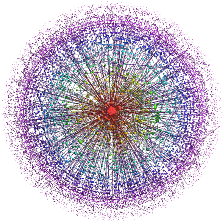

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 11 Depth First Search

Jiamou Liu
The University of Auckland



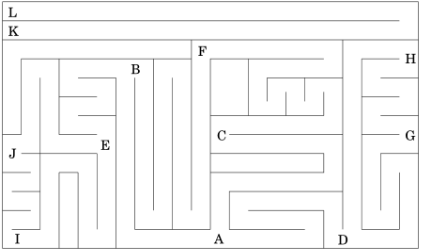
Graph Traversal

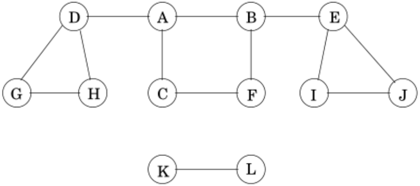
Question

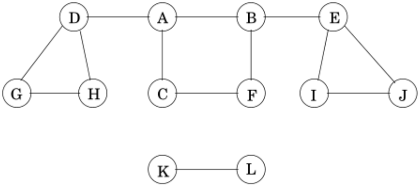
If I use a graph to store a collection of data, how can I search for information in the graph?

Answer

Traverse through each node of the graph.




D
A
B
E
G
H
C
F
I
J
K
L



String: Keep track of the path we are currently on

Chalk: Mark a node after we have finished visiting it

Simulating String and Chalk

Question

Can we use an algorithm to simulate the "string+chalk" procedure to traverse a graph?

Simulating String and Chalk

Question

Can we use an algorithm to simulate the "string+chalk" procedure to traverse a graph?

Strategy: Each node is processed in two stages:

- **Stage 1.** A node is discovered (preprocessed): the first time it is visited
- **Stage 2.** A node is finished (postprocessed): the last time it is visited

Simulating String and Chalk

Question

Can we use an algorithm to simulate the "string+chalk" procedure to traverse a graph?

Strategy: Each node is processed in two stages:

- **Stage 1.** A node is discovered (preprocessed): the first time it is visited
- **Stage 2.** A node is finished (postprocessed): the last time it is visited

Graph Traversal Problem

INPUT: A (representation of) digraph G
OUTPUT: Enumeration of all nodes in the digraph following arcs of the digraph

We would like a traversal algorithm that reveals also the link topology of the graph.

Depth First Search

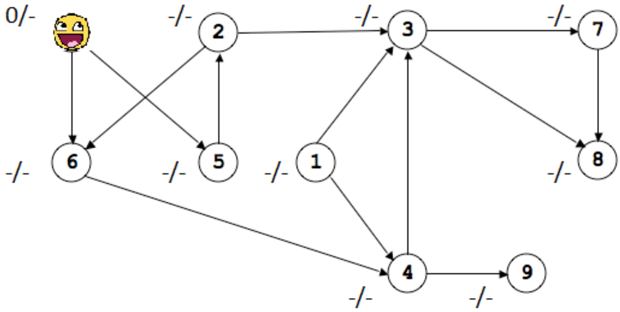
Strategy: Each node is processed in two stages:

- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited

Depth First Search

Strategy: Each node is processed in two stages:

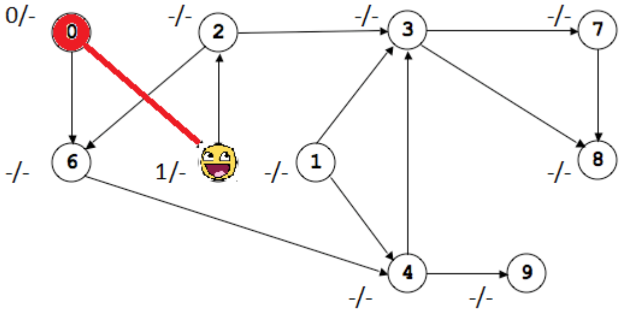
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

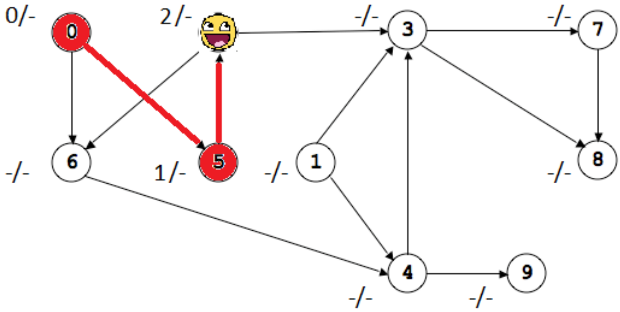
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

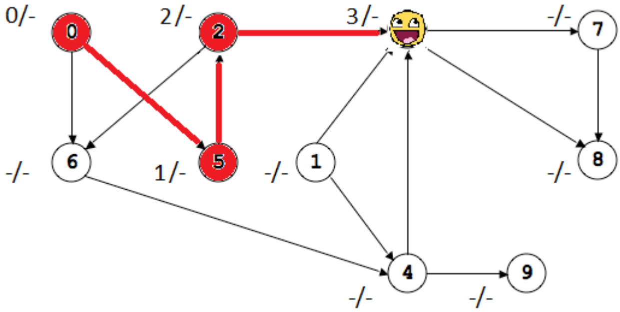
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

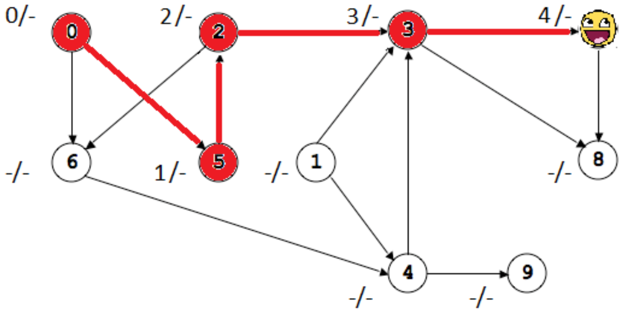
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

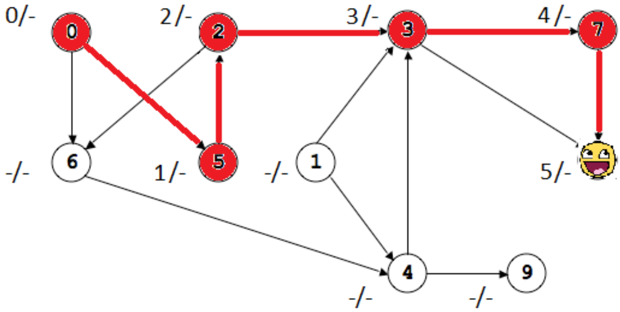
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

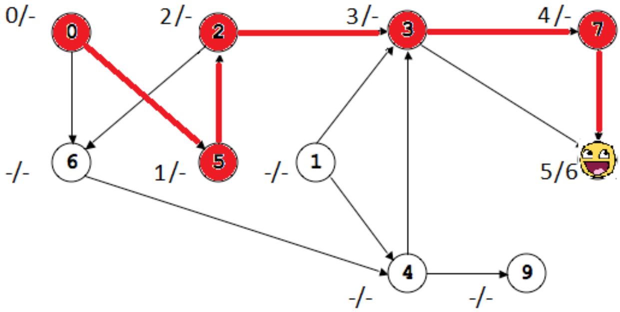
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

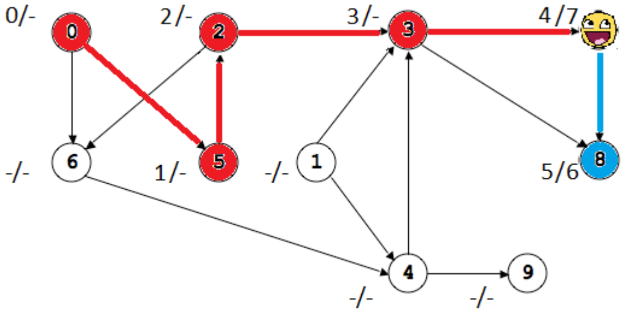
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

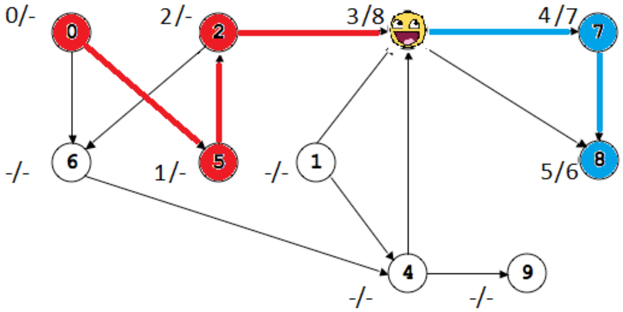
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

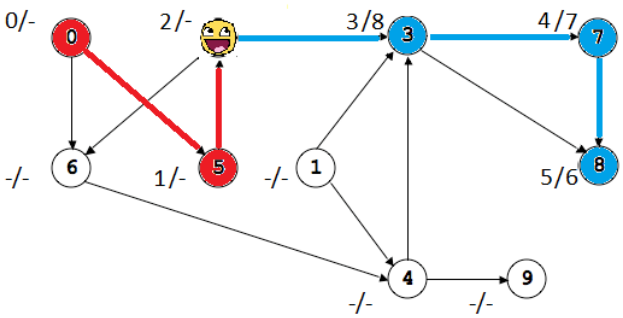
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

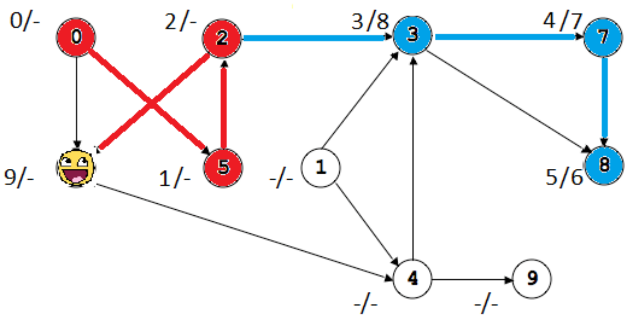
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

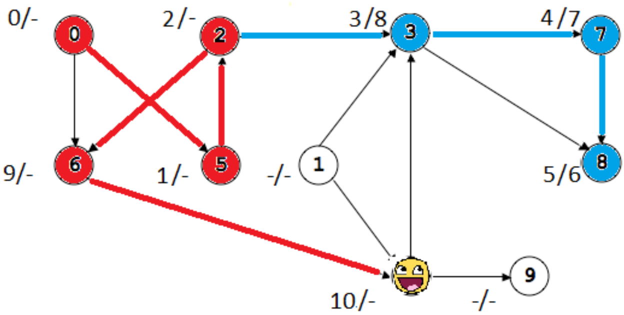
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

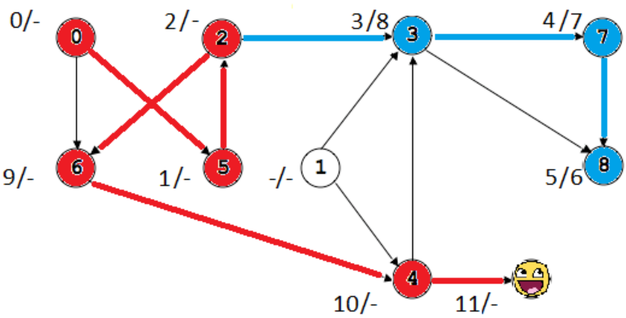
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

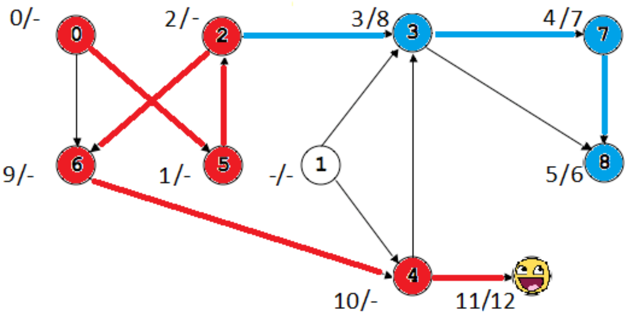
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

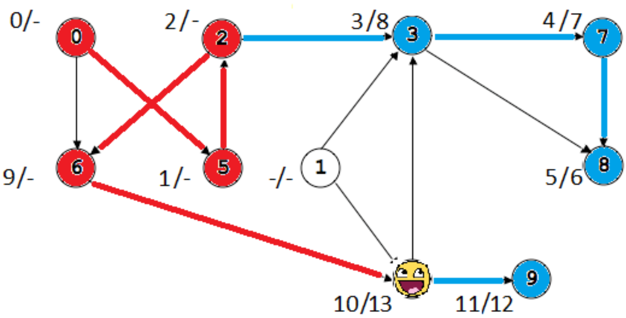
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

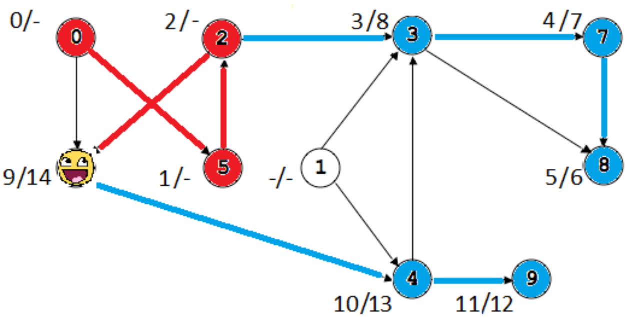
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

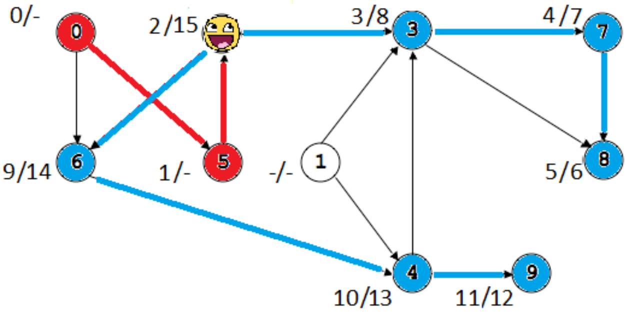
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

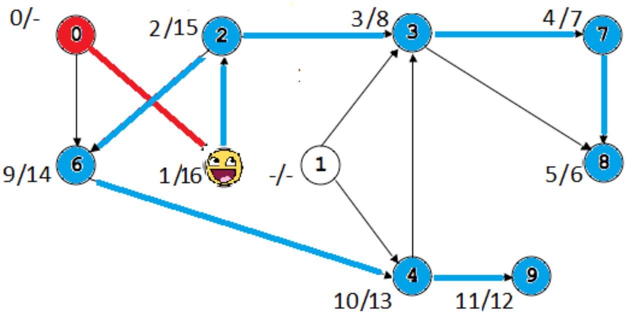
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

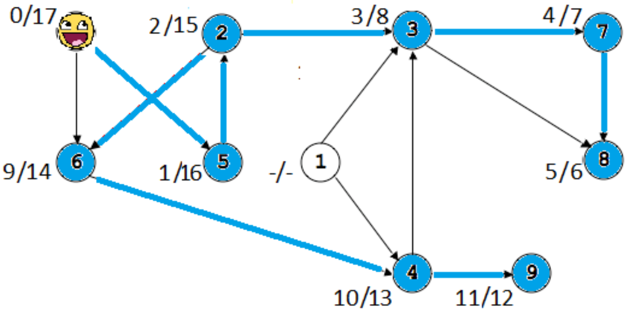
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

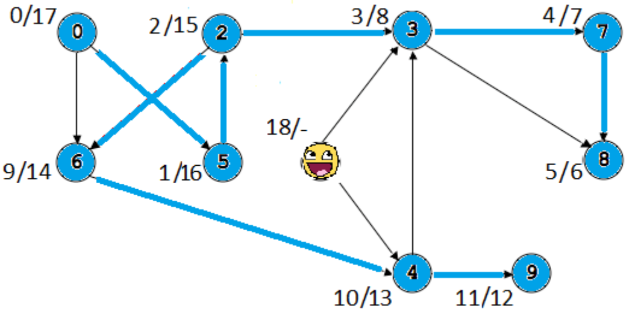
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

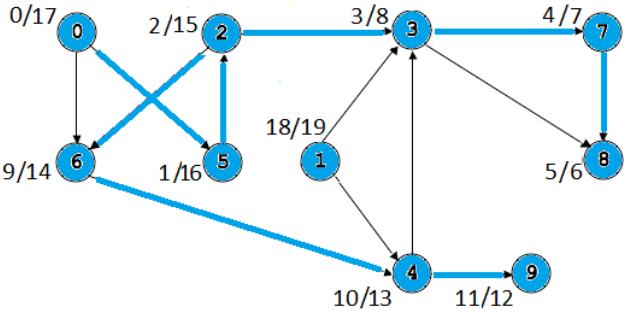
- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search

Strategy: Each node is processed in two stages:

- Stage 1. A node is discovered: the first time it is visited
- Stage 2. A node is finished: the last time it is visited



Depth First Search: Recursive Implementation

Maintain visited(v) for every $v \in V$.

Algorithm dfs(G)

INPUT: A digraph G
for $v \in V$ **do**
  visited(v) $\leftarrow$ *false*
for $v \in V$ **do**
  if visited(v) is false **do**
    `call` explore(G, v)

Algorithm explore(G, v)

INPUT: A digraph G and a node v
visited(v) $\leftarrow$ *true*
`call` discover(v) (perform operations to discover v)
for $(v, u) \in E$ **do**
  if $\neg$ visited(u) **do**
    `call` explore(G, u)
`call` finish(v) (perform operations to finish v)

Depth First Search: Stack Implementation

Note: The current path changes in a FILO order.

Algorithm explore_stack(G, v)

INPUT: A digraph G, and a starting node v

```
create an empty stack S
push v to S
visited(v) ← true
while S ≠ ∅ do
    u ← top element of S
    call discover(u)
    w ← first node such that (u, w) ∈ E and visited(w) is false
    if w does not exist then do
        call finish(u)
        pop u from S
    else do
        push w to S
        visited(w) ← true
```

Depth First Search: Complexity

Analysis

- Discover and finish each node
- Visiting the out-neighbors of each node

Depth First Search: Complexity

Analysis

- Discover and finish each node: $O(n)$
- Visiting the out-neighbors of each node:
  $O(n + m)$ (with adj.list); $O(n^2)$ (with adj.matrix)

Depth First Search: Complexity

Analysis

- Discover and finish each node: $O(n)$
- Visiting the out-neighbors of each node:
  $O(n+m)$ (with adj.list); $O(n^2)$ (with adj.matrix)

Fact

The DFS algorithm takes $O(n+m)$ time with adjacency list and $O(n^2)$ with adjacency matrix.

DFS and Reachability

Definition (Reachability)

We say a node u is reachable from a node v in a graph G if there is a path that starts at v and ends at u.

DFS and Reachability

Definition (Reachability)

We say a node u is reachable from a node v in a graph G if there is a path that starts at v and ends at u.

Fact.

Suppose we run explore(G, v) on input graph G and node v in G, any node u is visited by the algorithm if and only if it is reachable from v.

Why?

1. If u is visited, then u is reachable.
   This is because DFS only follows edges in G.
2. If u is reachable, then u is visited.

DFS and Reachability

Definition (Reachability)

We say a node u is reachable from a node v in a graph G if there is a path that starts at v and ends at u.

Fact.

Suppose we run explore(G, v) on input graph G and node v in G, any node u is visited by the algorithm if and only if it is reachable from v.

Why?

1. If u is visited, then u is reachable.

   This is because DFS only follows edges in G.

2. If u is reachable, then u is visited.

   Proof. Suppose w is reachable but not visited.
   Then there is a path $v \rightsquigarrow w$.
   Take the last visited u on the path ($v \rightsquigarrow u \to u' \rightsquigarrow w$).
   Then we must visit u' from u. Contradiction.

Depth First Search and Search Trees

Definition [Search Forest]

- DFS defines one or more search trees, forming a forest in the digraph.
- These trees contain all paths DFS used to visit nodes in G.

Depth First Search and Search Trees

Definition [Search Forest]

- DFS defines one or more search trees, forming a forest in the digraph.
- These trees contain all paths DFS used to visit nodes in G.

Question

How could we identify the search trees while running DFS?

Depth First Search and Search Trees

Definition [Search Forest]

- DFS defines one or more search trees, forming a forest in the digraph.
- These trees contain all paths DFS used to visit nodes in G.

Question

How could we identify the search trees while running DFS?

Solution

Maintain a “timer” in the algorithm, and two times $pre(u)$ and $post(u)$ for each node u:

- $pre(u)$: the time step in which u is first visited.
- $post(u)$: the time step in which u is last visited.

Observation

- If u is an ancestor of v, then

$$pre(u) < pre(v) < post(v) < post(u)$$

- If v is an ancestor of v, then

$$pre(v) < pre(u) < post(u) < post(v)$$

- If neither case, then

$$pre(u) < post(u) < pre(v) < post(v)$$

or

$$pre(v) < post(v) < pre(u) < post(u)$$

Therefore we can represent the search trees in parenthesis form:

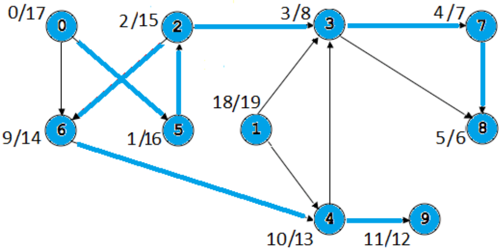
DFS-Forest(G)

Write down a sequence of symbols (n and n), where $n \in \{1, \ldots, n\}$ such that:

If $pre(u) = k$, then the kth symbol is $(u$

If $post(u) = k$, then the kth symbol is $u)$

e.g. (0 (5 (2 (3 (7 (8 8) 7) 3) (6 (4 (9 9) 4) 6) 2) 5) 0) (1 1)



Summary

- Graph traversal problem
- Depth-first search: Two implementations
  1. recursive implementation
  2. non-recursive stack-based implementation
- Depth-first search: Reachability
  v is reachable $\equiv$ v is visited by DFS
- Depth-first search trees:
  - *pre*(v): The time step in which v is first visited
  - *post*(v): The time step in which v is last visited
  - The parenthesis form gives the search trees


THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND