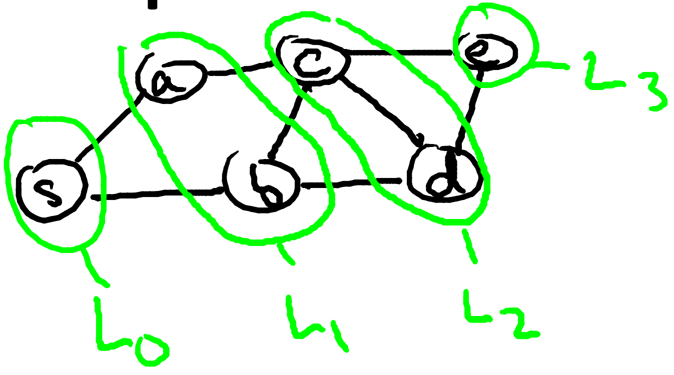
Graph Primitives

Breadth-First Search

Design and Analysis of Algorithms I

Overview and Example



Breadth-First Search (BFS)

-- explore nodes in "layers"
-- can compute shortest paths
-- connected components of undirected graph

Run time : O(m+n) [linear time]

The Code

BFS (graph G, start vertex s)

[all nodes initially unexplored]

-- mark s as explored

-- let Q = queue data structure (FIFO), initialized with s

-- while $Q \neq \phi$:

    -- remove the first node of Q, call it v

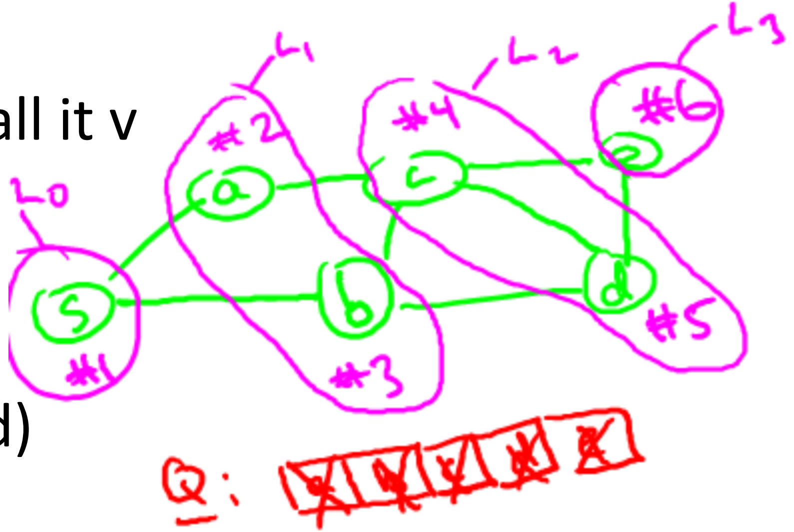
    -- for each edge(v,w) :

        -- if w unexplored

            --mark w as explored

            -- add w to Q (at the end)

O(1) time



Basic BFS Properties

Claim #1 : at the end of BFS, v explored <==> G has a path from s to v.

Reason : special case of the generic algorithm

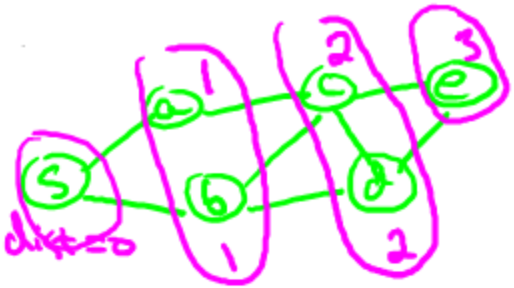
Claim #2 : running time of main while loop $= O(n_s+m_s)$, where n_s = # of nodes reachable from s

m_s = # of edges reachable from s

Reason : by inspection of code.

Application: Shortest Paths

Goal : compute dist(v), the fewest # of edges on a path from s to v.



Extra code : initialize $\text{dist}(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s \end{cases}$

-When considering edge (v,w) :

- if w unexplored, then set dist(w) = dist(v) + 1

Claim : at termination dist(v) = i <==> v in ith layer (i.e., shortest s-v path has i edges)

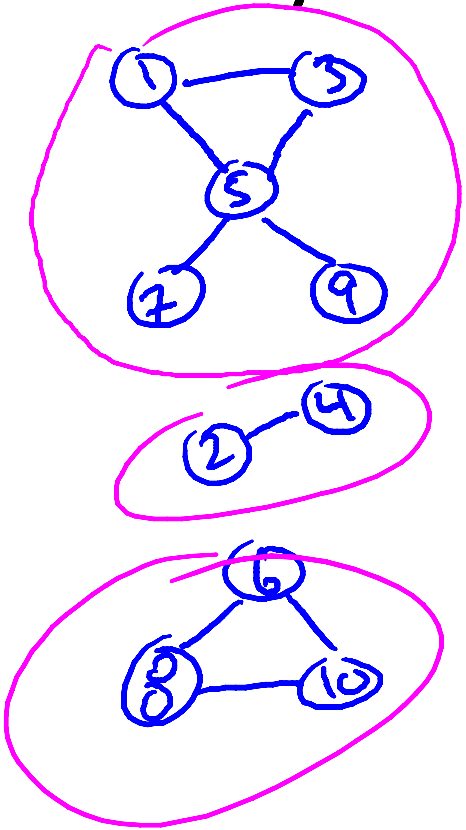
Proof Idea : every layer i node w is added to Q by a layer (i-1) node v via the edge (v,w)

Application: Undirected Connectivity

Let G = (V,E) be an undirected graph.

<u>Connected components</u> = the "pieces" of G.

<u>Formal Definition</u> : equivalence classes of the relation u<->v <==> there exists u-v path in G. [check: <-> is an equivalence relation]



<u>Goal</u> : compute all connected components

<u>Why?</u> - check if network is disconnected

- graph visualisation
- clustering

Connected Components via BFS

To compute all components : (undirected case)

-- initalize all nodes as unexplored   O(n)

[assume labelled 1 to n]

-- for i = 1 to n   O(n)

    -- if i not yet explored [in some previous BFS]   O(n)

        -- BFS(G,i)   [discovers precisely i's connected component]

Note : finds every connected component.

Running time : O(m+n)

O(1) per node

O(1) per edge in each BFS