Graph Traversal

Definition

Methods to visit all nodes in a graph.

- Depth-first Search
- Breadth-first Search

Each search produces a spanning tree: the nodes of the tree are the nodes of the graph, and the arcs are a subset of the arc of the graph

Depth-first Search

Pick an arbitrary node v, and start the process there:

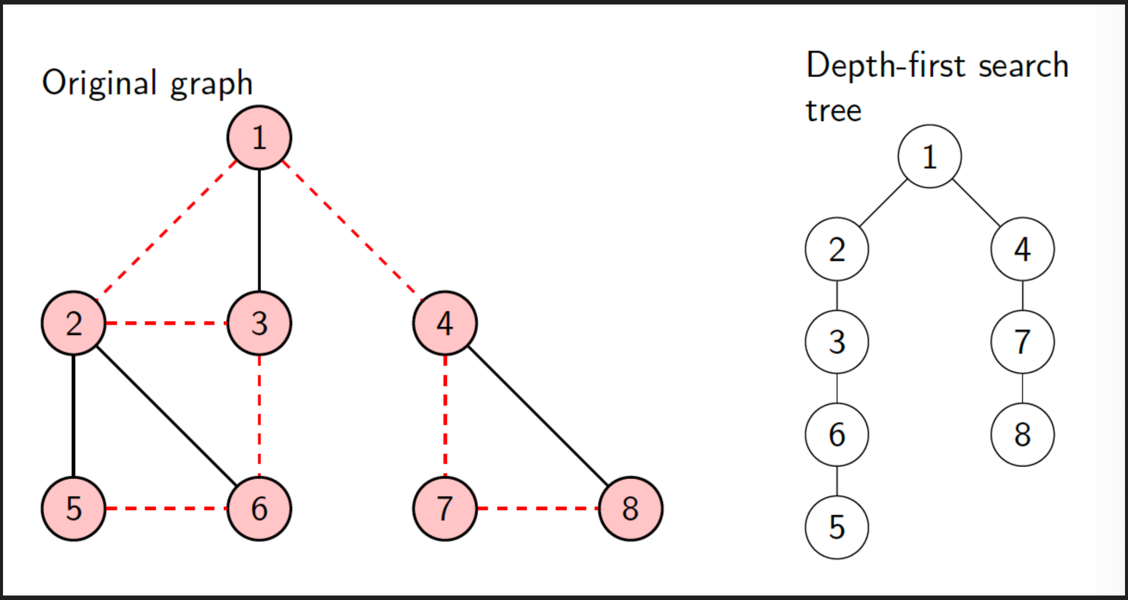
1. we visit it
    2. then if there is an adjacent unvisited node
2. we start the process again at the adjacent node
3. we continue recursively, until we visit a node and there are no unvisited adjacent nodes. - *this means that this call of `DepthFirstSearch` has been satisfied, and control moves back to the previous call of `DepthFirstSearch` where the process continues*
4. Eventually all the nodes adjacent to v will be visited
5. If the remain any other nodes in G unvisited, arbitrarily pick one of the unvisied nodes, and start the process at this new node
6. Continue until no unvisited nodes remain

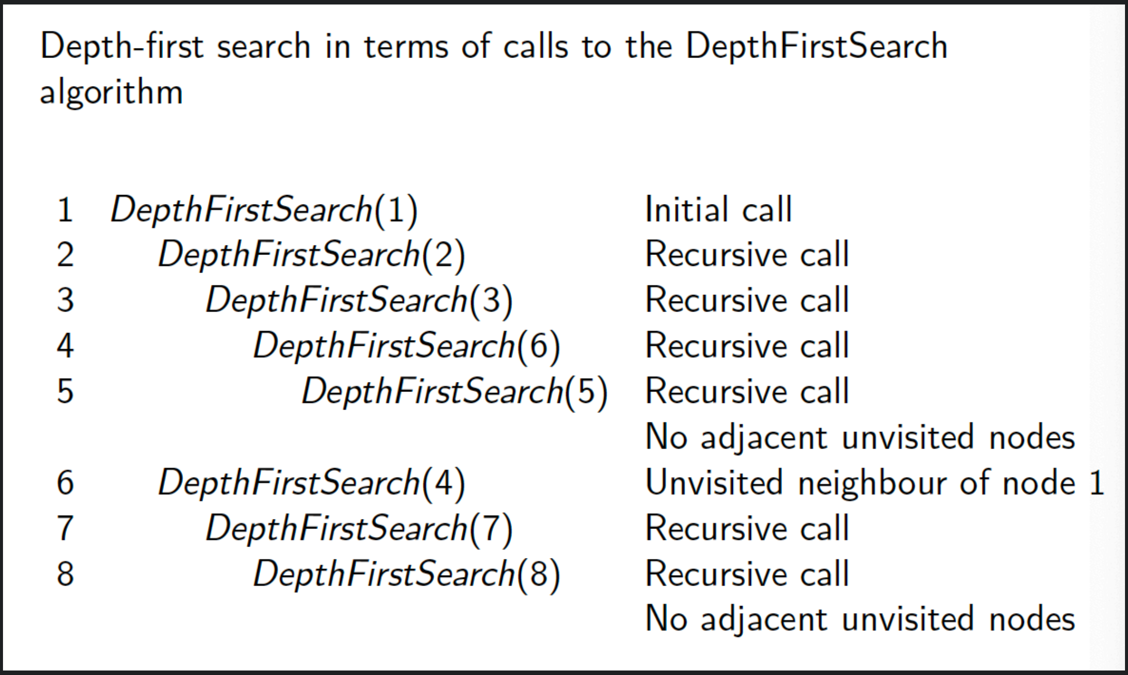
Pseudocode

```
FUNCTION Search(G)
     FOR v in N
          mark[v] = notVisited // set up an array called mark
     FOR v in N // arbitrarily pick
          IF mark[v] <> Visited THEN
               DepthFirstSearch(v)

FUNCTION DepthFirstSearch(v)
     mark[v] = visted
     FOR w in nodes adjacent to v
          IF mark[w] <> Visited
               DepthFirstSearch(w)
```

Example 1

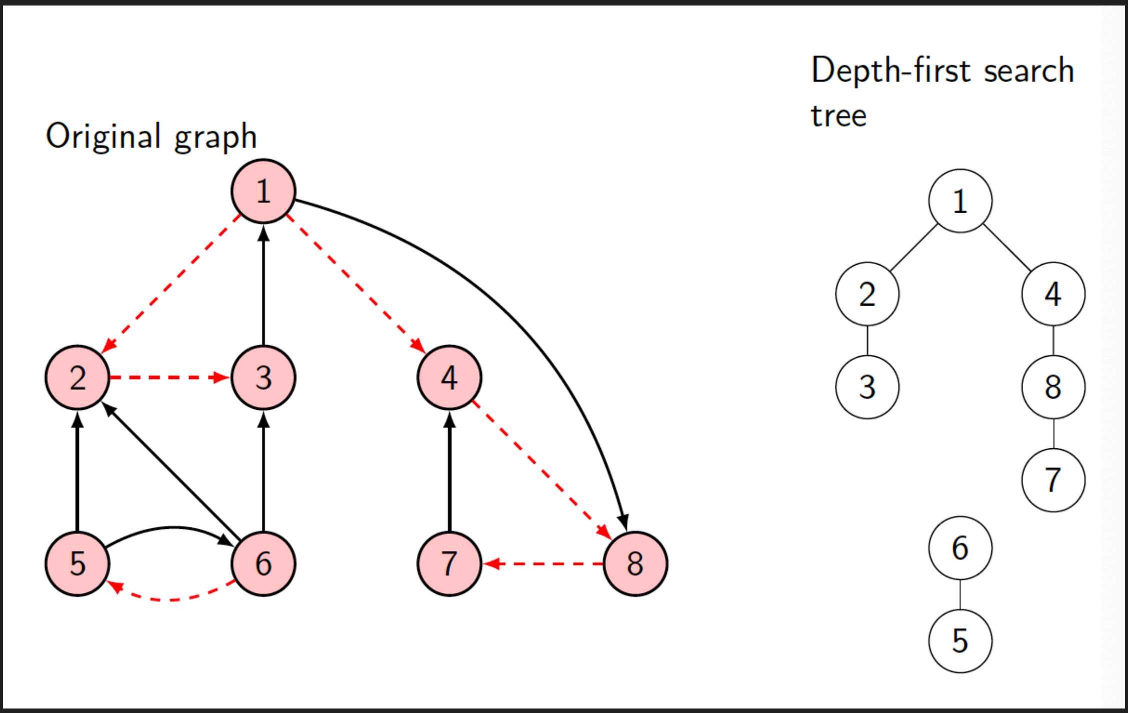


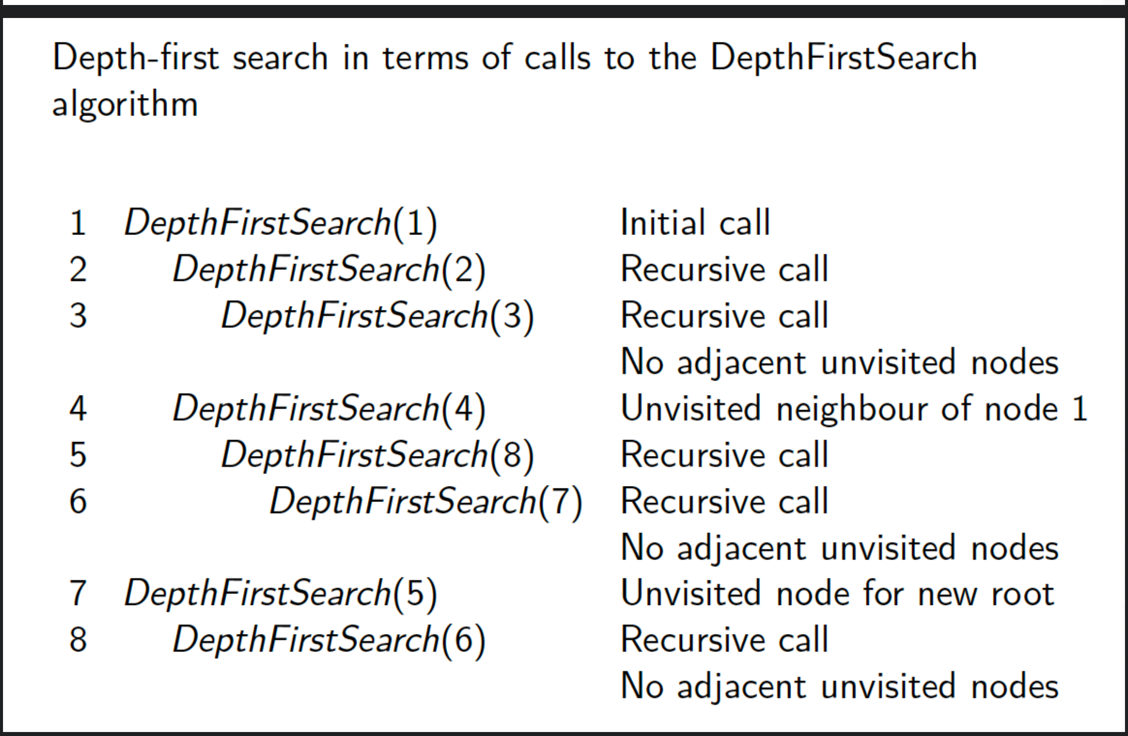


Depth-first search in terms of calls to the DepthFirstSearch algorithm

```
1  DepthFirstSearch(1)                  Initial call
2      DepthFirstSearch(2)              Recursive call
3          DepthFirstSearch(3)          Recursive call
4              DepthFirstSearch(6)      Recursive call
5                  DepthFirstSearch(5)  Recursive call
                                        No adjacent unvisited nodes
6      DepthFirstSearch(4)              Unvisited neighbour of node 1
7          DepthFirstSearch(7)          Recursive call
8              DepthFirstSearch(8)      Recursive call
                                        No adjacent unvisited nodes
```

Example 2: Directed Graph




Depth-first search in terms of calls to the DepthFirstSearch algorithm

1	*DepthFirstSearch*(1)	Initial call
2	*DepthFirstSearch*(2)	Recursive call
3	*DepthFirstSearch*(3)	Recursive call
		No adjacent unvisited nodes
4	*DepthFirstSearch*(4)	Unvisited neighbour of node 1
5	*DepthFirstSearch*(8)	Recursive call
6	*DepthFirstSearch*(7)	Recursive call
		No adjacent unvisited nodes
7	*DepthFirstSearch*(5)	Unvisited node for new root
8	*DepthFirstSearch*(6)	Recursive call
		No adjacent unvisited nodes

Comparing Depth-First Search and Breadth-First Search

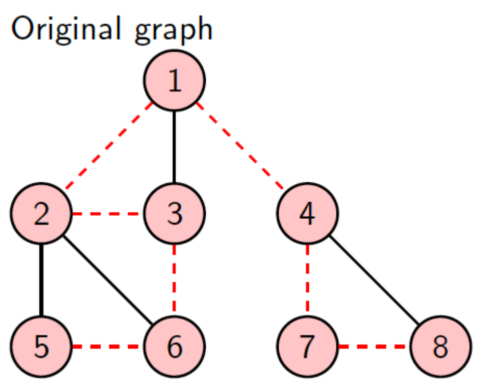
To compare, we consider a ***non-recursive*** formulation of depth-first search

Pseudocode

```
PROCEDURE DepthFirstSearch2(v)
     P = emptyStack
     mark[v] = visited
     push v on P
     WHILE P <> Null
          WHILE there is a node w adjacent to top(P)
//mark[w] <> visited
```

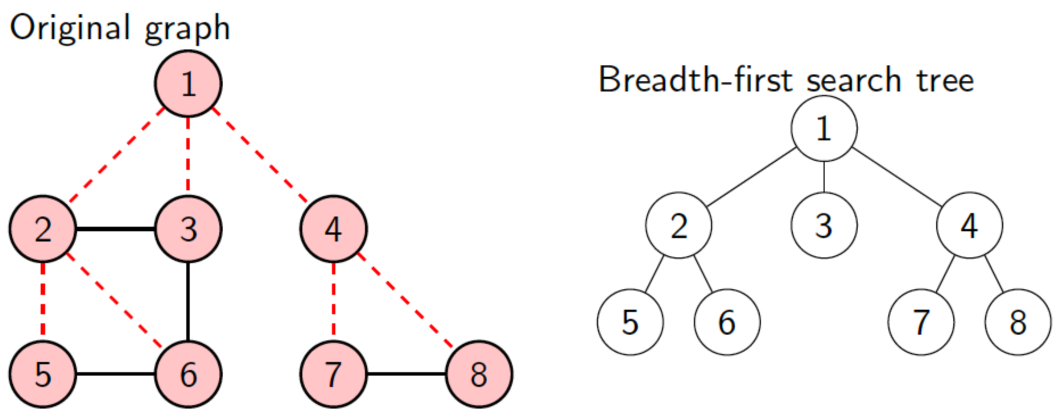
```
            mark[w] = visited
            push w on P
        pop top(P)

PROCEDURE Search2(G)
    FOR each v IN N
        mark[v] = not-visited
    FOR each v IN N
        IF mark[v] <> visited
            DepthFirstSearch2(v)
```



Operation	Stack after operation
Push 1	1
Push 2	2,1
Push 3	3,2,1
Push 6	6,3,2,1
Push 5	5,6,3,2,1
Pop 5	6,3,2,1
Pop 6	3,2,1
Pop 3	2,1
Pop 2	1
Push 4	4,1
Push 7	7,4,1
Push 8	8,7,4,1
Pop 8	7,4,1
Pop 7	4,1
Pop 4	1
Pop 1	emptystack

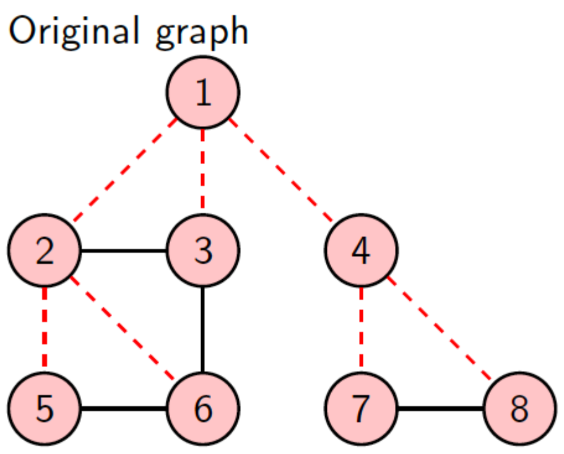
Breadth-First Search



Pseudocode

```
PROCEDURE BreadthFirstSearch(v)
    Q = emptyQueue
    mark[v] = visited
    enqueue v into Q
    WHILE Q <> NULL
        u = first(Q)
        dequeue u from Q
        FOR each node w adjacent to u
            IF mark[w] <> visited
                mark[w] = visited
                enqueue w into Q

PROCEDURE Search3(G)
    FOR v IN N
        mark[v] = not-visited
    FOR v IN N
        IF mark[v] <> visited
            BreadthFirstSearch(v)
```



Operation	Queue
Enqueue 1	1
Dequeue 1	emptyqueue
Enqueue 2	2
Enqueue 3	2,3
Enqueue 4	2,3,4
Dequeue 2	3,4
Enqueue 5	3,4,5
Enqueue 6	3,4,5,6
Dequeue 3	4,5,6
Dequeue 4	5,6
Enqueue 7	5,6,7
Enqueue 8	5,6,7,8
Dequeue 5	6,7,8
Dequeue 6	7,8
Dequeue 7	8
Dequeue 8	emptyqueue