

Course Algorithms and Data Structures · Lecture 14

Graphs: Depth-First Search and Applications

Depth-first search (DFS)

How you get out of a maze: keep walking forward; at a dead end go back to the last crossing with an untried way.

Go deep

from a vertex, go to an unvisited neighbor, then to its unvisited neighbor…

Backtrack

at a dead end, return to the vertex you came from most recently: a stack

DFS, recursively

```
ALGORITHM DFS(v)
    mark v as visited
    for each vertex w adjacent to v do
        if w is not visited
            DFS(w)
```

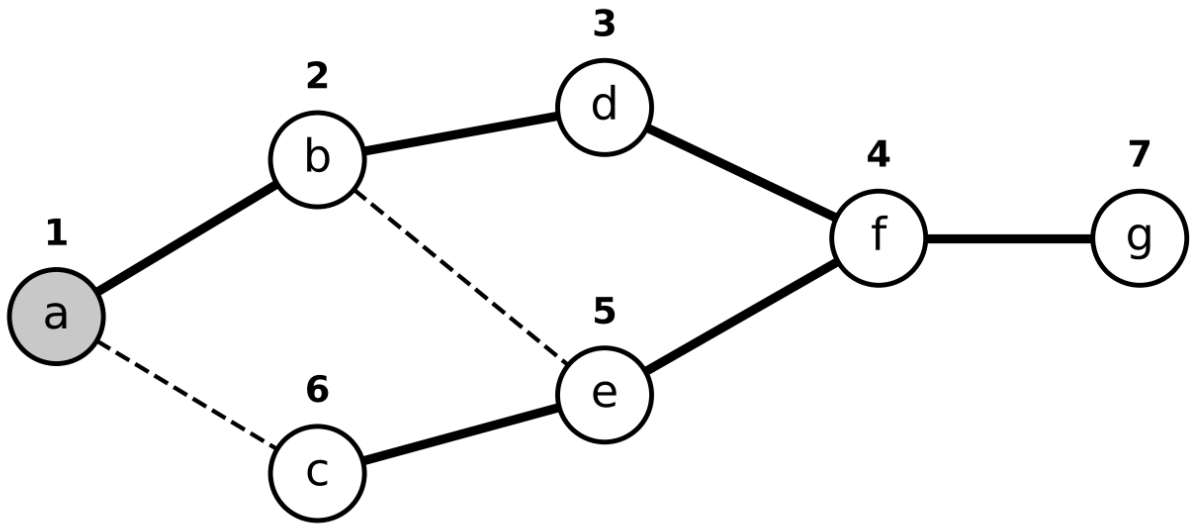
The call stack does the backtracking for us: when DFS(w) returns, we are back at v.

Tracing DFS from a

Step	Path (call stack)	What happens
1	a	visit a, go to b
2	a, b	visit b, go to d
3	a, b, d	visit d, go to f
4	a, b, d, f	visit f, go to e
5	a, b, d, f, e	visit e; b visited; go to c
6	a, b, d, f, e, c	visit c; dead end, back
7	a, b, d, f	back at f, go to g
8	a, b, d, f, g	visit g; dead end, back to a

Order: a, b, d, f, e, c, g. BFS gave a, b, c, d, e, f, g.

The DFS tree



- Thick: tree edges
- Dashed: **back edges** to an ancestor
- Numbers: visiting order
- Long and narrow, BFS: short and wide

DFS with an explicit stack

```
ALGORITHM DFSStack(s)
    create a stack containing s
    while the stack is not empty do
        v ← pop()
        if v is not visited
            mark v as visited
            for each w adjacent to v, in reverse order, do
                if w is not visited
                    push(w)
```

Same code as BFS with a stack instead of a queue. Safe for very deep graphs.

How fast is DFS?

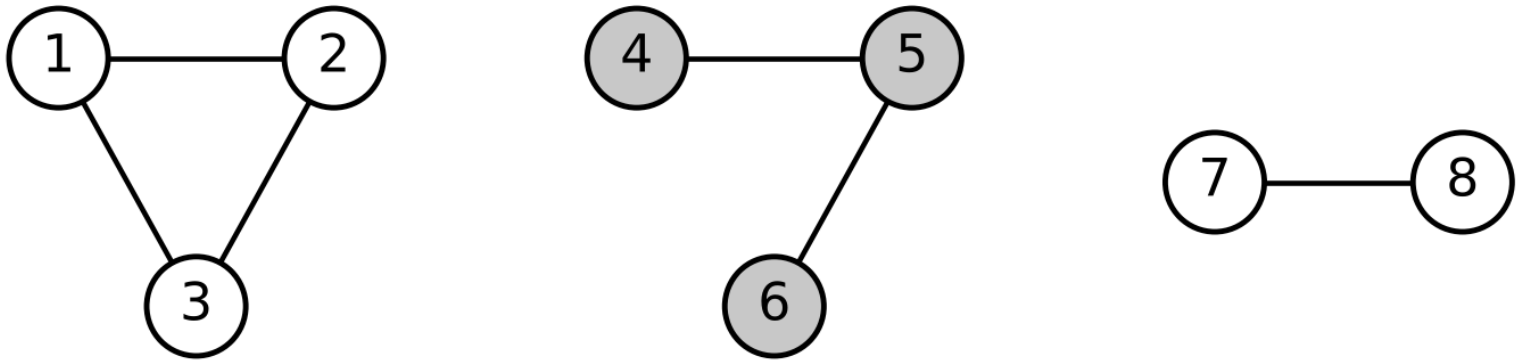
O(V + E) with adjacency lists

O(V^2) with an adjacency matrix

Each vertex is visited once, each adjacency list is read once. Exactly like BFS.

Applications: paths and components

- **Is there a path from u to v?** Run DFS(u): yes if v gets marked
- A **connected component** is a largest group of vertices reachable from each other



Counting connected components

```
ALGORITHM Components(G)
    mark every vertex as not visited
    count ← 0
    for each vertex v in V do
        if v is not visited
            count ← count + 1
            DFS(v)        //marks the whole component
    return count
```

In the picture: DFS starts from 1, 4 and 7, so there are 3 components. Still O(V + E).

Cycles and other uses

An undirected graph has a cycle **if and only if** DFS finds a back edge. Back edge e–b closes the cycle b, d, f, e, b. (The edge to the parent doesn’t count.)

- **Mazes and puzzles**: backtracking
- **Directed graphs**: topological sorting of tasks (later courses)

BFS or DFS?

	BFS	DFS
Data structure	queue	stack (or recursion)
Order	layer by layer	deep first, then back
Tree shape	short and wide	long and narrow
Non-tree edges	cross edges	back edges
Shortest paths by edges	yes	no
Natural for	shortest routes, levels	cycles, components, mazes
Time (lists)	O(V + E)	O(V + E)

Common mistakes

- **Not marking vertices**: DFS runs around a cycle forever
- Counting **the edge to the parent** as a back edge
- Expecting DFS to find **shortest** paths: use BFS
- **Very deep recursion** on huge graphs: use the explicit stack

Summary

- DFS goes as deep as possible and backtracks at dead ends, using a stack or recursion.
- DFS splits edges into tree edges and back edges and runs in O(V + E).
- DFS answers: is there a path, how many components, is there a cycle.
- BFS (queue) for shortest paths; DFS (stack) for exploring, cycles and backtracking.

Review questions

1. What is the main difference between BFS and DFS? Which data structure does each one use?
2. Run DFS on the example graph from g, taking neighbors in alphabetical order. In what order are the vertices visited?
3. How can DFS count the connected components of a graph? How many components does the graph in the lecture have?

Reading: Levitin, Section 3.5; Cormen et al., Section 20.3.