Graphs part 1

Motivation:

Graphs – fundamental to many problems. Web graphs. Biology. Other.

Connectivity – is network connected and can you get from one node to another, or what is the shortest path? Examples: Driving directions; get to one contact through another; social media or contact path.

Some notation: We'll usually describe the vertices (V) and edges (E) of graph G=(V, E). We sometimes talk about vertices v in V, and edges e in E. We denote the number of vertices and edges as n and m respectively. (The book sometimes also uses V and E to denote number of vertices and edges).

Two forms of representation:

(1) Adjacency list: each vertex has a linked list to each adjacent vertex (can save in space O(n + m); slower lookup; good in the case of a relatively sparse graph)

(2) Adjacency Matrix (takes up more space O(n squared), faster lookup; good in the case of a dense graph)

Note: number of edges can be between O(1) and O(n squared), depending on whether graph is sparse or dense.

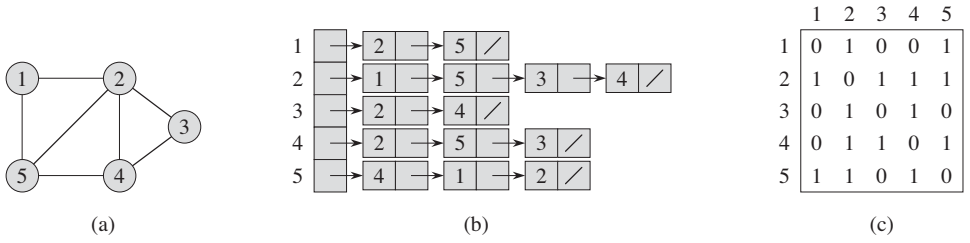
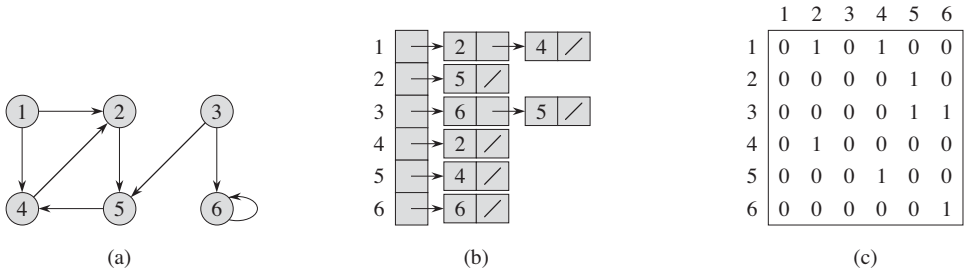


Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G with 5 vertices and 7 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

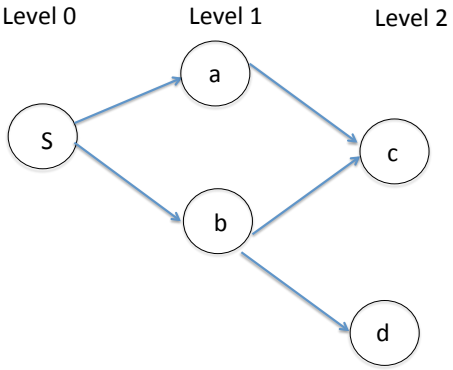


How do we find the vertices reachable from a given vertex?

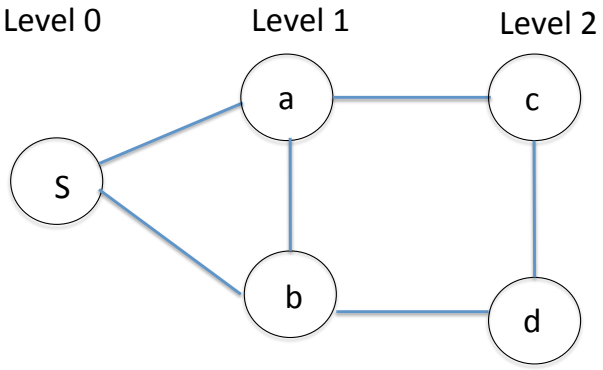
Two ways to search graphs: BFS; DFS

Breadth First Search (BFS):

Find all vertices distance 1; then distance 2; then distance 3; etc… by breadth. Like visiting according to levels (of distance 1; then 2; then 3; etc).



Also in undirected graph:



Application: finding shortest paths between vertices (eg, from source vertex s to all other vertices).

Run time: we will see that fast; linear in number of vertices and nodes: O(n + m)

We color the nodes to keep track of them (in BFS, DFS):

White: Vertices are initially white;

Gray: First time vertex encountered, make gray

Black: Only when all its adjacent neighbors discovered, turn to black.

Main approach: Uses **first in first out** queue Q to manage the discovered gray vertices, until we go through all their adjacency neighbors, and they then turn black. Same for directed and undirected graphs. We'll write this out higher level than book:

BFS(G, s)

// G is a graph and s is some source vertex

- initialize all vertices to white, and distance infinity

- initially make s vertex gray, distance 0, and Enqueue(Q, s)

- while Q not empty

    u = Dequeue(Q)

    for every vertex v in adjacency list of u

        if v white
        // first discovered; put in the back of queue

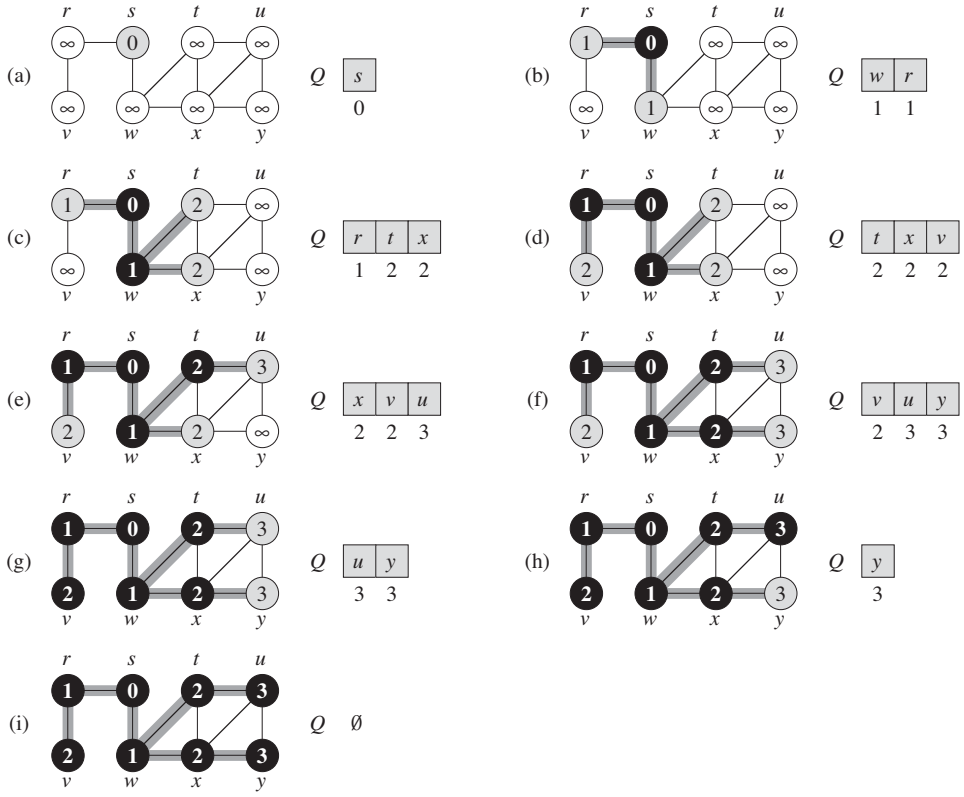
            make v gray

            v.d = u.d + 1 // distance

            v.parent = u

            Enqueue(Q,v)

    change vertex u to black // after going through adjacency



At the same time: make Breadth First tree. If node v is discovered after u, then edge (u,v) is added to the tree and we say that u is a predecessor (parent) of v. A vertex is discovered at most once.

Run time:

n =number vertices
m = number edges

- Each vertex enqueued and dequeued at most once (why? Because never white again). Cost of these is O(1). n = number of vertices. Total over all vertices O(n).

- Scans adjacency list only when dequeued, therefore each adjacency list only at most once. Sum of all adjacency lists depends on number of edges O(m) where m = number of edges.

- Total O(n + m). Linear in number of vertices and edges.

BFS allows to find Shortest paths:

BFS discovers shortest paths distances. Shortest path distance from s to v:

delta(s,v) = minimum number of edges from any path from vertex s to v.

(infinity if no path exists)

Main theorem (we've simplified from book format):

Let G=(V,E) be a graph. Suppose the BFS is run on G from given source s. Then upon termination, for each vertex v in V, v.d computed by BFS satisfies

v.d = delta(s,v)

That is, v.d is the shortest path, shortest number of edges from source s.

Main idea of proof: induction over distance i (node u with distance i-1 from source discovers vertex v, and v is distance one away from u, and therefore distance i from source).