CSCI 470: Strongly Connected Components, Breadth-First Search

Vijay Chaudhary

October 23, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. Quick updates
2. Strongly connected components
3. Breadth-First Search
4. Exam II

Quick updates

Updates

- Exam II will be on Monday, Oct 30.
- HW 04 will be out tomorrow, however, the deadline will be after Exam II.
- Practice Exam II will be posted on Canvas by tomorrow.

Strongly connected components

Strongly connected components

- An undirected graph is connected if every vertex is reachable from all other vertices. That is, for every pair of vertices $u, v \in V$, there is a path from u to v (and therefore, a path from v to u).

Strongly connected components

- An undirected graph is connected if every vertex is reachable from all other vertices. That is, for every pair of vertices $u, v \in V$, there is a path from u to v (and therefore, a path from v to u).
- A directed graph G is strongly connected if every two vertices are reachable from each other. That is, for every pair of vertices $u, v \in V$, there is both a path from u to v and a path from v to u.

Strongly connected components

- A strongly connected component of a directed graph $G = (V, E)$ is a maximal part of the graph that is strongly connected. Formally, it is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices $u, v \in C$, there is both a path from u to v and a path from v to u.
- A connected component of an undirected graph is a maximal set of vertices where every vertex in the set is reachable from every vertex in the set. That is, for every pair of vertices u, v in the connected component C, there is a path from u to v.

SCC: Figure

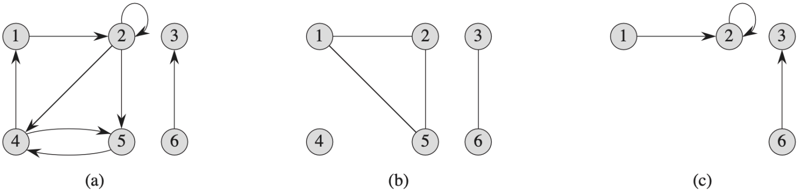


Figure B.2 Directed and undirected graphs. **(a)** A directed graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. The edge $(2, 2)$ is a self-loop. **(b)** An undirected graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (1, 5), (2, 5), (3, 6)\}$. The vertex 4 is isolated. **(c)** The subgraph of the graph in part (a) induced by the vertex set $\{1, 2, 3, 6\}$.

SCC

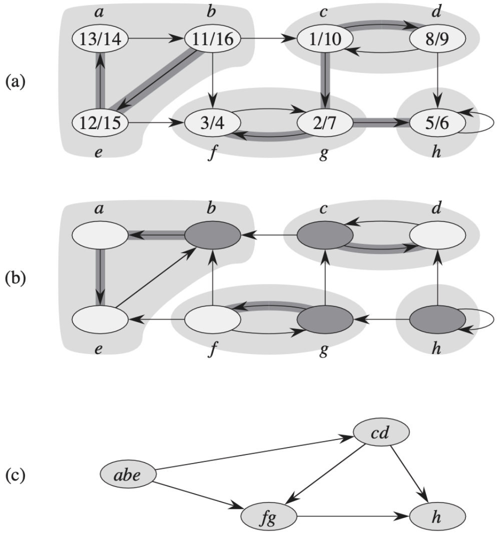
- Our algorithm for finding strongly connected components of a graph $G = (V, E)$ uses the transpose of G.
- We define $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$. That is, E^T consists of the edges of G with their directions reversed.
- It is interesting to observe that G and G^T have exactly the same strongly connected components: u and v are reachable from each other in G if and only if they are reachable from each other in G^T.

SCC: pesudocode

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^T), but in the main loop DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as separate strongly connected component

SCC: Figure

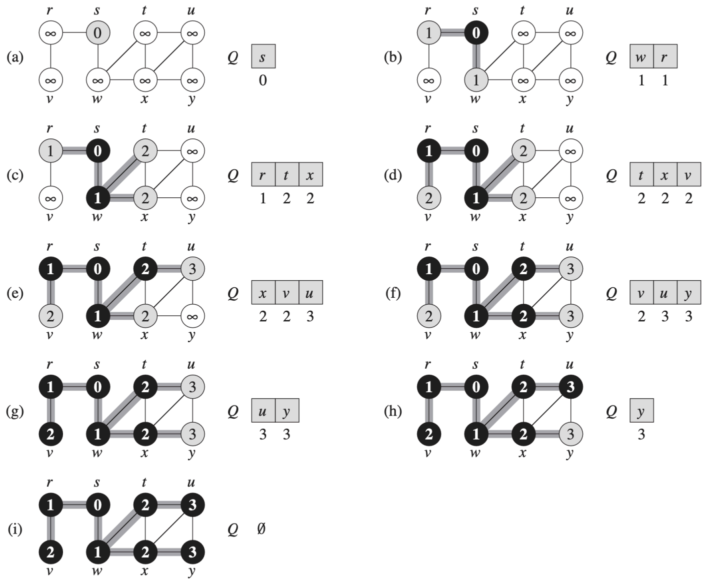

(a)
a
b
c
d
13/14
11/16
1/10
8/9
12/15
3/4
2/7
5/6
e
f
g
h
(b)
a
b
c
d
e
f
g
h
(c)
cd
abe
fg
h

Breadth-First Search

BFS

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

BFS: Illustration



BFS: Runtime

- The operations of enqueuing and dequeuing take $O(1)$ time, and so the total time devoted to queue operations in $O(V)$
- Because the procedure scans the adjacency list of each vertex only when the vertex is dequeued, it scans each adjacency list at most once.
- Since the sum of the lengths of all the adjacency lists is $\Theta(E)$, the total time spent in scanning adjacency lists is $O(E)$.
- The overhead for initialization is $O(V)$, and thus the total running time of the BFS procedure is $O(V + E)$.

Shortest paths

- The following procedure prints out the vertices on a shortest path from s to v, assuming that BFS has already computed a breadth-first tree:

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Exam II

Topics for Exam II

- Chapter 08: Sorting in Linear Time (Lower bounds for sorting, counting sort)
- Chapter 10: Elementary Data Structures (Stacks and queues, linked lists, binary trees)
- Chapter 11: Hash Tables (Direct-address tables, hash tables, hashing with chaining, hash functions)
- Chapter 12: Binary Search Trees (Binary search tree, querying a binary search tree, insertion & deletion)
- Chapter 22: Elementary graph algorithms (Representations of graphs, DFS, topological sort, strongly connected components, BFS, shortest-path)