Breadth First Search (BFS)

Time complexity

Using adjacency list

(n = number of vertex
and m = number of edges)

Recall: $\Sigma_{v \in V} \texttt{deg}(v) = 2m$

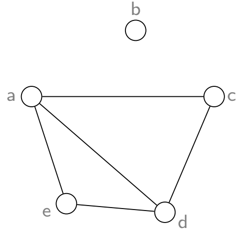
Total running time of the while loop
$= \Sigma_{v \in V} (\texttt{deg}(v) + 1)$
$= \Sigma_{v \in V} \texttt{deg}(v) + \Sigma_{v \in V} 1$
$= O(2m + n)$
$= O(m + n)$ (or $O(|E| + |V|)$)

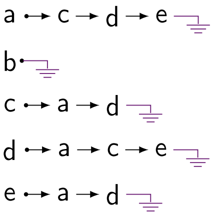
BFS(G, s)

for each vertex $u \in G.V$
    flag[u] = *false*
Q =empty queue
flag[s] = *true*
ENQUEUE(Q, s)
while Q is not empty ← Each vertex will enter Q at most once
    v = DEQUEUE(Q)
    for each w adjacent to v ← Each iteration takes time proportional to $\texttt{deg}(v) + 1$ (the number 1 is to account for the case where $\texttt{deg}(v) = 0$ — the work required is 1, not (0).
        if *flag*[w] = *false*
            flag[w] = *true*
            ENQUEUE(Q, w)

Breadth First Search (BFS) (cont.)

Time complexity






```
public int searchAdjacent(Node node) {

  Node currNode = node.getNext();

  while (null != currNode) {
    // do something

    currNode = currNode.getNext();
  }

}
```

Number of iterations
$= \deg(node) + 1$

CPT108 Data Structures and Algorithms

Lecture 23

Graph Traversal: Depth First Search

Depth First Search (DFS)

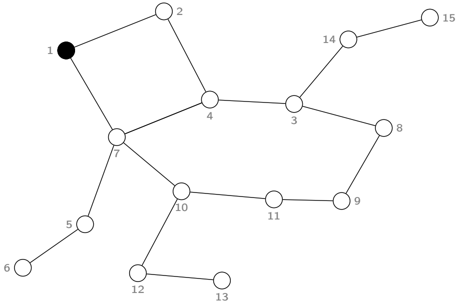
Applications

- Traverse and return value (such as max, min, etc.)
- Find a path from point A to B
- Find connected components
- Detect looping (cycles) and
- Solve combinatorial problems, such as:
  - How may ways are there to arrange something
  - Find all possible combinations of . . .
  - Find all solutions to a puzzle

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



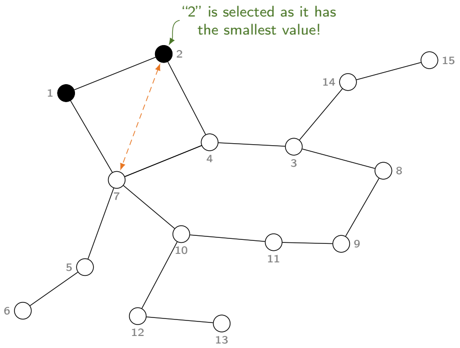
set source $s = \{1\}$
and pick the vertex with
smallest values if more than
one nodes can be chosen

`dfs(s)` $=\{1\}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



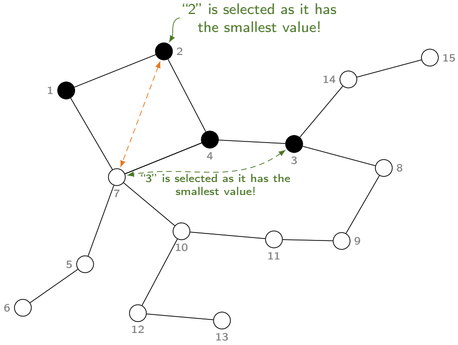
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

`dfs(s)` $= \{$ 1,
2
$\}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



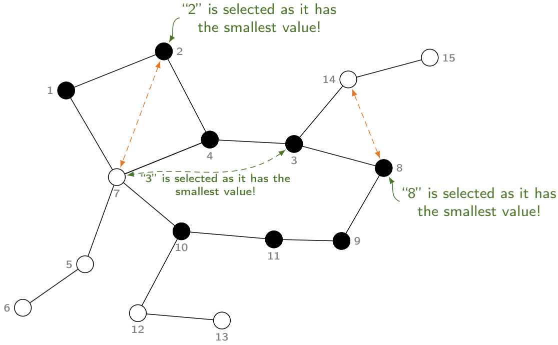
set source $s = \{1\}$ and pick the vertex with *smallest* values if more than one nodes can be chosen

`dfs(s)` $= \{ 1, 2, 4, 3 \}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



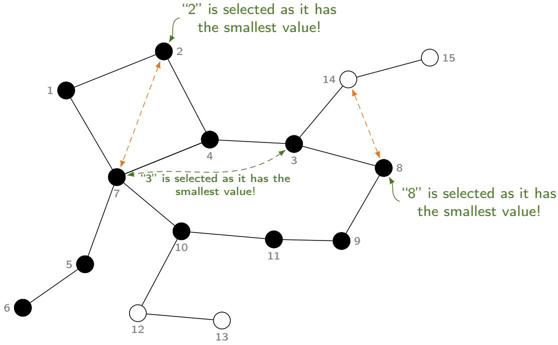
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

`dfs(s)` $= \{\ 1,$
$2, 4, 3,$
$8, 9, 11, 10$
$\}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



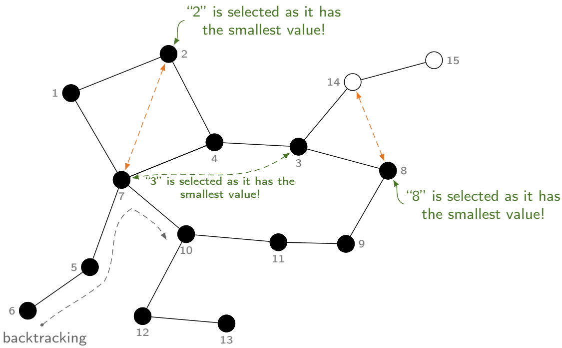
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

`dfs(s)` $= \{$ 1,
2, 4, 3,
8, 9, 11, 10,
7, 5, 6
$\}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



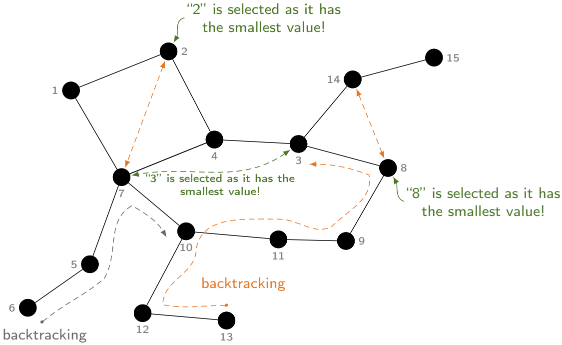
set source $s = \{1\}$ and pick the vertex with *smallest* values if more than one nodes can be chosen

$\texttt{dfs(s)} = \{$ 1, 2, 4, 3, 8, 9, 11, 10, 7, 5, 6, 12, 13 $\}$

Depth First Search (DFS)

Main concept

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes



set source $s = \{1\}$ and pick the vertex with *smallest* values if more than one nodes can be chosen

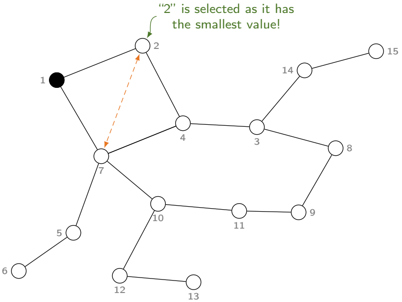
`dfs(s)` $= \{ 1,$
$2, 4, 3,$
$8, 9, 11, 10,$
$7, 5, 6,$
$12, 13,$
$14, 15$
$\}$

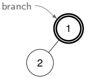
$\Rightarrow$ DFS completed!

Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

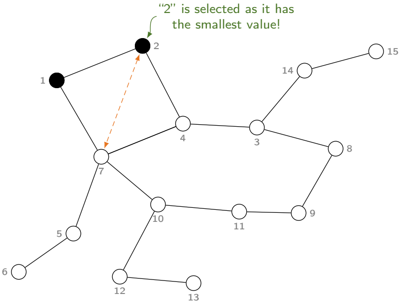


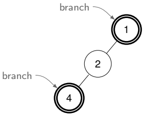


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

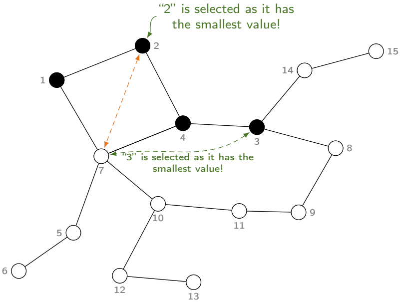


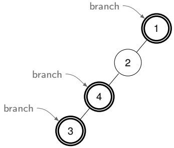


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

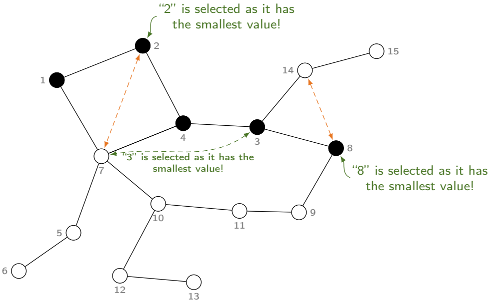


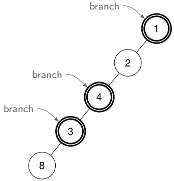


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

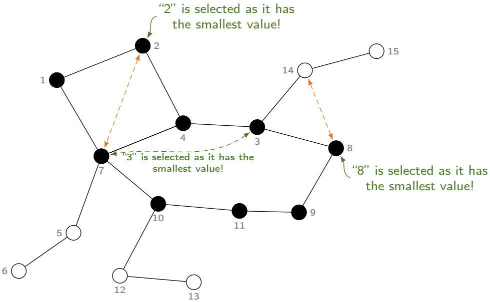


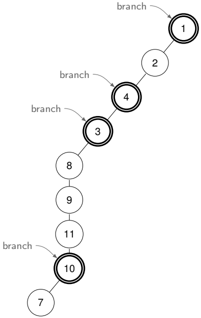


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

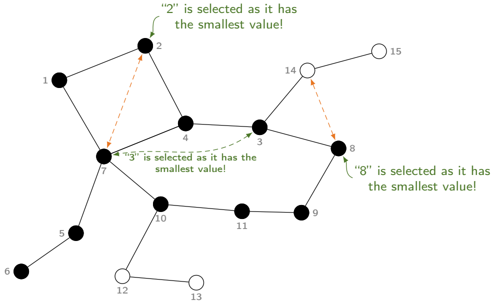


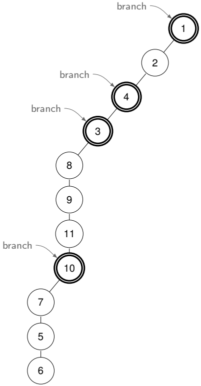


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

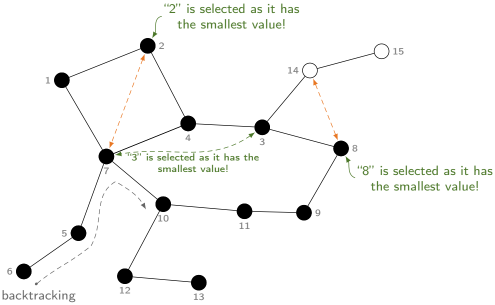


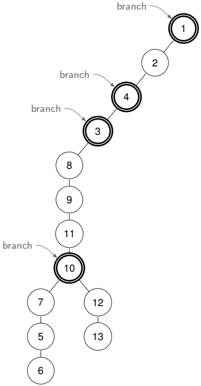


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen

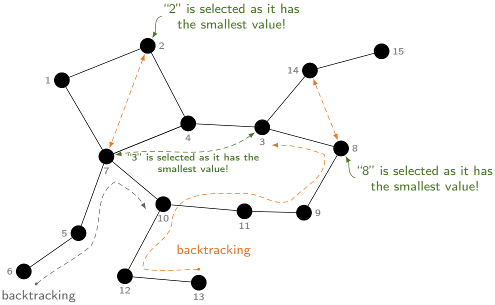


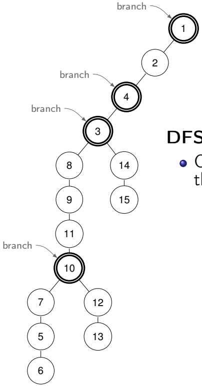


Depth First Search (DFS)

- *Mark* a neighbor of the current node as we traverse and don't traverse previously marked nodes

set source $s = \{1\}$
and pick the vertex with *smallest* values
if more than one nodes can be chosen





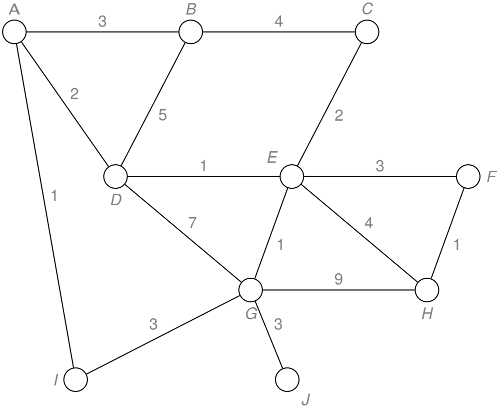
DFS tree

- Capture the structure of the recursive calls
  - When we visit an adjacent vertex of v, we add it as a child of v
  - Whenever DFS returns from a vertex v, we climb up in the tree from v to its parent

Depth First Search (DFS) (cont.)

Exercise

Report on the order of the vertices encountered on a DFS starting from vertex A. Break all ties by picking the edge with smallest weight.



Depth First Search (DFS) (cont.)

Pseudocode

1. Start by putting the source node on the top of a stack
2. Take the top node of the stack and add it to the visited list
3. Create a list of that vertex's adjacent nodes and add the ones which are not visited to the top of the stack
4. Keep repeating steps 2 and 3 until the stack is empty

Depth First Search (DFS) (cont.)

Time complexity

Using adjacency list

DFS(G)

```
for each vertex u ∈ G.V
        flag[v] = false
RDFS(v)
```

RDFS(v)

Flag the vertex v as visited

```
flag[v] = true
for each w adjacent to v
        if flag[w] = false
                RDFS(w)
```

Call RDFS recursively for each of v's adjacent vertices

- Each vertex will only visit at most once
- We had to examine all edges of the vertices
  - i.e., $\Sigma_{v \in V} \texttt{deg(}v\texttt{)} = 2m$, where m is the number of edges
- Therefore, the running time of DFS is proportional to the number of edges and the number of vertices (same as breadth first search (BFS))
  - $O(n + m)$ (or $O(|V| + |E|)$), where m is the number of vertices

Differences between BFS and DFS

	BFS	DFS
Definition	Traversal begins at the *root* node and walk through all nodes on the same level before moving on to the next level	Traversal begins at the *root* node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes
Conceptual Difference	Builds the tree *level by level*	Builds the tree *subtree by subtree*
Data structure	Queue (FIFO)	Stack (LIFO)
Suitable for	Searching vertices *closer* to the given source	Finding paths (or solutions) that are *away* from source
Applications	Finding Shortest path, bipartite graphs, GPS navigation, etc.	Cycles or loops detection, finding strongly connected components (SCC), etc.
Path generation	Traversals according to the tree level	Traversals according the tree depth
Backtracking	Not required	Required to follow a backtrack
Memory	More memory	Less memory
Loops	*Cannot* be trapped into *finite* loops	*Can* be trapped into *infinite* loops

Differences between BFS and DFS

	Adjacency list		Adjacency matrix	
	Time complexity	Auxiliary space	Time complexity	Auxiliary space
BFS	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$
DFS	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$

Reading

- Chapter 20, Cormen (2022)

References I

- algo.monster (2024). *Depth First Search*. Online: `https://algo.monster/problems/dfs_intro`. [last accessed: 20 Mar 2024].
- Geeksforgeeks.org (2024a). *Depth First Search or DFS for a Graph*. Online: `https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph`. [last accessed: 20 Mar 2024].
- — (2024b). *Difference between BFS and DFS*. Online: `https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/`. [last accessed: 20 Mar 2024].
- Programiz (2024). *Depth First Search (DFS)*. Online: `https://www.programiz.com/dsa/graph-dfs`. [last accessed: 20 Mar 2024].