Graph Search Algorithms

AU4606: Network Optimization

AI4702: Network Intelligence and Optimization

Xiaoming Duan
Department of Automation
Shanghai Jiao Tong University

September 21, 2023

Last time

- Complexity analysis
  - Complexity measures
  - Asymptotic notation
- Data structure
  - Why data structure?
  - Stacks and queues
  - d-heaps

Today

1. General search algorithms
   - Forward search
   - Reverse search
2. Particular search algorithms
   - Breadth-first search
   - Depth-first search
3. Applications
   - Strong connectivity
   - Topological ordering
   - Determining bipartite graphs
   - Finding Eulerian circuits in undirected graphs

Today

1. General search algorithms
   - Forward search
   - Reverse search
2. Particular search algorithms
   - Breadth-first search
   - Depth-first search
3. Applications
   - Strong connectivity
   - Topological ordering
   - Determining bipartite graphs
   - Finding Eulerian circuits in undirected graphs

What are search algorithms?

- Search algorithms are techniques to find nodes with special properties
  1. Find all nodes that are reachable by directed paths from a specific node
  2. Find all the nodes that can reach a specific node along directed paths
- They can also be utilized (as subroutines) to certify graph properties
  1. Check connectivity and find strongly connected components
  2. Identify a directed cycle, if no exists, find a topological ordering
  3. Determining whether a given network is bipartite
- Some search algorithms find certain objects in graphs
  1. Find Eulerian circuits

Search algorithms: nodes reachable from a source

A typical search process

1. Start from an initial node
2. Explore the neighboring nodes through directed edges
3. Explore the neighbors of neighbors and so on
4. Stop when all nodes are explored or a node of interest is found

Search algorithms: nodes reachable from a source

A typical search process

1. Start from an initial node
2. Explore the neighboring nodes through directed edges
3. Explore the neighbors of neighbors and so on
4. Stop when all nodes are explored or a node of interest is found

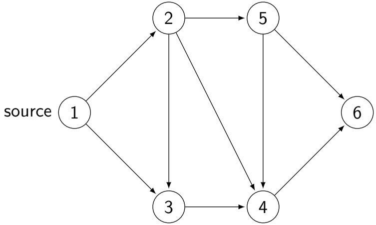
Some details

- How to know if nodes are explored or not
  - Designate all the nodes as being in one of the two states
    1. marked: explored
    2. unmarked: yet to be explored
  - Mark an unmarked node j if j is explored from marked i
- Predecessor relationship: when j is marked from i, set $\text{pred}(j) = i$
- Traversal order: record the order the marked nodes are traversed

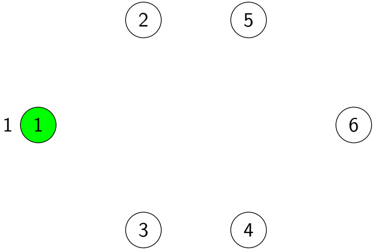
Algorithm Search

```
Unmark all nodes in N
Mark source node s
pred(s) ← 0
next← 1
order(s)← 1
LIST← {s}
while LIST≠ ∅ do
    Select a node i from LIST
    if node i is incident to an admissible arc (i, j) then
        Mark node j
        pred(j) ← i
        next←next+1
        order(j)←next
        Add j to LIST
    else
        Delete node i from LIST
    end if
end while
```

Admissible arc (i, j): node i is marked, and node j is not



(a) A directed graph

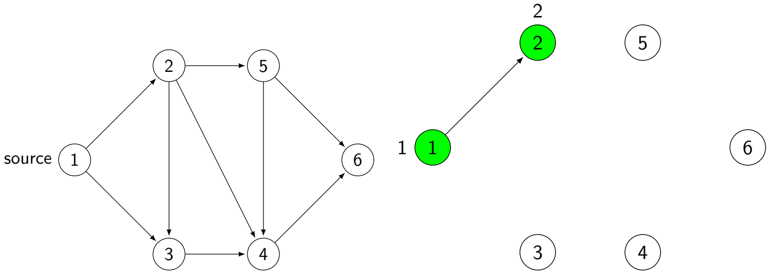


(b) Search process

1. Mark node 1 (source)
2. pred(1) $\leftarrow$ 0
3. next$\leftarrow$ 1
4. order(1) $\leftarrow$ 1

LIST=$\{1\}$

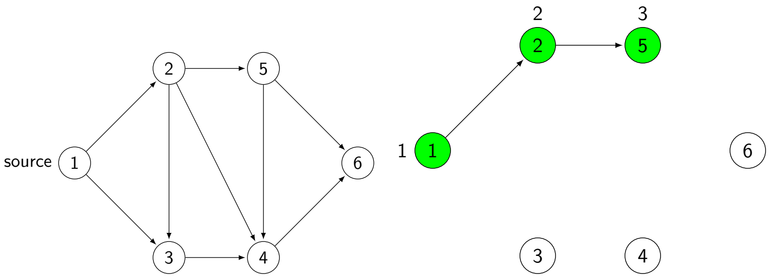
A search example



(a) A directed graph (b) Search process

1. Pick node 1 from LIST
2. Node 1 has an admissible arc $(1, 2)$
3. Mark node 2
4. pred(2) $\leftarrow$ 1
5. next$\leftarrow$ 2
6. order(2) $\leftarrow$ *next*
7. Add 2 to LIST

LIST=$\{1, 2\}$

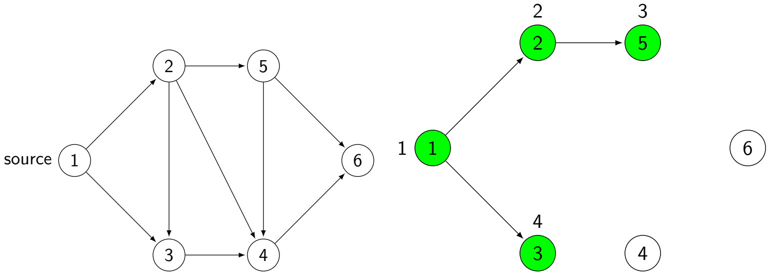


(a) A directed graph (b) Search process

1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 5)$
3. Mark node 5
4. pred$(5) \leftarrow 2$
5. next$\leftarrow 3$
6. order$(5) \leftarrow$ *next*
7. Add 5 to LIST

LIST$=\{1, 2, 5\}$

A search example

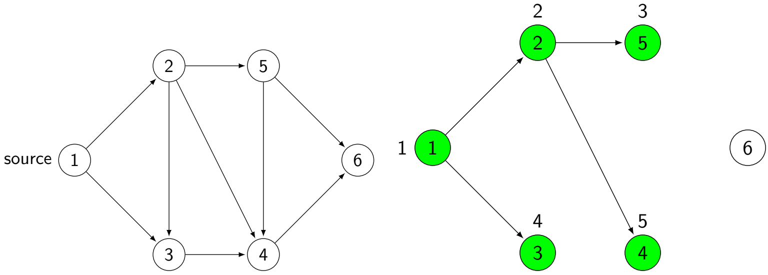


(a) A directed graph (b) Search process

1. Pick node 1 from LIST
2. Node 1 has an admissible arc $(1,3)$
3. Mark node 3
4. pred$(3) \leftarrow 1$
5. next$\leftarrow$ 4
6. order$(3) \leftarrow$ *next*
7. Add 3 to LIST

LIST$=\{1,2,3,5\}$

A search example

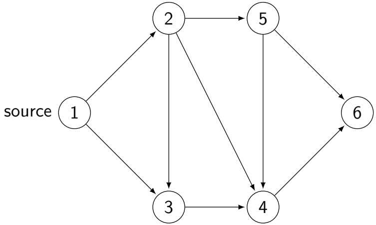


(a) A directed graph (b) Search process

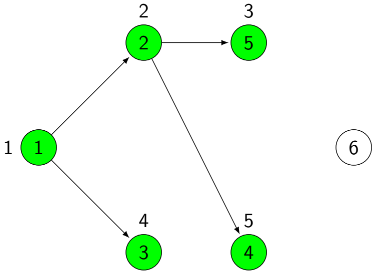
1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 4)$
3. Mark node 4
4. pred$(4) \leftarrow 2$
5. next$\leftarrow$ 5
6. order$(4) \leftarrow$ *next*
7. Add 4 to LIST

LIST$=\{1, 2, 3, 4, 5\}$

A search example



(a) A directed graph

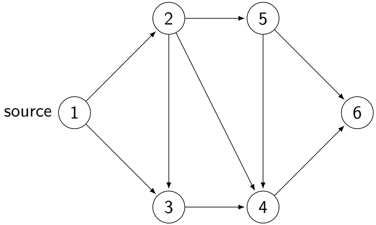


(b) Search process

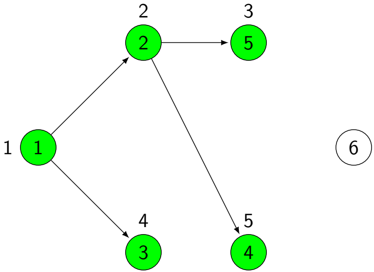
1. Pick node 1 from LIST
2. Node 1 has no admissible arcs
3. Delete node 1 from LIST

LIST$=\{2,3,4,5\}$

A search example



(a) A directed graph

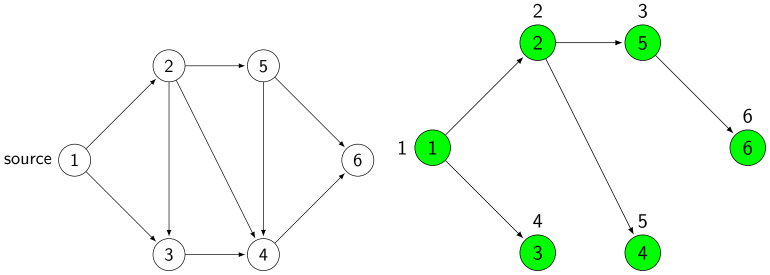


(b) Search process

1. Pick node 3 from LIST
2. Node 3 has no admissible arcs
3. Delete node 3 from LIST

LIST$=\{2, 4, 5\}$

A search example

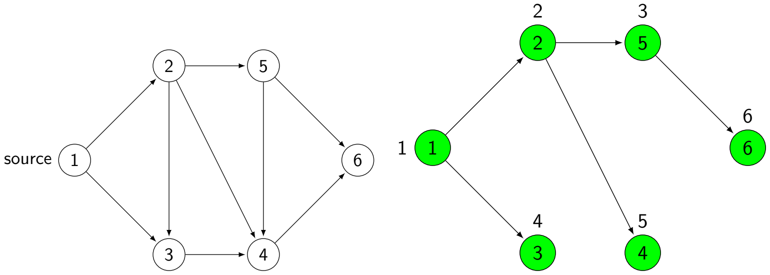


(a) A directed graph (b) Search process

1. Pick node 5 from LIST
2. Node 5 has an admissible arc $(5, 6)$
3. Mark node 6
4. pred$(6) \leftarrow 5$
5. next$\leftarrow 6$
6. order$(6) \leftarrow$ *next*
7. Add 6 to LIST

LIST$=\{2, 4, 5, 6\}$

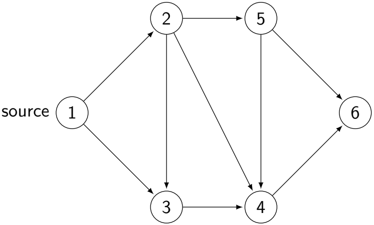
A search example



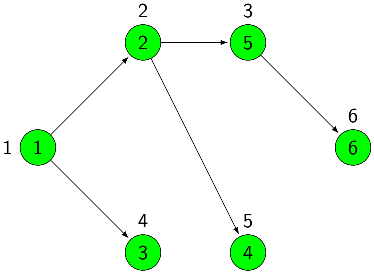
(a) A directed graph (b) Search process

1. Pick node 4 from LIST
2. Node 4 has no admissible arcs
3. Delete node 4 from LIST

LIST$=\{2, 5, 6\}$



(a) A directed graph

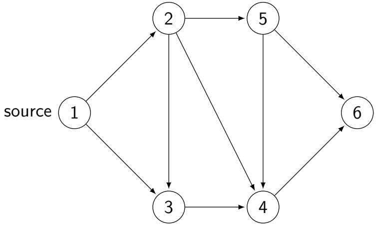


(b) Search process

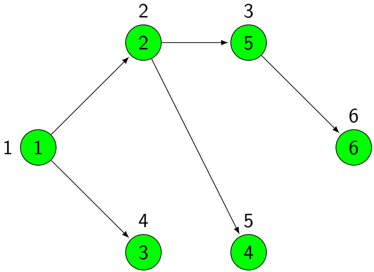
1. Pick node 6 from LIST
2. Node 6 has no admissible arcs
3. Delete node 6 from LIST

LIST$=\{2,5\}$

A search example



(a) A directed graph

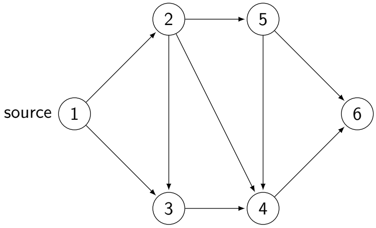


(b) Search process

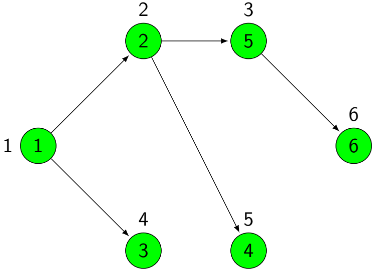
1. Pick node 2 from LIST
2. Node 2 has no admissible arcs
3. Delete node 2 from LIST

LIST={5}

A search example



(a) A directed graph



(b) Search process

1. Pick node 5 from LIST
2. Node 5 has no admissible arcs
3. Delete node 5 from LIST

LIST$=\emptyset$

Forward search: comments

Search tree

- The tree defined by the predecessor indices consisting of marked nodes (why is it a tree?)

Forward search: comments

Search tree

- The tree defined by the predecessor indices consisting of marked nodes (why is it a tree?)

Correctness

1. Soundness: all marked nodes are reachable
2. Completeness: all reachable nodes are marked

Forward search: comments

Search tree

- The tree defined by the predecessor indices consisting of marked nodes (why is it a tree?)

Correctness

1. Soundness: all marked nodes are reachable
2. Completeness: all reachable nodes are marked

Time complexity

- The search algorithm runs in $O(m+n)$ times, in iteration, either
  1. Find an admissible arc and mark a node, $O(m)$
  2. Does not find an admissible arc and delete a node, $O(n)$

Forward search: comments

Search tree

- The tree defined by the predecessor indices consisting of marked nodes (why is it a tree?)

Correctness

1. Soundness: all marked nodes are reachable
2. Completeness: all reachable nodes are marked

Time complexity

- The search algorithm runs in $O(m+n)$ times, in iteration, either
  1. Find an admissible arc and mark a node, $O(m)$
  2. Does not find an admissible arc and delete a node, $O(n)$

How to select a node from LIST is not specified!

- Implementation using queues: breadth-first search
- Implementation using stacks: depth-first search

Reverse search

How do we search for the set of nodes that can reach a destination node t?

Reverse search

How do we search for the set of nodes that can reach a destination node t?

1. Initialize LIST as LIST $= \{t\}$
2. When examining a node, scan the incoming arcs
3. Designate arc (i, j) as admissible if i is unmarked and j is marked

Reverse search

How do we search for the set of nodes that can reach a destination node t?

1. Initialize LIST as LIST $= \{t\}$
2. When examining a node, scan the incoming arcs
3. Designate arc (i, j) as admissible if i is unmarked and j is marked

Forward search vs reverse search

- Forward search creates a directed out-tree rooted at source s
- Reverse search creates a directed in-tree rooted at destination t

Today

1. General search algorithms
   - Forward search
   - Reverse search

2. Particular search algorithms
   - Breadth-first search
   - Depth-first search

3. Applications
   - Strong connectivity
   - Topological ordering
   - Determining bipartite graphs
   - Finding Eulerian circuits in undirected graphs

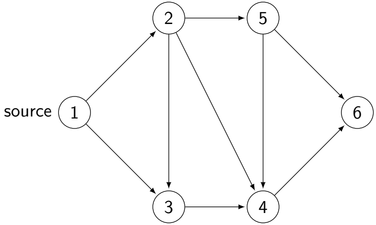
Breadth-first search: procedure

Algorithm Search

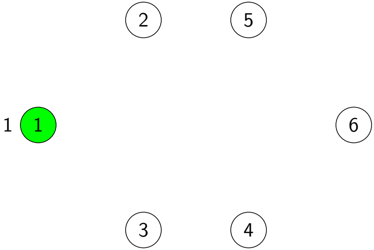
```
Unmark all nodes in N
Mark source node s
pred(s) ← 0
next ← 1
order(s) ← 1
LIST ← {s}
while LIST ≠ ∅ do
    Select a node i from LIST in a first-in-first-out manner
    if node i is incident to an admissible arc (i, j) then
        Mark node j
        pred(j) ← i
        next ← next+1
        order(j) ← next
        Add j to LIST
    else
        Delete node i from LIST
    end if
end while
```

- If LIST is maintained as a queue (FIFO), we get breadth-first search
- Explore all neighbors, then all neighbors of neighbors and so on

BFS: an example



(a) A directed graph

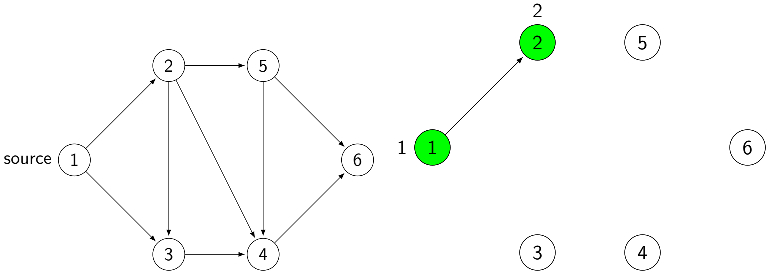


(b) Search process

1. Mark node 1 (source)
2. pred(1) $\leftarrow$ 0
3. next$\leftarrow$ 1
4. order(1) $\leftarrow$ 1

LIST=$\{1\}$

BFS: an example

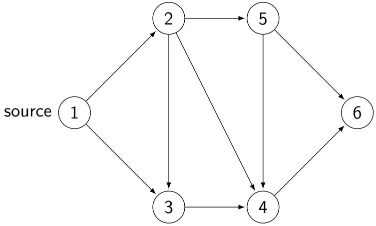


(a) A directed graph (b) Search process

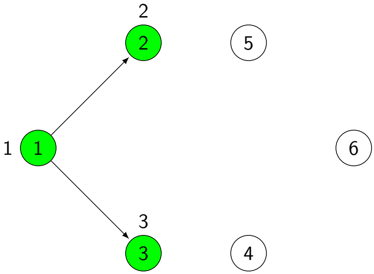
1. Pick node 1 from LIST
2. Node 1 has an admissible arc $(1, 2)$
3. Mark node 2
4. pred$(2) \leftarrow 1$
5. next$\leftarrow 2$
6. order$(2) \leftarrow$ *next*
7. Add 2 to LIST

LIST$=\{1, 2\}$

BFS: an example



(a) A directed graph

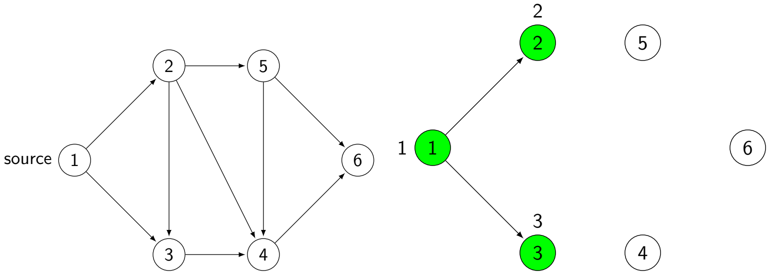


(b) Search process

1. Pick node 1 from LIST
2. Node 1 has an admissible arc $(1, 3)$
3. Mark node 3
4. pred(3) $\leftarrow$ 1
5. next$\leftarrow$ 3
6. order(3) $\leftarrow$ *next*
7. Add 3 to LIST

LIST$=\{1, 2, 3\}$

BFS: an example

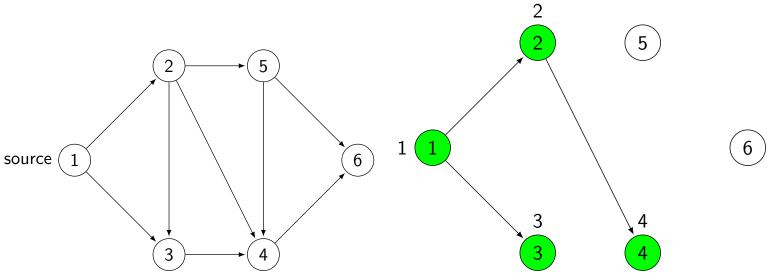


(a) A directed graph (b) Search process

1. Pick node 1 from LIST
2. Node 1 has no admissible arcs
3. Delete node 1 from LIST

LIST $=\{2, 3\}$

BFS: an example

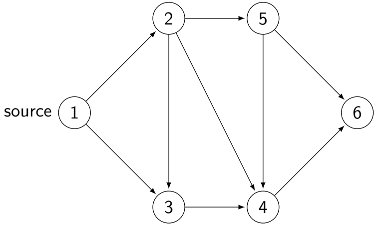


(a) A directed graph (b) Search process

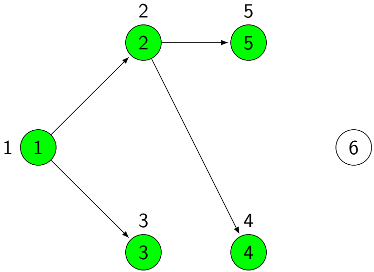
1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 4)$
3. Mark node 4
4. pred$(4) \leftarrow 2$
5. next$\leftarrow 4$
6. order$(4) \leftarrow$ *next*
7. Add 4 to LIST

LIST$=\{2, 3, 4\}$

BFS: an example



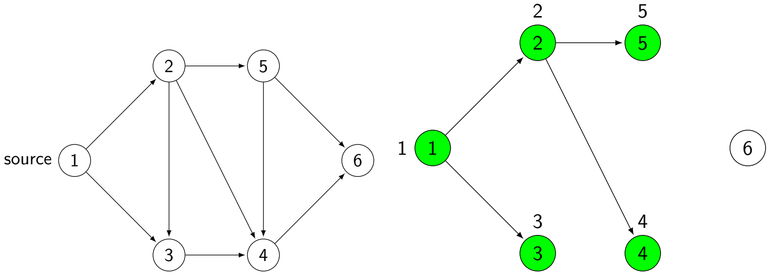
(a) A directed graph



(b) Search process

1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 5)$
3. Mark node 5
4. pred$(5) \leftarrow 2$
5. next$\leftarrow 5$
6. order$(5) \leftarrow$ *next*
7. Add 5 to LIST

LIST$=\{2, 3, 4, 5\}$

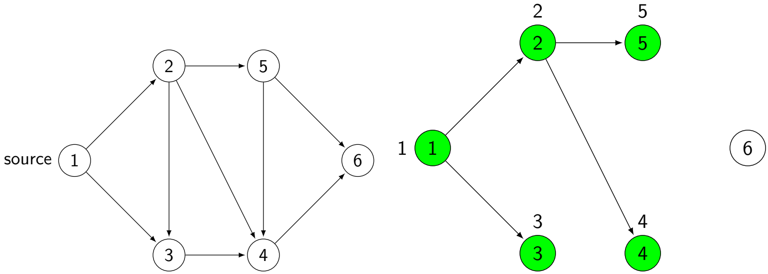


(a) A directed graph (b) Search process

1. Pick node 2 from LIST
2. Node 2 has no admissible arcs
3. Delete node 2 from LIST

LIST=$\{3, 4, 5\}$

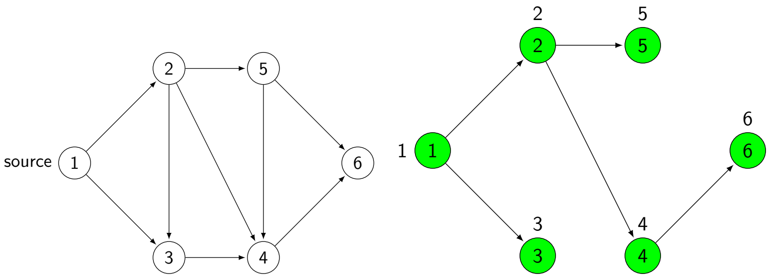
BFS: an example



(a) A directed graph (b) Search process

1. Pick node 3 from LIST
2. Node 3 has no admissible arcs
3. Delete node 3 from LIST

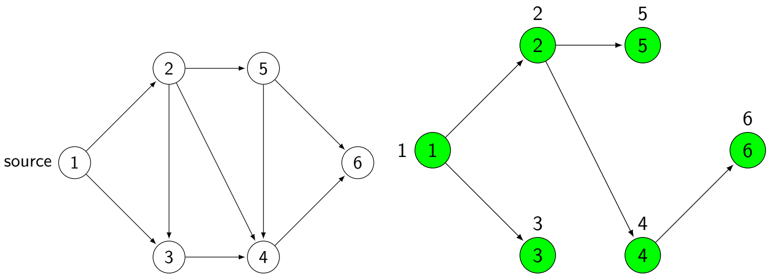
LIST$=\{4,5\}$



(a) A directed graph (b) Search process

1. Pick node 4 from LIST
2. Node 4 has an admissible arc $(4, 6)$
3. Mark node 6
4. pred(6) $\leftarrow$ 4
5. next$\leftarrow$ 6
6. order(6) $\leftarrow$ *next*
7. Add 6 to LIST

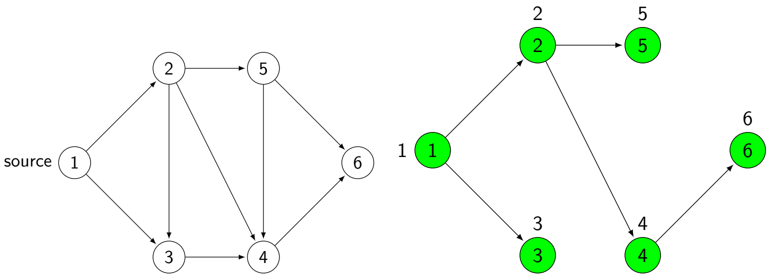
LIST$=\{4, 5, 6\}$



(a) A directed graph (b) Search process

1. Pick node 4 from LIST
2. Node 4 has no admissible arcs
3. Delete node 4 from LIST

LIST$=\{5,6\}$

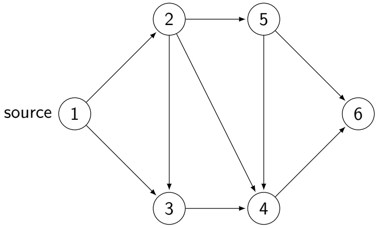


(a) A directed graph (b) Search process

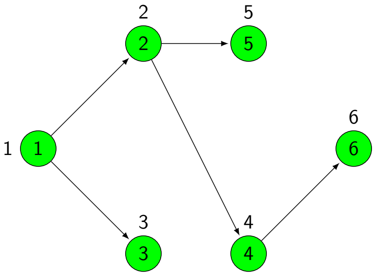
1. Pick node 5 from LIST
2. Node 5 has no admissible arcs
3. Delete node 5 from LIST

LIST={6}

BFS: an example



(a) A directed graph



(b) Search process

1. Pick node 6 from LIST
2. Node 6 has no admissible arcs
3. Delete node 6 from LIST

LIST=$\emptyset$

Breadth-first search: properties

Algorithm Search

```
Unmark all nodes in N
Mark source node s
pred(s) ← 0
next ← 1
order(s) ← 1
LIST ← {s}
while LIST ≠ ∅ do
    Select a node i from LIST in a first-in-first-out manner
    if node i is incident to an admissible arc (i, j) then
        Mark node j
        pred(j) ← i
        next ← next+1
        order(j) ← next
        Add j to LIST
    else
        Delete node i from LIST
    end if
end while
```

Properties of BFS

In a breadth-first search tree, the tree path from the source node s to any node i is a shortest path (i.e., contains the fewest number of arcs among all paths from s to i).

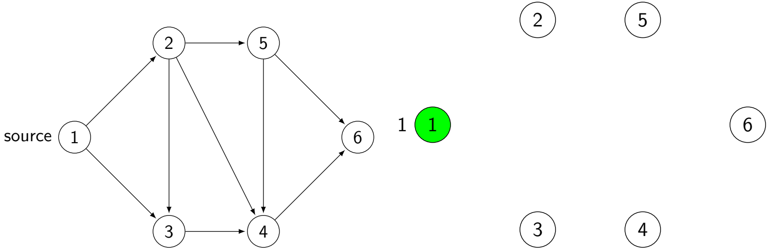
Depth-first search: procedure

Algorithm Search

```
Unmark all nodes in N
Mark source node s
pred(s) ← 0
next← 1
order(s)← 1
LIST← {s}
while LIST≠ ∅ do
    Select a node i from LIST in a first-in-last-out manner
    if node i is incident to an admissible arc (i, j) then
        Mark node j
        pred(j) ← i
        next←next+1
        order(j)←next
        Add j to LIST
    else
        Delete node i from LIST
    end if
end while
```

- If LIST is maintained as a stack (FILO), we get depth-first search
- Create a path as long as possible, until no new node can be marked

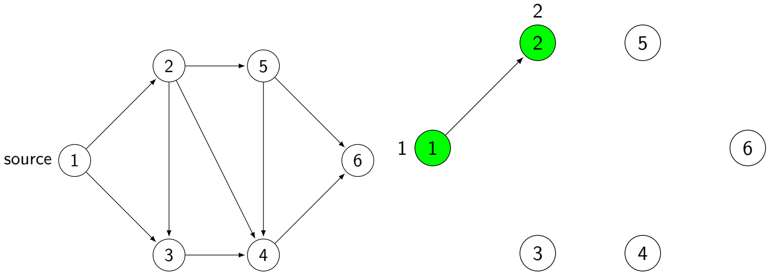
DFS: an example



(a) A directed graph (b) Search process

1. Mark node 1 (source)
2. pred(1) $\leftarrow$ 0
3. next$\leftarrow$ 1
4. order(1) $\leftarrow$ 1

LIST=$\{1\}$

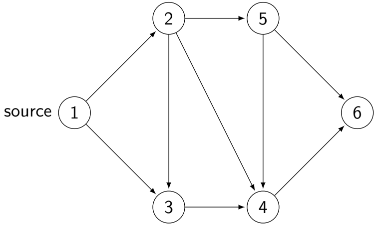


(a) A directed graph (b) Search process

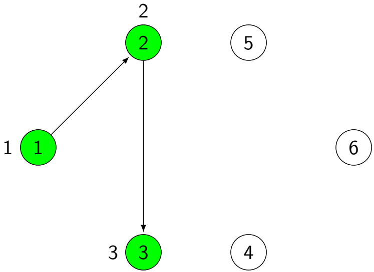
1. Pick node 1 from LIST
2. Node 1 has an admissible arc $(1, 2)$
3. Mark node 2
4. pred(2) $\leftarrow$ 1
5. next$\leftarrow$ 2
6. order(2) $\leftarrow$ *next*
7. Add 2 to LIST

LIST$=\{1, 2\}$

DFS: an example



(a) A directed graph

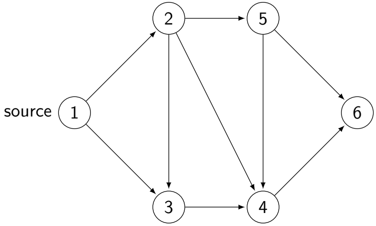


(b) Search process

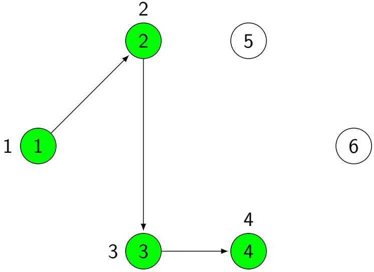
1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 3)$
3. Mark node 3
4. pred$(3) \leftarrow 2$
5. next$\leftarrow 3$
6. order$(3) \leftarrow$ *next*
7. Add 3 to LIST

LIST$=\{1, 2, 3\}$

DFS: an example



(a) A directed graph

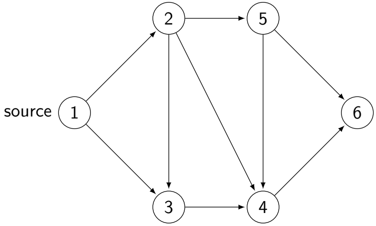


(b) Search process

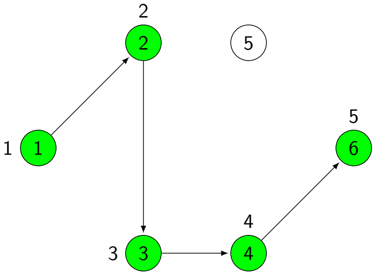
1. Pick node 3 from LIST
2. Node 3 has an admissible arc $(3, 4)$
3. Mark node 4
4. pred$(4) \leftarrow 3$
5. next$\leftarrow$ 4
6. order$(4) \leftarrow$ *next*
7. Add 4 to LIST

LIST$=\{1, 2, 3, 4\}$

DFS: an example



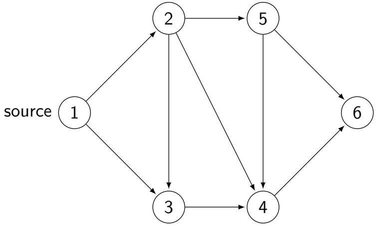
(a) A directed graph



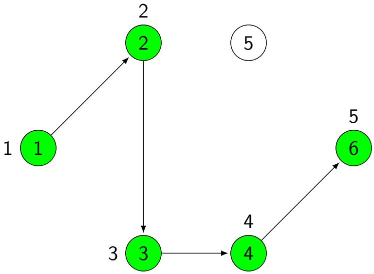
(b) Search process

1. Pick node 4 from LIST
2. Node 4 has an admissible arc $(4, 6)$
3. Mark node 6
4. pred(6) $\leftarrow$ 4
5. next$\leftarrow$ 5
6. order(6) $\leftarrow$ *next*
7. Add 4 to LIST

LIST$=\{1, 2, 3, 4, 6\}$



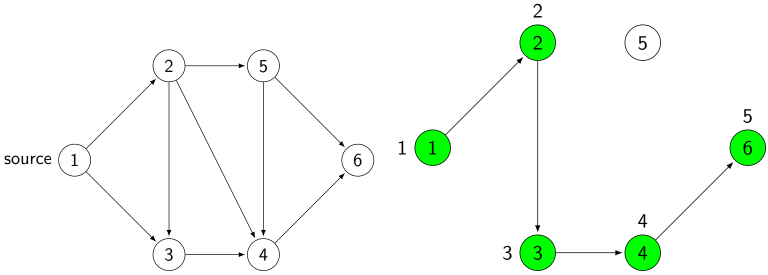
(a) A directed graph



(b) Search process

1. Pick node 6 from LIST
2. Node 6 has no admissible arcs
3. Delete node 6 from LIST

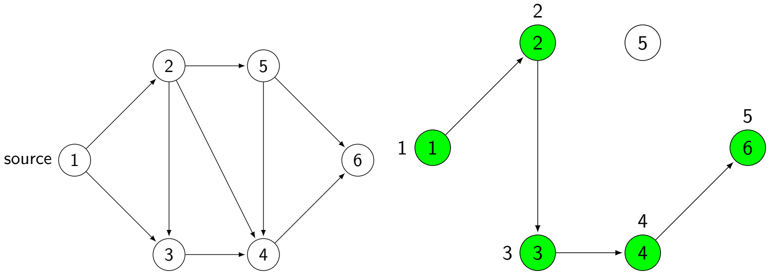
LIST$=\{1, 2, 3, 4\}$



(a) A directed graph (b) Search process

1. Pick node 4 from LIST
2. Node 4 has no admissible arcs
3. Delete node 4 from LIST

LIST$=\{1,2,3\}$

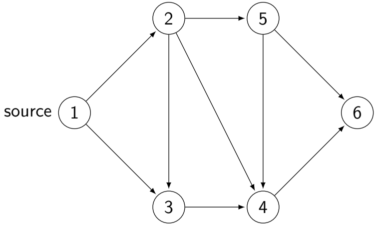


(a) A directed graph (b) Search process

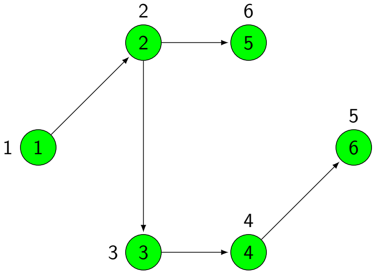
1. Pick node 3 from LIST
2. Node 3 has no admissible arcs
3. Delete node 3 from LIST

LIST$=\{1, 2\}$

DFS: an example



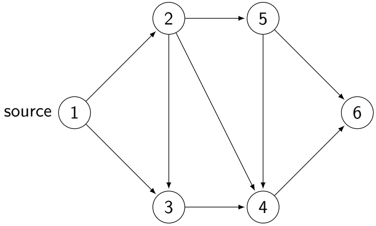
(a) A directed graph



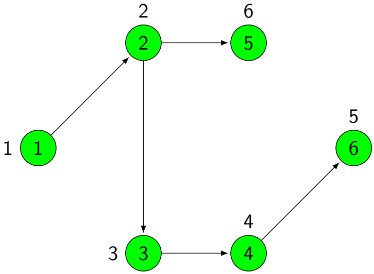
(b) Search process

1. Pick node 2 from LIST
2. Node 2 has an admissible arc $(2, 5)$
3. Mark node 5
4. pred$(5) \leftarrow 2$
5. next$\leftarrow 5$
6. order$(5) \leftarrow$ *next*
7. Add 5 to LIST

LIST$=\{1, 2, 5\}$



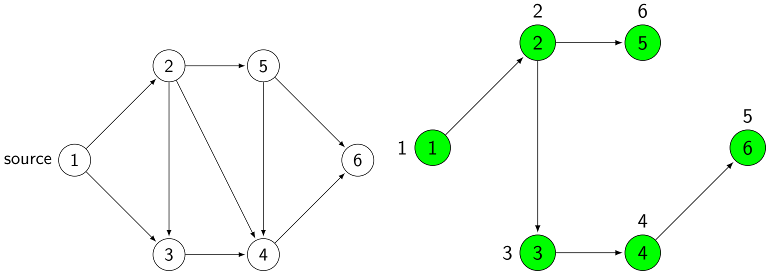
(a) A directed graph



(b) Search process

1. Pick node 5 from LIST
2. Node 5 has no admissible arcs
3. Delete node 5 from LIST

LIST$=\{1, 2\}$

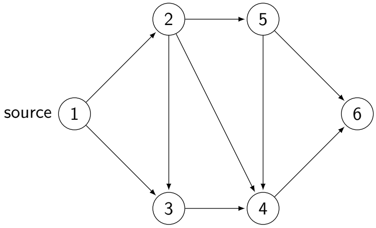


(a) A directed graph (b) Search process

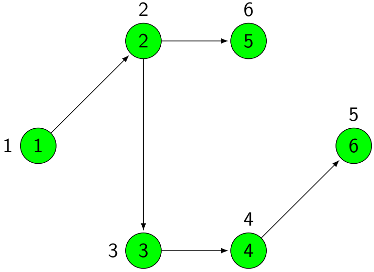
1. Pick node 2 from LIST
2. Node 2 has no admissible arcs
3. Delete node 2 from LIST

LIST=$\{1\}$

DFS: an example



(a) A directed graph



(b) Search process

1. Pick node 1 from LIST
2. Node 1 has no admissible arcs
3. Delete node 1 from LIST

LIST=$\emptyset$

Depth-first search: properties

Algorithm Search

```
Unmark all nodes in N
Mark source node s
pred(s) ← 0
next ← 1
order(s) ← 1
LIST ← {s}
while LIST ≠ ∅ do
    Select a node i from LIST in a first-in-last-out manner
    if node i is incident to an admissible arc (i, j) then
        Mark node j
        pred(j) ← i
        next ← next+1
        order(j) ← next
        Add j to LIST
    else
        Delete node i from LIST
    end if
end while
```

Properties of DFS

1. If node j is a descendant of node i and $j \neq i$, then order(j) $>$ order(i)
2. All the descendants of any node are ordered consecutively in sequence

Today

1. General search algorithms
   - Forward search
   - Reverse search
2. Particular search algorithms
   - Breadth-first search
   - Depth-first search
3. Applications
   - Strong connectivity
   - Topological ordering
   - Determining bipartite graphs
   - Finding Eulerian circuits in undirected graphs

Determining strong connectivity

Start from an arbitrary node s in $G = (N, A)$

- A forward search finds set of nodes U reachable from s
- A reverse search finds set of nodes V that can reach s

Determining strong connectivity

Start from an arbitrary node s in $G = (N, A)$

- A forward search finds set of nodes U reachable from s
- A reverse search finds set of nodes V that can reach s

Is it enough?

Determining strong connectivity

Start from an arbitrary node s in $G = (N, A)$

- A forward search finds set of nodes U reachable from s
- A reverse search finds set of nodes V that can reach s

Is it enough? YES!

Determining strong connectivity

A graph $G = (N, A)$ is strongly connected if and only if $U = V = N$.

Determining strongly connected components

Transitive closure

A transitive closure of a graph $G = (N, A)$ is a matrix $\Gamma = \gamma_{ij}$ defined as follows

$$\gamma_{ij} = \begin{cases} 1, & \text{if graph } G \text{ contains a directed path form node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases}$$

Determining strongly connected components

Transitive closure

A transitive closure of a graph $G = (N, A)$ is a matrix $\Gamma = \gamma_{ij}$ defined as follows

$$\gamma_{ij} = \begin{cases} 1, & \text{if graph } G \text{ contains a directed path form node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases}$$

How to find transitive closure of a graph in $O(mn)$ time?

Determining strongly connected components

Transitive closure

A transitive closure of a graph $G = (N, A)$ is a matrix $\Gamma = \gamma_{ij}$ defined as follows

$$\gamma_{ij} = \begin{cases} 1, & \text{if graph } G \text{ contains a directed path form node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases}$$

How to find transitive closure of a graph in $O(mn)$ time?

- Run search algorithm starting from each node once
- Search algorithm runs in $O(m)$ time

Determining strongly connected components

Transitive closure

A transitive closure of a graph $G = (N, A)$ is a matrix $\Gamma = \gamma_{ij}$ defined as follows

$$\gamma_{ij} = \begin{cases} 1, & \text{if graph } G \text{ contains a directed path form node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases}$$

How to find transitive closure of a graph in $O(mn)$ time?

- Run search algorithm starting from each node once
- Search algorithm runs in $O(m)$ time

How to find strongly connected components given the transitive closure?

Determining strongly connected components

Transitive closure

A transitive closure of a graph $G = (N, A)$ is a matrix $\Gamma = \gamma_{ij}$ defined as follows

$$\gamma_{ij} = \begin{cases} 1, & \text{if graph } G \text{ contains a directed path form node } i \text{ to node } j \\ 0, & \text{otherwise.} \end{cases}$$

How to find transitive closure of a graph in $O(mn)$ time?

- Run search algorithm starting from each node once
- Search algorithm runs in $O(m)$ time

How to find strongly connected components given the transitive closure?

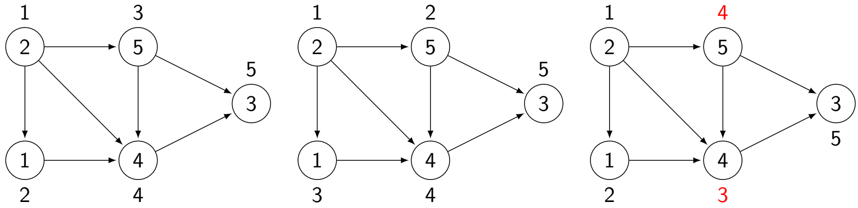
Algorithm Finding SCCs

```
Unlabel all nodes, next ← 1
while There are unlabeled nodes do
    Select an unlabeled node i, label(i) ← next
    for j = 1 : n do
        if γ_ij = 1 and γ_ji = 1 then
            label(j) ← next
        end if
    end for
    next ← next + 1
end while
```

Topological ordering

Label nodes of a network $G = (N, A)$ by distinct numbers from 1 to n

- Let order(i) be the label of node i
- The labeling is a topological ordering of nodes if for every arc $(i, j) \in A$, we have order(i) $<$ order(j)



(a) Topologically ordered (b) Topologically ordered (c) Not topol. ordered

- A network might have several topological orderings
- Some networks cannot be topologically ordered

Topological ordering

Topological ordering

A network is acyclic if and only if it possesses a topological ordering.

Algorithm Topological ordering

```
indegree(i) ← 0 for all i ∈ N
for (i, j) ∈ A do
    indegree(j) ← indegree(j) + 1
end for
LIST ← ∅, next ← 0
for i ∈ N do
    if indegree(i) = 0 then
        LIST ← LIST ∪ {i}
    end if
end for
while LIST ≠ ∅ do
    Select a node i from LIST and delete it
    next ← next+1, order(i) ← next
    for (i, j) ∈ A do
        indegree(j) ← indegree(j) − 1
        if indegree(j) = 0 then
            LIST ← LIST ∪ {j}
        end if
    end for
end while
The network is acyclic if and only if next = n and order is a topological ordering
```

Repeatedly find nodes with zero indegree, delete nodes and arcs

Topological ordering

Topological ordering

A network is acyclic if and only if it possesses a topological ordering.

Algorithm Topological ordering

indegree(i) $\leftarrow 0$ for all $i \in N$
for $(i, j) \in A$ **do**
    indegree(j) $\leftarrow$ indegree(j) $+ 1$
end for
LIST$\leftarrow \emptyset$, next$\leftarrow 0$
for $i \in N$ **do**
    if indegree(i) $= 0$ **then**
        LIST$\leftarrow$ LIST $\cup \{i\}$
    end if
end for
while LIST$\neq \emptyset$ **do**
    Select a node i from LIST and delete it
    next$\leftarrow$next+1, order(i)$\leftarrow$next
    for $(i, j) \in A$ **do**
        indegree(j) $\leftarrow$ indegree(j) $- 1$
        if indegree(j) $= 0$ **then**
            LIST$\leftarrow$ LIST $\cup \{j\}$
        end if
    end for
end while
The network is acyclic if and only if next$=$ n and order is a topological ordering

Can also be done using DFS

Trees are bipartite

Why title?

- How to partition node set N of a tree $G = (N, A)$ into N_1 and N_2?

Spanning trees

- A spanning subgraph $G' = (N', A')$ of $G = \{N, A\}$
  1. $N' = N$
  2. $A' \subset A$
- A tree $T = (N, A')$ is a spanning tree of $G = (N, A)$ if T is a spanning subgraph
  1. The set of arcs A' are tree arcs
  2. The set of arcs $A \setminus A'$ are nontree arcs

Spanning trees are bipartite graphs!

Spanning trees and bipartite graphs

Given an arbitrary spanning tree $T = (N, A')$ of a graph $G = (N, A)$. A graph G is bipartite if and only if for every nontree arc $(k, \ell) \in A \setminus A'$, the distance between node k and node ℓ in T is odd.

Determining bipartite graphs

1. Start from any node s, run BFS to obtain search tree T
2. For nontree arc (k, ℓ), check parity of distance between k and ℓ in T

Finding an Eulerian circuit: procedure

- Whether a graph has an Eulerian circuit can be checked in $O(m)$
  1. Check connectivity
  2. Check degrees
- When there is an Eulerian circuit:

Algorithm Finding an Eulerian circuit

STACK $\leftarrow \emptyset$, LIST $\leftarrow \emptyset$
Select an arbitrary node s
STACK.add(s)
while STACK $\neq \emptyset$ **do**
  $i \leftarrow$ STACK.top()
  if i has zero degrees **then**
    LIST $\leftarrow$ [LIST, i]
    STACK.pop()
  else
    Select an edge $(i, j) \in A$
    $A \leftarrow A \setminus \{(i, j)\}$
    STACK.add(j)
  end if
end while

Upcoming

Week 1-8 (AU4606 & AI4702):

- Introduction (1 lecture)
- Preparations (3 lectures)
  - basics of graph theory
  - algorithm complexity and data structure
  - graph search algorithm (this lecture)
- Shortest path problems (next lecture)
- Maximum flow problems (5 lectures)
- Minimum cost flow problems (3 lectures)
- Introduction to multi-agent systems (1 lecture)
- Introduction to cloud networks (1 lecture)

Week 9-16 (AU4606):

- Simplex and network simplex methods (2 lectures)
- Global minimum cut problems (3 lectures)
- Minimum spanning tree problems (3 lectures)