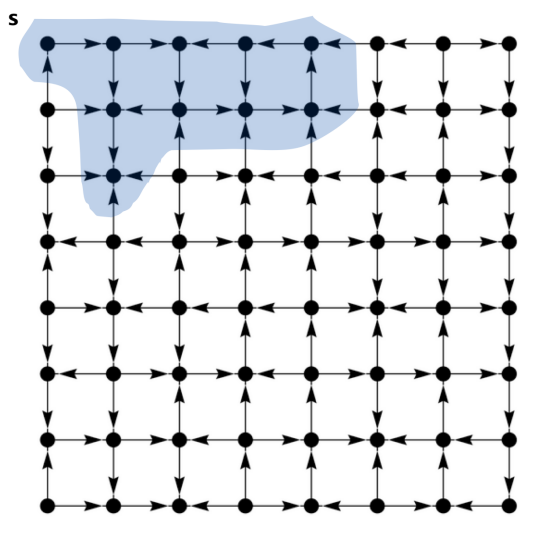
https://algs4.cs.princeton.edu

4. Graphs and Digraphs I

- introduction
- graph representation
- depth-first search
- path finding
- undirected graphs

Reachability problem in a digraph

Reachability problem. Given a digraph G and vertex s, find all vertices reachable from s.



Reachability problem in a digraph

Reachability problem. Given a digraph G and vertex s, find all vertices reachable from s.

Depth-first search. A systematic method to explore all vertices reachable from s.

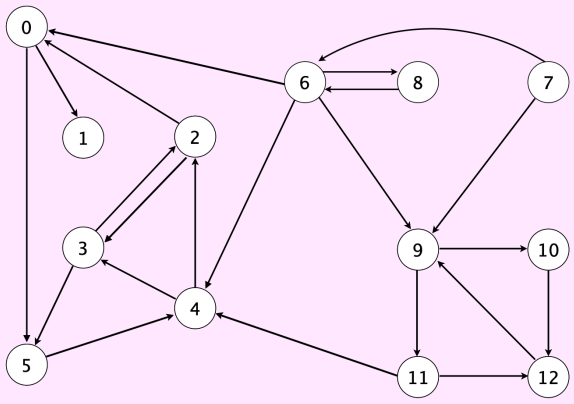
DFS (to visit a vertex v)

Mark vertex v.
Recursively visit all unmarked vertices w adjacent from v.

Depth-first search (in a digraph) demo

To visit a vertex v :

- Mark vertex v.
- Recursively visit all unmarked vertices adjacent from v.



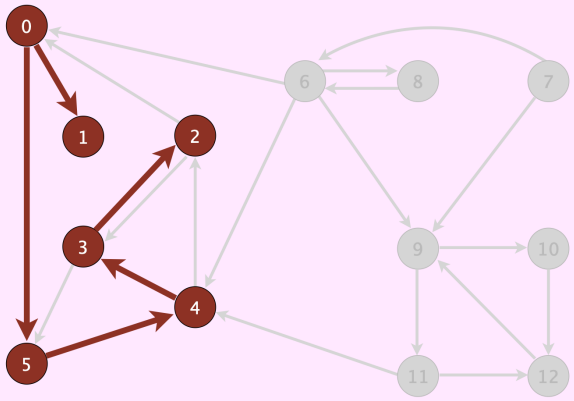
a directed graph

4→2
2→3
3→2
6→0
0→1
2→0
11→12
12→9
9→10
9→11
8→9
10→12
11→4
4→3
3→5
6→8
8→6
5→4
0→5
6→4
6→9
7→6

Depth-first search (in a digraph) demo

To visit a vertex v :

- Mark vertex v.
- Recursively visit all unmarked vertices adjacent from v.



reachable from 0

v	marked[]
0	T
1	T
2	T
3	T
4	T
5	T
6	F
7	F
8	F
9	F
10	F
11	F
12	F

reachable from vertex 0

Graphs and digraphs I: poll 3

Run DFS using the given adjacency-lists representation of digraph G, starting at vertex 0. In which order is `dfs(digraph, v)` called?

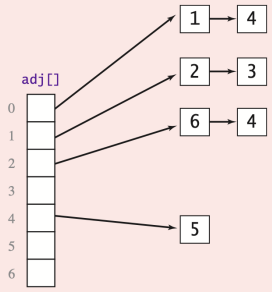
DFS preorder

A. 0 1 2 4 5 3 6

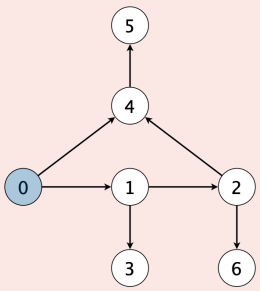
B. 0 1 2 4 5 6 3

C. 0 1 3 2 6 4 5

D. 0 1 2 6 4 5 3



adjacency-lists representation



digraph G

Depth-first search: analysis

Proposition. DFS uses $\Theta(V)$ extra space (not including the digraph itself).

Pf.

- The `marked[]` array uses $\Theta(V)$ space.
- The function-call stack uses $O(V)$ space. ← *`dfs()` called at most once per vertex*

Proposition. DFS marks all vertices reachable from s in $\Theta(E + V)$ time in the worst case.

Pf.

- Initializing the `marked[]` array takes $\Theta(V)$ time.
- Each vertex is visited at most once.
- Visiting a vertex takes time proportional to its outdegree:

$$outdegree(v_0) + outdegree(v_1) + outdegree(v_2) + \ldots = E$$

↑ *in the worst case, all vertices are reachable from s*

Note. If all vertices are reachable from s, then $E \geq V - 1$ and running time simplifies to $\Theta(E)$.

Graphs and digraphs I: poll 4

What could happen if we marked a vertex at the end of the DFS call (instead of beginning)?

A. Marks a vertex not reachable from s.

B. Compile-time error.

C. Infinite loop / stack overflow.

D. None of the above.

```
private void dfs(Digraph digraph, int v) {
 marked[v] = true;
  for (int w : digraph.adj(v))
     if (!marked[w])
        dfs(digraph, w);
 marked[v] = true;
}
```

Reachability application: program control-flow analysis

Every program is a digraph.

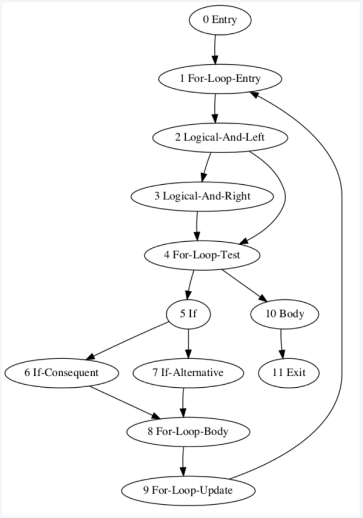
- Vertex = basic block of instructions (straight-line program).
- Edge = jump.

Dead-code elimination.

Find (and remove) unreachable code.

Infinite-loop detection.

Determine whether exit is unreachable.



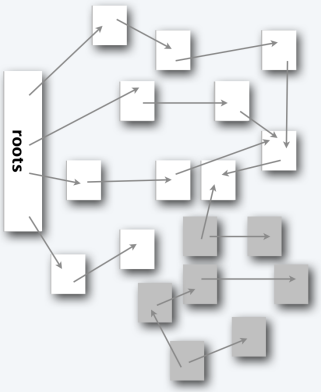
Reachability application: mark-sweep garbage collector

Every data structure is a digraph.

- Vertex = object.
- Edge = reference/pointer.

Roots. Objects known to be directly accessible by program (e.g., stack frame).

Reachable objects. Objects indirectly accessible by program (starting at a root and following a chain of pointers).

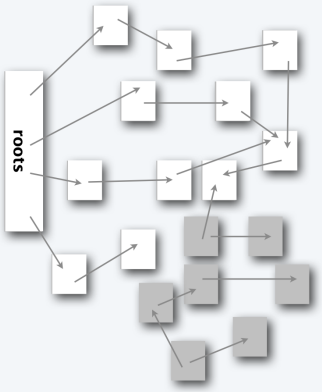


Reachability application: mark-sweep garbage collector

Mark-sweep algorithm. [McCarthy, 1960]

- Mark: mark all reachable objects.
- Sweep: if object is unmarked, it is garbage (so add to free list).

Memory cost. Uses one extra mark bit per object (plus DFS function-call stack).



https://algs4.cs.princeton.edu

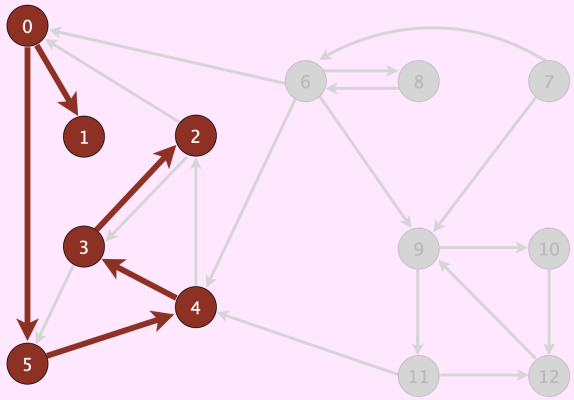
4. Graphs and Digraphs I

- *introduction*
- *graph representation*
- *depth-first search*
- *path finding*
- *undirected graphs*

Directed paths DFS demo

Goal. DFS determines which vertices are reachable from s. How to reconstruct paths?

Solution. Use parent-link representation.



vertices reachable from 0 (and directed paths)

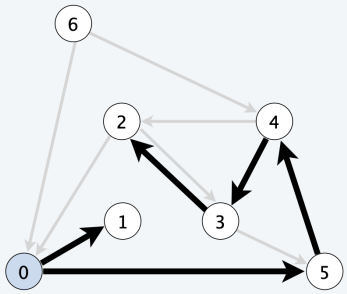
v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

parent-link representation of paths from vertex 0

Depth-first search: path finding

Parent-link representation of paths from vertex s.

- Maintain an integer array `edgeTo[]`.
- Interpretation: `edgeTo[v]` is the next-to-last vertex on a directed path from s to v.
- To reconstruct path from s to v, trace `edgeTo[]` backward from v to s (and reverse).



v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-

```
public Iterable<Integer> pathTo(int v) {
   if (!marked[v]) return null;
   Stack<Integer> path = new Stack<>();
   for (int x = v; x != s; x = edgeTo[x])
      path.push(x);
   path.push(s);
   return path;
}
```

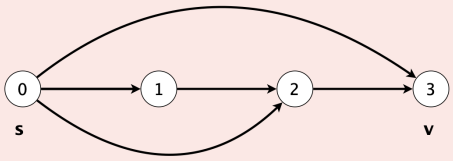
Graphs and digraphs I: poll 5

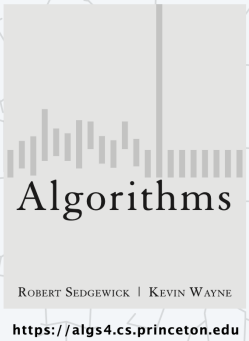
Suppose there are many paths from s to v. Which one does `DepthFirstDirectedPaths` find?

A. A shortest path (fewest edges).

B. A longest path (most edges).

C. Depends on digraph representation.





4. Graphs and Digraphs I

- *introduction*
- *graph representation*
- *depth-first search*
- *path finding*
- *undirected graphs*

Flood fill

Problem. Implement flood fill (Photoshop magic wand).

Depth-first search in undirected graphs

Connectivity problem. Given an undirected graph G and vertex s, find all vertices connected to s.

Solution. Use DFS. ← *but now, for each undirected edge v–w: v is adjacent with w (and w is adjacent with v)*

DFS (to visit a vertex v)

Mark vertex v.

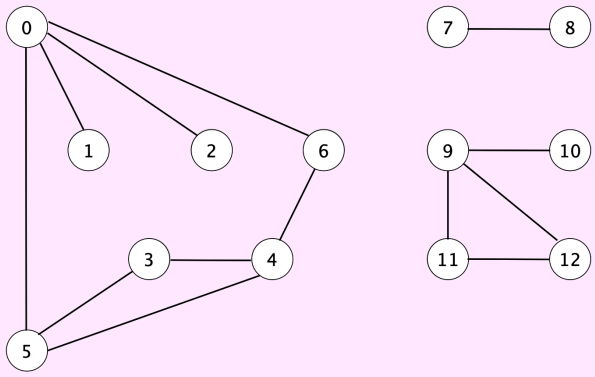
Recursively visit all unmarked vertices w adjacent with v.

Proposition. DFS marks all vertices connected to s in $\Theta(E + V)$ time in the worst case.

Depth-first search (in an undirected graph) demo

To visit a vertex v :

- Mark vertex v.
- Recursively visit all unmarked vertices adjacent with v.



graph G

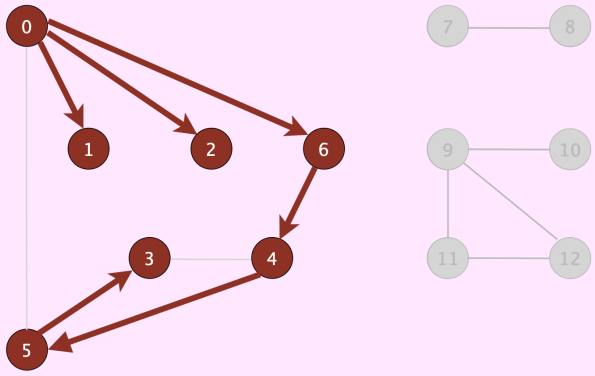
tinyG.txt

V → 13
E ← 13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3

Depth-first search (in an undirected graph) demo

To visit a vertex v :

- Mark vertex v.
- Recursively visit all unmarked vertices adjacent with v.



vertices connected to 0
(and associated paths)

v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Graphs and digraphs I: poll 6

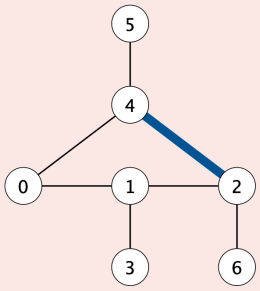
How to represent an undirected edge v–w using adjacency lists?

A. Add w to adjacency list for v.

B. Add v to adjacency list for w.

C. Both A and B.

D. None of the above.



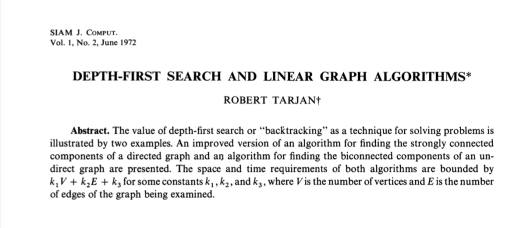
Depth-first search summary

DFS enables direct solution of several elementary graph and digraph problems.

- Reachability (in a digraph). ✓
- Connectivity (in a graph). ✓
- Path finding (in a graph or digraph). ✓
- Topological sort. ← *next lecture*
- Directed cycle detection. ← *precept*

DFS is also core of solution to more advanced problems.

- Euler cycle.
- Biconnectivity.
- 2-satisfiability.
- Planarity testing.
- Strong components.
- Nonbipartite matching.
- ...



SIAM J. COMPUT.
Vol. 1, No. 2, June 1972

DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by $k_1V + k_2E + k_3$ for some constants k_1, k_2, and k_3, where V is the number of vertices and E is the number of edges of the graph being examined.

Credits

media	source	license
Function Graph	Adobe Stock	Education License
Pac–Man Graph	Oatzy	
Pac–Man Game	Old Classic Retro Gaming	
London Tube Map	Transport for London	
London Tube Graph	visualize.org	
LinkedIn Social Network	Caleb Jones	
Twitter Graph	Caleb Jones	
Protein Interaction Graph	Hawing Jeong / KAIST	
PageRank	Wikipedia	public domain
Control Flow Graph	Stack Exchange	
DFS Graph Visualization	Gerry Jenkins	