CPT108 Data Structures and Algorithms

Lecture 22 and 23

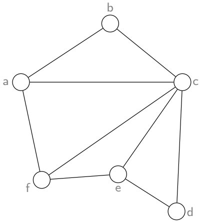
Graph Traversal: BFS and DFS

Graphs

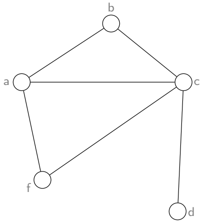
Some definitions (cont.)

Subgraph

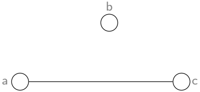
- A graph $H(V_H, E_H)$ is a subgraph of $G(V_G, E_G)$ if and only iff $V_H \subset V_G$ and $E_H \subset V_G$



graph G

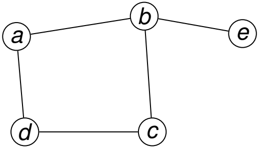


graph H_1

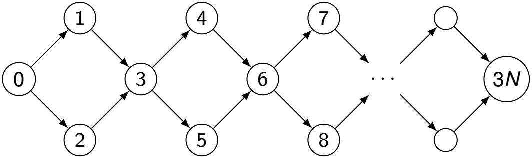


graph H_2

Traversing a Graph



- Many algorithms on graphs depend on traversing all or some nodes
  - i.e., given a graph representation and a vertex s in the graph, find *ALL* paths from s to other vertices
- Can we use recursion when traversing a graph?
  - Possible, but with caution due to *cycle*
- Even in acyclic graphs, can get combinatorial explosions:



Treat "0" as the root and do recursive traversal down the two edges out of each node: $O(2^N)$ operations!

- So, typically try to visit each node constant number of times (e.g., once)

Traversing a Graph (cont.)

- Two common graph traversal algorithms
  - Depth first search (DFS)
  - Breadth first search (BFS)

Breadth First Search (BFS)

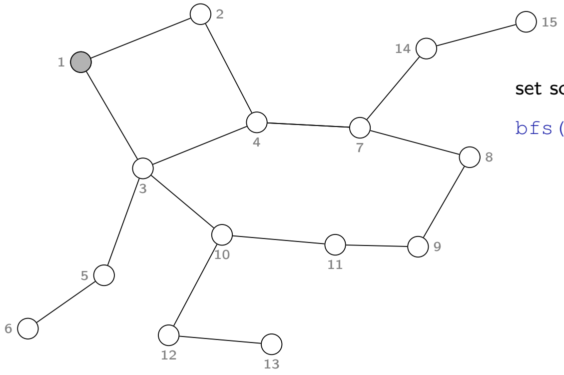
Applications

- Find shortest path between nodes in *unweighted graph*
- Cycle detection in *undirected graph*
- GPS navigation for neighboring locations
- Find person in social networks
- Devices connected to a particular network
- Crawlers in Search Engines

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



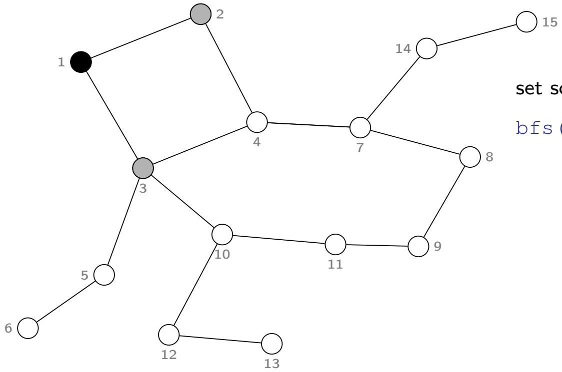
set source $s = \{1\}$

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

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node *s*
  - What do we mean by "distances"?
    - the number of edges on a path from *s*



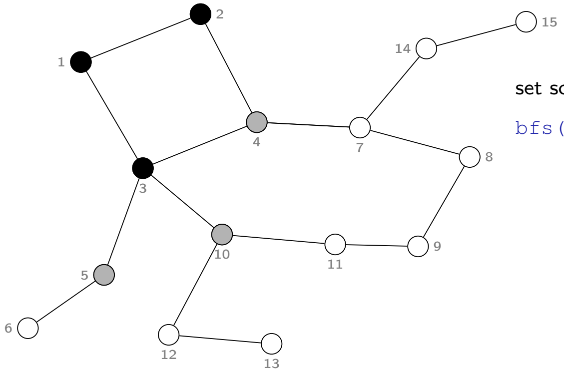
set source $s = \{1\}$

`bfs(s)` $= \{ 1,$
$2, 3 \; // d = 1$
$\}$

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by “distances”?
    - the number of edges on a path from s



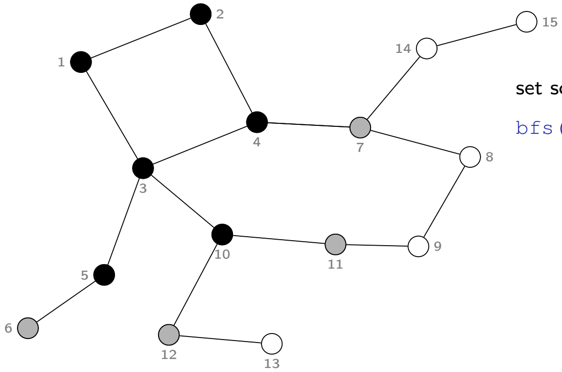
set source $s = \{1\}$

`bfs(s)` $= \{\ 1,$
$2, 3, \quad //d = 1$
$4, 5, 10 \ //d = 2$
$\}$

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



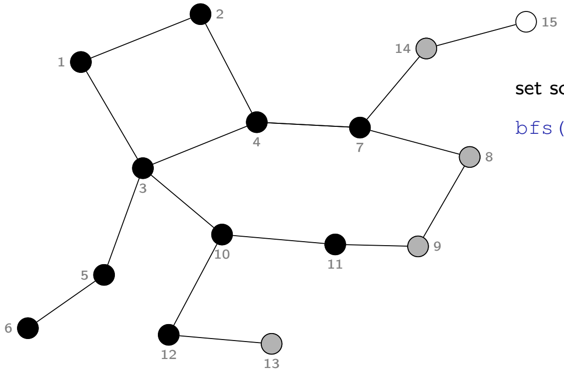
set source $s = \{1\}$

`bfs(s)` $= \{ 1,$
$2, 3,$ $// d = 1$
$4, 5, 10,$ $// d = 2$
$6, 7, 11, 12$ $// d = 3$
$\}$

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by “distances”?
    - the number of edges on a path from s



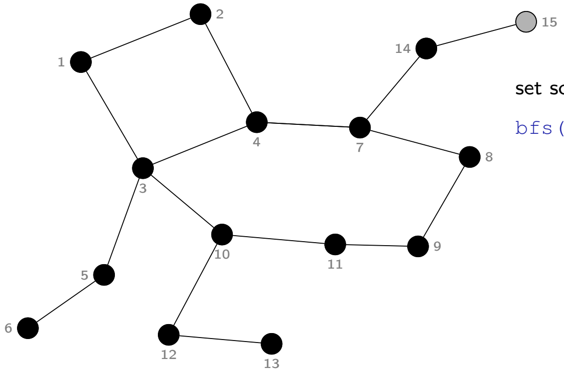
set source $s = \{1\}$

$$
\texttt{bfs(s)} = \{\ 1,\\
2, 3, \quad //d = 1\\
4, 5, 10, \quad //d = 2\\
6, 7, 11, 12, \quad //d = 3\\
8, 9, 13, 14 \quad //d = 4\\
\}
$$

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



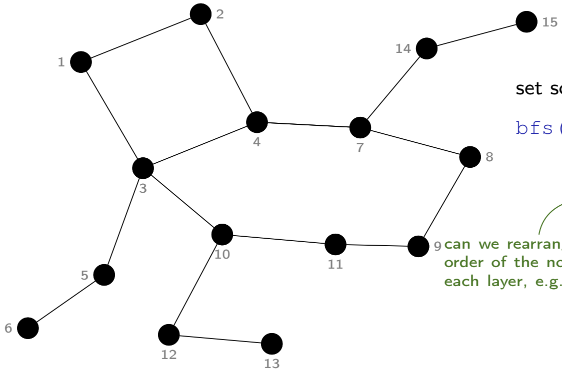
set source $s = \{1\}$

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

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



set source $s = \{1\}$

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

can we rearrange the order of the nodes at each layer, e.g., here?

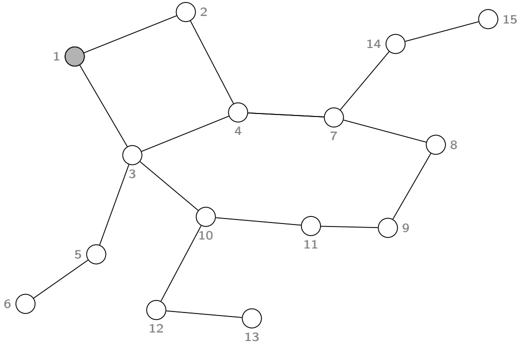
$\Rightarrow$ BFS completed!

Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s

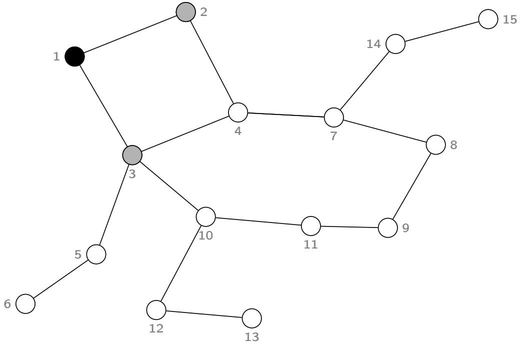
set source $s = \{1\}$



Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



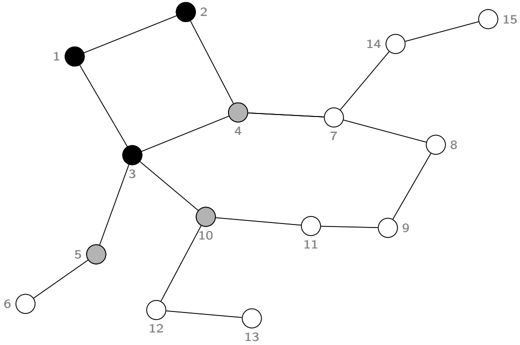
set source $s = \{1\}$

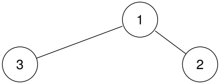
Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s

set source $s = \{1\}$

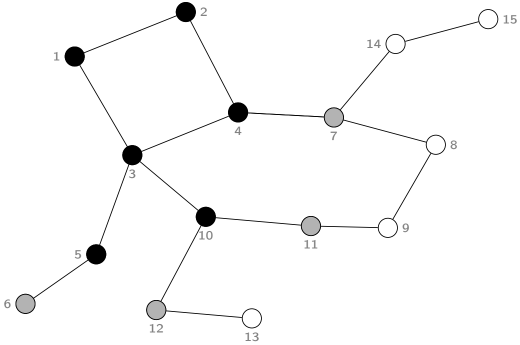




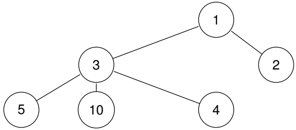
Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s



set source $s = \{1\}$

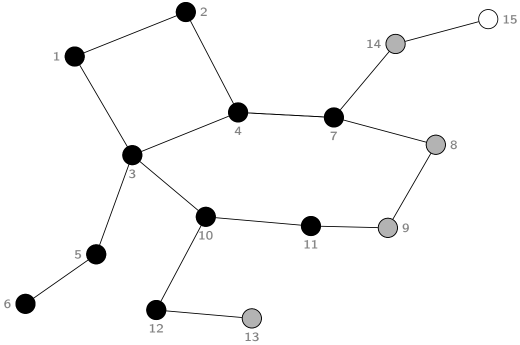


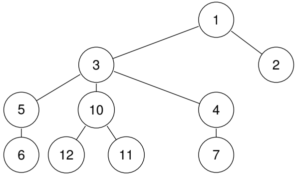
Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by "distances"?
    - the number of edges on a path from s

set source $s = \{1\}$



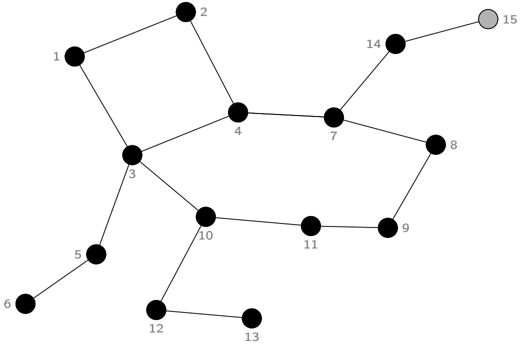


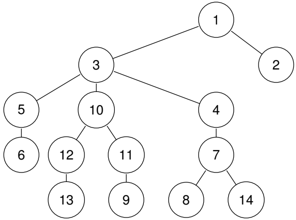
Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node *s*
  - What do we mean by “distances”?
    - the number of edges on a path from *s*

set source $s = \{1\}$

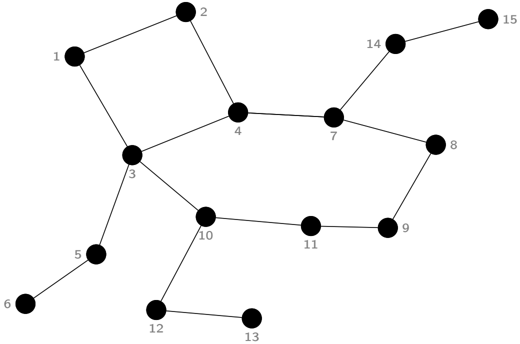




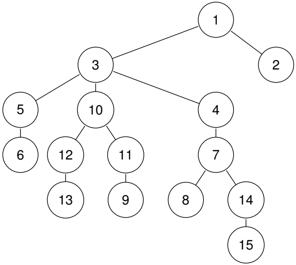
Breadth First Search (BFS)

Main concept

- Visit other nodes at increasing distances from a source node s
  - What do we mean by “distances”?
    - the number of edges on a path from s



set source $s = \{1\}$



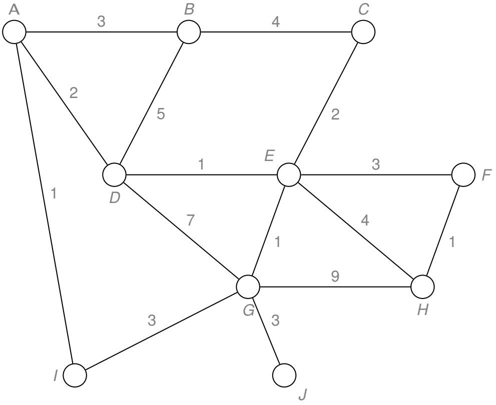
BFS tree / *BFS forest*

- What would a “Level” order traversal tell you?

Breadth First Search (BFS) (cont.)

Exercise

Report on the order of the vertices encountered on a BFS starting from vertex *A*. Break all ties by picking the vertices in alphabetic order (i.e., *A* before *Z*).



Breadth First Search (BFS) (cont.)

Pseudocode

1. **Initialization**: Enqueue the starting node into a queue and mark it as visited
2. **Exploration**: While the queue is not empty
    - Dequeue a node from the queue and visit it (e.g., print its value)
    - For each unvisited neighbor of the dequeued node:
        - Enqueue the neighbor into the queue
        - Mark the neighbor as visited
3. **Termination**: Repeat Step 2 until the queue is empty

Recap: Adjacency list vs Adjacency matrix

Given a graph $G = (V, E)$:
$n =$ number of vertex, and
$m =$ number of edges

- **Adjacency list**
  - More compact than adjacency matrix if graph has few edges
  - Requires a scan of adjacency list to check if an edge exists
  - Requires a scan to obtain all edges!
- **Adjacency matrix**
  - Always require n^2 space
    - This can waste a lot of space if the number of edges are sparse
  - Find if an edge exist if $O(1)$
  - Obtain all edges in $O(n^2)$

Breadth First Search (BFS) (cont.)

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 ∈ G.V
    flag[u] = false
Q = empty queue
flag[s] = true
ENQUEUE(Q, s)
while Q is not empty
    v = DEQUEUE(Q)
    for each w adjacent to v
        if flag[w] = false
            flag[w] = true
            ENQUEUE(Q, w)
```

Each vertex will enter Q at most once

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).

Breadth First Search (BFS) (cont.)

Time complexity

Using adjacency matrix

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

BFS(G, s)

```
for each vertex u ∈ G.V
    flag[u] = false
Q = empty queue
flag[s] = true
ENQUEUE(Q, s)
while Q is not empty
    v = DEQUEUE(Q)
    for each w adjacent to v
        if flag[w] = false
            flag[w] = true
            ENQUEUE(Q, w)
```

Each vertex will enter Q at most once

Finding the adjacent vertices of v requires checking all elements in the row, which takes $O(n)$.

Total running time of the while loop

$= \Sigma_{v \in V}(n)$

$= O(n \times n)$

$= O(n^2)$ (or $O(|V|^2)$)

which is independent to the number of edges m

Depth First Search (DFS)

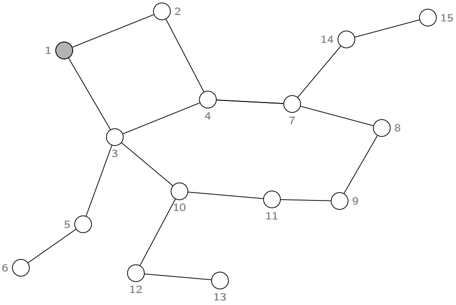
Applications

- 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



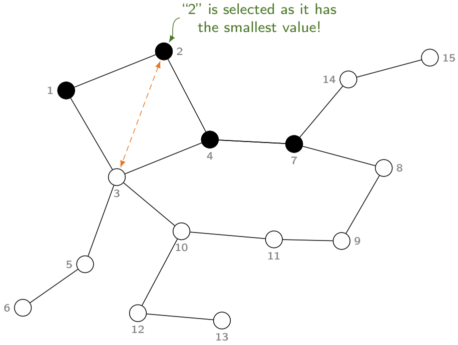
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

`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



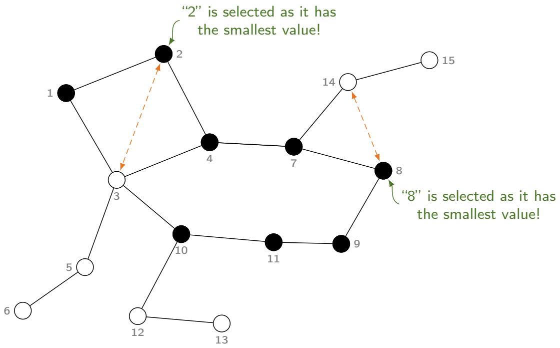
set source $s = \{1\}$
and pick the vertex with
smallest values if more than
one nodes can be chosen

`dfs(s)` $= \{ 1,$
$2, 4, 7$
$\}$

Depth First Search (DFS)

Main concept

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



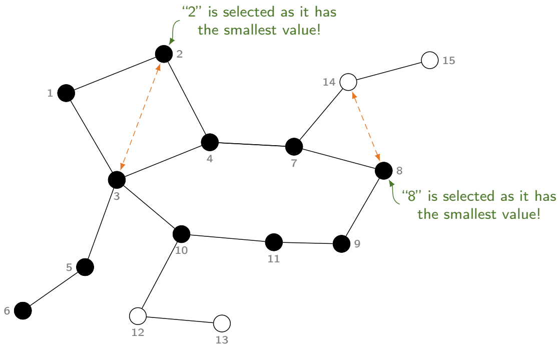
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

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

Depth First Search (DFS)

Main concept

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



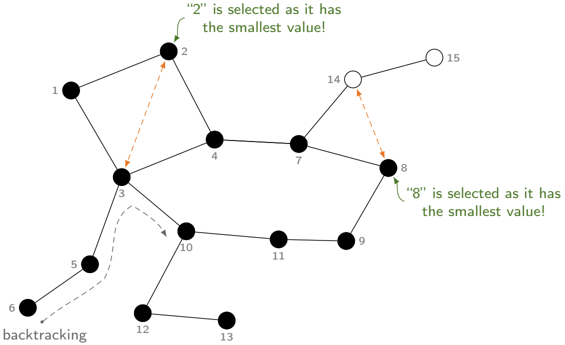
set source $s = \{1\}$
and pick the vertex with
smallest values if more than
one nodes can be chosen

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



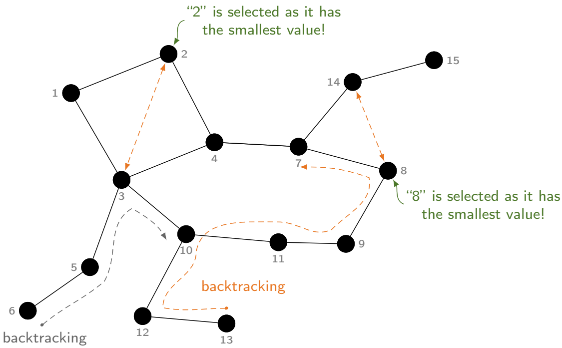
set source $s = \{1\}$
and pick the vertex with
smallest values if more than
one nodes can be chosen

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



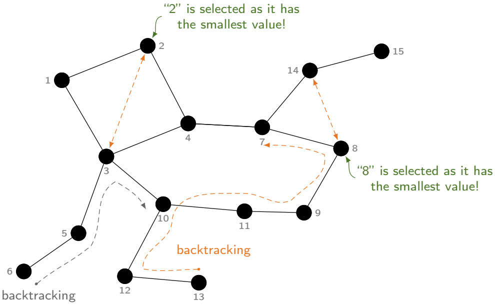
set source $s = \{1\}$
and pick the vertex with *smallest* values if more than one nodes can be chosen

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

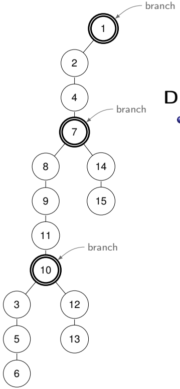
$\Rightarrow$ DFS completed!

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



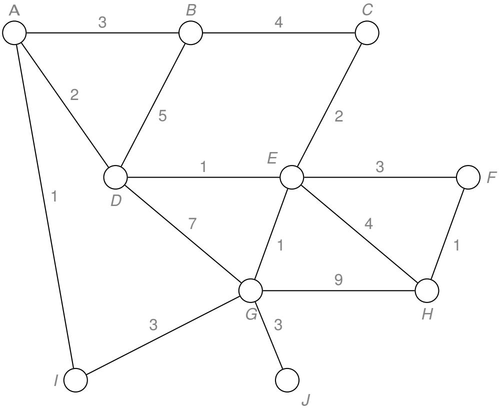
DFS tree

- 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 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). *Breadth First Search or BFS for a Graph*. Online: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph. [last accessed: 20 Mar 2024].
- — (2024b). *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].
- — (2024c). *Difference between BFS and DFS*. Online: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/. [last accessed: 20 Mar 2024].
- Programiz (2024a). *Breadth first search*. Online: https://www.programiz.com/dsa/graph-bfs. [last accessed: 20 Mar 2024].

References II

Programiz (2024b). *Depth First Search (DFS)*. Online: https://www.programiz.com/dsa/graph-dfs. [last accessed: 20 Mar 2024].