Graph Algorithms

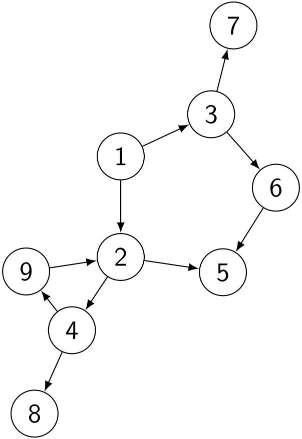
Peter Lammich

27. Januar 2020

Outline

Directed Graphs

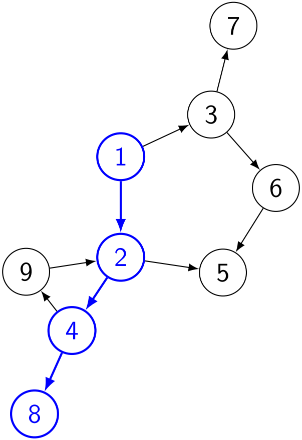
- A *directed graph* is a set of nodes V and edges $E \subseteq V \times V$



$$
\begin{aligned}
V &= \{1, 2, \ldots, 9\} \\
E &= \{(1,2), (1,3), (2,4), (2,5), (3,6), \\
&\quad (3,7), (4,8), (4,9), (6,5), (9,2)\}
\end{aligned}
$$

Directed Graphs

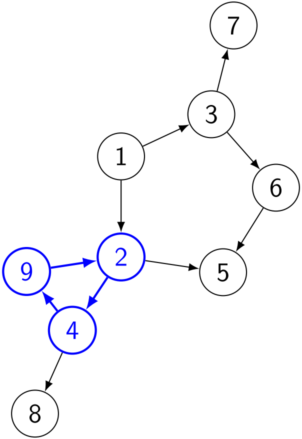
- A *directed graph* is a set of nodes V and edges $E \subseteq V \times V$



Path: Sequence of nodes $u_1 \dots u_n$,
with $\forall i \in \{1 \dots n-1\}.\ (u_i, u_{i+1}) \in E$
e.g. $1, 2, 4, 8$
If not noted otherwise: paths are cycle-free, e.g., no repeated nodes!

Directed Graphs

- A *directed graph* is a set of nodes V and edges $E \subseteq V \times V$

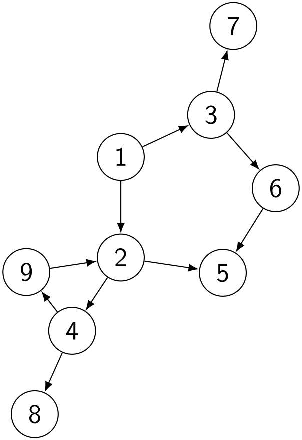


Cycle: Path with same start and end node
e.g. $2, 4, 9, 2$

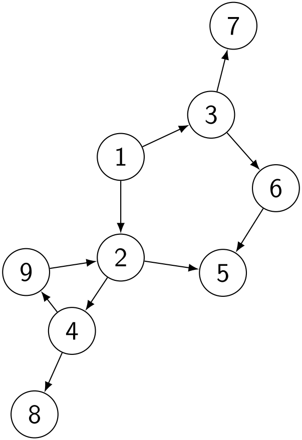
Operations

- Here: nodes implicitly given by edges:
  $V := \{u \mid \exists v.\ (u, v) \in E \vee (v, u) \in E\}$
- EMPTY returns empty graph. $E \leftarrow \emptyset$
- ADDEDGE(u,v) adds an edge. $E \leftarrow E \cup \{(u, v)\}$
- REMOVEEDGE(u,v) removes an edge. $E \leftarrow E \setminus \{(u, v)\}$
- ISEDGE(u,v) checks for edge. $(u, v) \in E$
- SUCCS(u) returns successors of node. $\{v \mid (u, v) \in E\}$

Implementation



Implementation

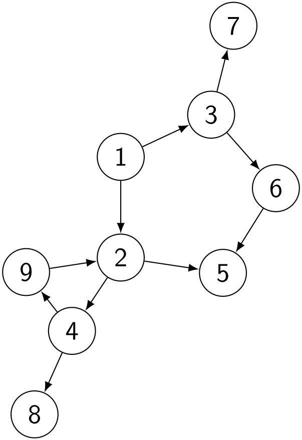


Adjacency list

Store list/set of successors for each node

$succs[1] = \{2, 3\}$
$succs[2] = \{4, 5\}$
$succs[3] = \{6, 7\}$
$succs[4] = \{8, 9\}$
$succs[5] = \{\}$
$succs[6] = \{5\}$
$succs[7] = \{\}$
$succs[8] = \{\}$
$succs[9] = \{2\}$

Implementation



Adjacency Matrix
$m(u, v) = T$ iff edge from u to v

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

Adjacency Lists

- Map each node to list of successors
  - $E = \{(u, v) \mid v \in succs(u)\}$
  - directly implements $\textsc{succs}$(u)
  - for integer nodes: use array of (dynamic) arrays
- Memory $O(|V| + |E|)$

Operation	Implementation	Complexity
$\textsc{addEdge}(u, v)$	append v to $succs(u)$	$O(1)$ (amortized)
$\textsc{removeEdge}(u, v)$	remove v from $succs(u)$	$O(\lvert V\rvert)$
$\textsc{isEdge}(u, v)$	search for v in $succs(u)$	$O(\lvert V\rvert)$
$\textsc{succs}(u)$	return $succs(u)$	$O(1)$

Adjacency Matrix

- Store $|V| \times |V|$ map to Booleans
  - $E = \{(u, v) \mid m(u, v) = \text{true}\}$
  - for integer nodes: use 2 dimensional array
- Memory $O(|V|^2)$.
  - Bad for sparse graphs ($|E| << |V|^2$).

Operation	Implementation	Complexity
$\textsc{addEdge}(u, v)$	$m(u, v) \leftarrow \text{true}$	$O(1)$
$\textsc{removeEdge}(u, v)$	$m(u, v) \leftarrow \text{false}$	$O(1)$
$\textsc{isEdge}(u, v)$	return $m(u, v)$	$O(1)$
$\textsc{succs}(u)$	return $\{v \mid m(u, v) = \text{true}\}$	$O(\lvert V \rvert)$

Adjacency Matrix + Adjacency List

- Store graph simultaneously as adjacency list and matrix
- Memory: $O(|V|^2)$
- addEdge, isEdge, succs — $O(1)$
- removeEdge — $O(|V|)$

Outline

Generic Graph Traversal

- Explore edges to new nodes, until all nodes discovered

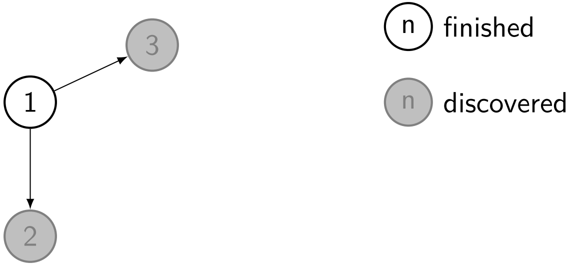
procedure EXPLORE(s)
  $D \leftarrow \{s\}$, $F \leftarrow \emptyset$
  while $D \neq \emptyset$ **do**
    $u \leftarrow \text{Some } u \in D$
    $D \leftarrow D \setminus \{u\}$, $F \leftarrow F \cup \{u\}$
    $D \leftarrow D \cup \{v \mid (u, v) \in E \land v \notin F\}$
  return F

- F finished, outgoing edges have been explored
- D discovered, but outgoing edges yet to be explored
- EXPLORE(s) returns exactly the nodes reachable from s

Example

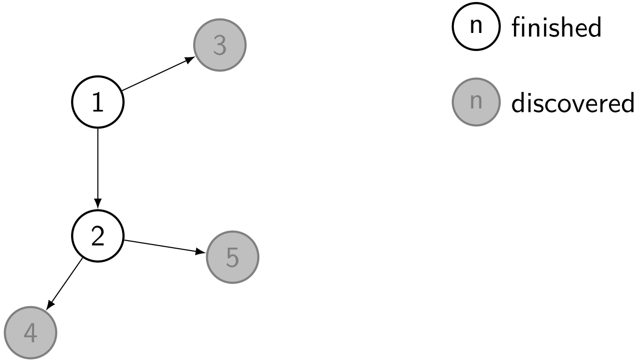
$D = \{ 1 \}$

Example



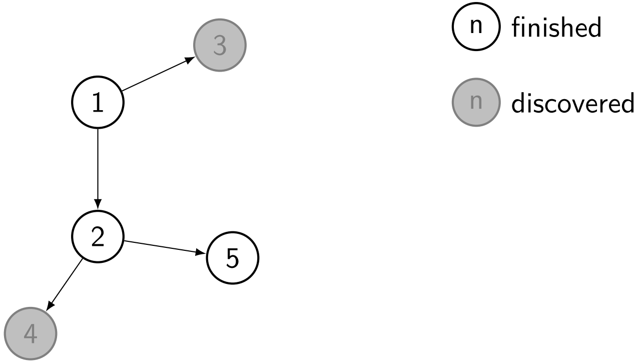
D = { 2 3 }

Example



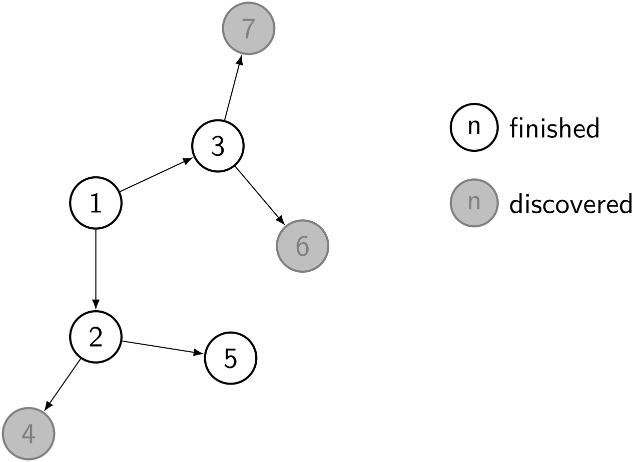
D = { 3 4 5 }

Example



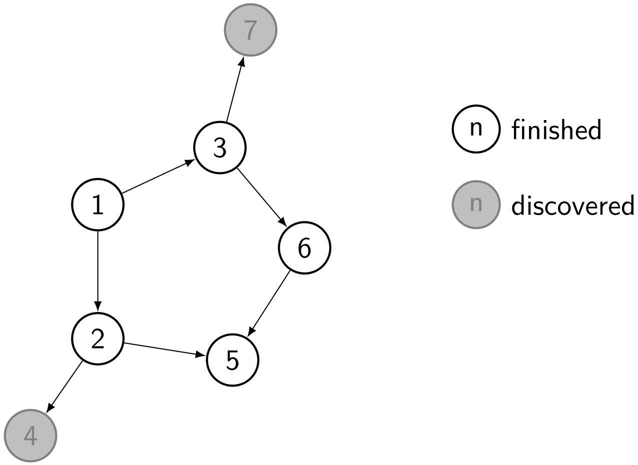
D = { 3 4 }

Example



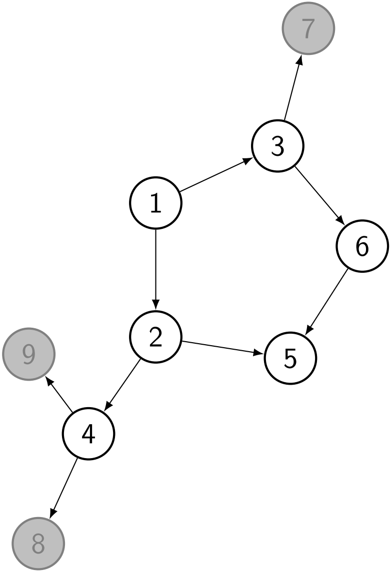
D = { 4 6 7 }

Example



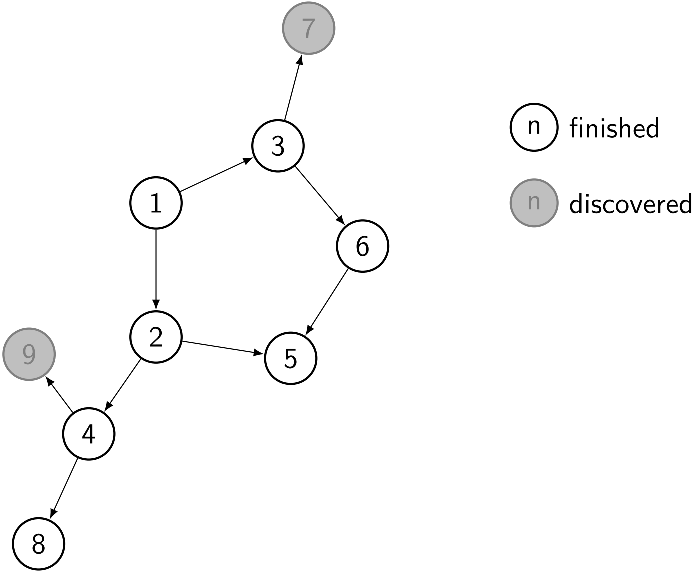
D = { 4 7 }

Example



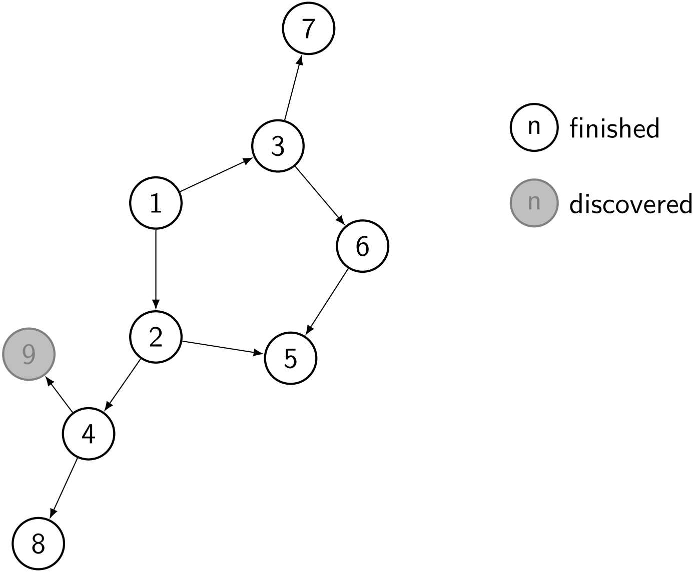
D = { 7 8 9 }

Example



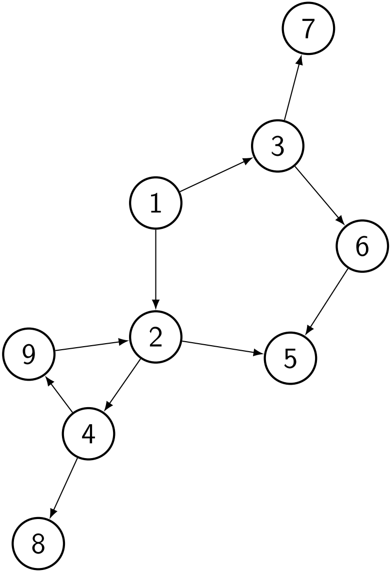
D = { 7 9 }

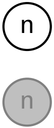
Example



D = { 9 }

Example





D = { }

Generic Graph Traversal (Correctness)

$\textsc{explore}(s)$ returns exactly the nodes reachable from s

- Any node in $D \cup F$ is reachable:
  - Initially, only $s \in D \cup F$
  - only successors of nodes in $D \cup F$ added
- If a node is reachable, it will be included in F:
  - During the loop, successors of finished nodes are finished or discovered
  - Finally, $D = \emptyset$, thus successors of finished nodes are finished

$\Longrightarrow$ every reachable node is finished (follow path from s)

DFS and BFS

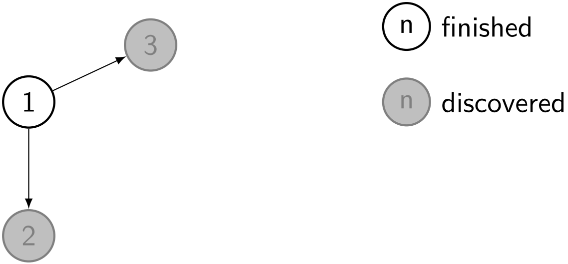
procedure $\textsc{Explore}(s)$
    $D \leftarrow \{s\}$, $F \leftarrow \emptyset$
    while $D \neq \emptyset$ **do**
        $u \leftarrow \text{Some } u \in D$
        $D \leftarrow D \setminus \{u\}$, $F \leftarrow F \cup \{u\}$
        $D \leftarrow D \cup \{v \mid (u, v) \in E \wedge v \notin F\}$
    return F

- In which order do we process nodes from D
- Last in / first out (stack): *Depth First Search* (DFS)
- First in / first out (queue): *Breadth First Search* (BFS)

Example: DFS

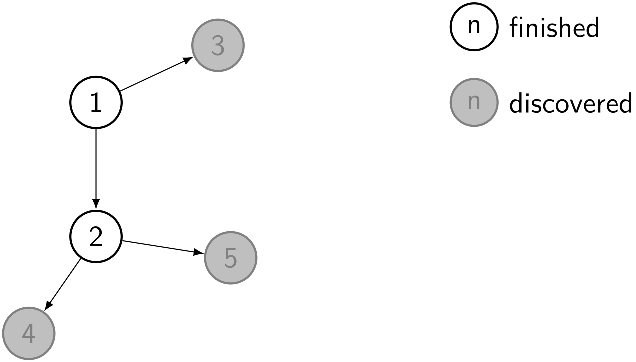
D (LiFo): [1]

Example: DFS



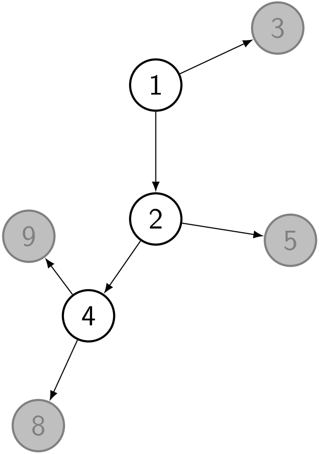
D (LiFo): [3 2]

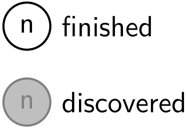
Example: DFS



D (LiFo): [3 5 4]

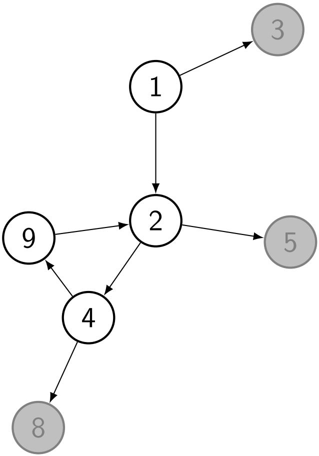
Example: DFS





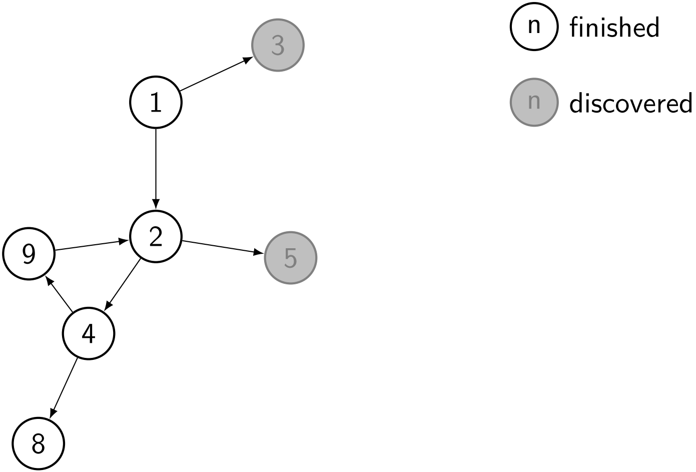
D (LiFo): [3 5 8 9]

Example: DFS



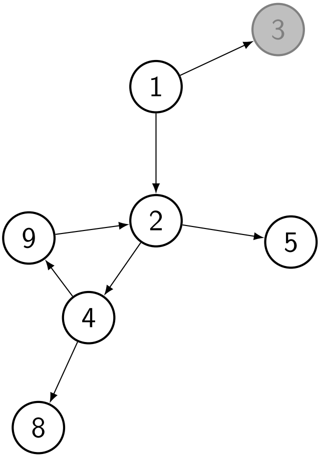
D (LiFo): [3 5 8]

Example: DFS



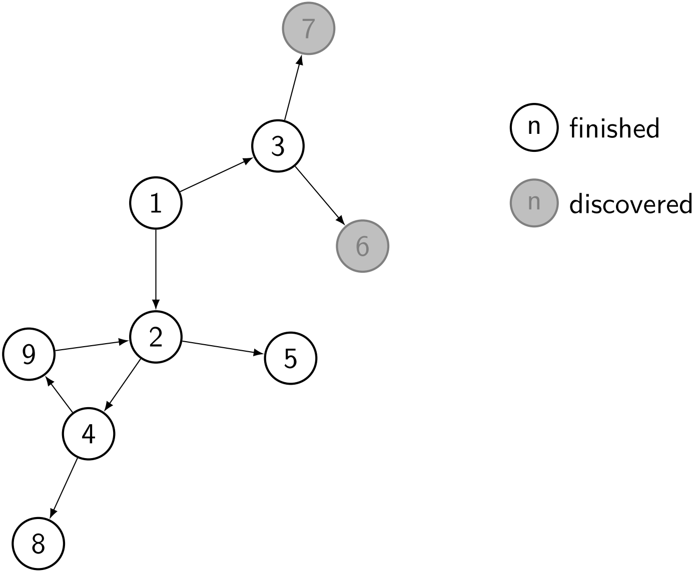
D (LiFo): [3 5]

Example: DFS



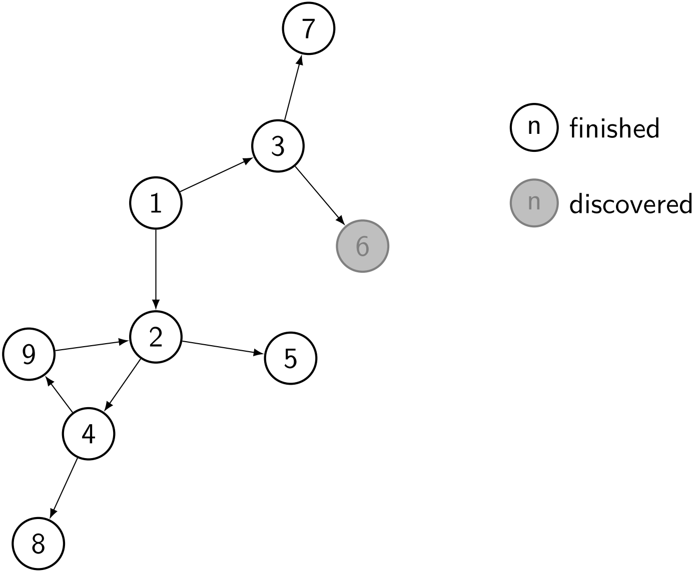
D (LiFo): [3]

Example: DFS



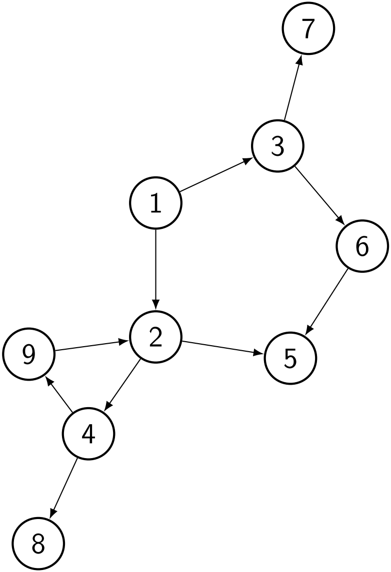
D (LiFo): [6 7]

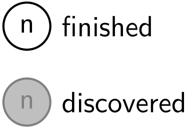
Example: DFS



D (LiFo): [6]

Example: DFS



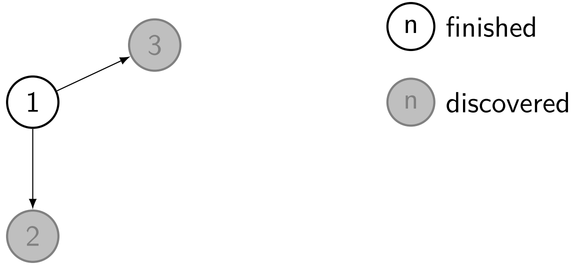


D (LiFo): [ ]

Example: BFS

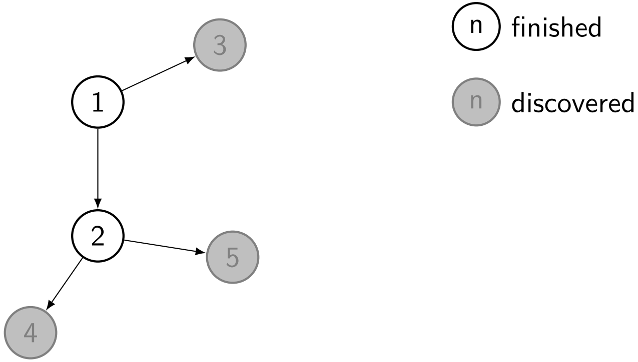
D (FiFo): [1]

Example: BFS



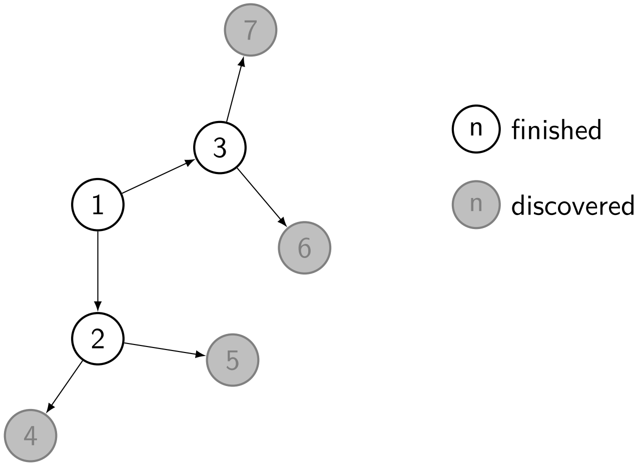
D (FiFo): [2 3]

Example: BFS



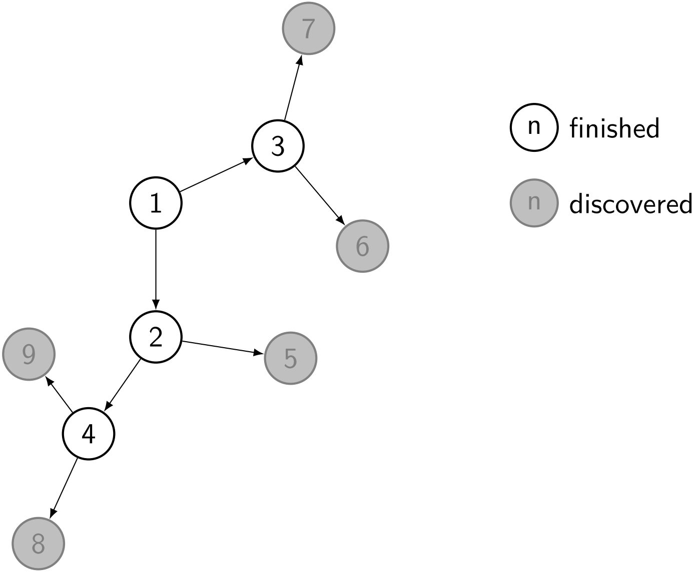
D (FiFo): [3 4 5]

Example: BFS



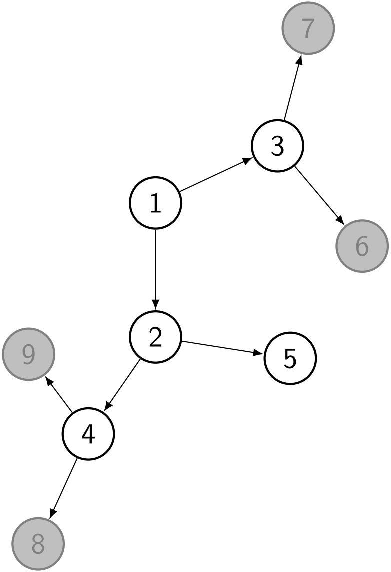
D (FiFo): [4 5 7 6]

Example: BFS



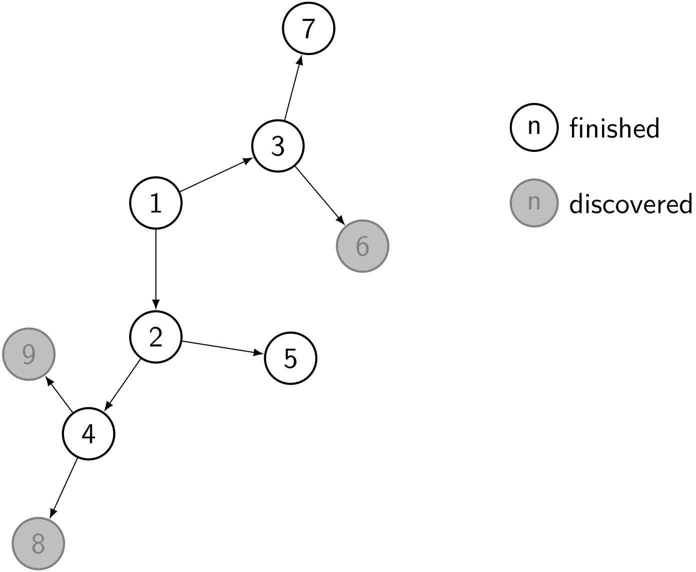
D (FiFo): [5 7 6 9 8]

Example: BFS



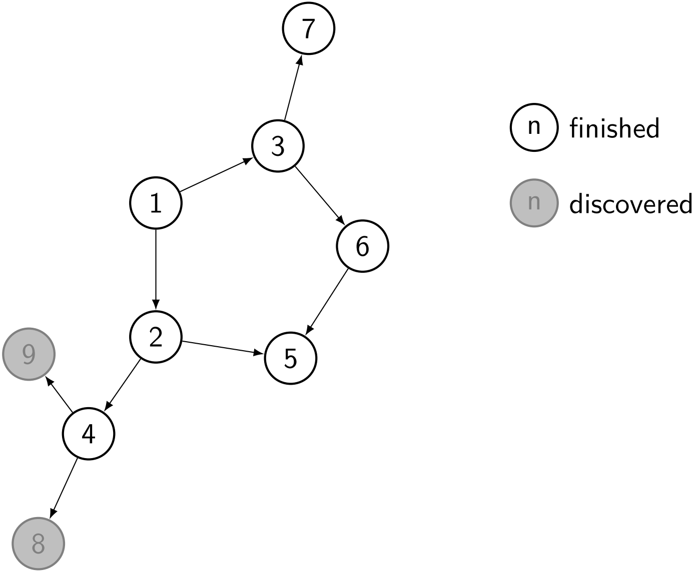
D (FiFo): [7 6 9 8]

Example: BFS



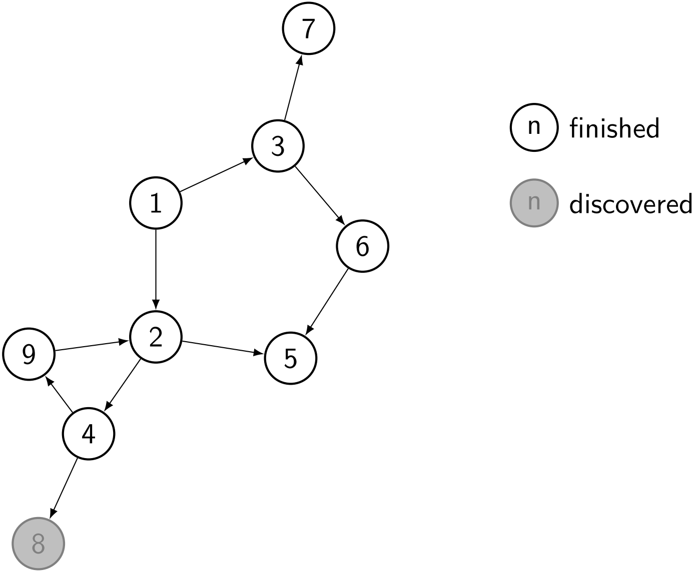
D (FiFo): [6 9 8]

Example: BFS



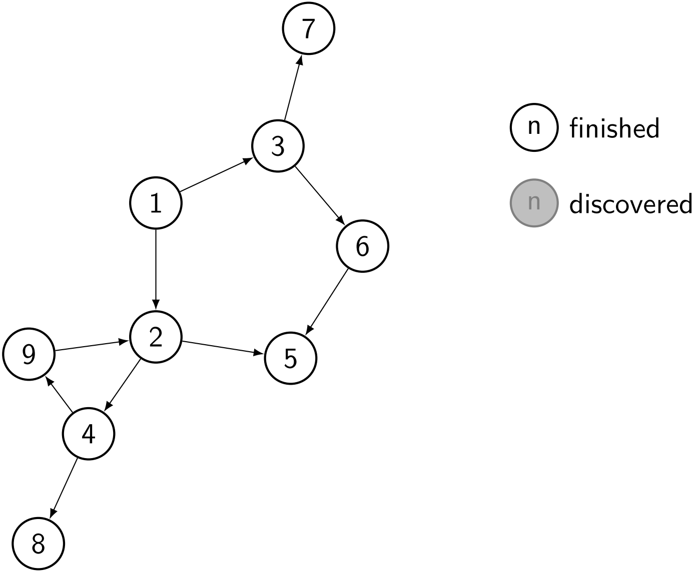
D (FiFo): [9 8]

Example: BFS



D (FiFo): [8]

Example: BFS



D (FiFo): [ ]

Recursive DFS

- DFS has nice recursive implementation

procedure $\textsc{dfsRec}(F, u)$
  if $u \notin F$ **then**
    $F \leftarrow F \cup \{u\}$
    for all v with $(u, v) \in E$ **do**
      $F \leftarrow \textsc{dfsRec}(\mathsf{F,v})$
  return F
procedure $\textsc{dfs}(s)$ **return** $\textsc{dfsRec}(\emptyset, s)$

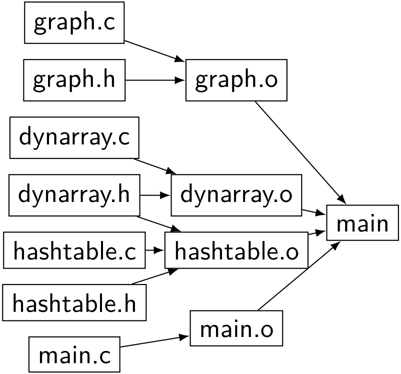
Topological Sorting

- Set of tasks (e.g. build jobs in Makefile)
- Dependencies, i.e. tasks that needs to be completed before a task can be started
- Model as directed graph. Edge (u, v): v depends on u.
- Find a *build sequence*

Example

```
main: main.o hashtable.o dynarray.o graph.o
  gcc ...

main.o: main.c
hashtable.o: hashtable.h hashtable.c dynarray.h
dynarray.o: dynarray.h dynarray.c
graph.o: graph.h graph.c
```



Possible build sequence:
graph.o, dynarray.o, hashtable.o, main.o, main

Topological Sorting

- Arrange nodes sequentially, such that all edges point forwards
  - Intuition: All prerequisites come earlier in sequence
- Topological sorting possible iff graph has no cycles
  - *Directed Acyclic Graph* (DAG)
- Now: DFS based algorithm for topological sorting and cycle detection

Cycle detection with DFS

- When first encountering a node, mark it as *open*.
- Only when finished exploring its children, mark it as *done*
- Whenever we encounter an open node again, it's a cycle

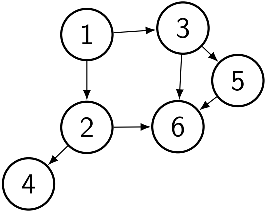
procedure $\textsc{dfsRec}(F, u)$
  if $F(u) = \mathrm{N}$ **then**
    $F(u) \leftarrow \mathrm{O}$ // Start processing node
    for all v with $(u, v) \in E$ **do**
      $F \leftarrow \textsc{dfsRec}(\mathrm{F,v})$
    $F(u) \leftarrow \mathrm{D}$ // Done processing node
  else if $F(u) = \mathrm{O}$ **then**
    Error: found cycle
  return F
procedure $\textsc{dfs}(s)$
  $F(u) \leftarrow \mathrm{N}$ for all nodes u **return** $\textsc{dfsRec}(F, s)$

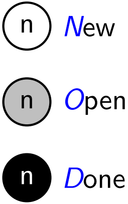
Topological Sorting

procedure $\textsc{dfsRec}(F, u)$
  if $F(u) = \mathrm{N}$ **then**
    $F(u) \leftarrow \mathrm{O}$ // Start processing node
    for all v with $(u, v) \in E$ **do**
      $F \leftarrow \textsc{dfsRec}(F,v)$
    $F(u) \leftarrow \mathrm{D}$ // Done processing node
  else if $F(u) = \mathrm{O}$ **then**
    Error: found cycle
  return F
procedure $\textsc{dfs}(s)$
  $F(u) \leftarrow \mathrm{N}$ for all nodes u **return** $\textsc{dfsRec}(F, s)$

- When node is done, all its successors are already done
  $\Longrightarrow$ Nodes done in reverse topological order
- to get topological sort: *prepend* nodes to list when marked as done

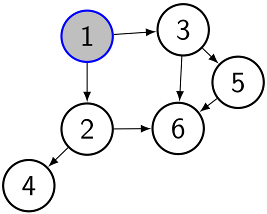
Example

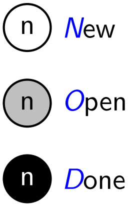




Result List:

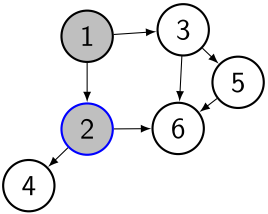
Example

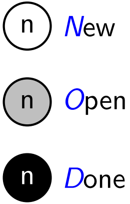




Result List:

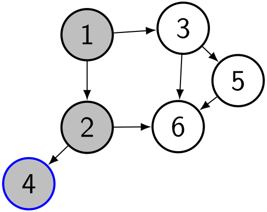
Example

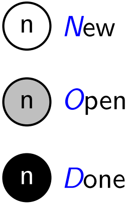




Result List:

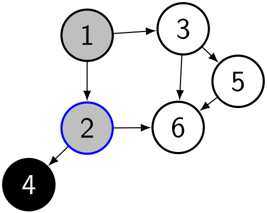
Example



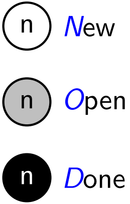


Result List:

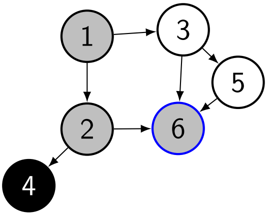
Example

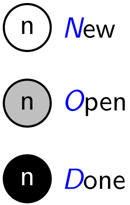


Result List: 4



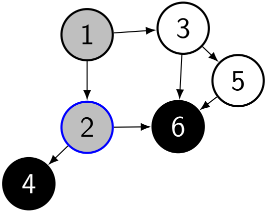
Example

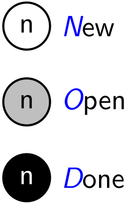




Result List: 4

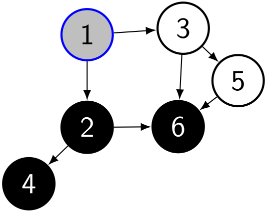
Example



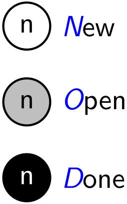


Result List: 6 4

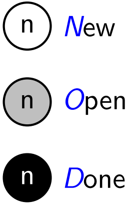
Example

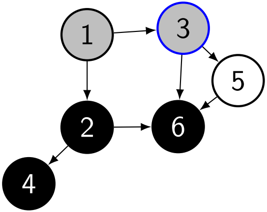


Result List: 2 6 4



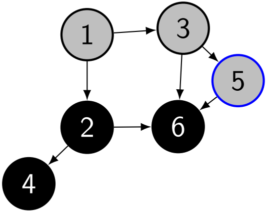
Example

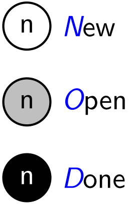




Result List: 2 6 4

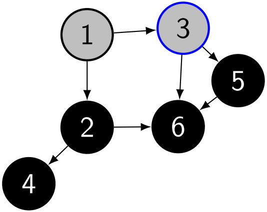
Example



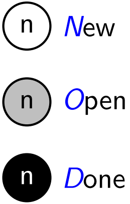


Result List: 2 6 4

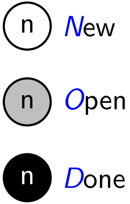
Example

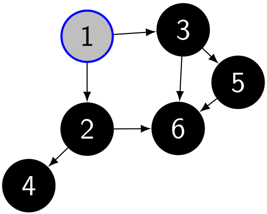


Result List: 5 2 6 4



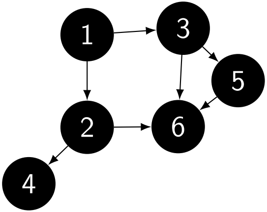
Example

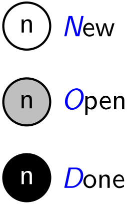




Result List: 3 5 2 6 4

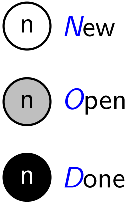
Example

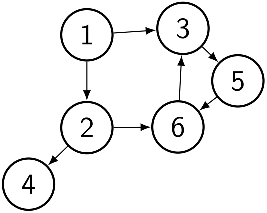




Result List: 1 3 5 2 6 4

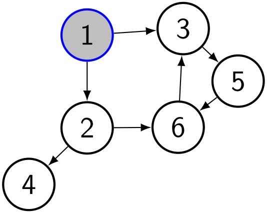
Example



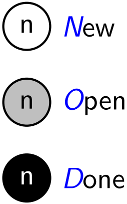


Result List:

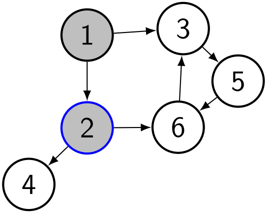
Example



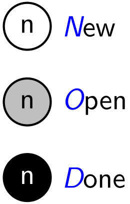
Result List:



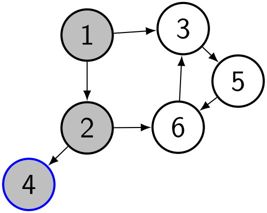
Example



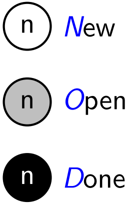
Result List:



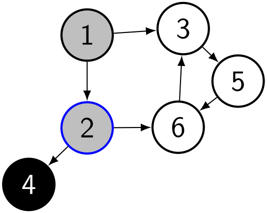
Example



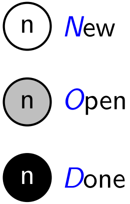
Result List: 4



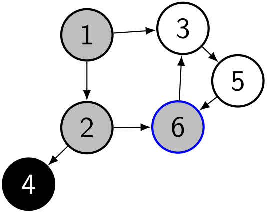
Example



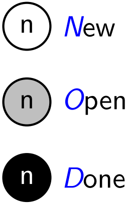
Result List: 4



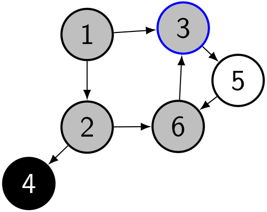
Example



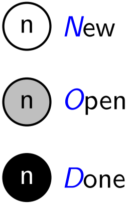
Result List: 4



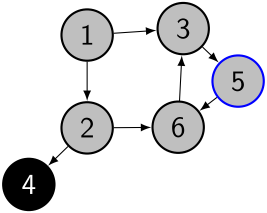
Example



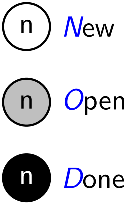
Result List: 4



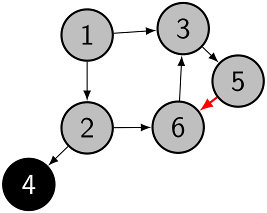
Example



Result List: 4

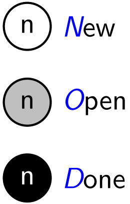


Example



Result List: 4

Error: found cycle



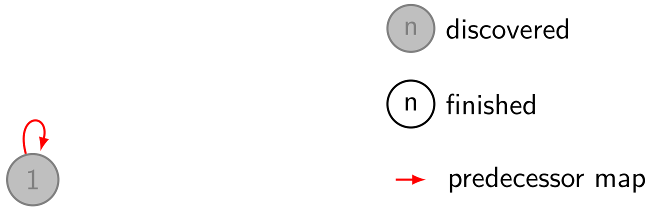
Shortest paths with BFS

- Shortest path from s to some node?
  - shortest = minimal number of edges
- BFS visits nodes in order of their distance from s!
- Obtain path via *predecessor map*:
  - For each node, store node from which it was discovered
  - Follow these nodes backwards, until s is reached
  - Convention: predecessor of s is s itself.

BFS Shortest Path

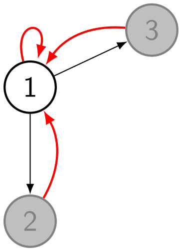
procedure $\text{SHORTESTPATHS}(s)$
    $D \leftarrow [s]$, $F \leftarrow \emptyset$, $\pi(s) \leftarrow s$
    while $D \neq []$ **do**
        $(u, D) \leftarrow \text{DEQUEUE}(D)$
        $D \leftarrow D \setminus \{u\}$, $F \leftarrow F \cup \{u\}$
        for all v with $(u, v) \in E \wedge v \notin F$ **do**
            $D \leftarrow \text{ENQUEUE}(v)$, $\pi(v) \leftarrow u$
    return π
procedure $\text{GETPATH}(\pi, u)$
    $p \leftarrow [u]$
    while $\pi(u) \neq u$ **do**
        $u \leftarrow \pi(u)$, $p \leftarrow up$

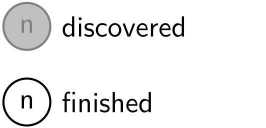
Example: BFS shortest paths



D (FiFo): [1 |]

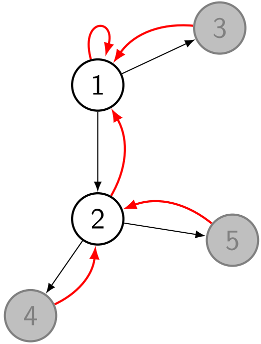
Example: BFS shortest paths

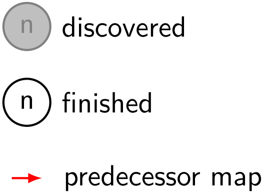




D (FiFo): [1 | 2 3 |]

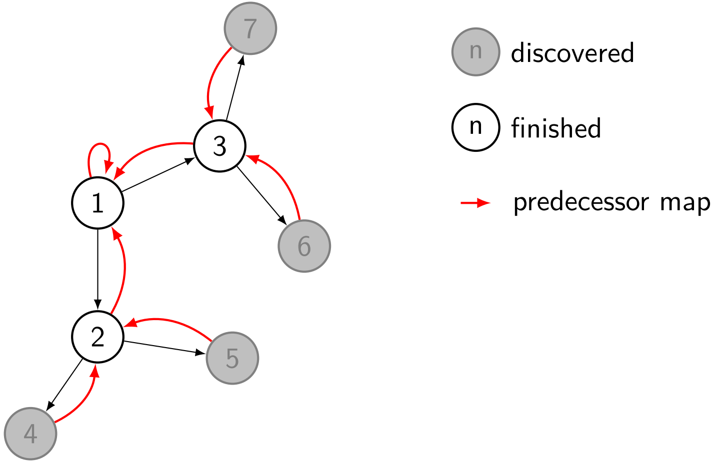
Example: BFS shortest paths





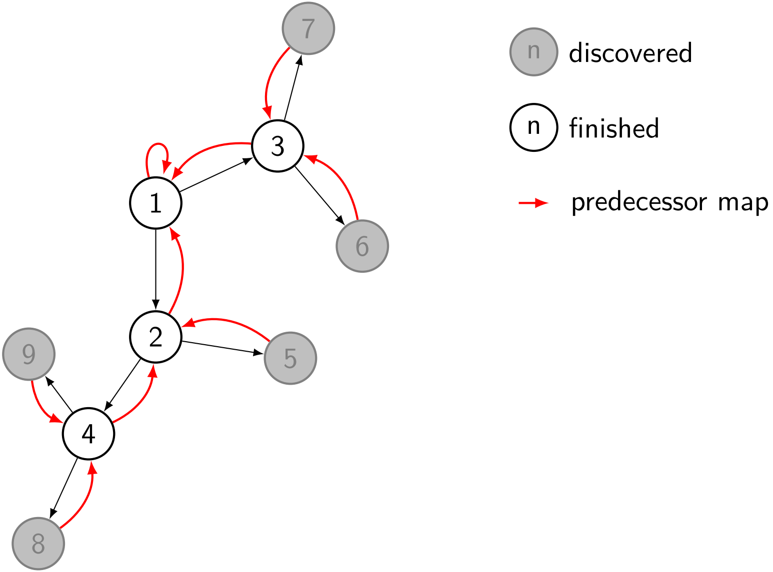
D (FiFo): [1 | 2 3 | 4 5]

Example: BFS shortest paths



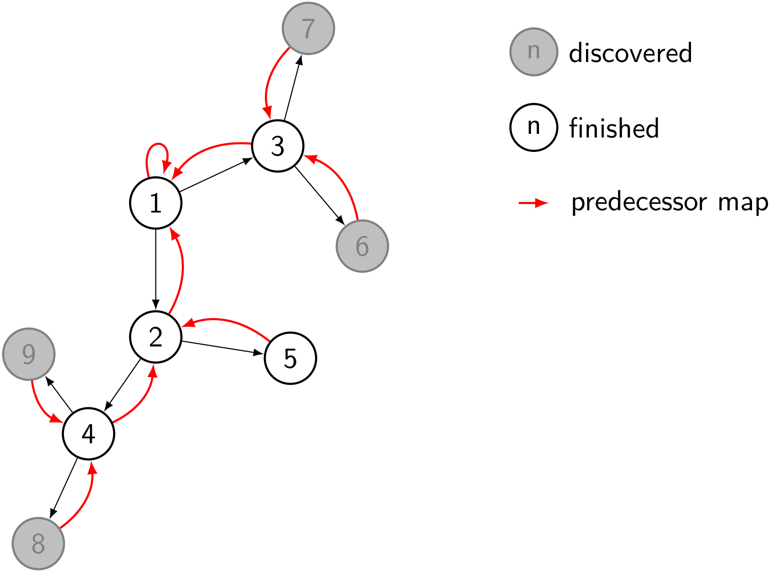
D (FiFo): [1 | 2 3 | 4 5 7 6 |]

Example: BFS shortest paths



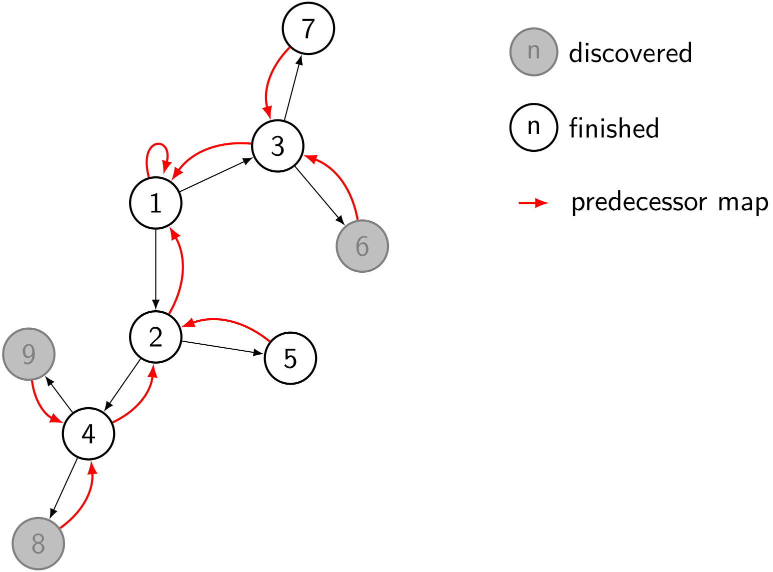
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



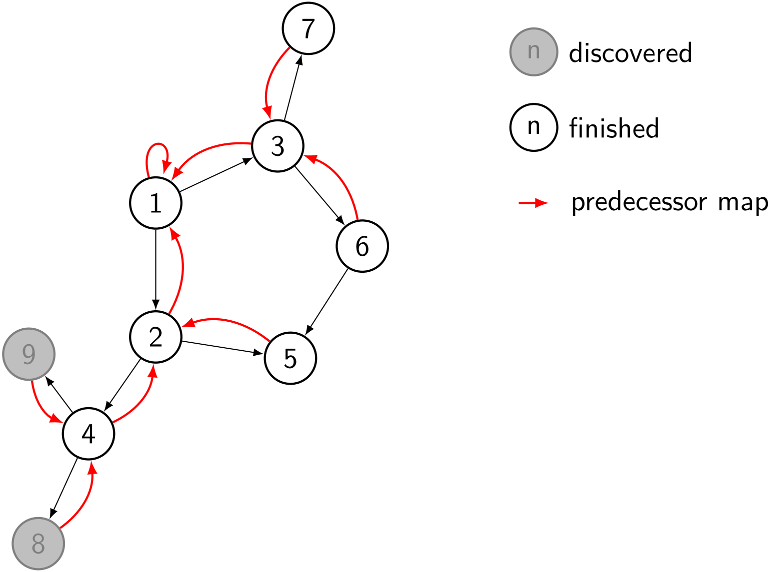
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



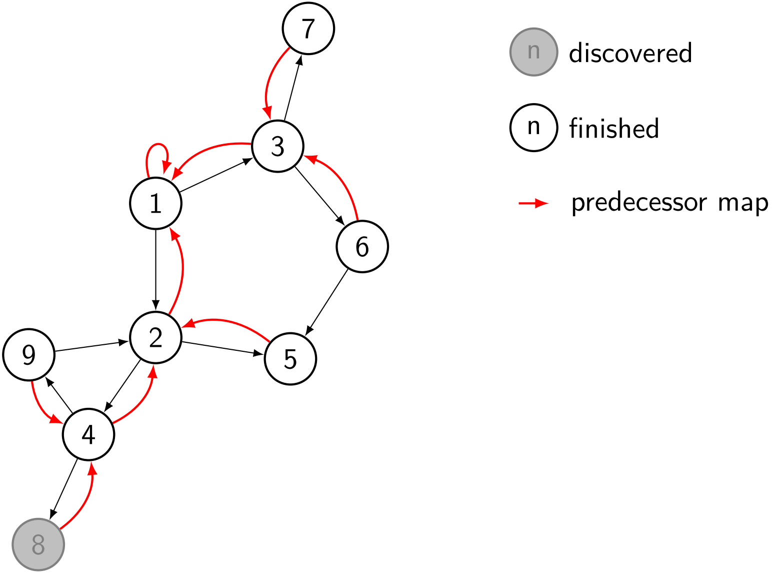
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



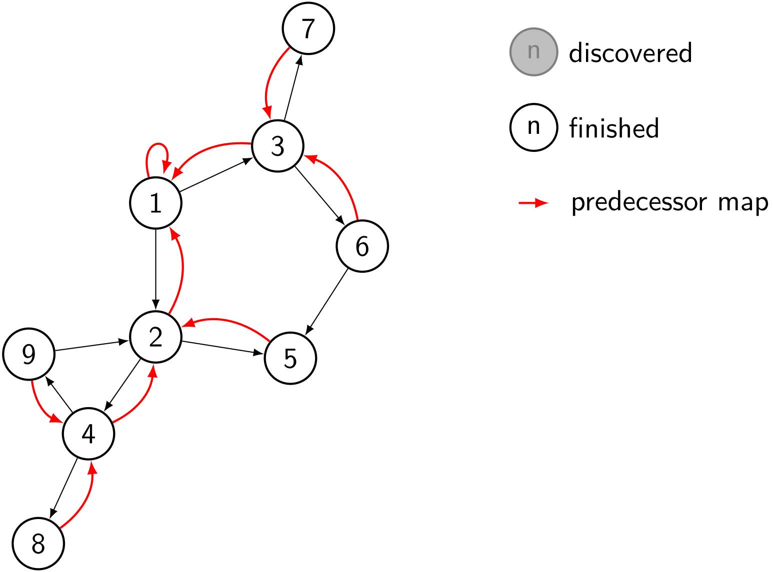
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



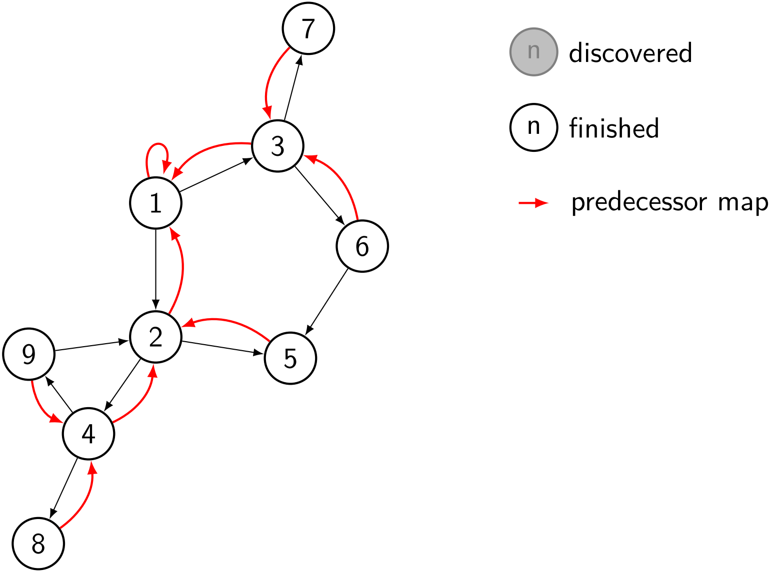
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



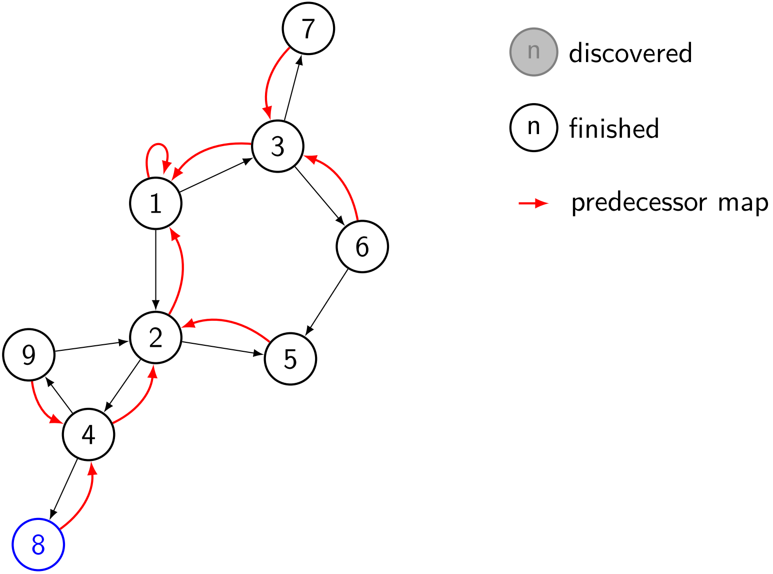
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



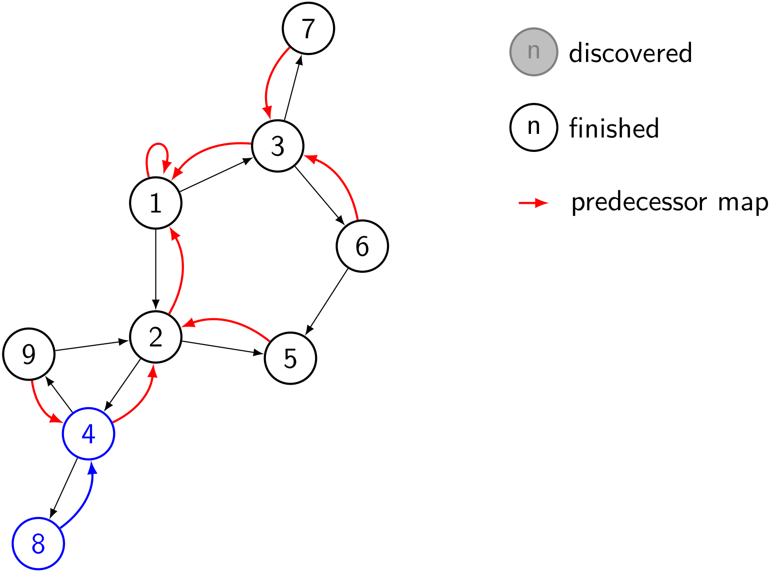
D (FiFo): [1 | 2 3 | 4 5 7 6 | 9 8]

Example: BFS shortest paths



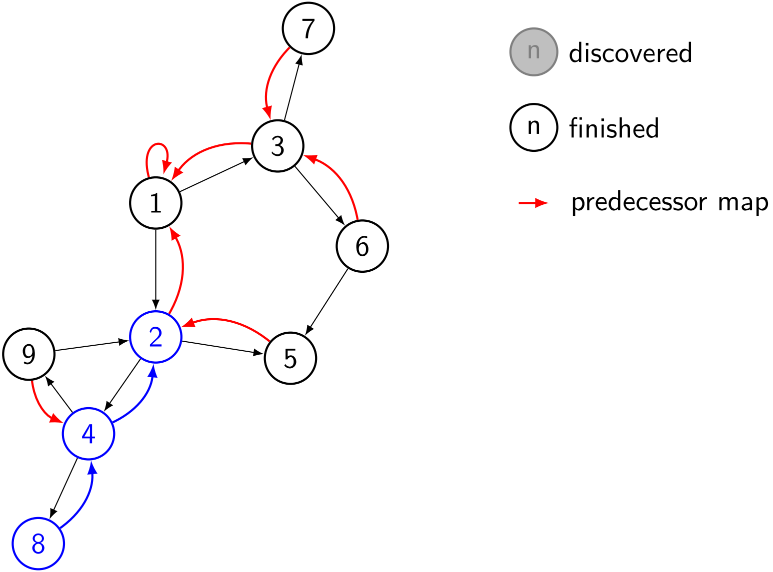
GETPATH(8): p = [8]

Example: BFS shortest paths



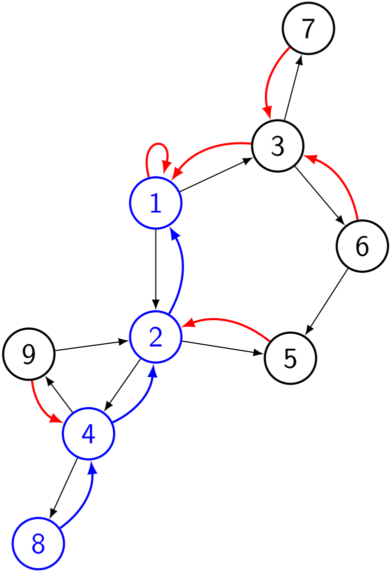
GETPATH(8): p = [4 8]

Example: BFS shortest paths



GETPATH(8): p = [2 4 8]

Example: BFS shortest paths



GETPATH(8): p = [1 2 4 8]

Outline