



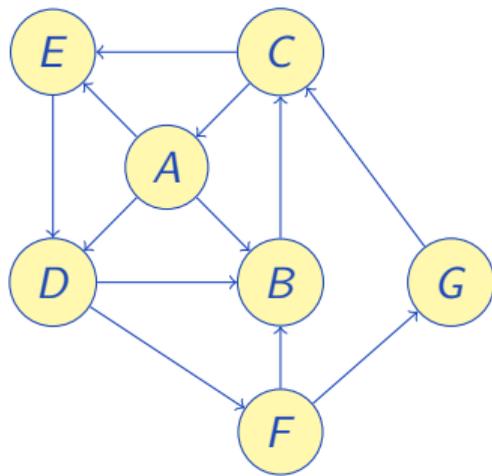
ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

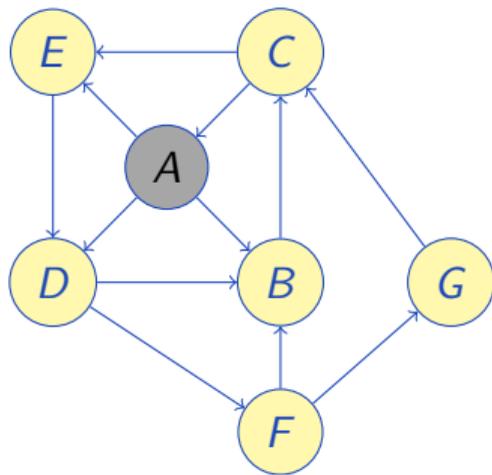
Week 4: Algorithms on Graphs Lecture 8: Breadth First Search

Maxim Buzdalov
Saint Petersburg 2016

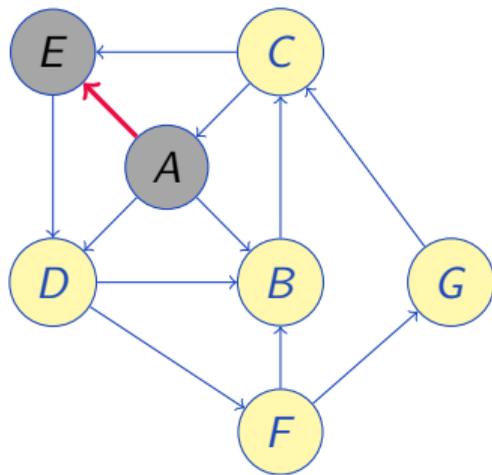
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



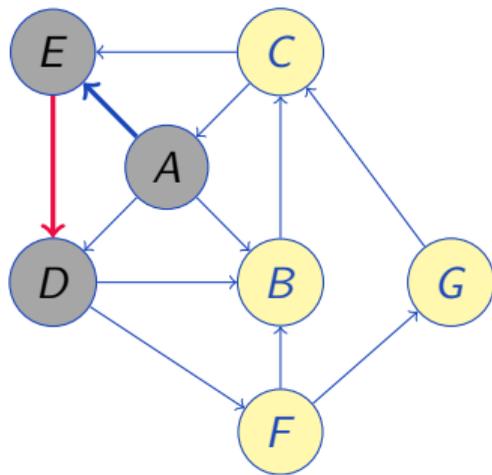
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



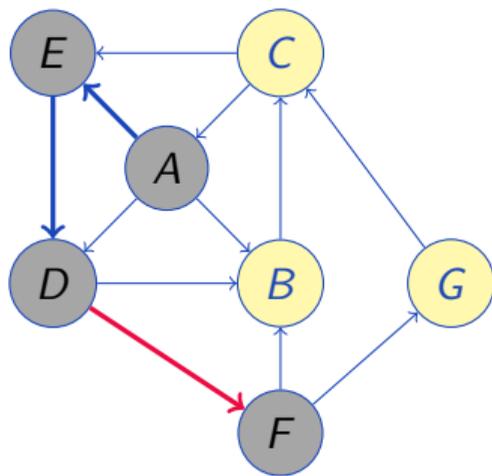
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



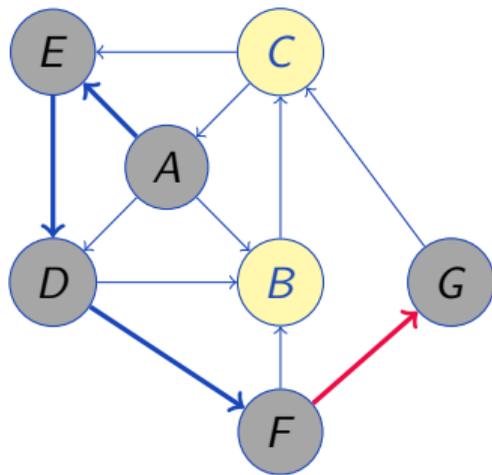
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



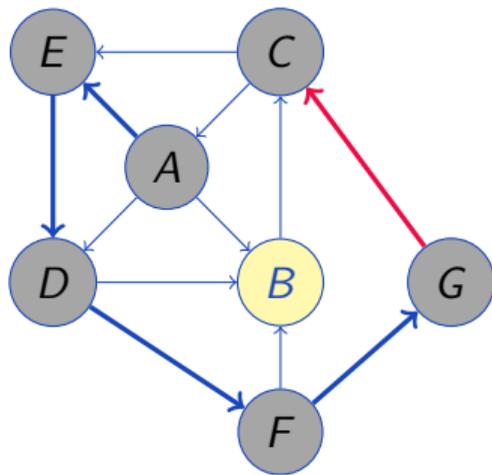
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



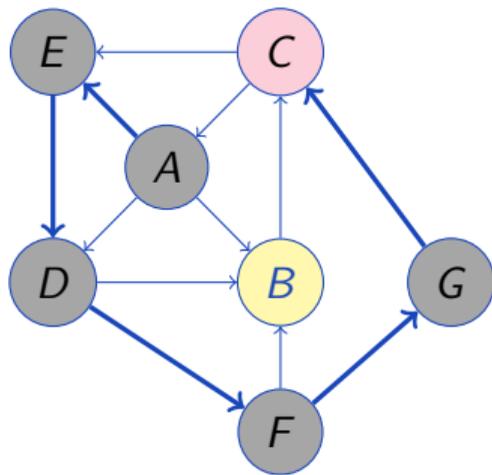
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



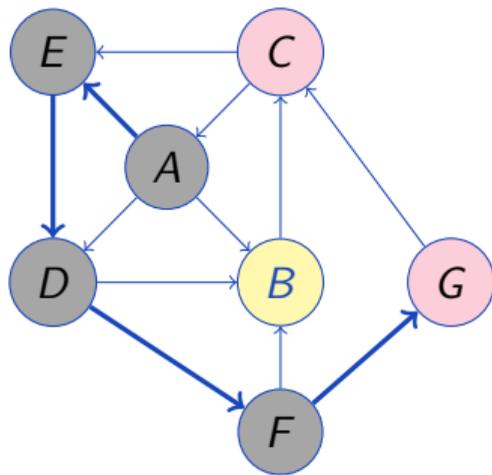
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



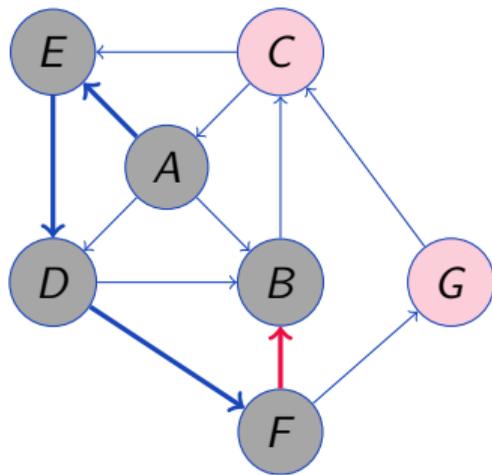
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



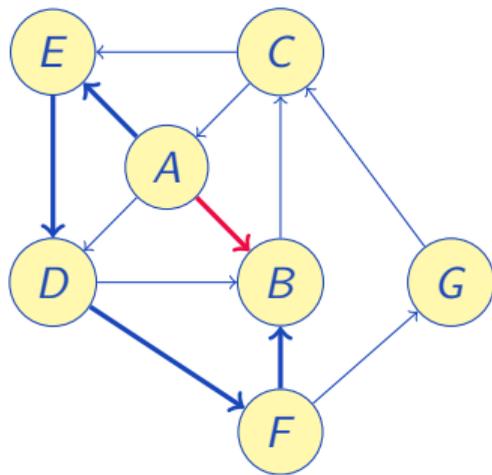
Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



Depth First Search can check if vertex B is reachable from vertex A .
But the path can be quite long. . .



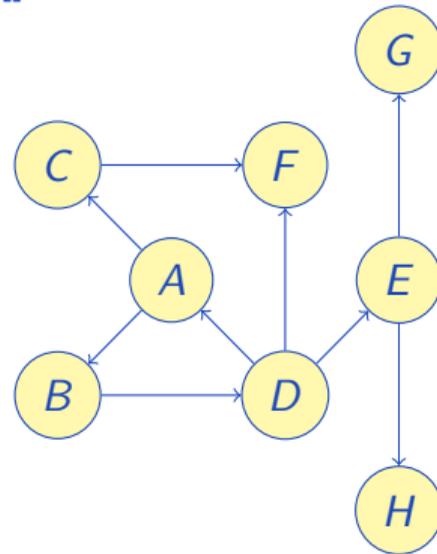
What if we want to find the **shortest** path?

```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

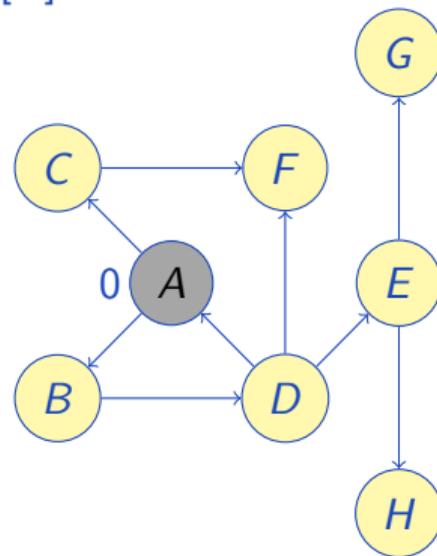


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [A]

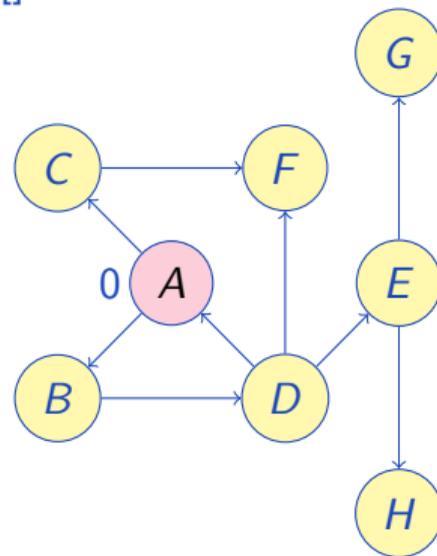


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0, \text{PUSH}(Q, v_0)$    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow \text{POP}(Q)$                  ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
         $\text{PUSH}(Q, u)$                  ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

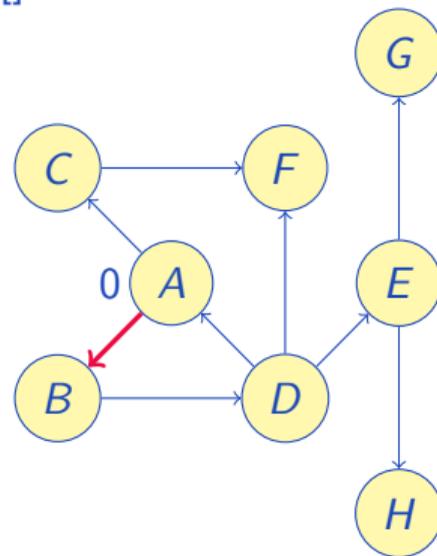


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

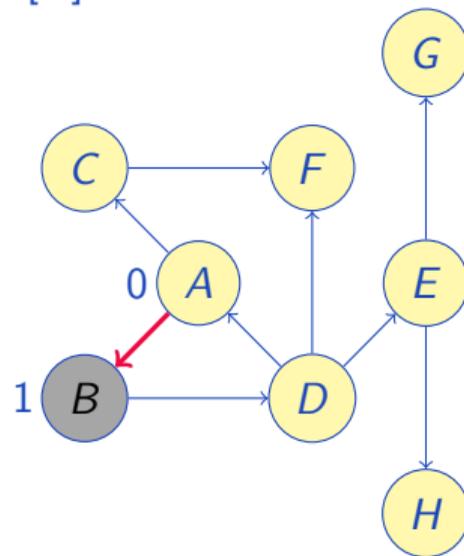


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

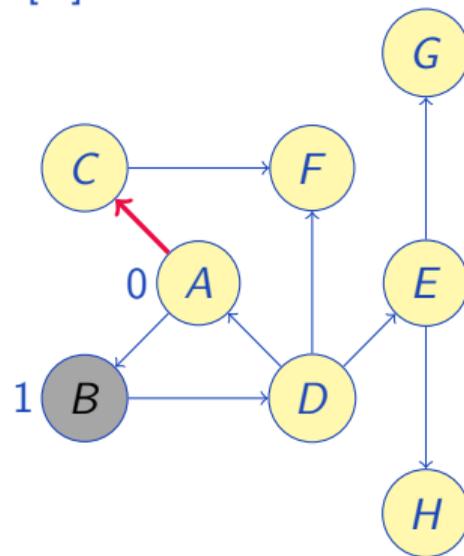
Queue: [B]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0, \text{PUSH}(Q, v_0)$    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow \text{POP}(Q)$                  ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
         $\text{PUSH}(Q, u)$                  ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

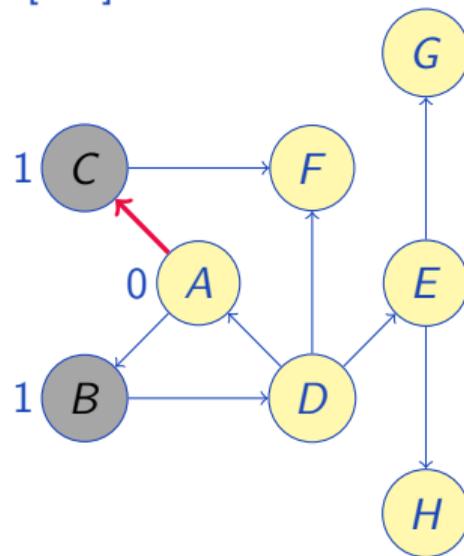
Queue: [B]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

Queue: [CB]

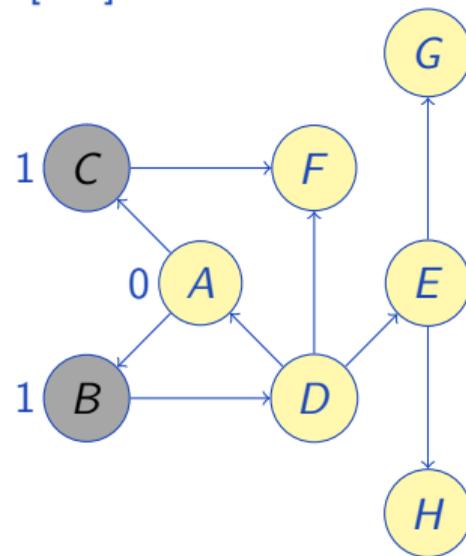


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [CB]

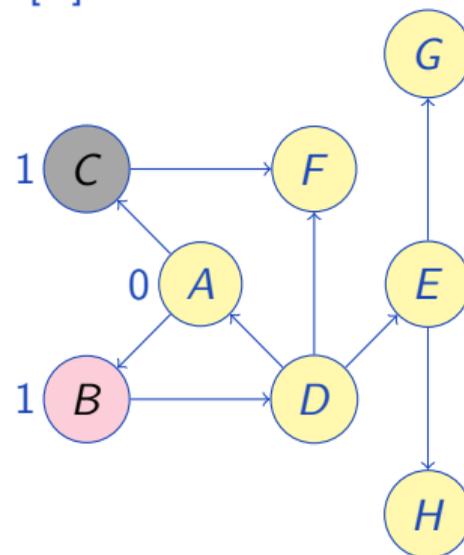


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0, \text{PUSH}(Q, v_0)$    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow \text{POP}(Q)$                  ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
         $\text{PUSH}(Q, u)$                  ▷ Put to queue
      end if
    end for
  end while
end procedure

```

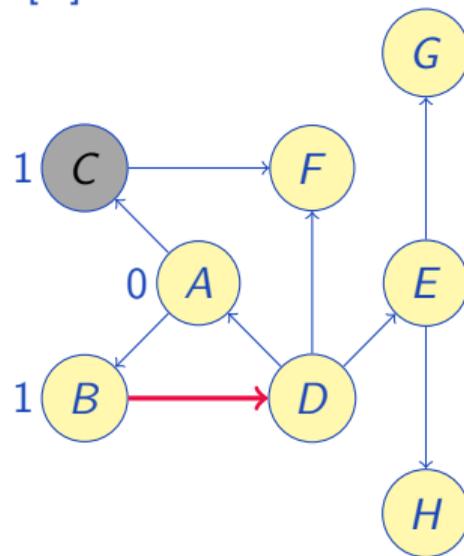
Queue: [C]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

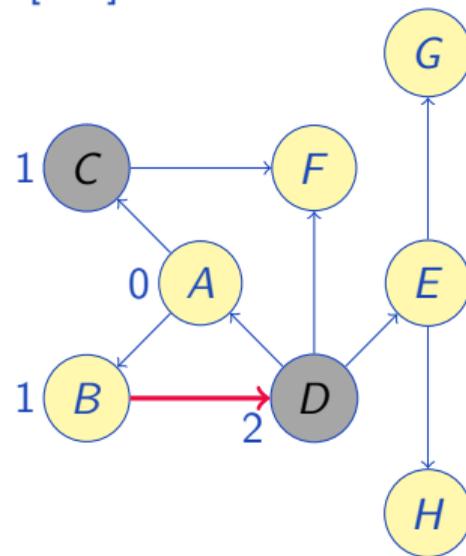
Queue: [C]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

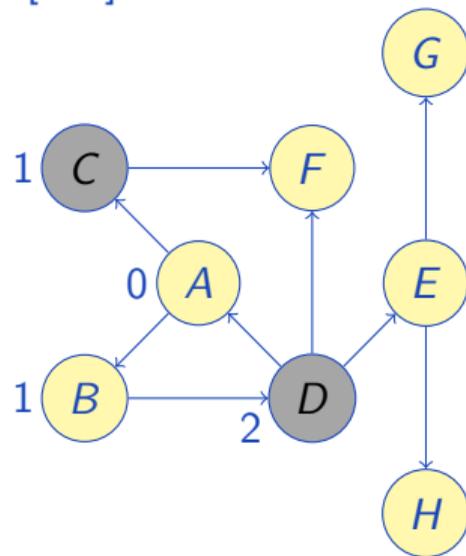
Queue: [DC]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

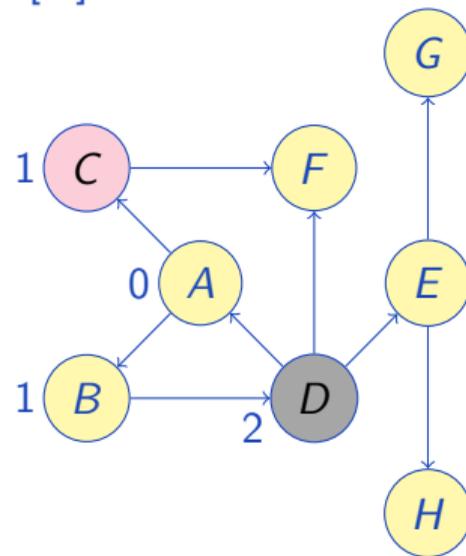
Queue: [DC]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                    ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

Queue: [D]

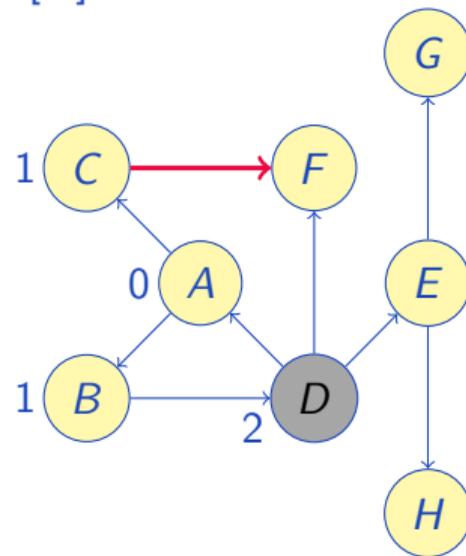


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                      ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                ▷ Get next vertex
    for  $u \leftarrow A(v)$  do          ▷ Check adjacent vertices
      if  $D[u] = \infty$  then          ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$       ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [D]

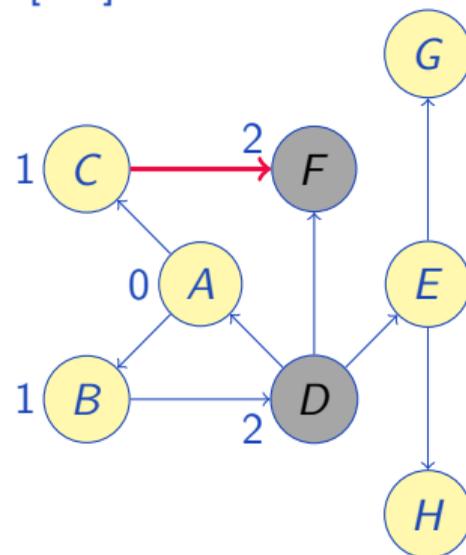


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

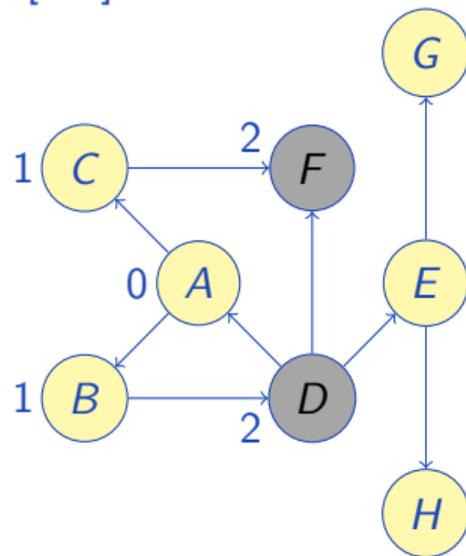
Queue: [FD]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

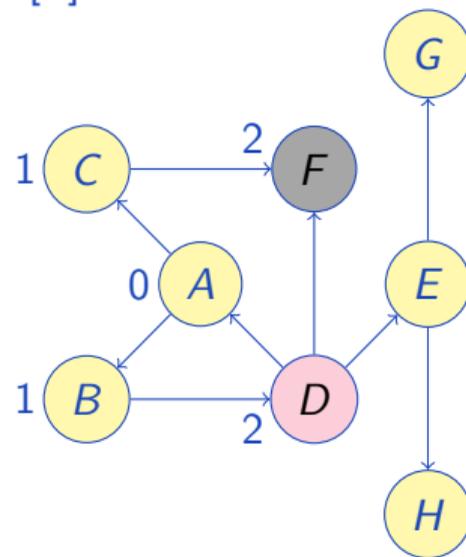
Queue: [FD]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0, \text{PUSH}(Q, v_0)$    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow \text{POP}(Q)$                  ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
         $\text{PUSH}(Q, u)$                  ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

Queue: [F]

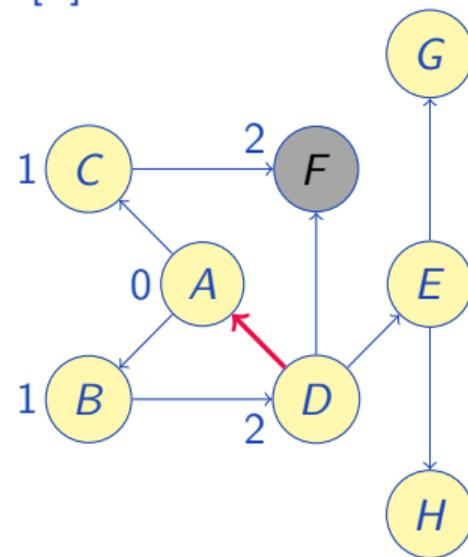


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [F]

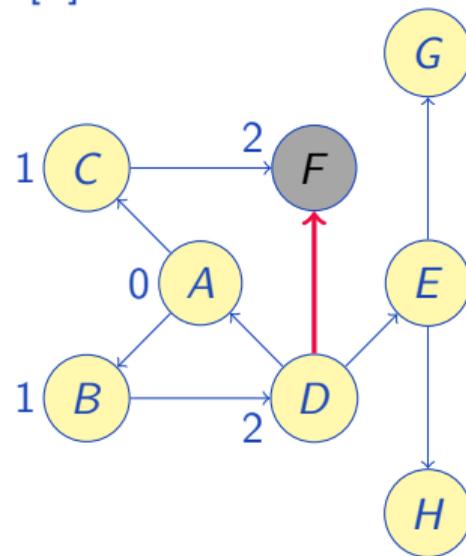


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [F]

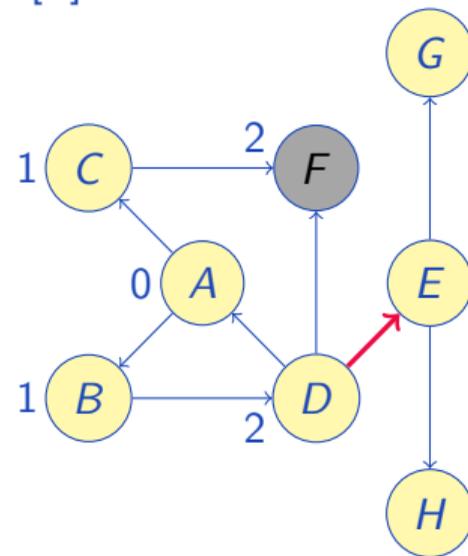


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [F]

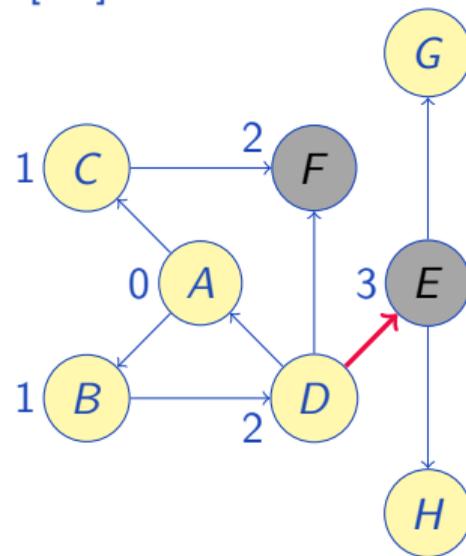


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [EF]

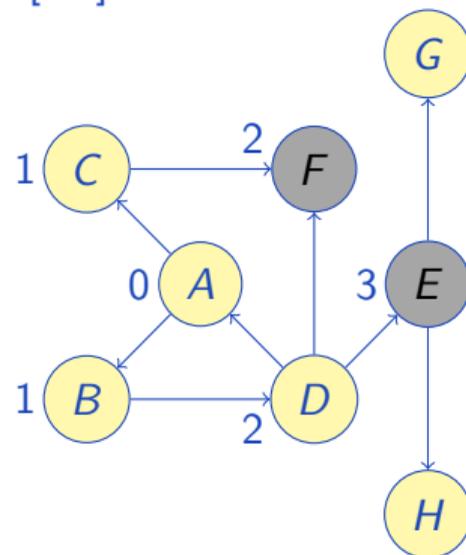


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [EF]

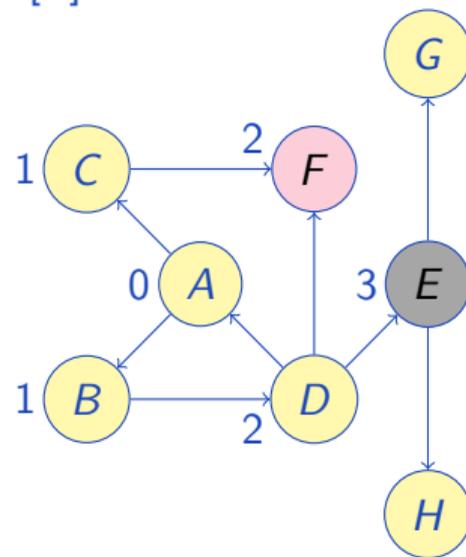


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [E]

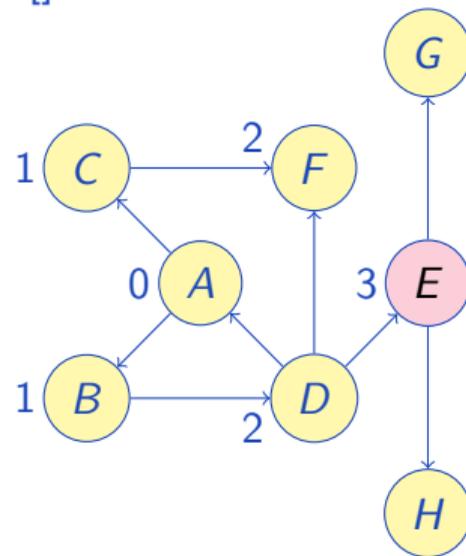


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

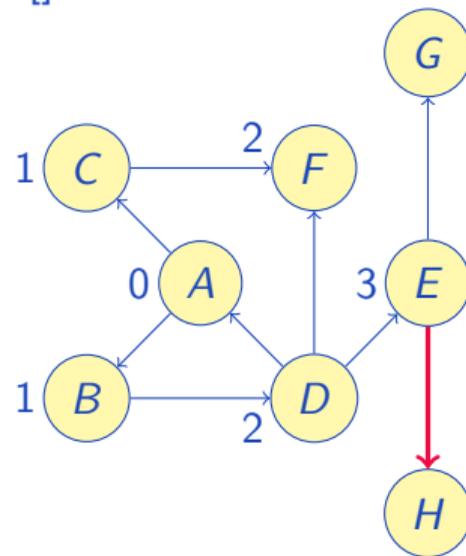


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                      ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )    ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                ▷ Get next vertex
    for  $u \leftarrow A(v)$  do          ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$       ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

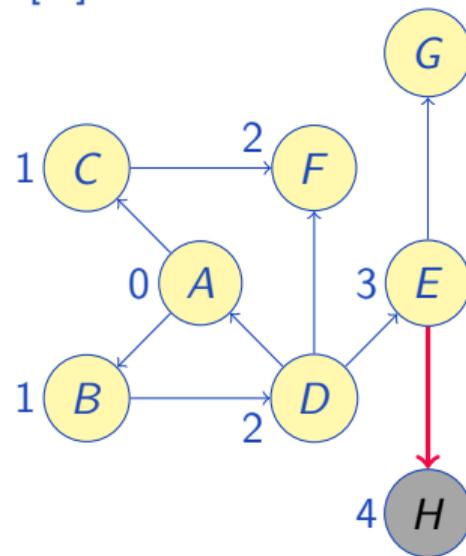


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [H]

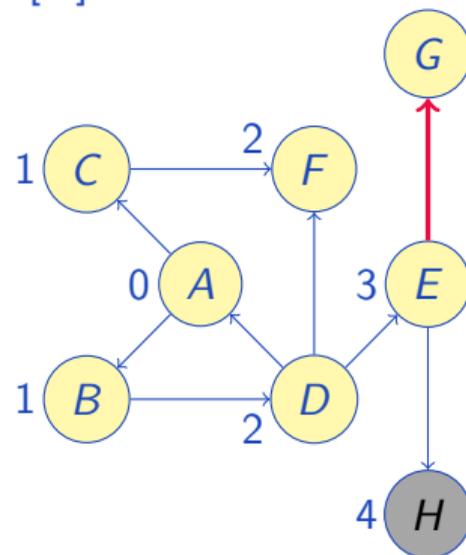


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [H]

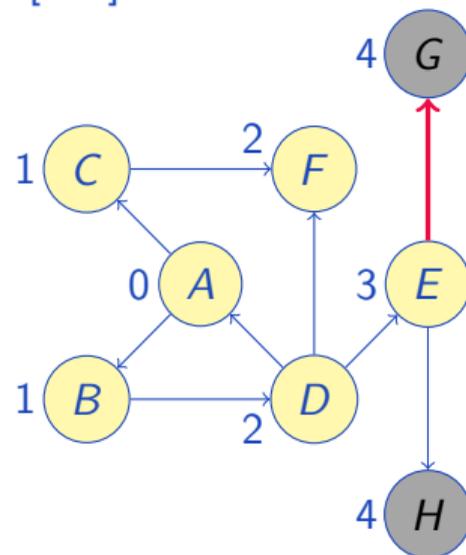


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

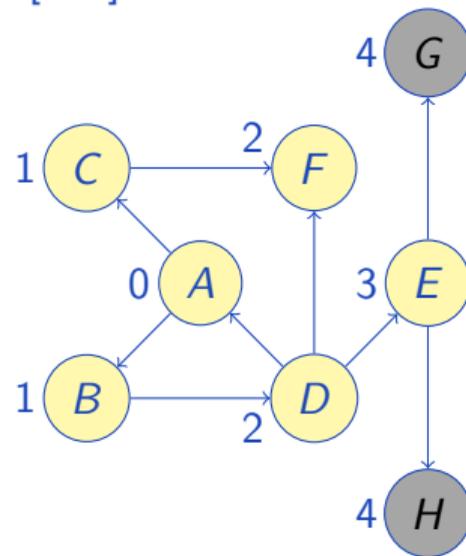
Queue: [GH]



```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                   ▷ Put to queue
      end if
    end for
  end while
end procedure
  
```

Queue: [GH]

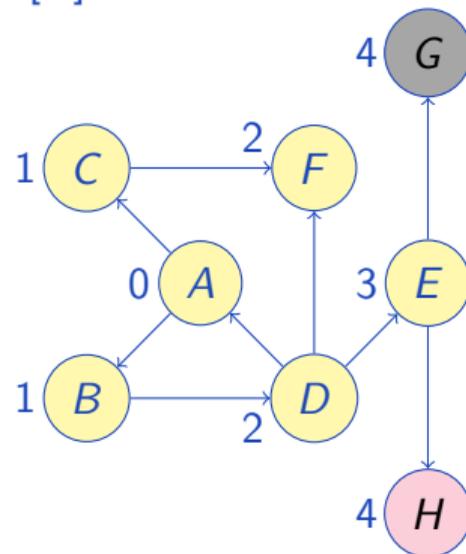


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: [G]

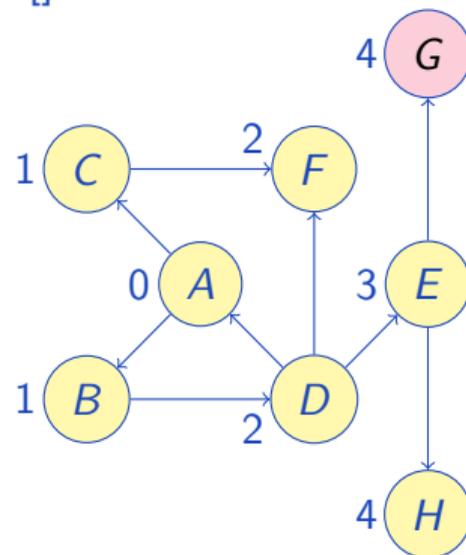


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

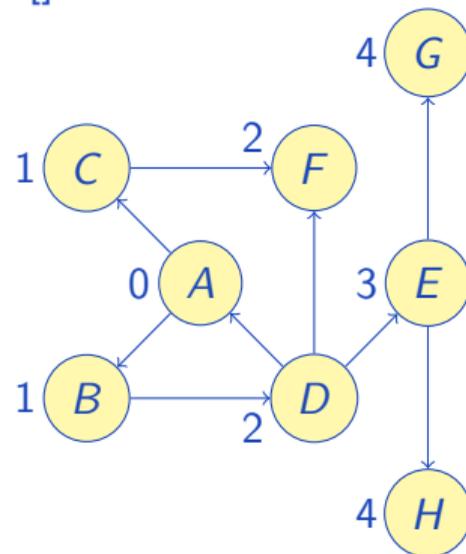


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

```

Queue: []

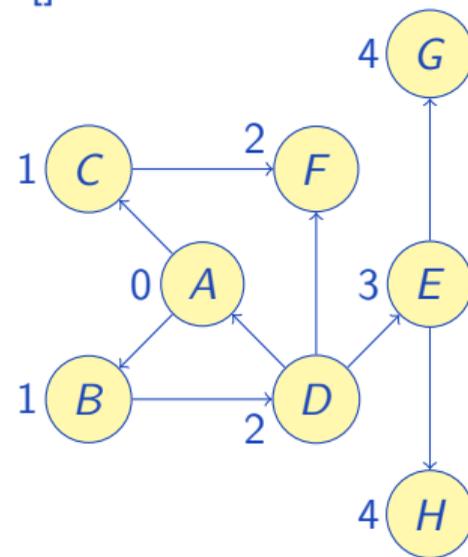


```

procedure BFS( $V, E, v_0$ )
   $A(v) = \{u \mid (v, u) \in E\}$            ▷ Adjacency list
   $D \leftarrow \{\infty\}$                  ▷ Distances to vertices
   $Q \leftarrow []$                        ▷ Queue of vertices
   $D[v_0] \leftarrow 0$ , PUSH( $Q, v_0$ )     ▷ Initialization
  while not ISEMPTY( $Q$ ) do
     $v \leftarrow$  POP( $Q$ )                 ▷ Get next vertex
    for  $u \leftarrow A(v)$  do           ▷ Check adjacent vertices
      if  $D[u] = \infty$  then           ▷ If not seen yet...
         $D[u] \leftarrow D[v] + 1$        ▷ Update distance
        PUSH( $Q, u$ )                     ▷ Put to queue
      end if
    end for
  end while
end procedure

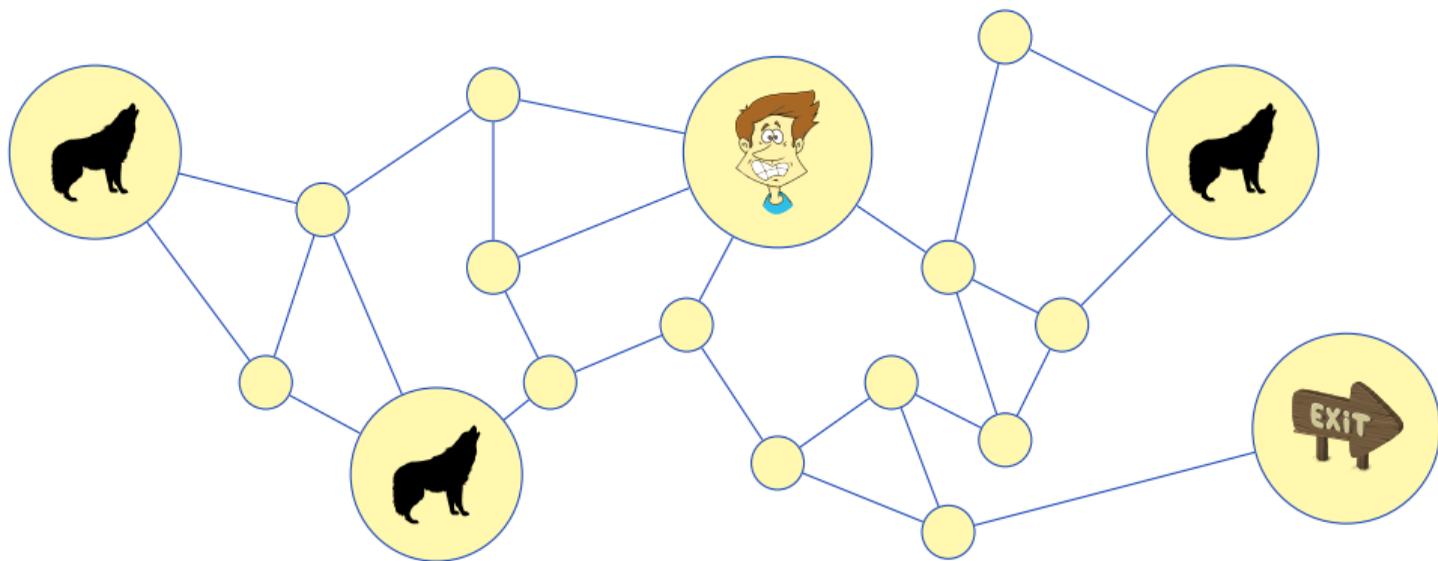
```

Queue: []



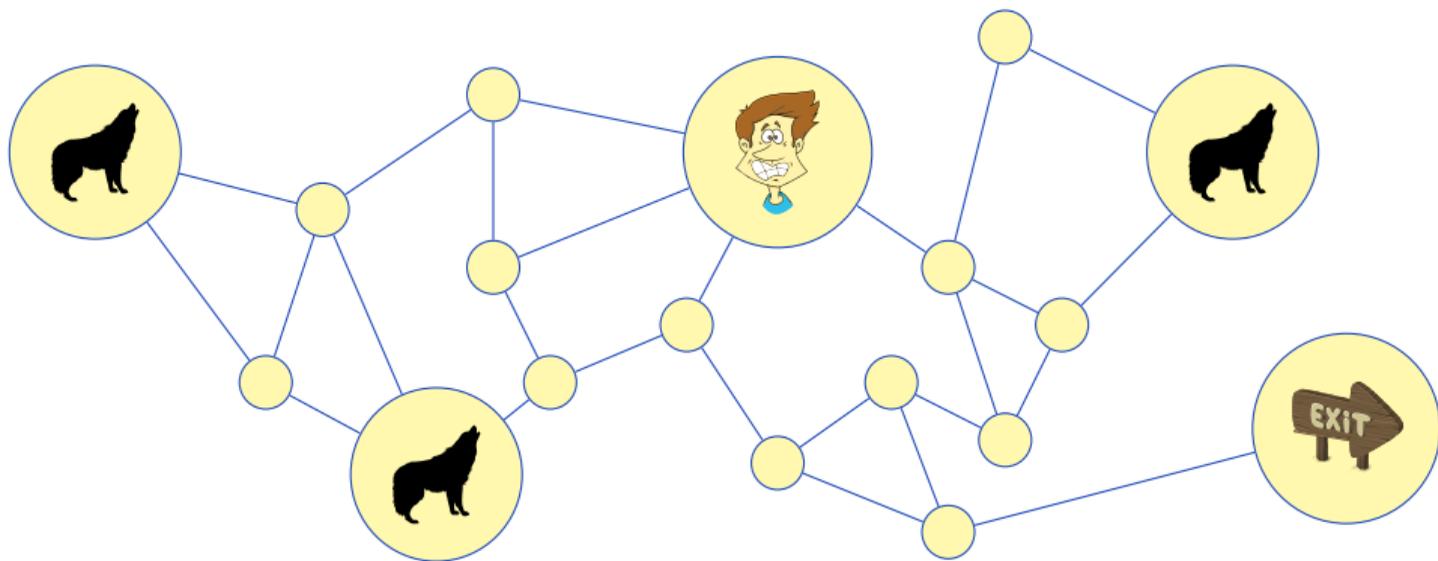
Running time: $O(|V| + |E|)$ when using adjacency list, $O(|V|^2)$ for adjacency matrix

You are in the forest, and there are wolves.
How to get out alive?



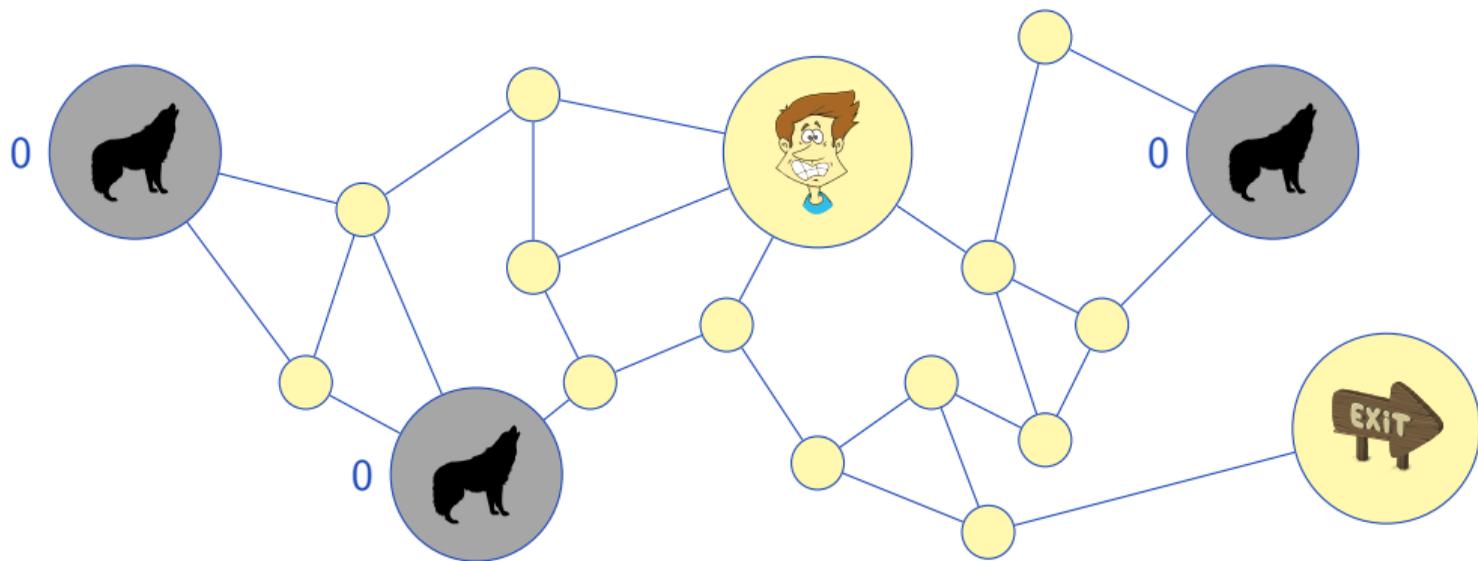
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



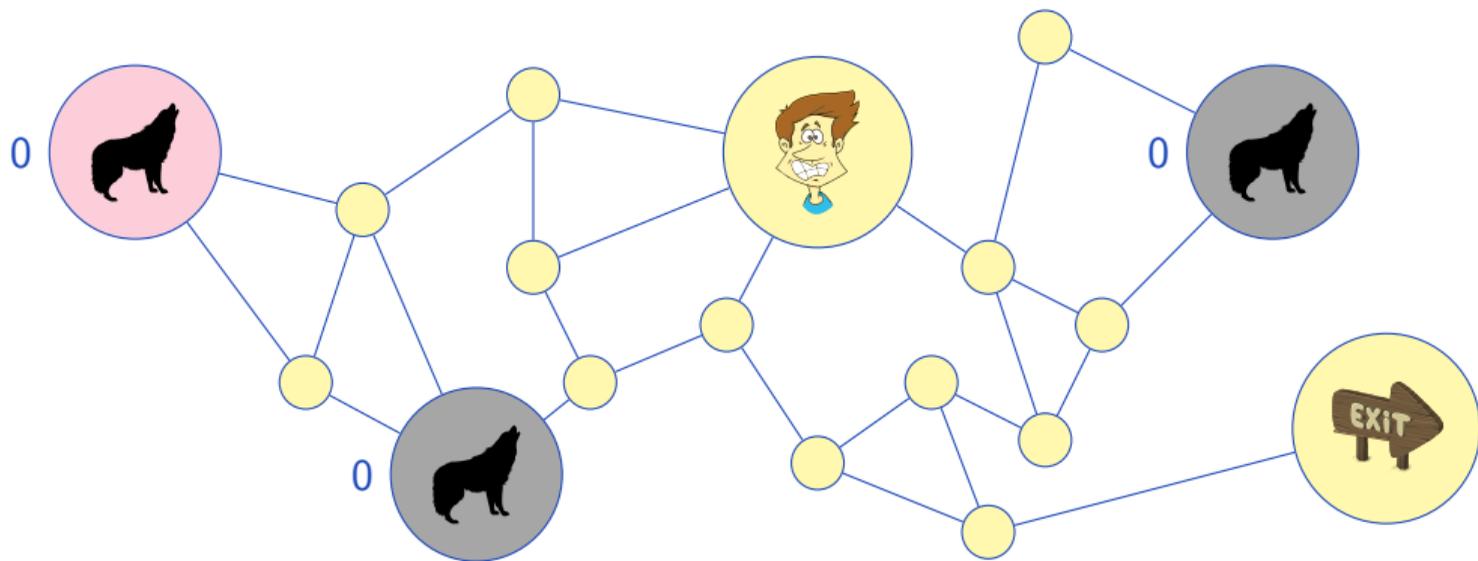
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



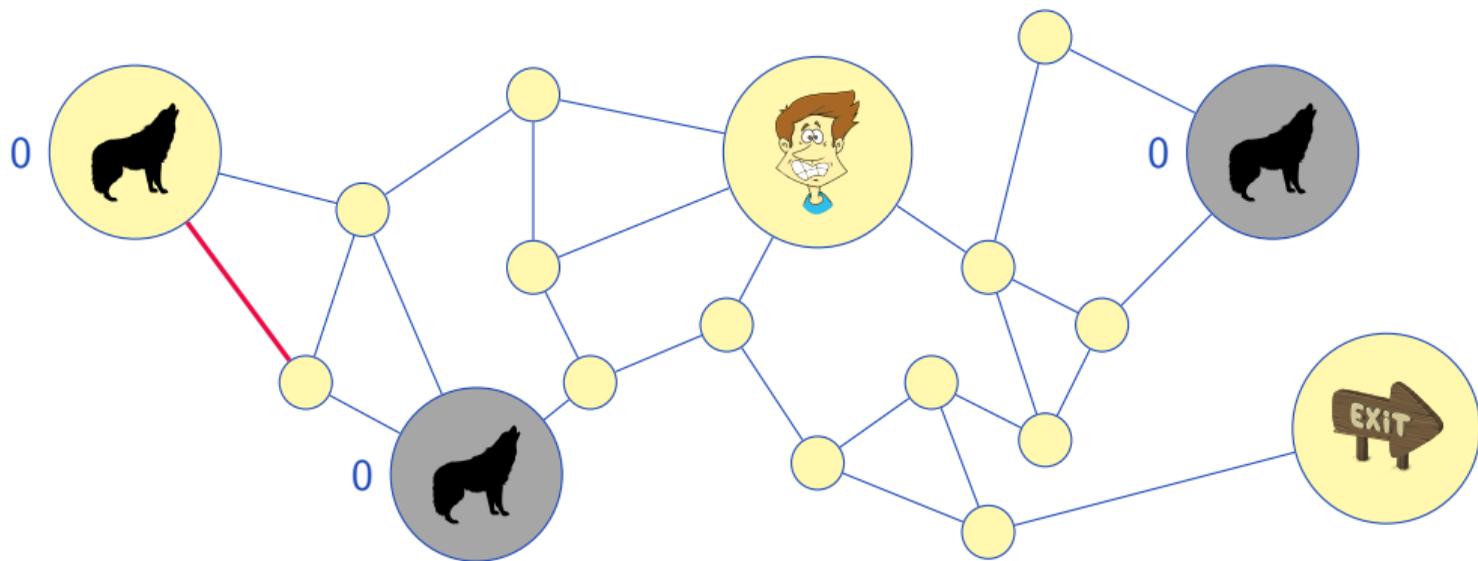
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



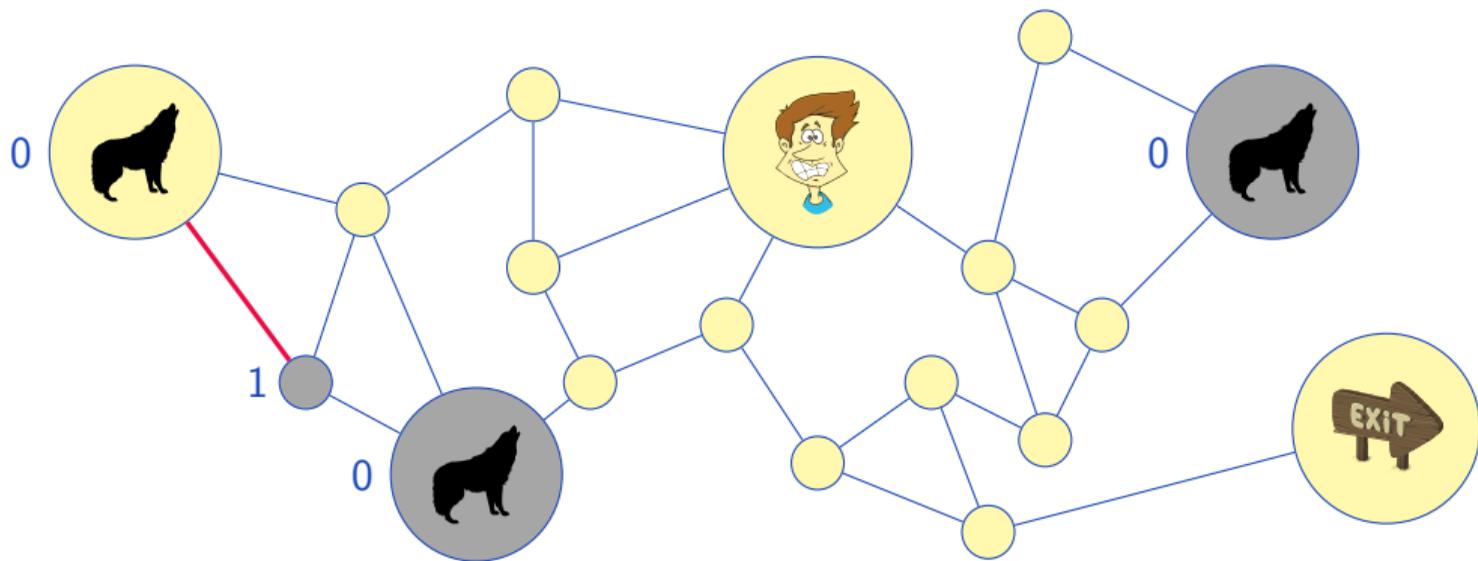
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



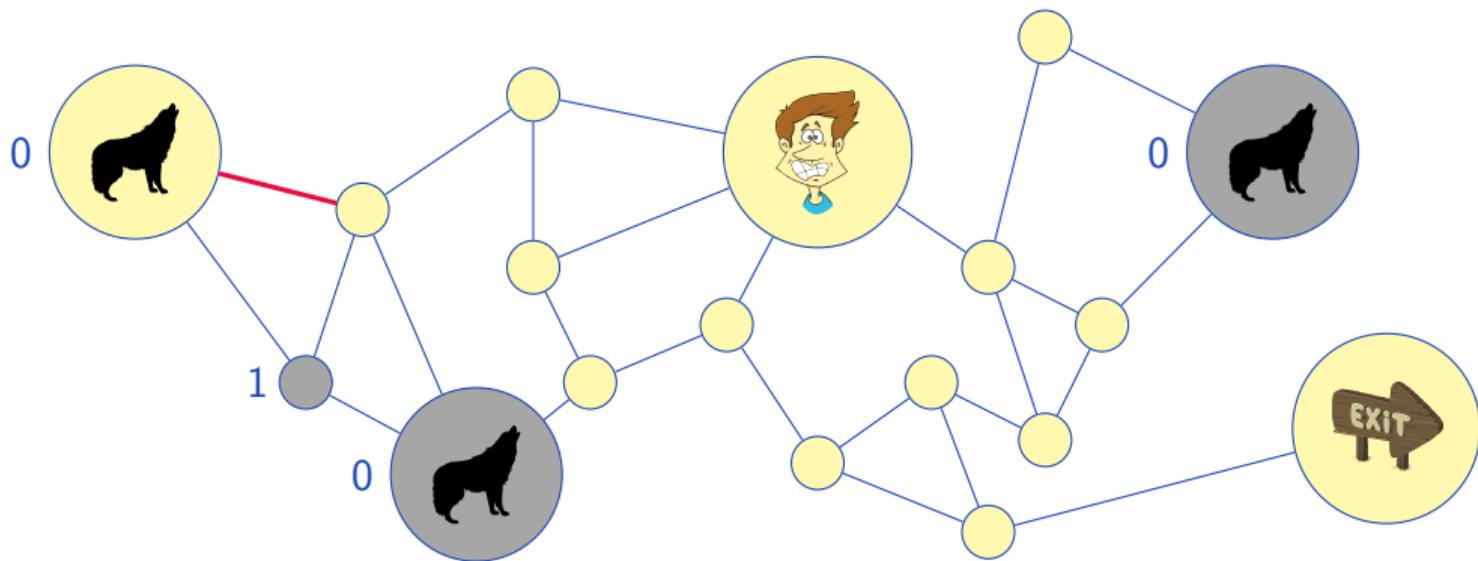
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



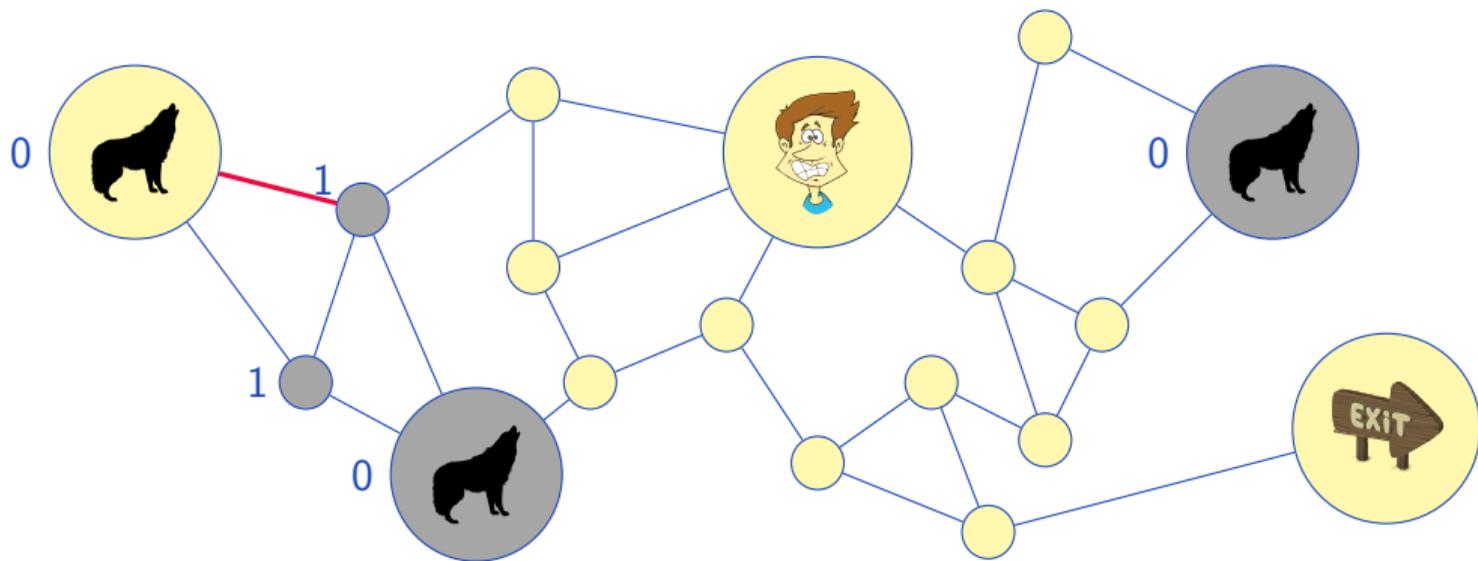
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



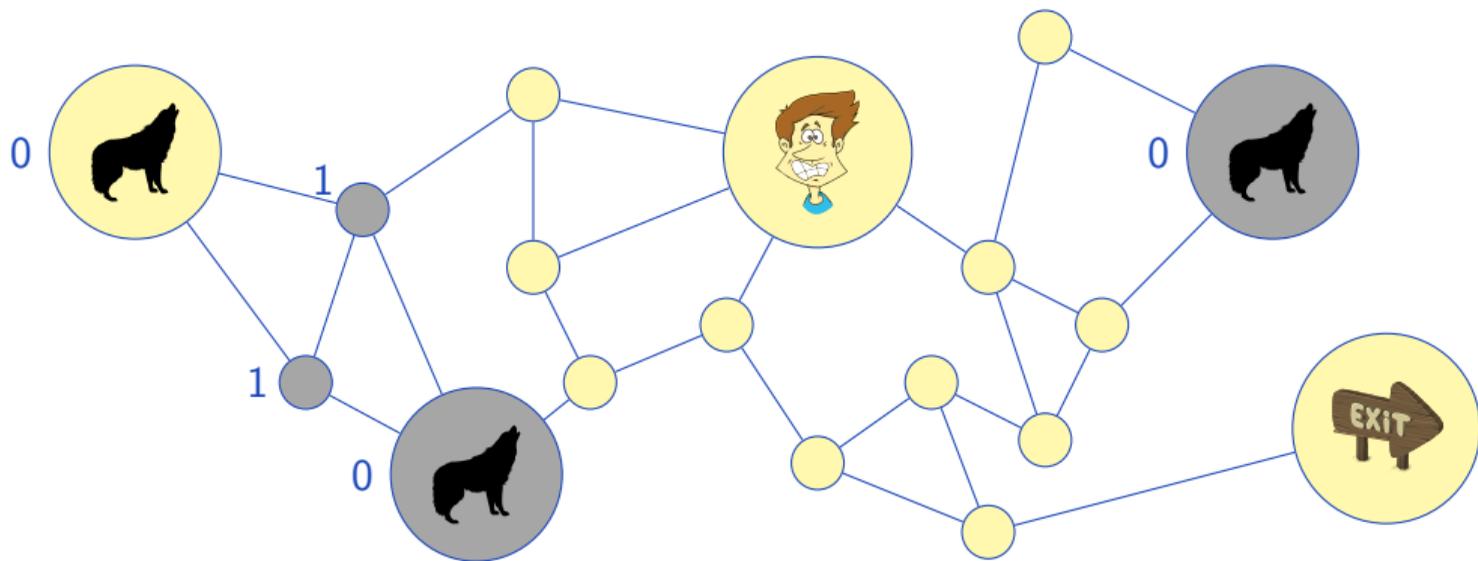
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



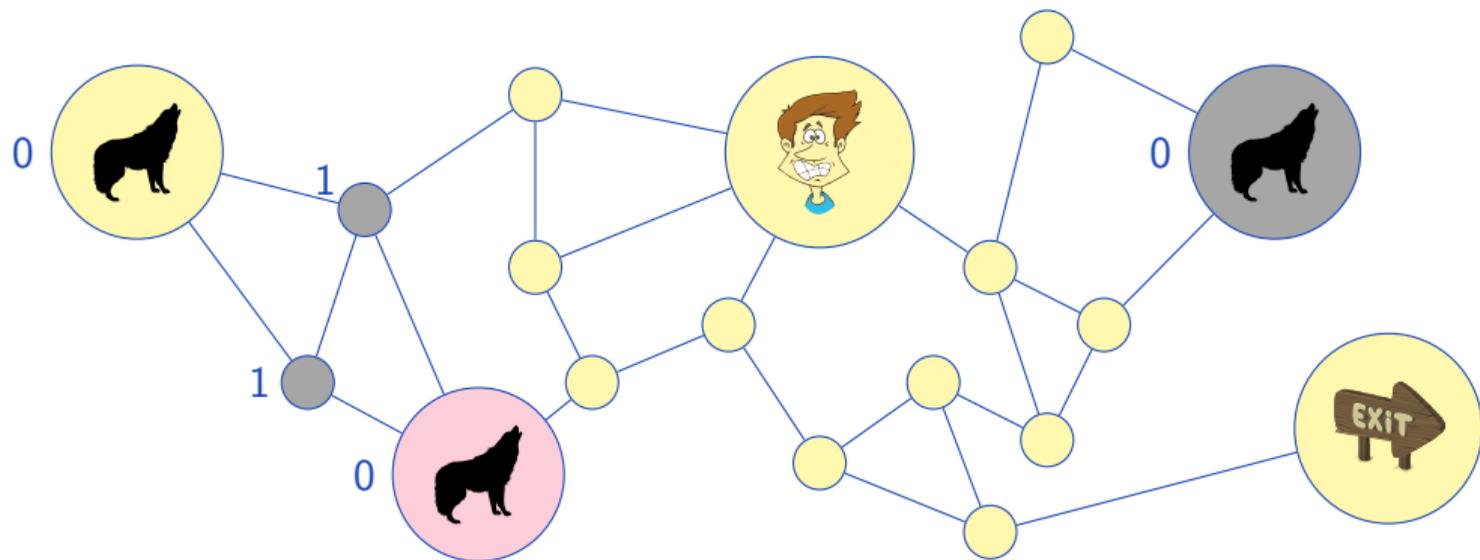
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



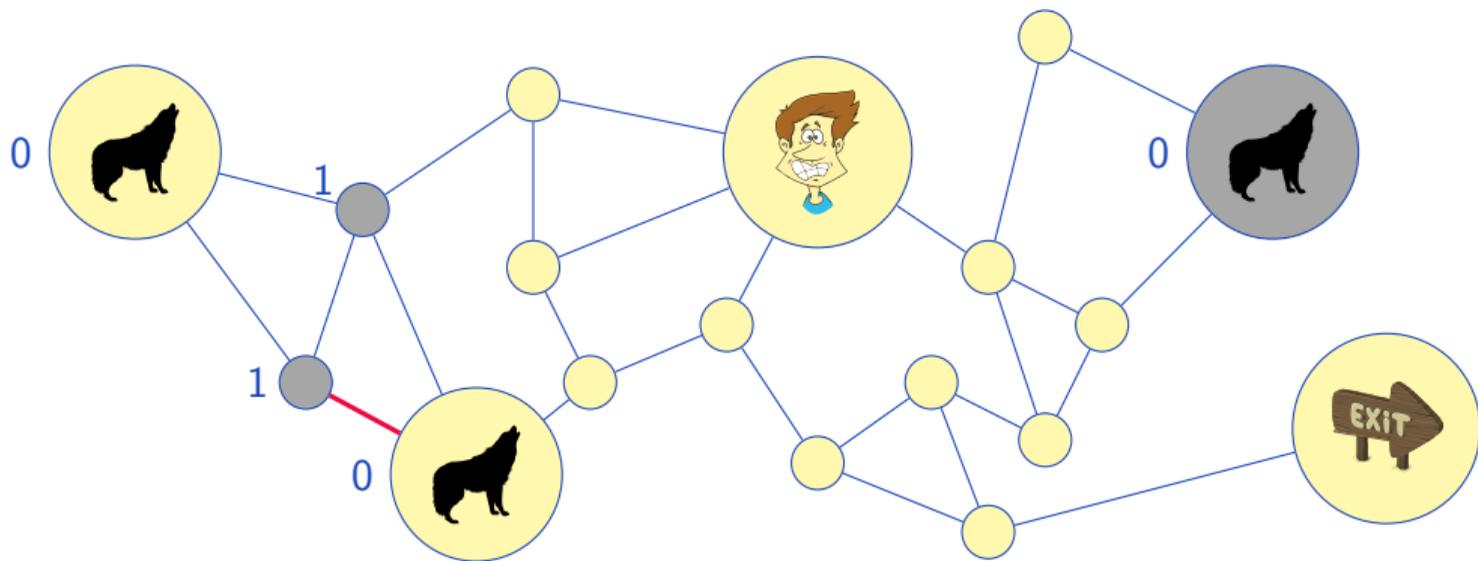
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



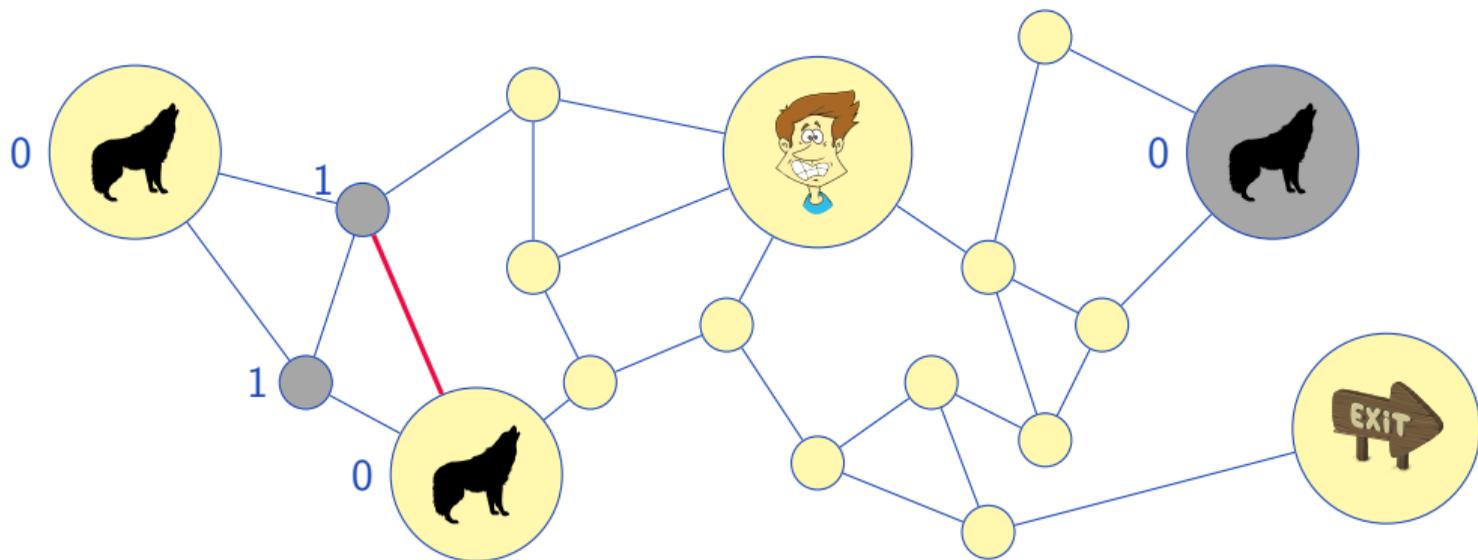
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



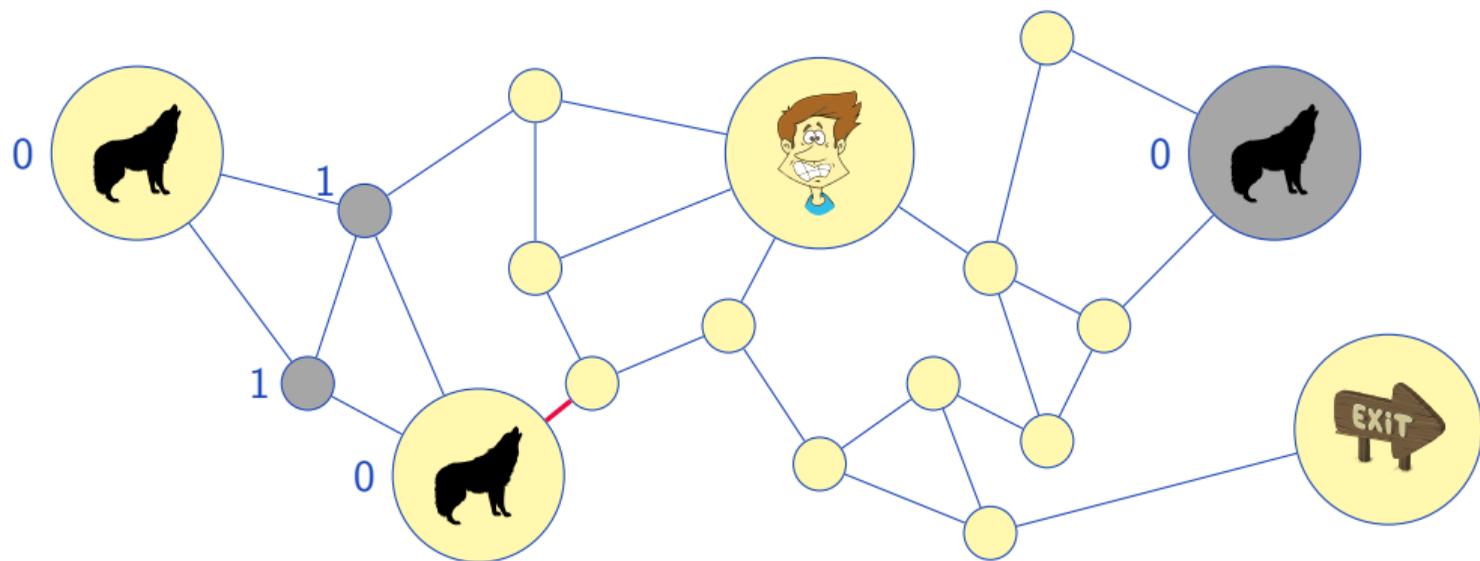
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



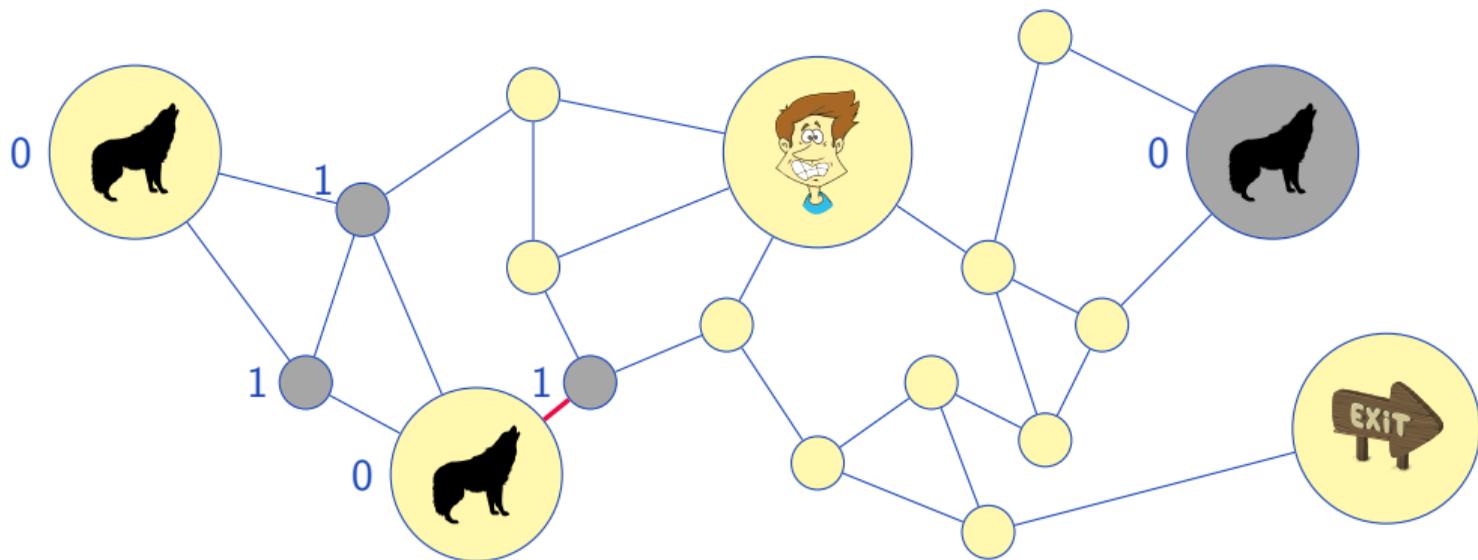
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



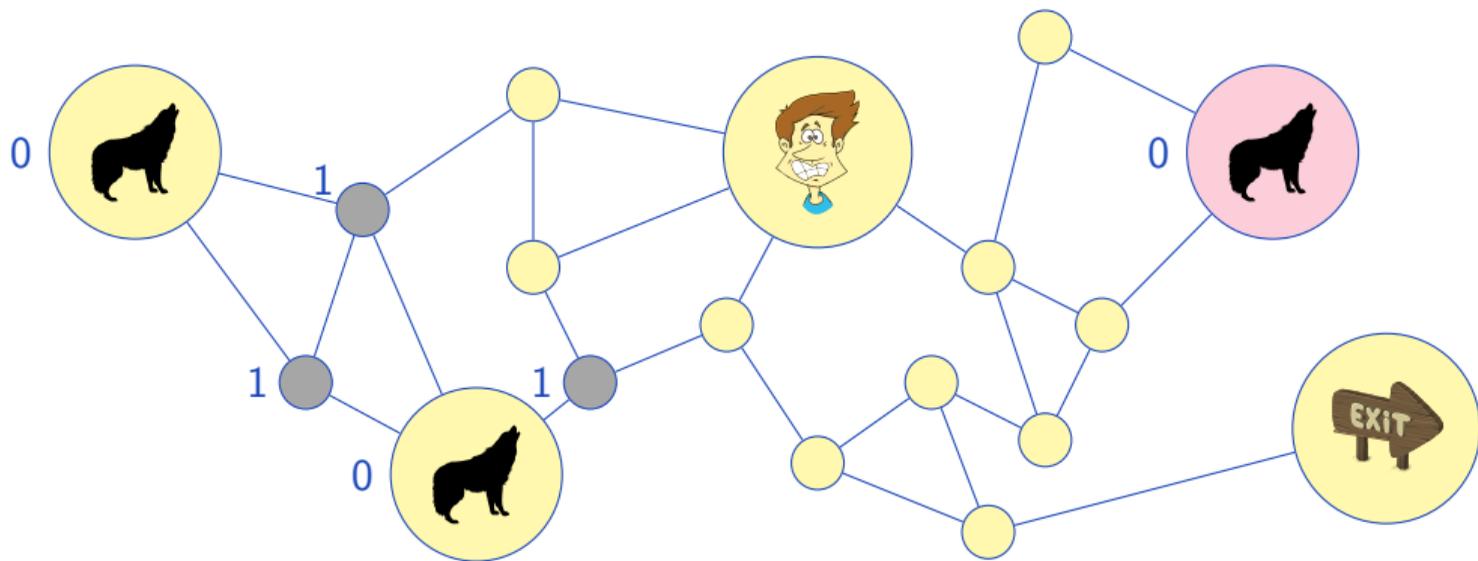
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



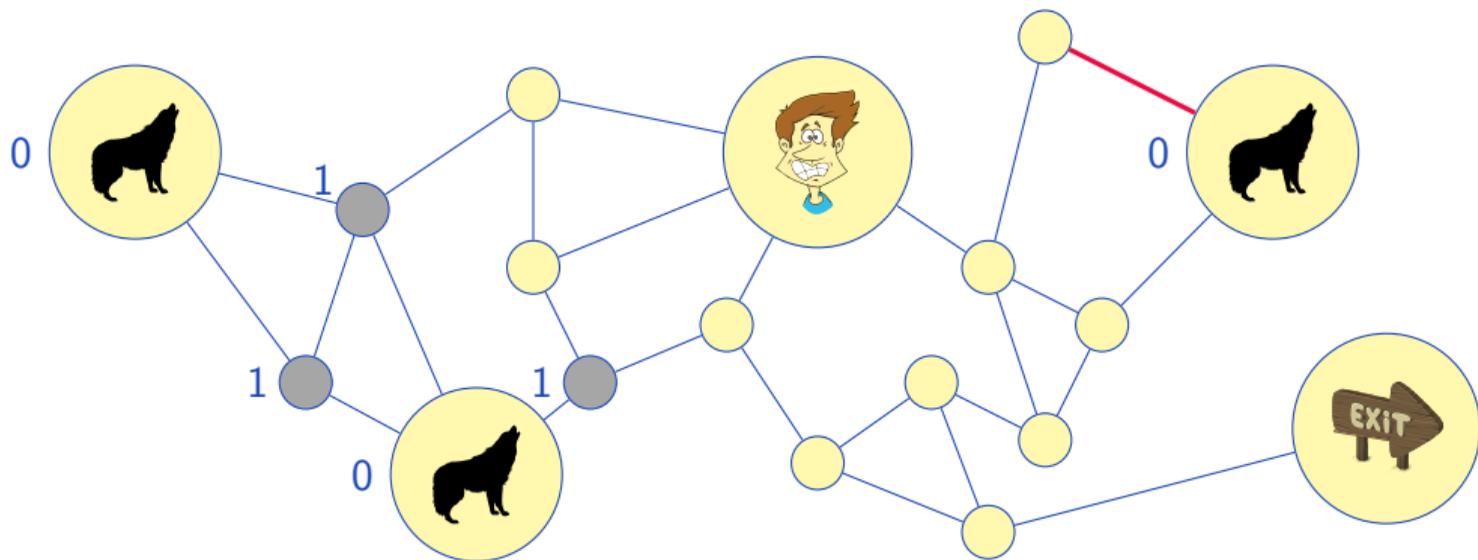
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



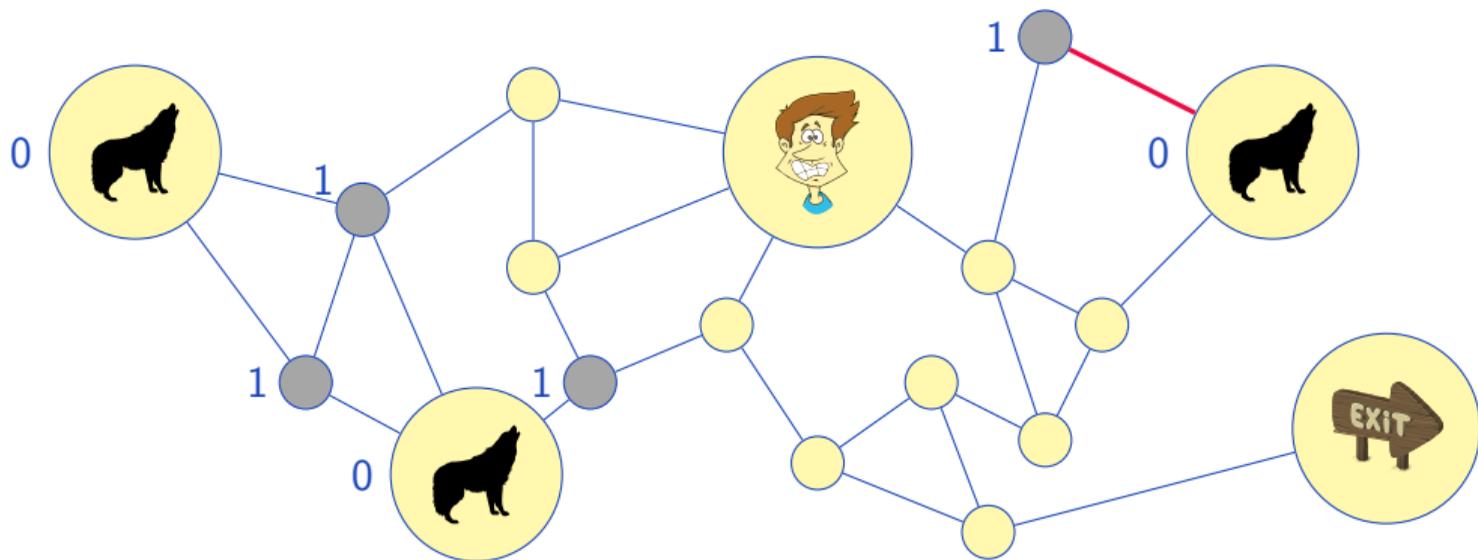
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



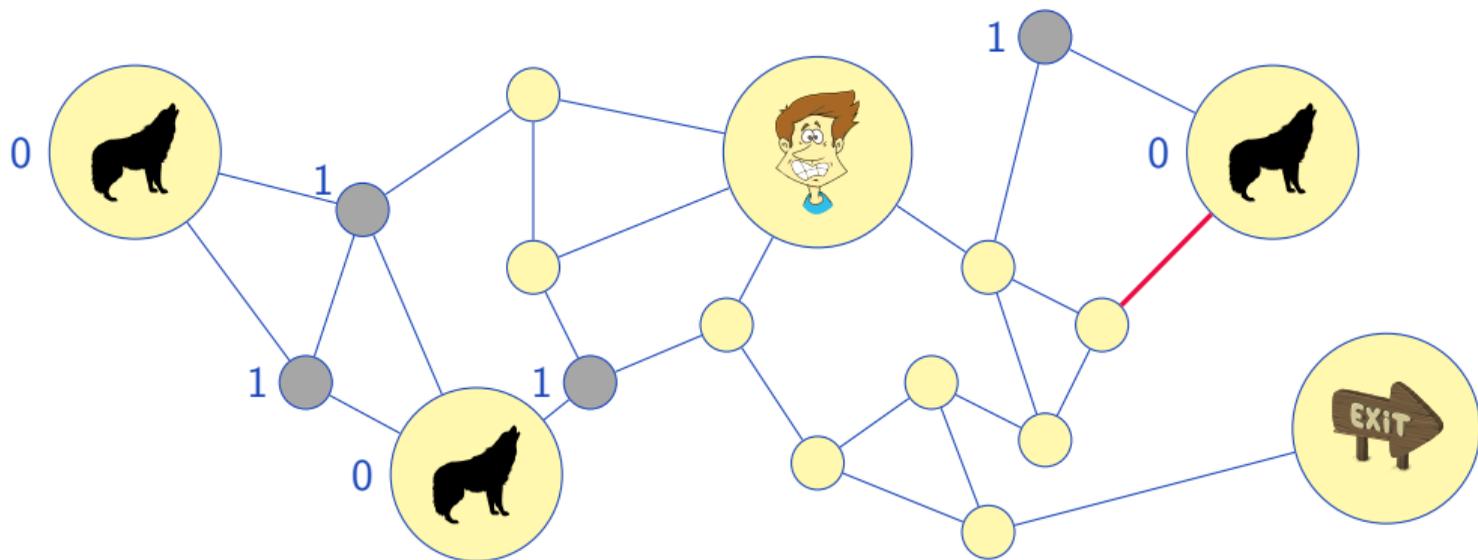
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



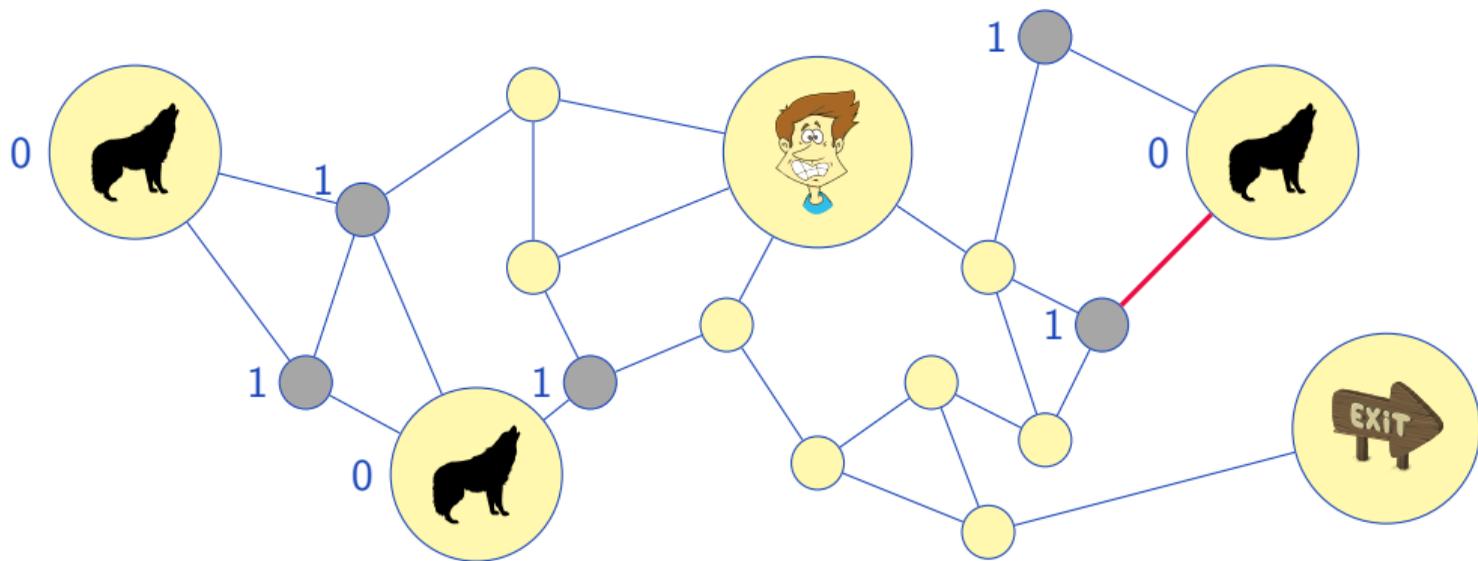
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



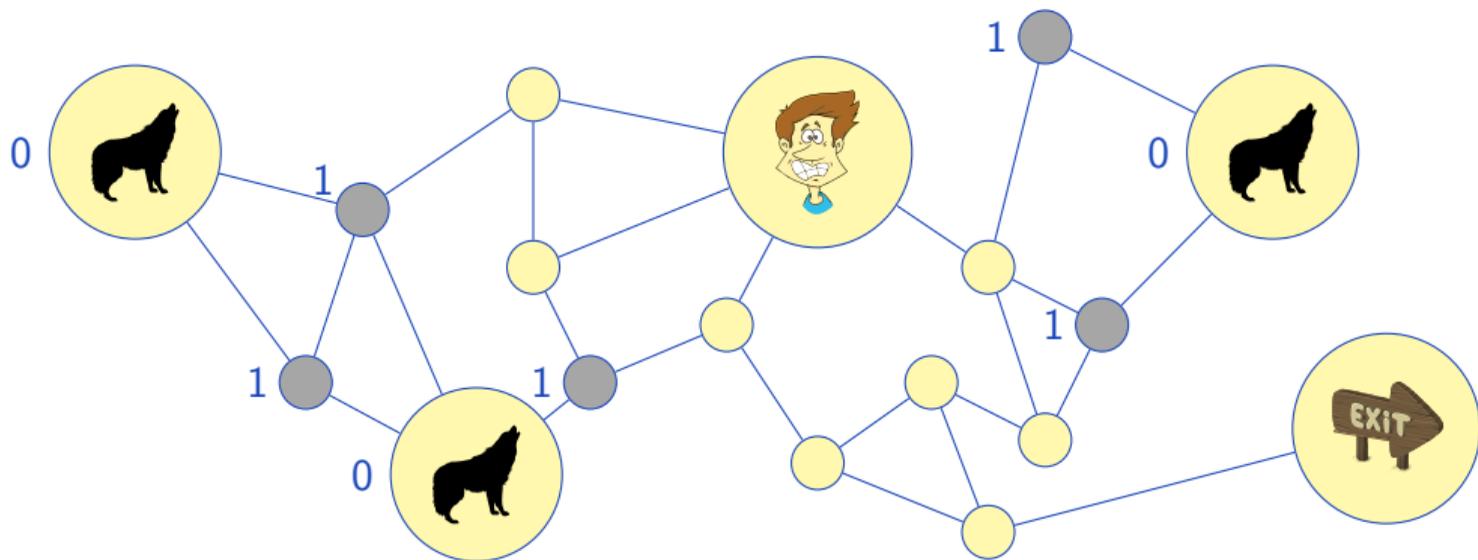
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



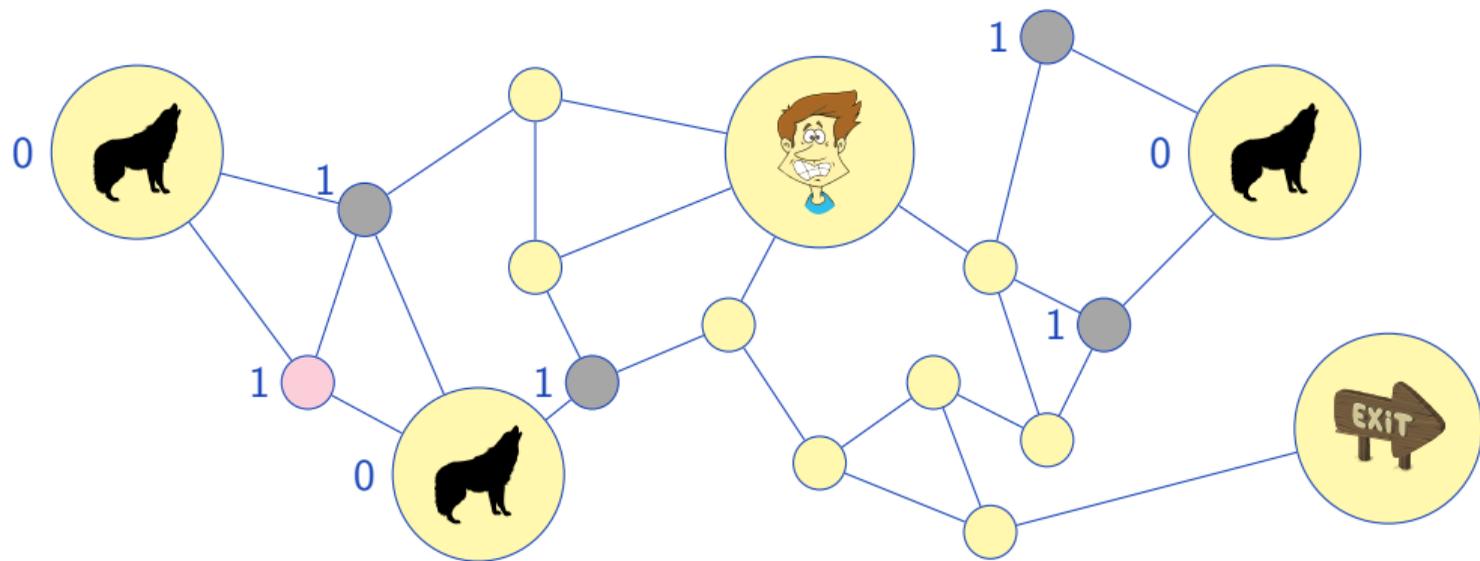
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



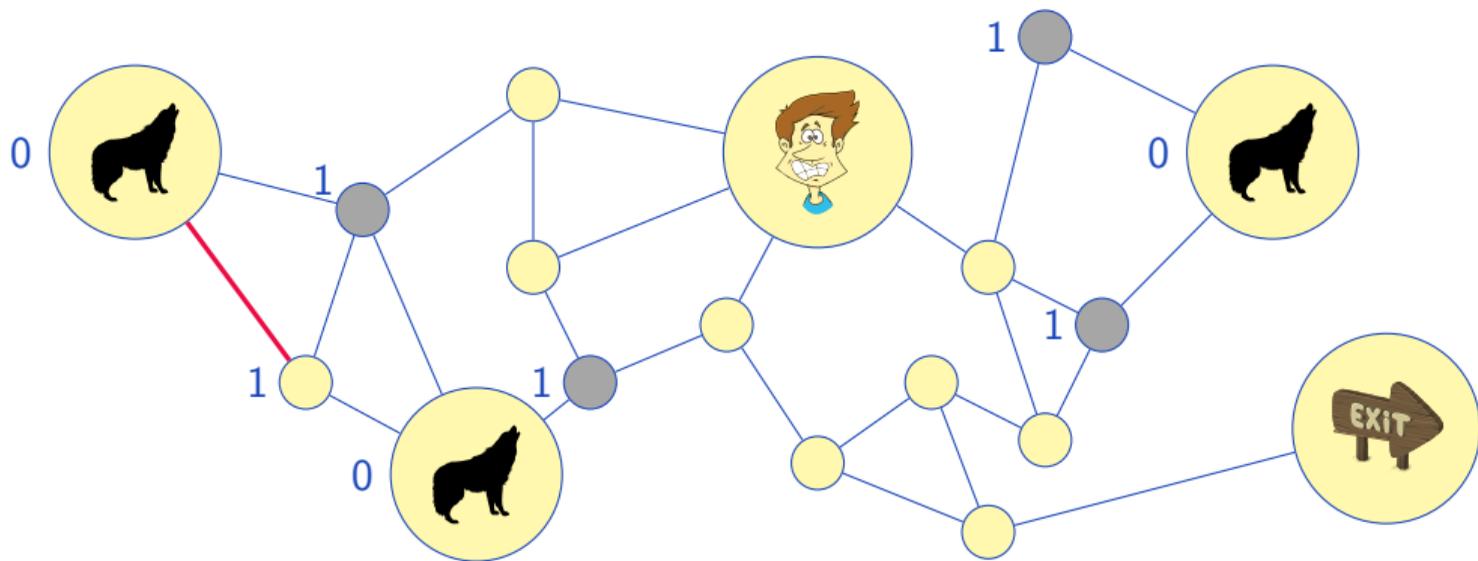
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



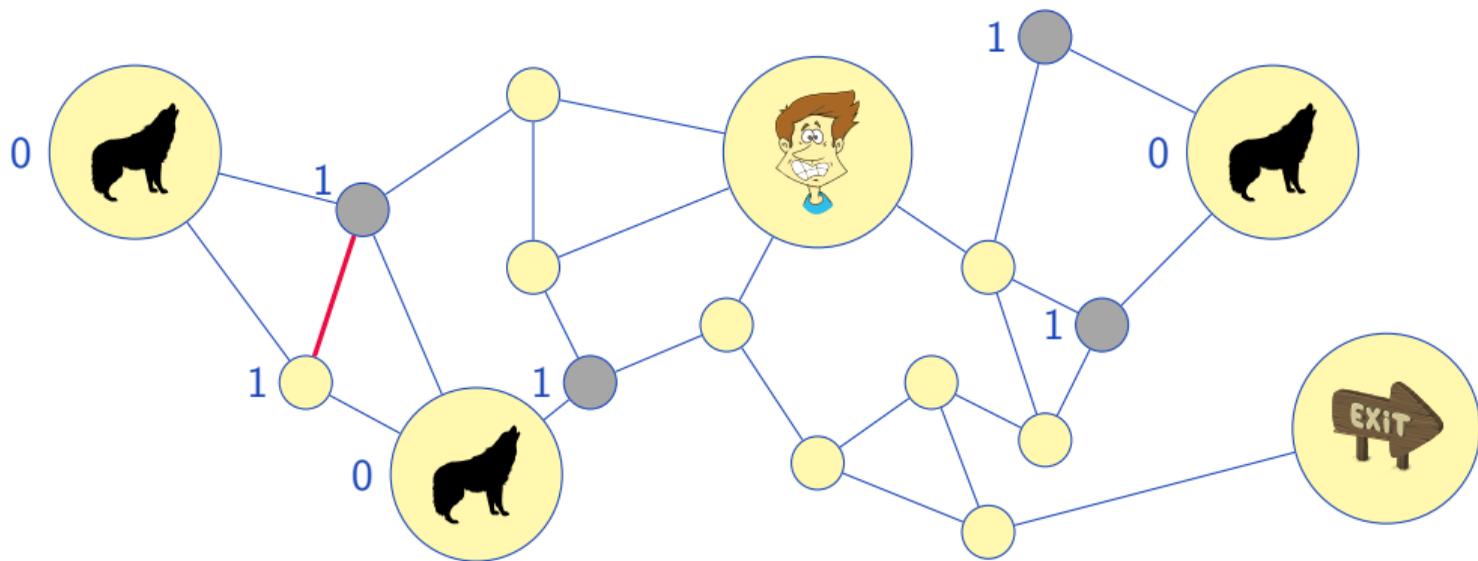
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



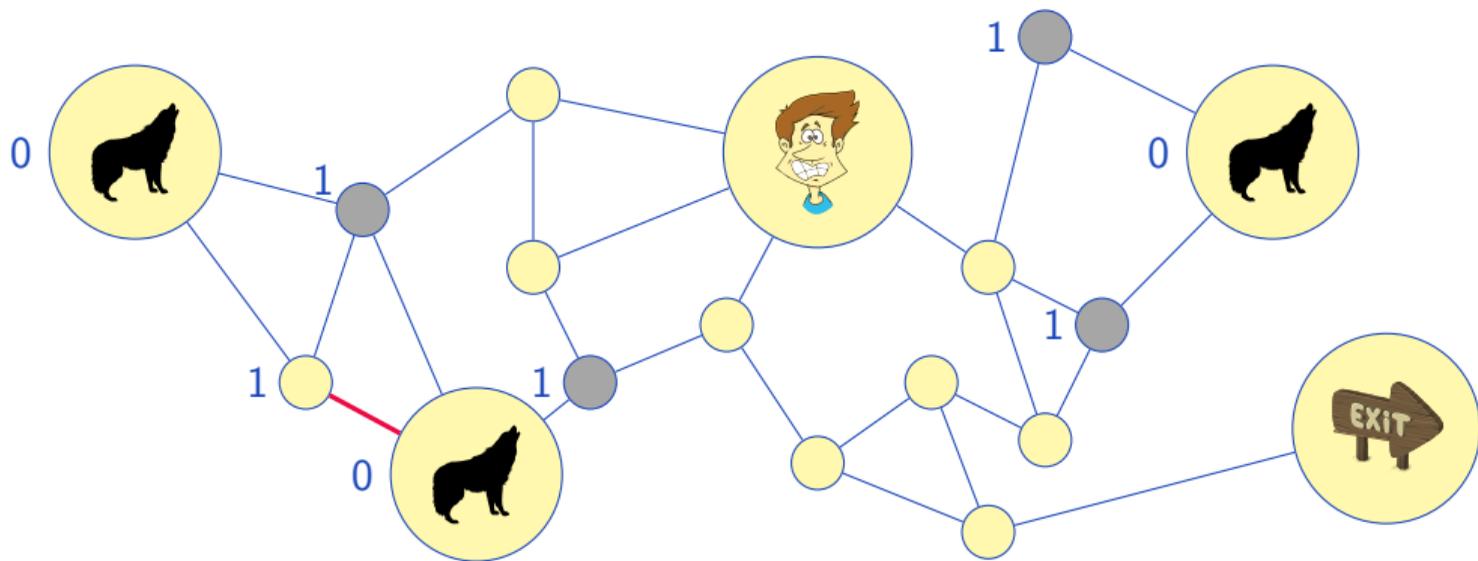
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



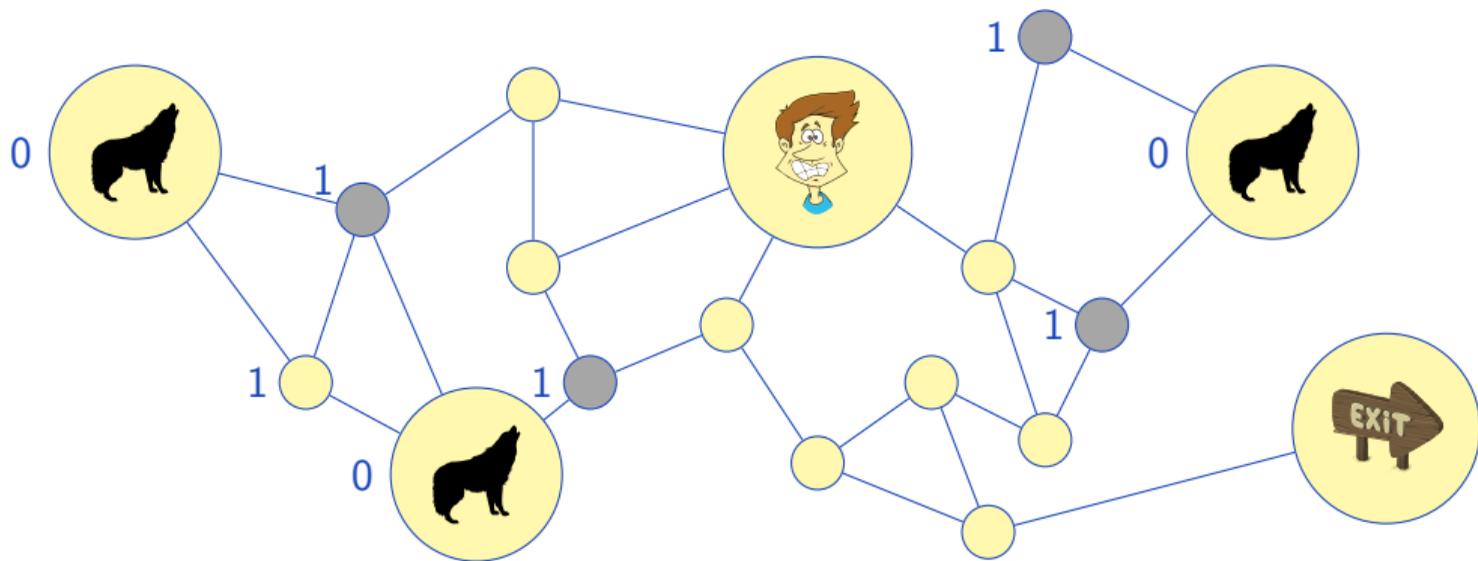
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



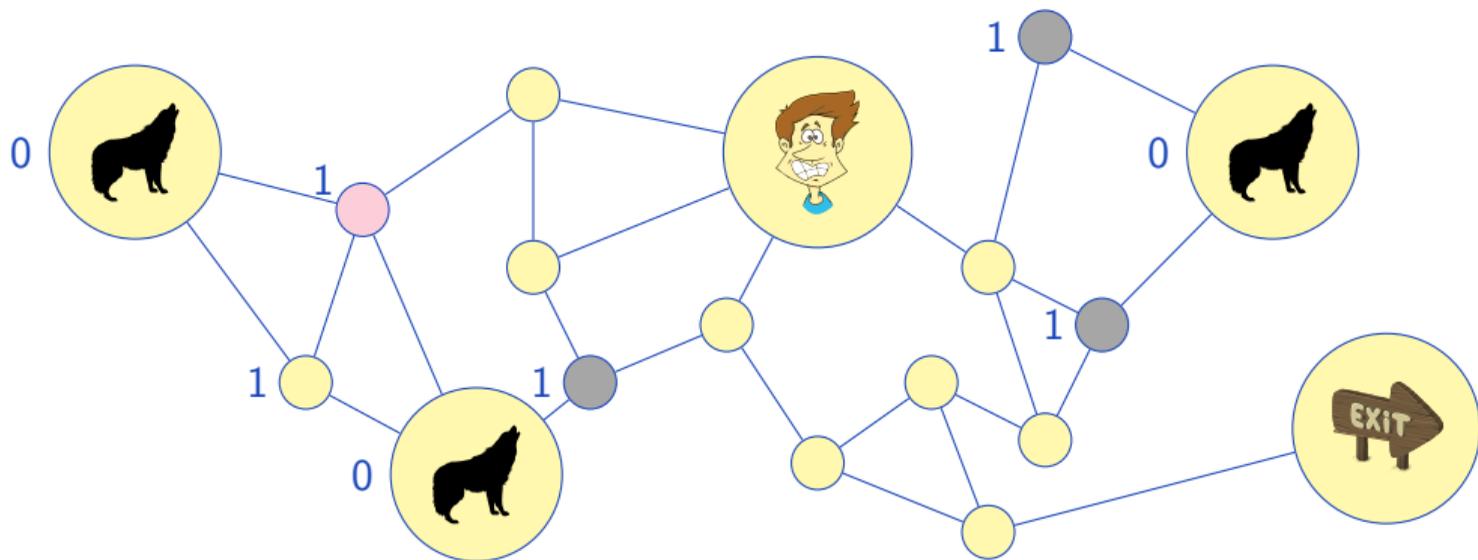
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



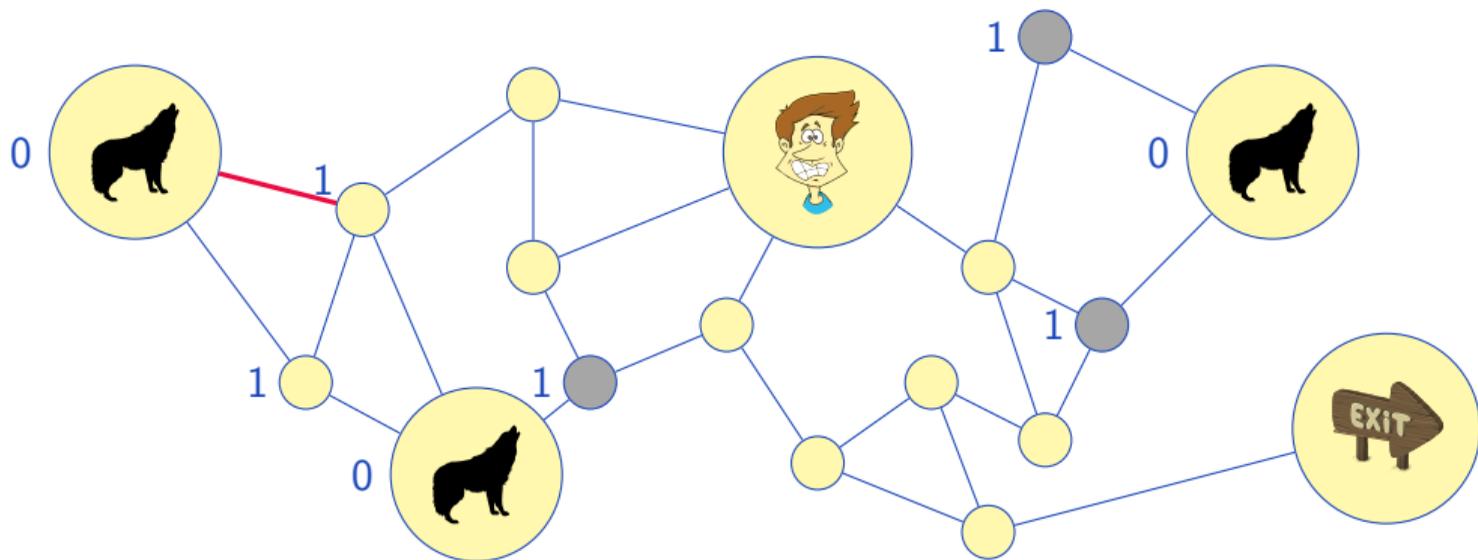
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



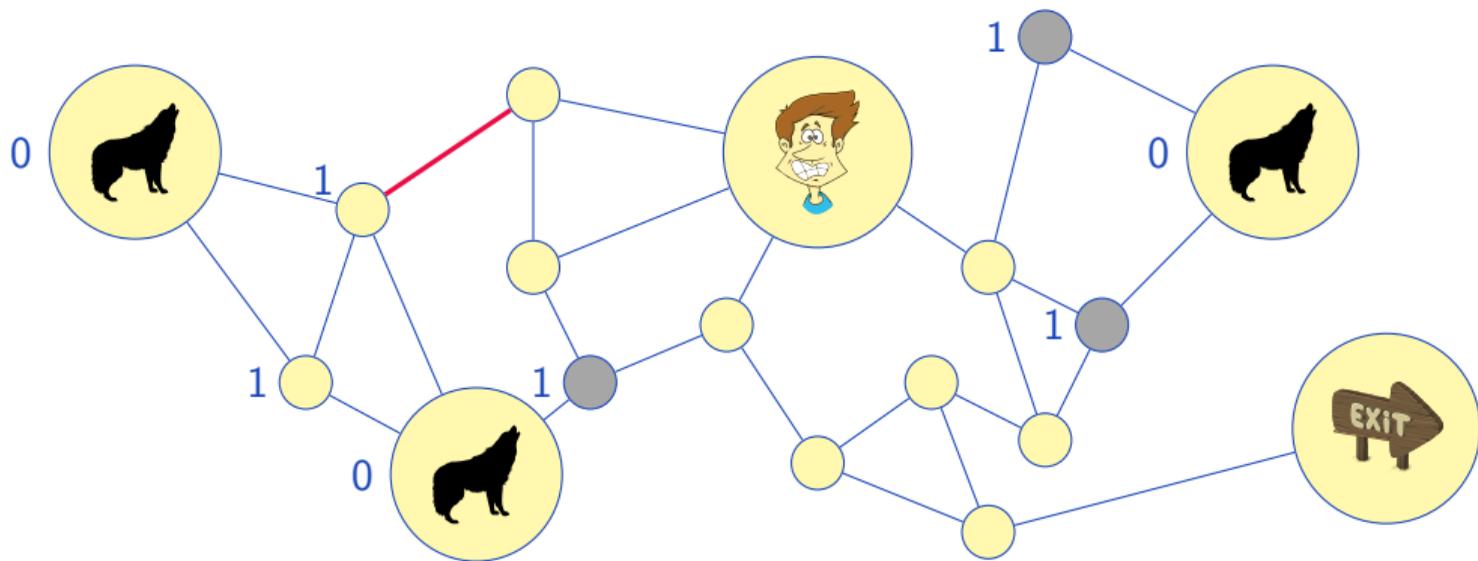
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



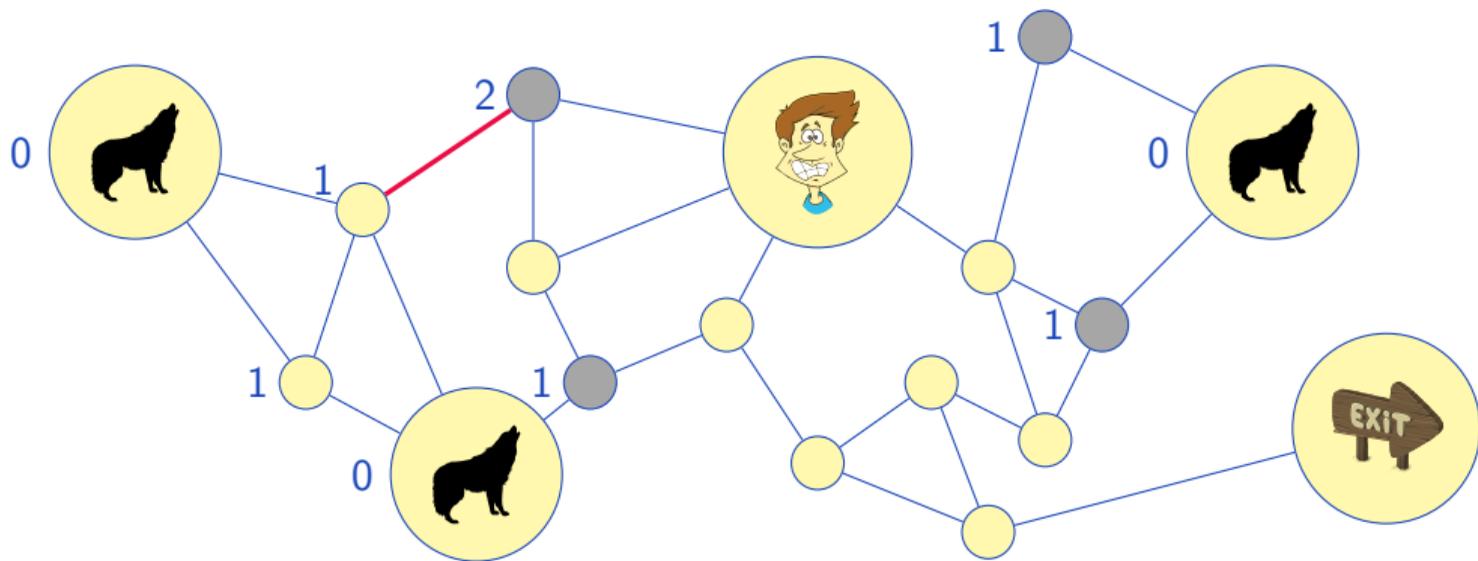
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



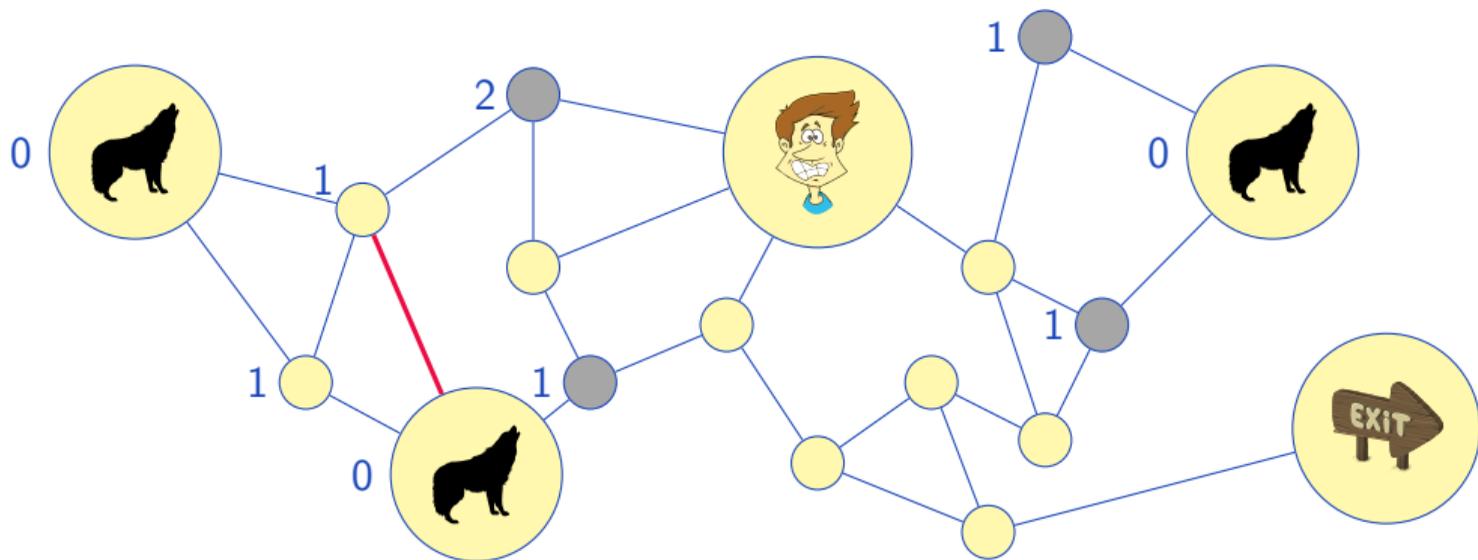
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



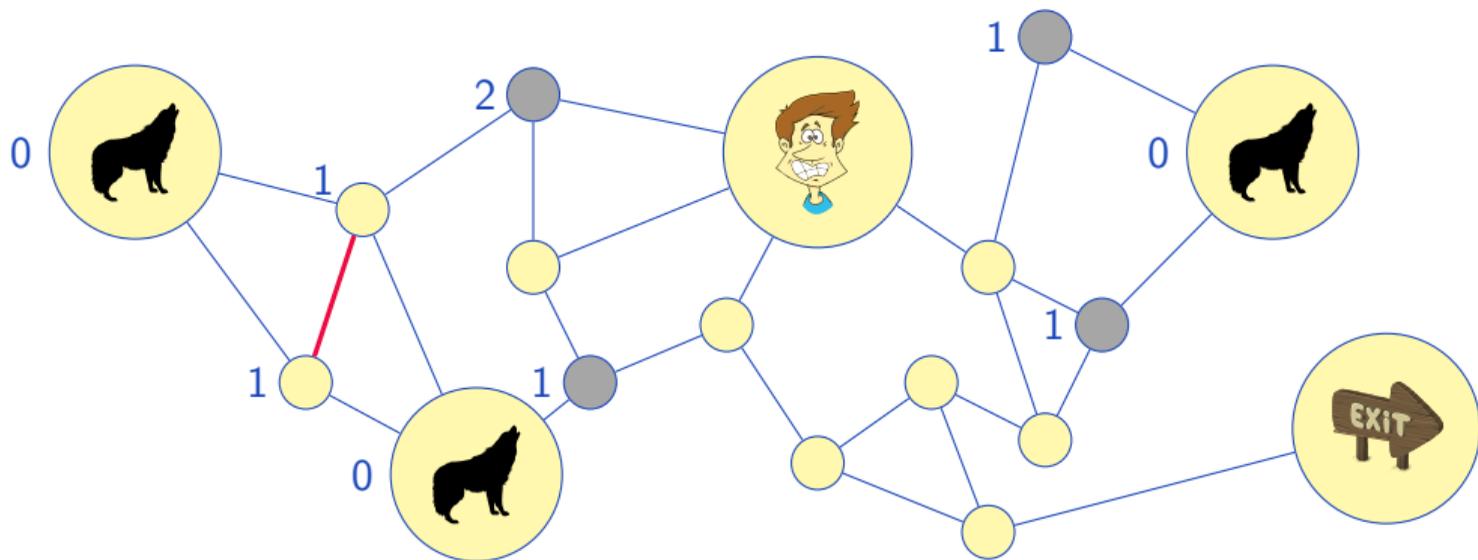
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



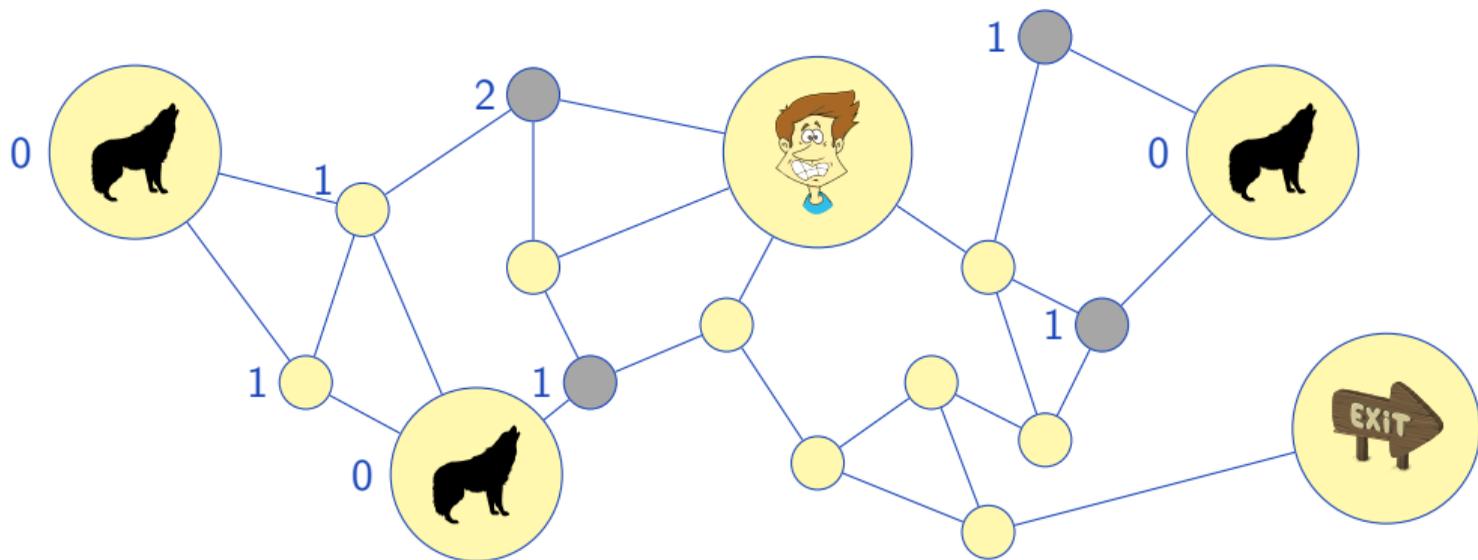
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



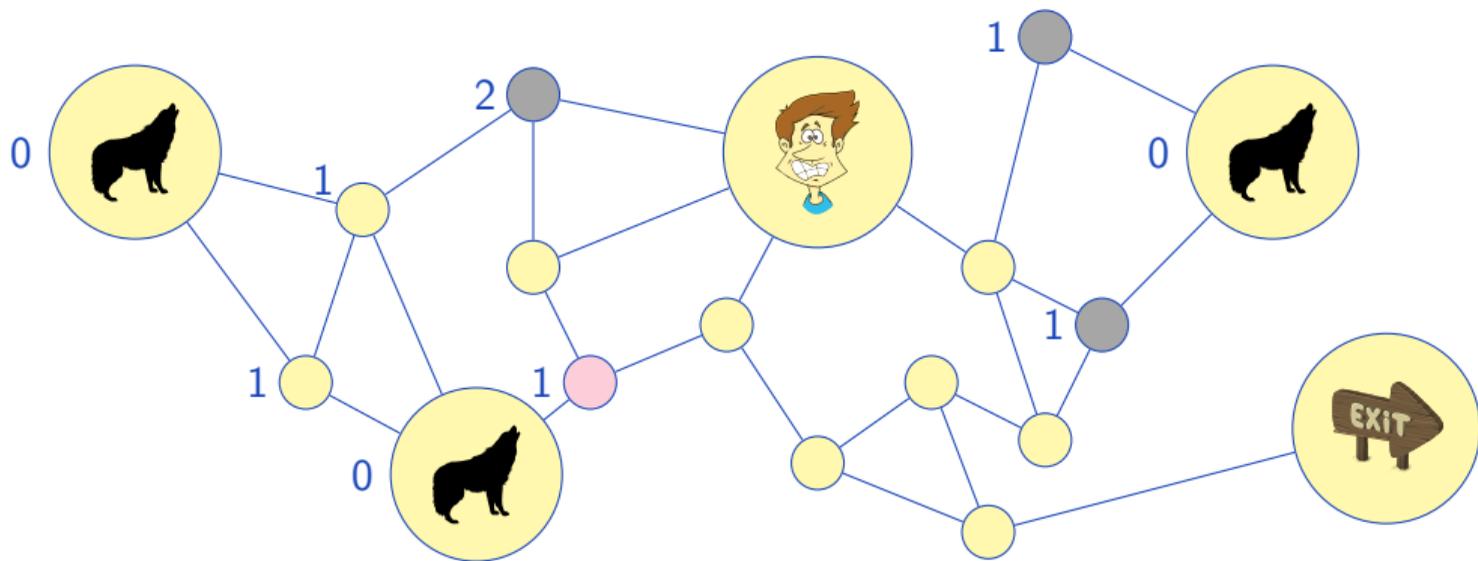
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



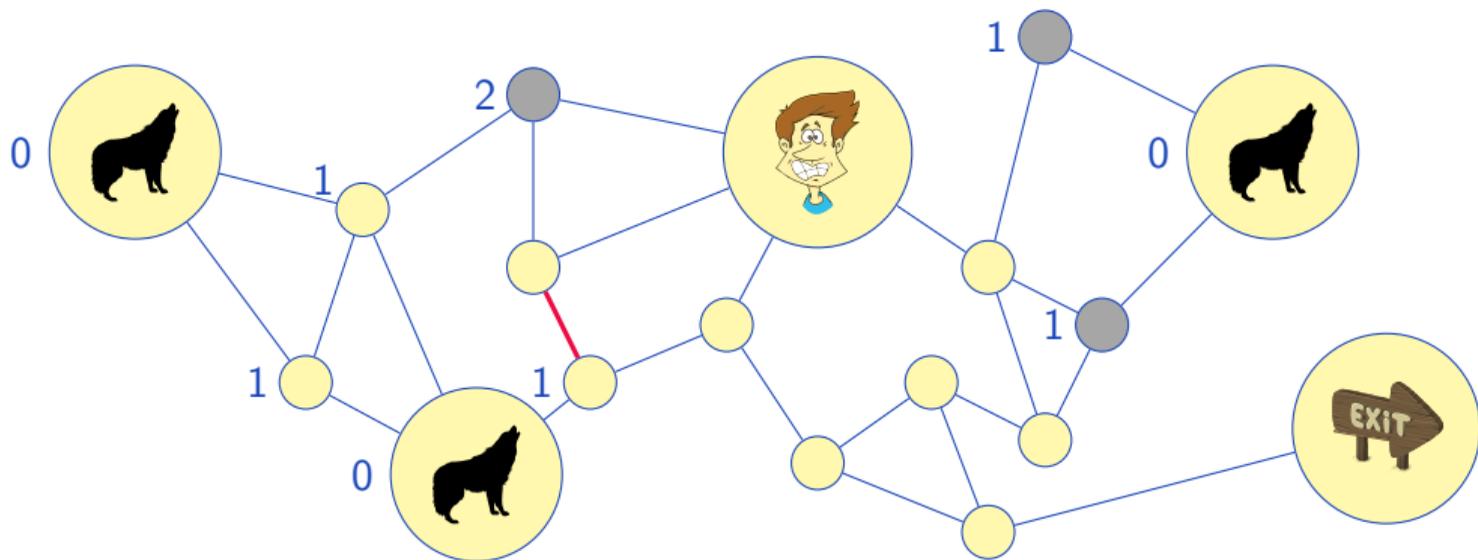
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



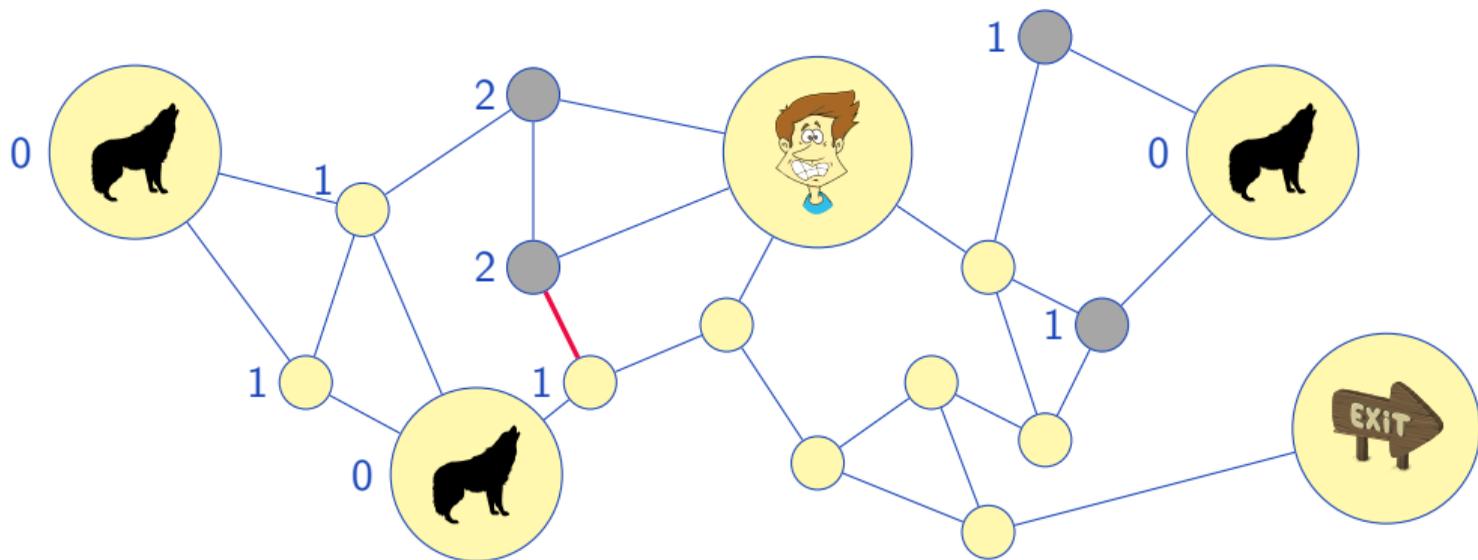
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



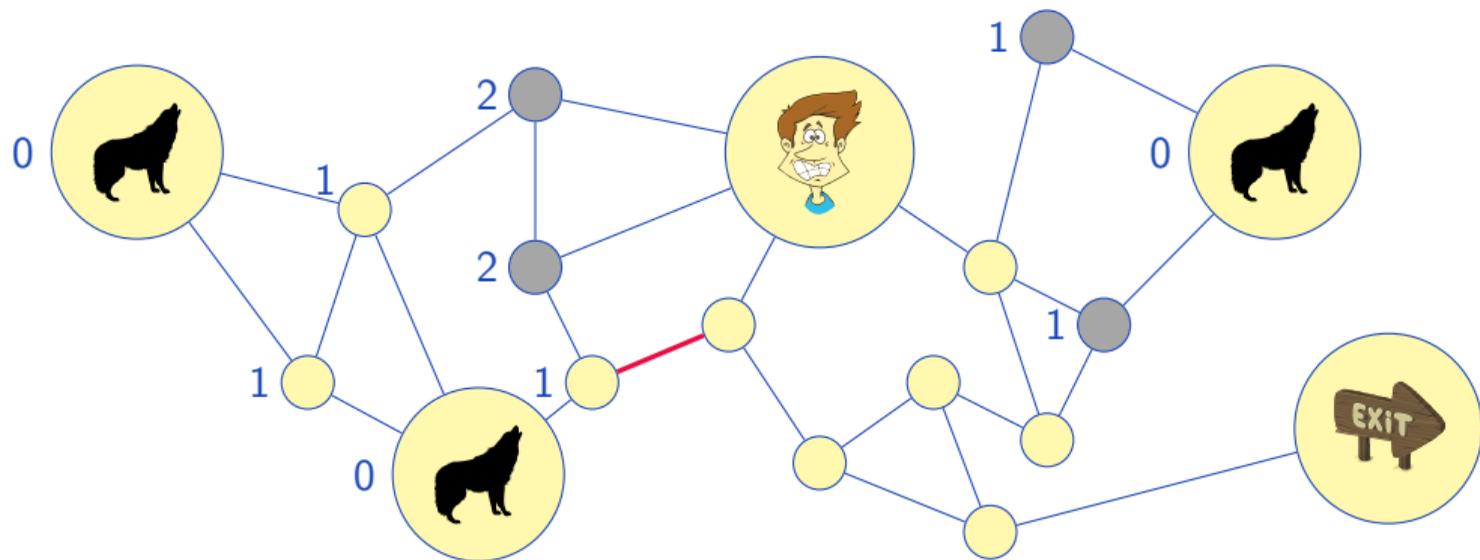
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



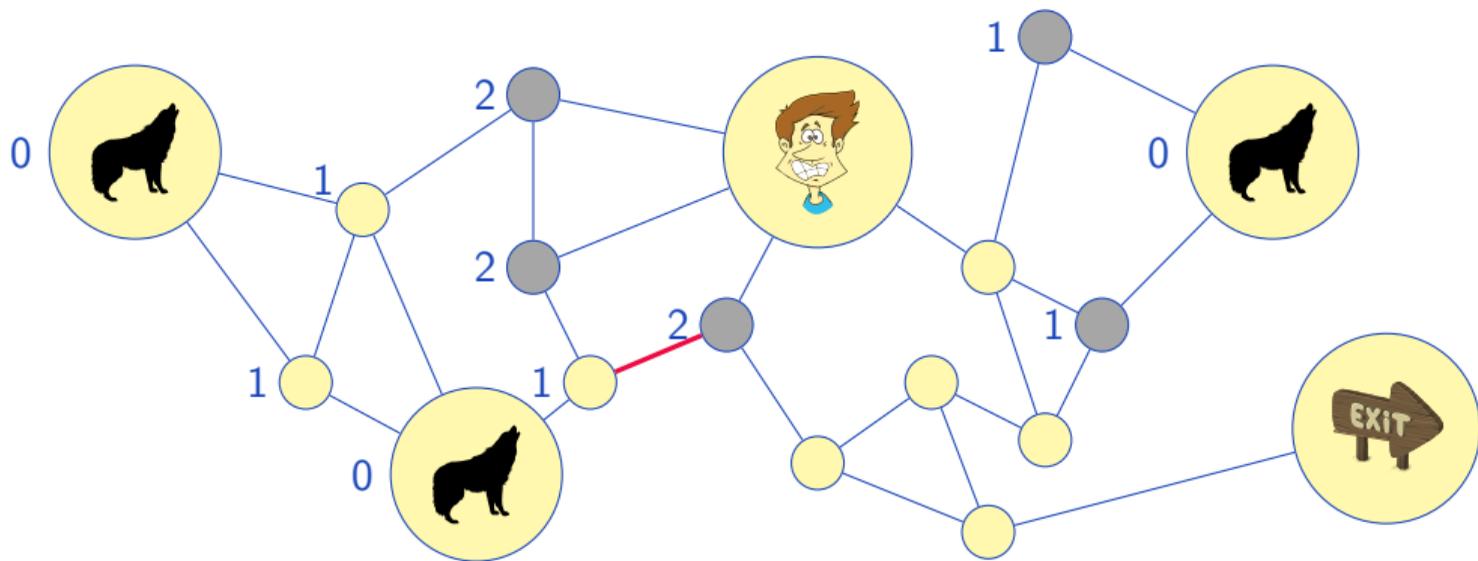
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



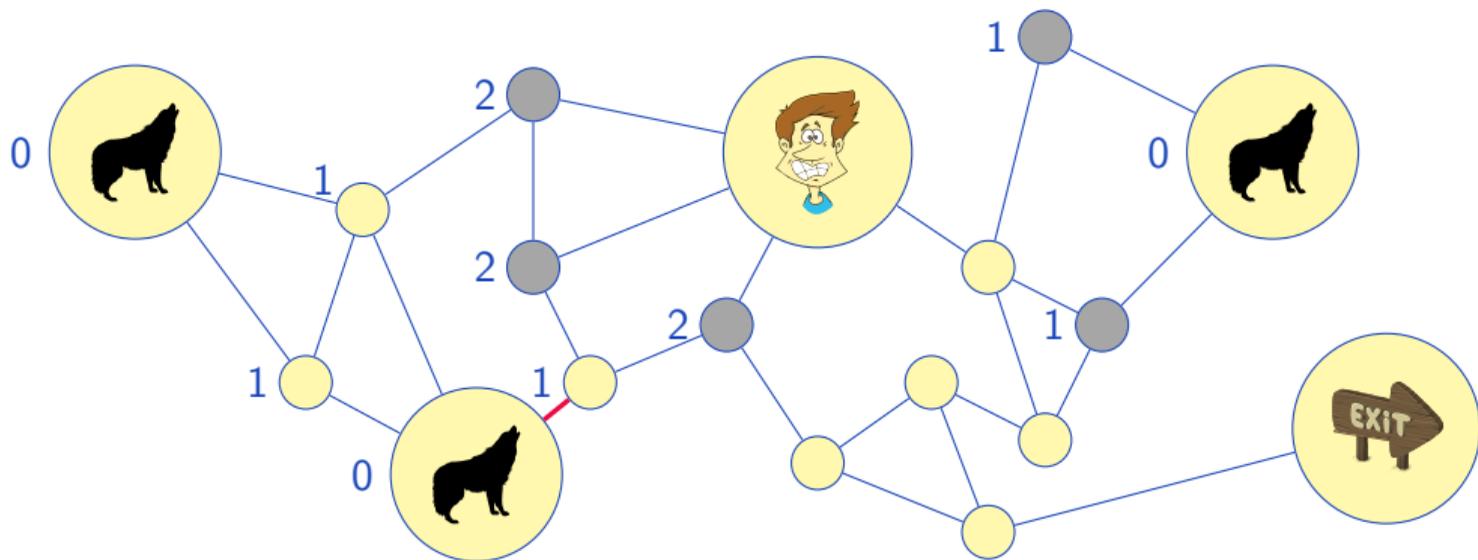
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



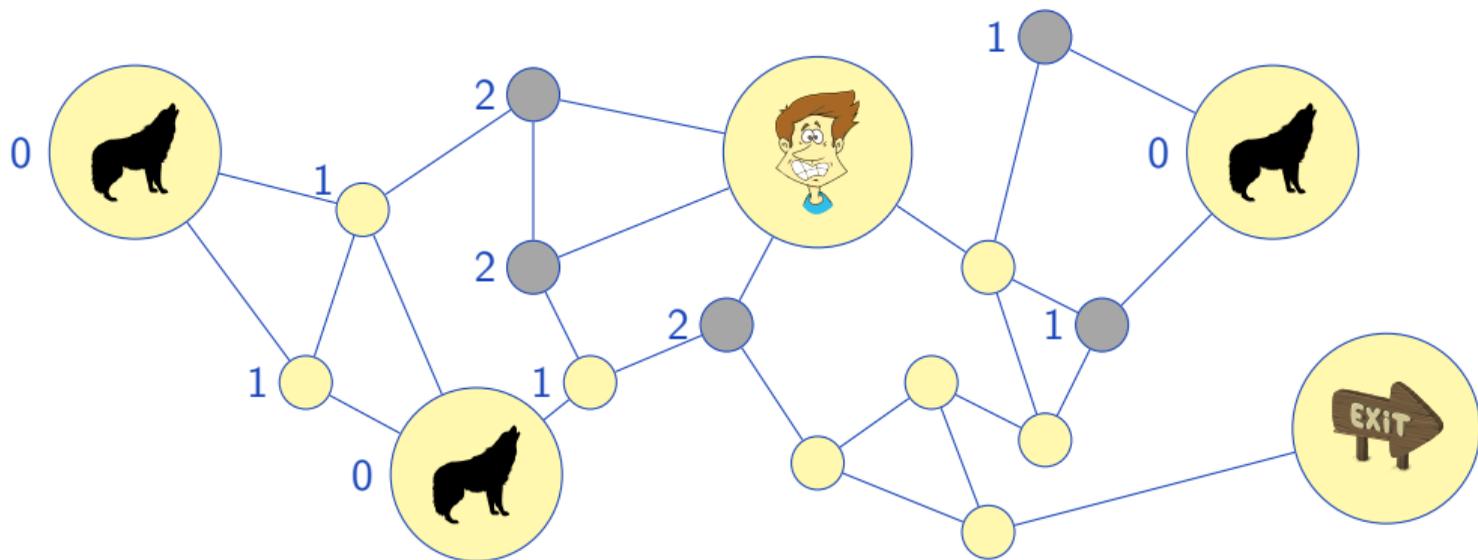
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



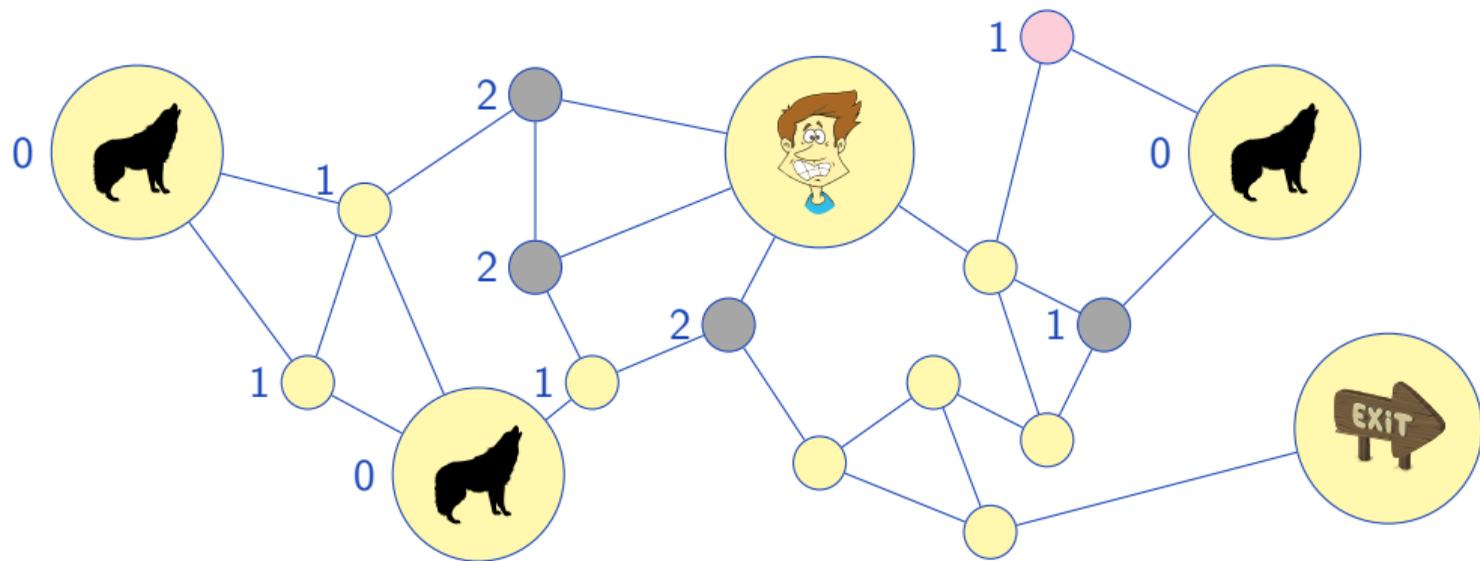
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



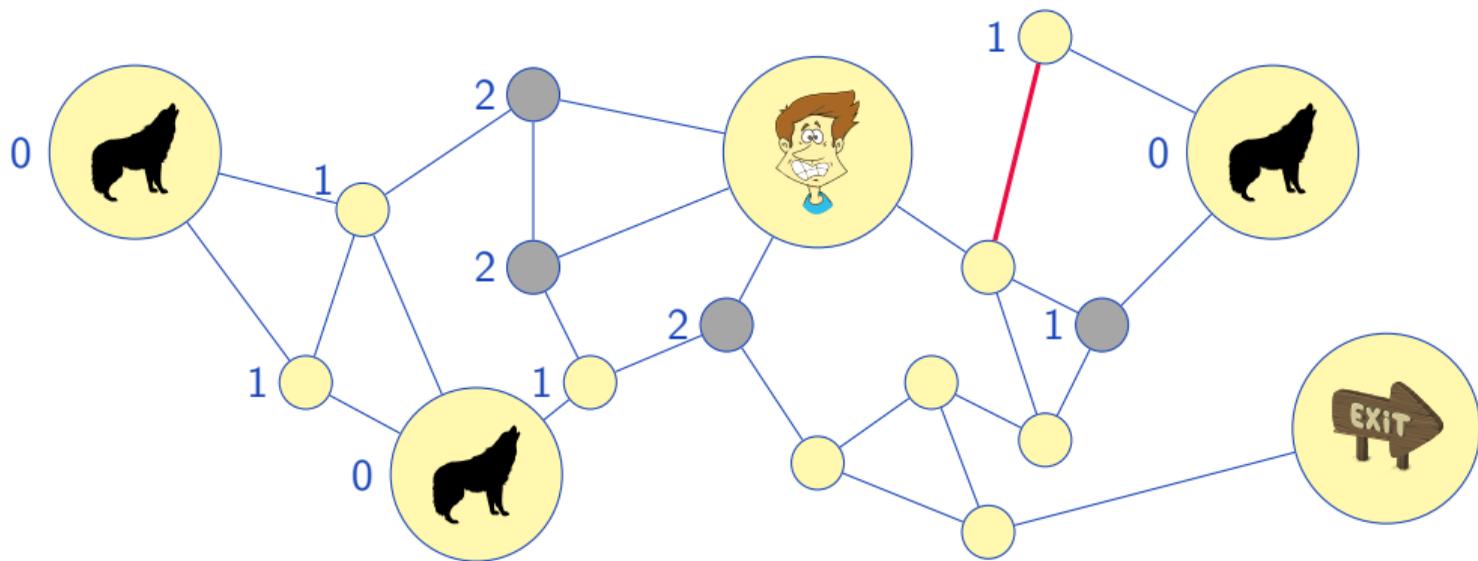
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



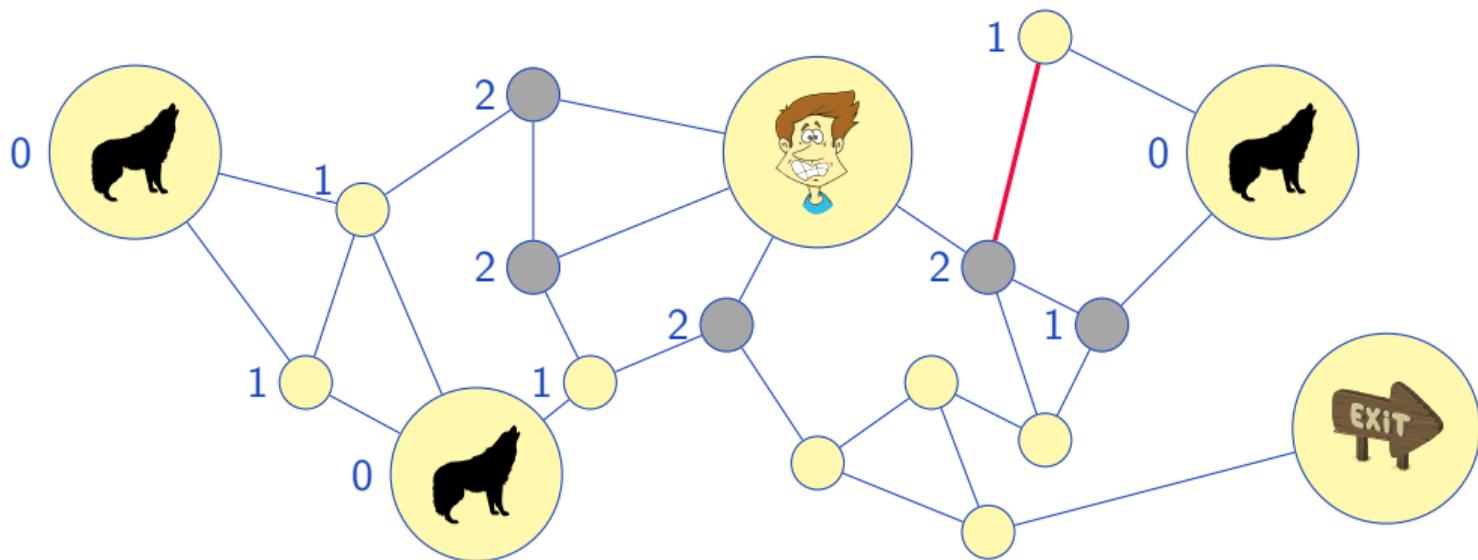
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



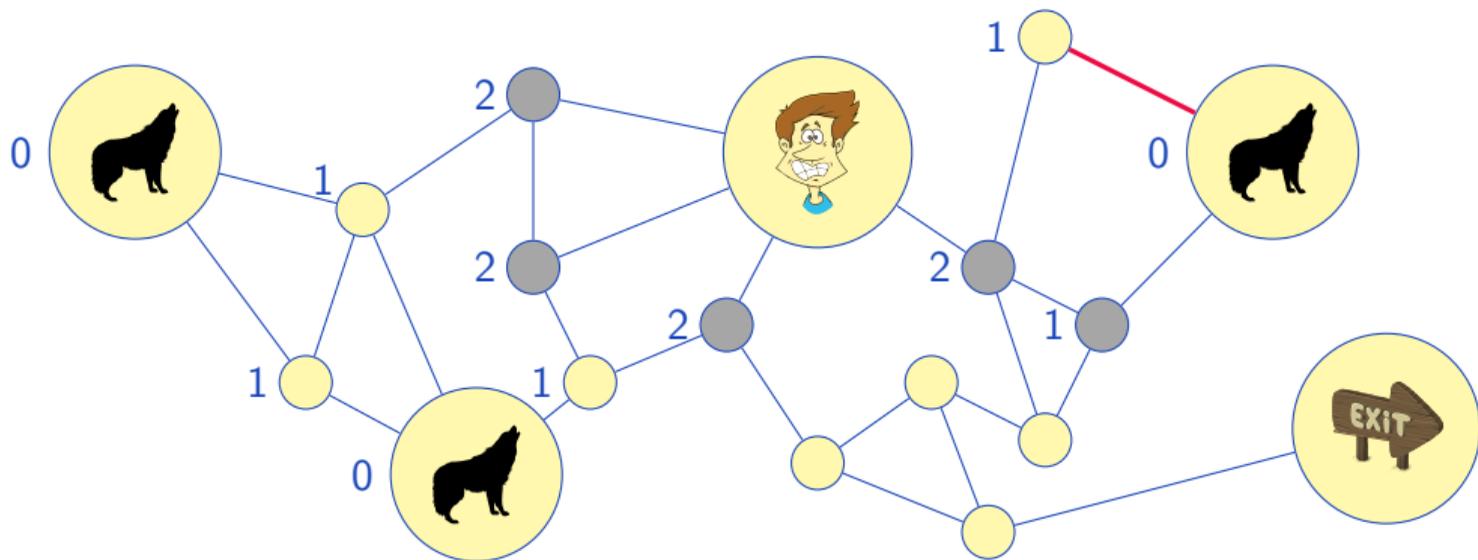
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



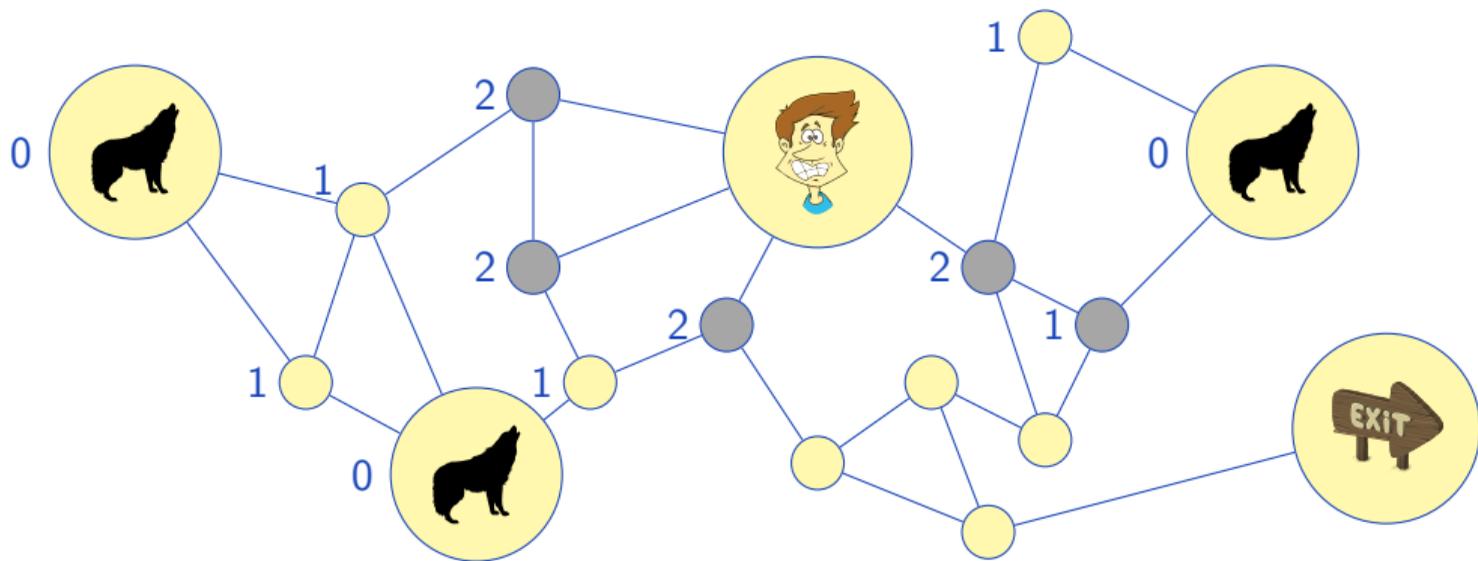
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



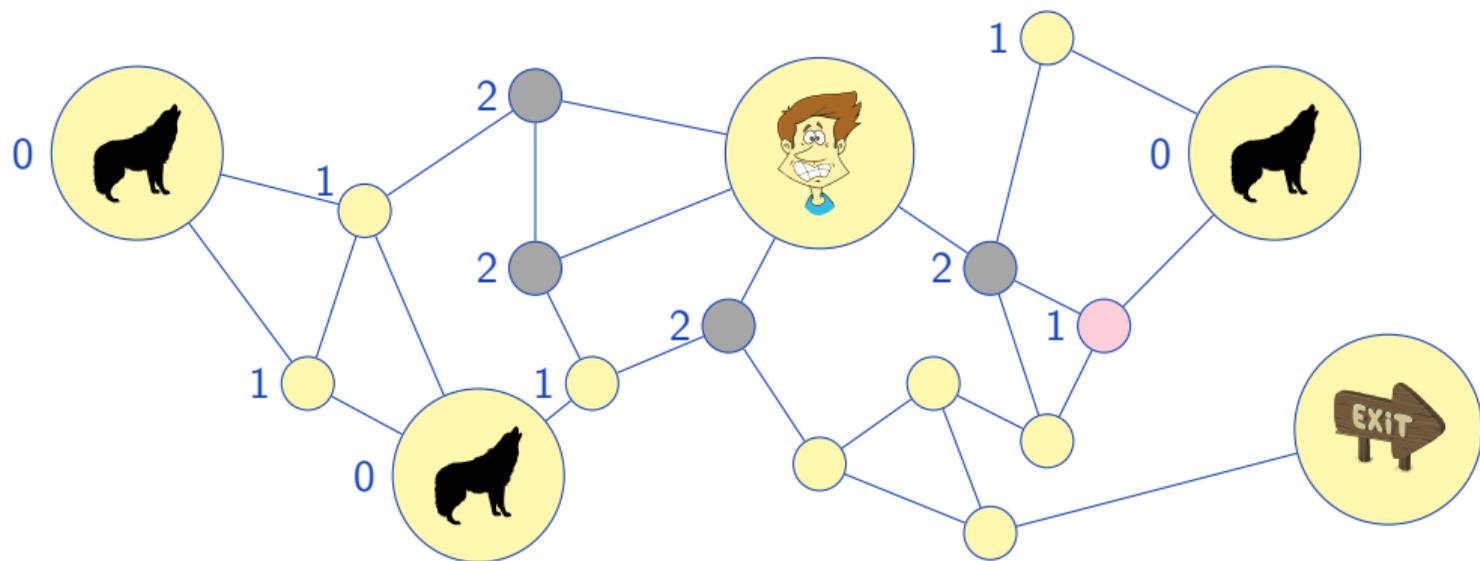
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



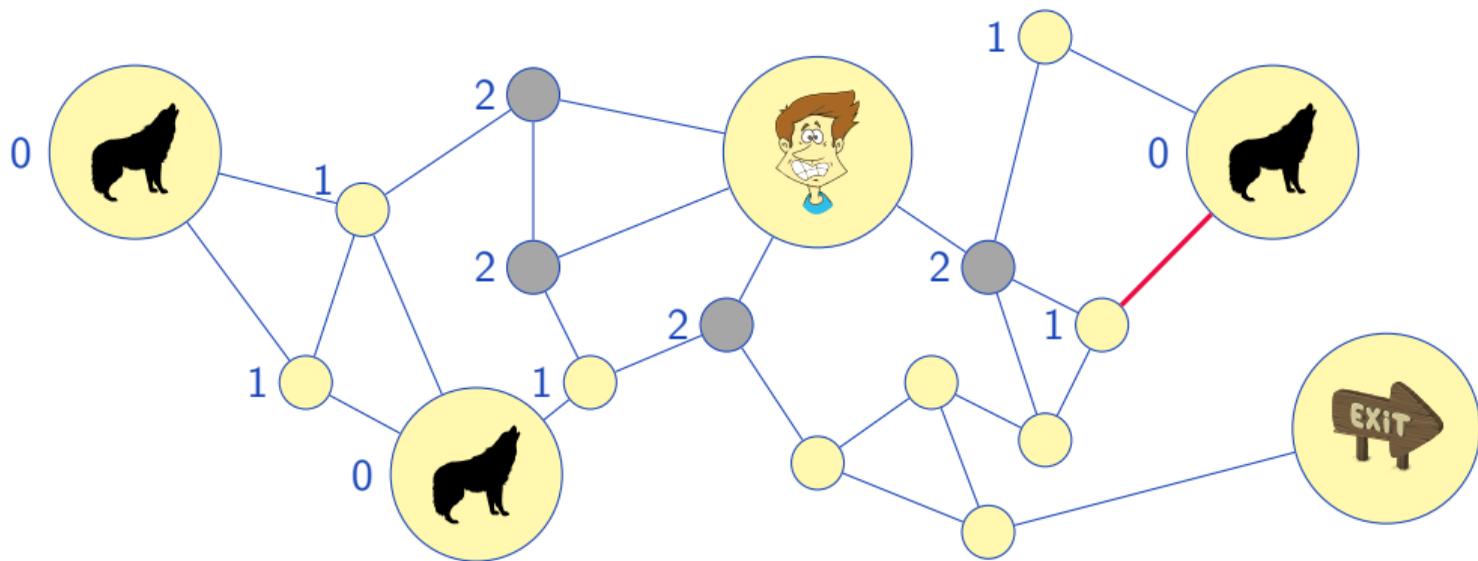
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



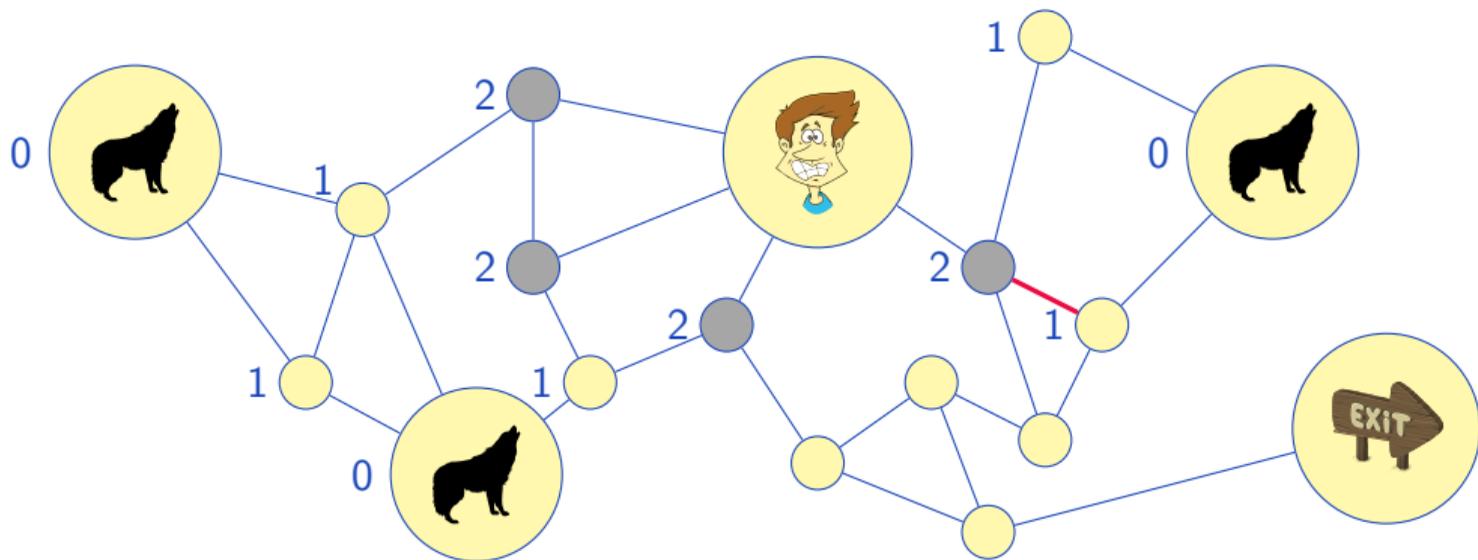
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



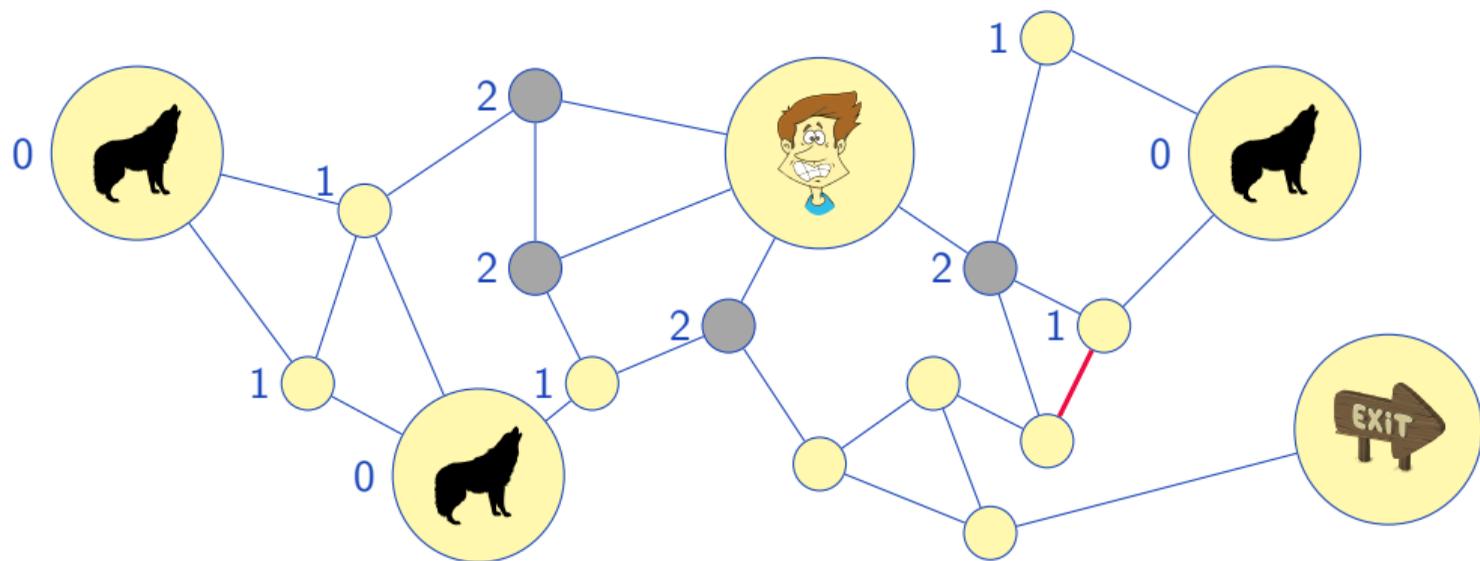
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



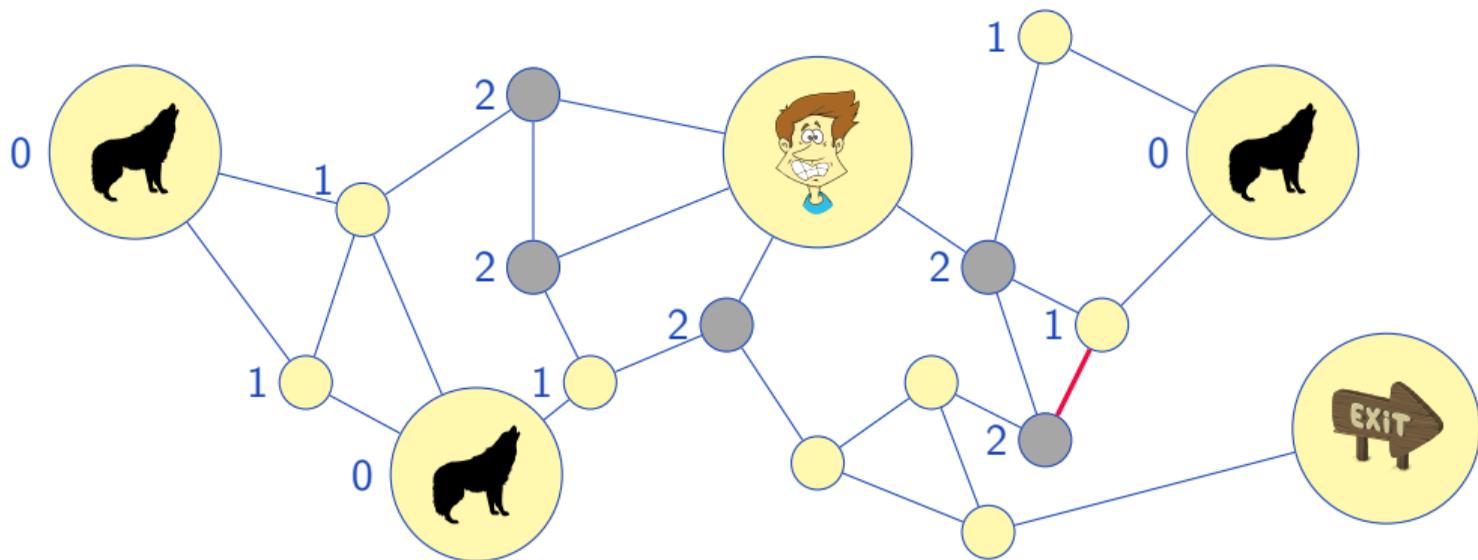
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



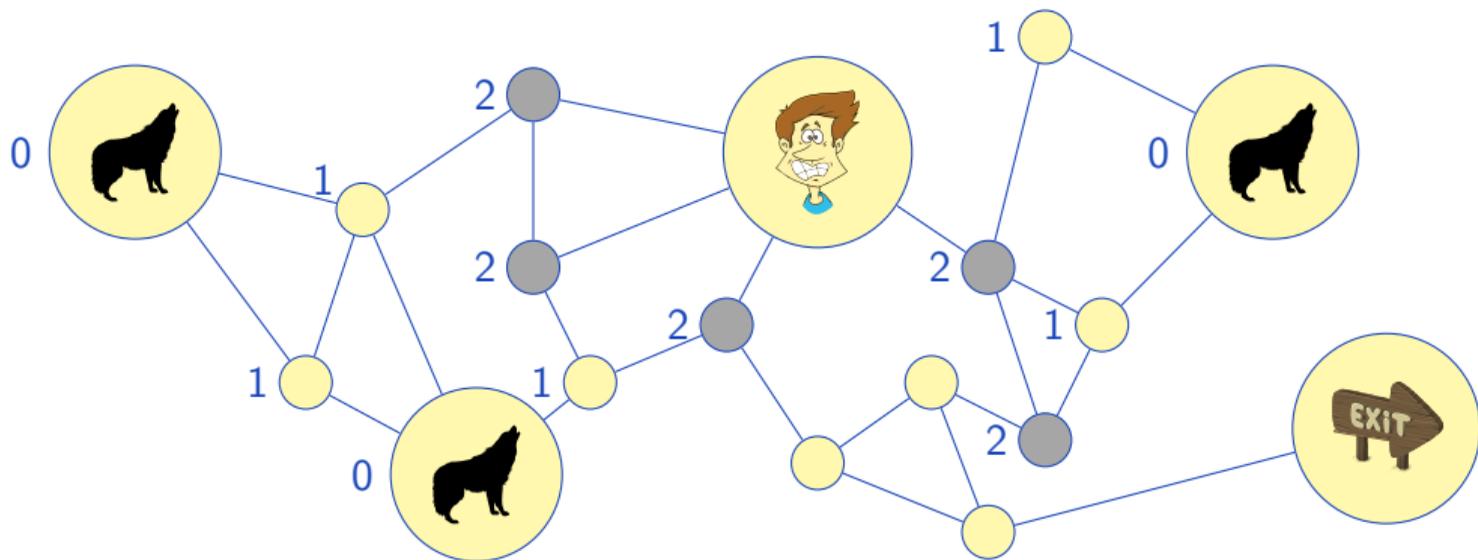
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



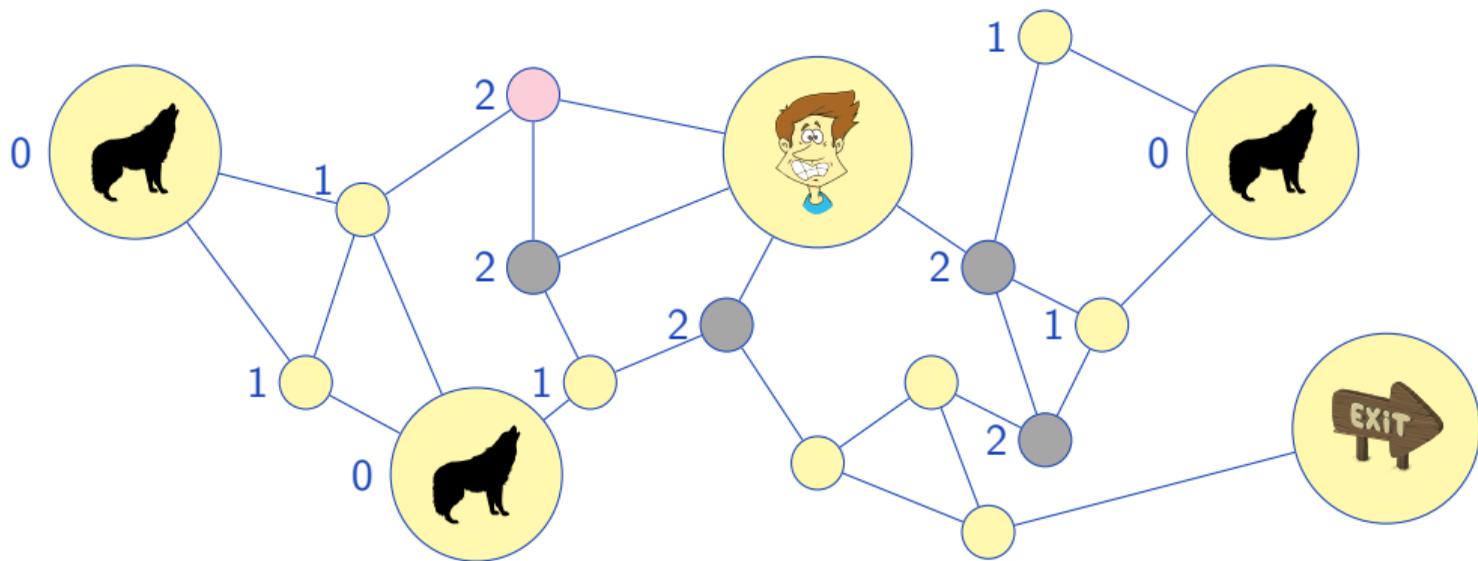
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



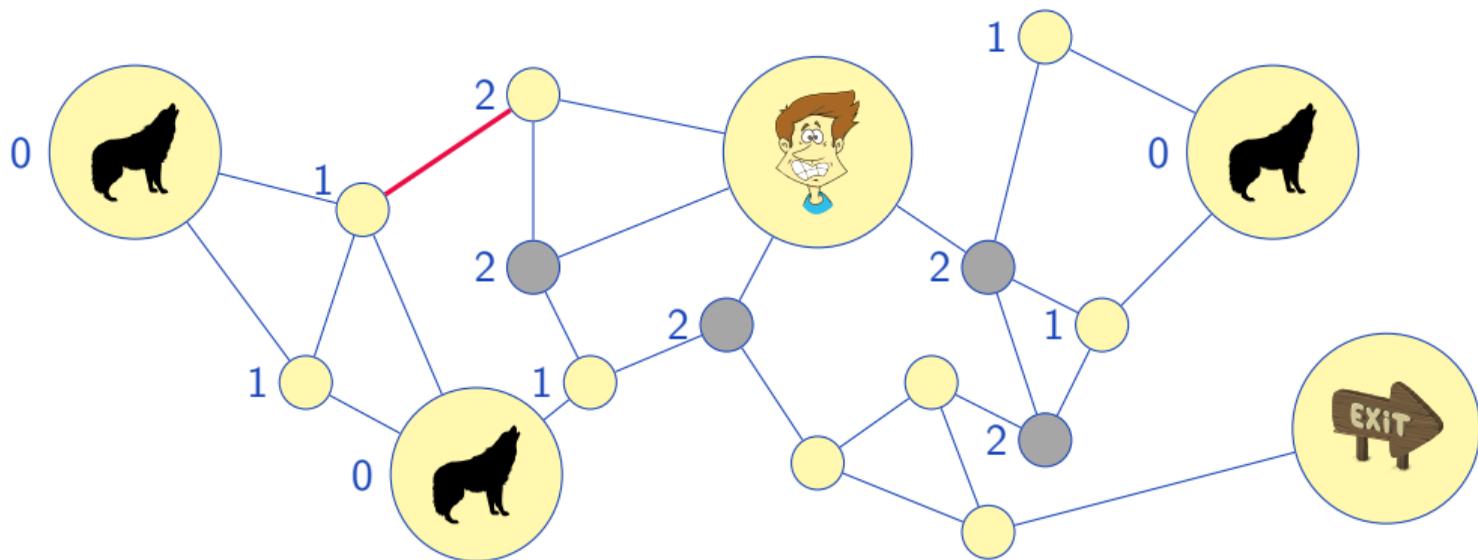
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



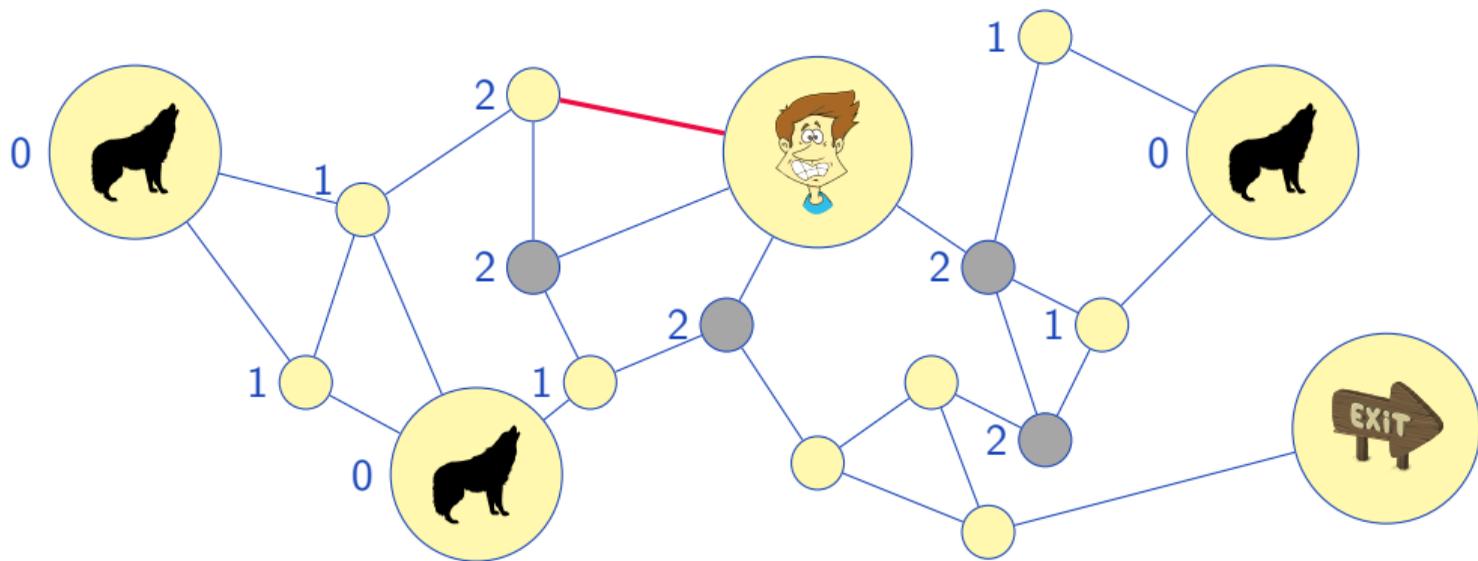
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



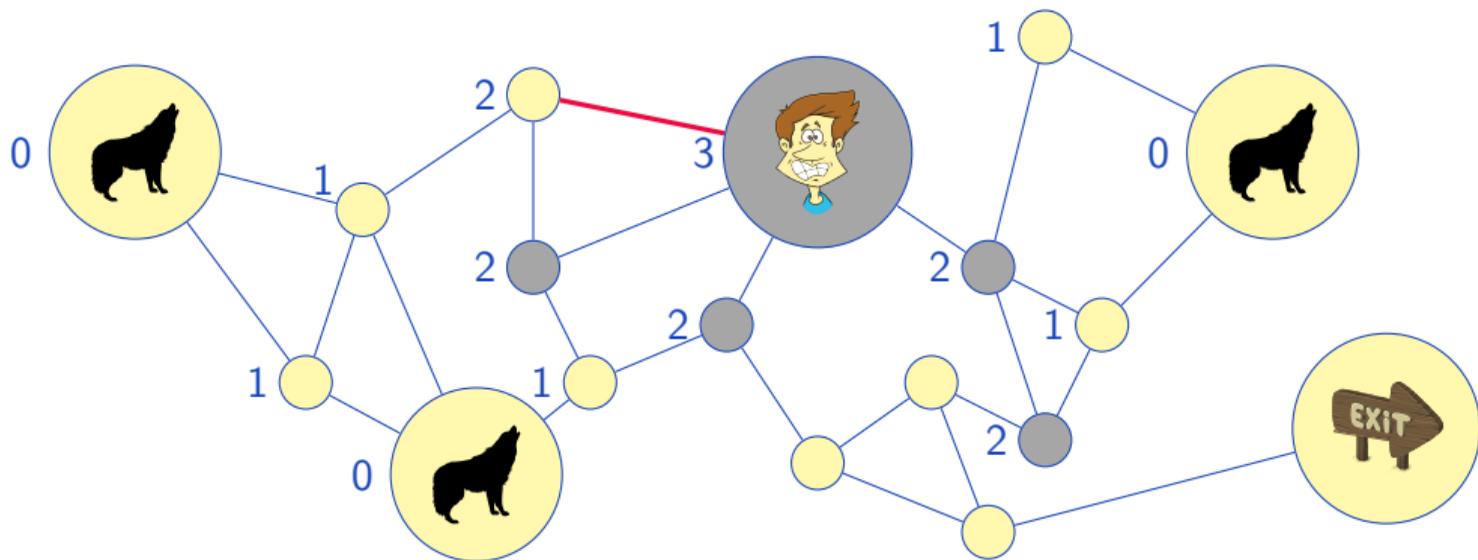
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



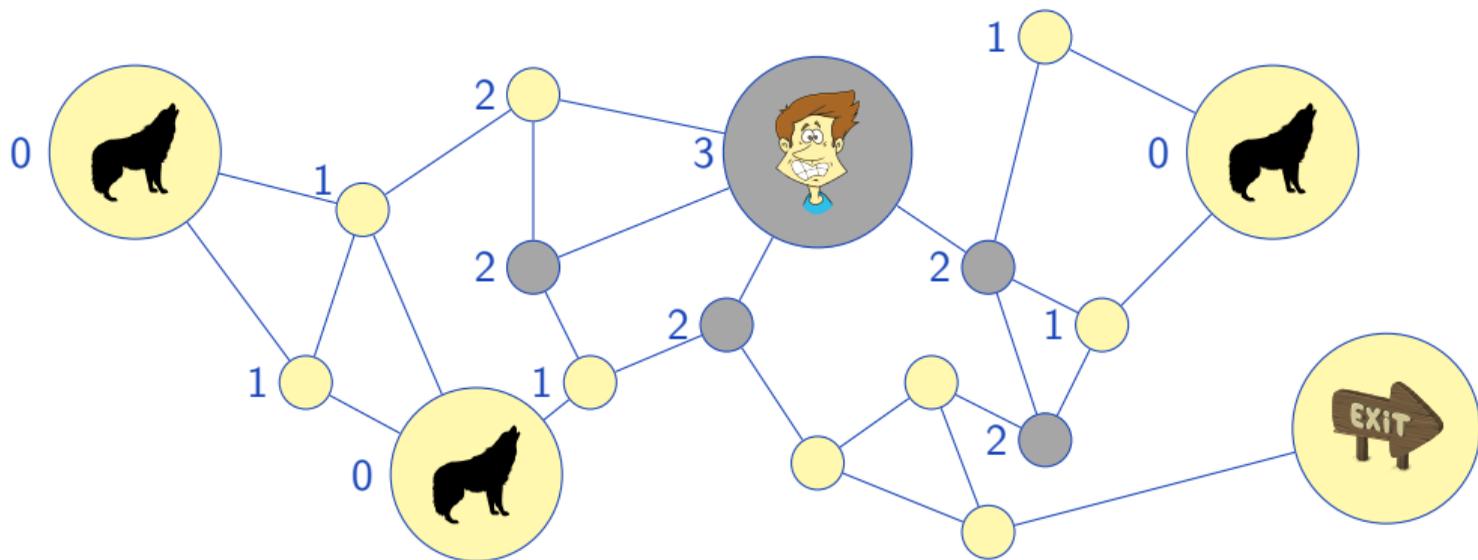
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



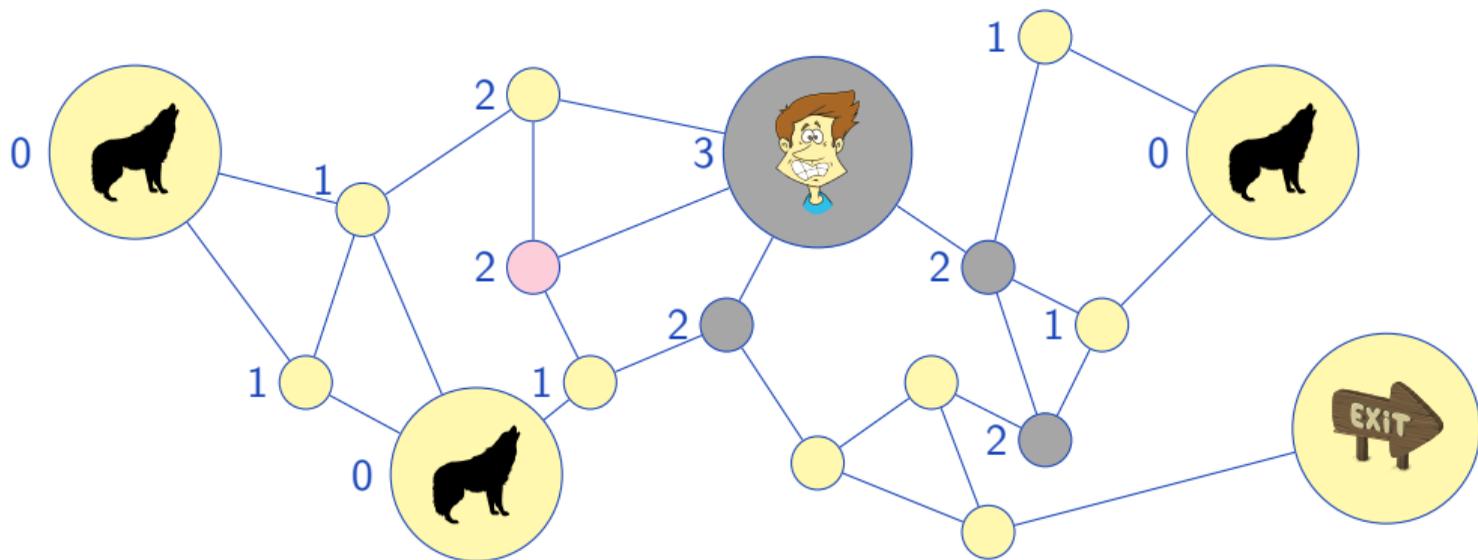
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



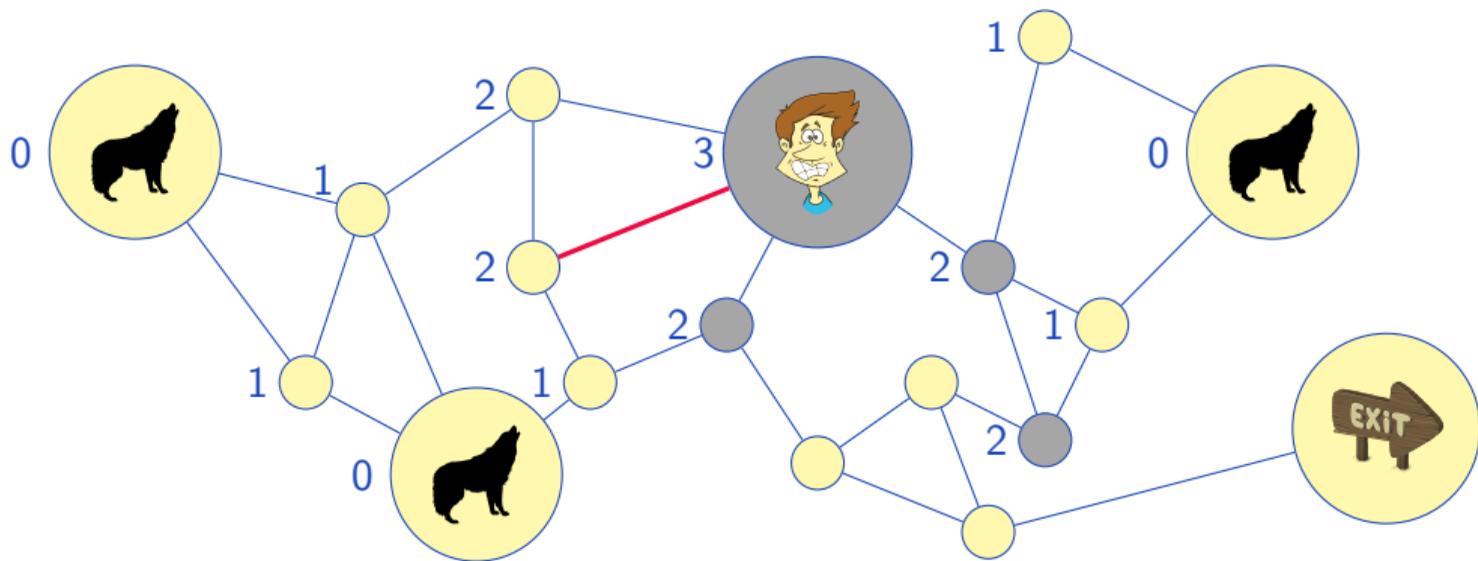
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



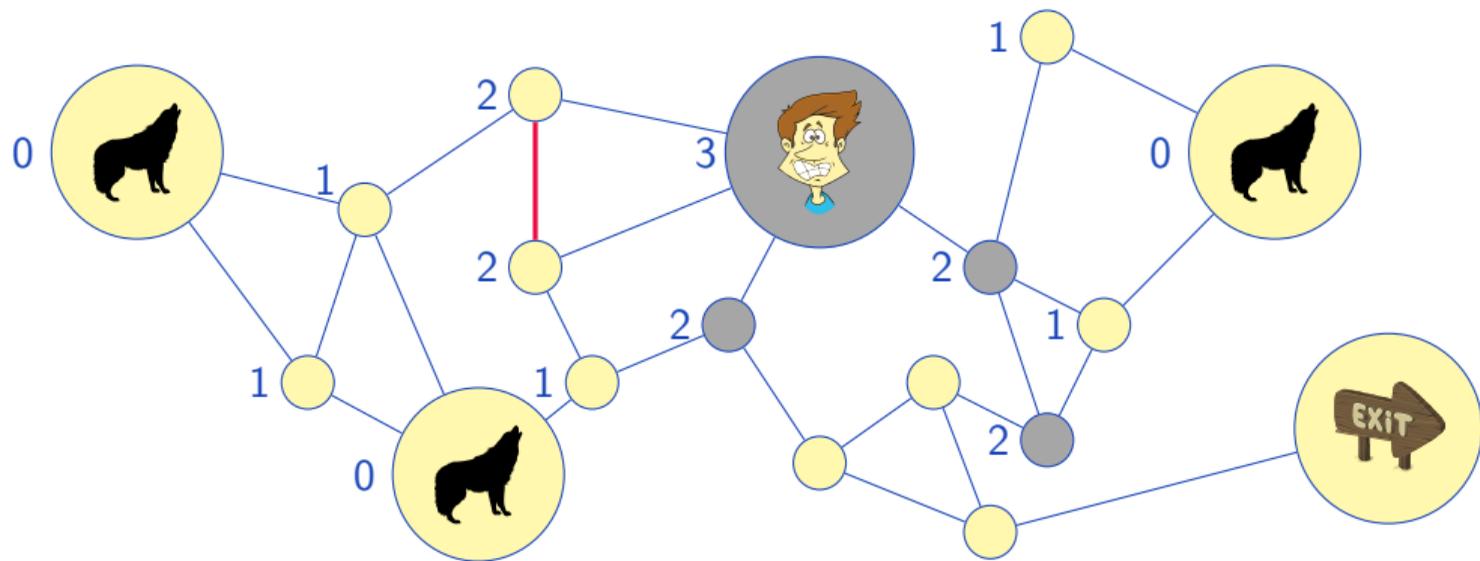
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



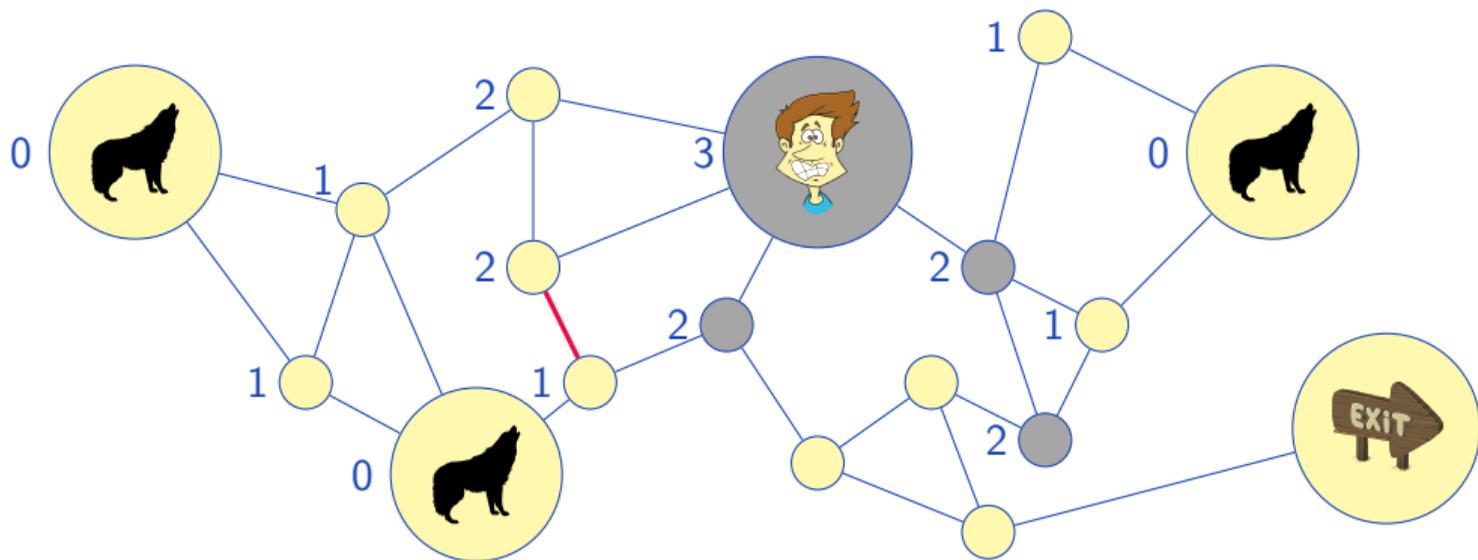
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



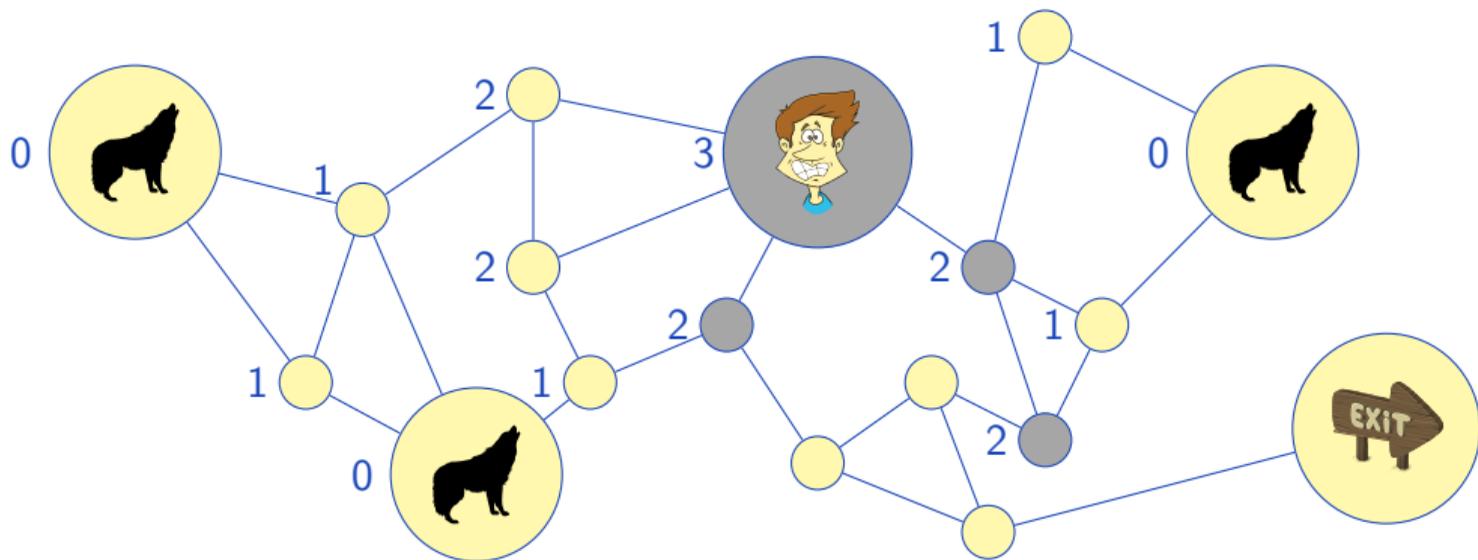
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



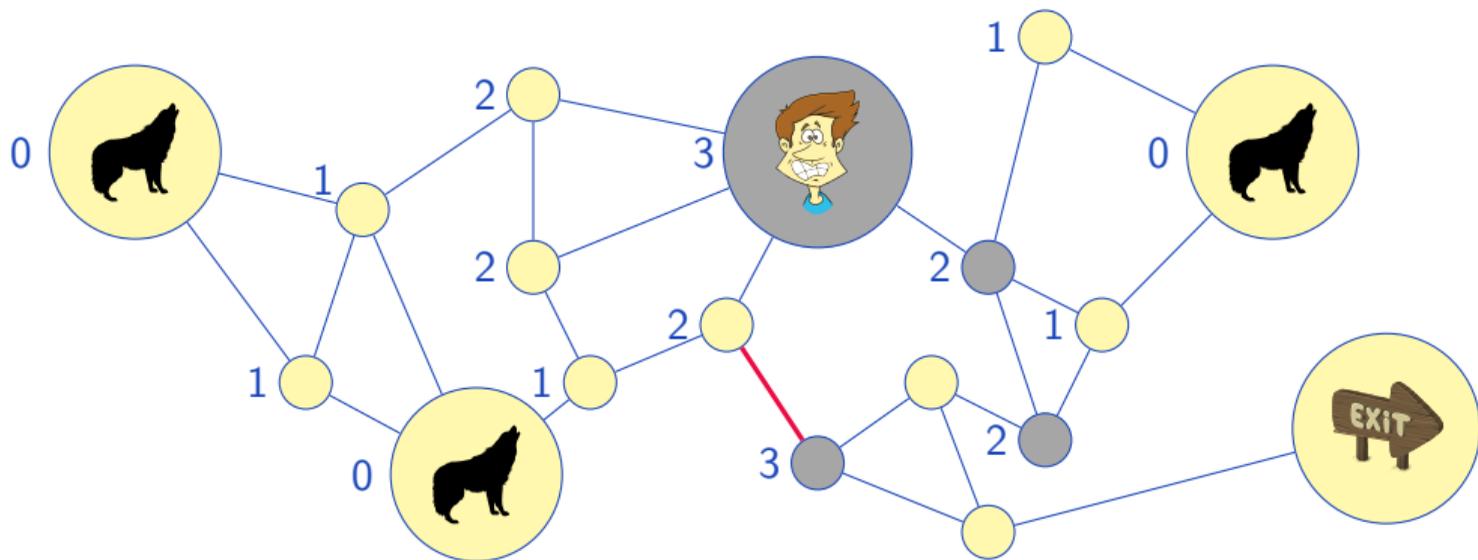
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



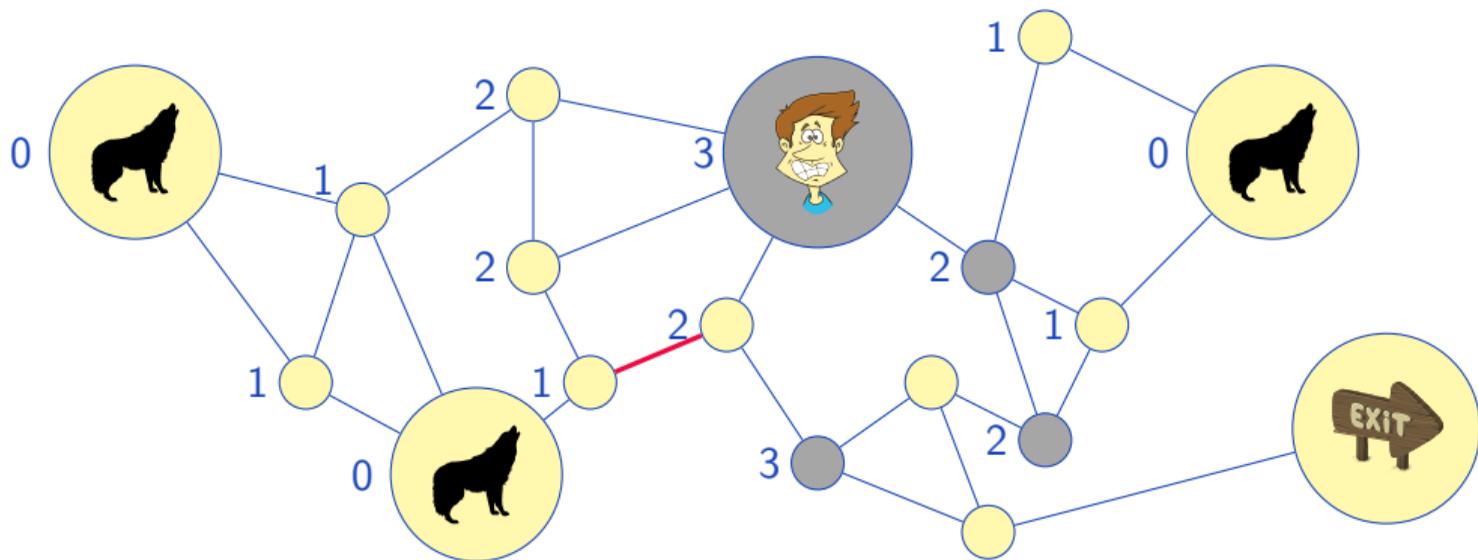
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



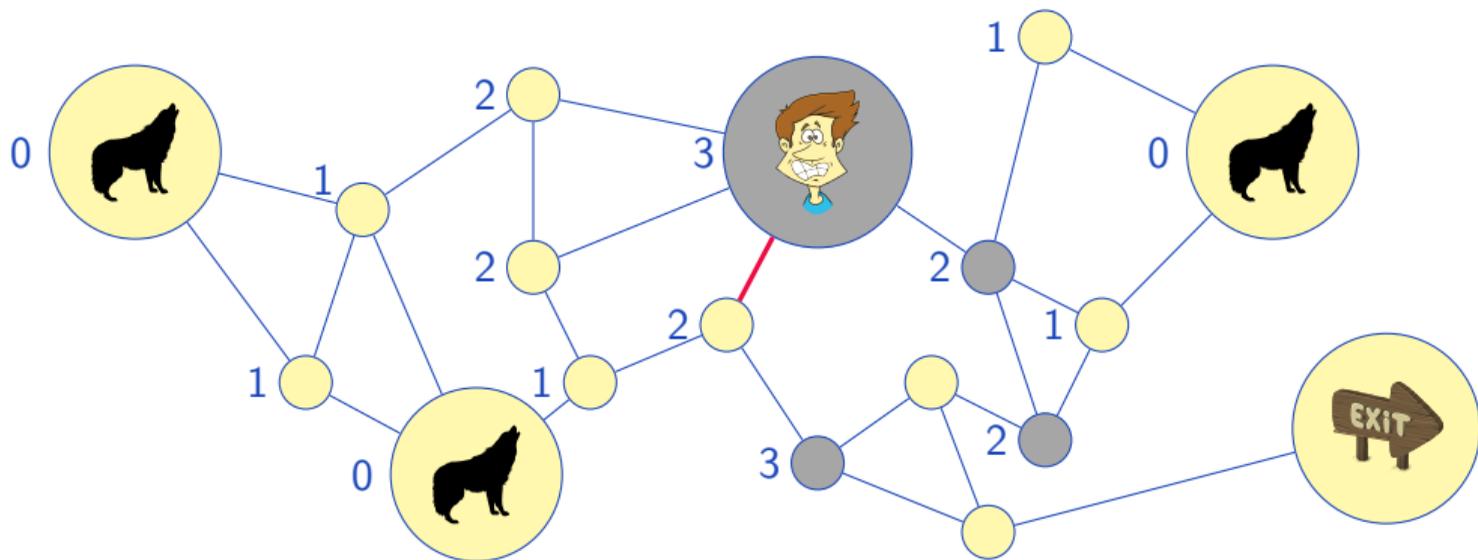
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



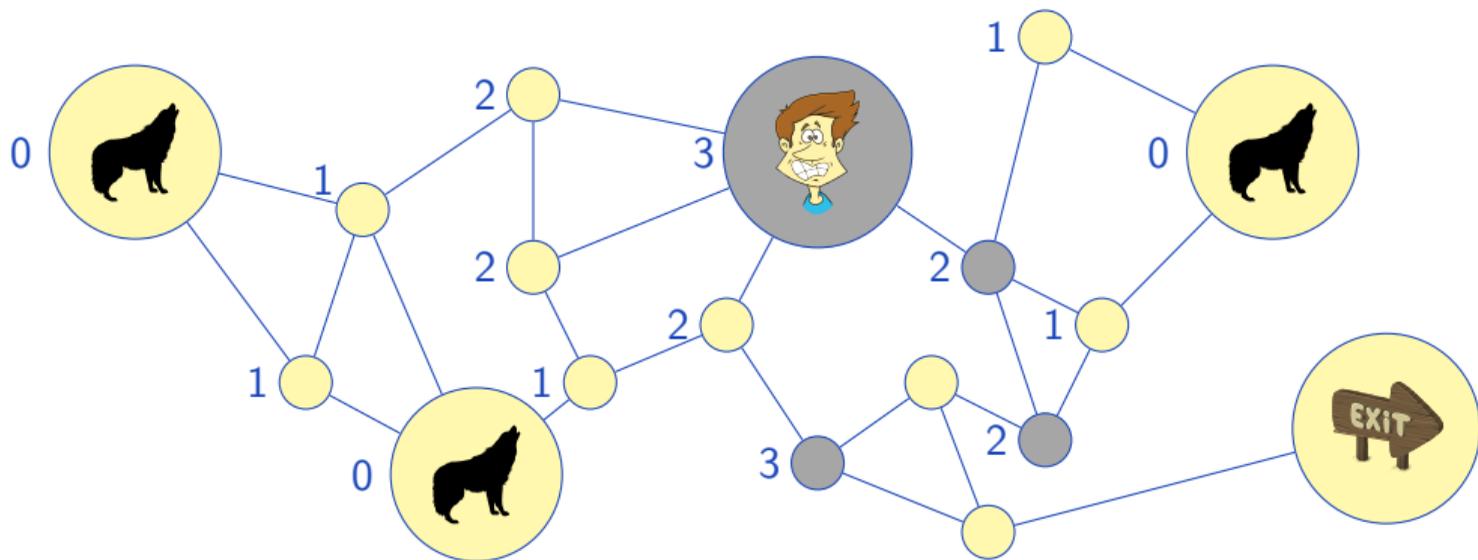
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



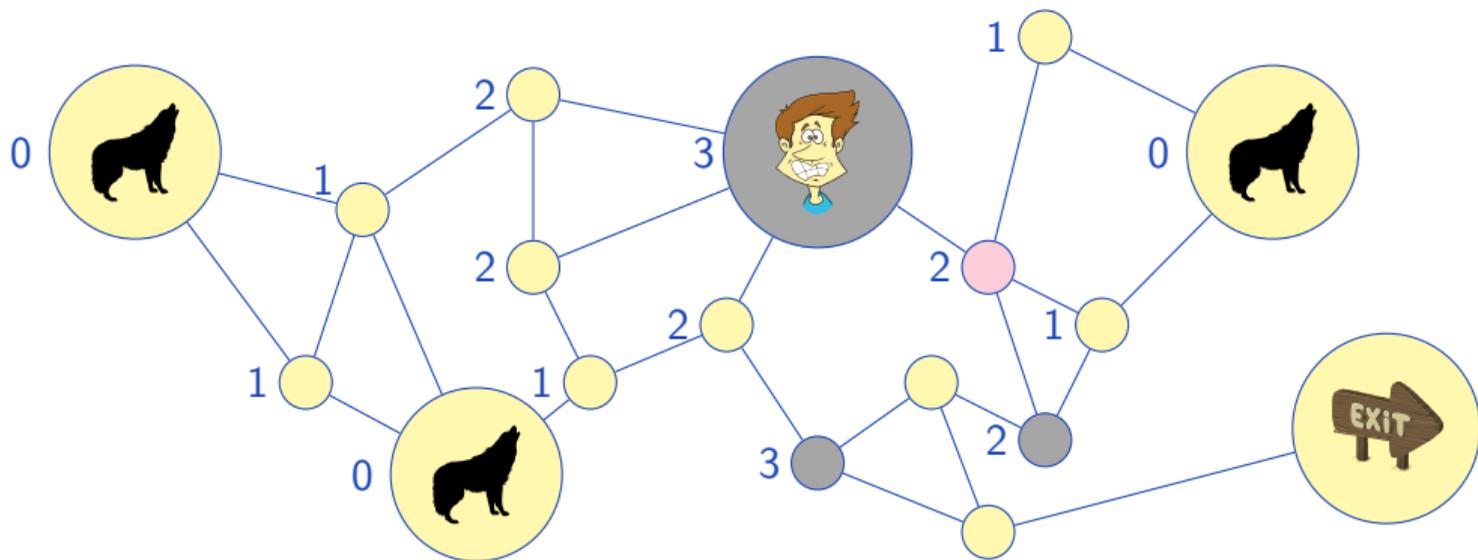
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



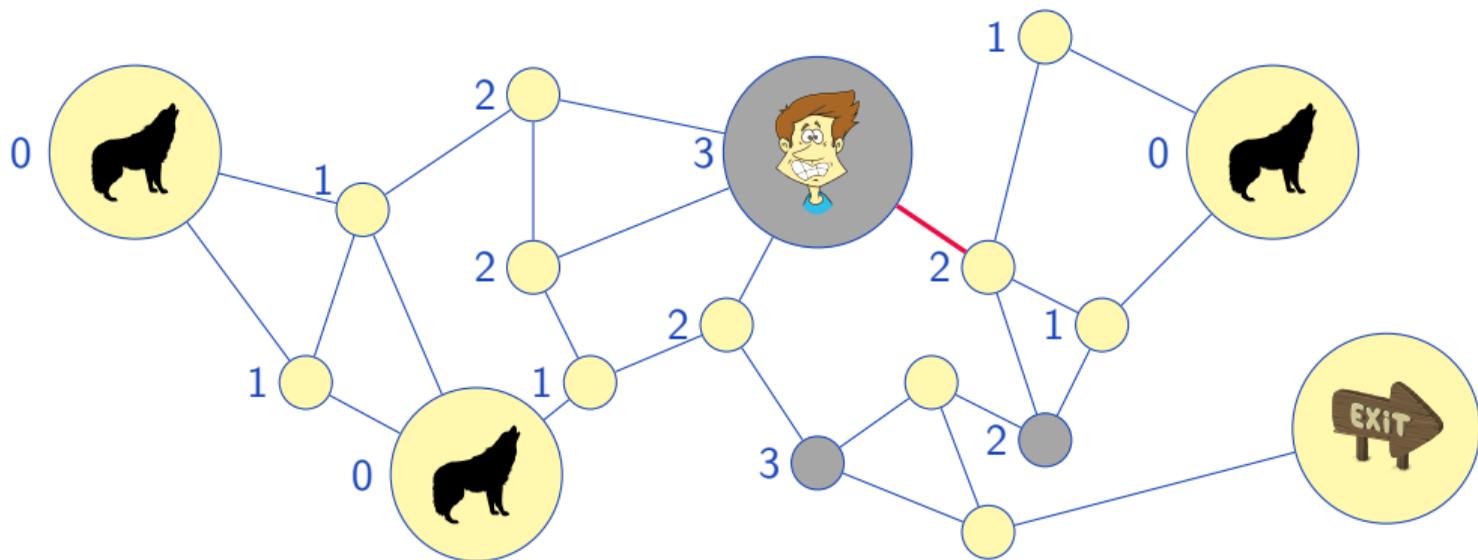
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



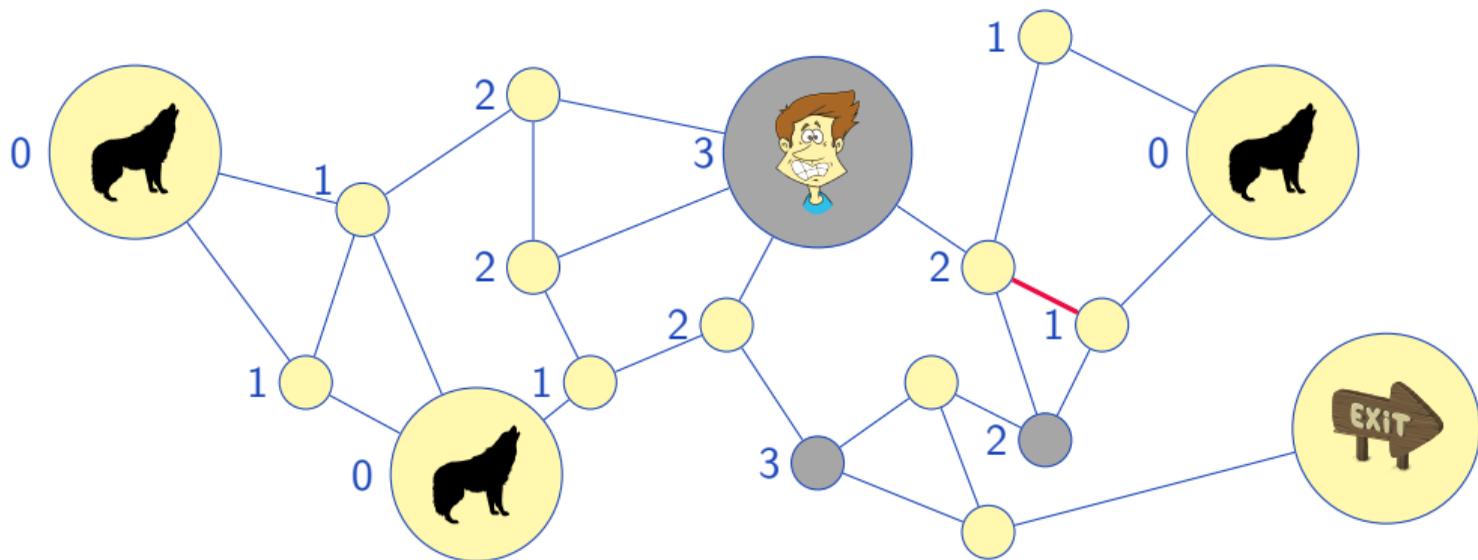
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



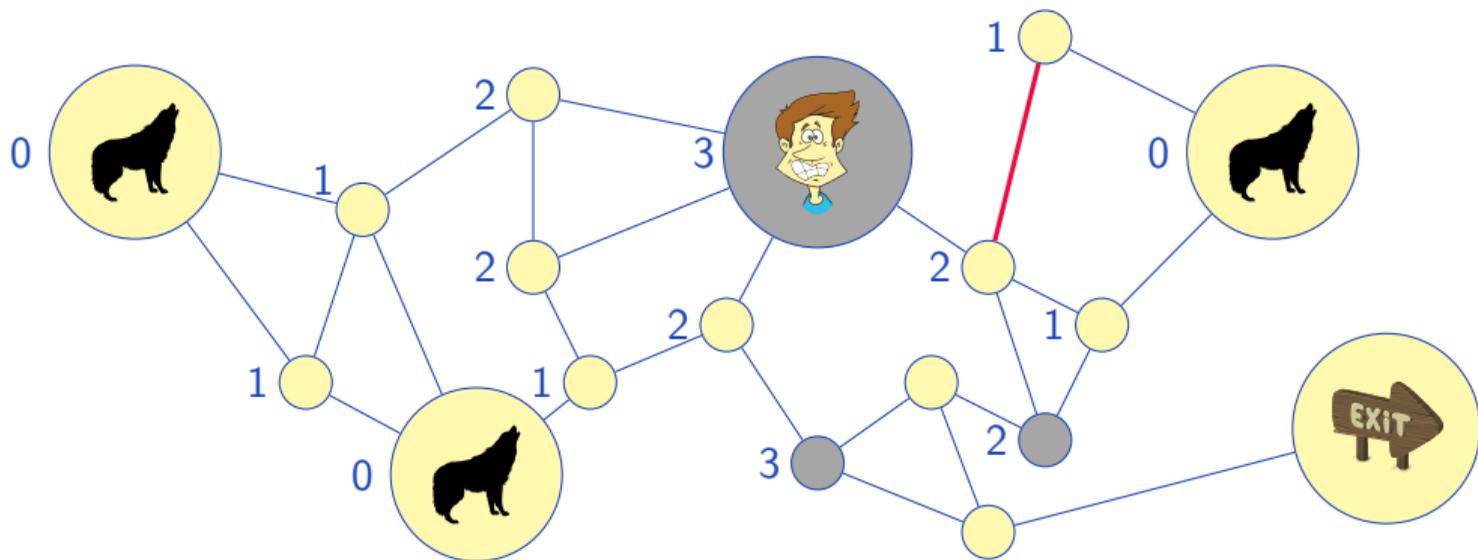
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



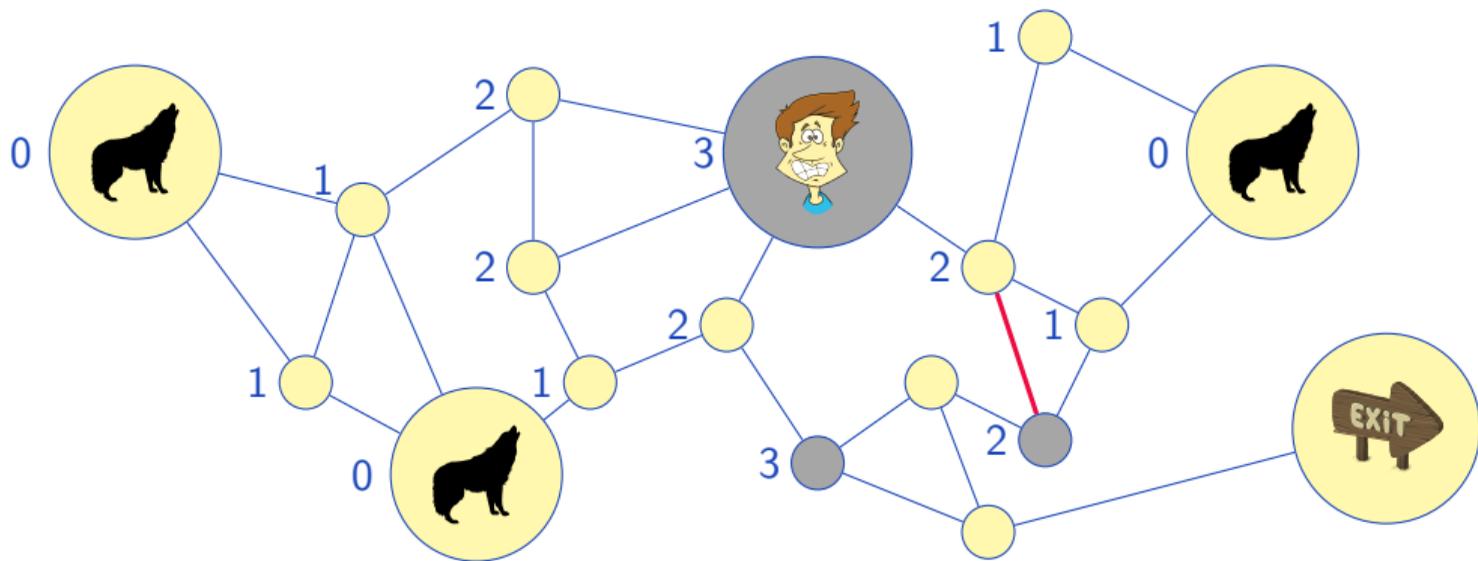
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



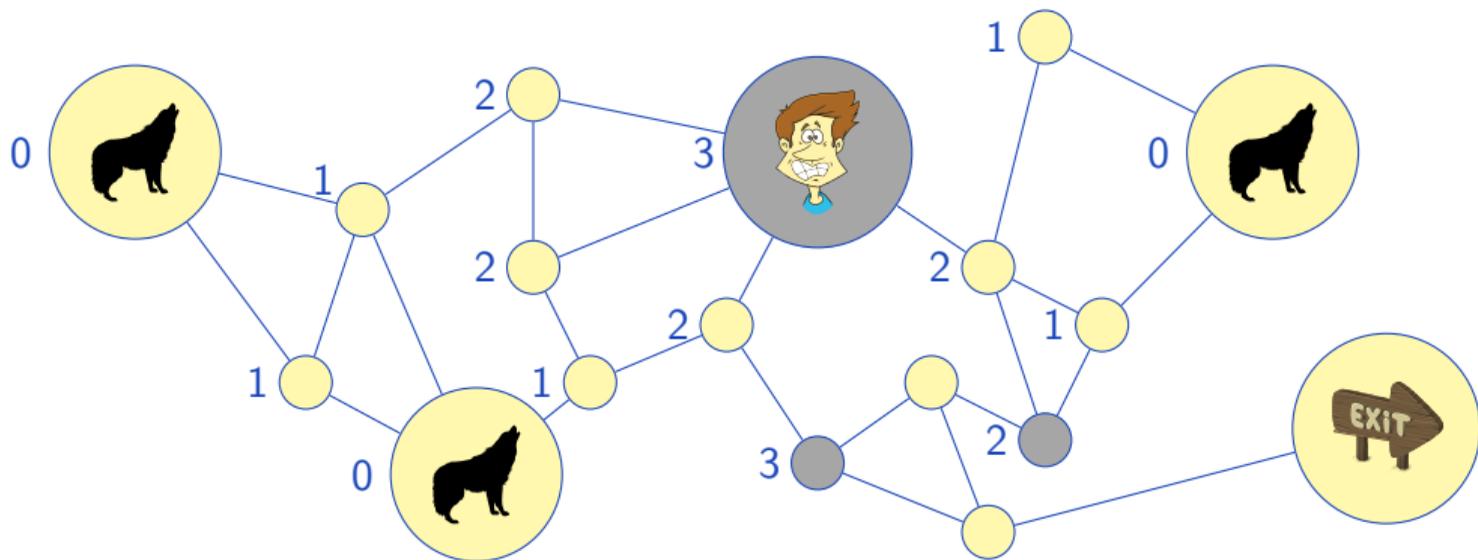
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



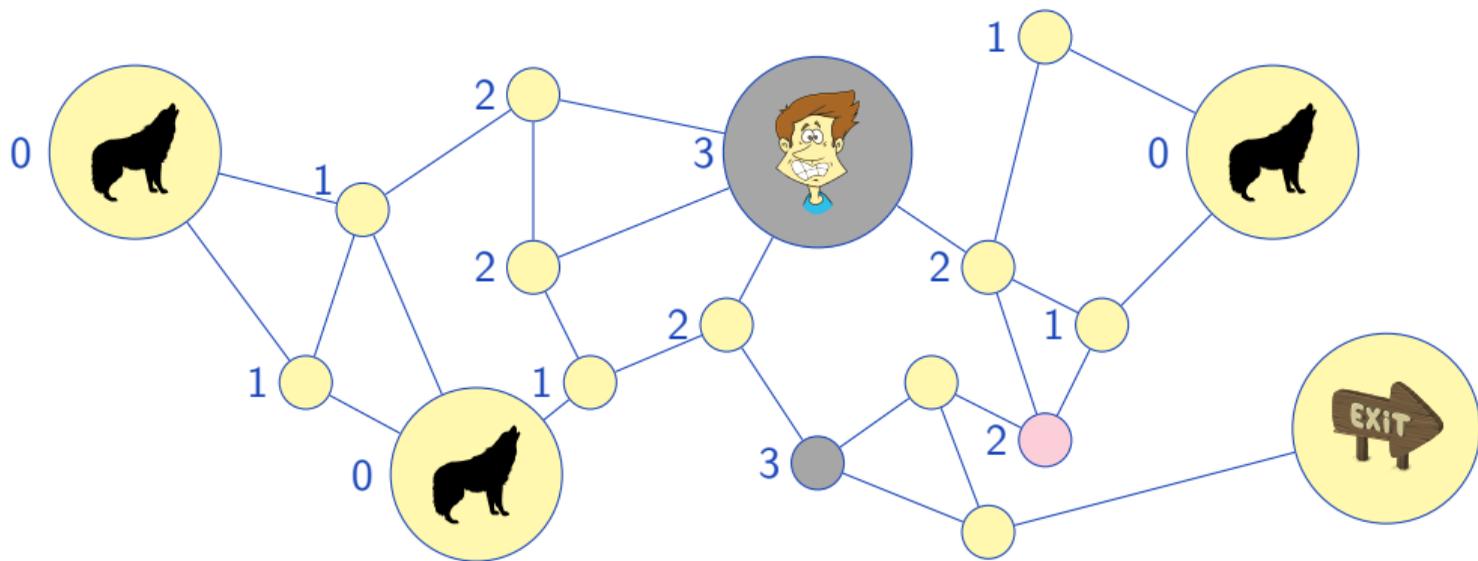
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



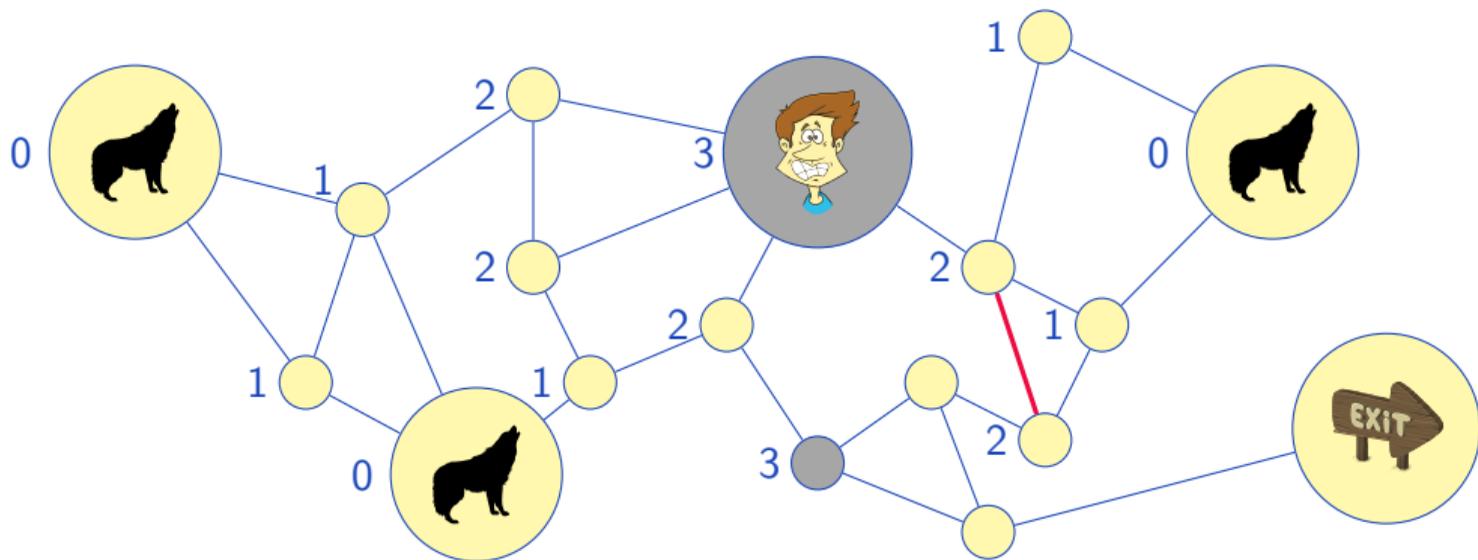
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



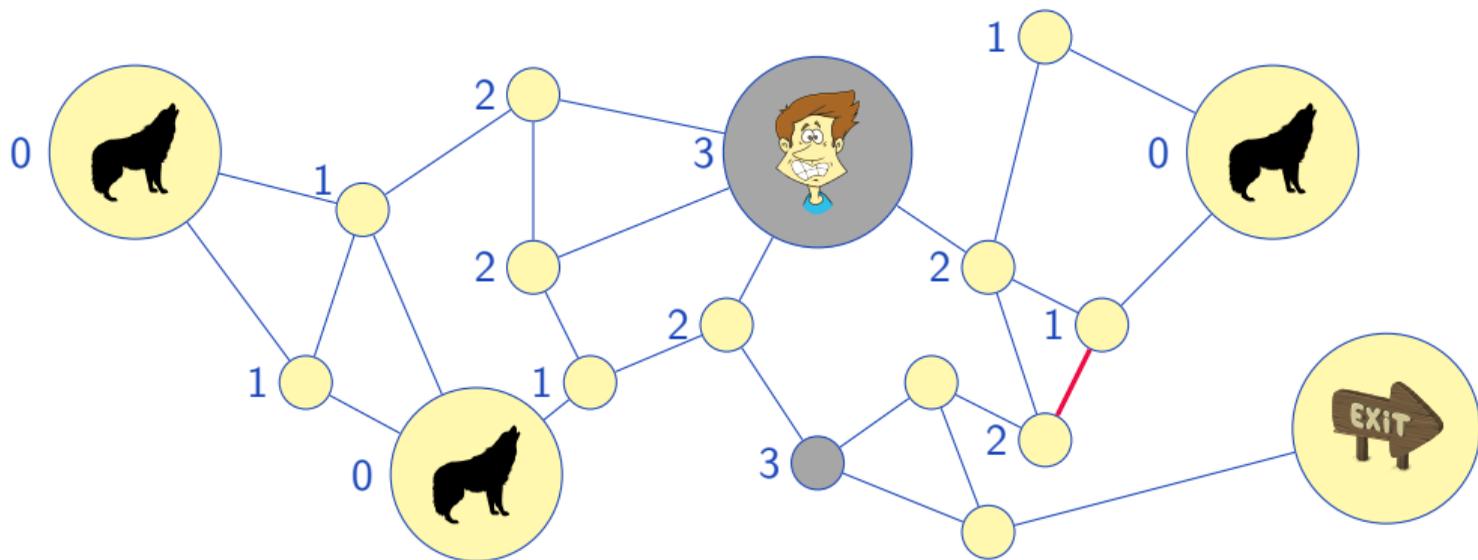
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



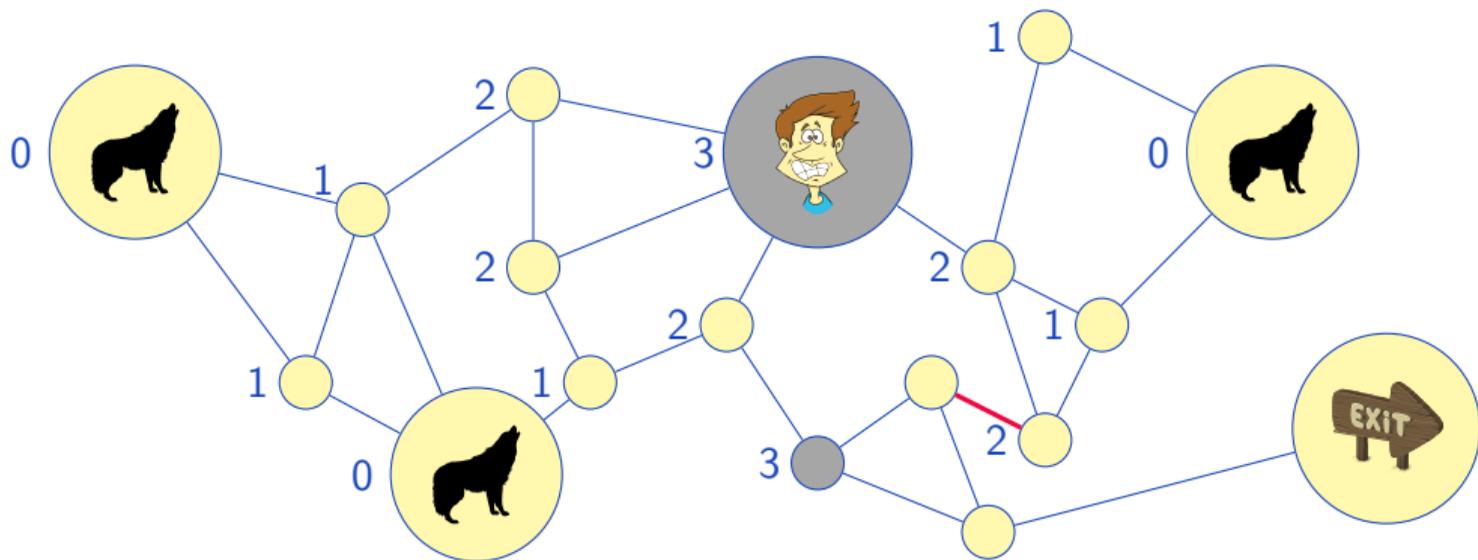
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



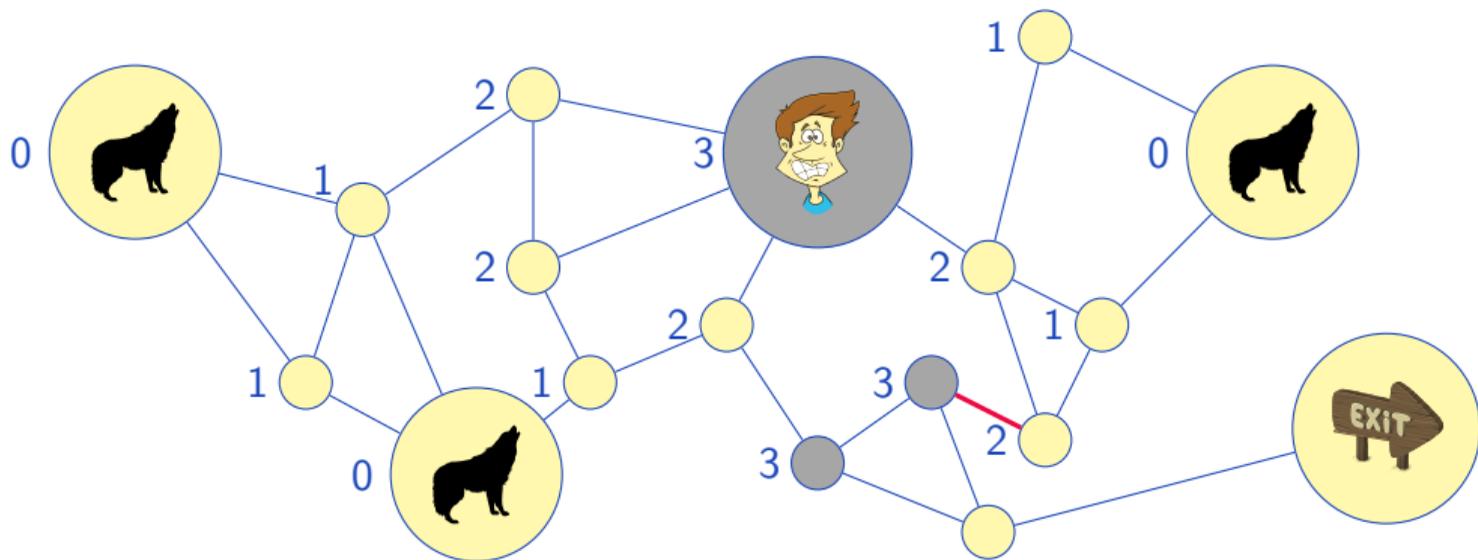
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



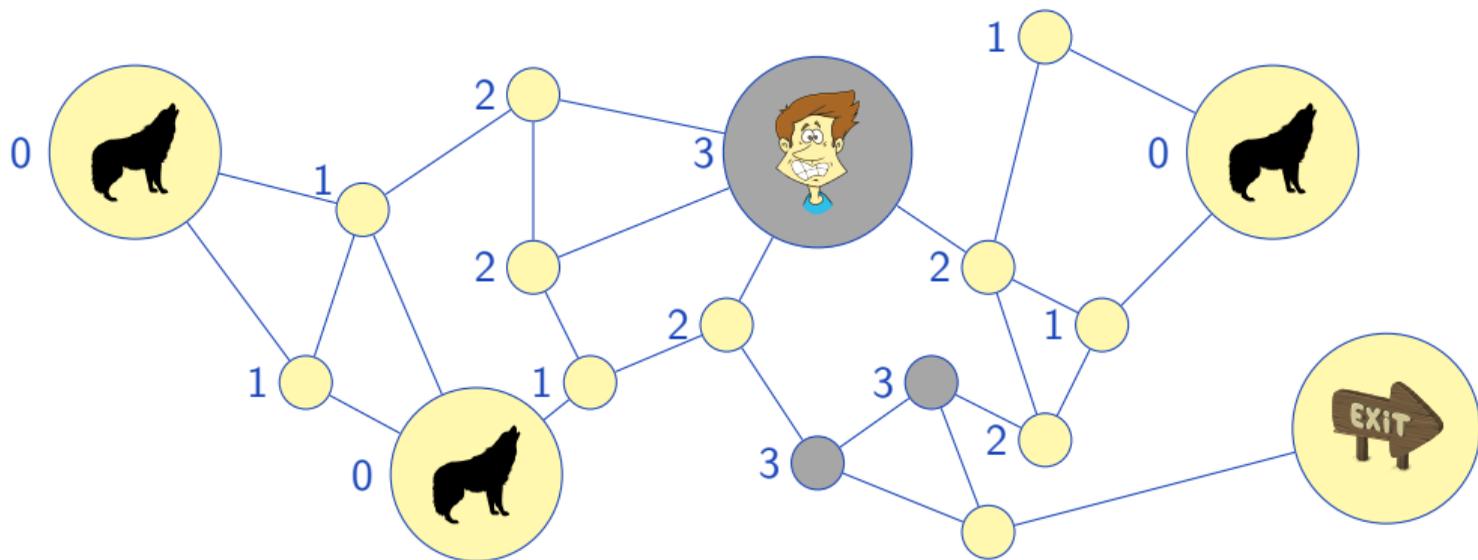
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



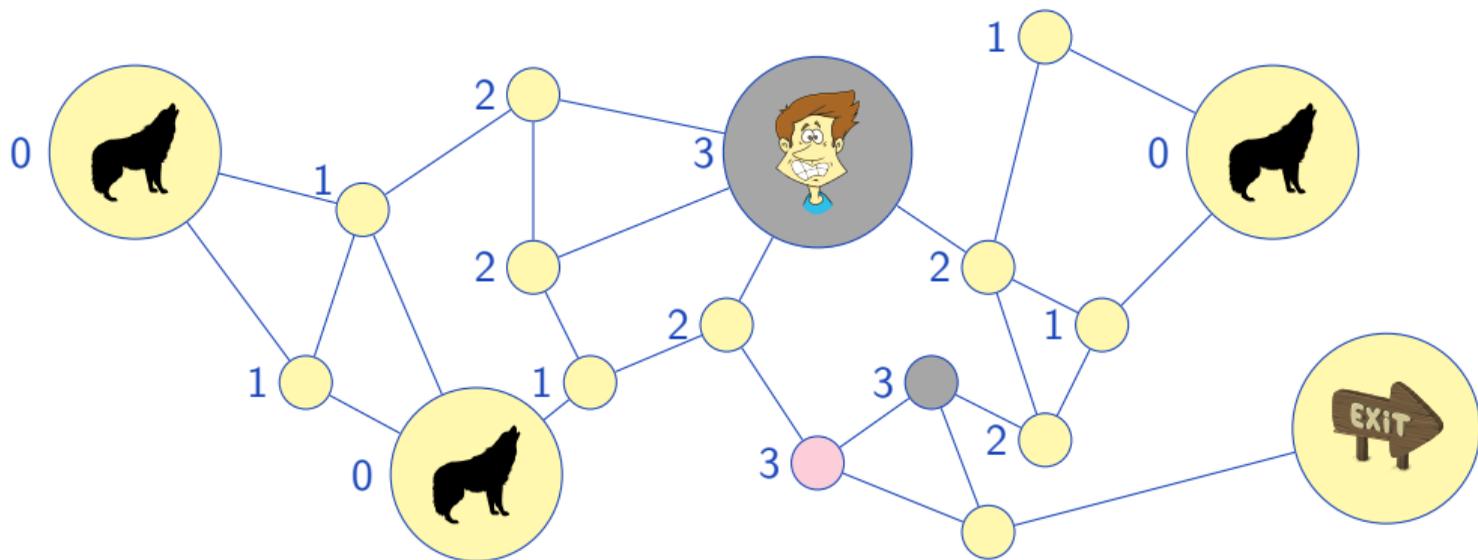
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



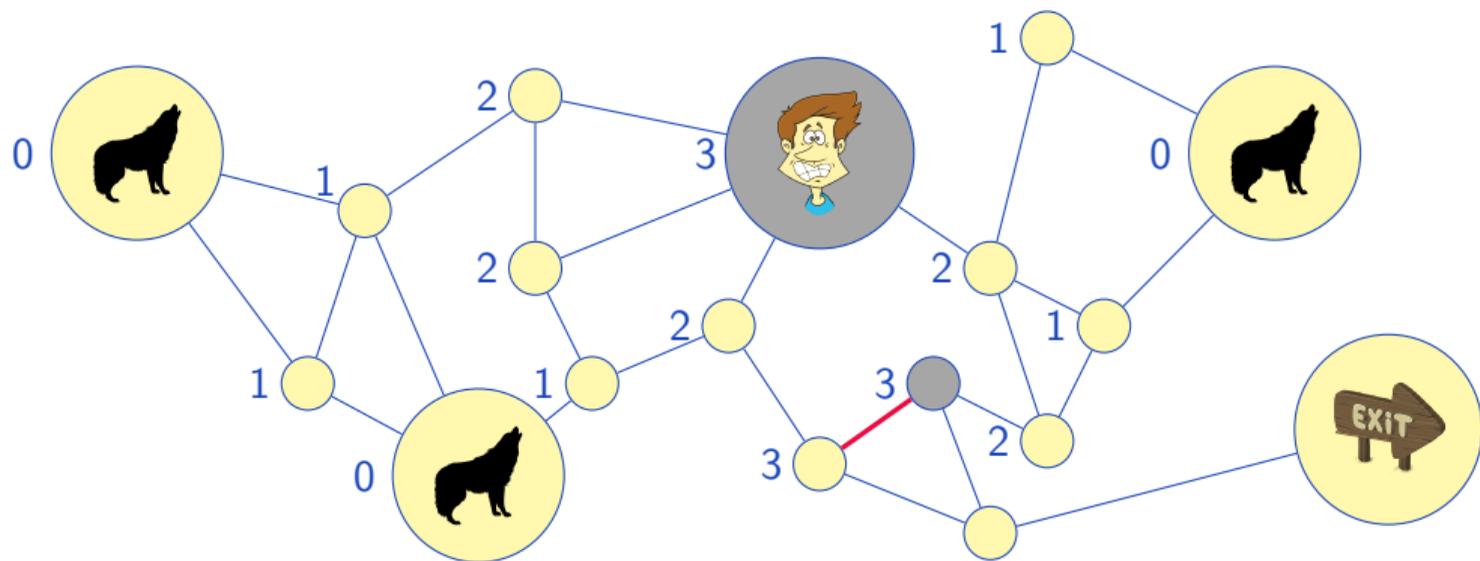
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



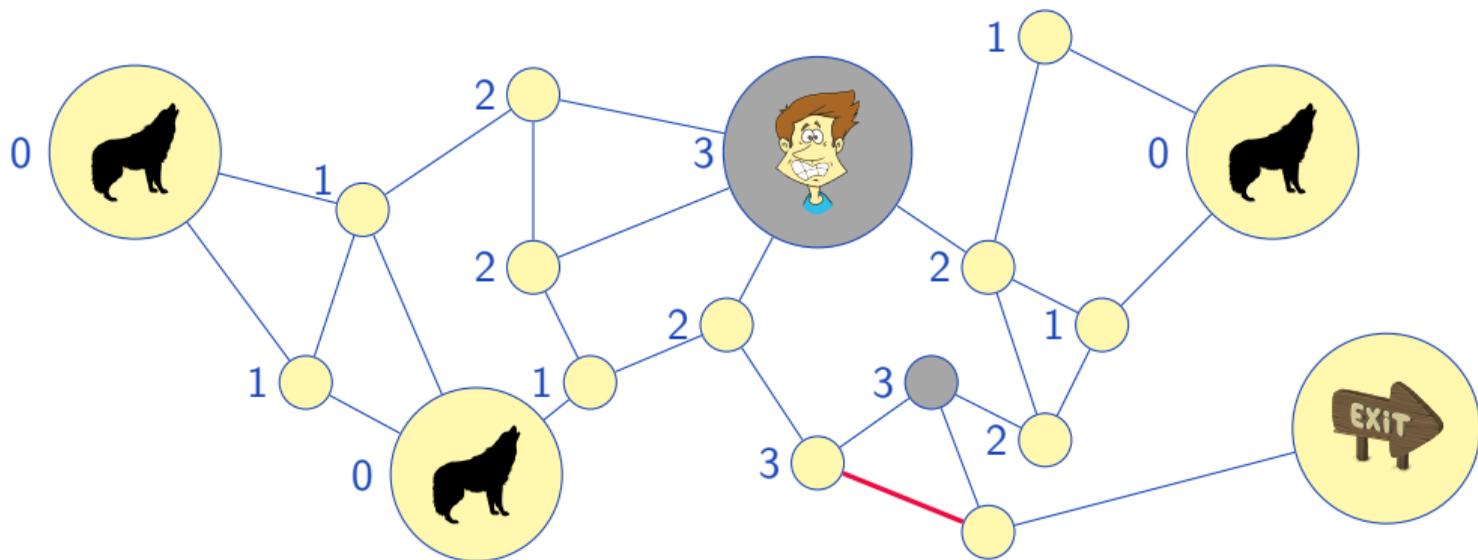
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



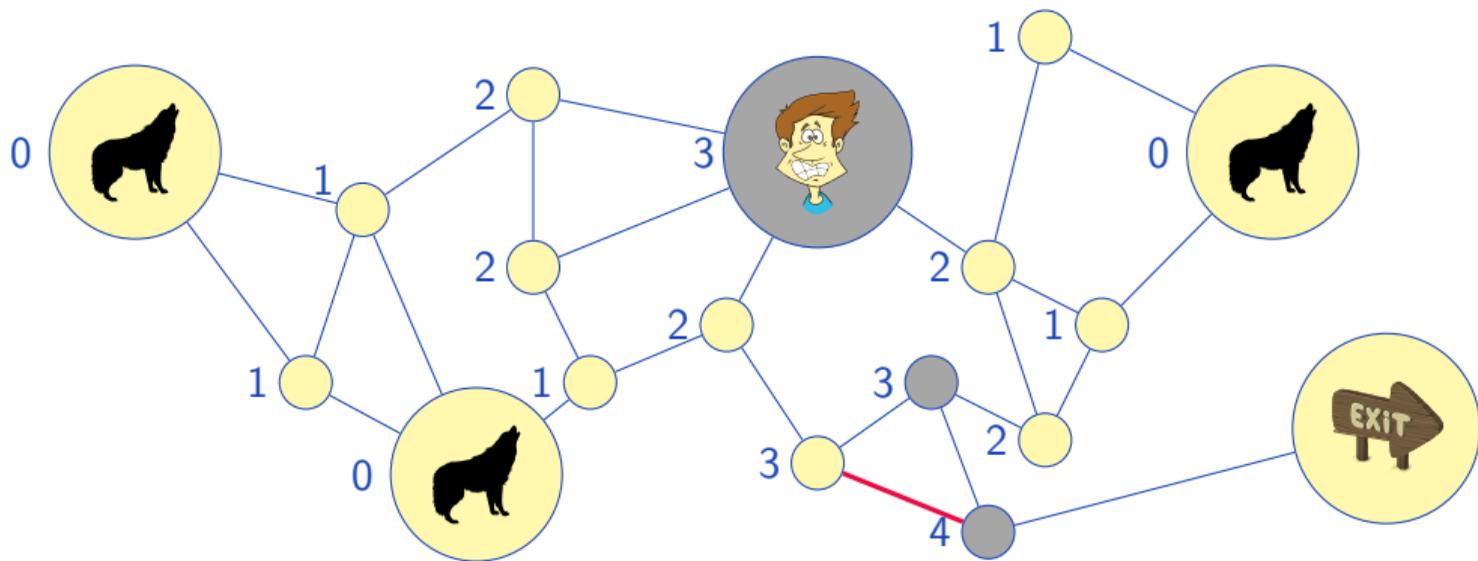
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



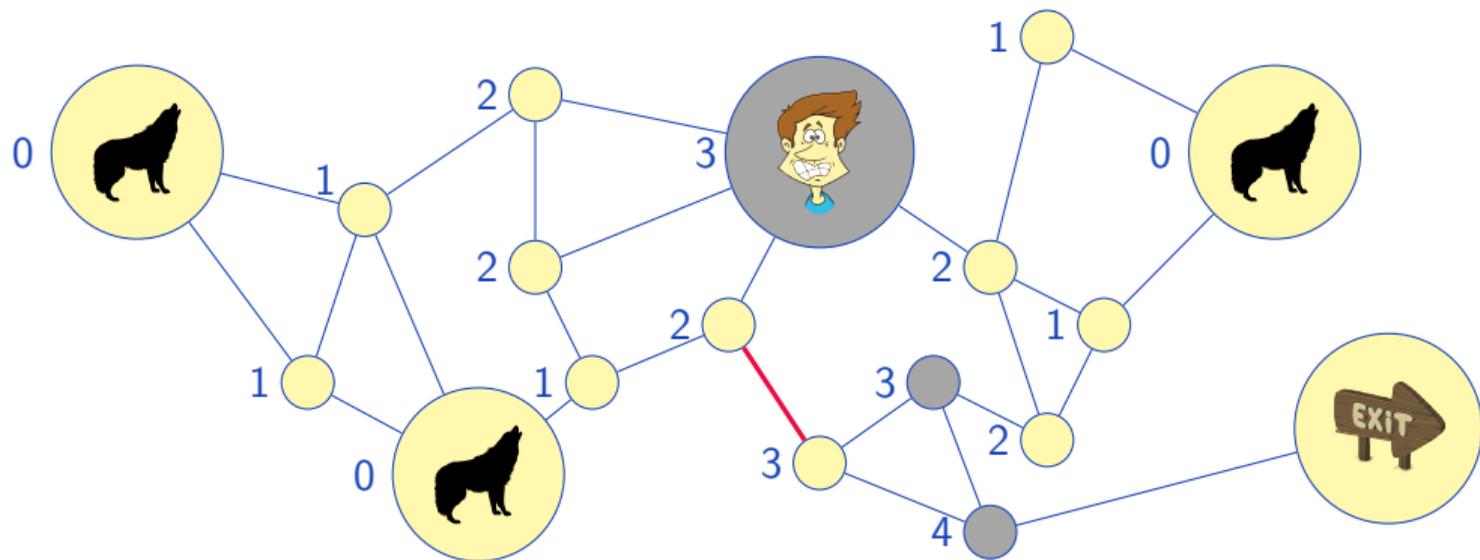
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



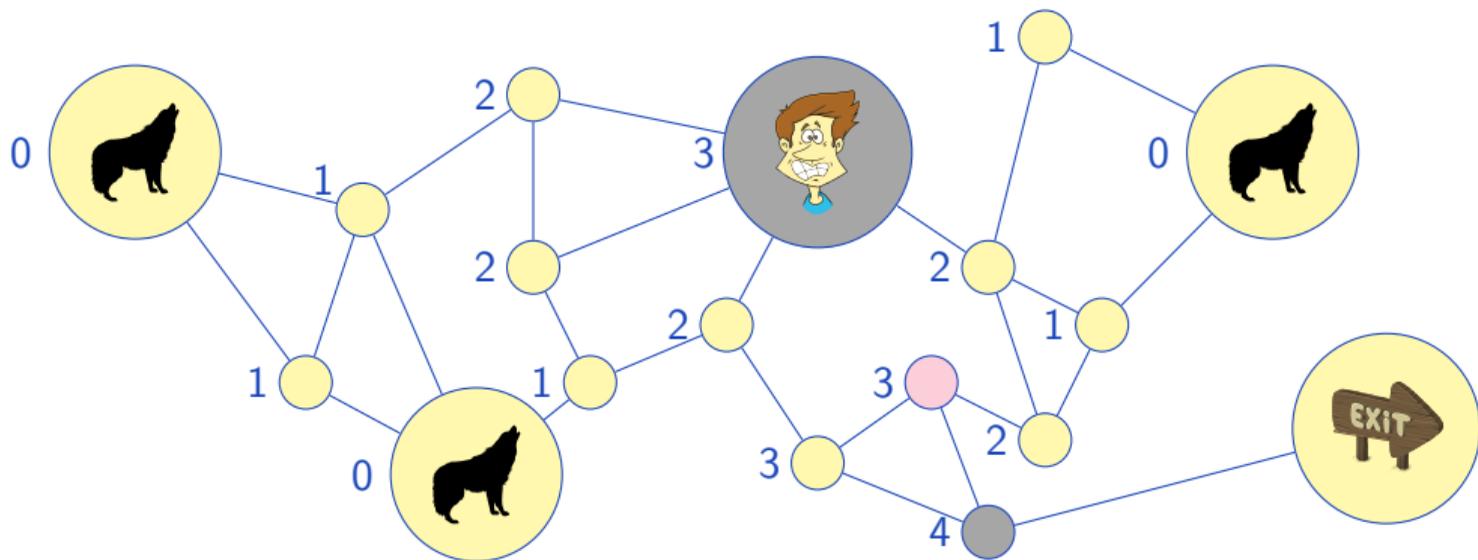
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



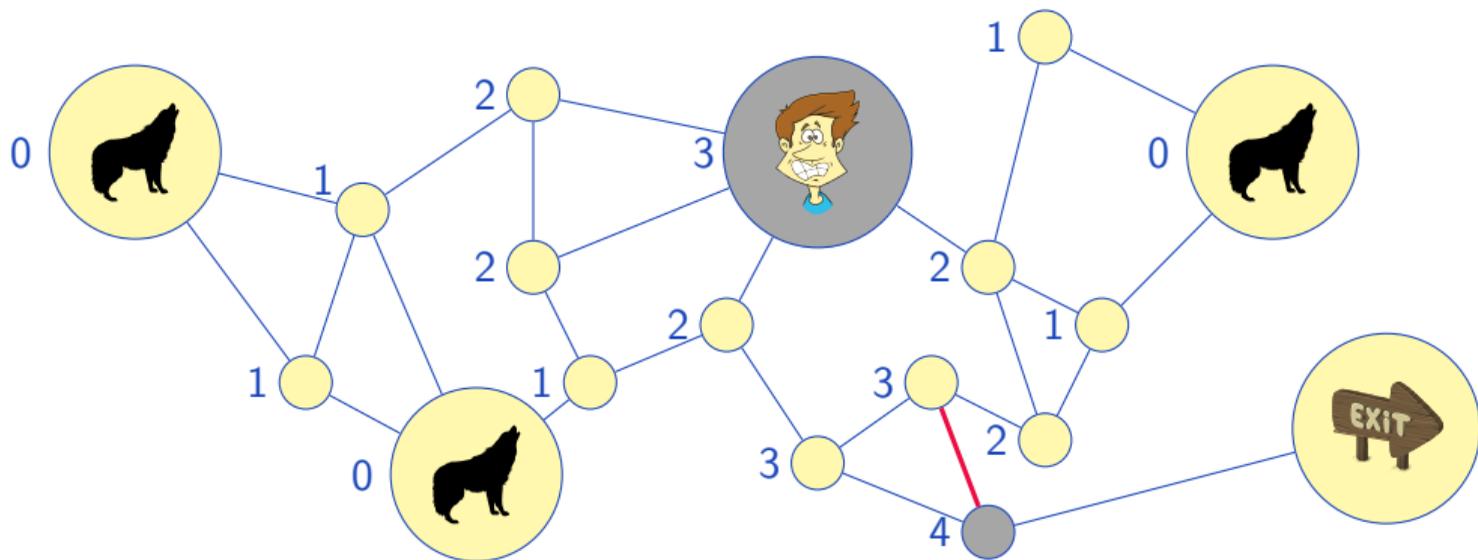
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



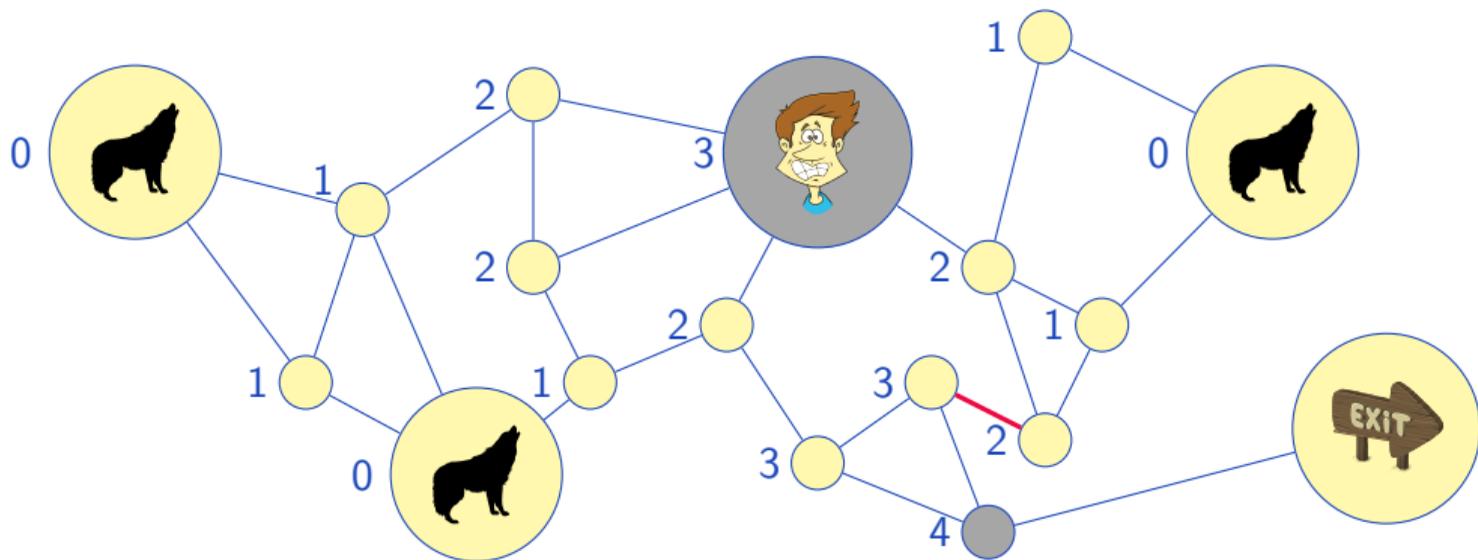
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



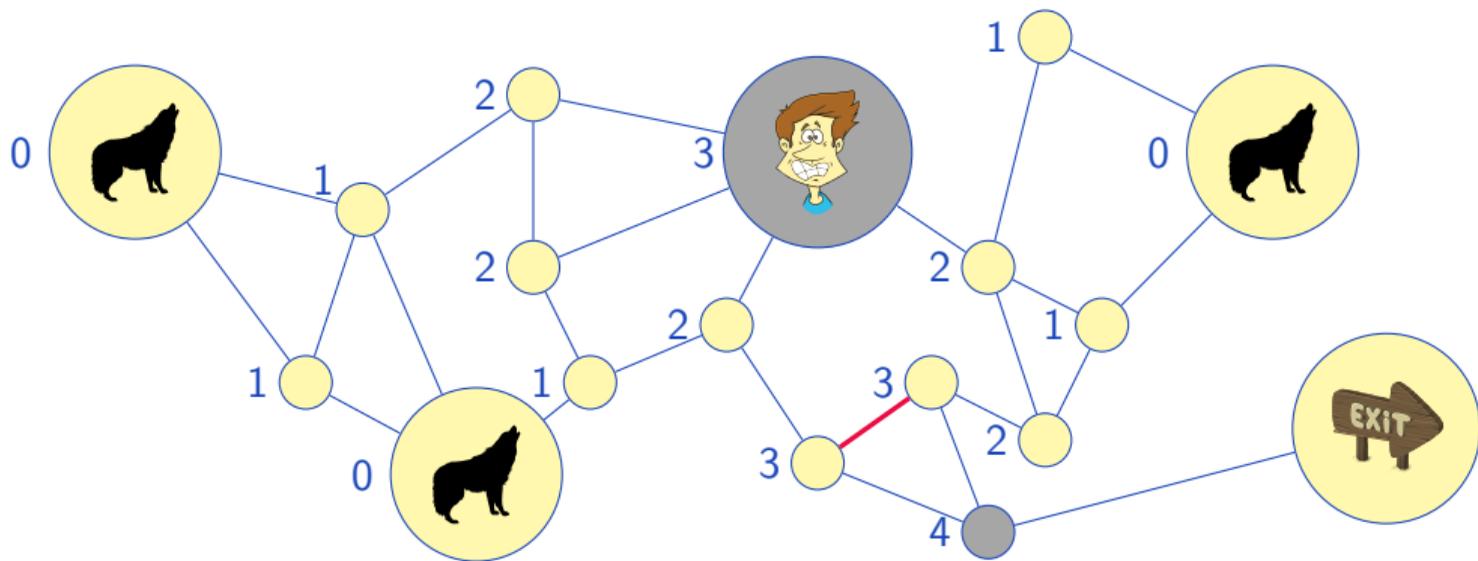
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



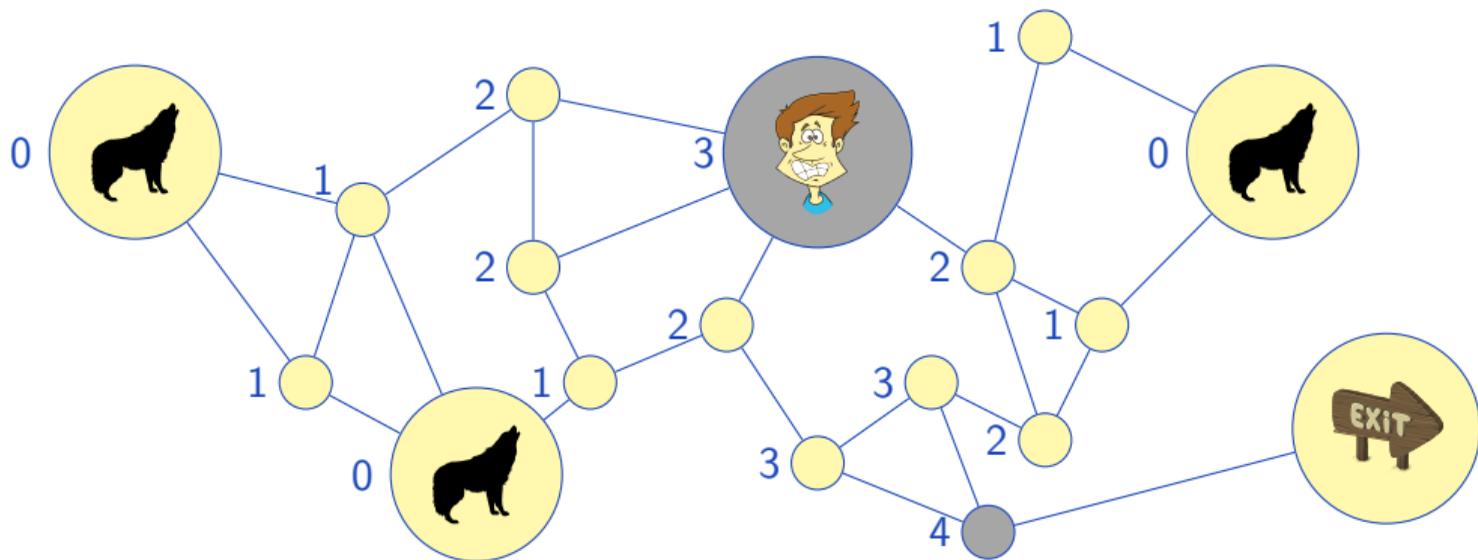
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



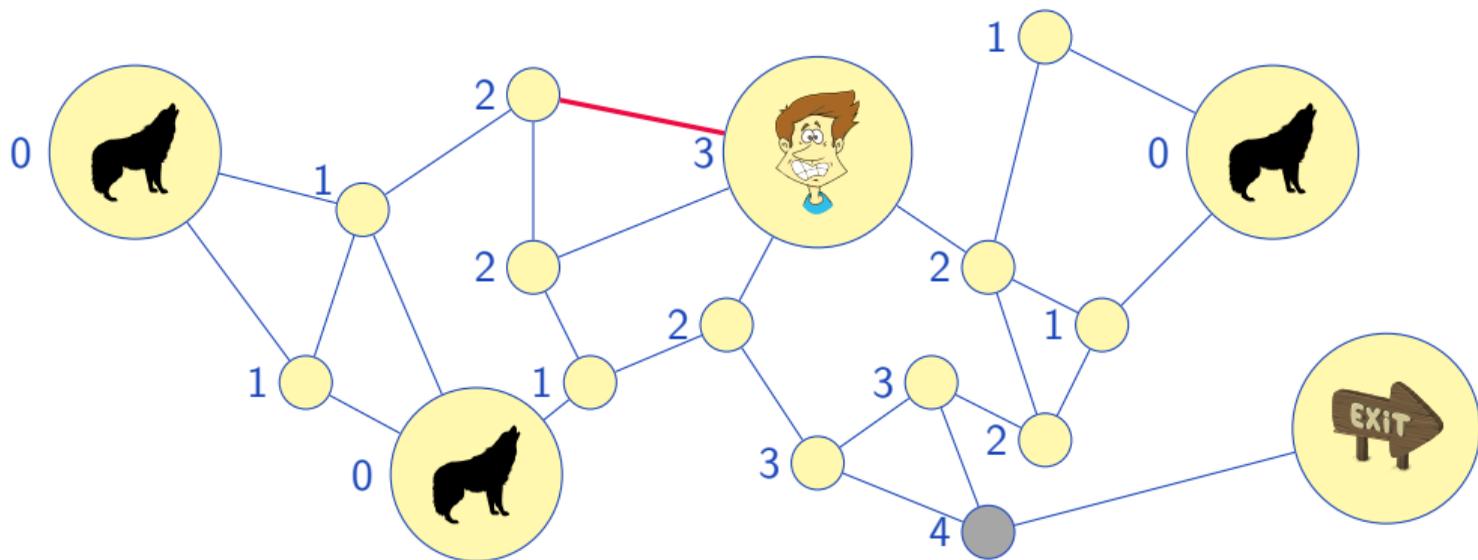
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



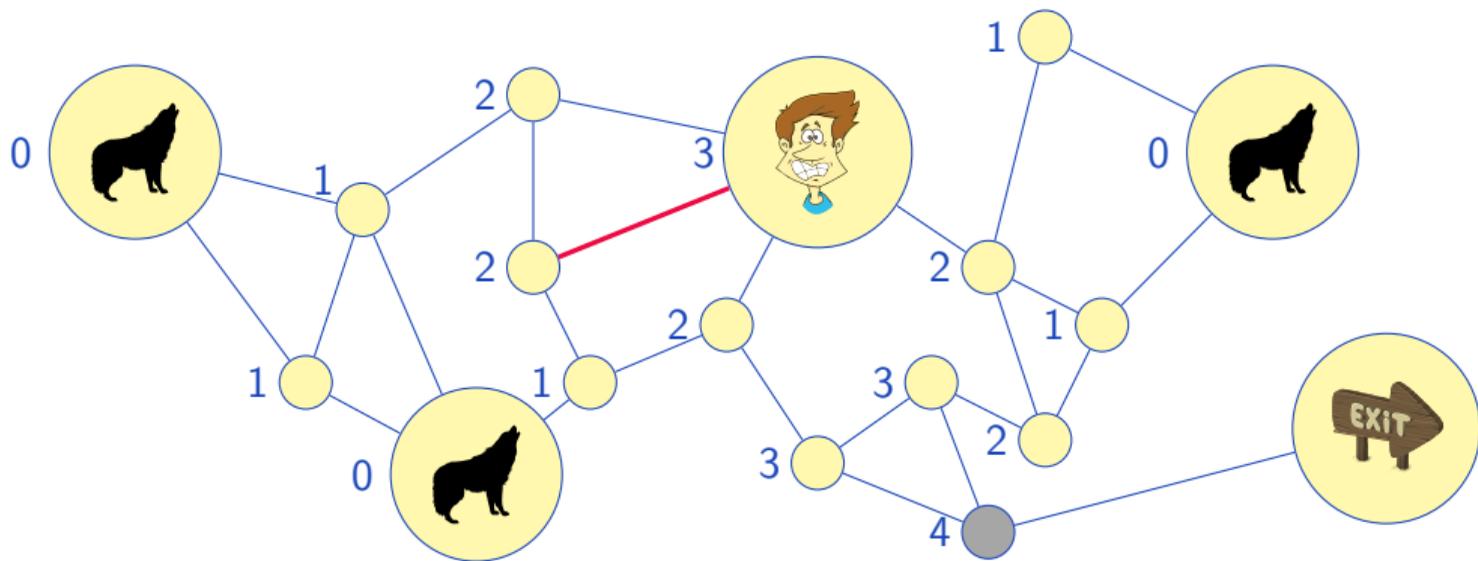
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



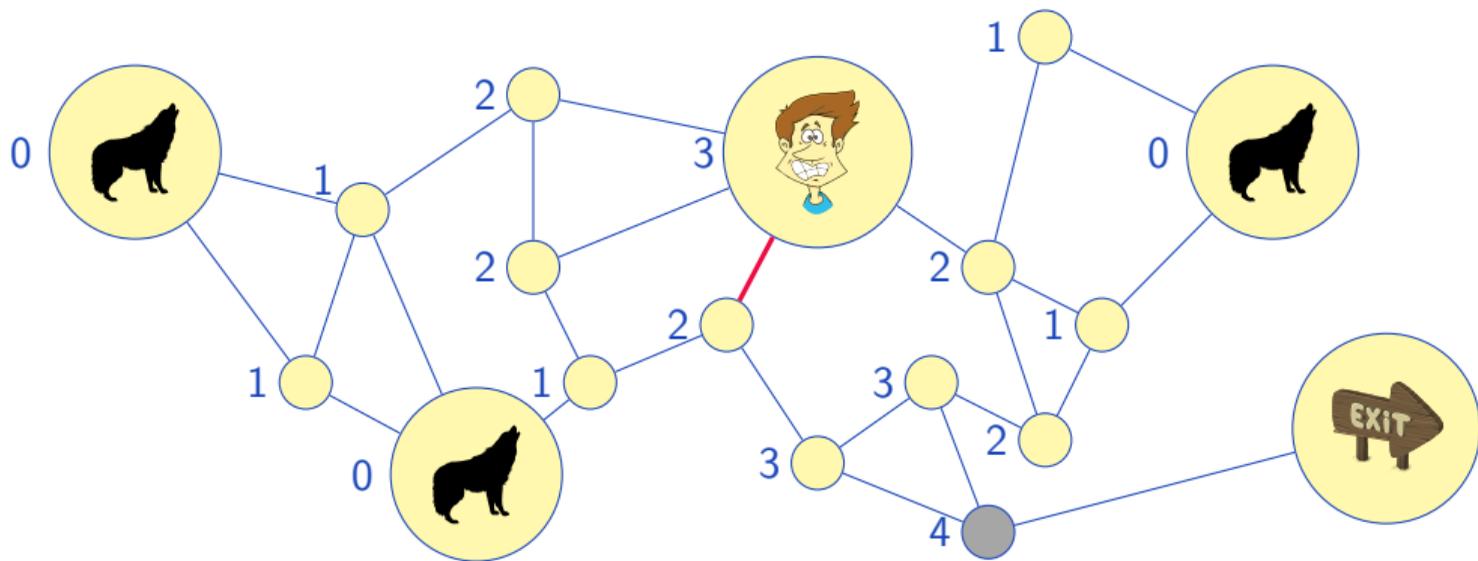
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



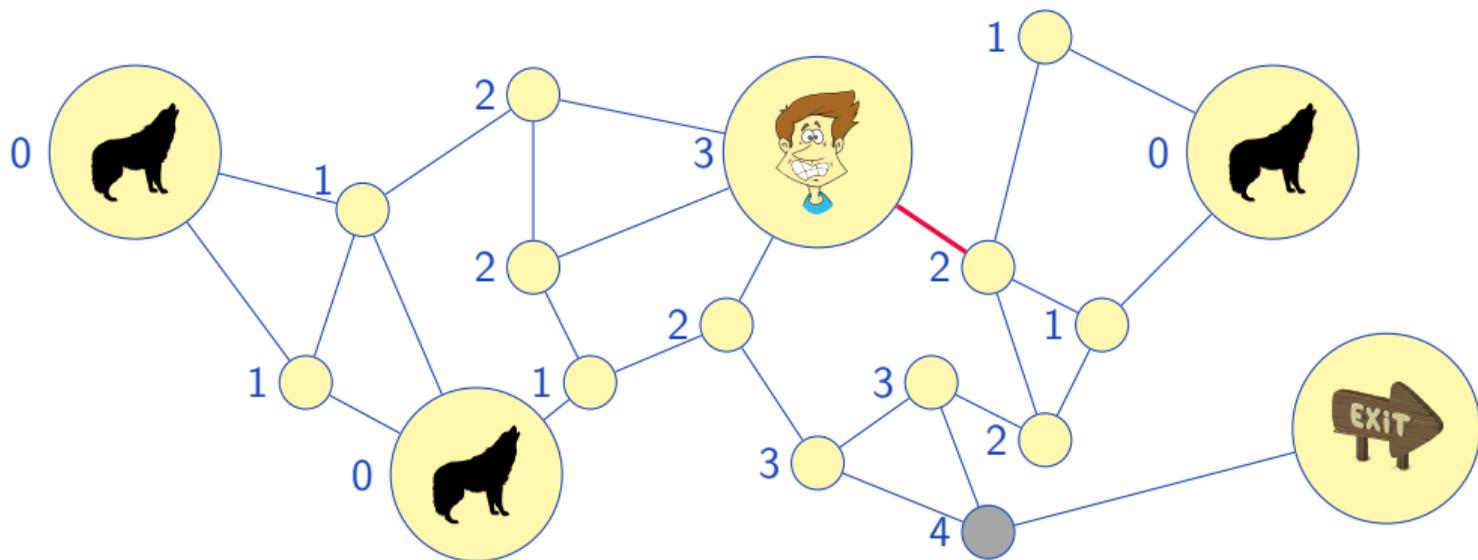
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



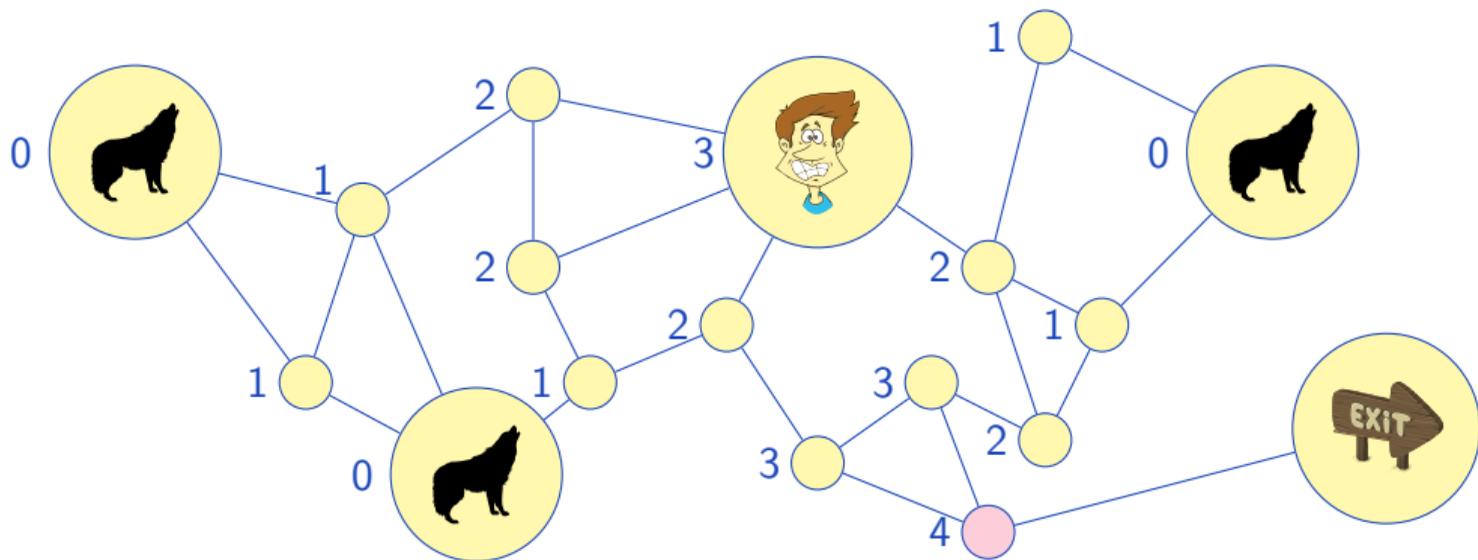
Images are courtesy of *GDJ*, *knollbaco*, *glitch* from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



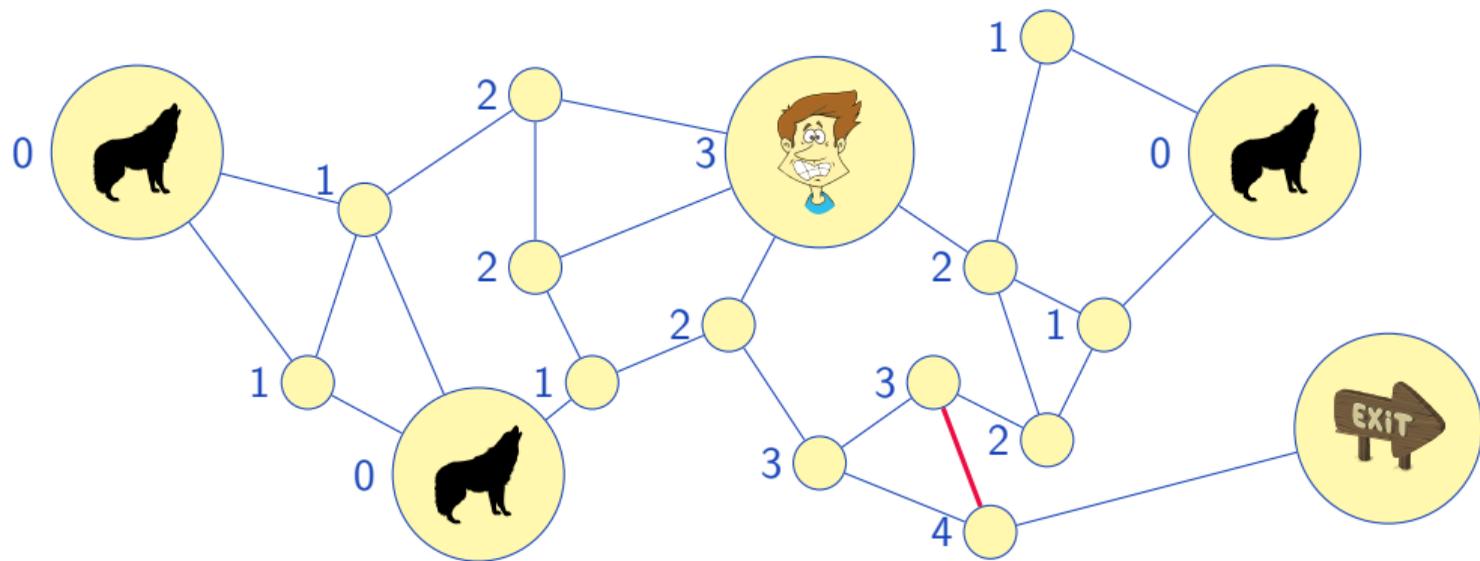
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



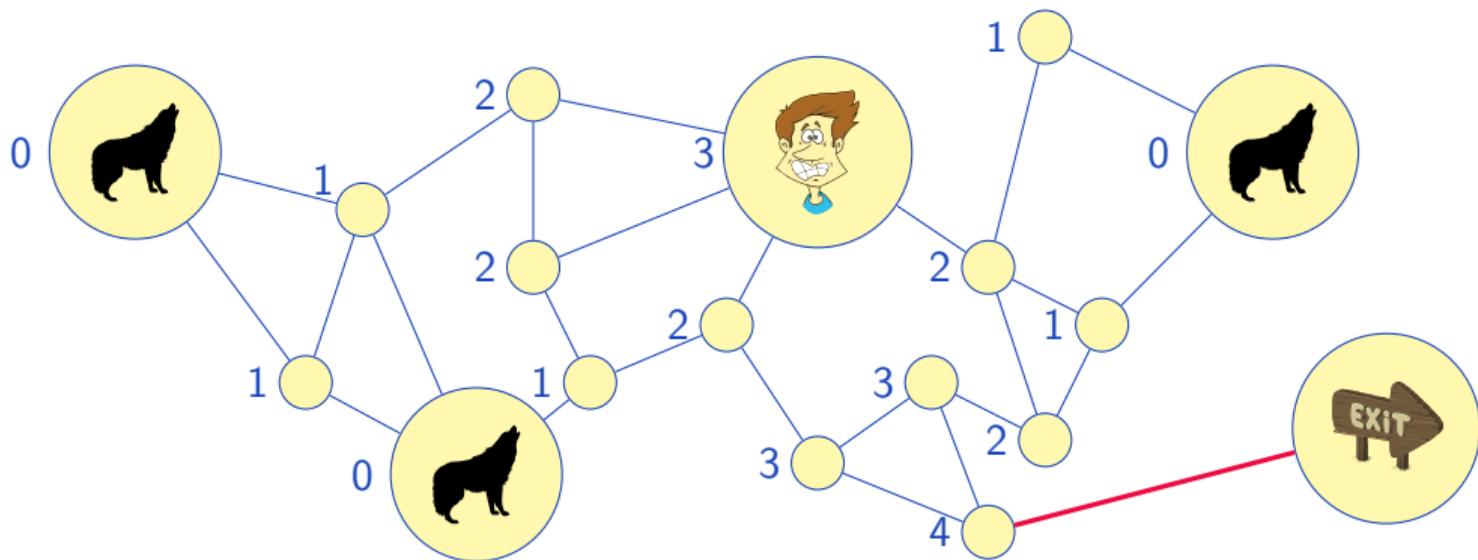
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



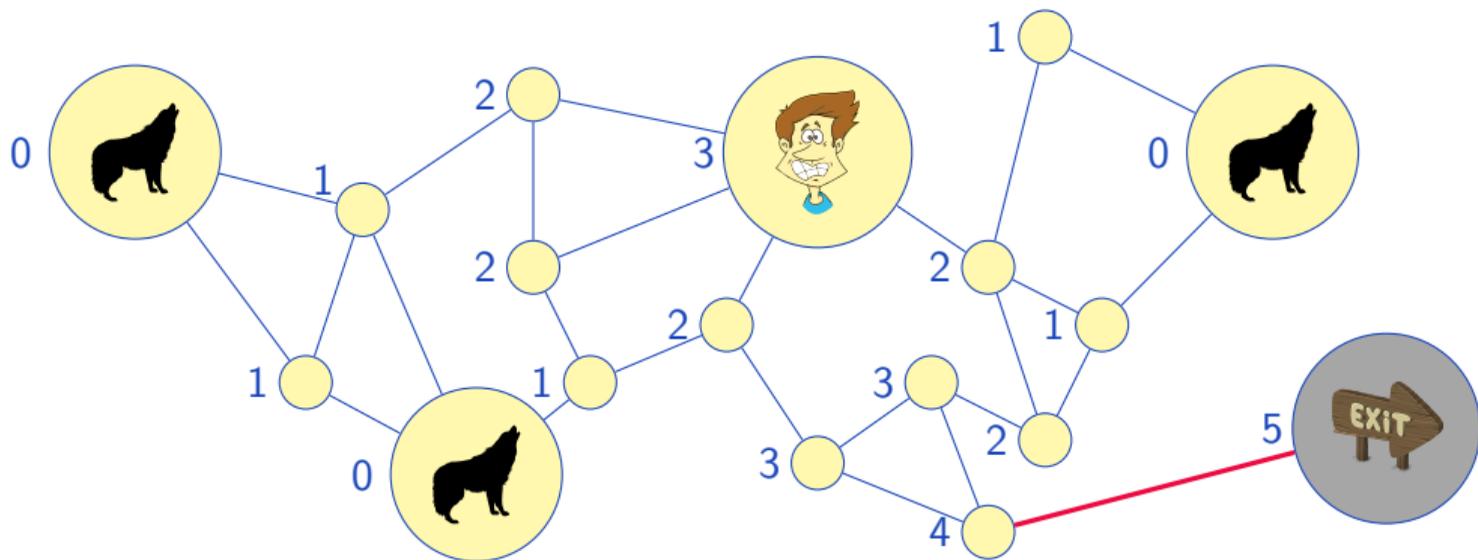
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



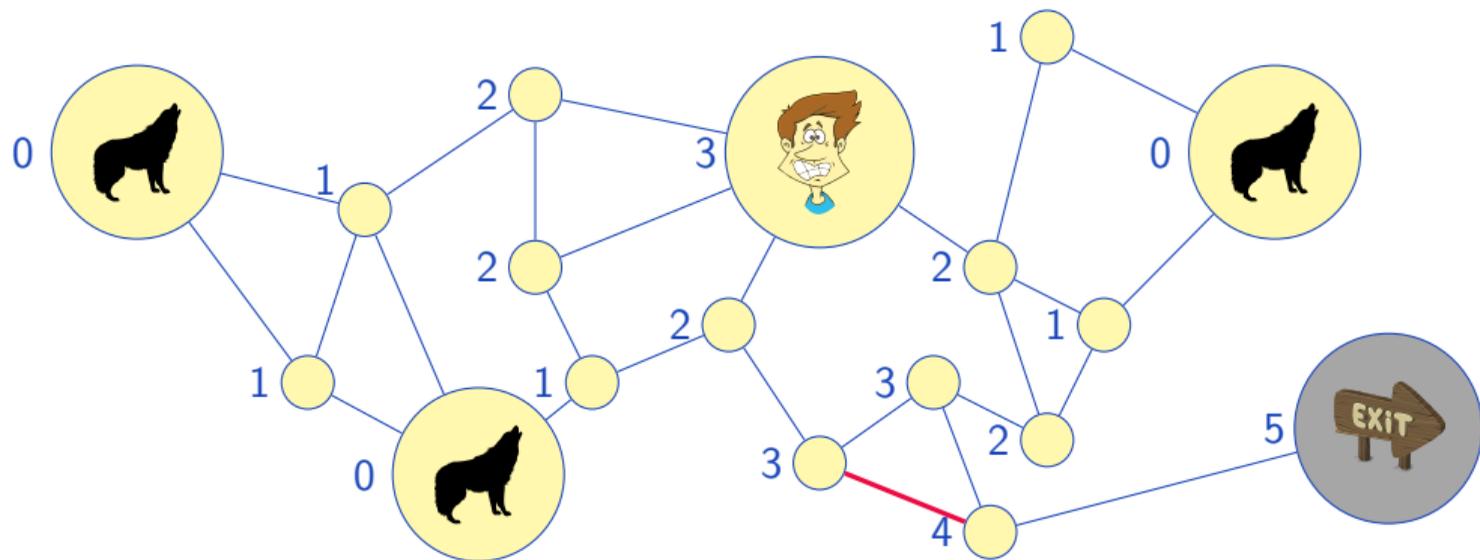
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



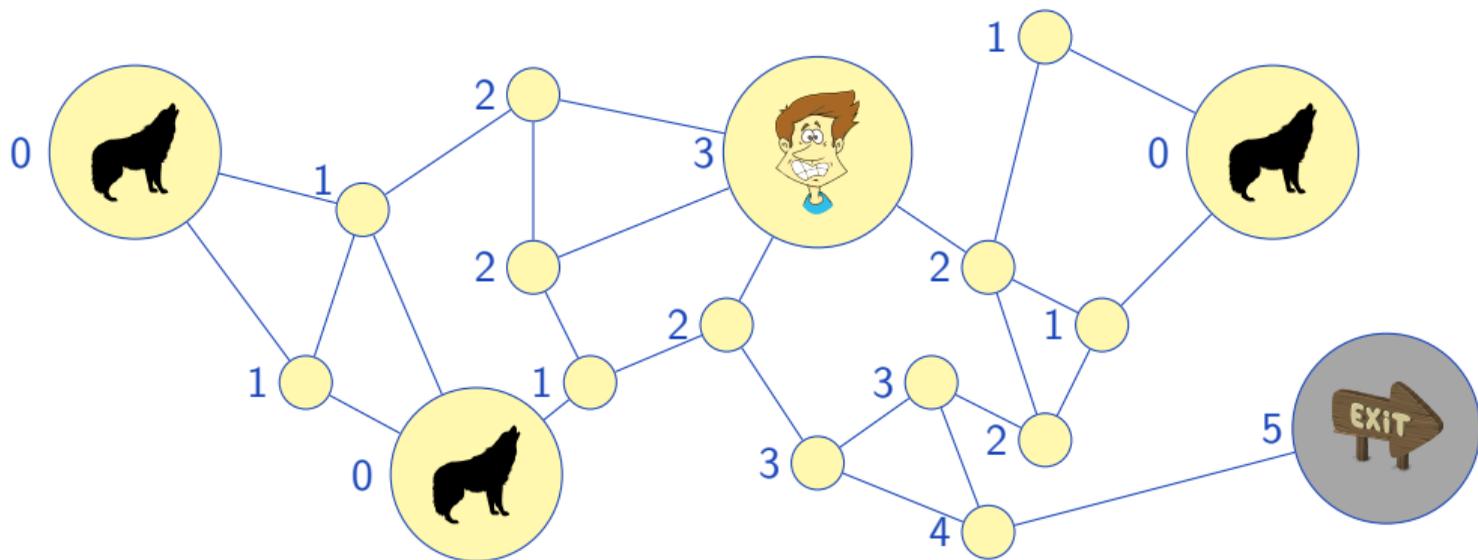
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



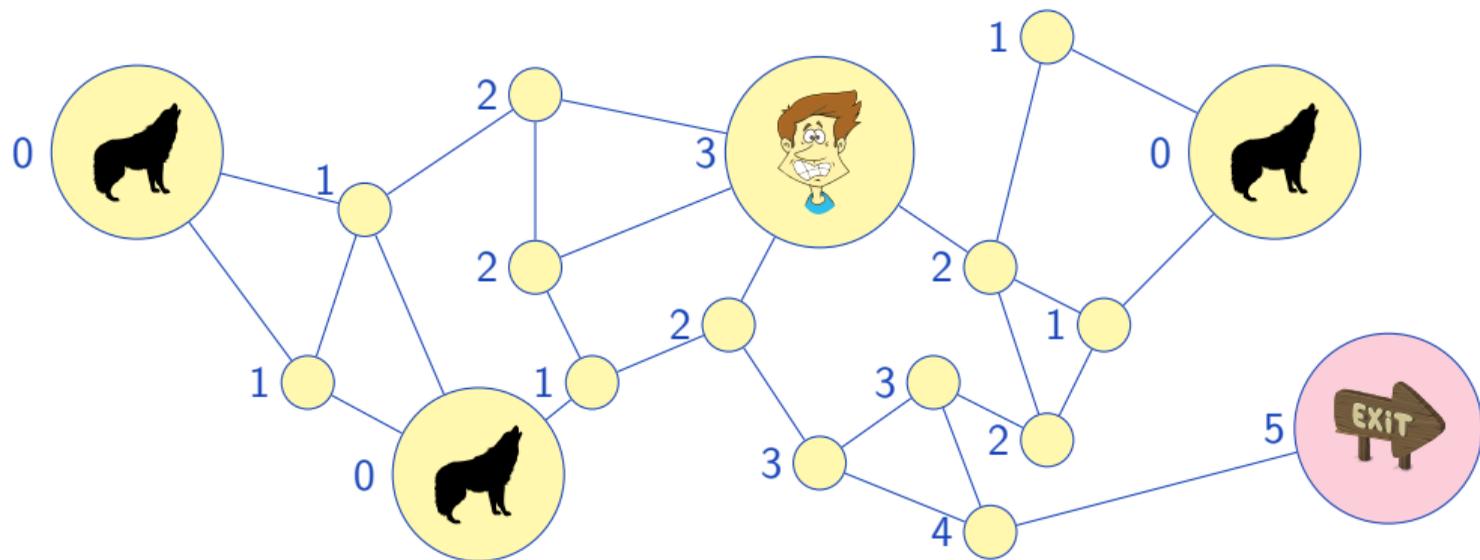
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



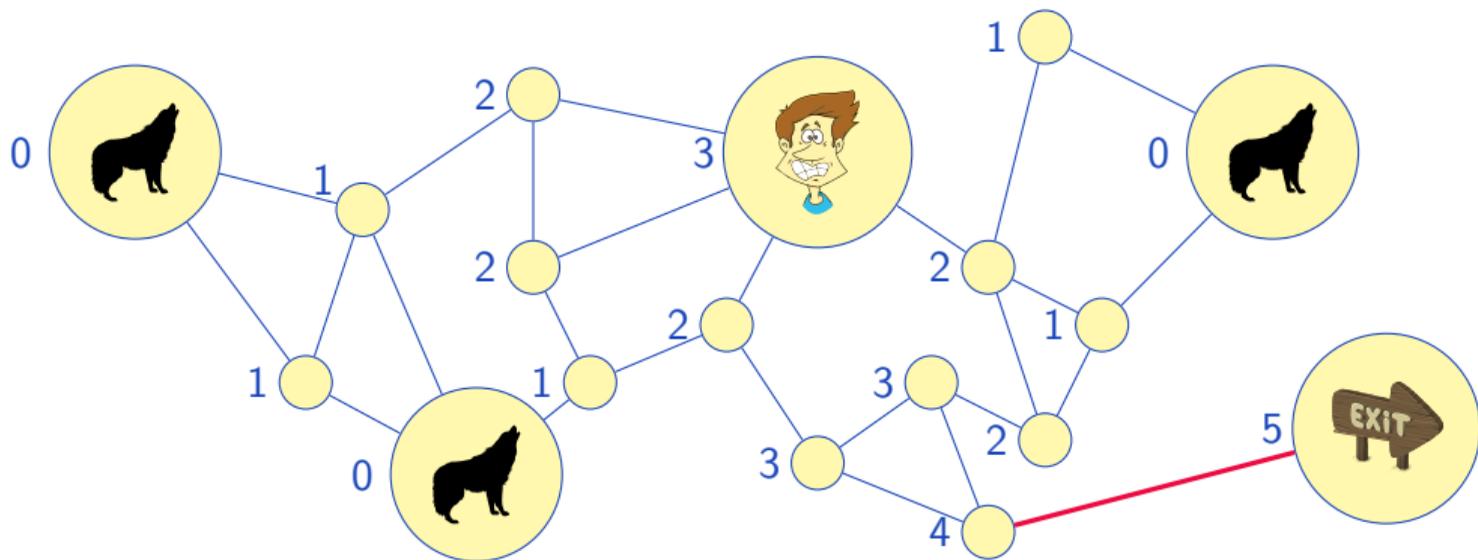
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



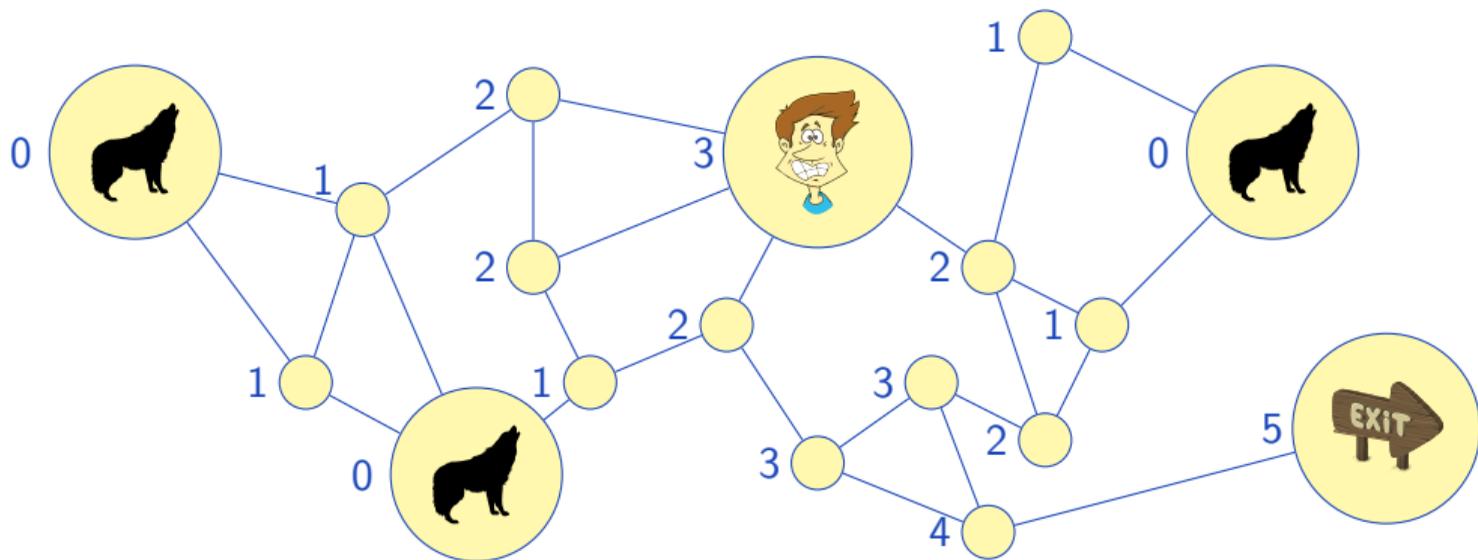
Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

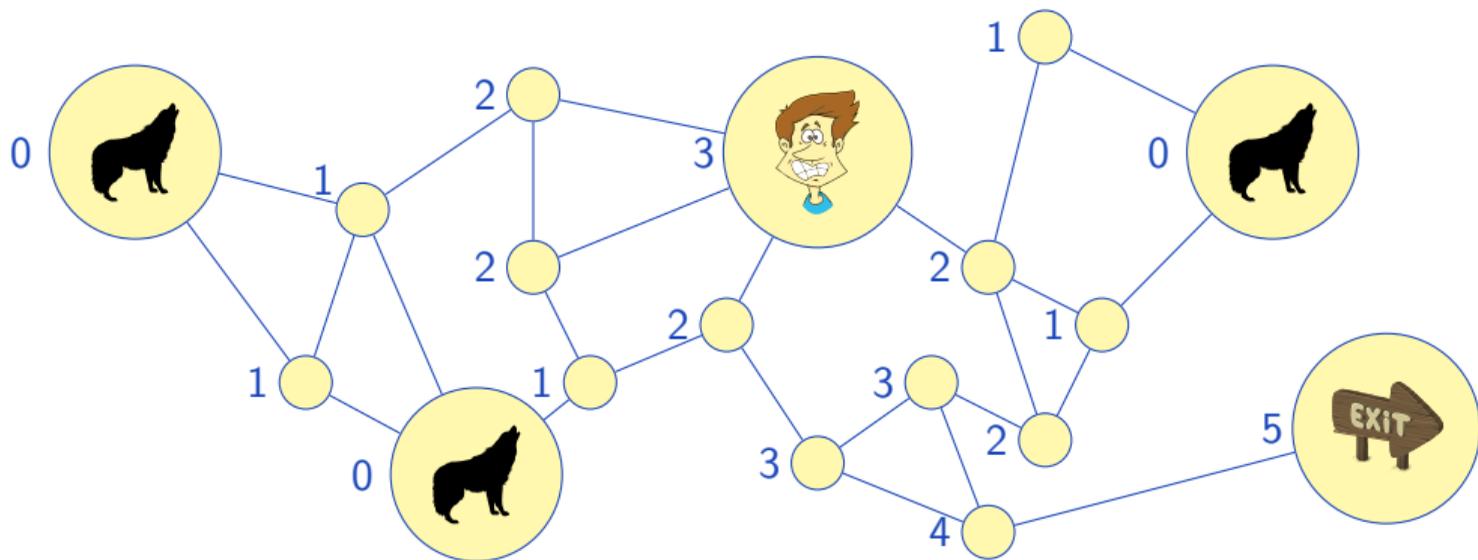
You are in the forest, and there are wolves.
How to get out alive? Use Multiple Source BFS!



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

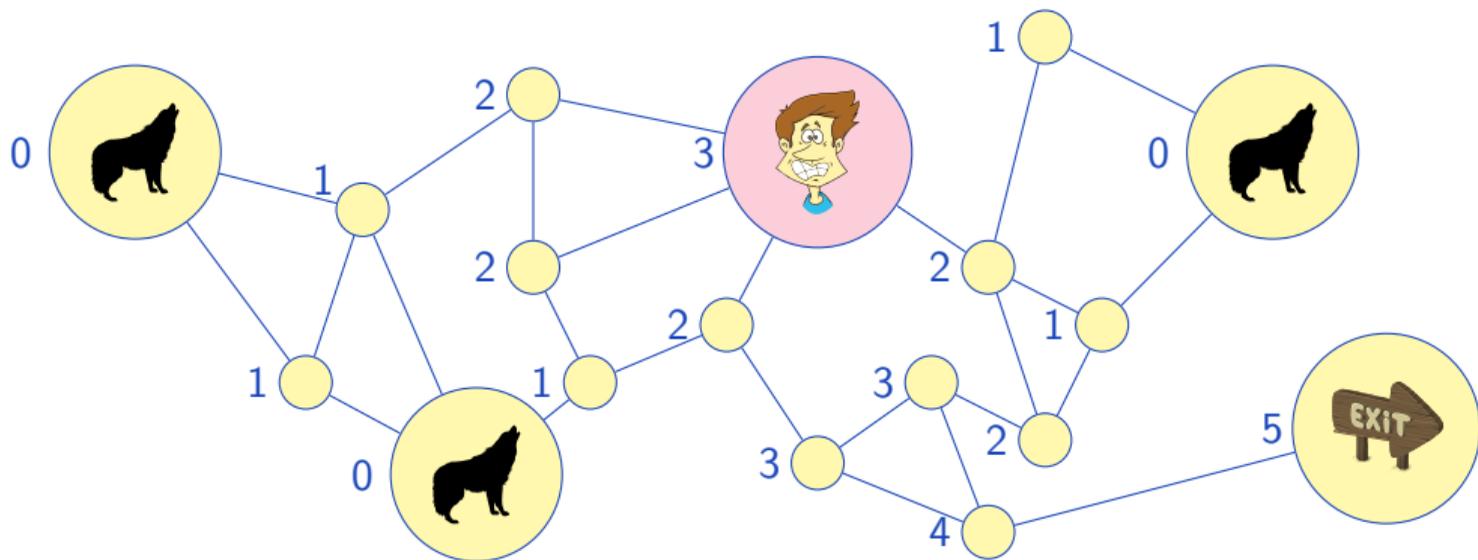
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

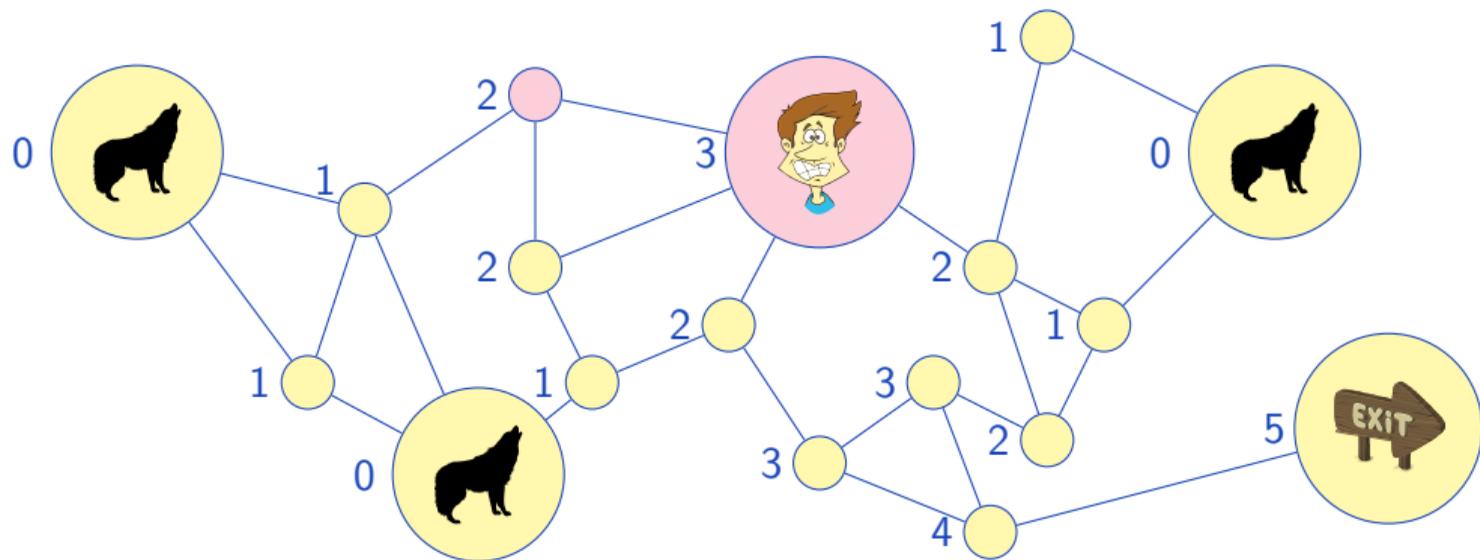
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

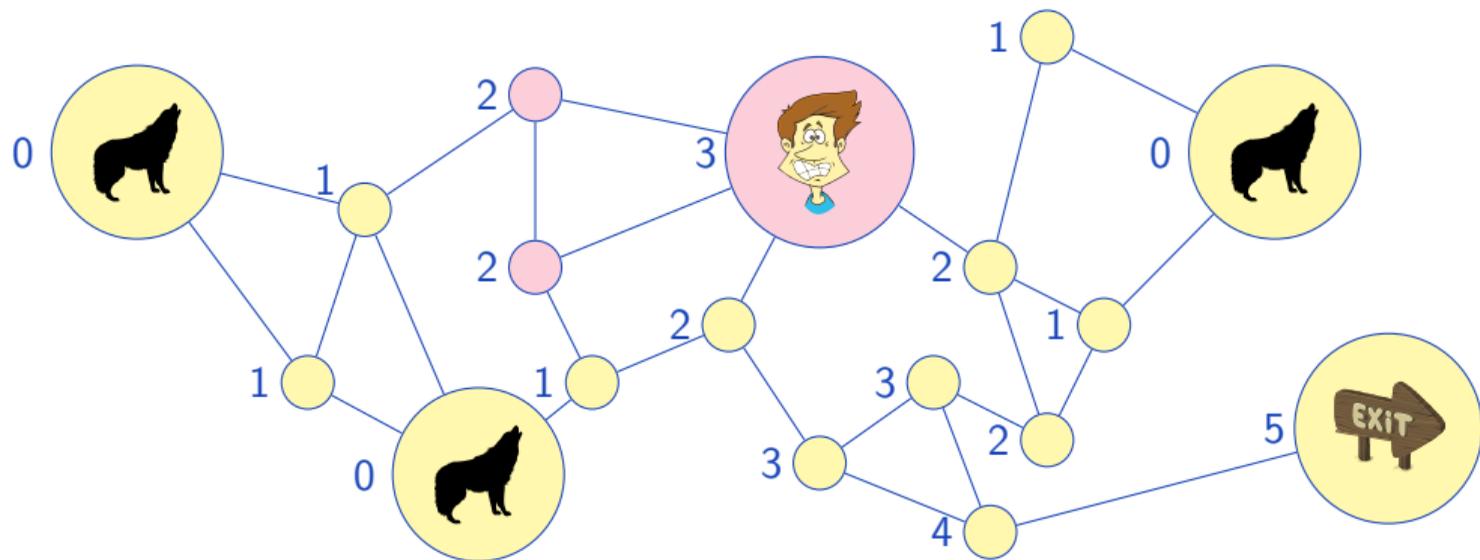
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

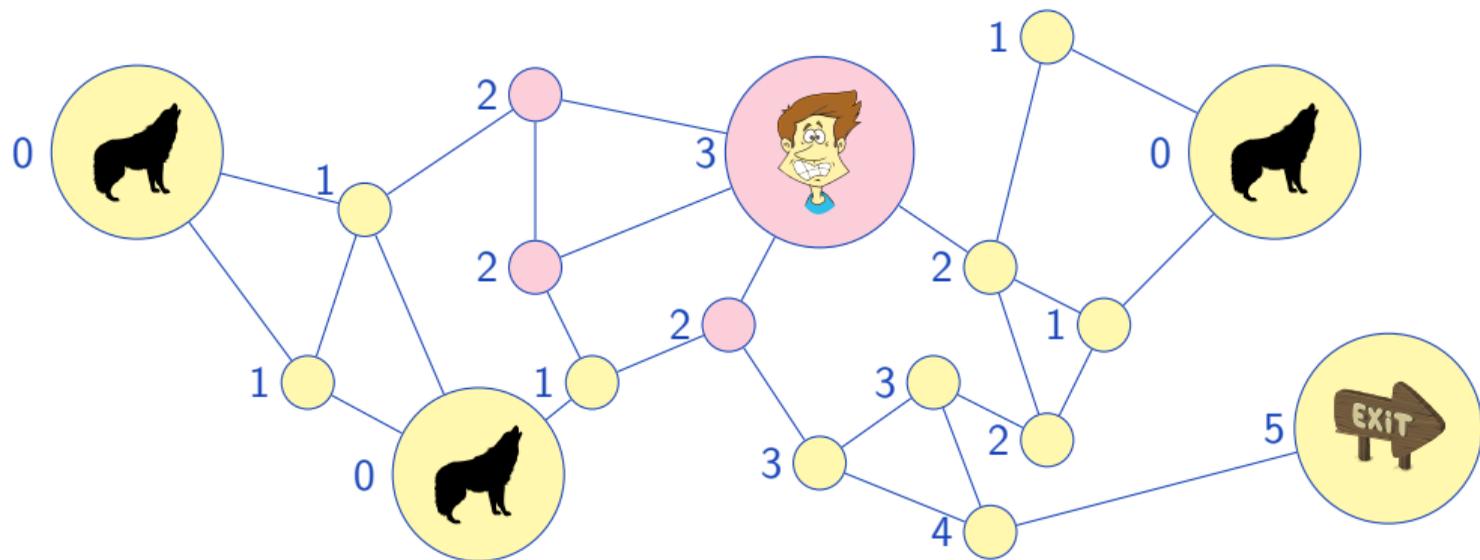
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

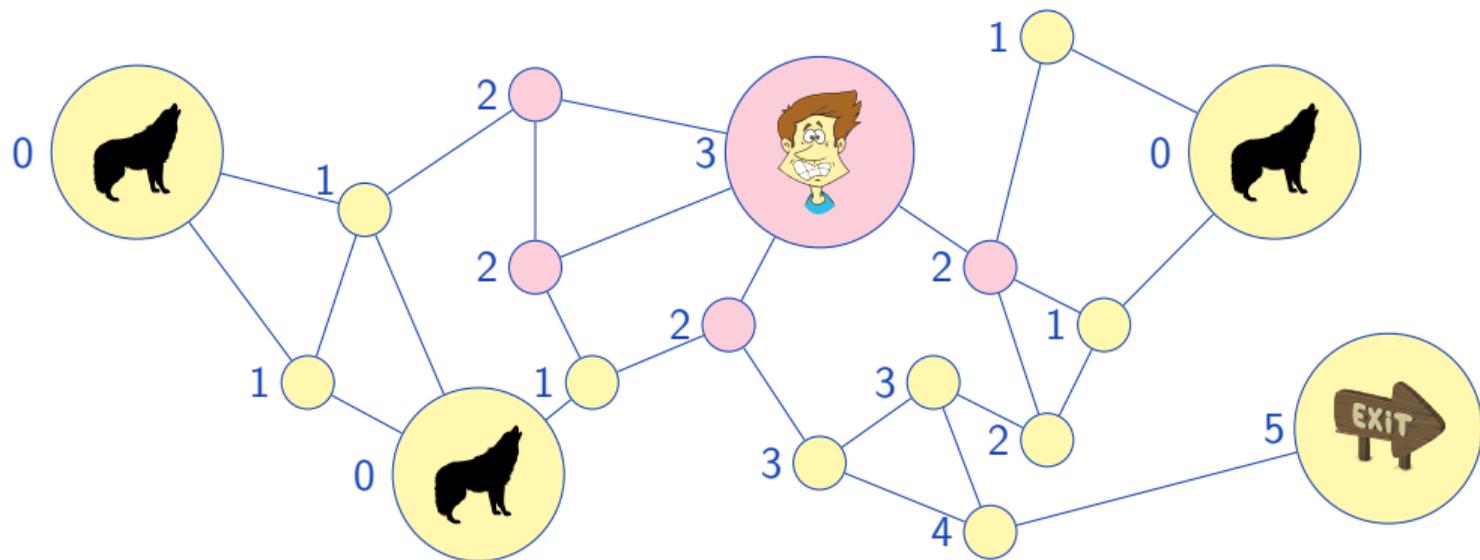
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

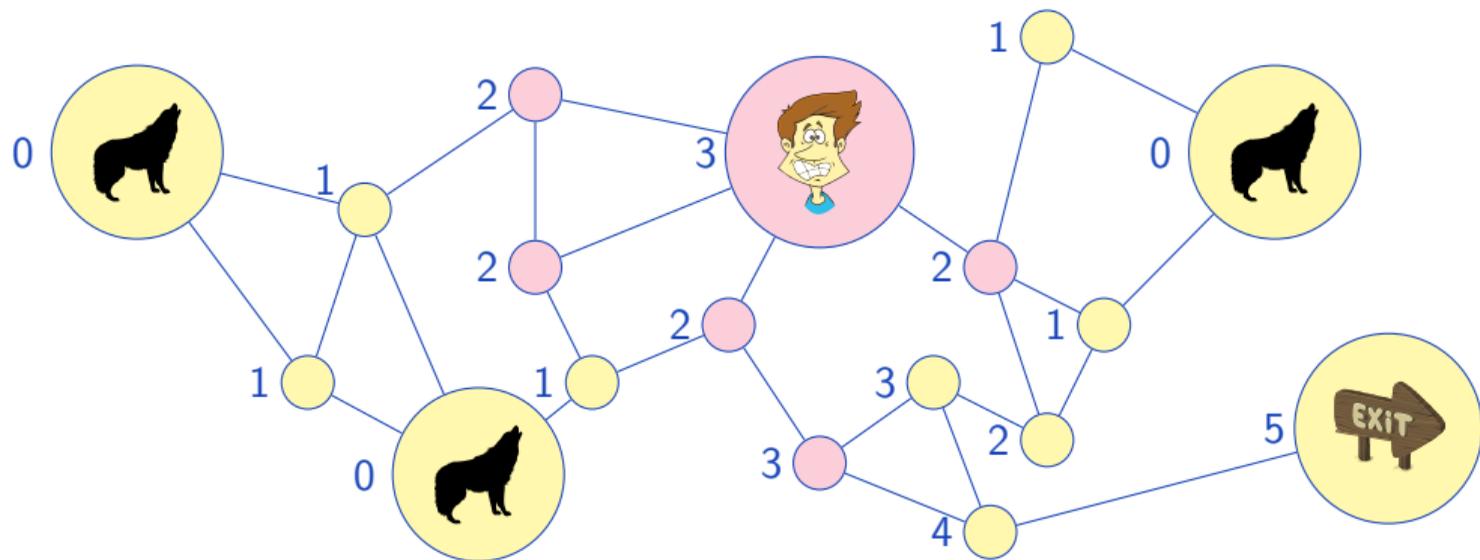
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

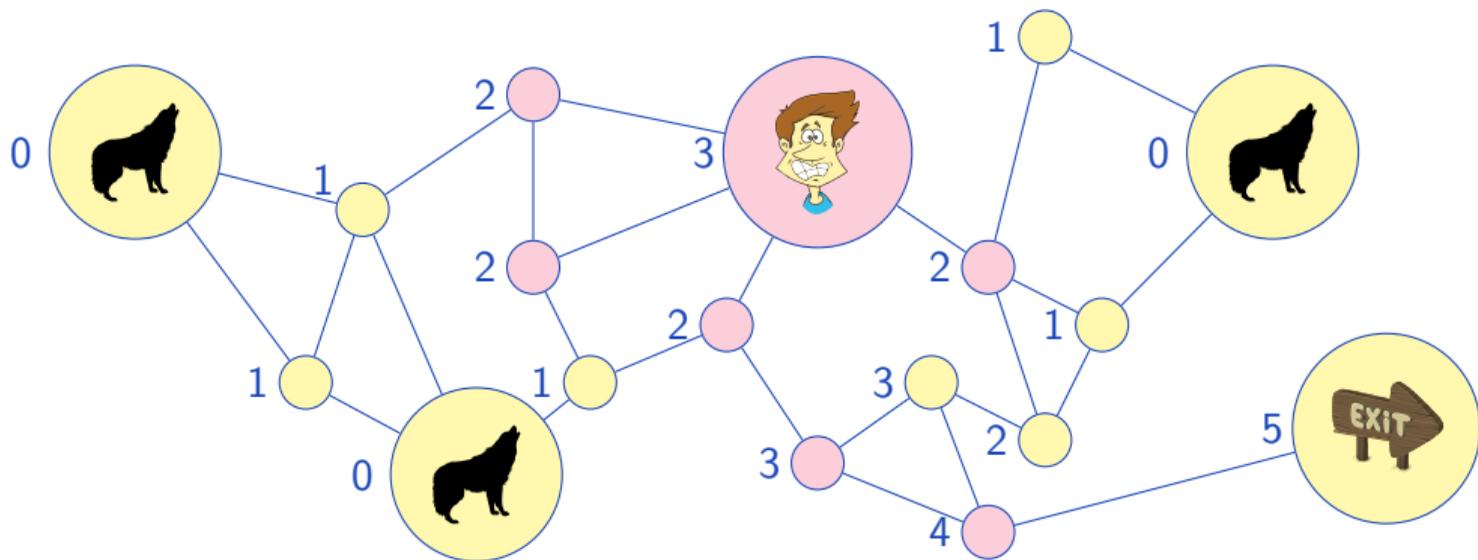
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

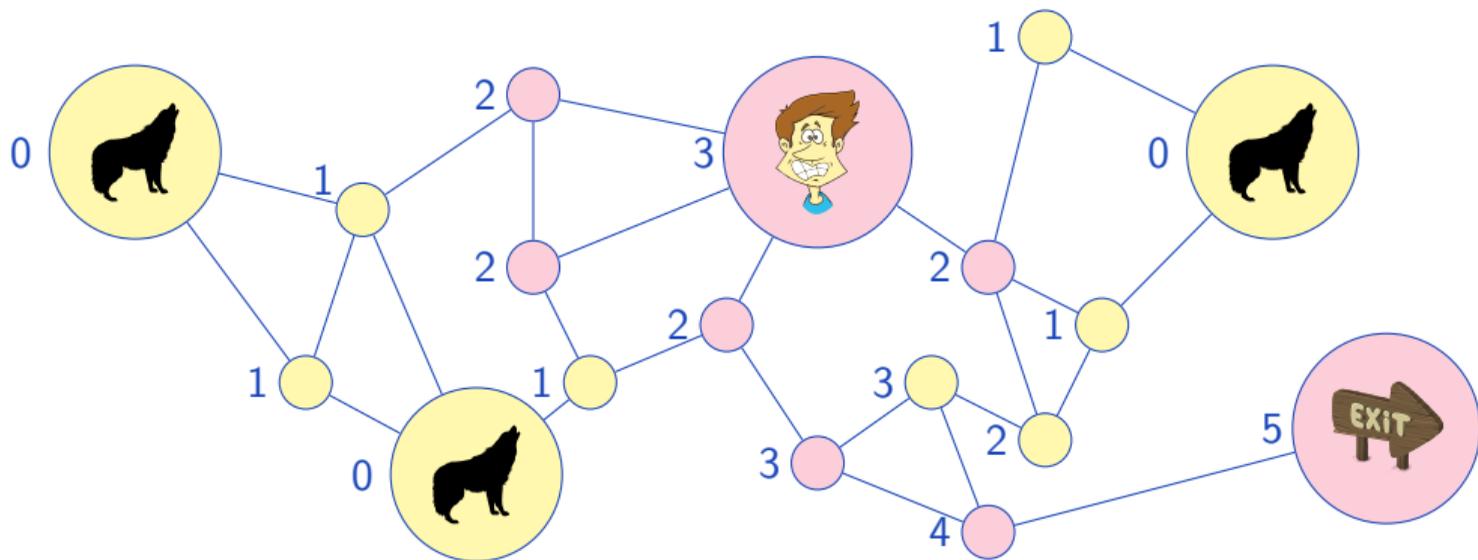
How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**



Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

You are in the forest, and there are wolves.

How to get out alive? Use Multiple Source BFS! **And another BFS to save yourself**

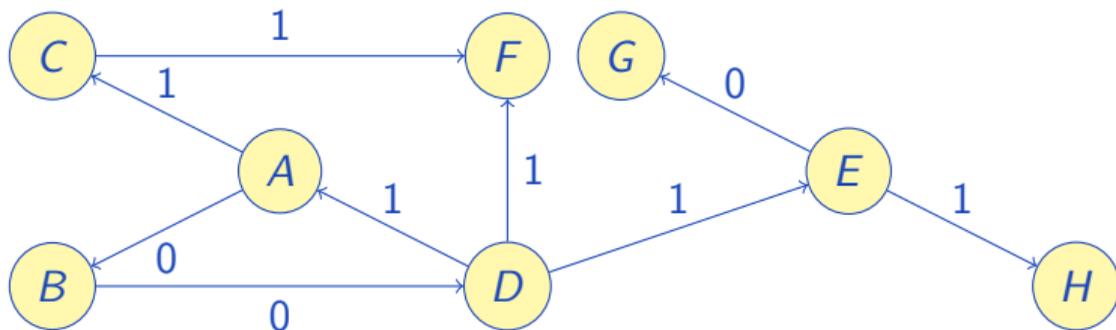


Images are courtesy of GDJ, knollbaco, glitch from openclipart.org. License: CC Zero 1.0

0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

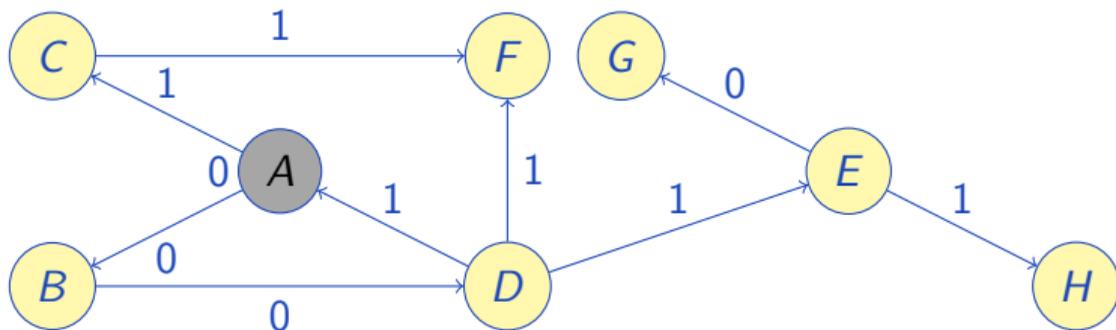
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

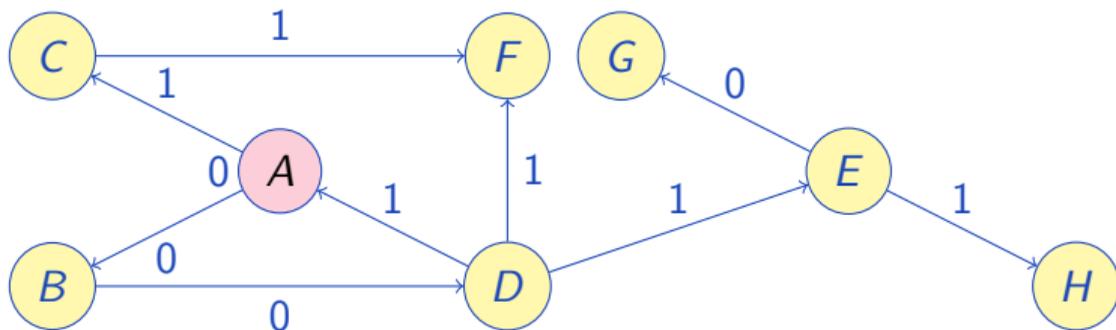
Queue: [A]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

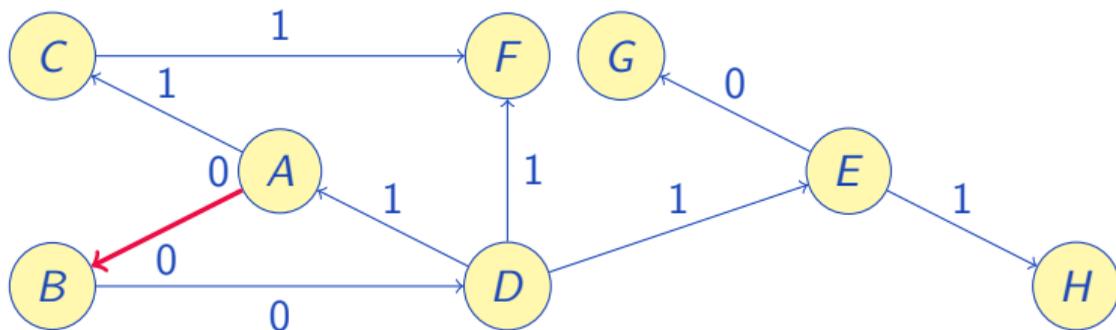
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

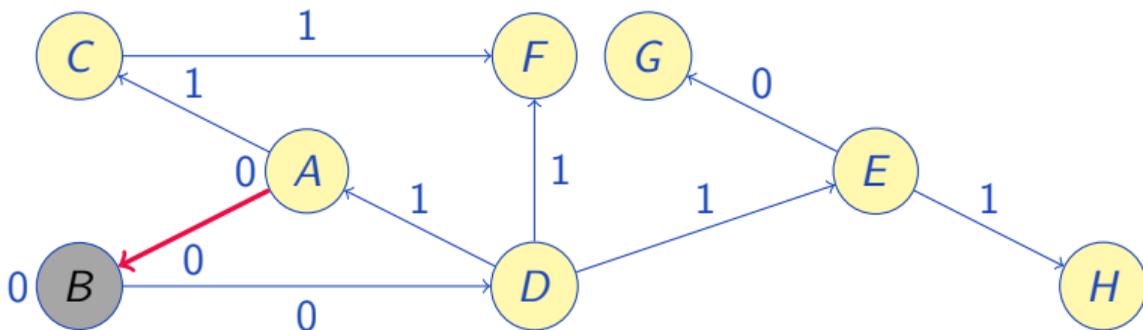
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

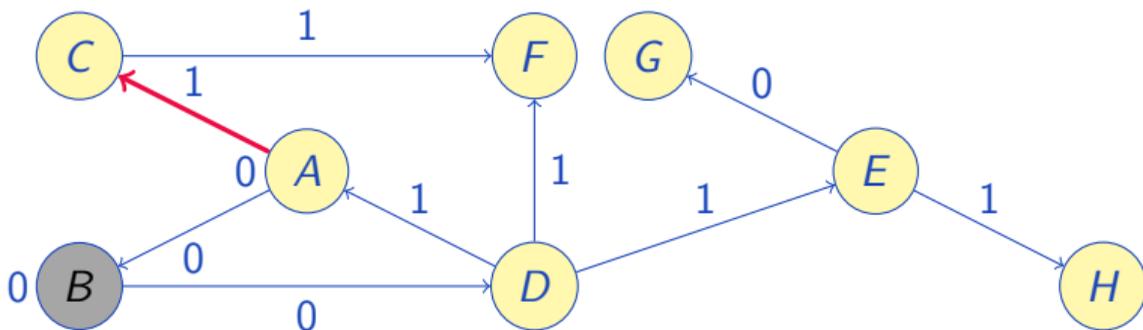
Queue: [B]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

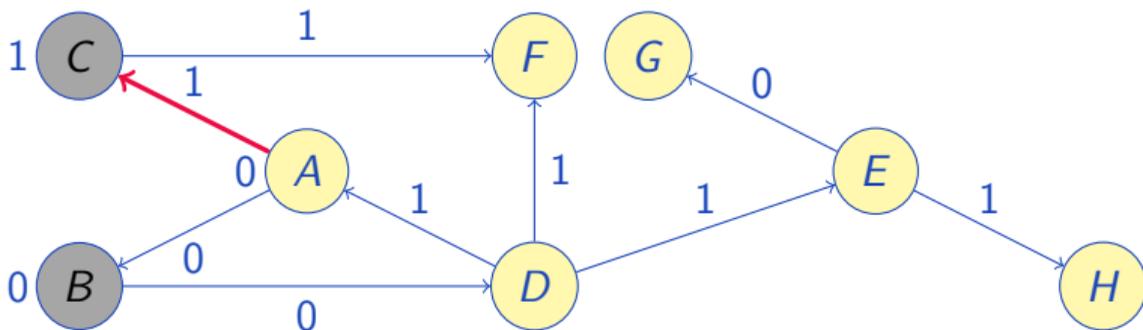
Queue: [B]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

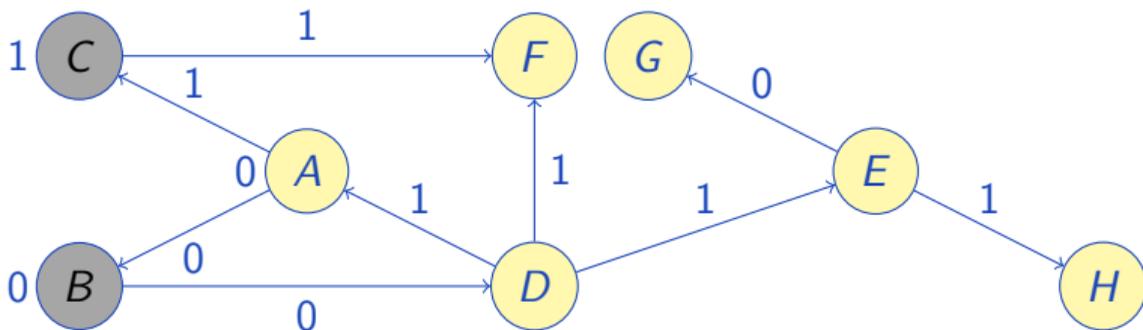
Queue: [CB]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

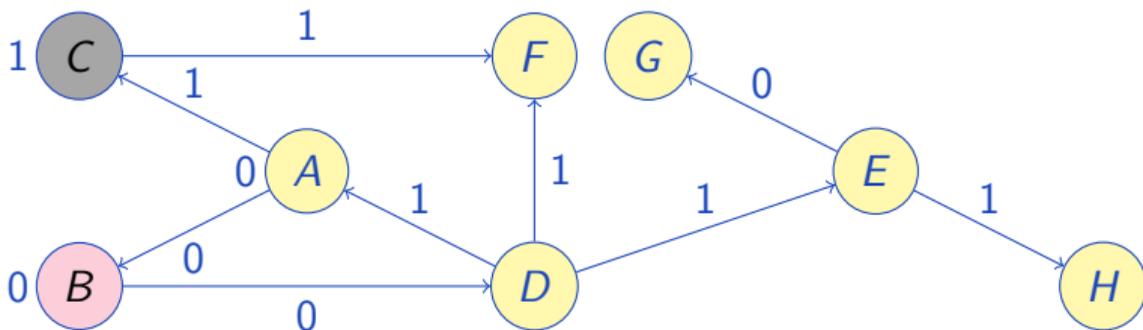
Queue: [CB]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

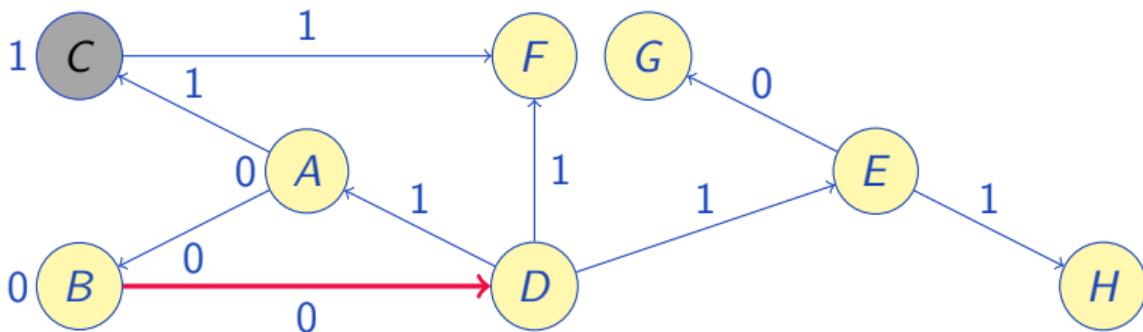
Queue: [C]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

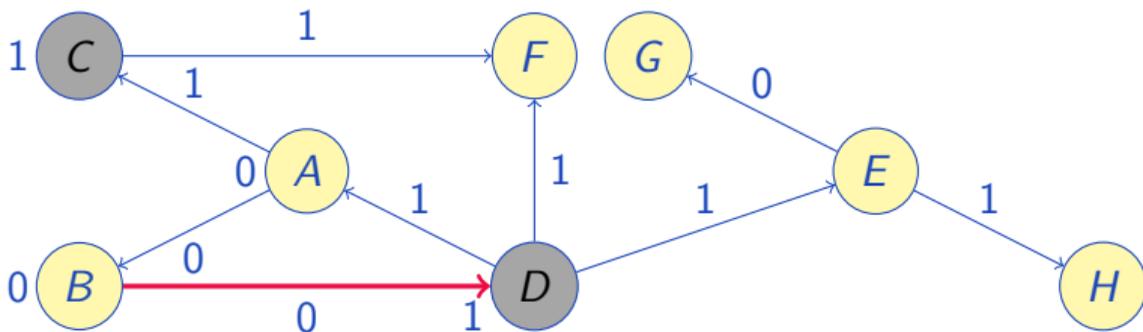
Queue: [C]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

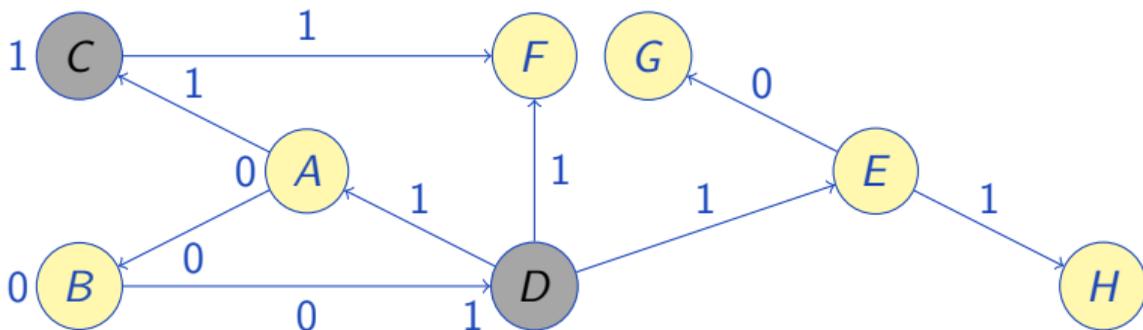
Queue: [CD]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

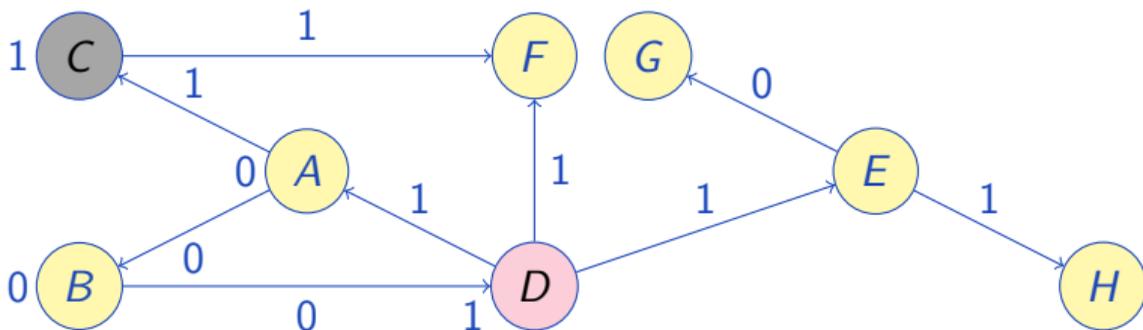
Queue: [CD]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

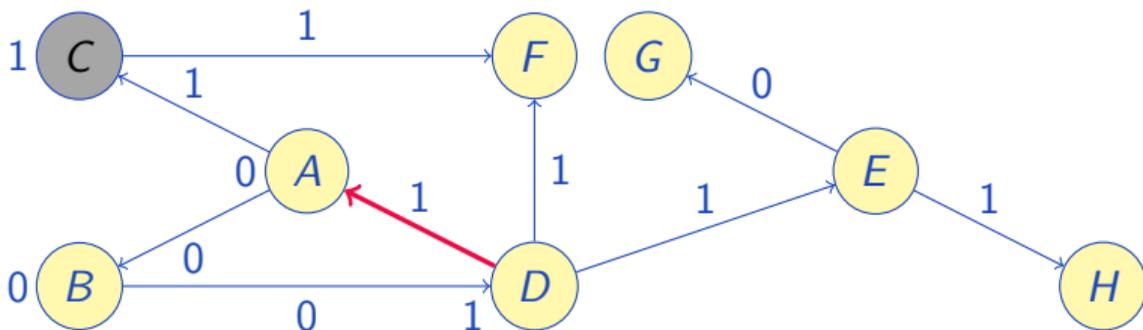
Queue: [C]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

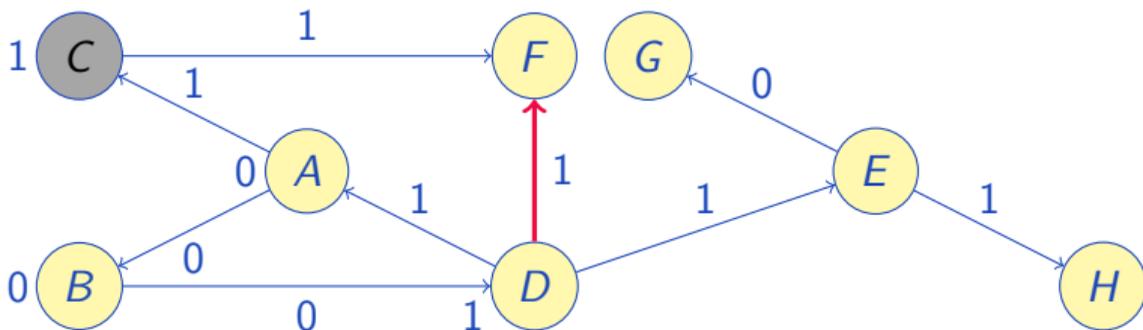
Queue: [C]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

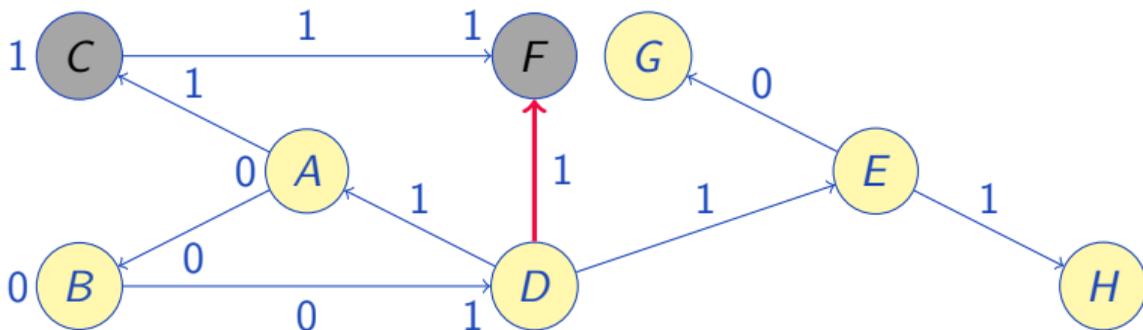
Queue: [C]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

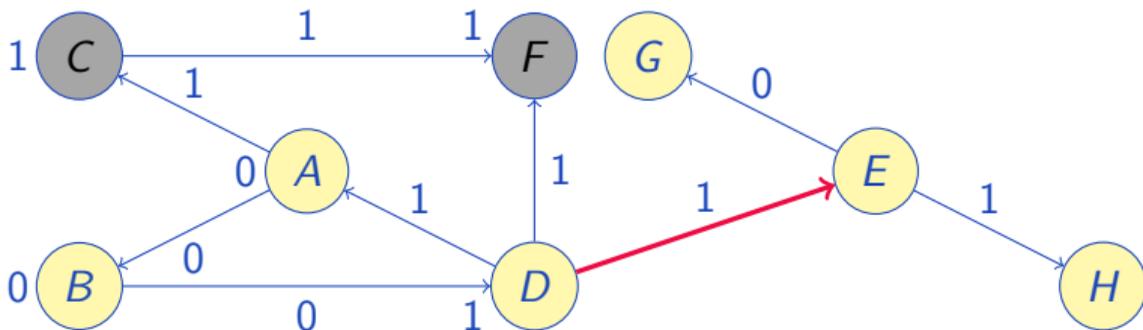
Queue: [FC]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

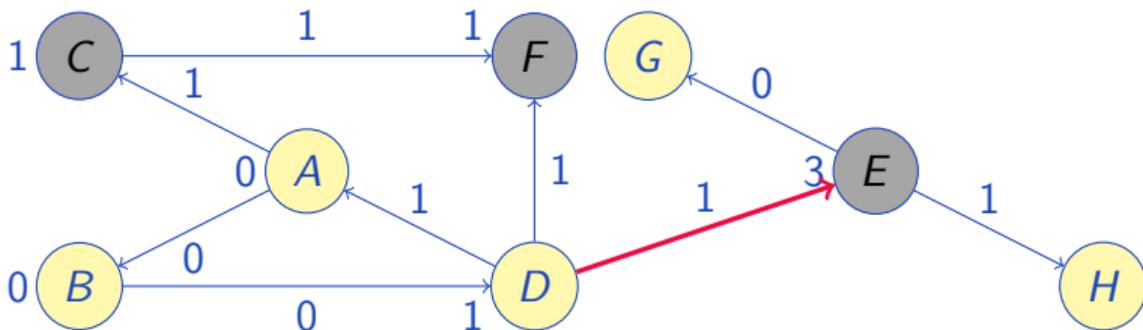
Queue: [FC]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

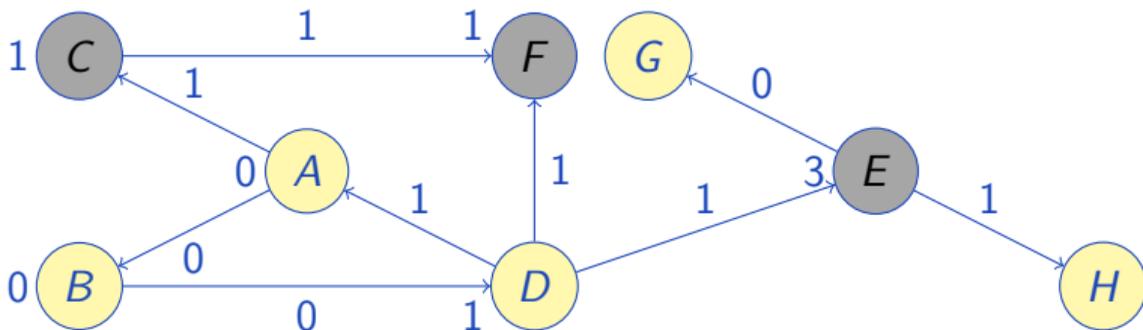
Queue: [EFC]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

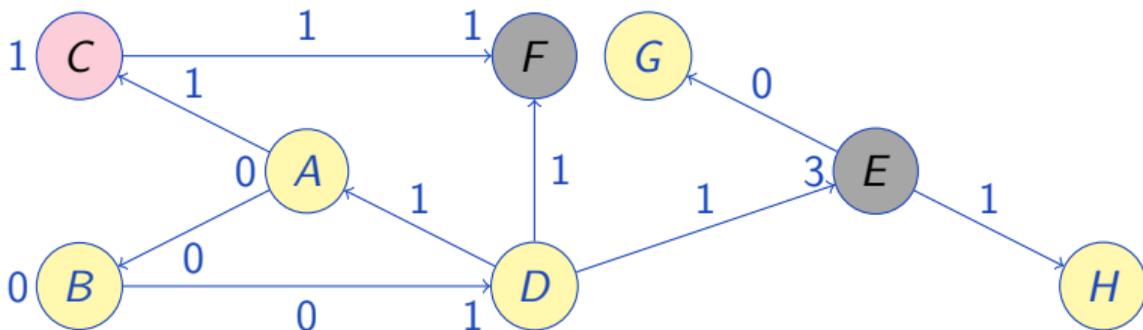
Queue: [EFC]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

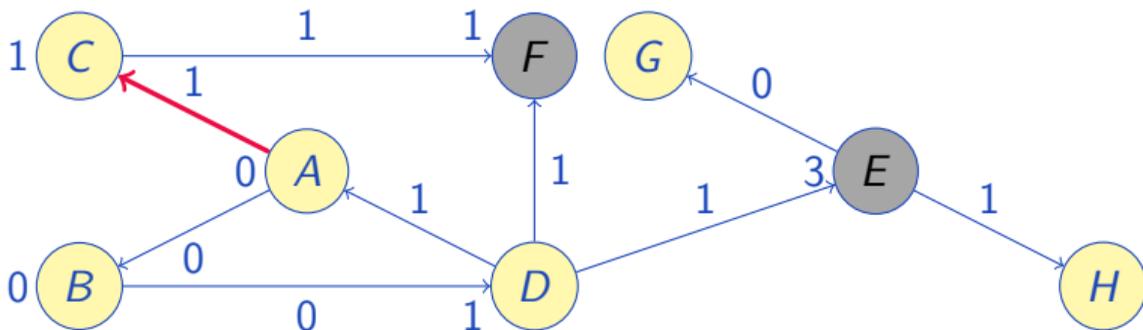
Queue: [EF]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

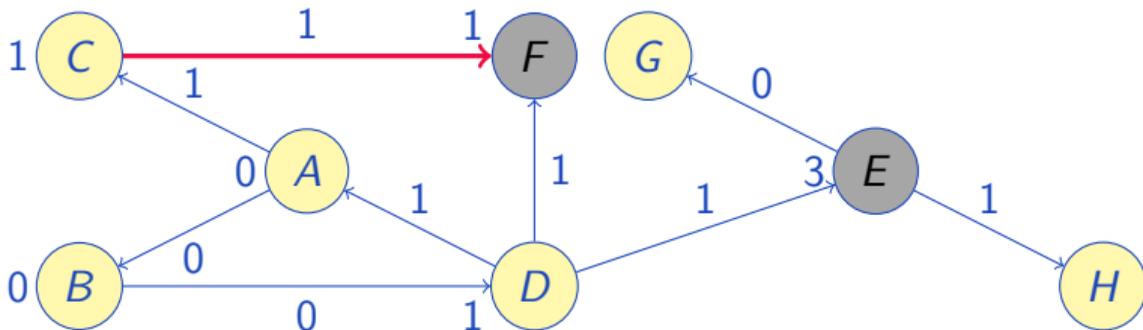
Queue: [EF]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

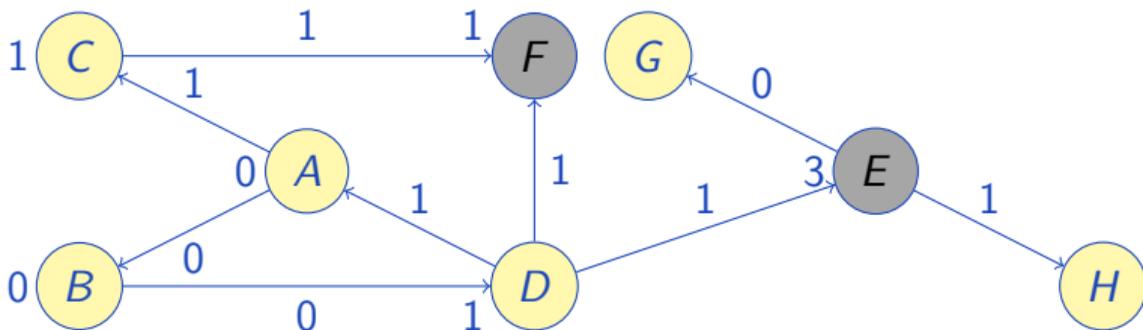
Queue: [EF]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

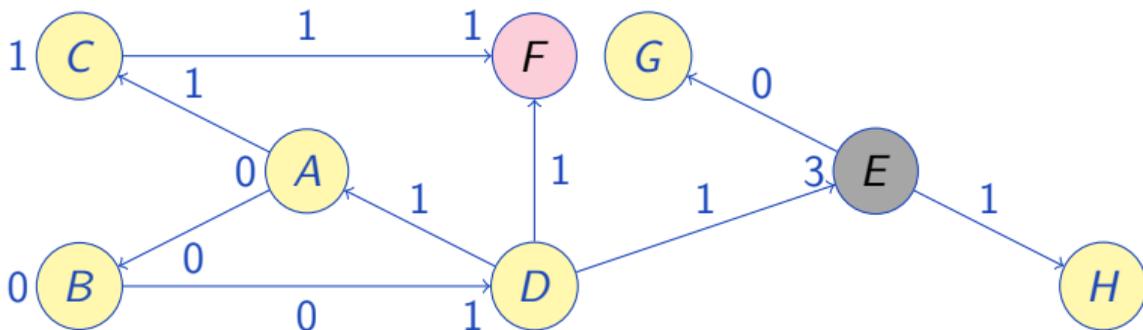
Queue: [EF]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

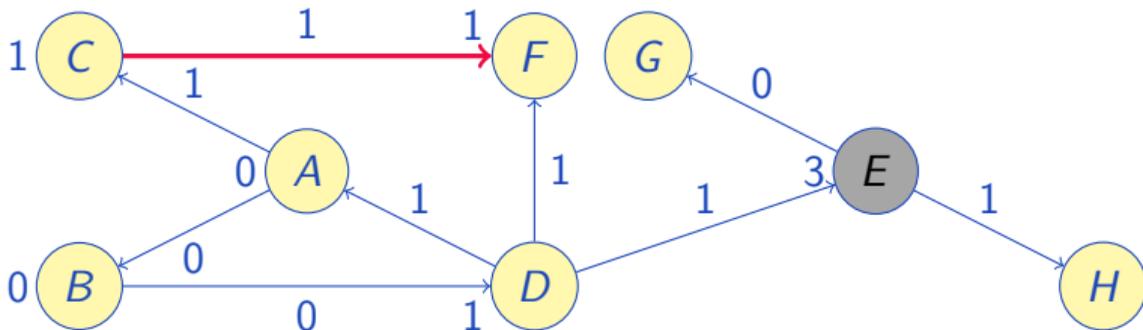
Queue: [E]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

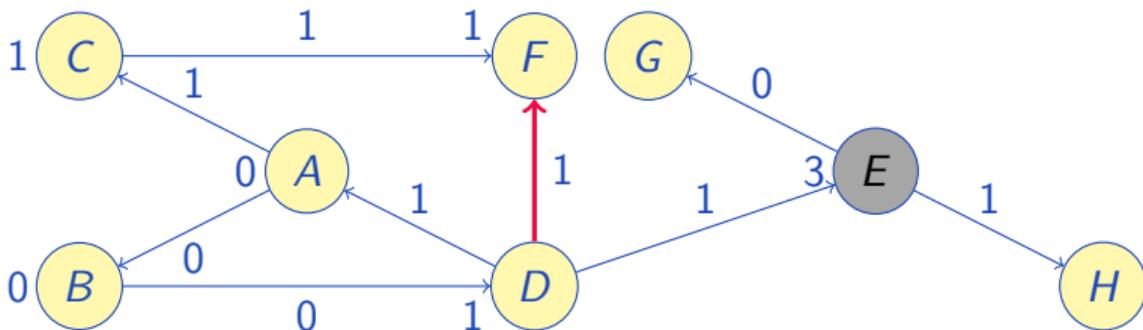
Queue: [E]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

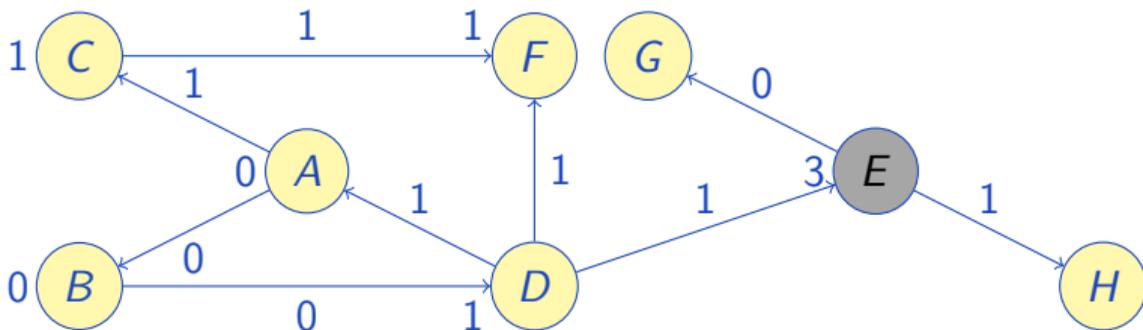
Queue: [E]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

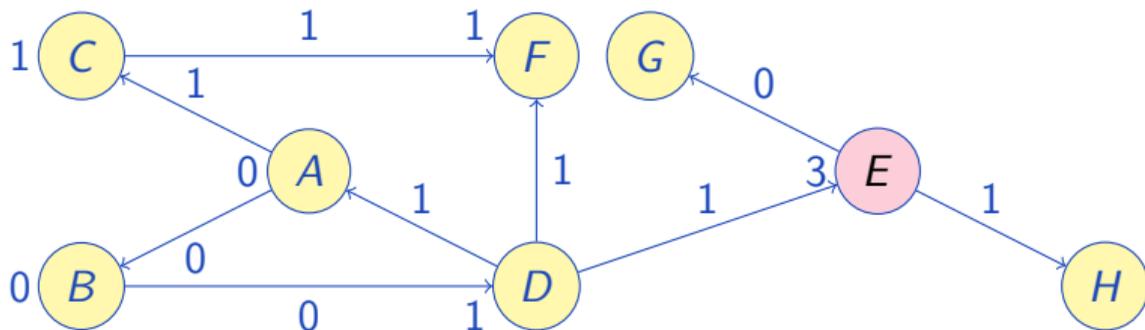
Queue: [E]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

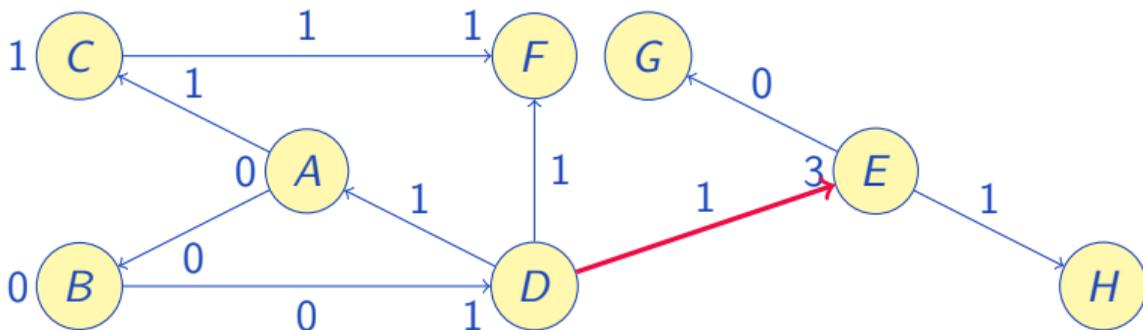
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

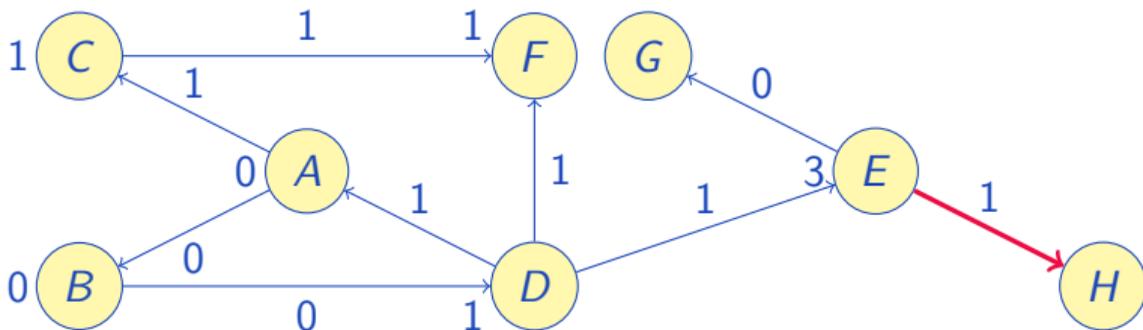
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

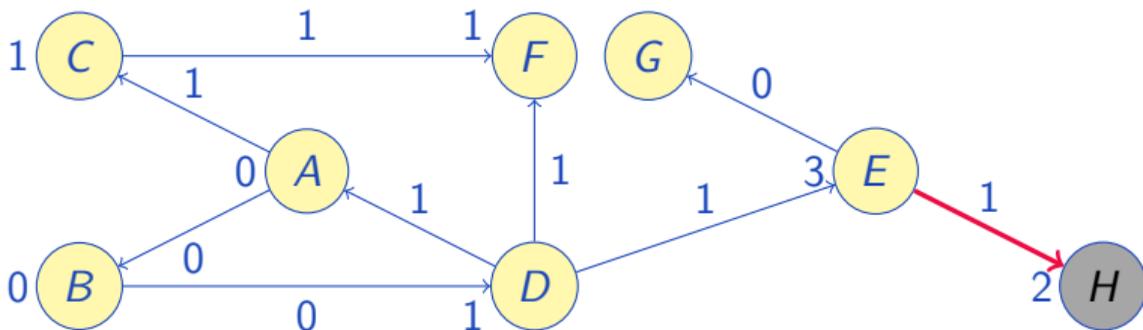
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

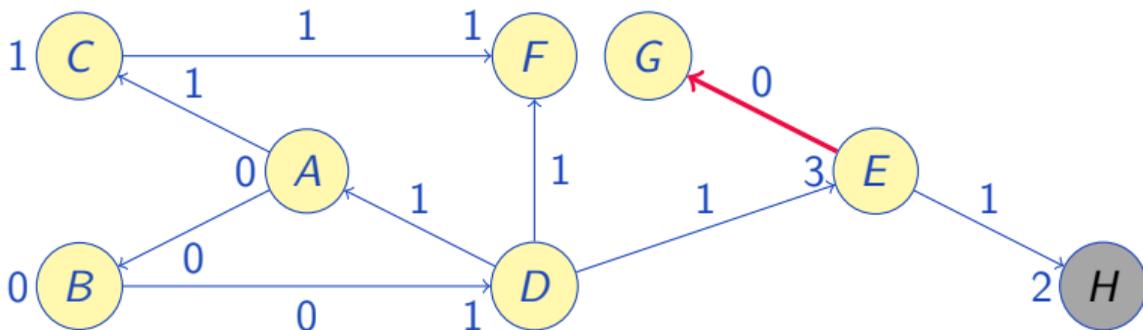
Queue: [**H**]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

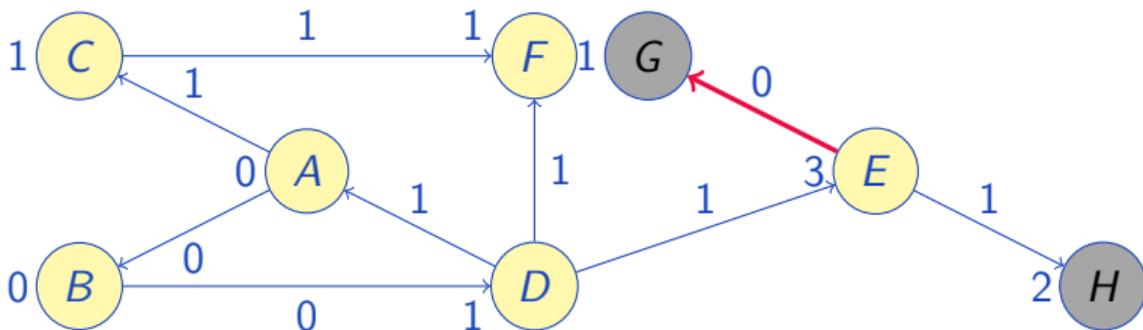
Queue: [**H**]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

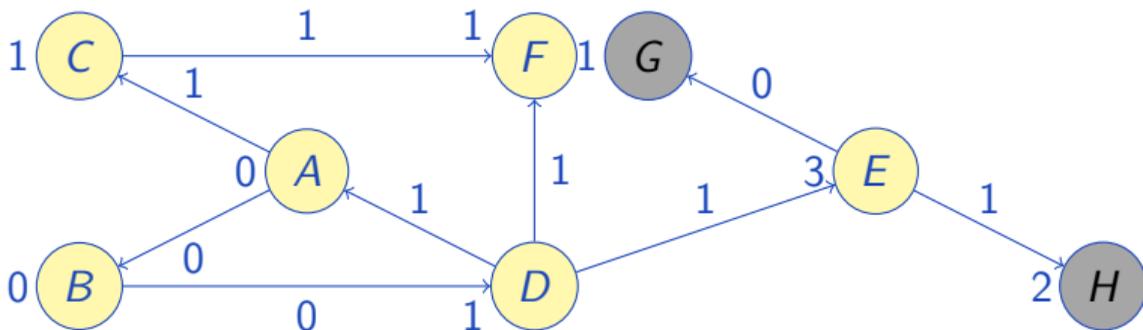
Queue: [HG]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

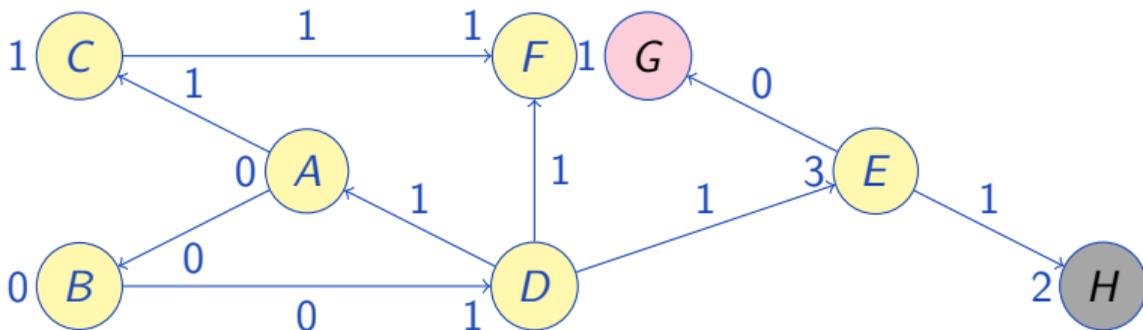
Queue: [HG]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

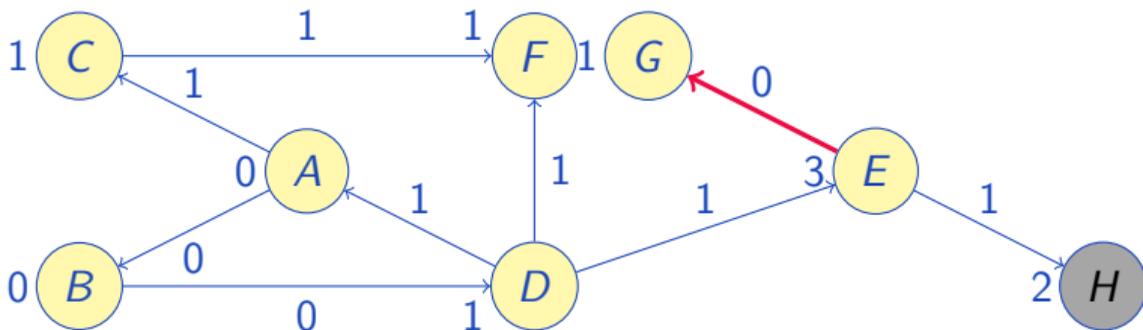
Queue: [**H**]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

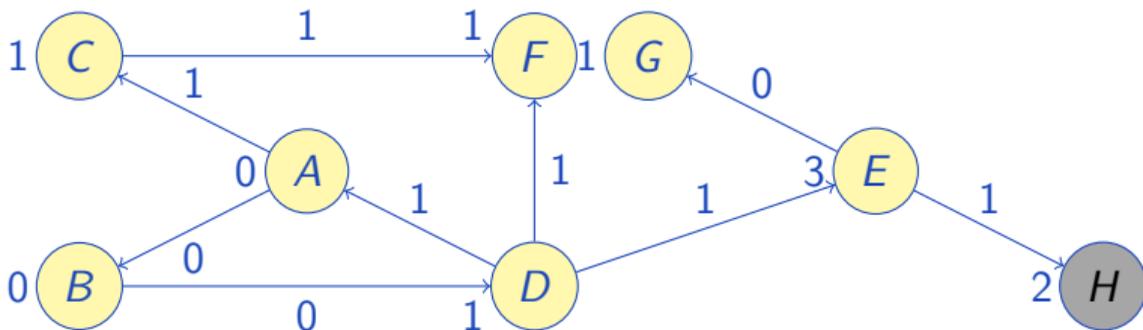
Queue: [**H**]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

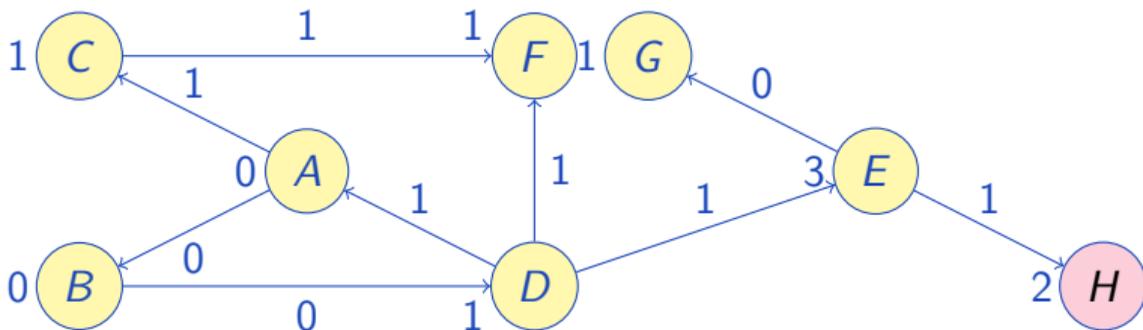
Queue: [**H**]



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

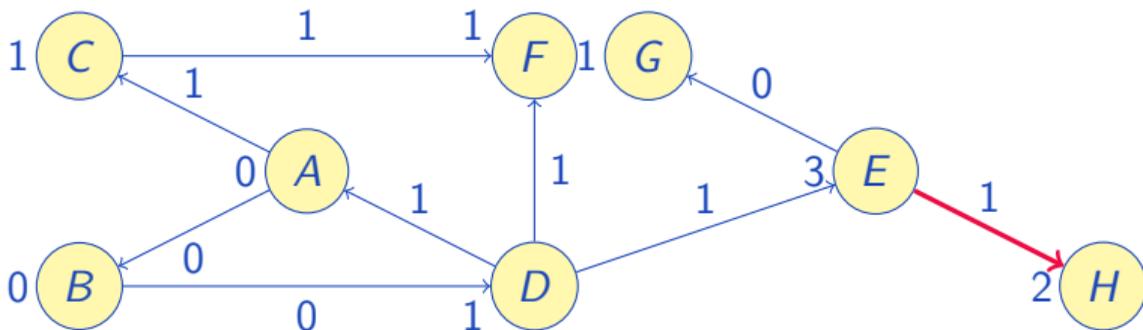
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

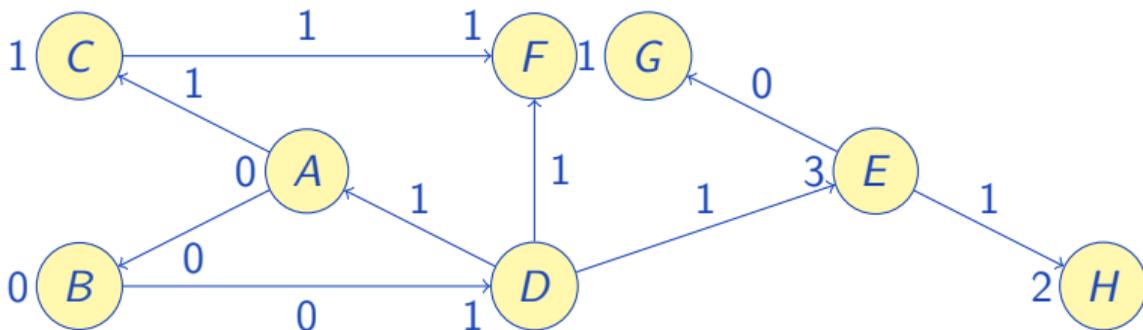
Queue: []



0-1 BFS: an extension to Breadth First Search to support edge lengths of 0 and 1

- ▶ For edge length 1, push the vertex to the **head** of the queue
- ▶ For edge length 0, push the vertex to the **tail** of the queue
- ▶ ... assuming the vertices are popped from the **tail**

Queue: []



procedure BFS0K(V, E, K, v_0)

$A(v) = \{(u, |(v, u)|) \mid (v, u) \in E\}$

$Q \leftarrow (K + 1) \cdot []$

$D \leftarrow \{\infty\}$

$d \leftarrow 0, \Sigma \leftarrow 0$

$D[v_0] \leftarrow 0, \text{PUSH}(Q[0], v_0), \Sigma \leftarrow \Sigma + 1$

while $\Sigma > 0$ **do**

while $\text{ISEMPTY}(Q[d \bmod (K + 1)])$ **do** $d \leftarrow d + 1$ **end while**

$v \leftarrow \text{POP}(Q[d \bmod (K + 1)]), \Sigma \leftarrow \Sigma - 1$

for $(u, s) \leftarrow A(v)$ **do**

if $D[u] > d + s$ **then**

if $D[u] < \infty$ **then**

$\text{REMOVE}(Q[D[u] \bmod (K + 1)], u), \Sigma \leftarrow \Sigma - 1$

end if

$D[u] \leftarrow d + s$

$\text{PUSH}(Q[(d + s) \bmod (K + 1)], u), \Sigma \leftarrow \Sigma + 1$

end if

end for

end while

end procedure

▷ Supports integer edge lengths in $[0; K]$

▷ The adjacency list with edge lengths

▷ $K + 1$ queues – or maybe sets

▷ Distances from v_0

▷ Current distance and sum of size of queues

▷ Initialize v_0

▷ Find next length to work with

▷ Pop the current vertex

▷ Check all outgoing edges

▷ If shorter path found...

▷ If already was there...

▷ Remove from the queue (in $O(1)$)

▷ Update the length

▷ Push into the right queue