Graph Search

- Breadth first
- Depth first
  - Topological sort

Graph search: some concepts

- To keep track of progress, graph search colors each node *white, gray, or black*
  - All nodes start with *white*
  - A node is *discovered* at the first time it is encountered during the search, at which time it becomes *non-white*
  - Different search distinguishes itself by a different way to *blacken* or *gray* nodes

Breadth-first search

- Given a graph $G=<N, E>$, and a source node, s, start breadth-first search from s.
- Expands the frontier between discovered and undiscovered nodes uniformly across the breadth of the frontier
  - Discovers all nodes at distance k from s before discovering any nodes at distance $k+1$.
- Coloring: if $(u,v) \in E$ and vertex u is black, then node v is either black or gray
  - Black node: discovered and the node itself is finished
  - Gray node: discovered but not finished

Breadth-first tree

- Breadth-first search constructs a breadth-first tree
  - Initially starts with its root, the source node
  - Whenever a white node v is discovered in scanning the adjacency list of an already discovered node u, the node v and the edge (u,v) are added into the tree. Now u is the *parent* of v.

Breadth-first search algorithm

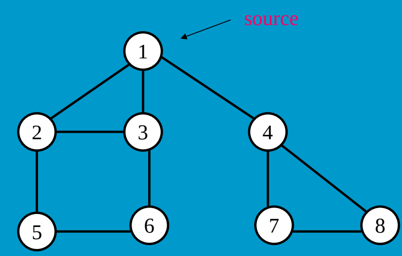
```
BFS(G,s)
{
   for each node u ∈ N – {s} {
      color[u] = WHITE;
      d[u] = ∞;
      π[u] = null;
   }

   color[s] = GRAY;
   d[s] =0
   enqueue(Q, s);
```

```
while (!empty(Q)) {
    u = dequeue(Q);
    for each v adjacent to u {
        if (color[v] == WHITE) {
            color[v] = GRAY;
            d[v] = d[u] + 1;
            π[v] = u;
            enqueue(Q, v);
        }
    }
    color[u] = BLACK;
}
```

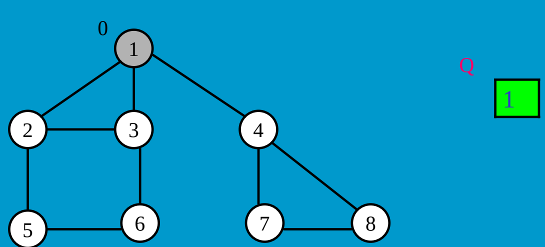
d[]: tracks shortest distance, assuming each edge's weight is 1

π[]: tracks the parent-child relationship in the breadth-first tree

Breadth-first Example

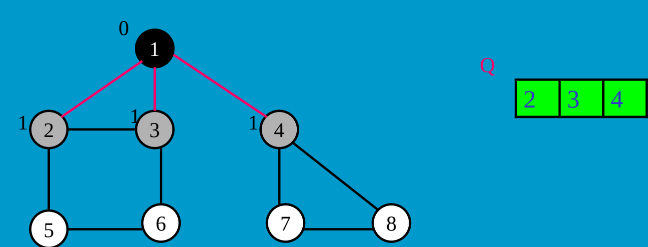


Breadth-first Example

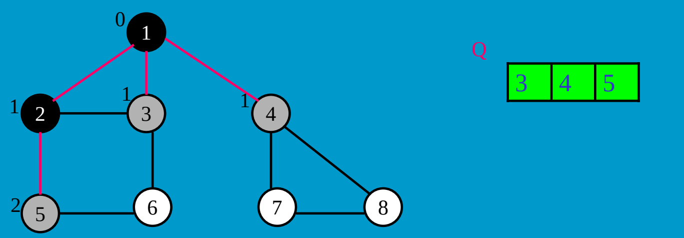


The number on the left of each node *i* is *d[i]*, the shortest distance

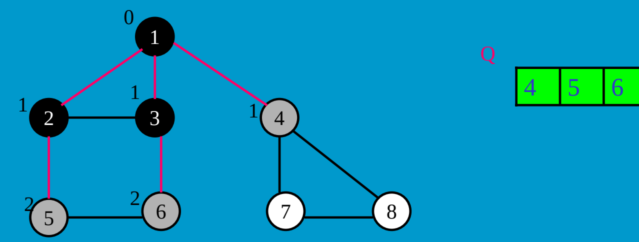
Breadth-first Example



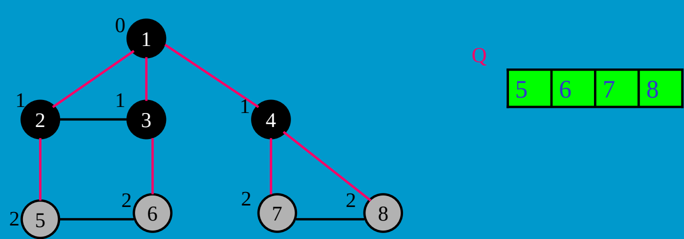
Breadth-first Example



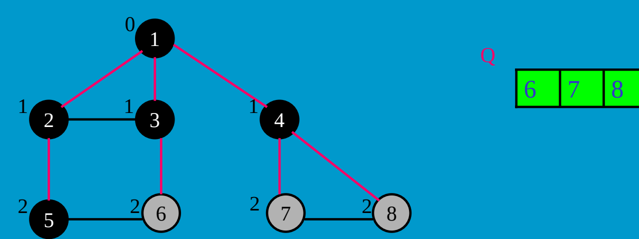
Breadth-first Example



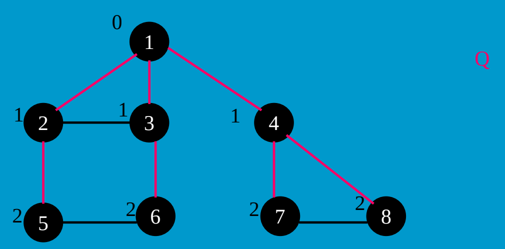
Breadth-first Example



Breadth-first Example



Breadth-first Example



Theorem: Correctness of breadth-first search

- Let G=(N,E) be a directed or undirected graph, and suppose that BFS is run on G from a given source node $s \in N$.
  - Then during its execution, BFS discovers every node $v \in N$ that is reachable from the source s,
  - and upon termination, d[v] is the shortest distance from s. And moreover, for $v \neq s$, one the shortest path from s to v is the shortest path from s to $\pi[v]$ followed by edge $(\pi[v], v)$

Depth-first search

- Search deeper in the graph whenever possible
  - Edges are explored out of the most recently discovered node *v* that still has undiscovered edges leaving it
  - When all of *v*'s edges have been explored, the search "backtracks" to explore edges leaving the vertex from which *v* was discovered
  - This process finishes until all nodes reachable from the original source are discovered
  - Select one undiscovered node as the new source and continue the process

Depth-first search coloring and time stamps

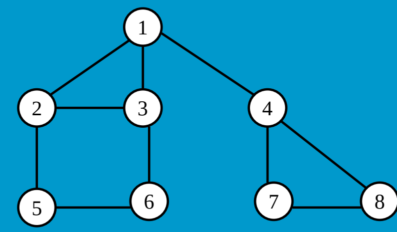
- Coloring
  - Each nodes is initially *white*
  - A node is *grayed* if it is discovered during the search and *blackened* if it is finished, that is, when its adjacency list has been examined completely
- Timestamps
  - Each node *v* has two timestamps
    - *d[v]* records when v is discovered (grayed)
    - *f[v]* records when v is finished (blackened)

Depth-first search algorithm

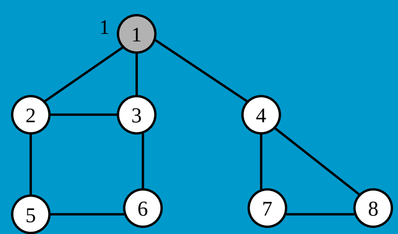
```
DFS(G)
{
    for each node u ∈ N {
        color[u] = WHITE;
        π[u] = null;
    }

    time = 0;

    for each node u ∈ N {
        if (color[u] == WHITE)
            DFS-Visit(u);
    }
}
```

```
DFS-Visit(u)
{
    color[u] = GRAY;
    d[u] = ++time;

    for each v adjacent to u {
        if (color[v] == WHITE) {
            π[v] = u;
            DFS-Visit(v)
        }
    }

    color[u] = BLACK;
    f[u] = ++time;
}
```

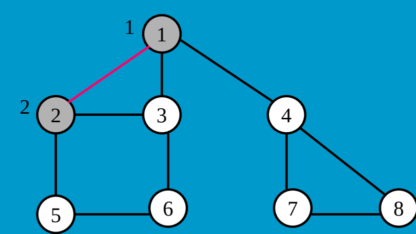
Example: depth-first search undirected graph



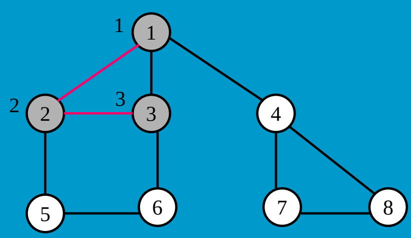
Example: depth-first search undirected graph



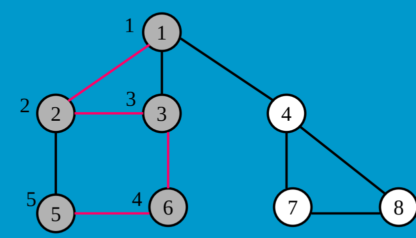
Example: depth-first search undirected graph



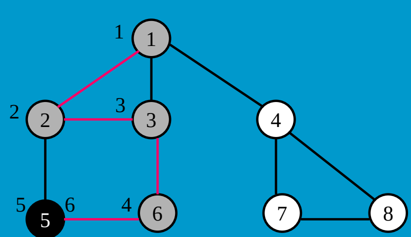
Example: depth-first search undirected graph



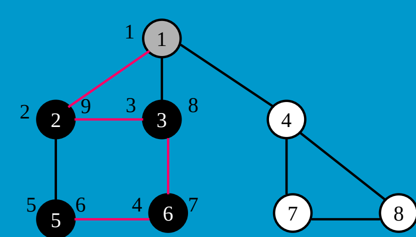
Example: depth-first search undirected graph



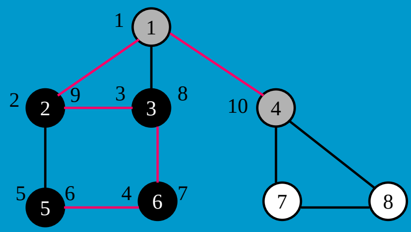
Example: depth-first search undirected graph



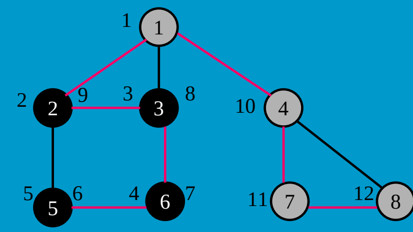
Example: depth-first search undirected graph



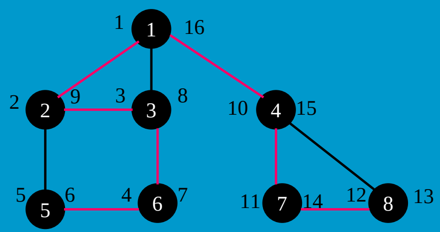
Example: depth-first search undirected graph



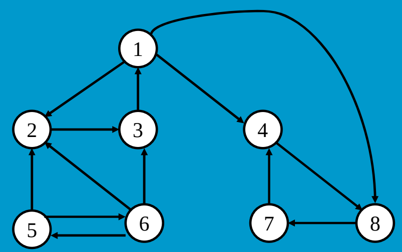
Example: depth-first search undirected graph



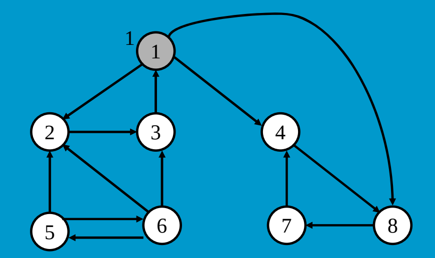
Example: depth-first search undirected graph



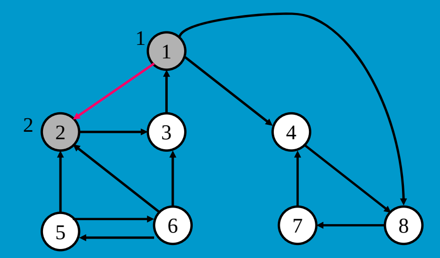
Example: depth-first search directed graph



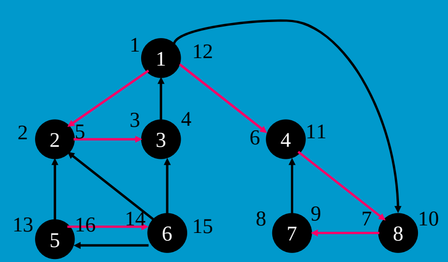
Example: depth-first search directed graph



Example: depth-first search directed graph



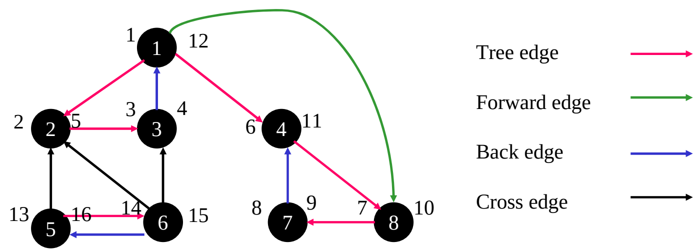
Example: depth-first search directed graph



Classification of graph edges

- After depth-first search of a directed graph, we can classify the graph edges into four categories
  - Tree edge
    - An edge in the search tree
  - Back edge
    - An edge *(u,v)* not in search tree and *v* is an ancestor of *u*
    - Indicates a loop
  - Forward edge
    - An edge *(u,v)* not in search tree and *u* is an ancestor of *v*
  - Cross edge
    - An edge *(u,v)* not in search tree and *v* is neither an ancestor nor a descendant of *u*

Example: depth-first search directed graph



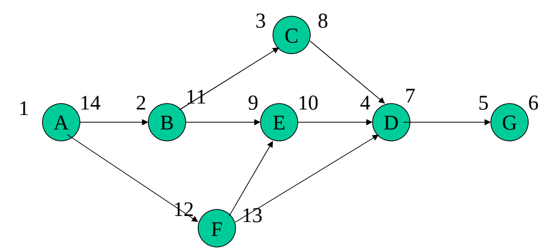
Classify edges during search

- When edge *(u,v)* is first explored
  - If *v* is white, *(u,v)* is a tree edge
  - If *v* is gray, *(u,v)* is a back edge
  - If *v* is black, *(u,v)* is a forward edge or a cross edge

Topological Sort

- Given an acyclic directed graph, topological sort finds a topological ordering of the nodes such that if there exists an edge *(u,v)*, then node *u* precedes node *v* in the ordering list.
- The finished time numbering gives us a reverse topological ordering
  - A node is finished after all the nodes it reaches have finished

Example



Ordered by d[]: A B C D G E F
Ordered by f[]: G D C E B F A
Reverse: A F B E C D G