ITMO UNIVERSITY

# How to Win Coding Competitions: Secrets of Champions

## Week 5: Algorithms on Graphs 1
## Lecture 3: Introduction to Depth First Search

Maxim Buzdalov
Saint Petersburg 2016

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them

How to check if the given graph is connected?

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!
- If there is a path from $a$ to every other vertex, then the graph is connected

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!
- If there is a path from $a$ to every other vertex, then the graph is connected

Idea 2: Solve the one-to-all problem

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them

How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!
- If there is a path from $a$ to every other vertex, then the graph is connected

Idea 2: Solve the one-to-all problem

- Traverse the graph, starting from some vertex

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!
- If there is a path from $a$ to every other vertex, then the graph is connected

Idea 2: Solve the one-to-all problem

- Traverse the graph, starting from some vertex
- Required property: if you visit a vertex, you also visit all adjacent vertices

Recall: an undirected graph is connected if for every pair of vertices $a$ and $b$ there is a path between them
How to check if the given graph is connected?

Idea 1: Reduce the all-to-all problem to one-to-all problem

- If there is a path from $a$ and $b$, and between $b$ and $c$, then there is a path between $a$ and $c$
  - Just concatenate these paths!
- If there is a path from $a$ to every other vertex, then the graph is connected

Idea 2: Solve the one-to-all problem

- Traverse the graph, starting from some vertex
- Required property: if you visit a vertex, you also visit all adjacent vertices
- Meet Depth First Search!

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $(v, u) \in E$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$
    **for** $(v, u) \in E$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $(v, u) \in E$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $(v, u) \in E$ **do** ▷ visiting all outgoing edges
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

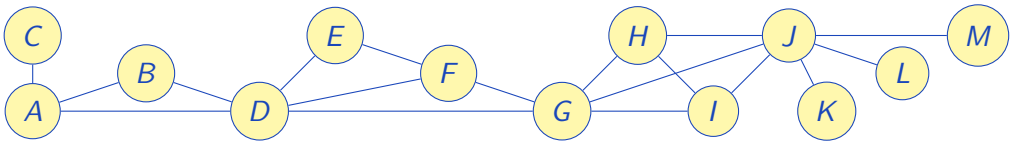
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $(v, u) \in E$ **do** ▷ visiting all outgoing edges
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**

$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices

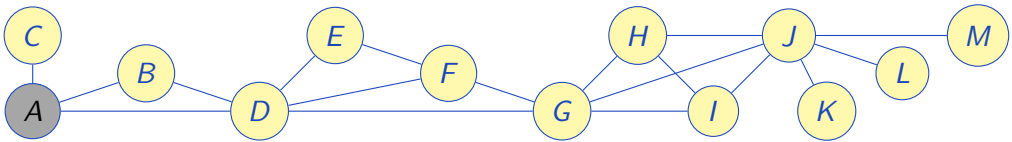
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
  $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
  **for** $u \in V$ **if** $(v, u) \in E$ **do** ▷ visiting all outgoing edges: more explicit
    **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
  **end for**
**end procedure**

```
G = ⟨V, E⟩                                  ▷ the graph
U ← ∅                                       ▷ set of visited vertices
A(v) = {u | (v, u) ∈ E}                     ▷ Adjacent vertex function: for free with adjacency lists
procedure DFS(v)                            ▷ recursive procedure, argument: current vertex
    U ← U ∪ {v}                             ▷ marking current vertex visited
    for u ∈ A(v) do                         ▷ visiting all outgoing edges: more efficient
        if u ∉ U then DFS(u) end if         ▷ if target is not visited, calling recursively
    end for
end procedure
```

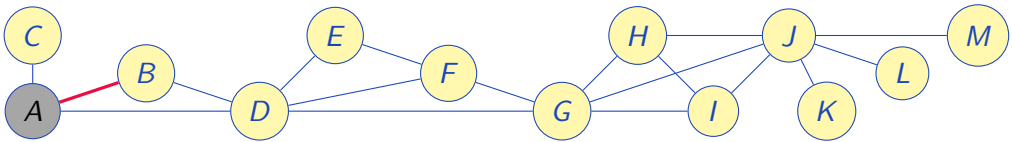
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



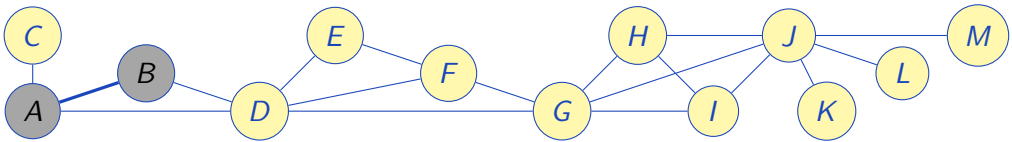
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



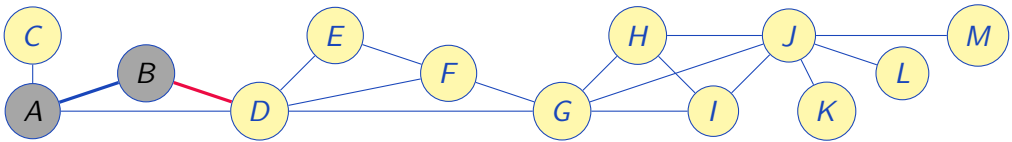
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



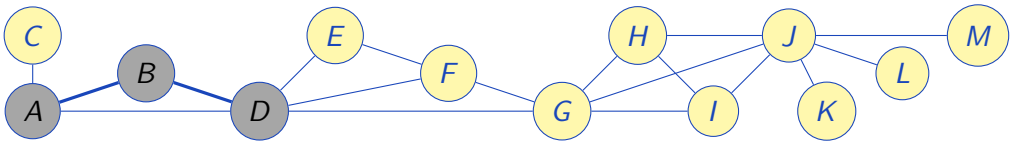
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



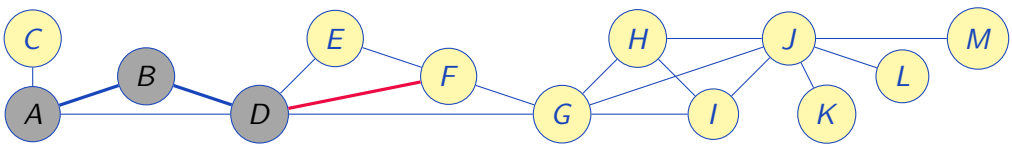
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



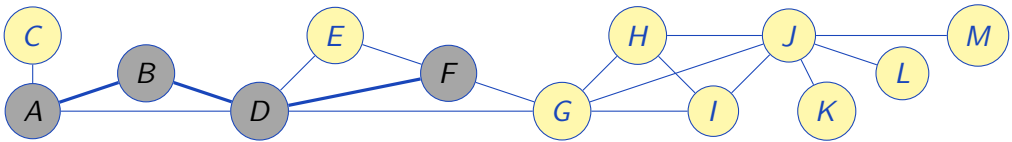
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



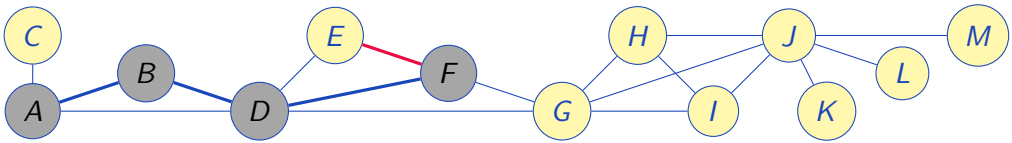
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



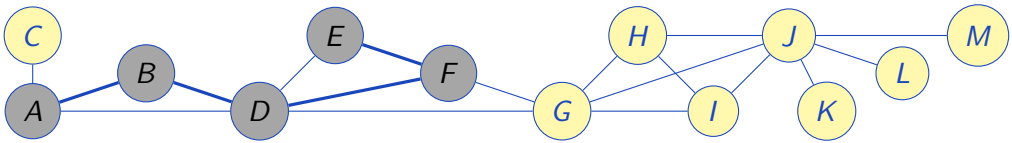
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



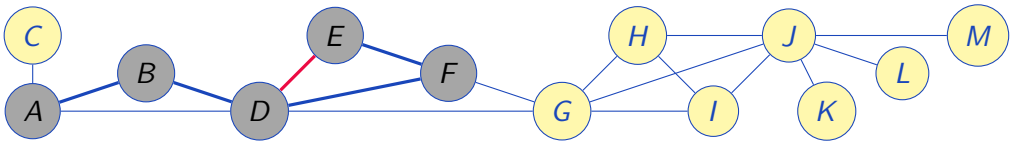
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



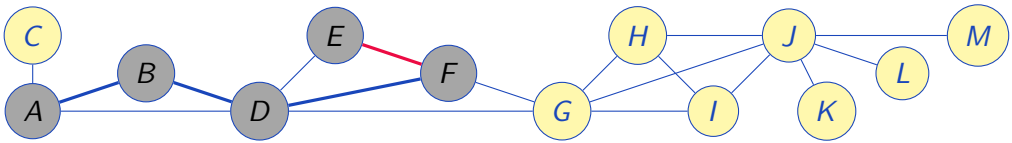
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



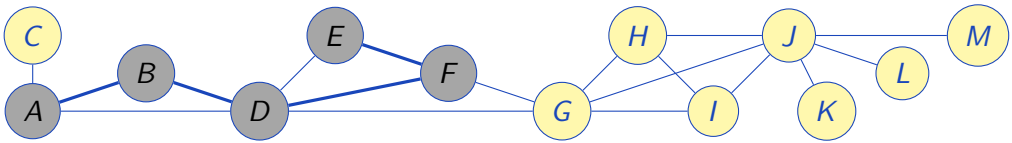
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



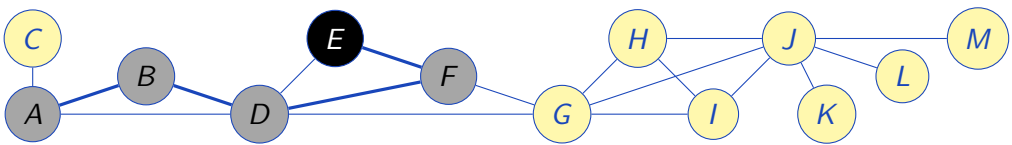
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



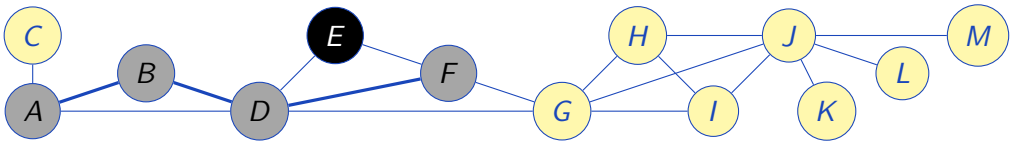
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



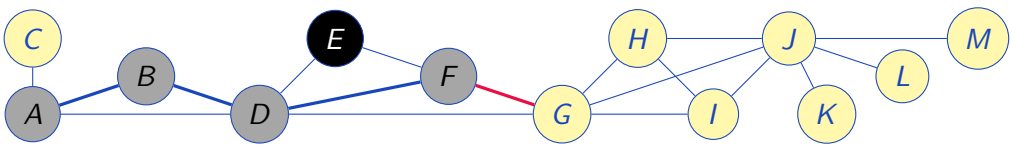
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



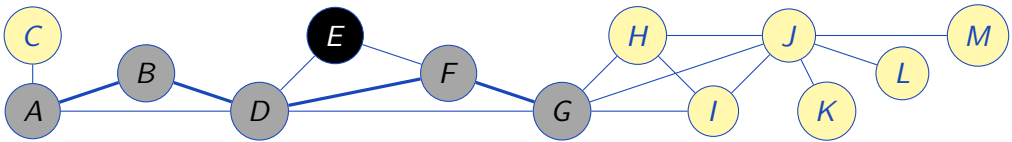
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



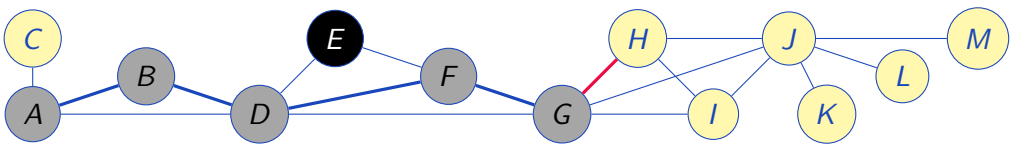
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



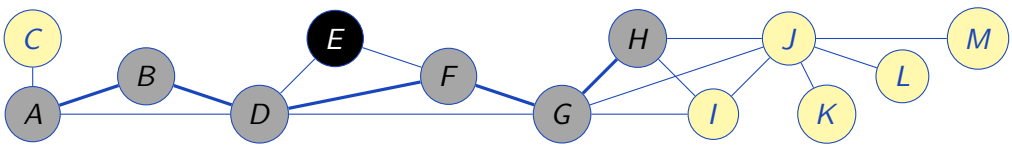
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



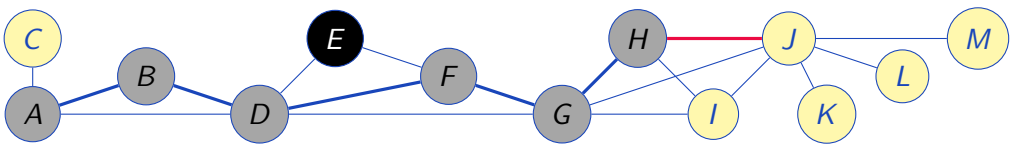
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



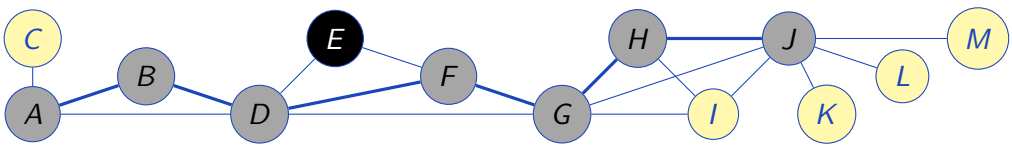
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



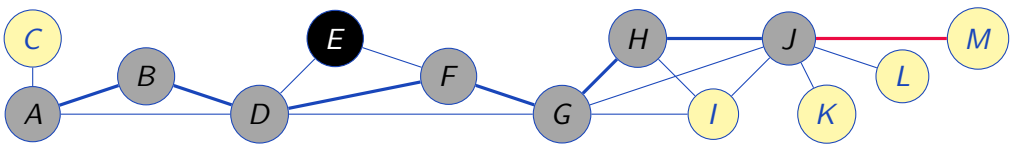
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



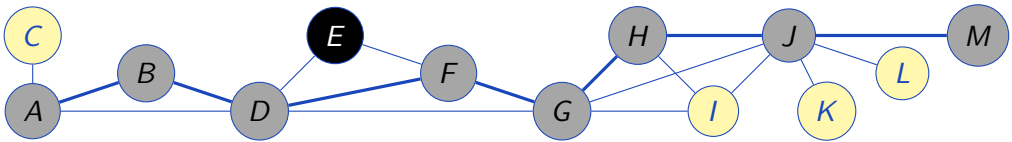
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



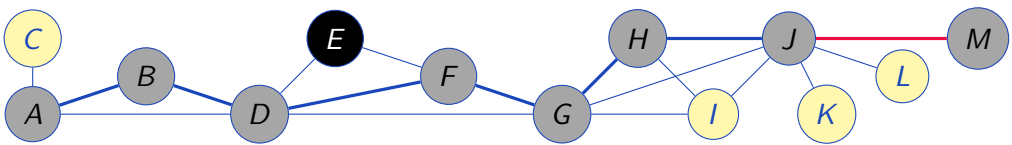
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



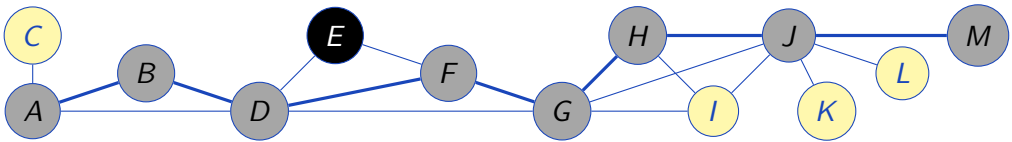
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



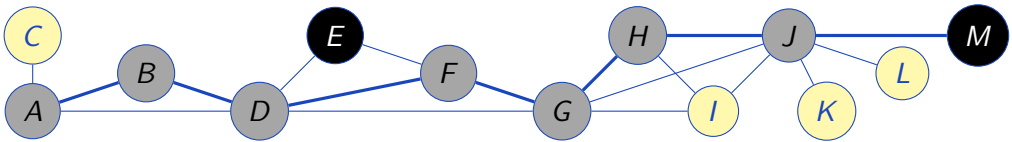
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



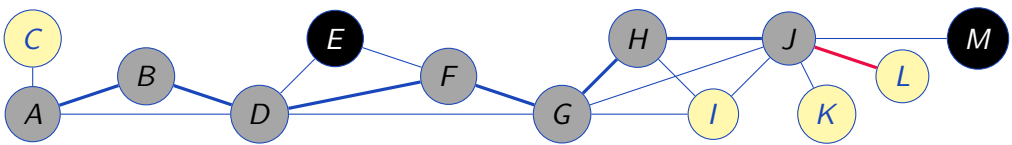
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



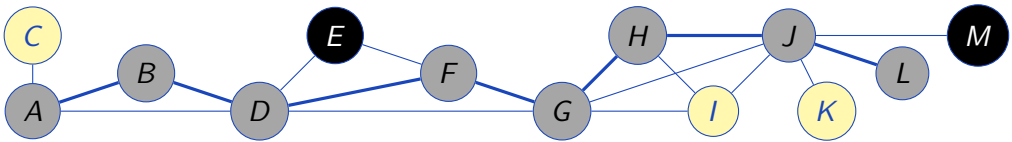
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



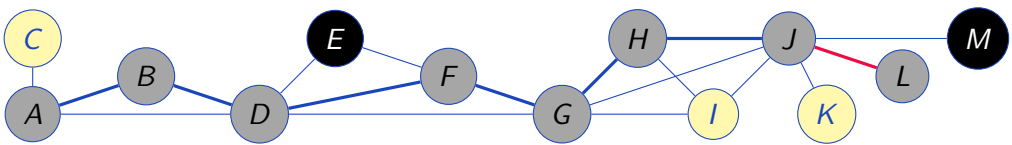
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



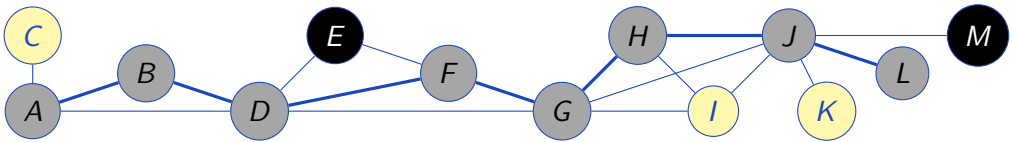
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



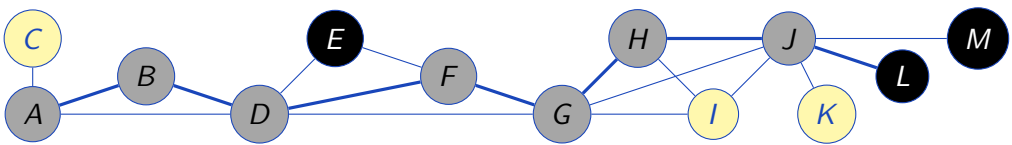
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



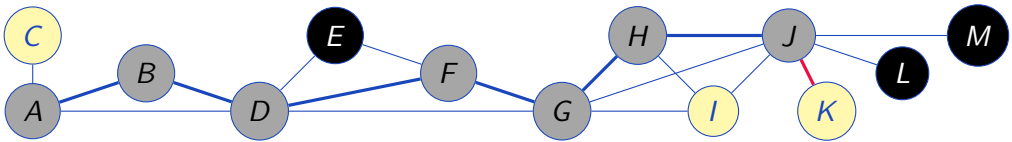
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



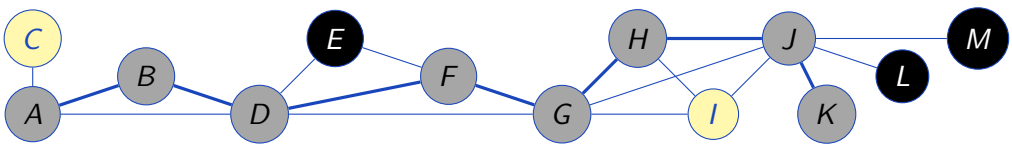
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



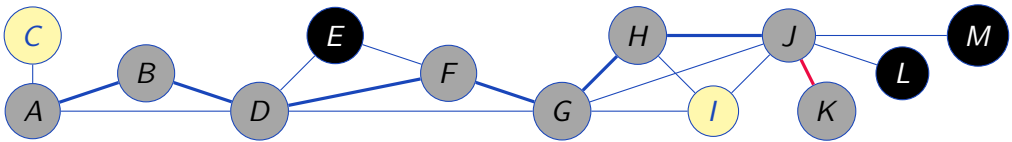
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



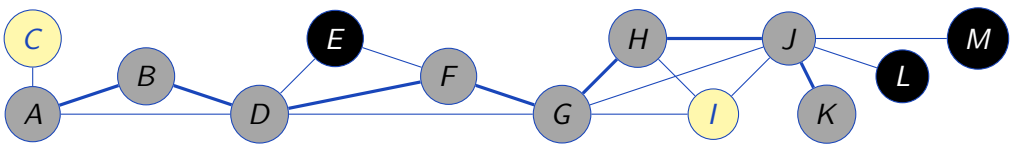
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



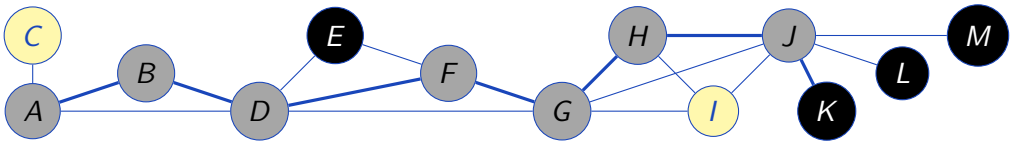
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



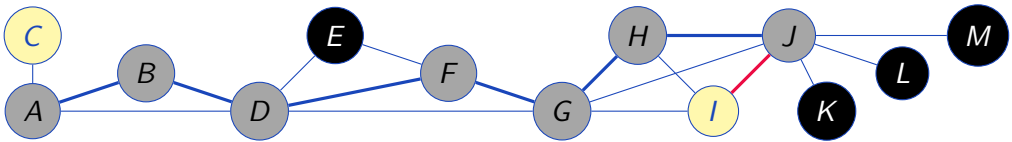
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



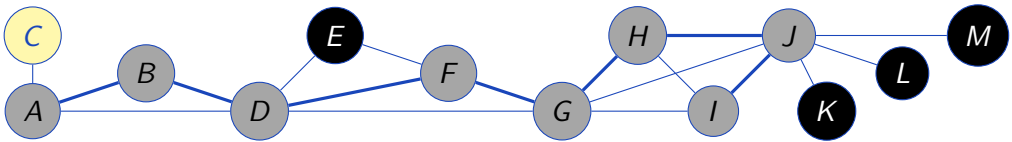
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



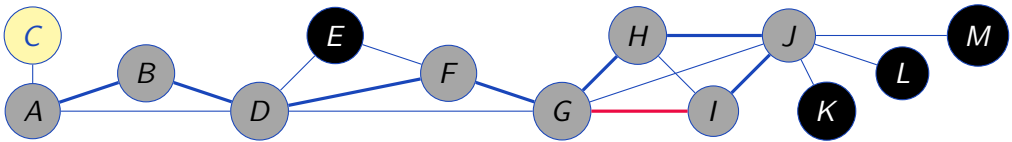
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



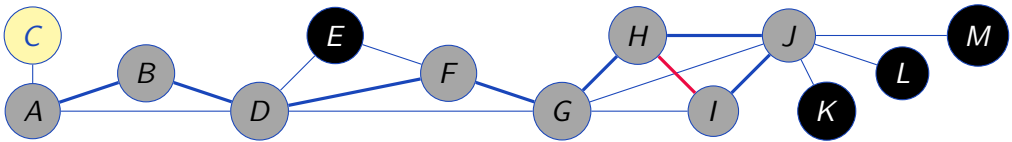
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



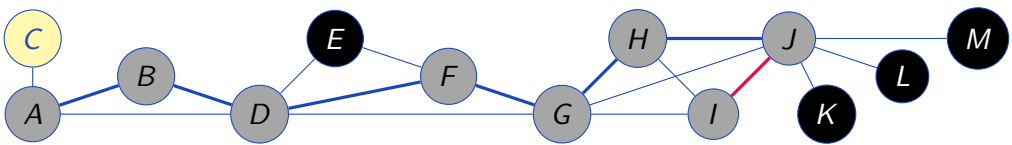
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



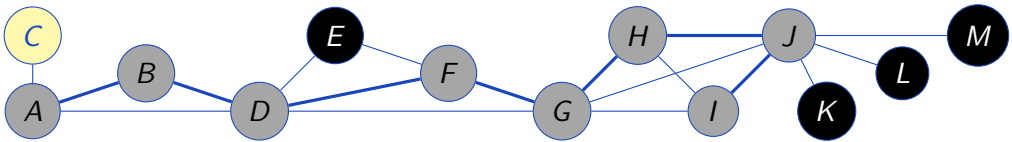
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



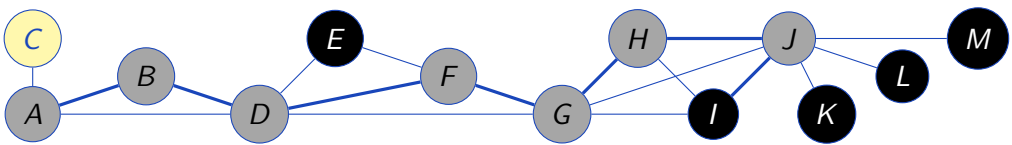
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



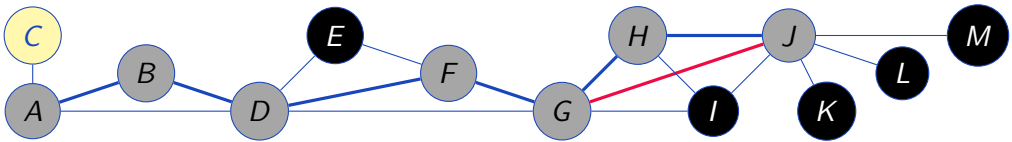
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



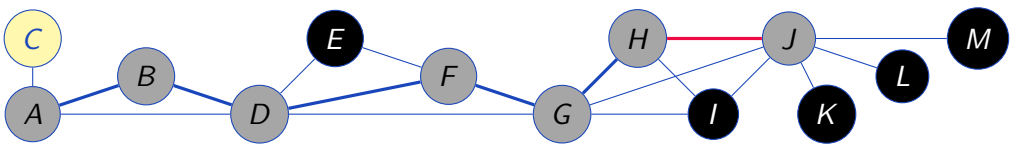
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



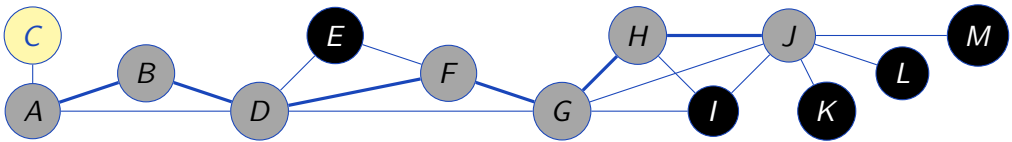
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



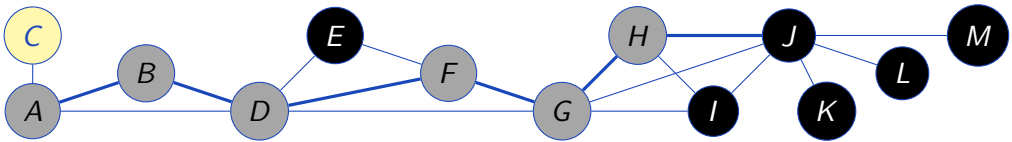
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



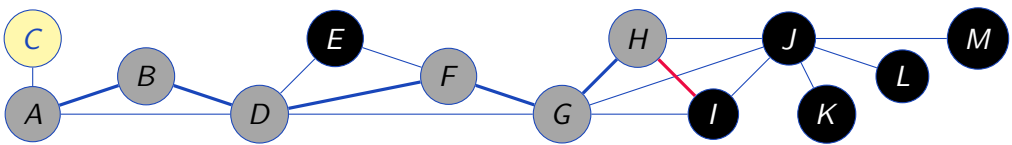
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



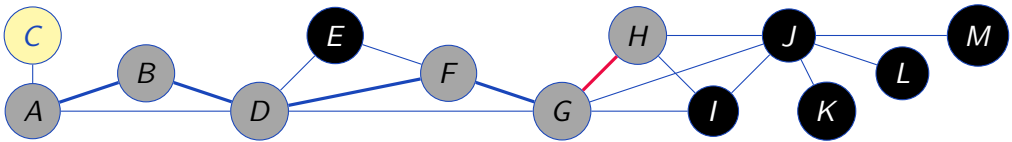
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



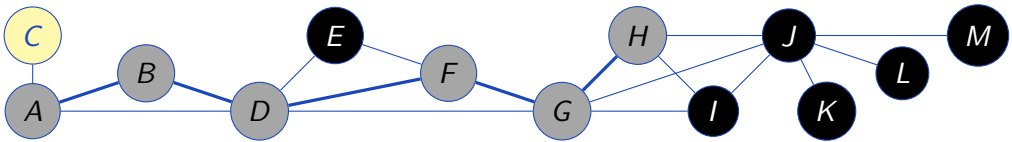
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



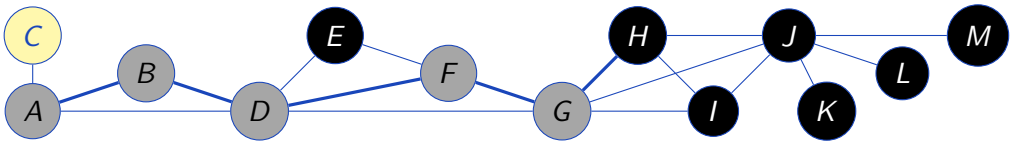
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



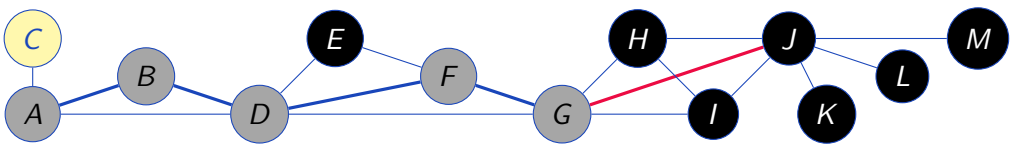
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



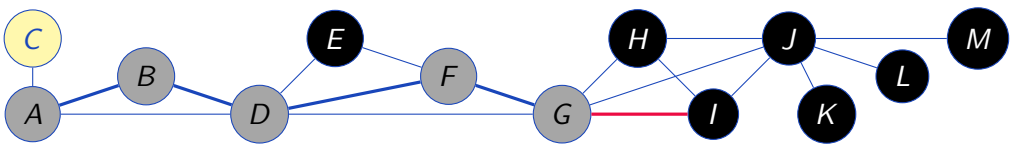
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



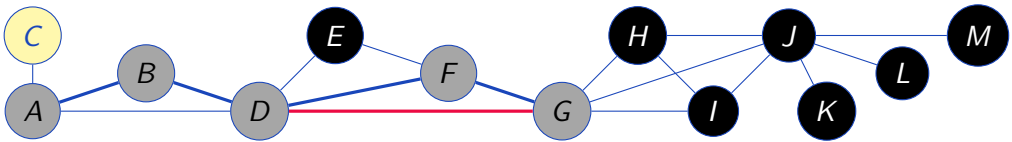
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



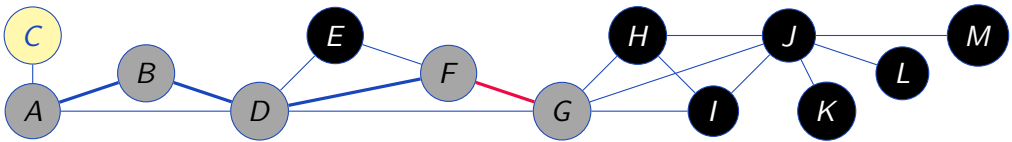
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



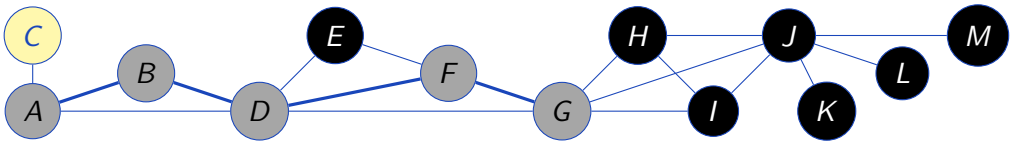
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



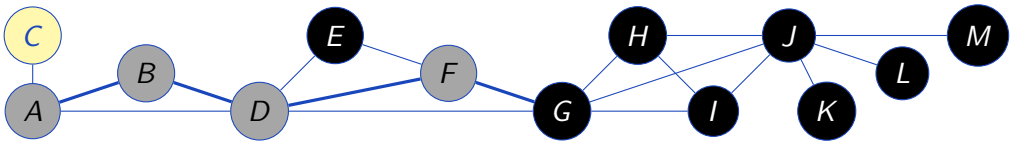
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



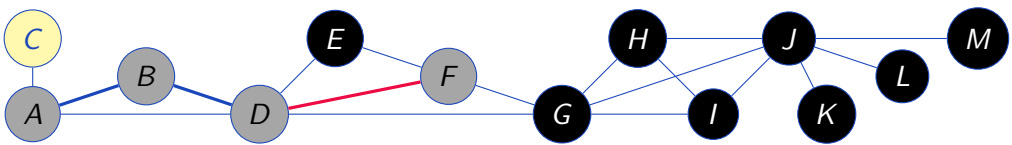
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



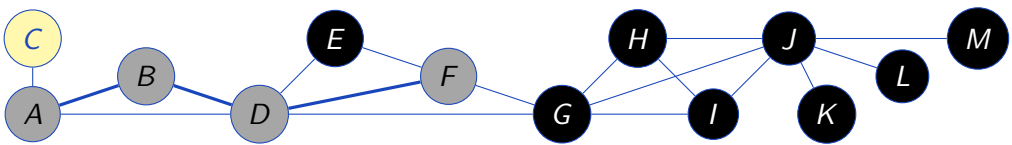
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



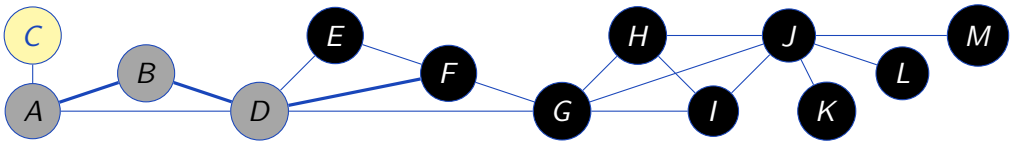
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



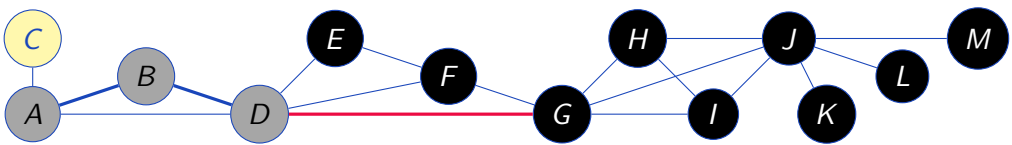
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



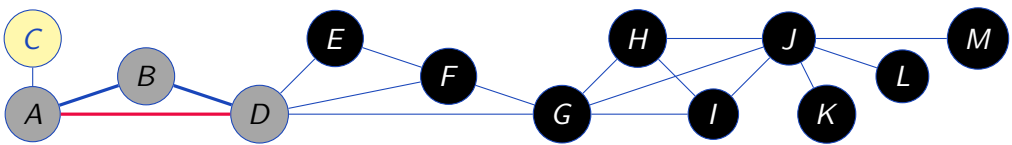
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



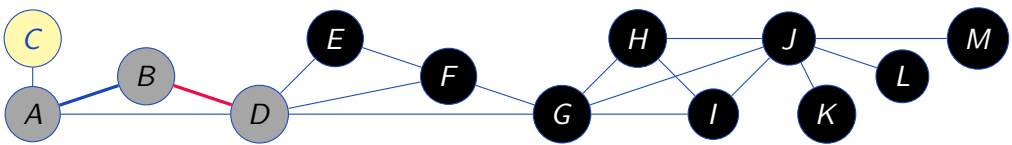
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



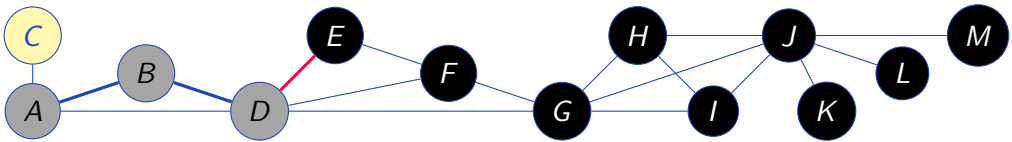
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



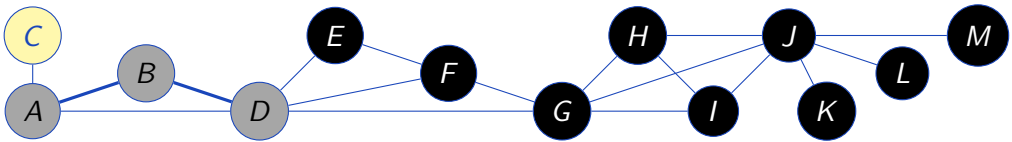
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



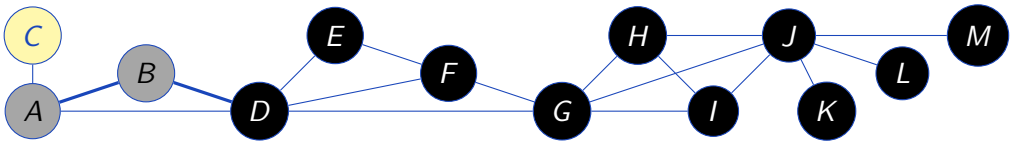
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



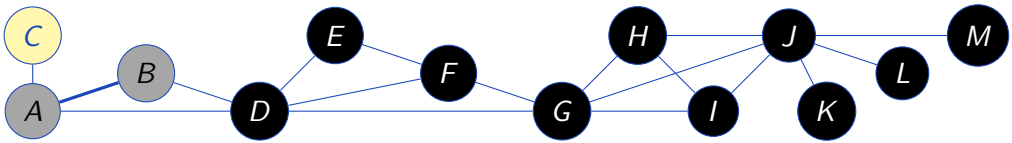
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



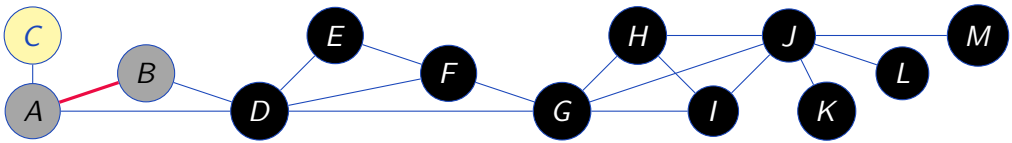
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



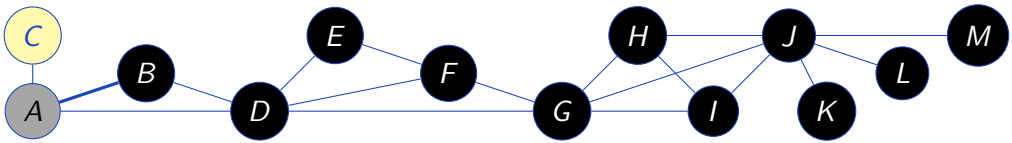
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



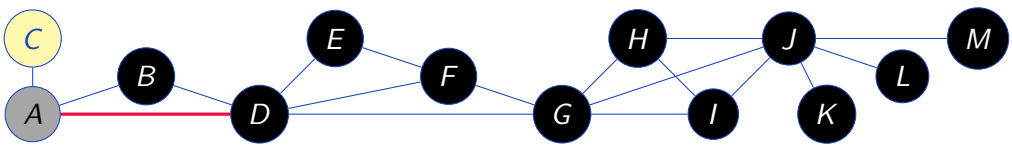
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



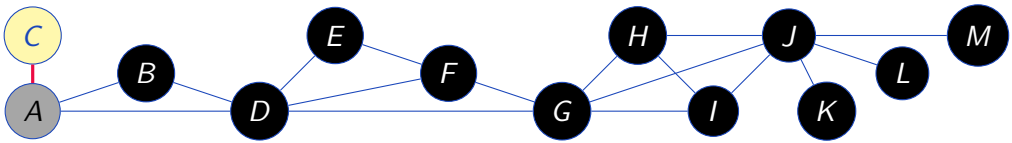
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



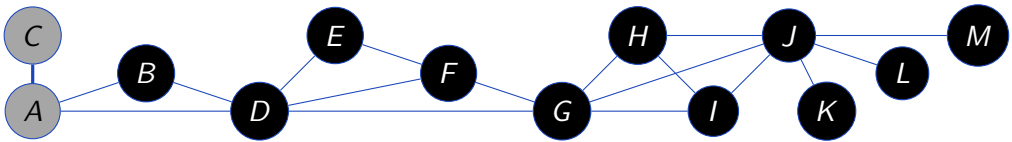
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



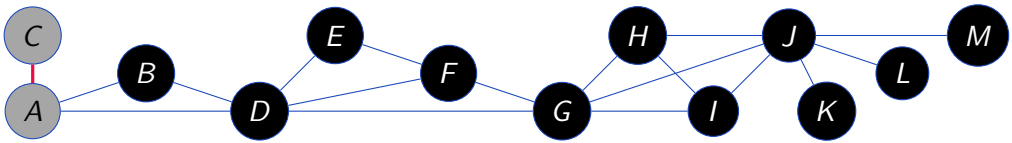
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



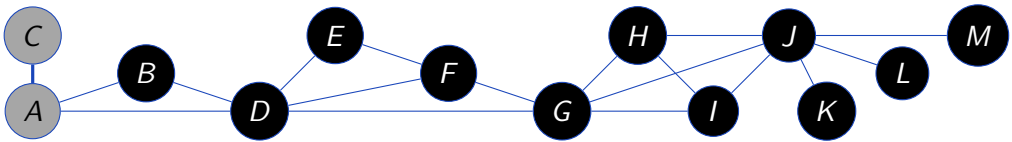
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



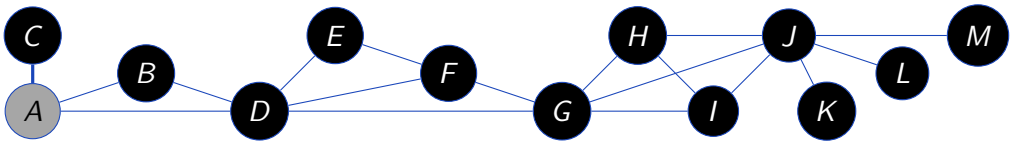
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



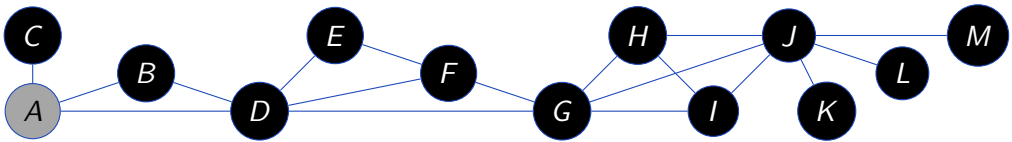
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



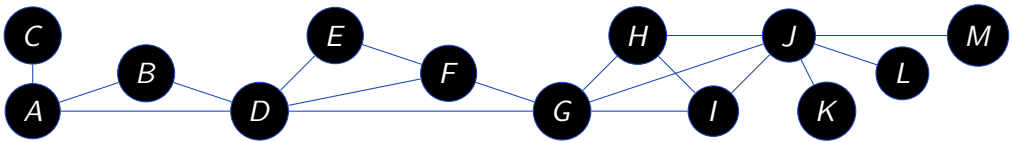
$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**

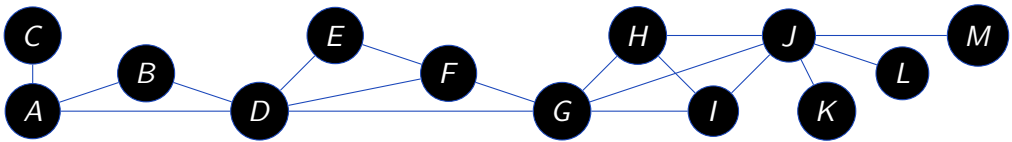


$G = \langle V, E \rangle$ ▷ the graph
$U \leftarrow \emptyset$ ▷ set of visited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ Adjacent vertex function: for free with adjacency lists
**procedure** $\mathrm{DFS}(v)$ ▷ recursive procedure, argument: current vertex
    $U \leftarrow U \cup \{v\}$ ▷ marking current vertex visited
    **for** $u \in A(v)$ **do** ▷ visiting all outgoing edges: more efficient
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if** ▷ if target is not visited, calling recursively
    **end for**
**end procedure**



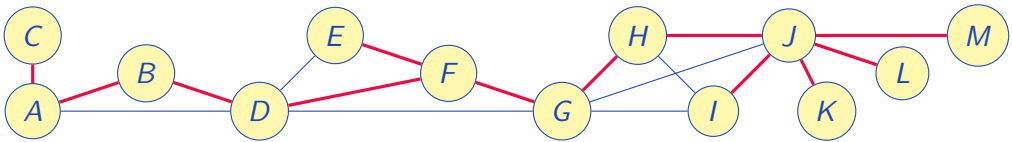
$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

**procedure** IsConnected($V$, $E$)
    DFS(arbitrary vertex from $V$)
    **return** $U = V$
**end procedure**



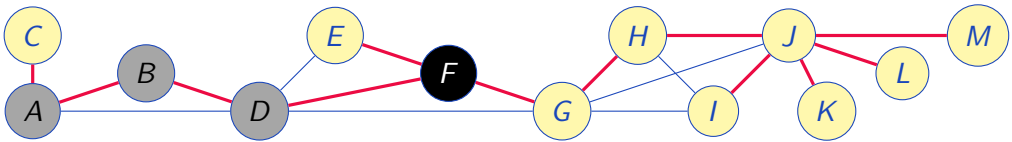
$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

- DFS tree: all traversed edges



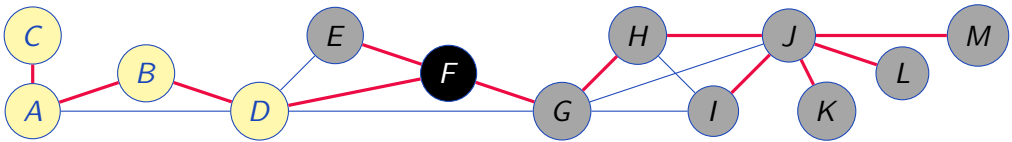
$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

- DFS tree: all traversed edges
- Ancestors of $v$: all vertices up the DFS tree from $v$



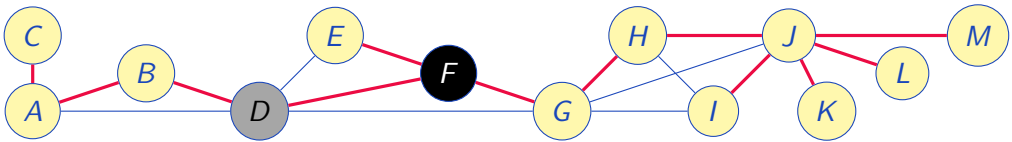
$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

- DFS tree: all traversed edges
- Ancestors of $v$: all vertices up the DFS tree from $v$
- Descendants of $v$: all vertices down the DFS tree from $v$



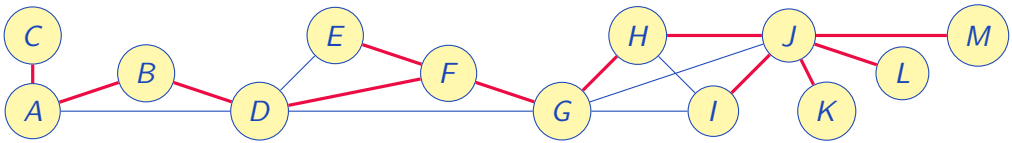
$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

- DFS tree: all traversed edges
- Ancestors of $v$: all vertices up the DFS tree from $v$
- Descendants of $v$: all vertices down the DFS tree from $v$
- Parent of $v$: the immediate ancestor of $v$



$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
  $U \leftarrow U \cup \{v\}$
  **for** $u \in A(v)$ **do**
    **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
  **end for**
**end procedure**

- DFS tree: all traversed edges
- Ancestors of $v$: all vertices up the DFS tree from $v$
- Descendants of $v$: all vertices down the DFS tree from $v$
- Parent of $v$: the immediate ancestor of $v$
- Undirected: Non-DFS-tree edges connect vertices with ancestors or descendants



$G = \langle V, E \rangle$
$U \leftarrow \emptyset$
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \notin U$ **then** $\mathrm{DFS}(u)$ **end if**
    **end for**
**end procedure**

$G = \langle V, E \rangle$
$U \leftarrow \emptyset$, $X \leftarrow \emptyset$ ▷ $X$: the set of exited vertices
$A(v) = \{u \mid (v, u) \in E\}$
**procedure** $\mathrm{DFS}(v)$
    $U \leftarrow U \cup \{v\}$
    **for** $u \in A(v)$ **do**
        **if** $u \in U$ **and** $u \notin X$ **then**
            **return** true ▷ If hitting a visited and not exited vertex, found a cycle
        **end if**
        **if** $u \notin U$ **and** $\mathrm{DFS}(u)$ **then return** true **end if**
    **end for**
    $X \leftarrow X \cup \{v\}$
    **return** false
**end procedure**

$G = \langle V, E \rangle$
$U \leftarrow \emptyset$, $X \leftarrow \emptyset$ ▷ $X$: the set of exited vertices
$A(v) = \{u \mid (v, u) \in E\}$ ▷ $U$ and $X$ are typically implemented as a single array
**procedure** $\mathrm{DFS}(v)$ ▷ color[v] = 0: $v \notin U$, $v \notin X$
  $U \leftarrow U \cup \{v\}$ ▷ color[v] = 1: $v \in U$, $v \notin X$
  **for** $u \in A(v)$ **do** ▷ color[v] = 2: $v \in U$, $v \in X$
    **if** $u \in U$ **and** $u \notin X$ **then**
      **return** true ▷ If hitting a visited and not exited vertex, found a cycle
    **end if**
    **if** $u \notin U$ **and** $\mathrm{DFS}(u)$ **then return** true **end if**
  **end for**
  $X \leftarrow X \cup \{v\}$
  **return** false
**end procedure**