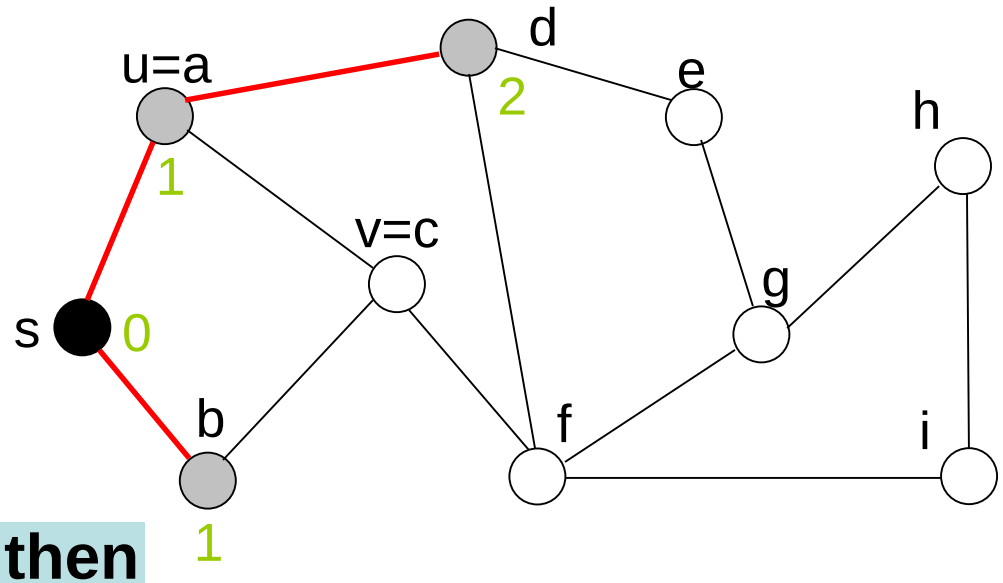
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breadth-First Search Example

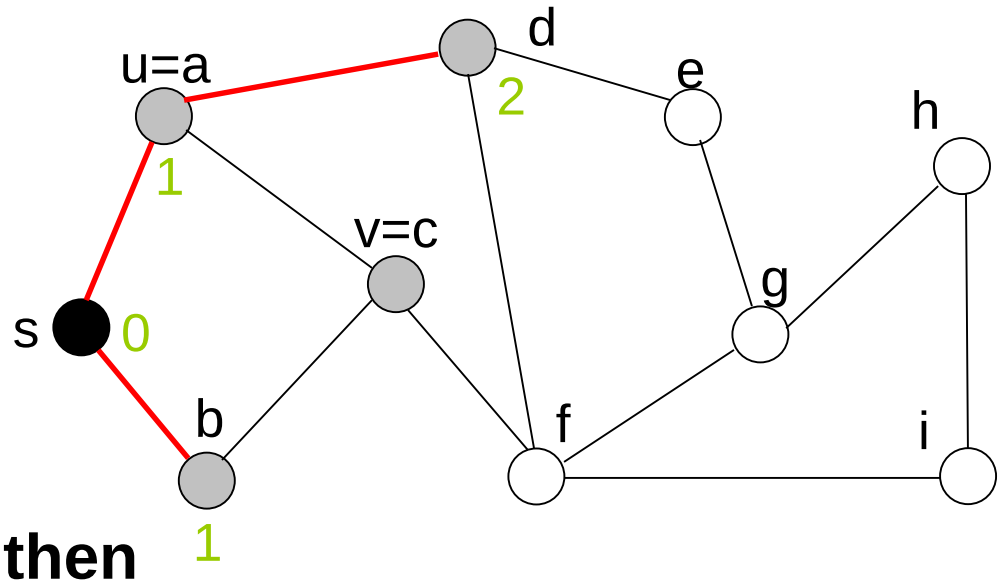
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breadth-First Search Example

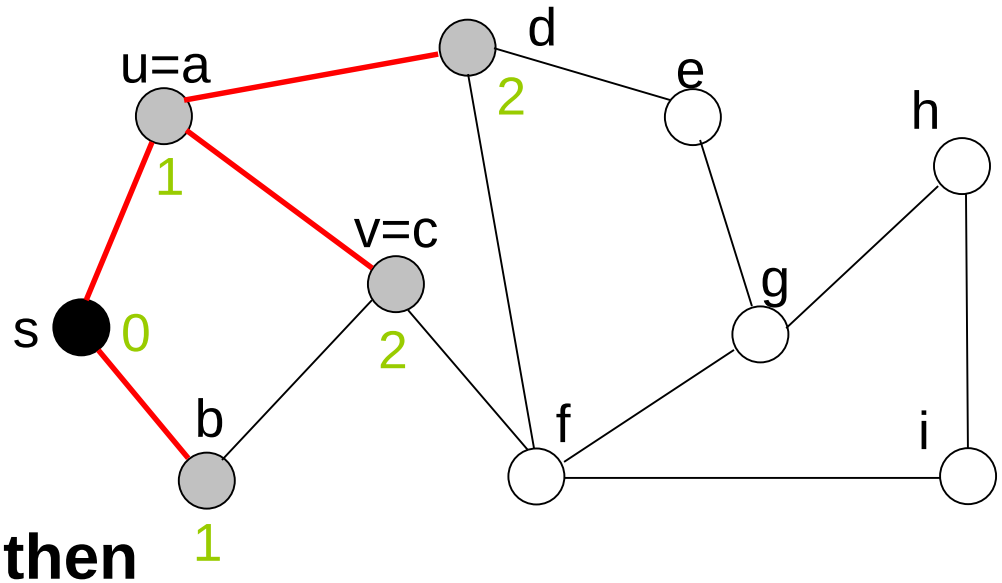
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breadth-First Search Example

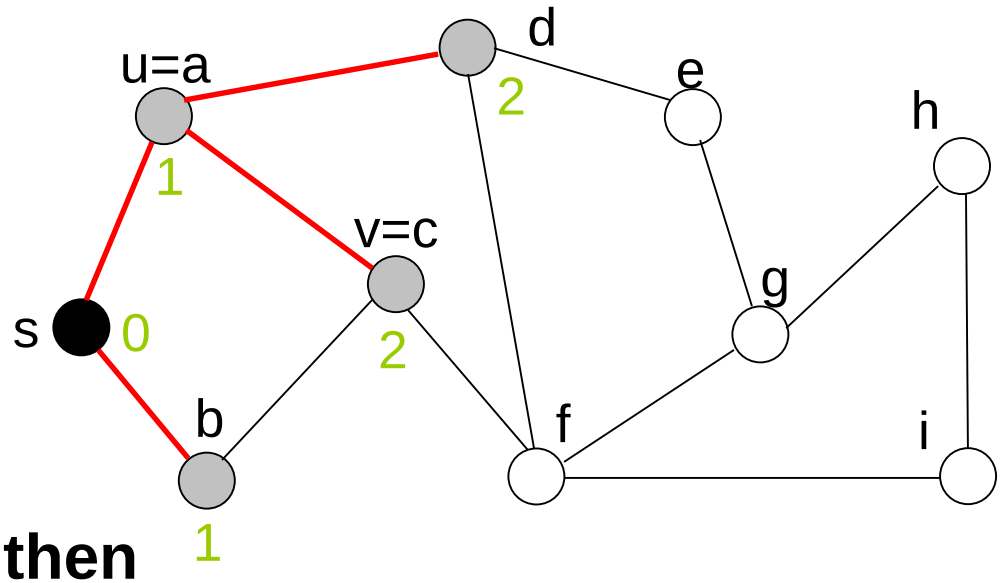
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breadth-First Search Example

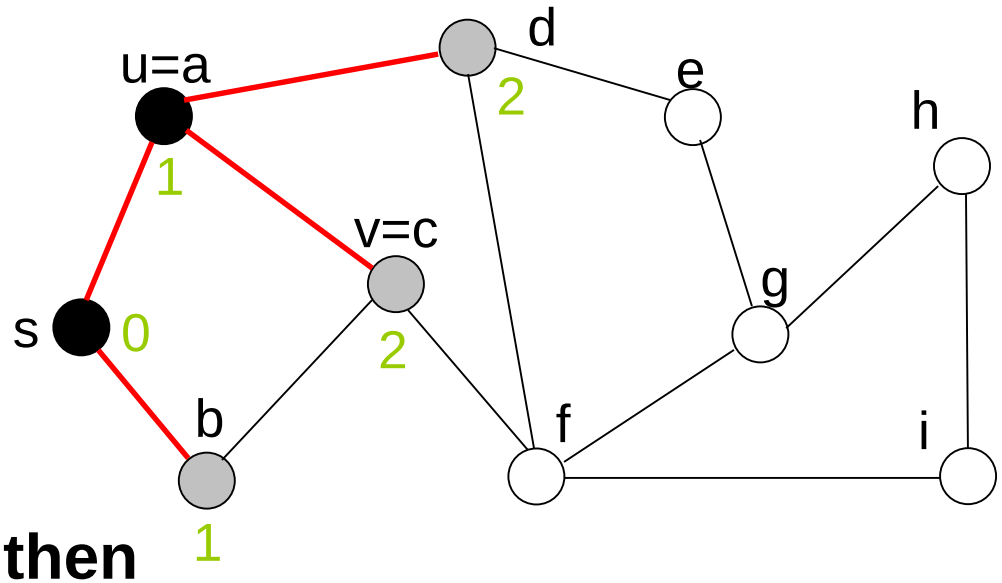
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breadth-First Search Example

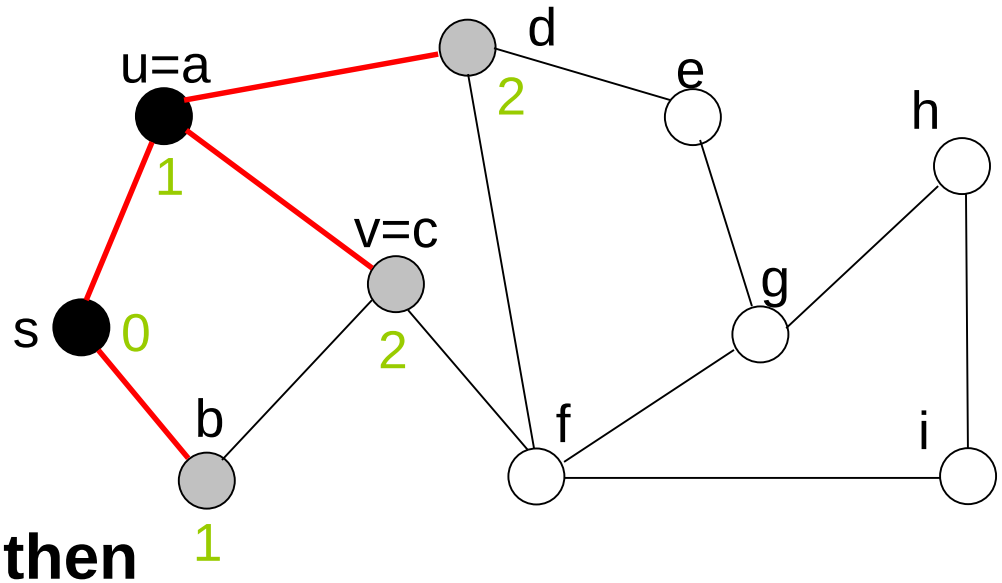
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breadth-First Search Example

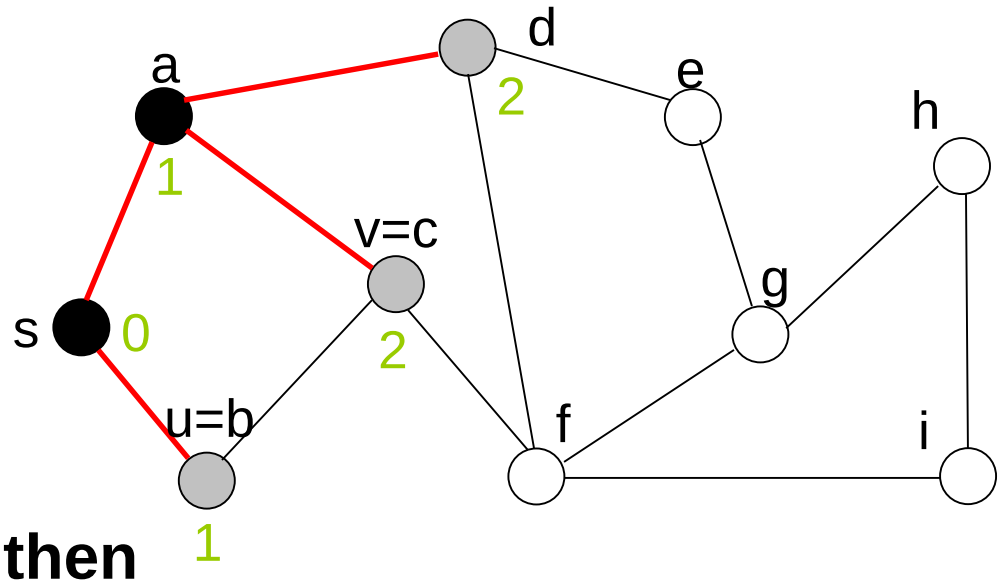
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breadth-First Search Example

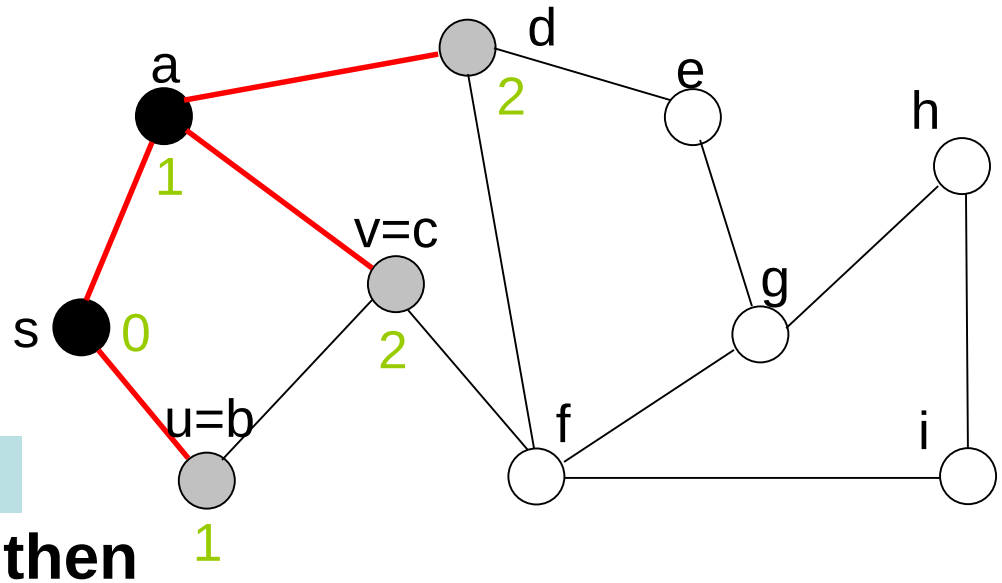
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breadth-First Search Example

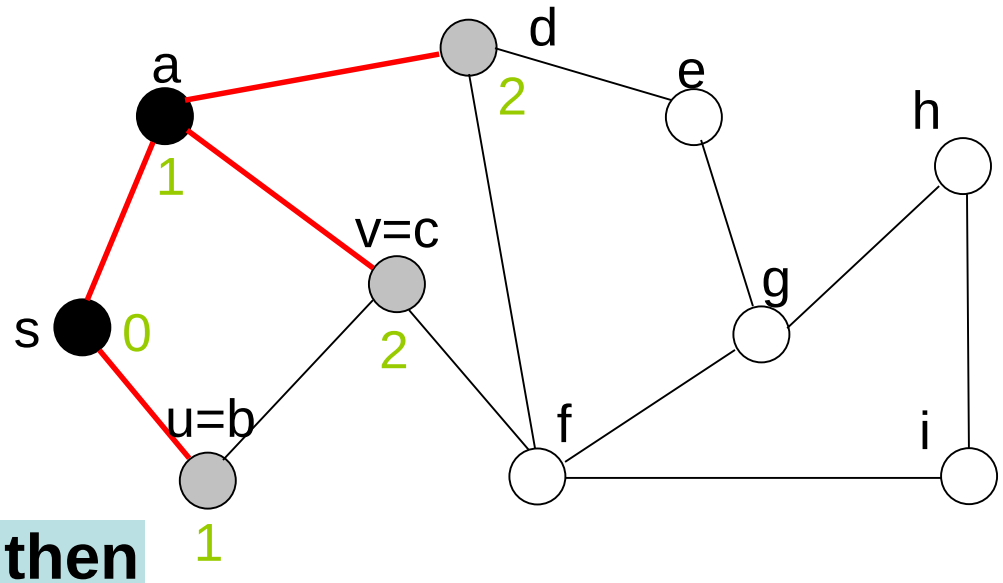
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breadth-First Search Example

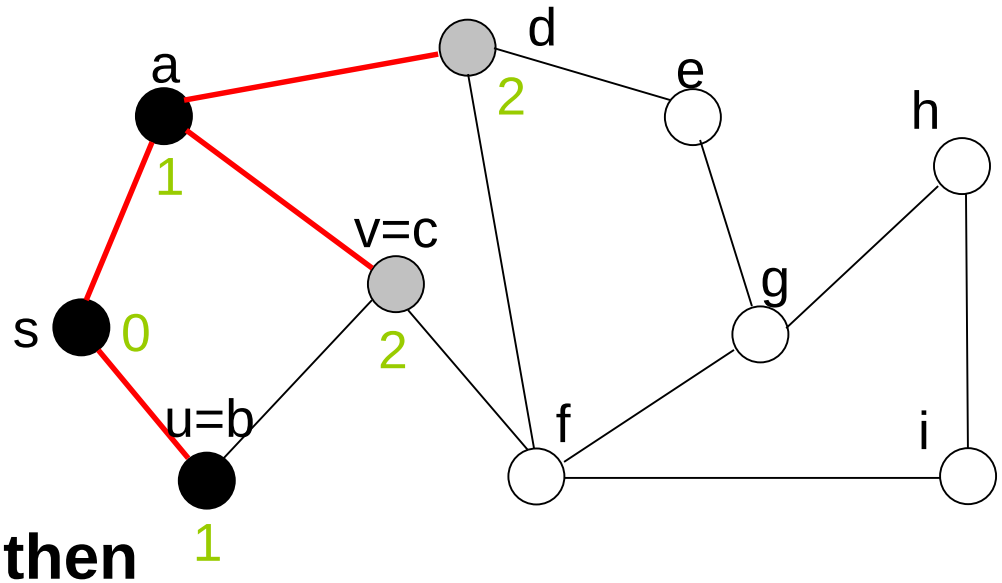
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breadth-First Search Example

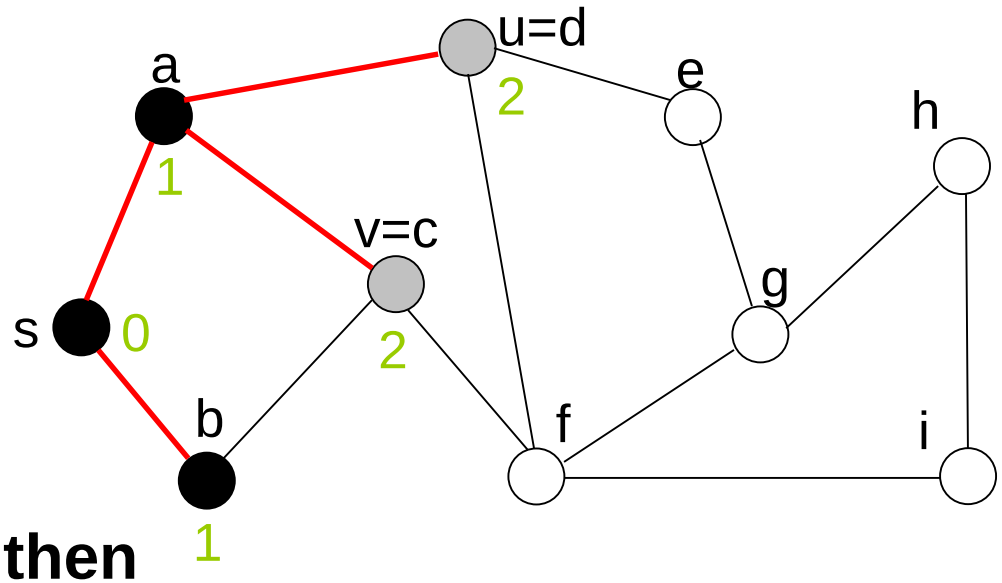
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breadth-First Search Example

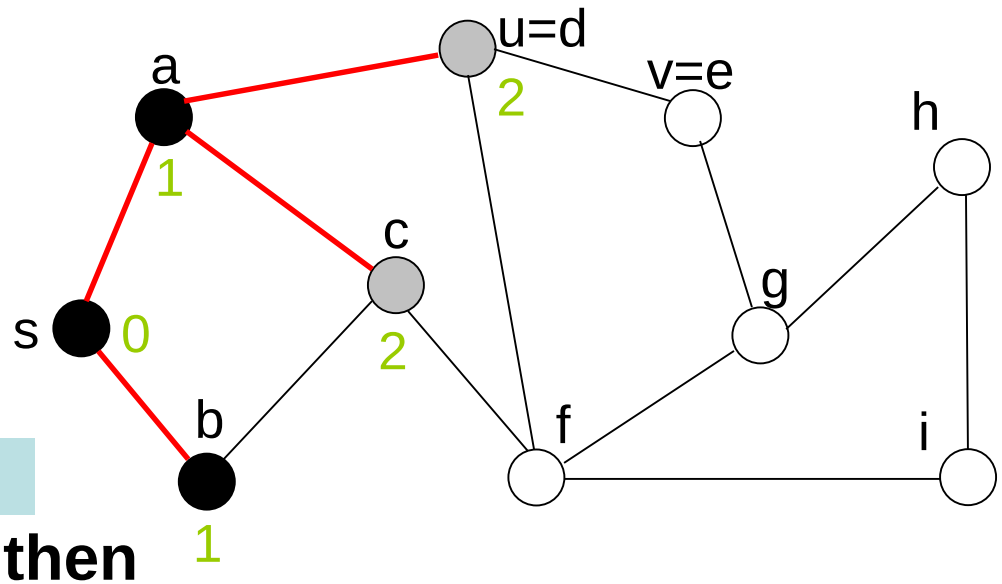
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c

Breadth-First Search Example

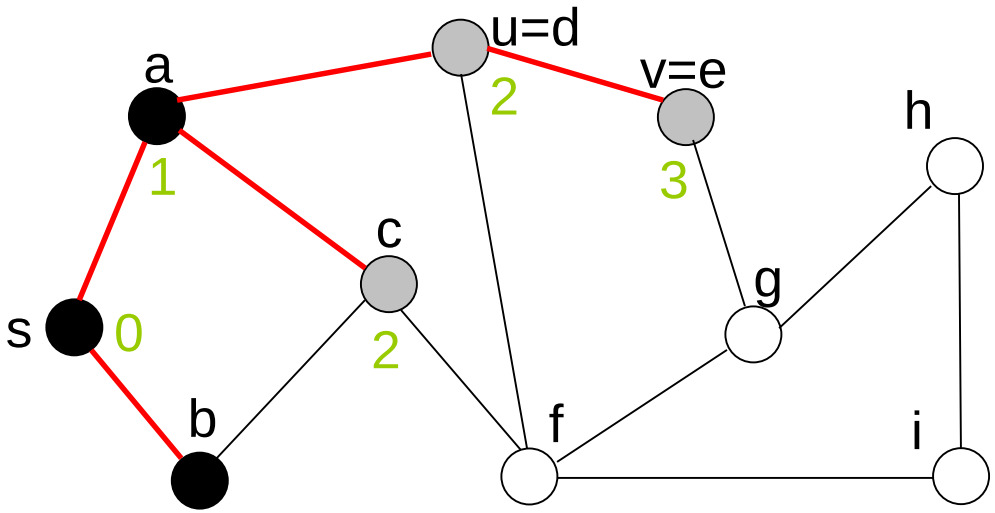
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c

Breadth-First Search Example

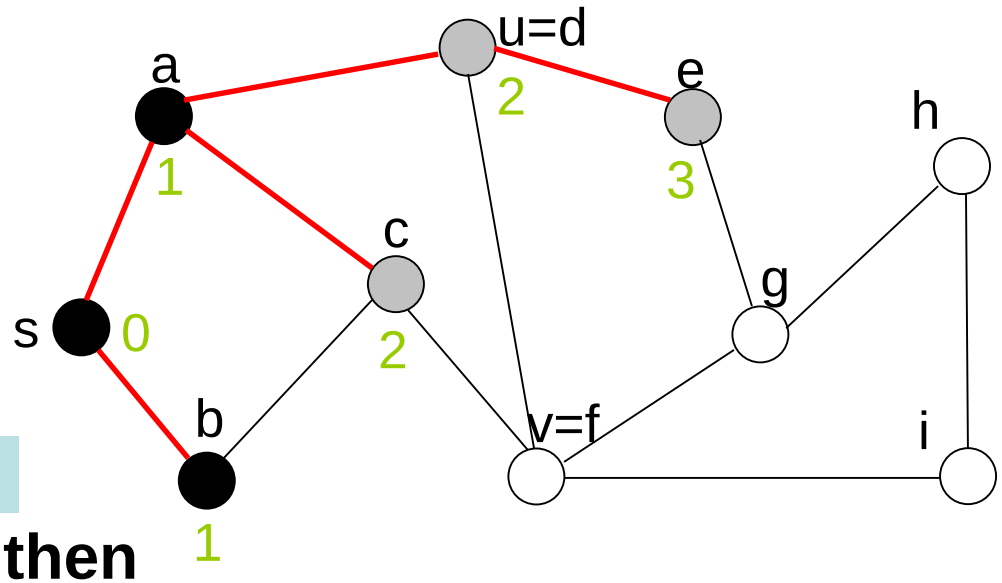
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e

Breadth-First Search Example

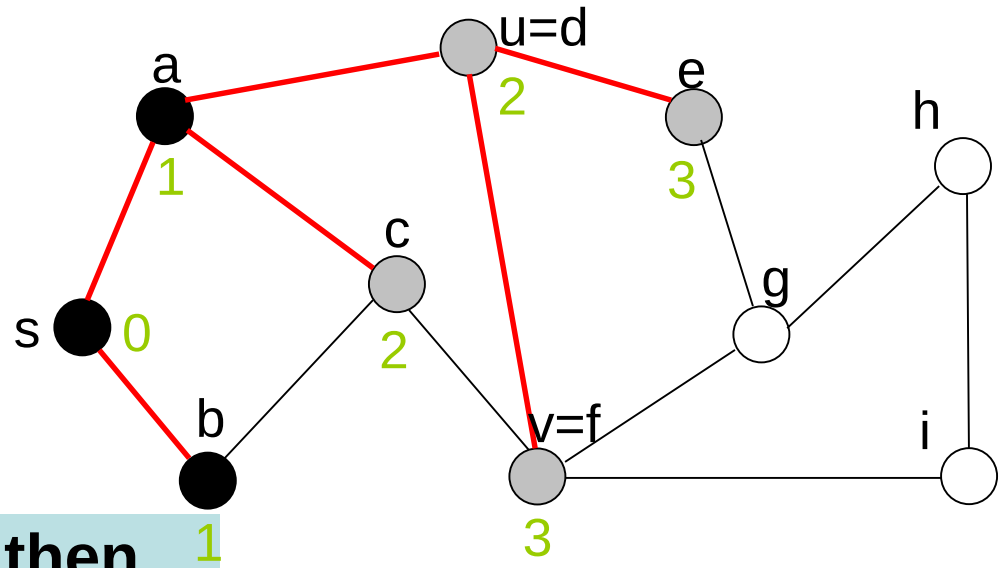
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e

Breadth-First Search Example

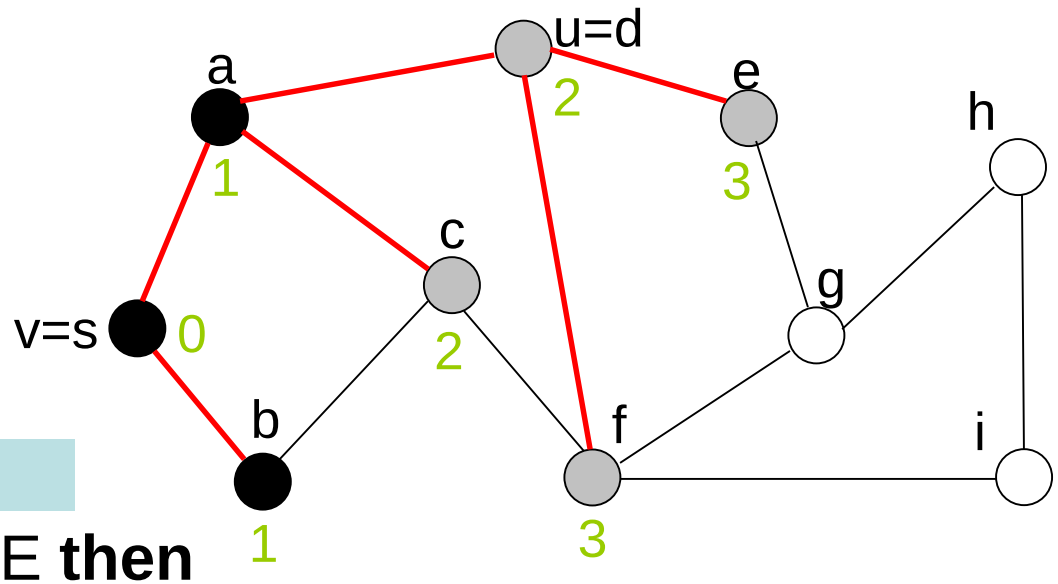
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breadth-First Search Example

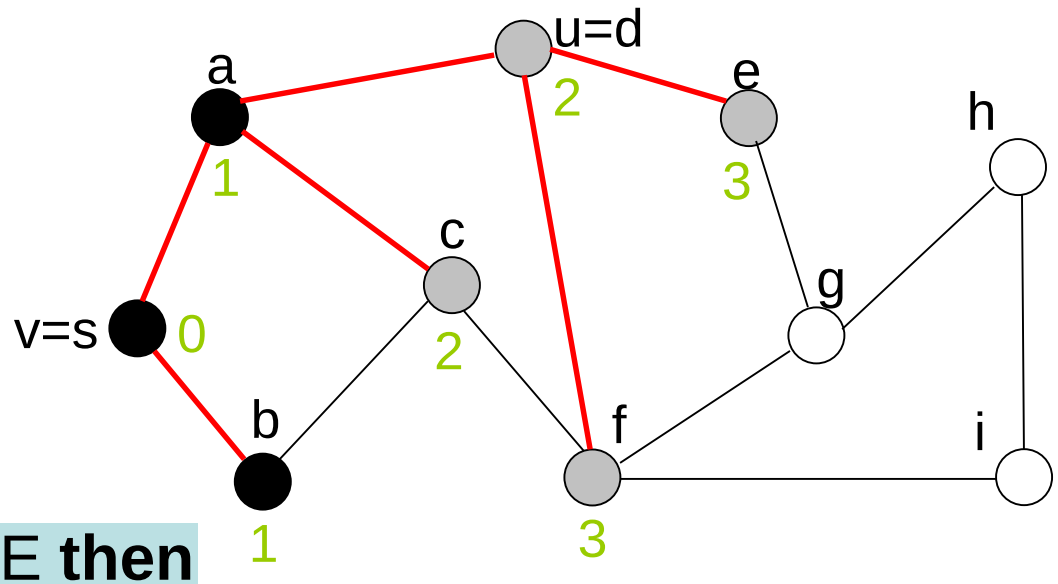
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breadth-First Search Example

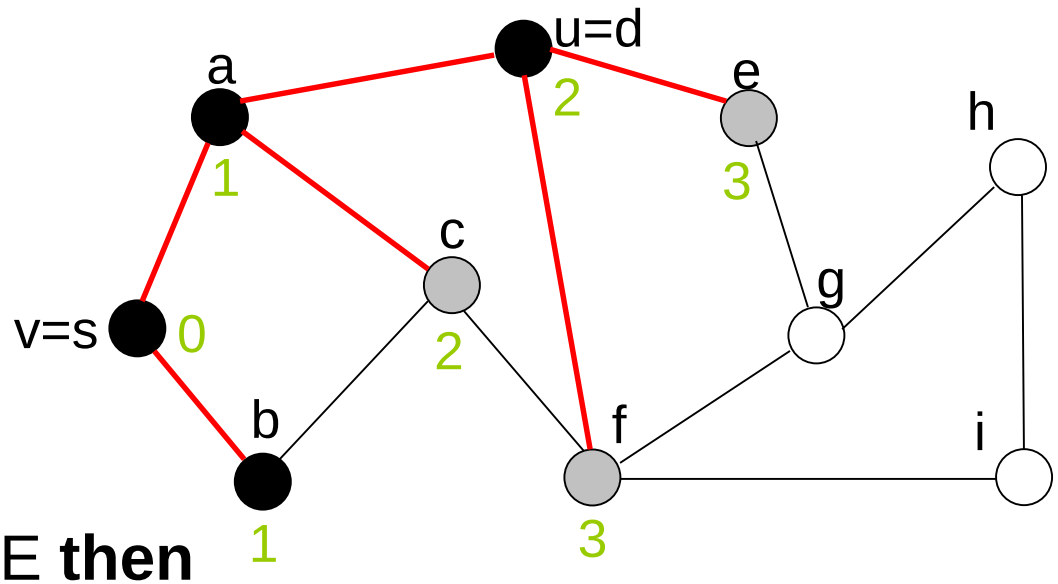
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breadth-First Search Example

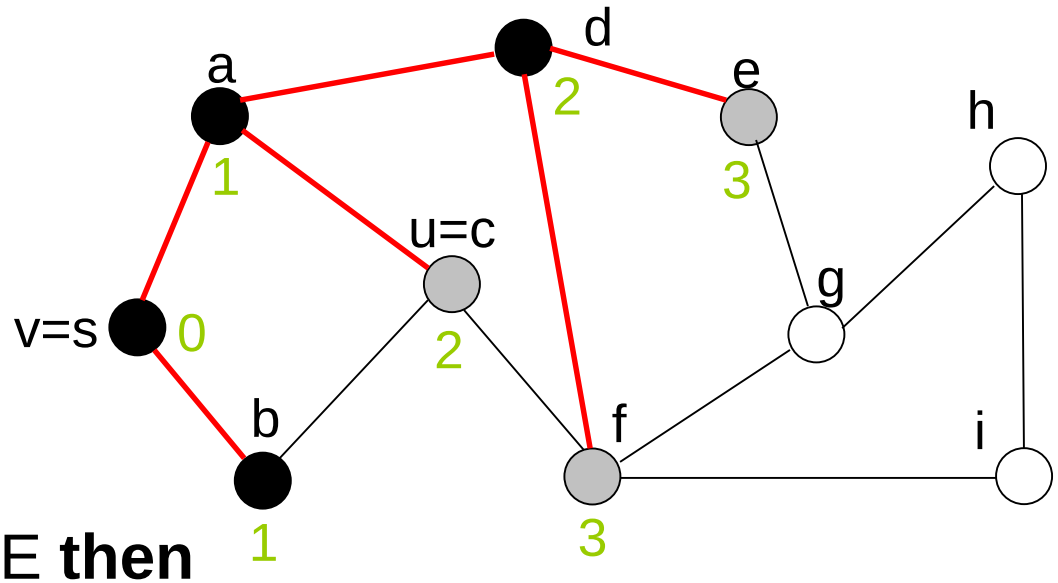
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breadth-First Search Example

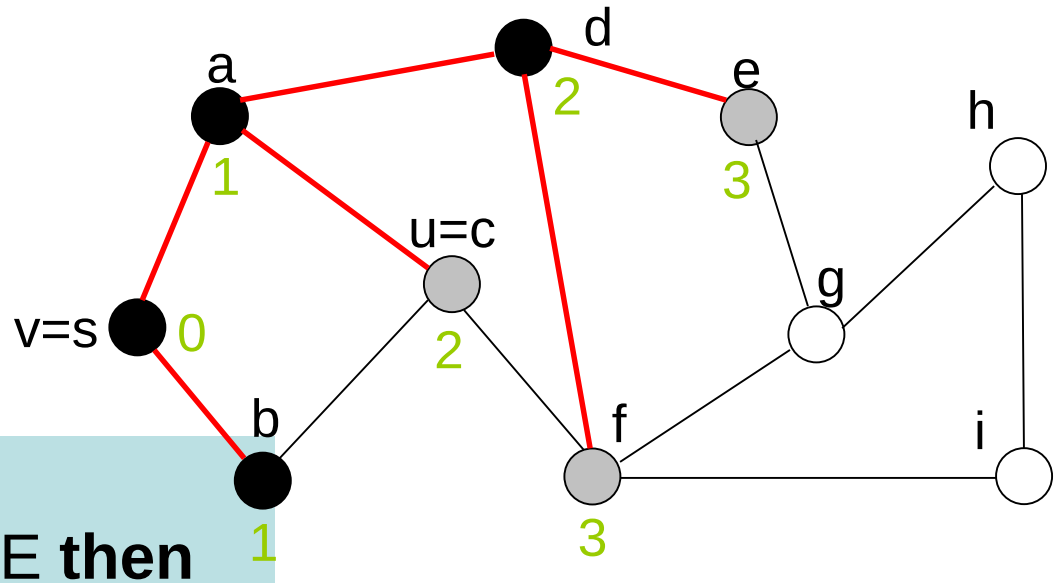
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: e, f

Breadth-First Search Example

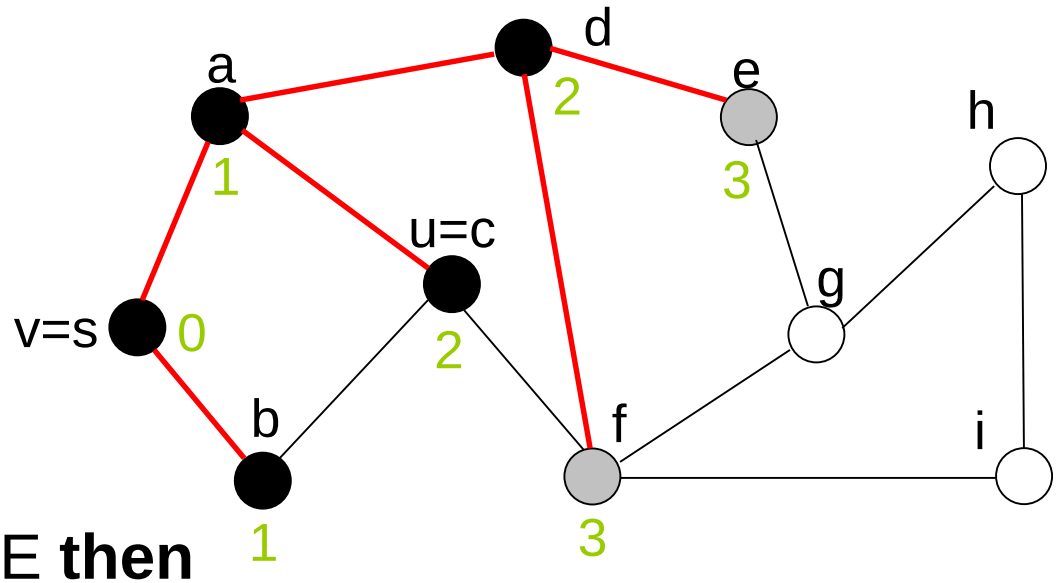
```
BFS(G,s)
1.  „initialize BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: e, f

Breadth-First Search Example

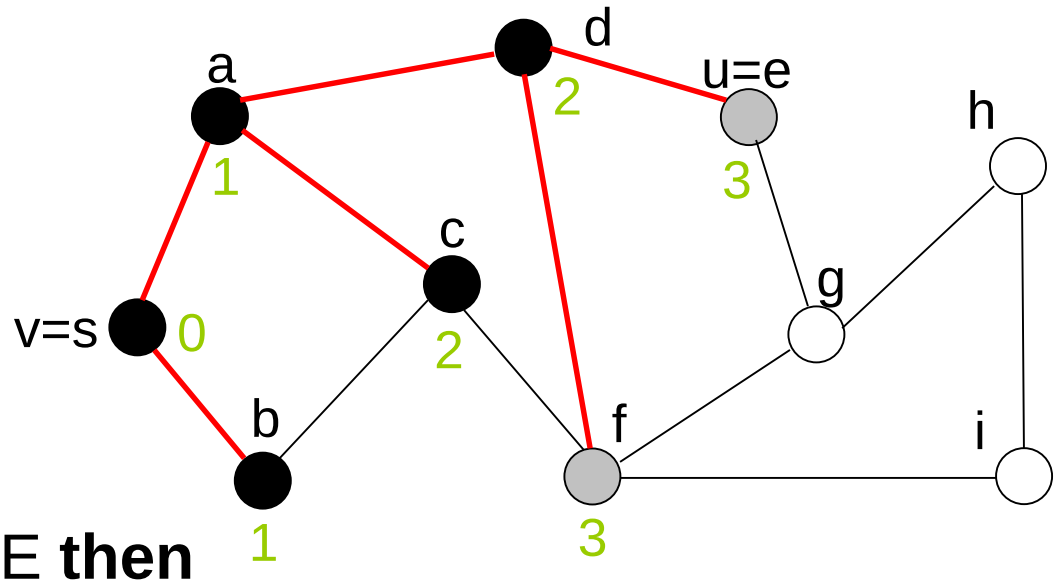
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: e, f

Breadth-First Search Example

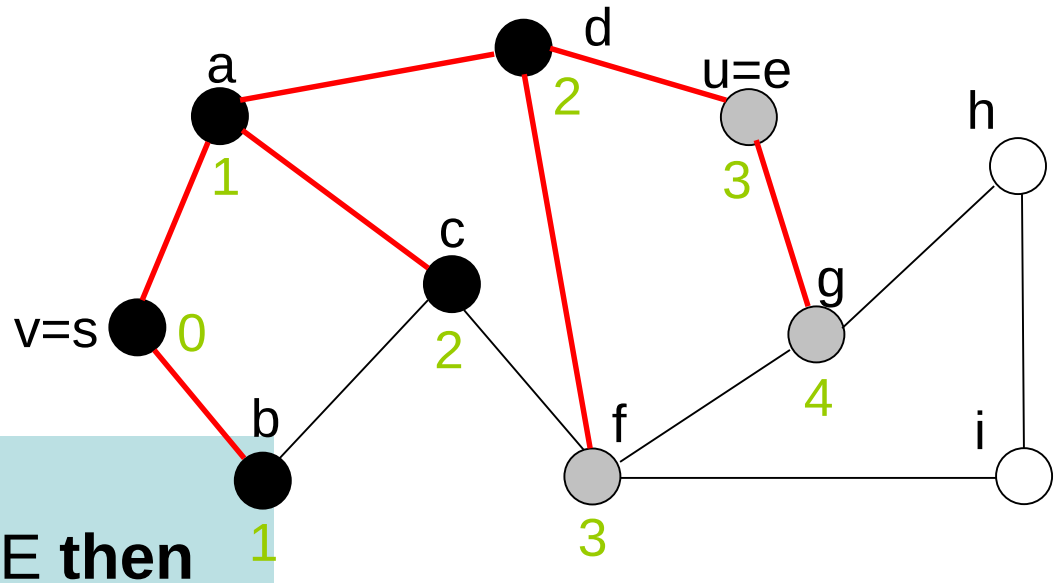
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: f

Breadth-First Search Example

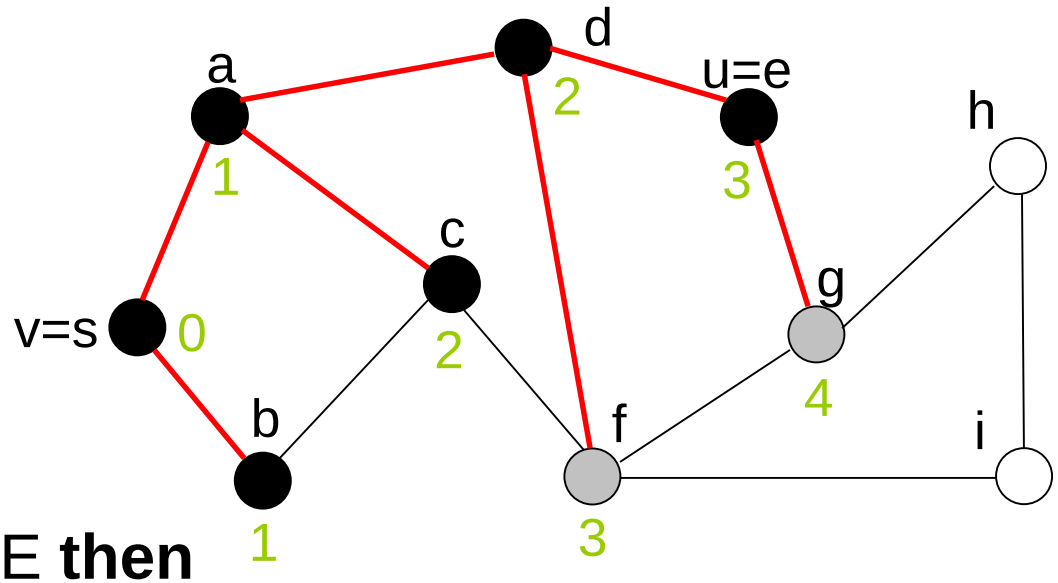
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: f, g

Breadth-First Search Example

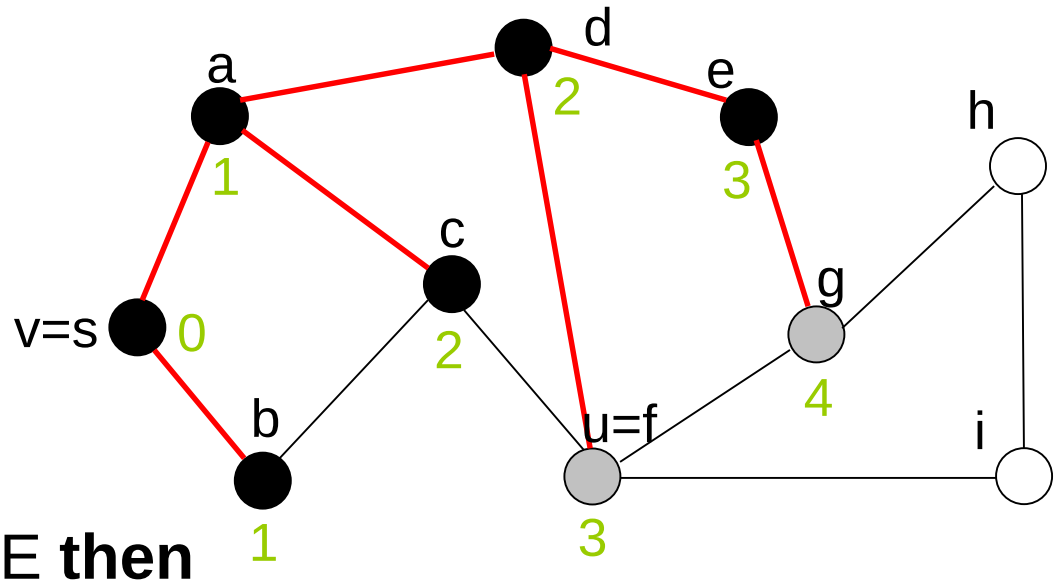
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: f, g

Breadth-First Search Example

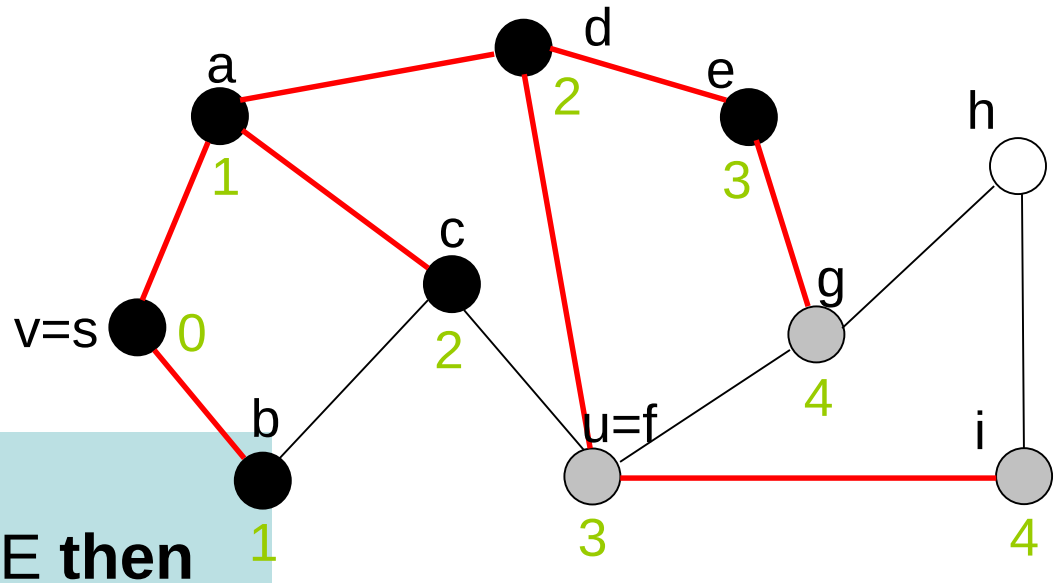
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: g

Breadth-First Search Example

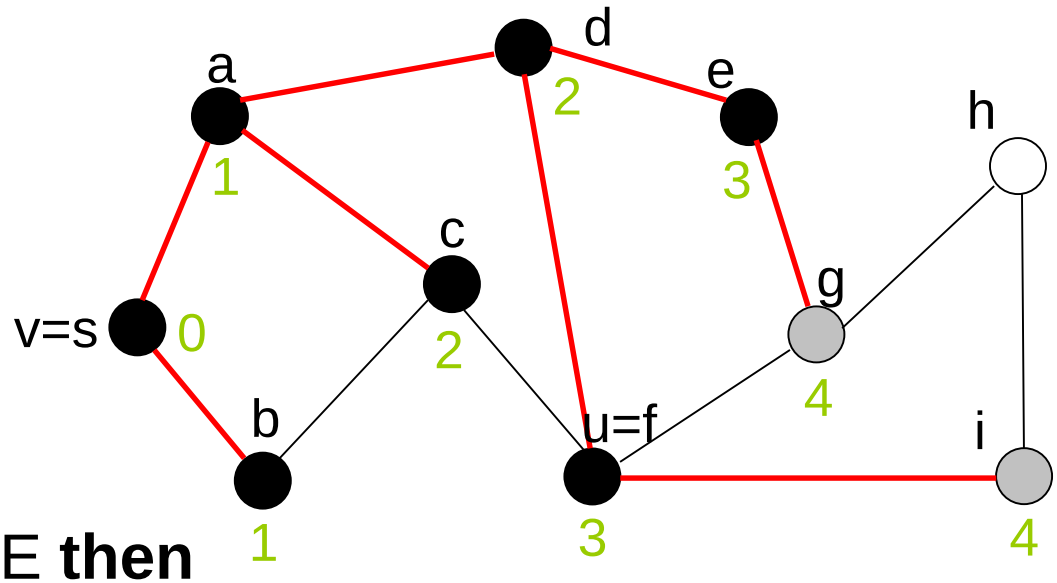
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: g, i

Breadth-First Search Example

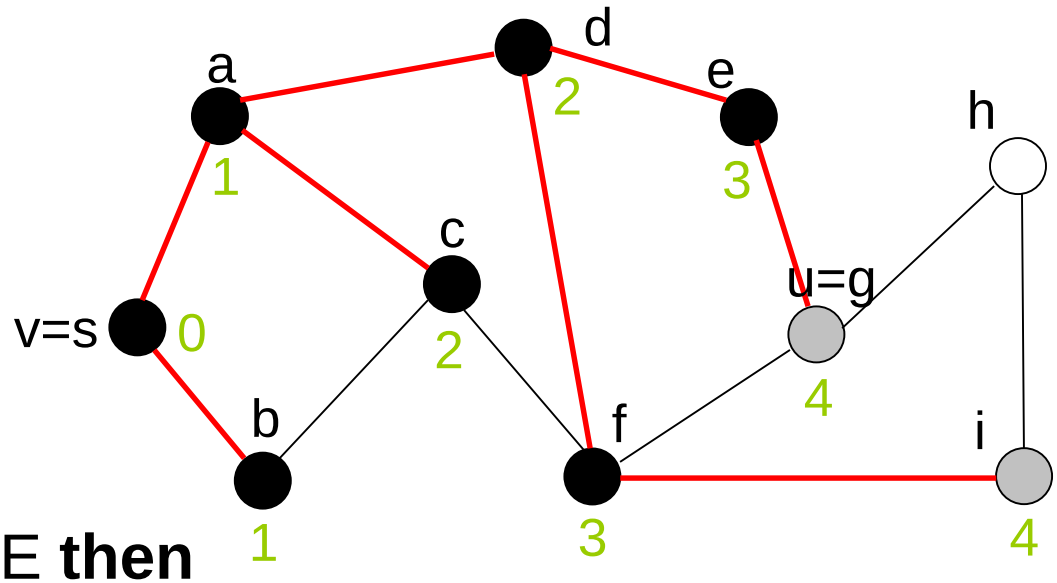
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: g, i

Breadth-First Search Example

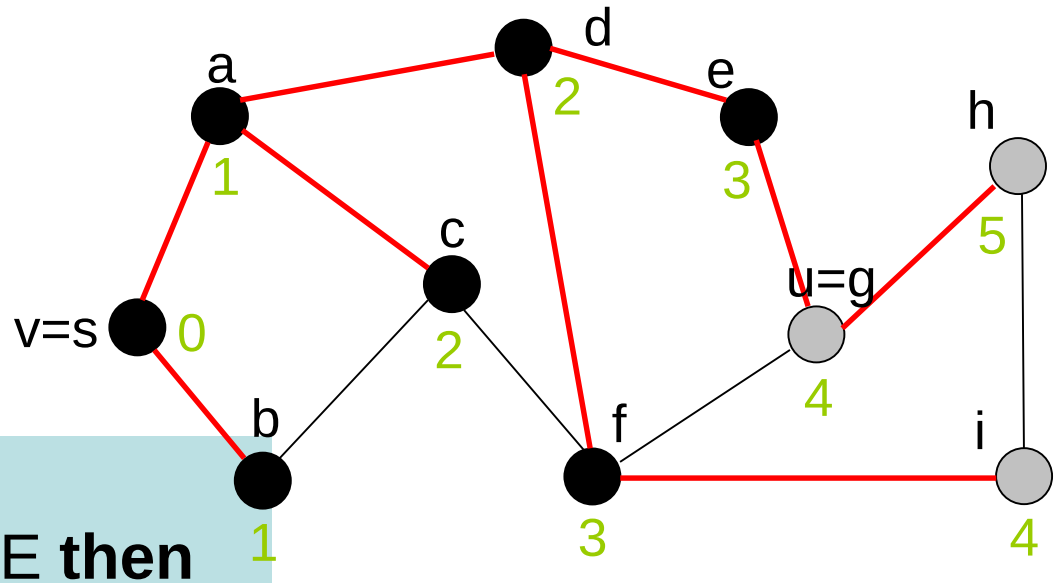
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: i

Breadth-First Search Example

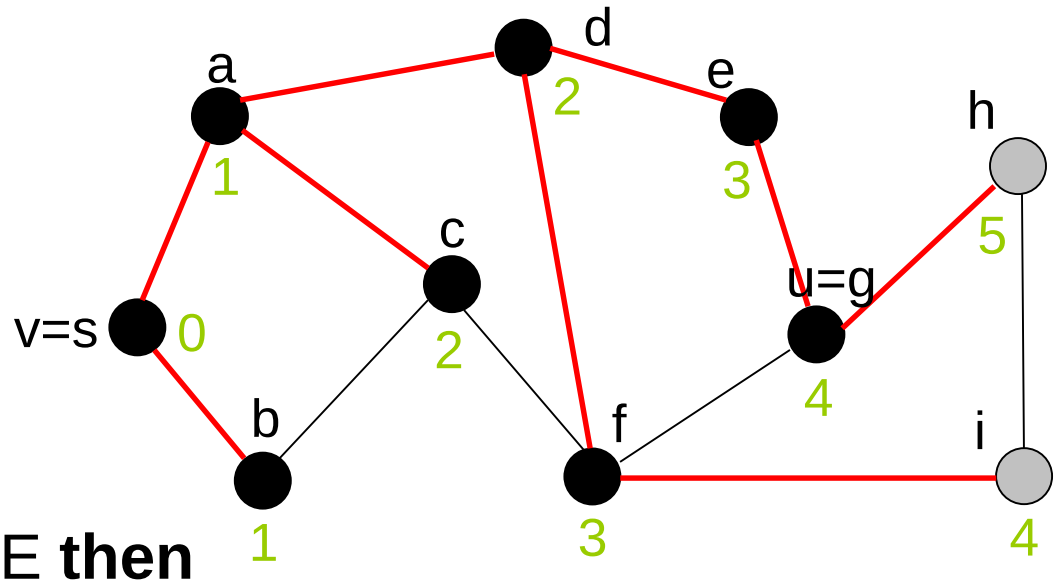
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: i, h

Breadth-First Search Example

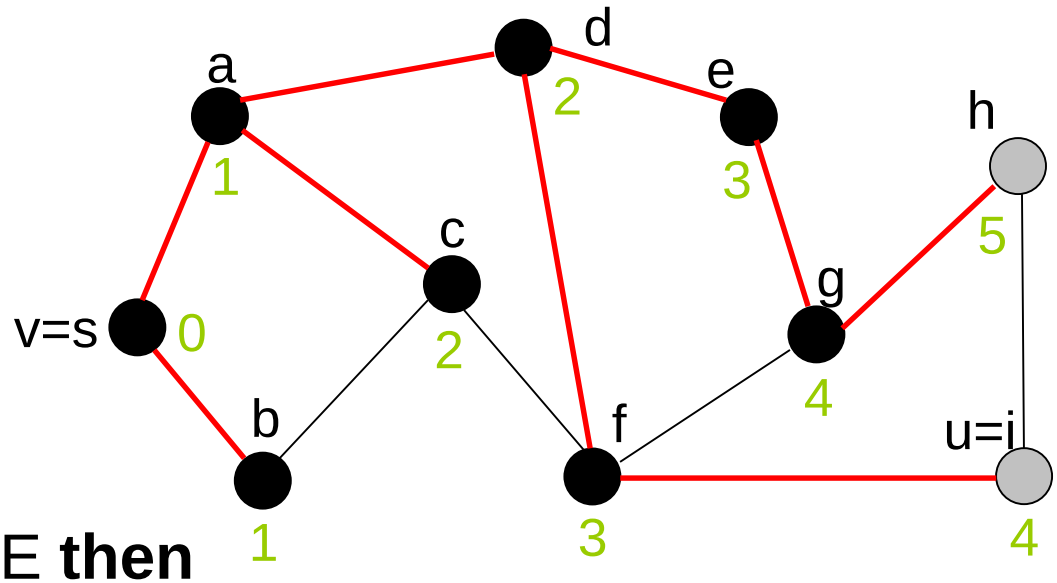
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: i, h

Breadth-First Search Example

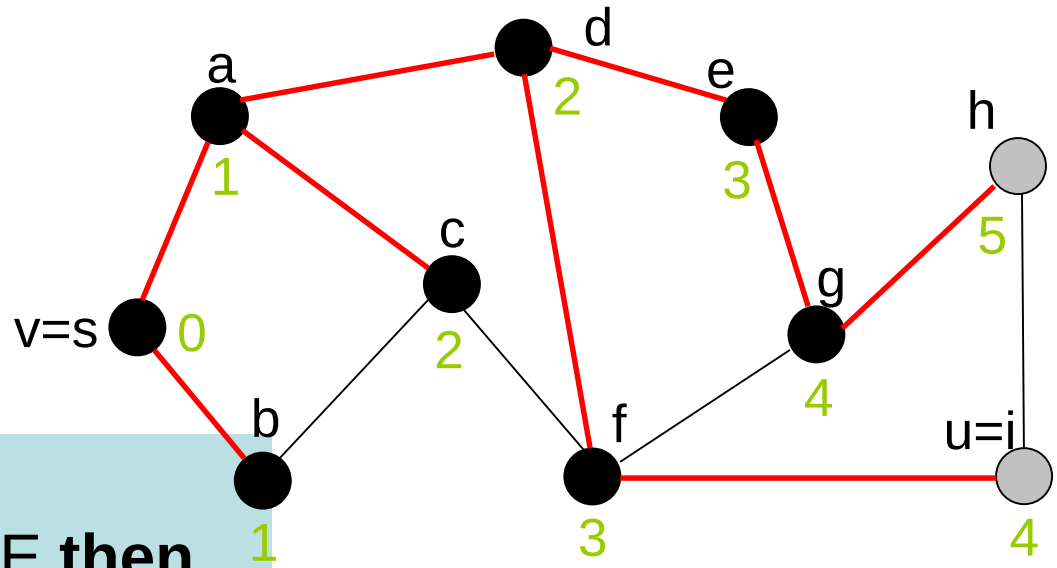
```
BFS(G,s)
1.  „initialize BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: h

Breadth-First Search Example

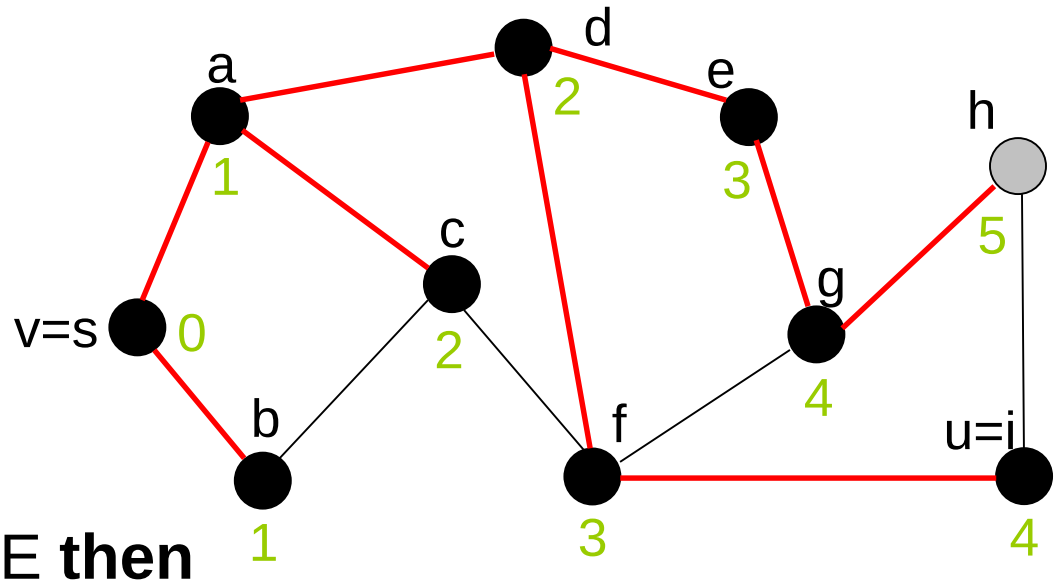
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: h

Breadth-First Search Example

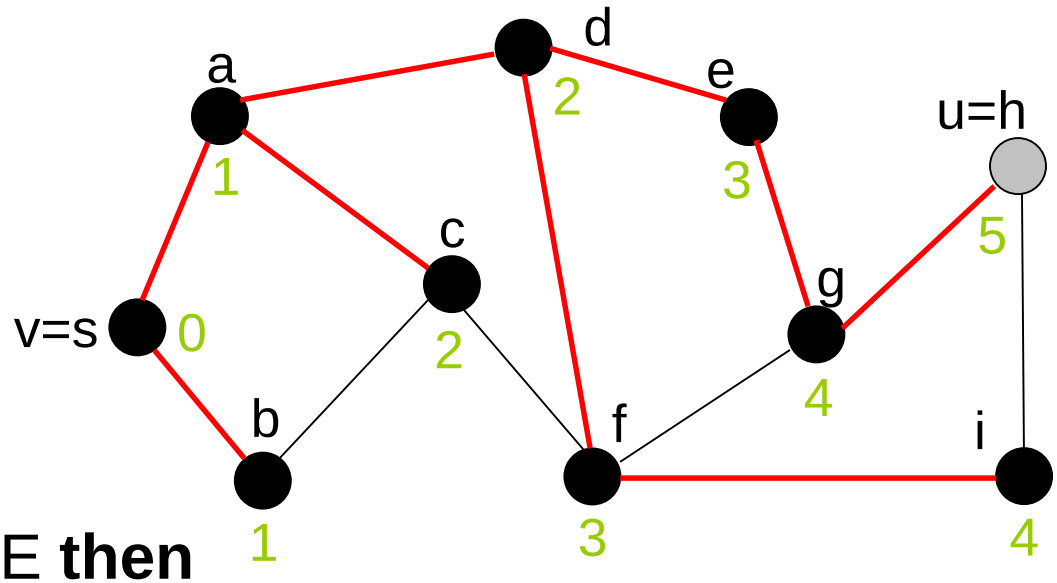
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: h

Breadth-First Search Example

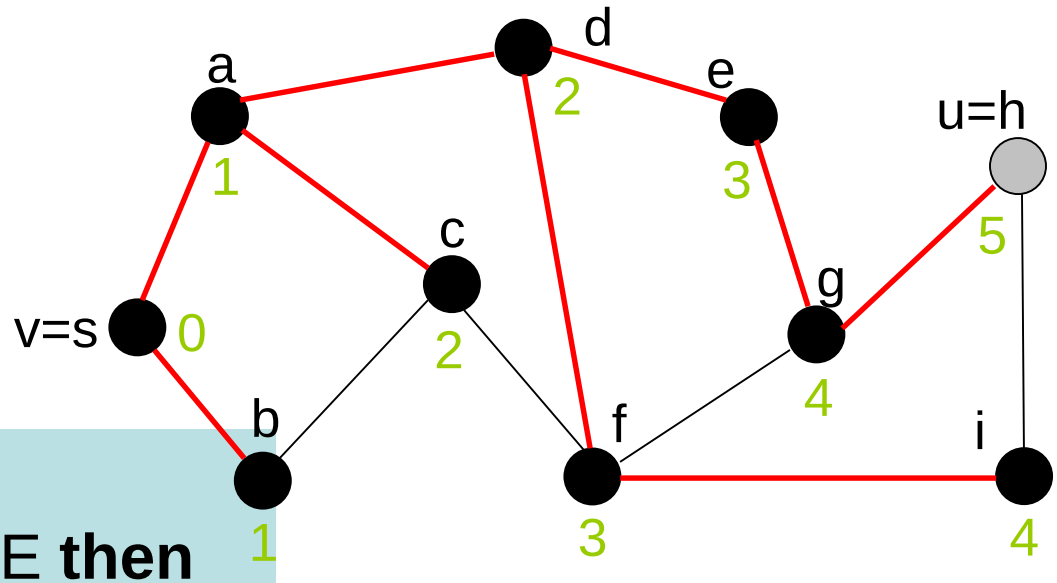
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

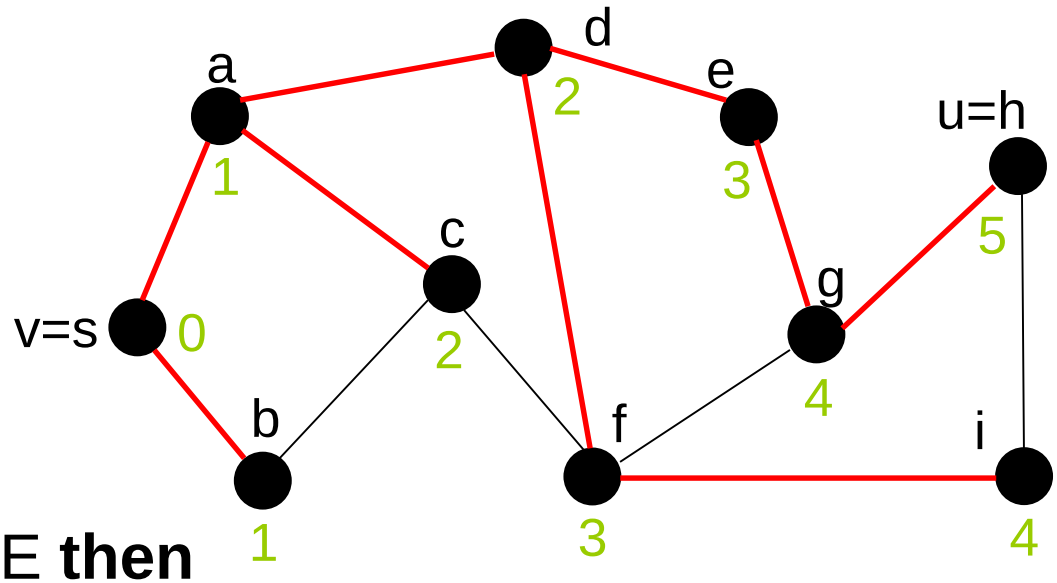
BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breadth-First Search Example

BFS(G,s)
1. „initialize BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

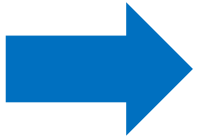
Breadth-First Search Runtime

Theorem 5.1: Given a graph G=(V,E) and a source s, algorithm BFS has a runtime of

$$O(|V| + |E|)$$

Depth-First Search

```
BFS(G,S)
1    for each node u∈V\{s} do
2        color[u]←WHITE
3        p[u] ← NIL
4    color[s] ← GRAY
5    π[s] ← NIL
6    Q ← { }
7    Enqueue(Q,s)
8    while Q ≠ { } do
9        u ← Dequeue(Q)
10       for each v∈Adj[u] do
11           if color[v]=WHITE then
12               color[v] ← GRAY
13               π[v] ← u
14               Enqueue(Q,v)
15       color[u] ← BLACK
```



```
DFS(G,S)
1    for each node u∈V\{s} do
2        color[u]←WHITE
3        p[u] ← NIL
4    color[s] ← GRAY
5    π[s] ← NIL
6    Q ← { }
7    Push(Q,s)
8    while Q ≠ { } do
9        u ← Pop(Q)
10       for each v∈Adj[u] do
11           if color[v]=WHITE then
12               color[v] ← GRAY
13               π[v] ← u
14               Push(Q,v)
15       color[u] ← BLACK
```

Depth-First Search

Instead of an iterative approach we will look at a recursive approach of realizing depth-first search.

Recursive approach:

- Initially: all nodes are white
- Discovered nodes become gray
- Fully processed nodes become black

Two time stamps: d[v] and f[v] (lie between 1 and 2|V|)

- d[v]: time at which v is discovered
- f[v]: time at which v has been processed

Depth-First Search

DFS(G)

1. **for each** vertex $u \in V$ **do** color[u] $\leftarrow$ weiß ; π[u] $\leftarrow$ nil
2. time $\leftarrow$ 0
3. **for each** vertex $u \in V$ **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

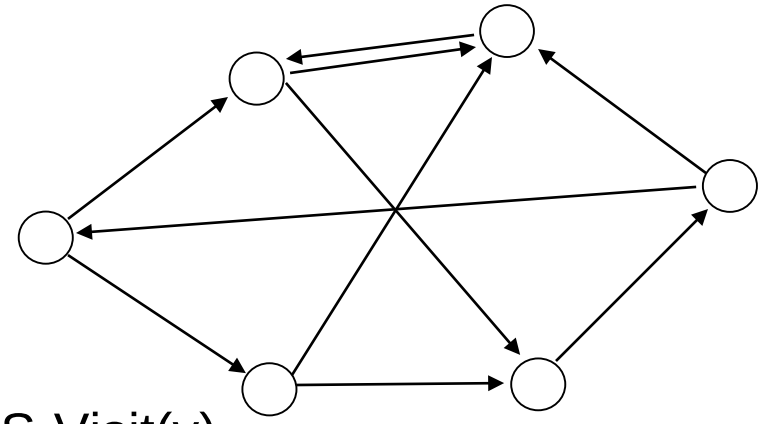
DFS-Visit(u)

1. color[u] $\leftarrow$ grau
2. time $\leftarrow$ time +1; d[u] $\leftarrow$ time
4. **for each** $v \in$ Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] $\leftarrow$ u ; DFS-Visit(v)
6. color[u] $\leftarrow$ schwarz
7. time $\leftarrow$ time+1; f[u] $\leftarrow$ time

Depth-First Search

DFS(G)

1. **for each** vertex u$\in$V **do** color[u] $\leftarrow$ weiß ; π[u] $\leftarrow$ nil
2. time $\leftarrow$ 0
3. **for each** vertex u$\in$V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)



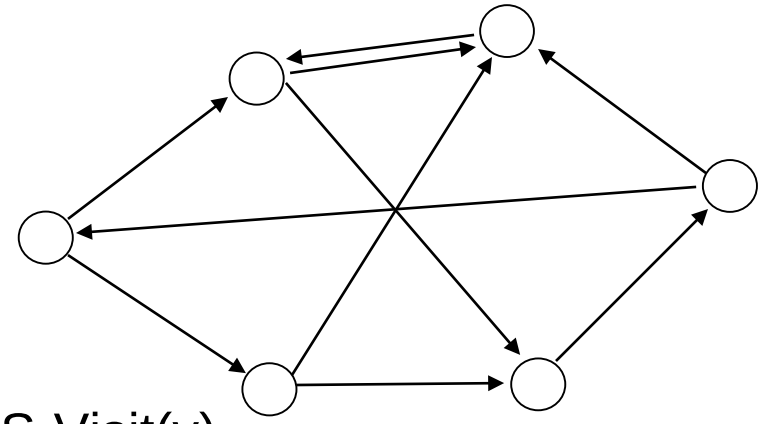
DFS-Visit(u)

1. color[u] $\leftarrow$ grau
2. time $\leftarrow$ time +1; d[u] $\leftarrow$ time
4. **for each** v$\in$Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] $\leftarrow$ u ; DFS-Visit(v)
6. color[u] $\leftarrow$ schwarz
7. time $\leftarrow$ time+1; f[u] $\leftarrow$ time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)



DFS-Visit(u)

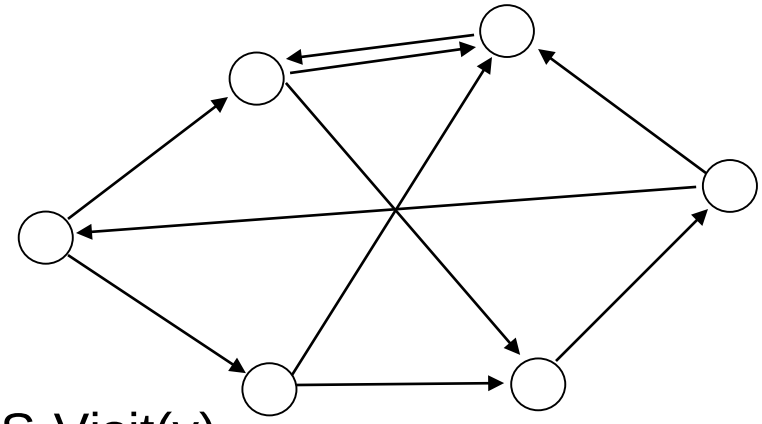
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

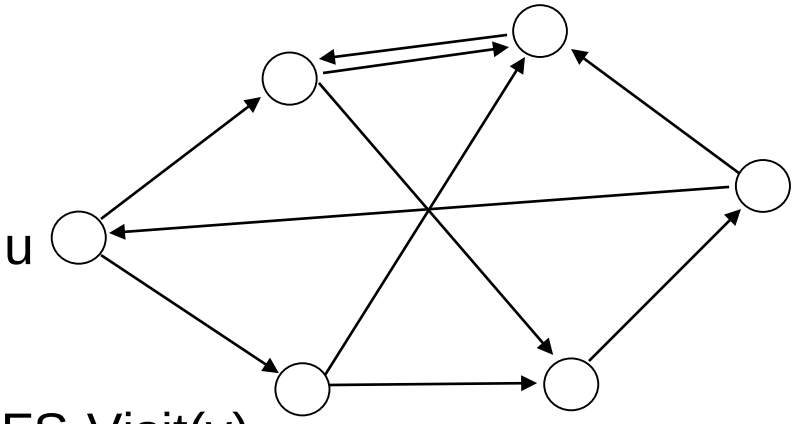
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time



Depth-First Search

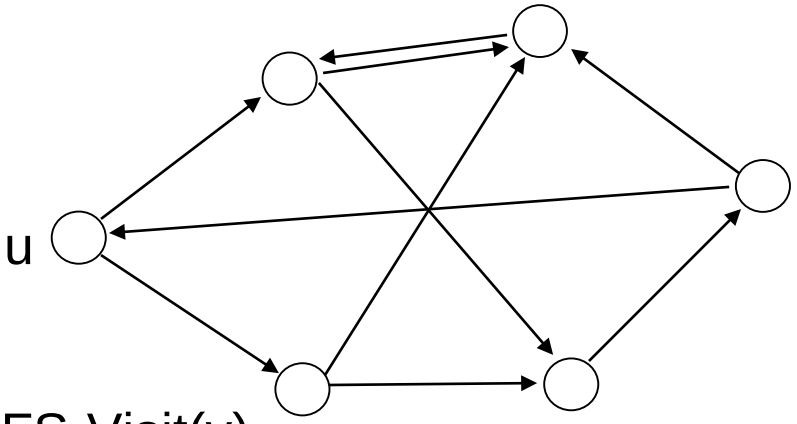
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

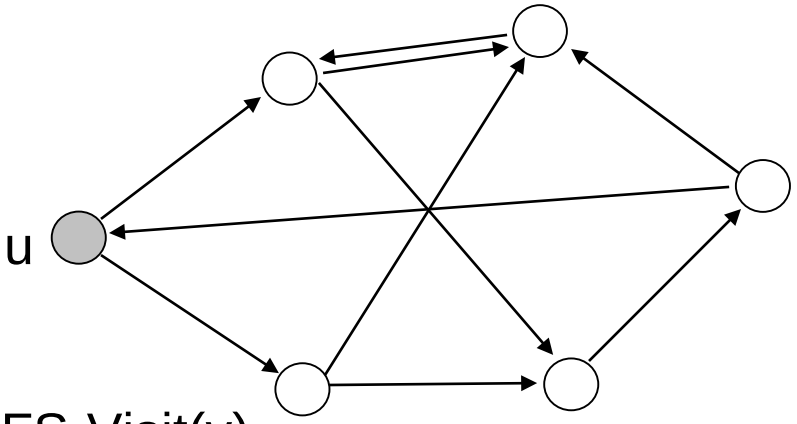


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0



DFS-Visit(u)

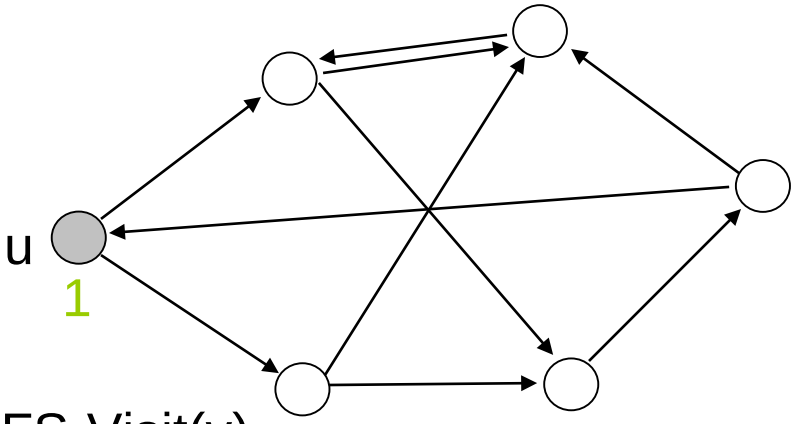
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

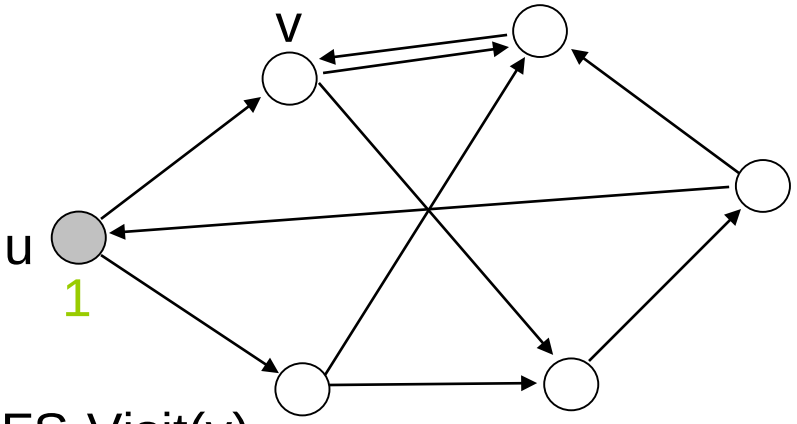
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

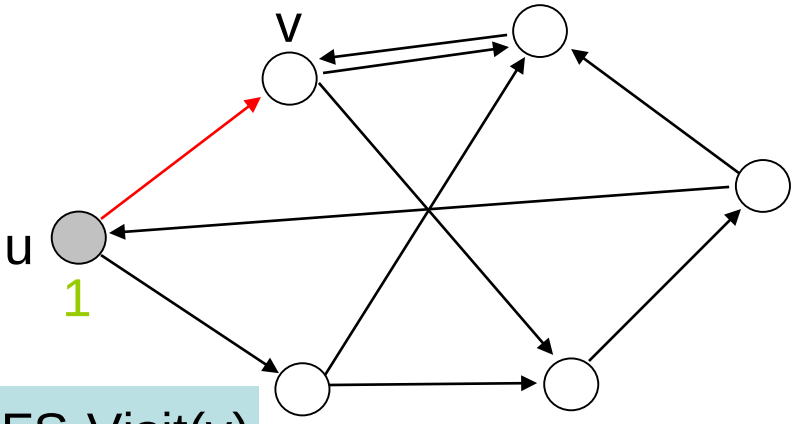
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

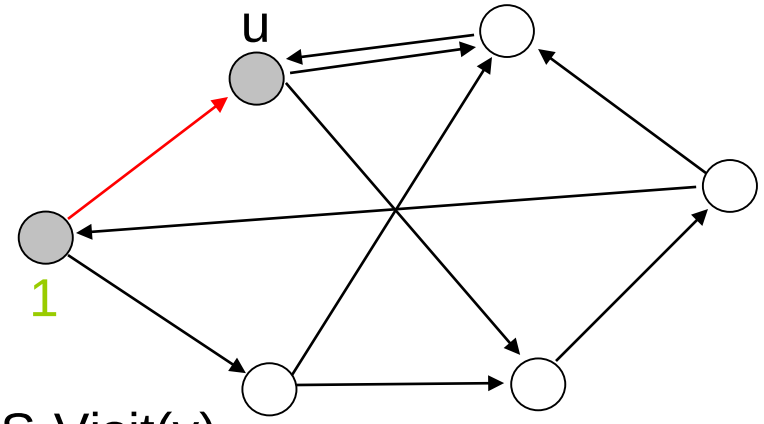
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

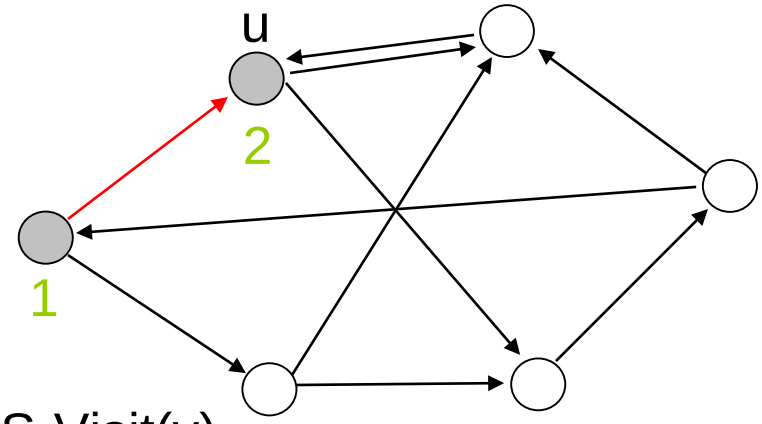
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

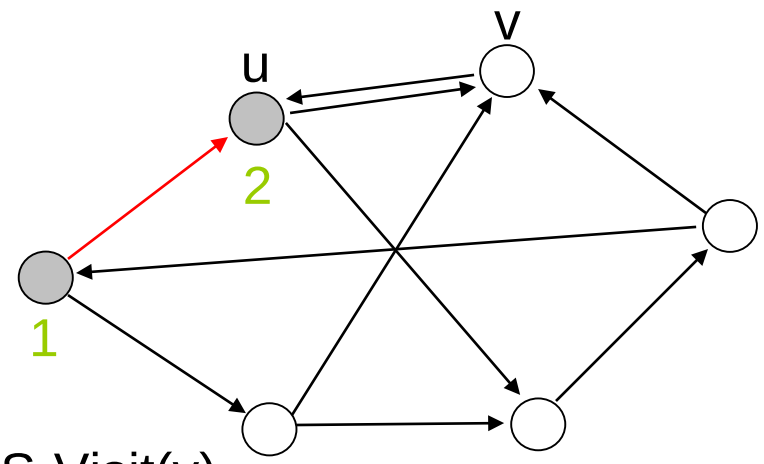
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

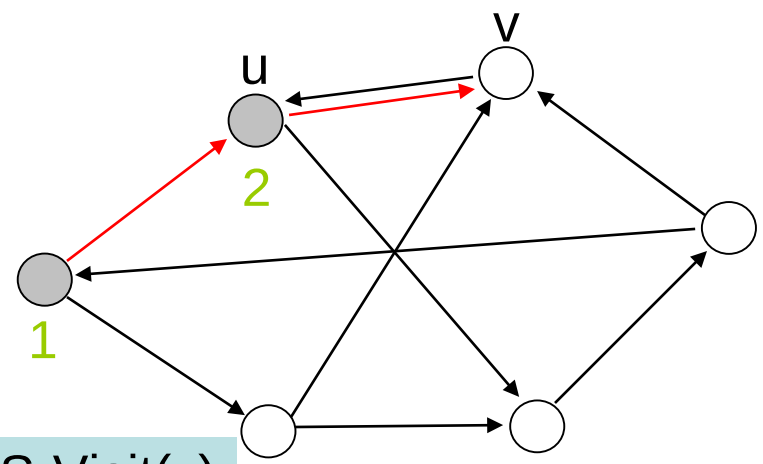
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

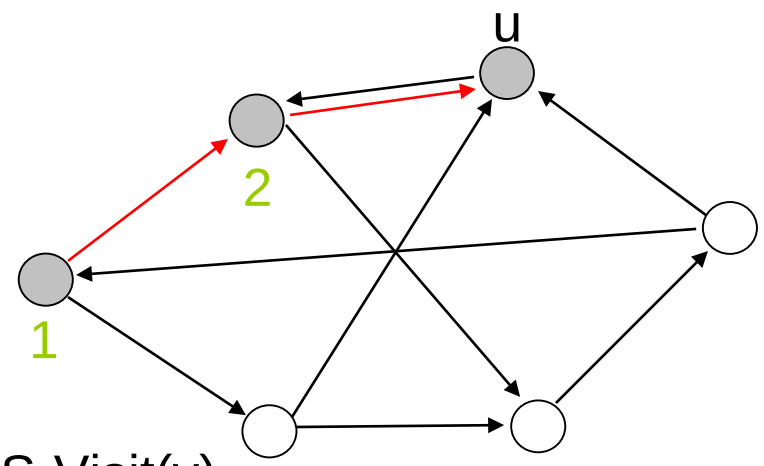
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

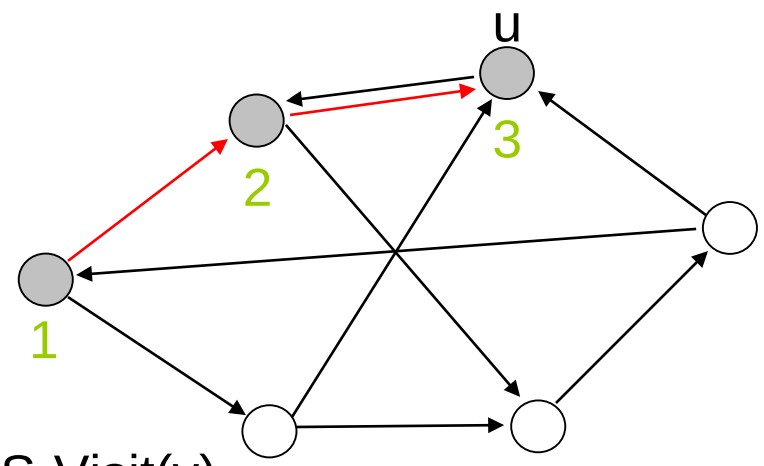
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

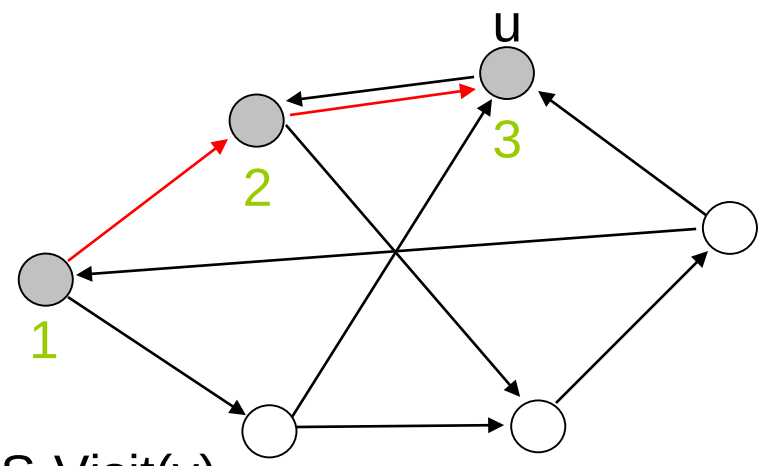
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

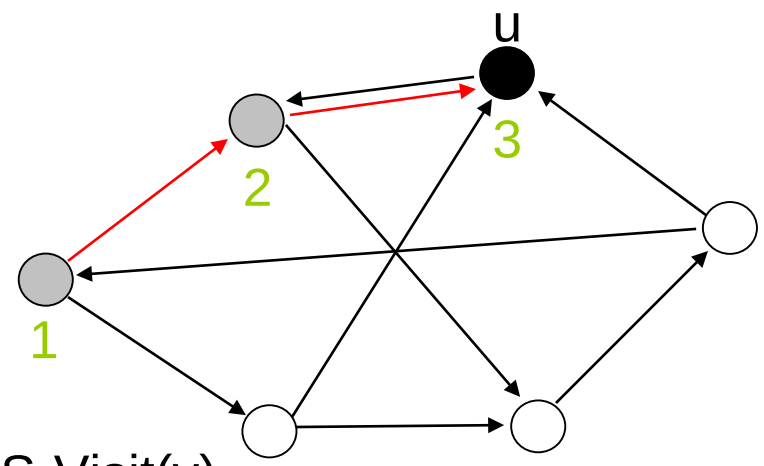
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

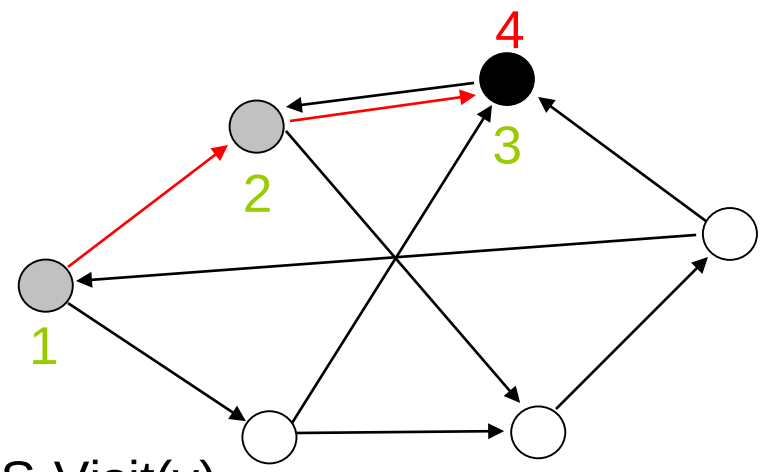
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=4



Depth-First Search

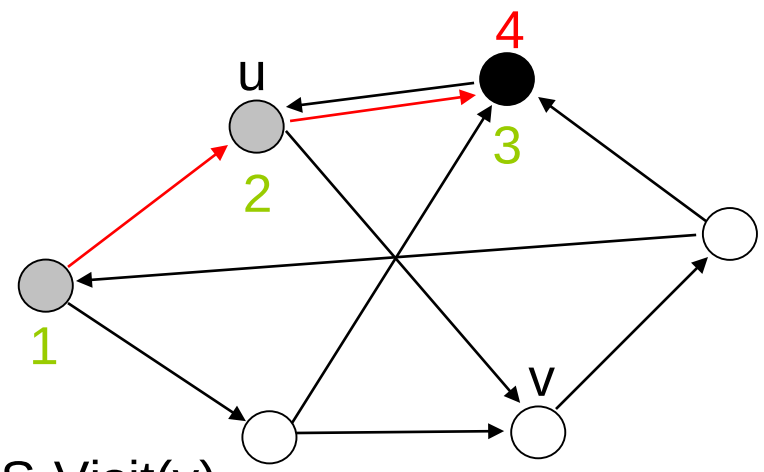
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=4



Depth-First Search

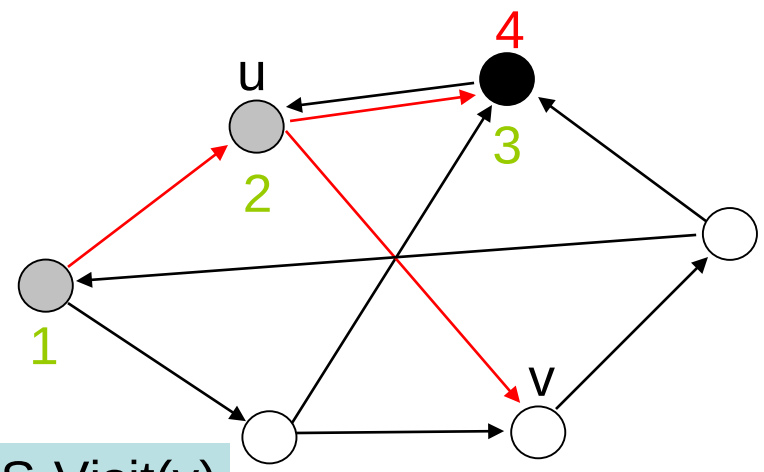
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=4



Depth-First Search

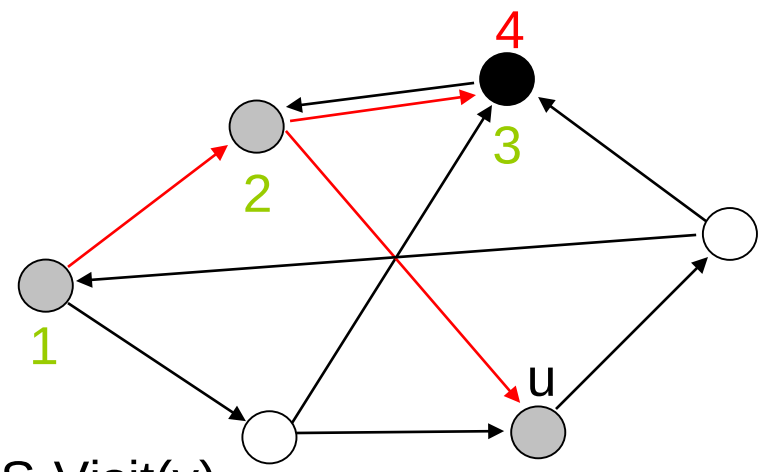
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=4

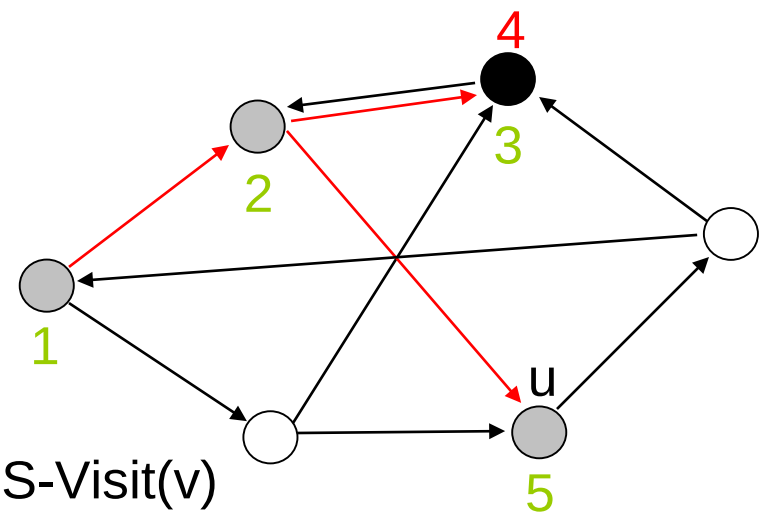


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=5



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

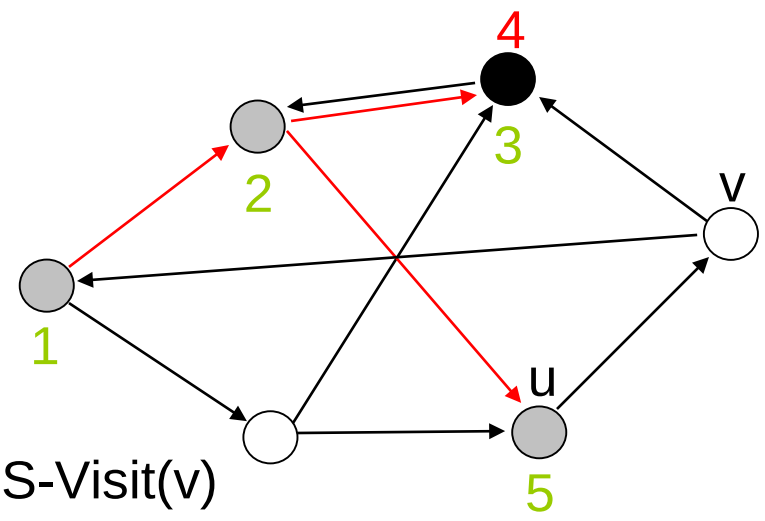
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=5



Depth-First Search

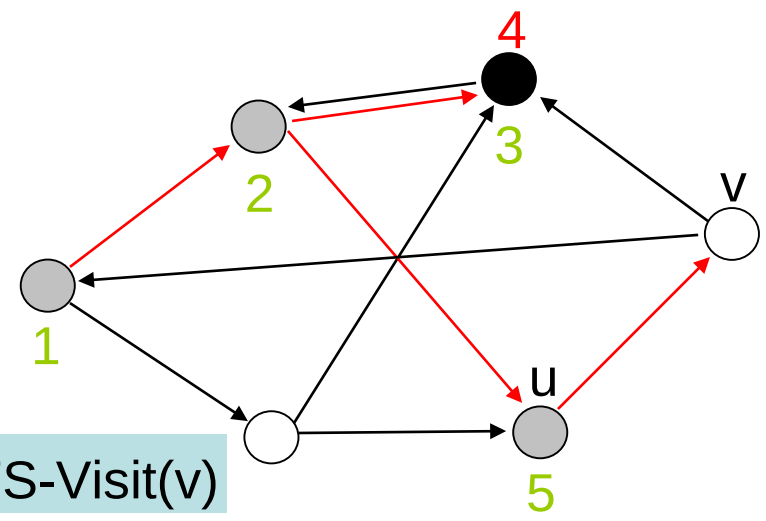
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=5

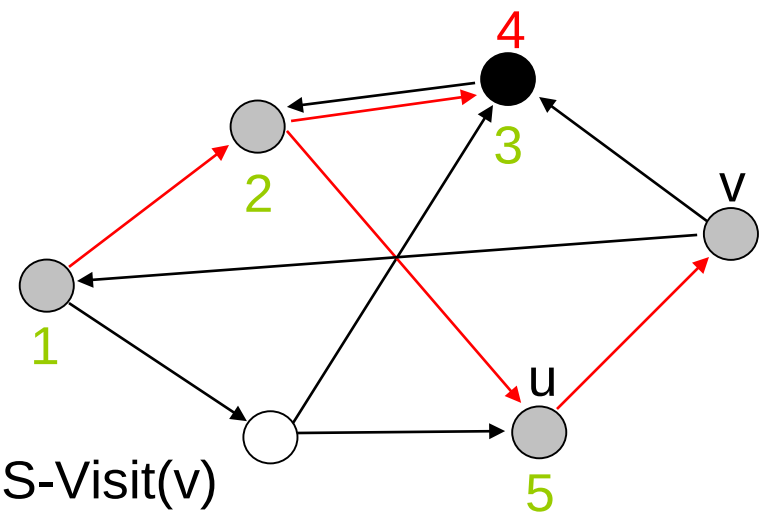


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=5



DFS-Visit(u)

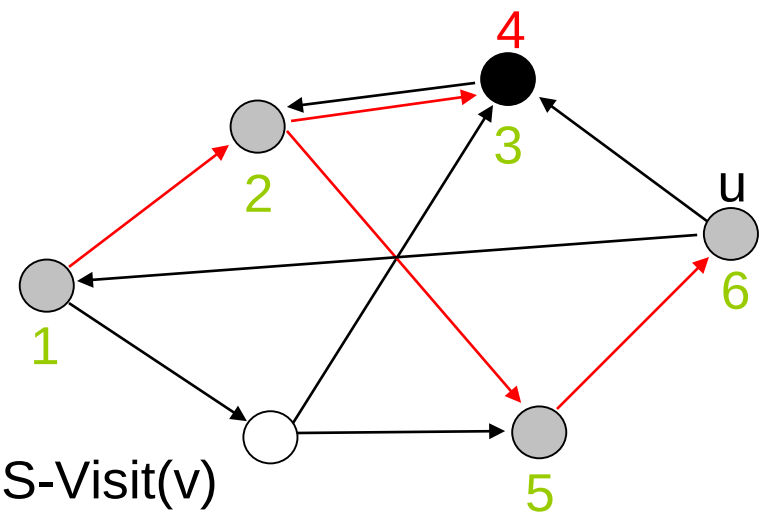
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

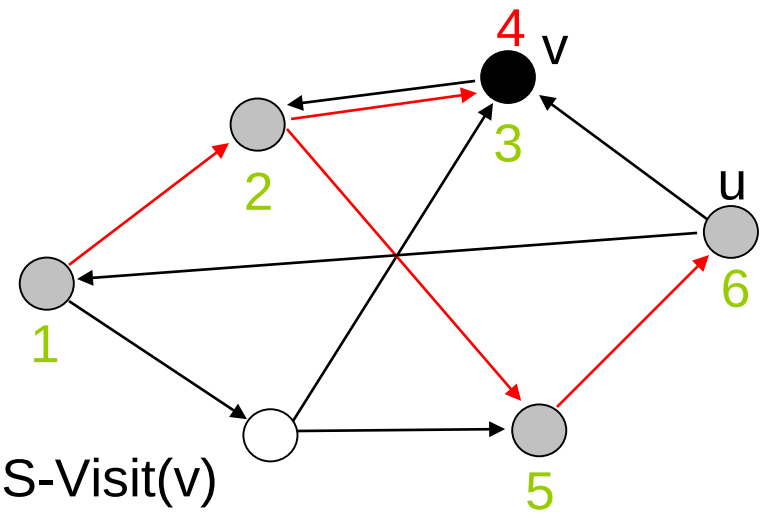
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=6



Depth-First Search

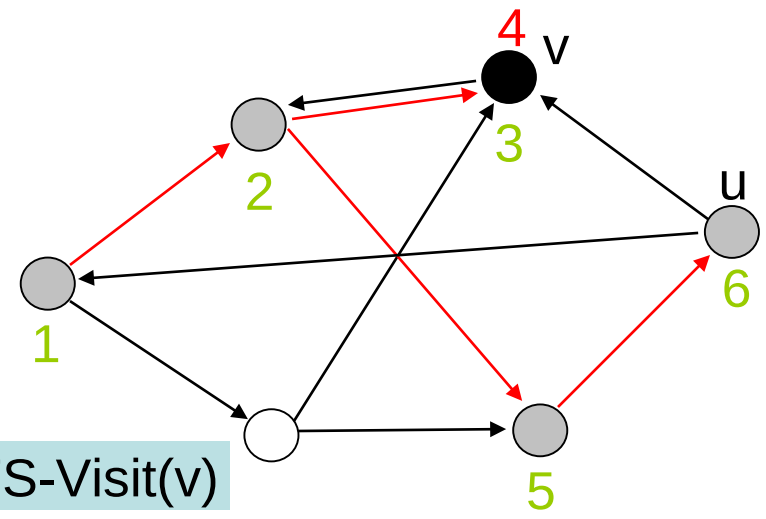
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=6

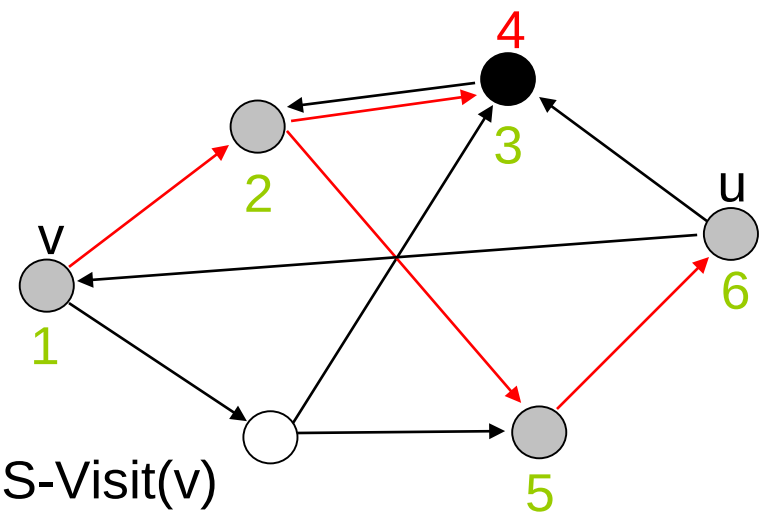


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

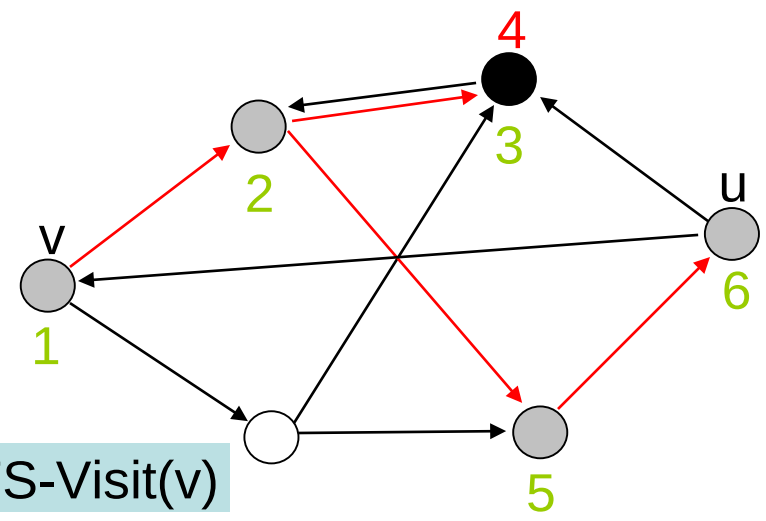
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

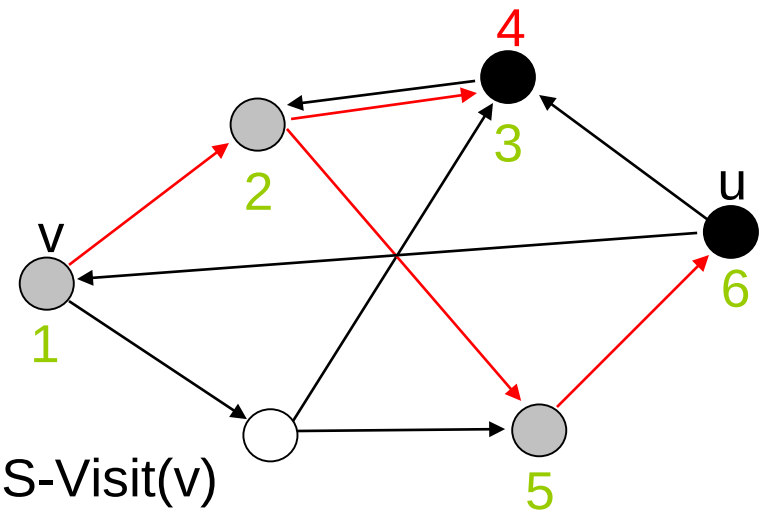
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

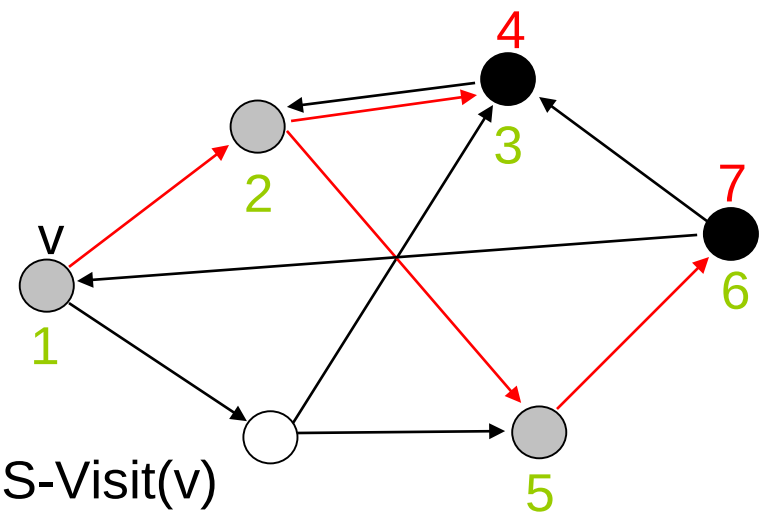
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=7

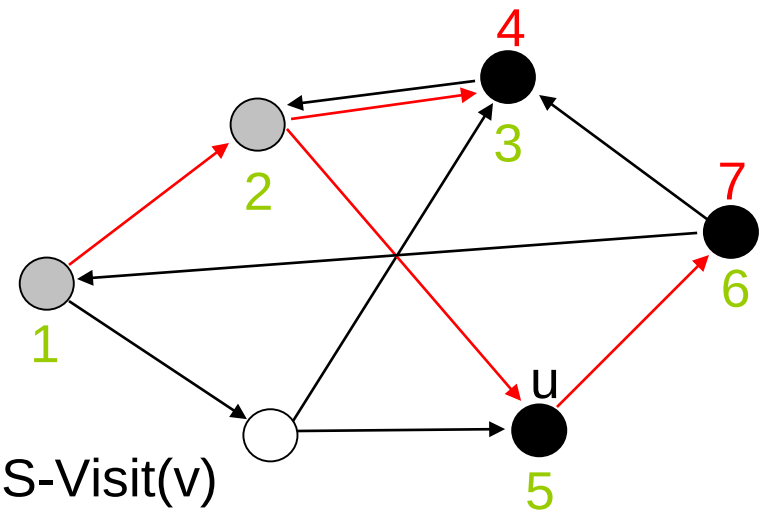


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=7



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

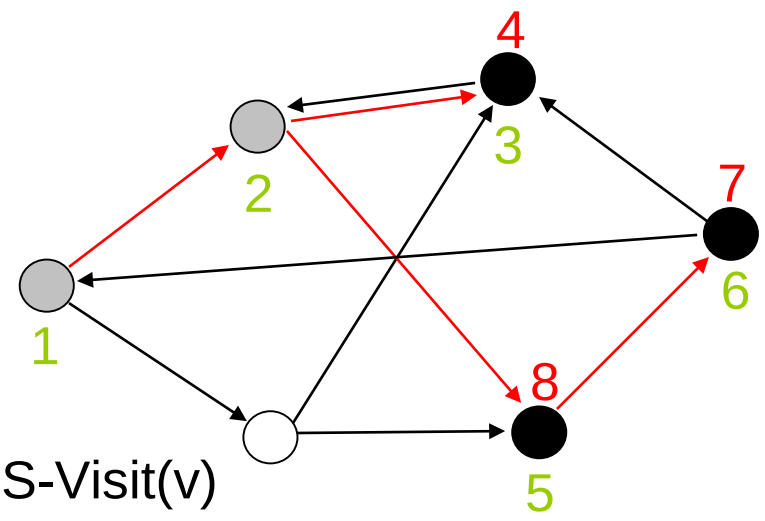
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=8

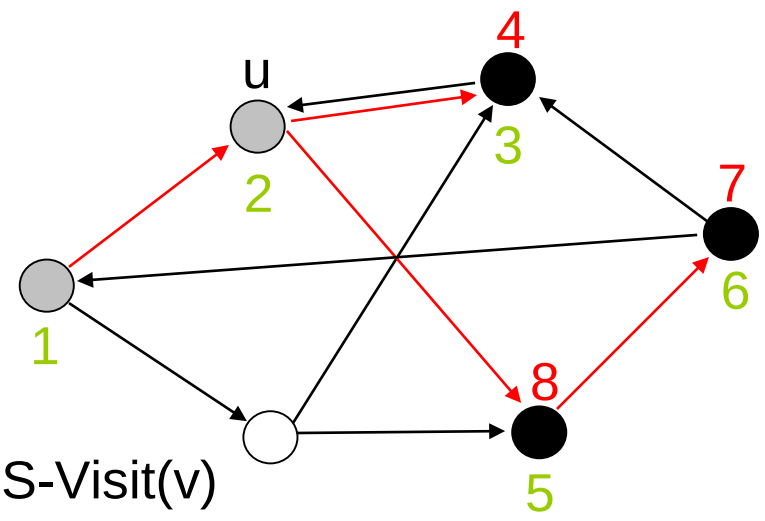


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=8



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

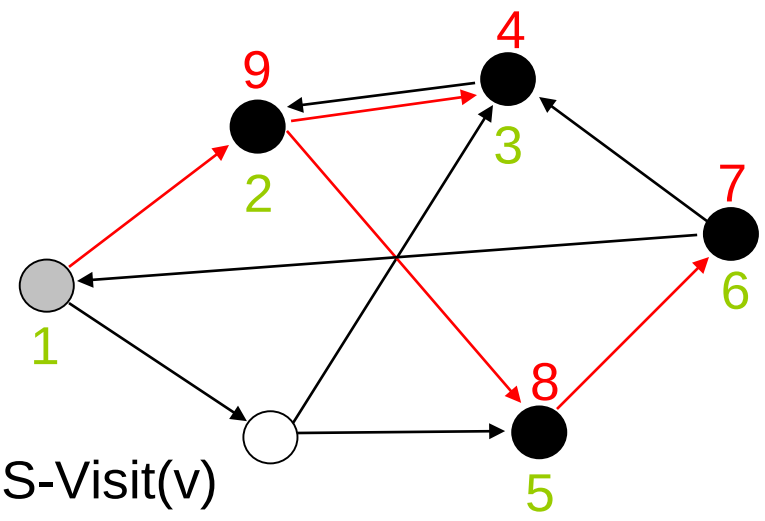
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=9



Depth-First Search

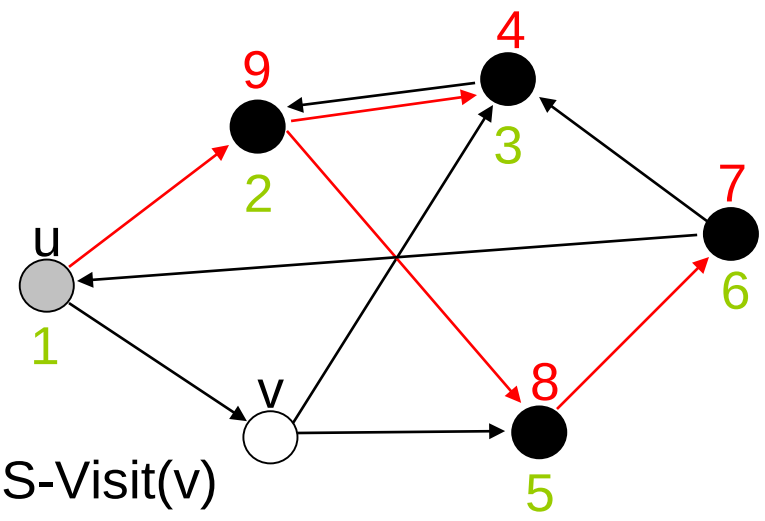
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=9

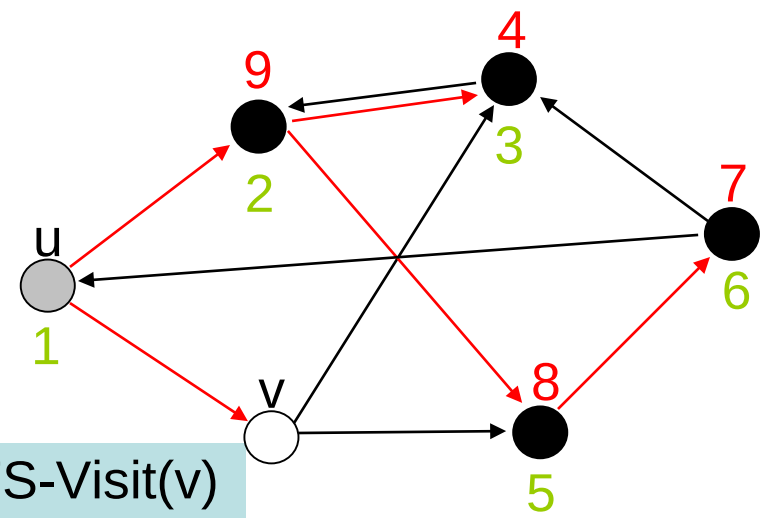


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

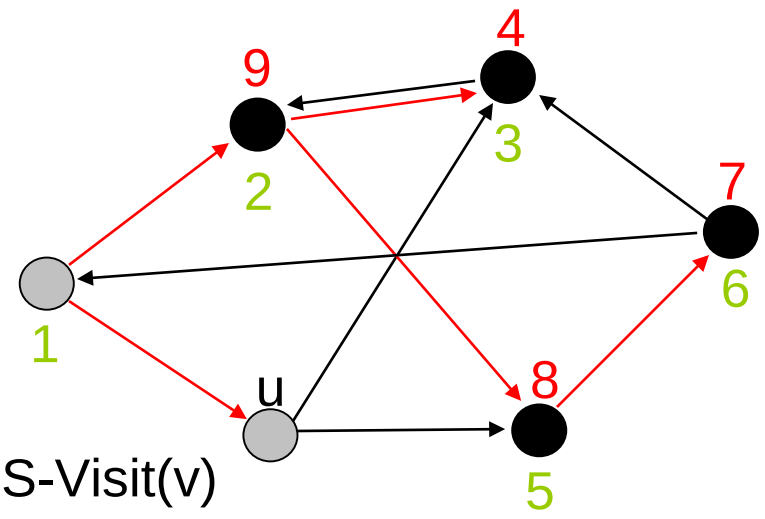
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

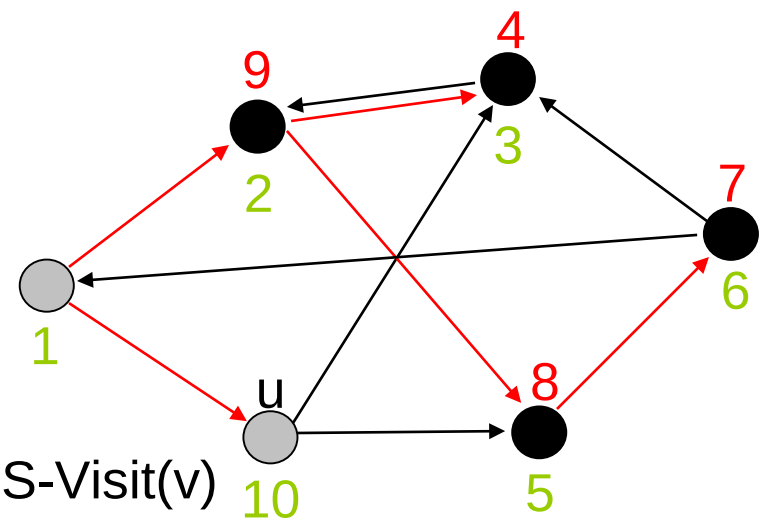
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

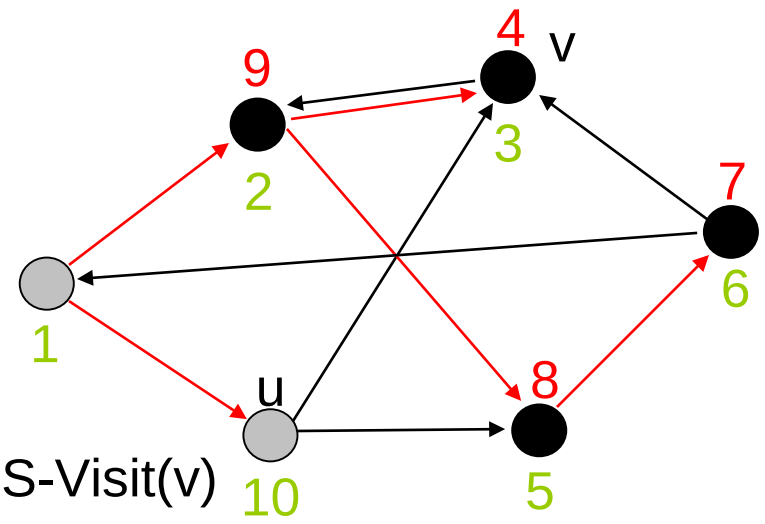
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=10

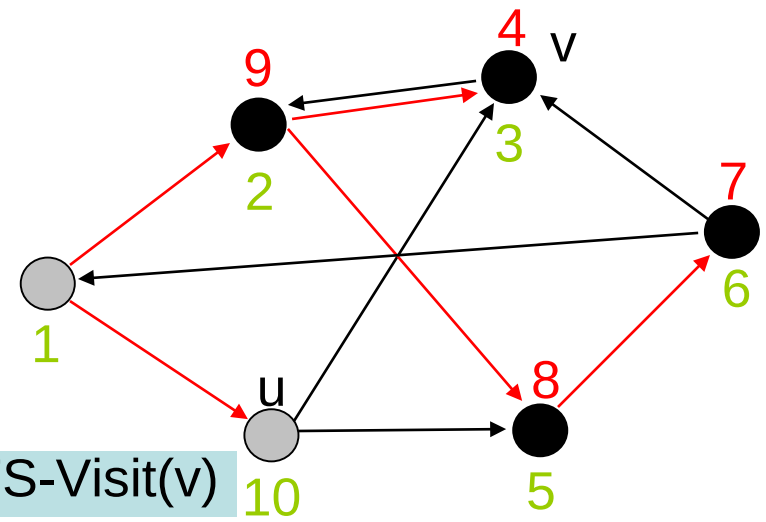


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

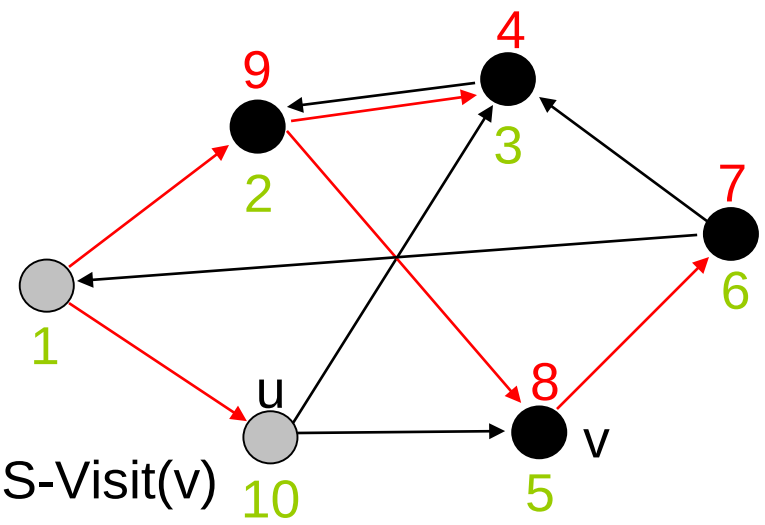
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

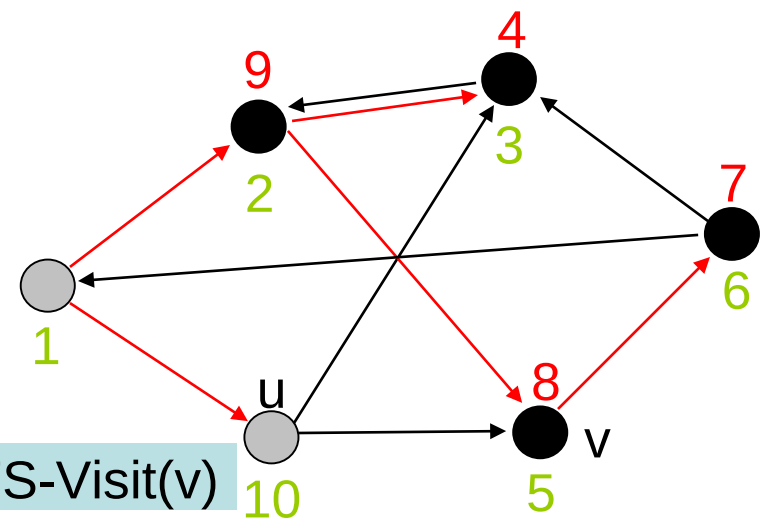
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

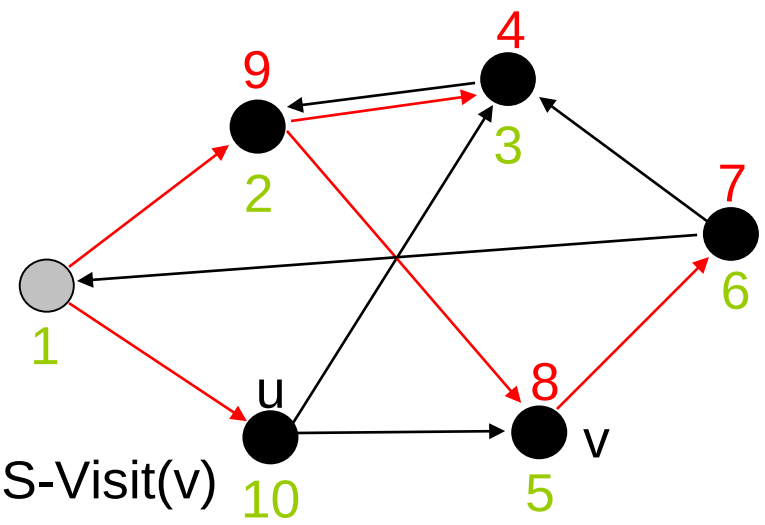
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

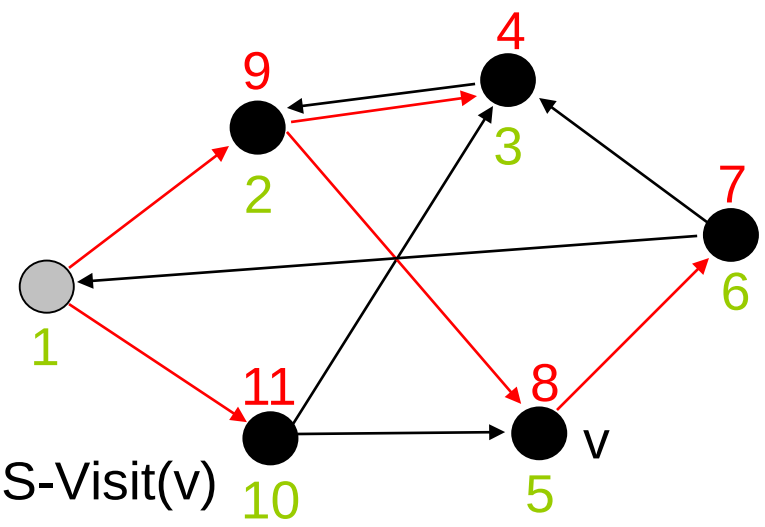
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=11

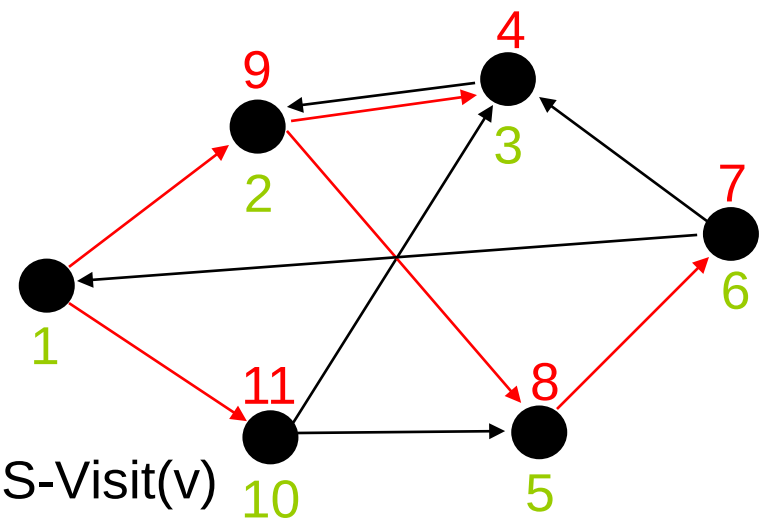


Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=11



DFS-Visit(u)

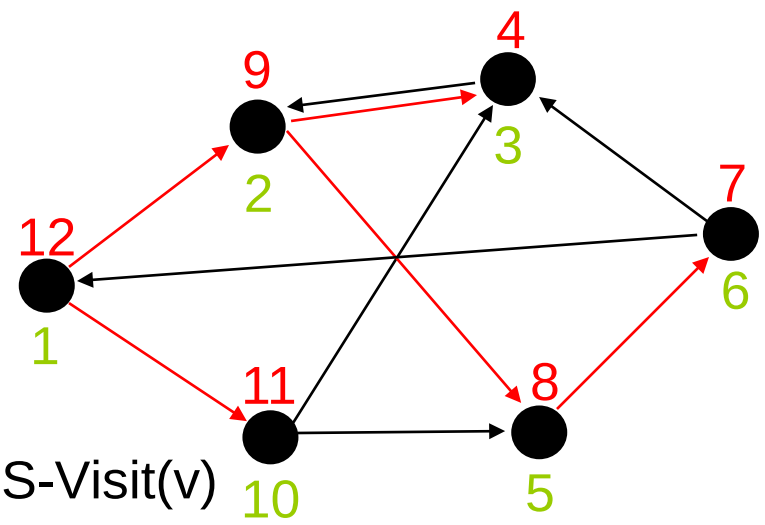
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=12



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Depth-First Search

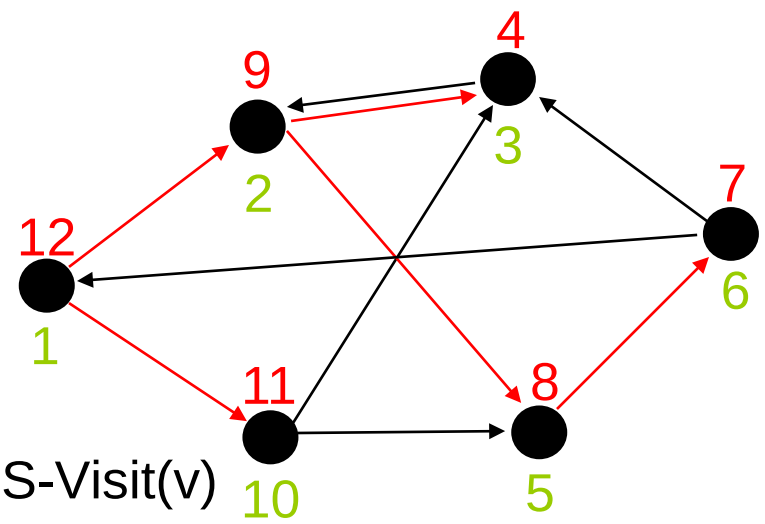
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=12



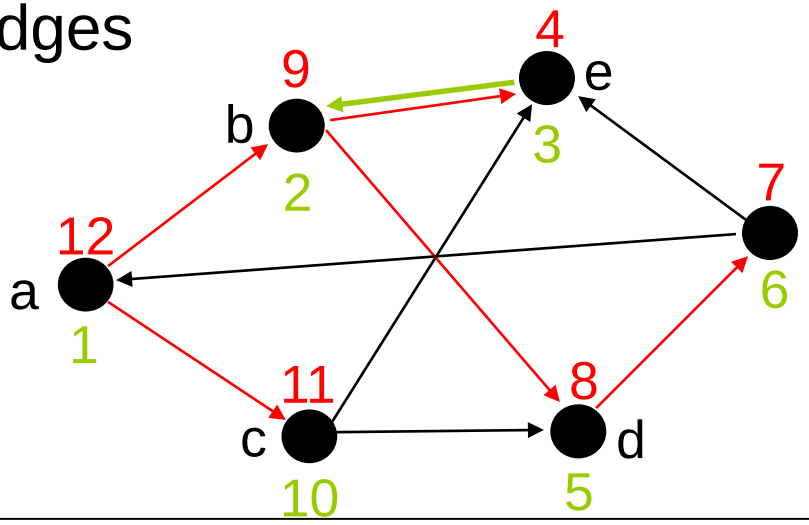
runtime:
O(|V|+|E|)

Depth-First Search Runtime

Theorem 5.2: Given a graph G=(V,E), algorithm DFS has a runtime of $O(|V| + |E|)$.

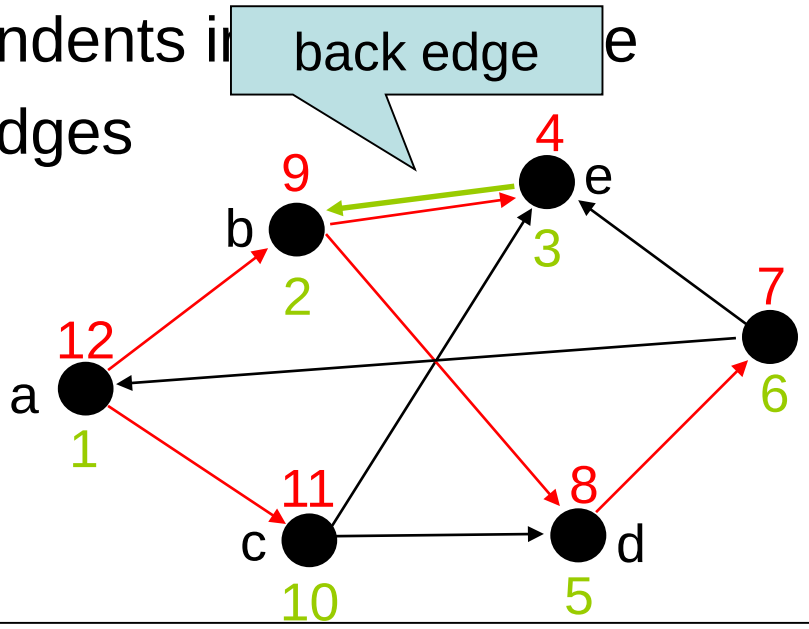
Classification of edges:

- Tree edges are edges of the DFS-forest in G
- Back edges are edges (u,v) that connect node u with one of its ancestors in the DFS-tree
- Forward edges are non-tree edges (u,v) that connect node u with one of its descendents in the DFS-tree
- Cross edges are the other edges



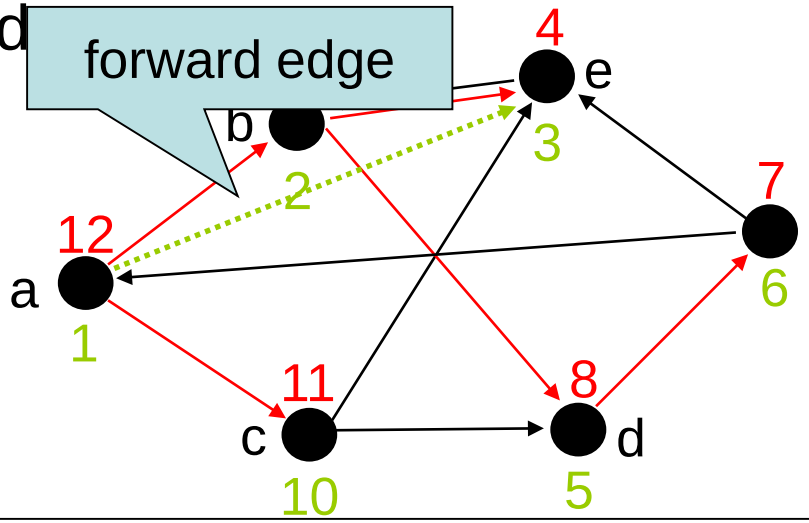
Classification of edges:

- Tree edges are edges of the DFS-forest in G
- Back edges are edges (u,v) that connect node u with one of its ancestors in the DFS-tree
- Forward edges are non-tree edges (u,v) that connect node u with one of its descendents i e
- Cross edges are the other edges



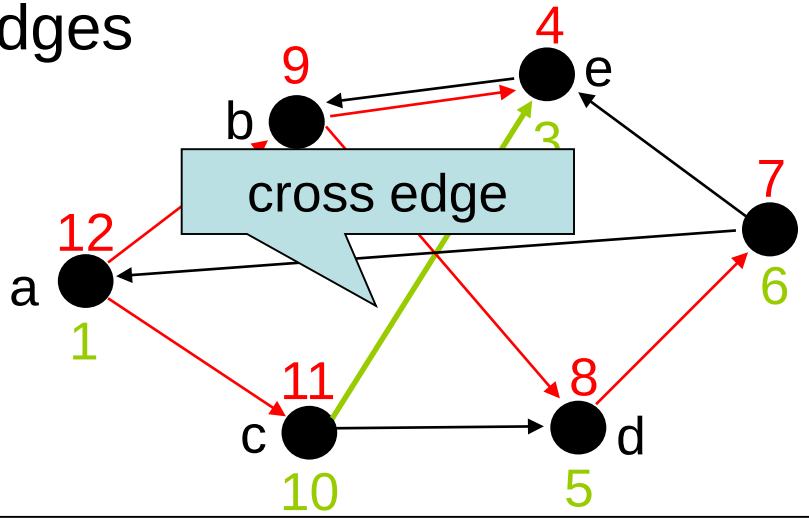
Classification of edges:

- Tree edges are edges of the DFS-forest in G
- Back edges are edges (u,v) that connect node u with one of its ancestors in the DFS-tree
- Forward edges are non-tree edges (u,v) that connect node u with one of its descendents in the DFS-tree
- Cross edges are the other ed



Classification of edges:

- Tree edges are edges of the DFS-forest in G
- Back edges are edges (u,v) that connect node u with one of its ancestors in the DFS-tree
- Forward edges are non-tree edges (u,v) that connect node u with one of its descendents in the DFS-tree
- Cross edges are the other edges



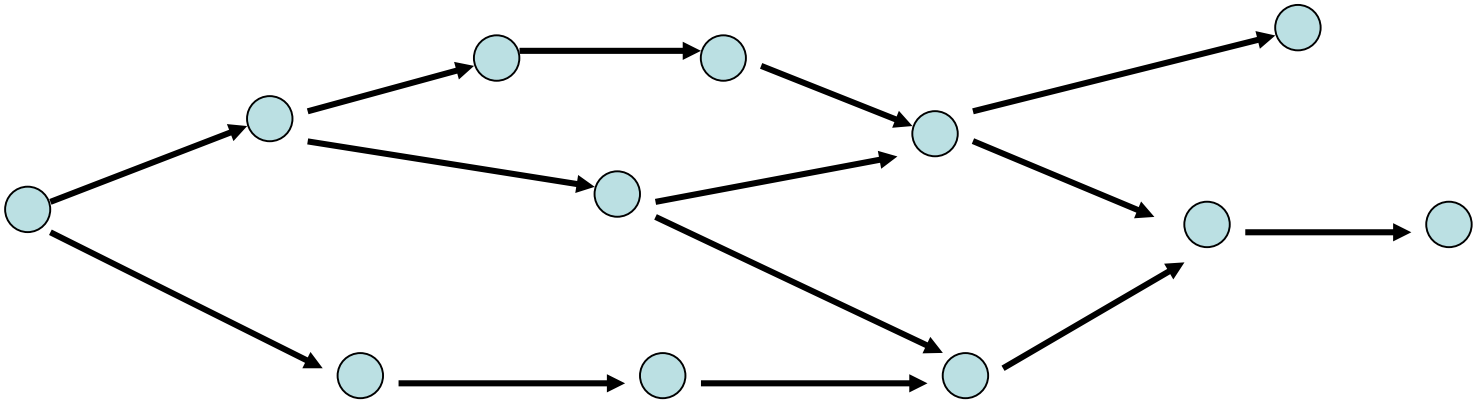
Depth-First Search

Classification of edges (v,w):

Edge type	d[v]< d[w]	f[v]>f[w]
tree & forward	yes	yes
back	no	no
cross	no	yes

Application:

- Recognition of acyclic directed graphs (DAG)



Property: no directed cycles

Depth-First Search

Theorem 3: The following statements are equivalent:

1. G is a DAG
2. The DFS-tree does not contain a back edge
3. $\forall(v,w)\in E: f[v]>f[w]$

Topological sort: Assign numbers $s(v)$ to the nodes v so that for all edges $(v,w)\in E$, $s(v)<s(w)$.

Theorem 3 implies: When sorting the nodes in decreasing order of their finishing times, we obtain a topological sort.

Minimum Spanning Trees

Definition 4:

1. A weighted undirected graph (G,w) is an undirected graph $G=(V,E)$ with a weight function $w:E\to\mathbb{R}$

2. For any subgraph $H=(U,F)$ (i.e., $U\subseteq V$, $F\subseteq E$) of G, the weight $w(H)$ of H is defined as

$$w(H) = \sum_{e\in F} w(e)$$

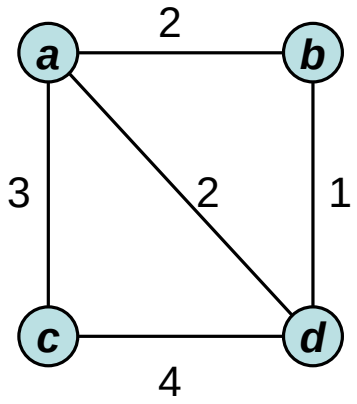
Minimum Spanning Trees

Definition 5:

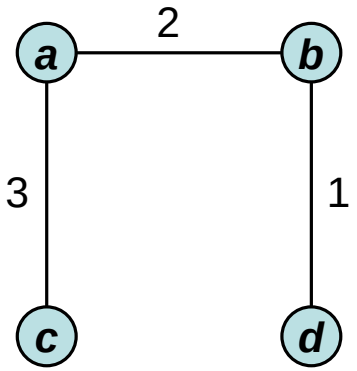
1. A subgraph H of an undirected graph G is called a spanning tree of G=(V,E) if H is a tree over all nodes of G.

2. A spanning tree S of a weighted undirected graph G is called a minimum spanning tree (MST) of G if the weight of S is minimal among all spanning trees of G.

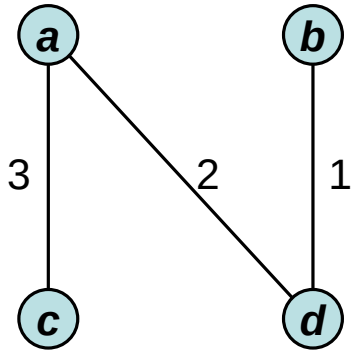
Minimum Spanning Trees



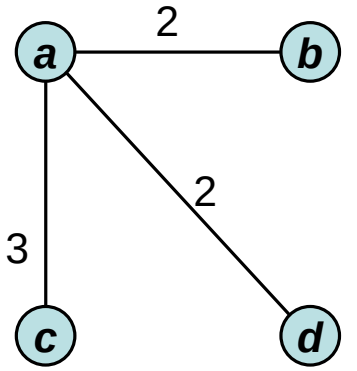
Graph G=(V,E)



Spanning tree of G (minimal)

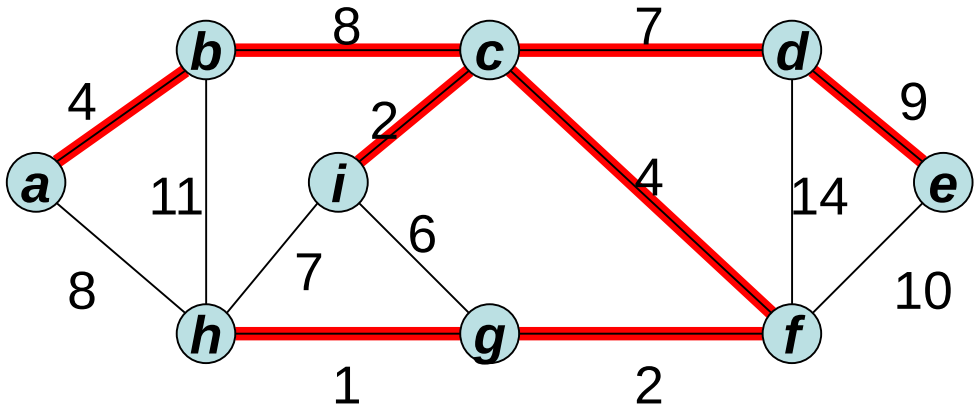


Spanning tree of G (minimal)



Spanning tree of G (not minimal)

Minimum Spanning Trees



Minimum Spanning Trees

Goal: Given a weighted undirected graph (G,w) with $G=(V,E)$, find a minimum spanning tree of (G,w).

Approach: We successively extend the edge set $A \subseteq E$ to a minimum spanning tree.

1. Initially, $A=\{\ \}$.
2. In each step, we extend A to $A \cup \{\ \{u,v\}\ \}$, where $\{u,v\}$ is an A-safe edge, until $|A|=|V|-1$.

Definition 6: $\{u,v\}$ is called A-safe if under the assumption that A can be extended to an MST, also $A \cup \{\ \{u,v\}\ \}$ can be extended to an MST.

Generic MST-Algorithm

Generic-MST(G,w)

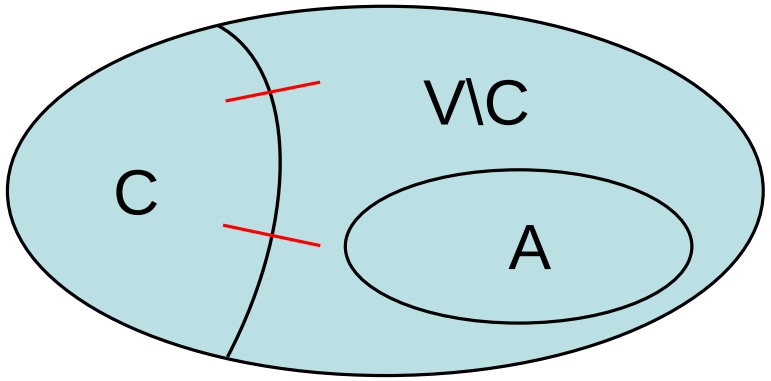
1. $A \leftarrow \{\ \}$
2. **while** A is not a spanning tree **do**
3. find an A-safe edge {u,v}
4. $A \leftarrow A \cup \{\ \{u,v\}\ \}$
5. **return** A

Correctness: results from the definition of A-safe edges

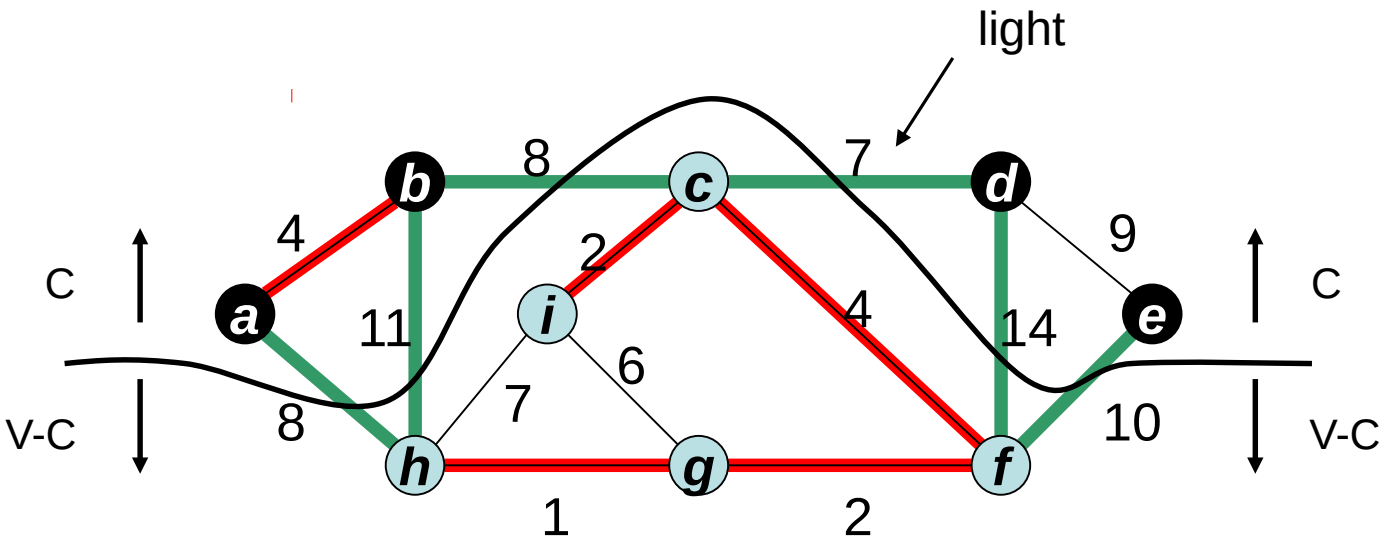
Cuts in Graphs

Definition 7:

1. A cut $(C,V\setminus C)$ in a graph $G=(V,E)$ is a partition of the node set V of the graph.
2. An edge of G is crossing a cut $(V,V\setminus C)$ if one node of it is in C and the other node in $V\setminus C$.
3. A cut $(C,V\setminus C)$ respects a subset $A\subseteq E$ if no edge in A crosses the cut.
4. An edge crossing the cut $(C,V\setminus C)$ is called light if it has a minimal weight among all edges crossing $(C,V\setminus C)$.



Cuts in Graphs



crossing edges

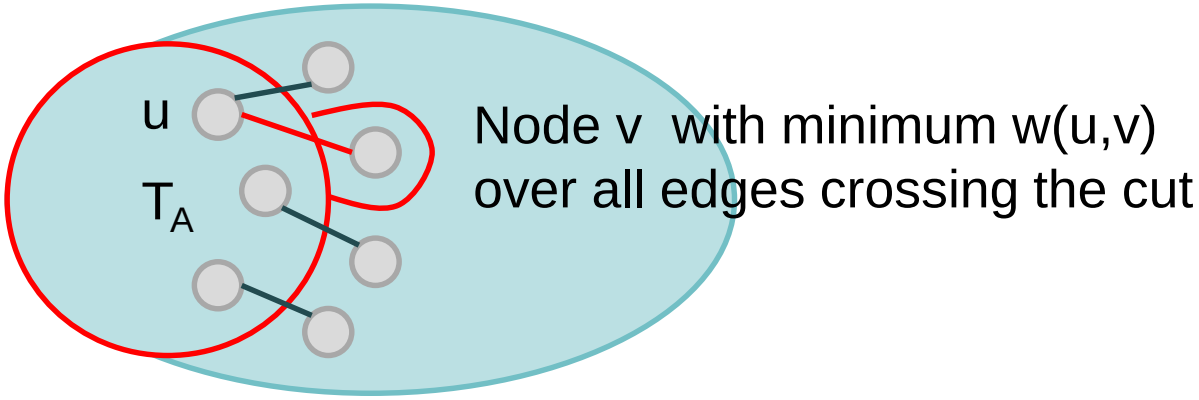
Characterization of Safe Edges

Theorem 8: Let (G,w) with $G=(V,E)$ be a weighted undirected graph, and let $A \subseteq E$ be contained in a minimum spanning tree of (G,w). Moreover, let $(C,V\setminus C)$ be a cut respecting A and $\{u,v\}$ be a light edge crossing $(C,V\setminus C)$. Then $\{u,v\}$ is an A-safe edge.

Corollary 9: Let (G,w) with $G=(V,E)$ be a weighted undirected graph, and let $A \subseteq E$ be contained in a minimum spanning tree of (G,w). If $\{u,v\}$ is an edge of mimimal weight that connected a connected component C of $G_A=(V,A)$ with the rest of the graph G_A, then $\{u,v\}$ is an A-safe edge.

Prim‘s Algorithm

- At any time of the algorithm, $G_A=(V,A)$ consists of a tree T_A and a set of isolated nodes I_A.
- A edge of minimal weight that connects I_A with T_A is added to A.



Prim‘s Algorithm

- At any time of the algorithm, $G_A=(V,A)$ consists of a tree T_A and a set of isolated nodes I_A.
- A edge of minimal weight that connects I_A with T_A is added to A.

- The nodes in I_A are organized in a Min-Heap. The key $d[v]$ of a node $v \in I_A$ is given by the minimal weight of an edge that connects v with T_A.

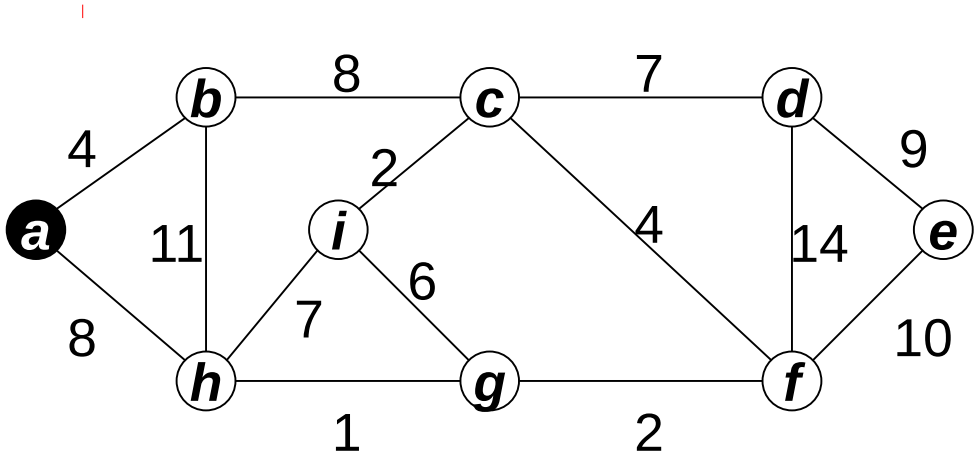
Prim‘s Algorithm

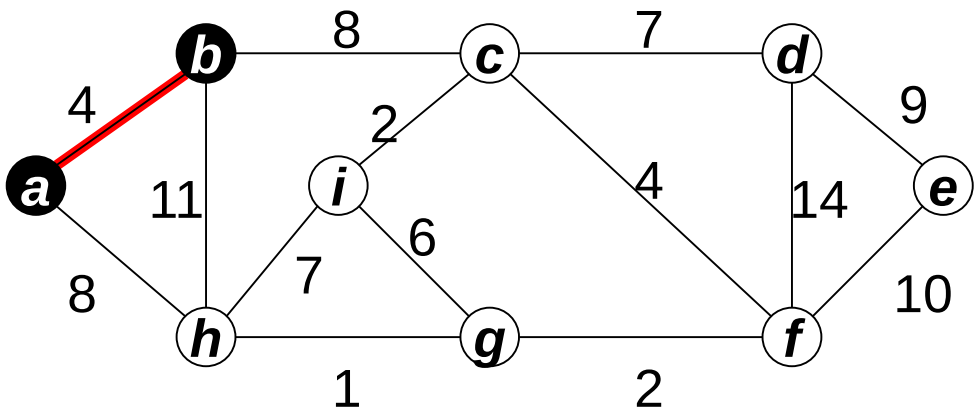
Prim(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $Q \leftarrow$ BuildHeap(V)
3. **while** $Q \neq \emptyset$ **do**
4. u←deleteMin(Q)
5. **for each** vertex $v \in$ Adj[u] **do**
6. **if** $v \in Q$ and $d[v] > w(u,v)$ **then**
7. DecreaseKey(Q,v,w(u,v)); $\pi[v] \leftarrow u$

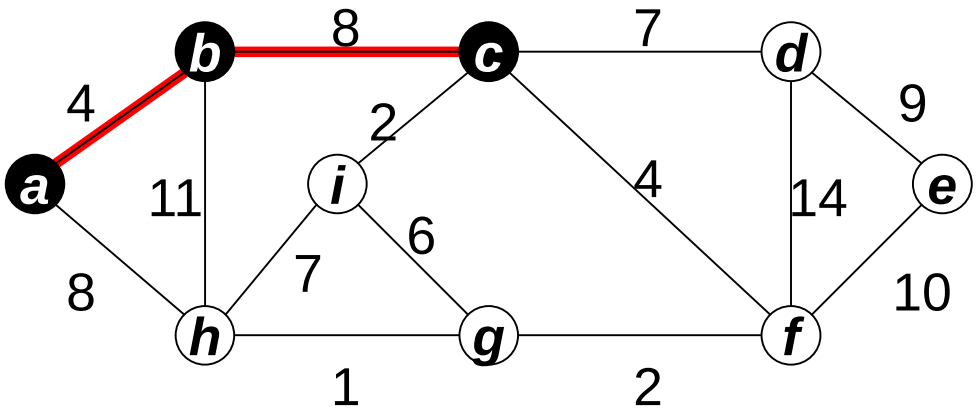
DecreaseKey(Q,v,d) operation:
Reduces d[v] of v in Q to d and performs heapifyUp from the current position of v to repair heap property.

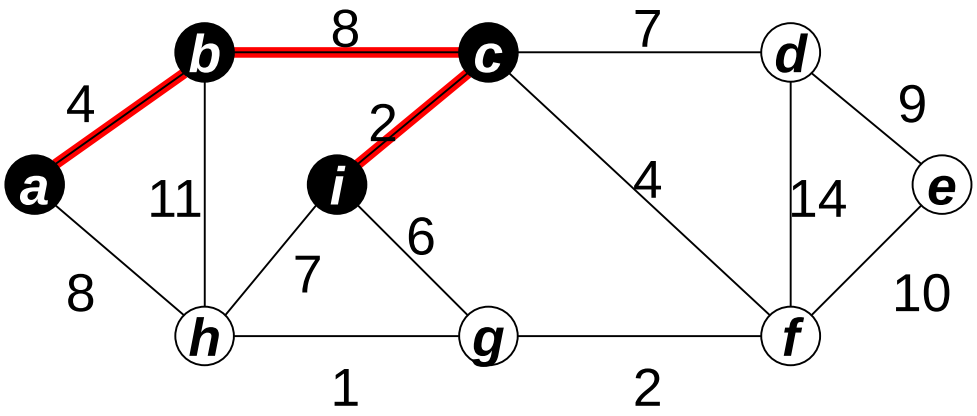
Prim‘s Algorithm



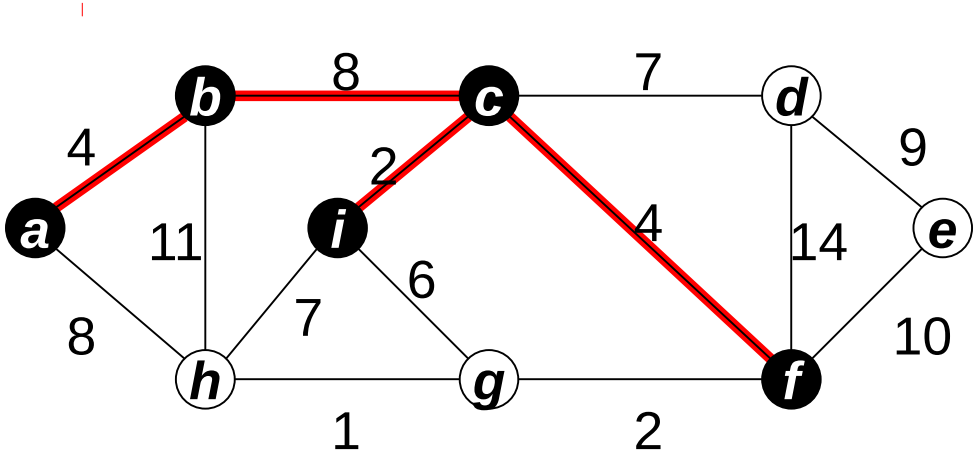


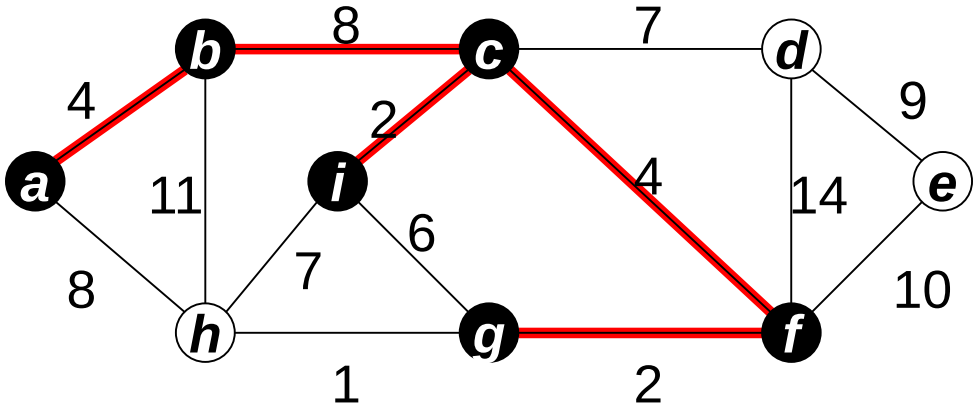
Prim‘s Algorithm



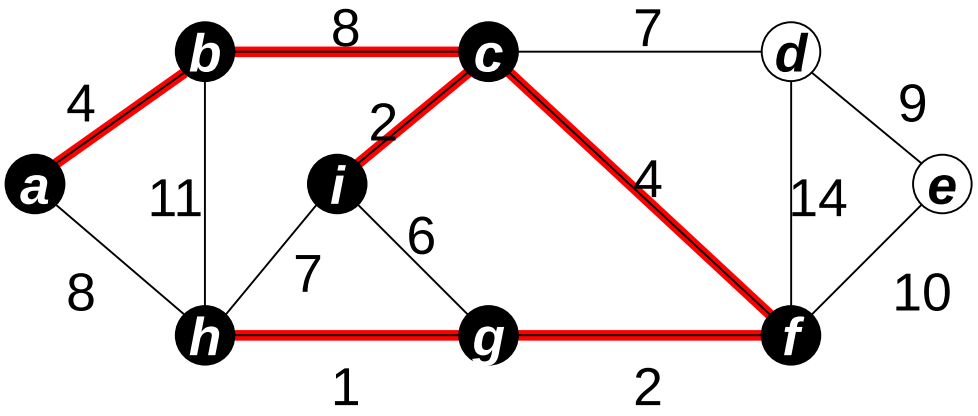


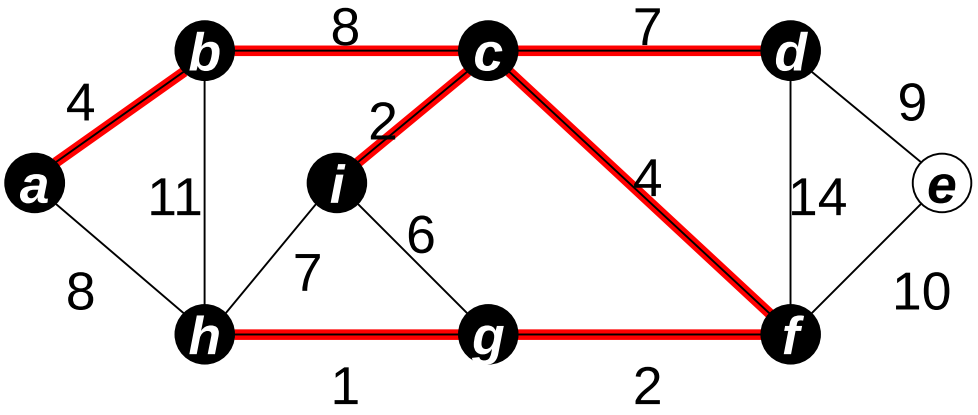
Prim‘s Algorithm



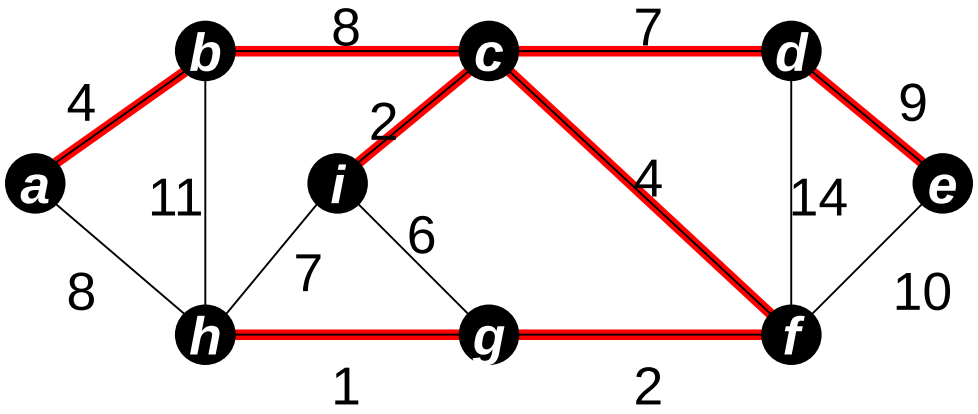


Prim‘s Algorithm





Prim‘s Algorithm



Correctness of Prim‘s algorithm for general instances (G,w): results from correctness of Generic-MST and Corollary 9.