Breadth First Search & Depth First Search

Problem Given a graph $G(V,E)$, and a start node $s \in V$, find all nodes reachable from s (i.e. in the connected component of s).

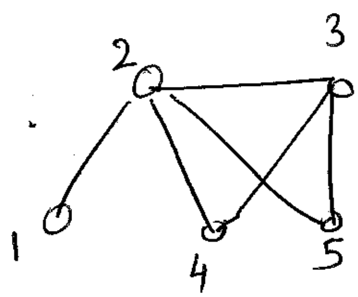
Representing graphs

(1) Adjacency Matrix

$$a_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

- $O(n^2)$ storage space
- Given i, find all neighbors of i. $O(n)$ time.
- Is $\{i,j\}$ an edge? $O(1)$ time.

(2) Adjacency list representation.



① : 2
② : 1-4-5-3
③ : 2-4-5
④ : 2-3
⑤ : 2-3

- For every node, list of its neighbors is given.
- $O(m)$ space where $m = \#$ edges.
- Find all neighbors of i : $O(d)$ time
  Is $\{i,j\}$ an edge ? : $O(d)$ time
  d = maximum degree of any node.

— x —

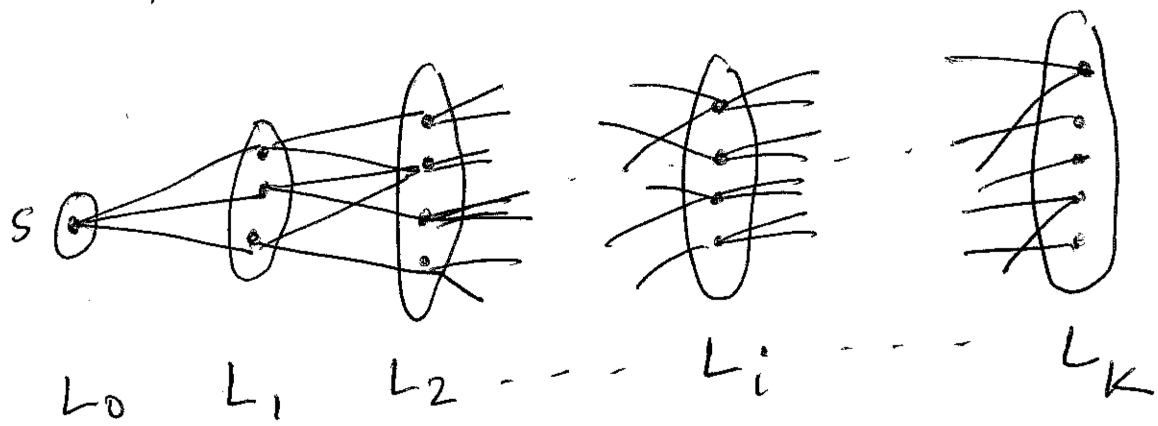
<u>Theorem</u>. Given a graph $G(V,E)$ in adjacency list rep., connected component of s can be found in $O(m+n)$ time using B.F.S. or D.F.S.

Breadth First Search

- Let layer L_0 be single node s. $L_0 = \{s\}$.
- Let
$$L_{i+1} = \left\{ v \;\middle|\; \begin{array}{l} v \notin L_0 \cup L_1 \cup L_2 \cdots \cup L_i, \\ (u,v) \in E \text{ for some } u \in L_i \end{array} \right\}.$$
  for $i = 0, 1, 2, \ldots, k$ so that $L_{k+1} = \phi$ then stop.
- Vertices reachable from s are

decomposed into layers

$L_0, L_1, L_2, \ldots, L_k$.



<u>Claim</u>

(1) There are no edges between layers L_i and L_j if $i+1 < j$. I.e. all edges are <u>either</u> inside some layer L_i <u>or</u> between adjacent layers L_i, L_{i+1}.

(2) L_i is precisely the set of vertices at distance i from s.

(3) $L_0 \cup L_1 \cdots \cup L_k$ is the connected component of s.

<u>proof</u>. Exercise.

Theorem BFS can be performed in $O(m+n)$ time.

Proof Here is the algorithm.

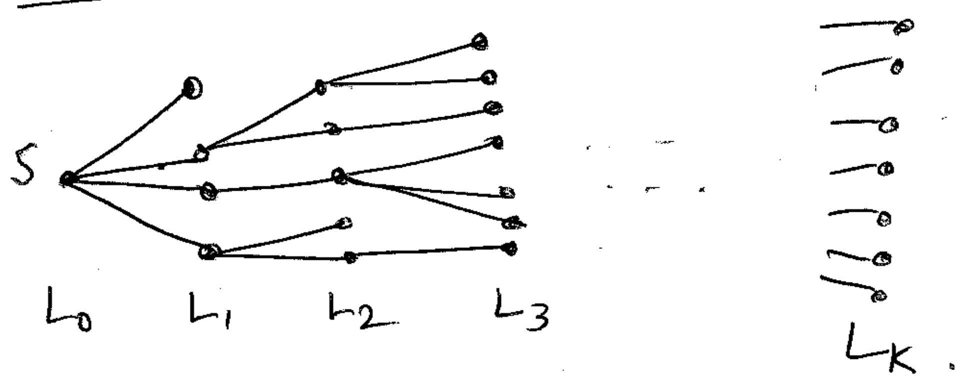
Maintain a marker for all vertices indicating whether they have been visited.

Initialize $L_0 = \{s\}$ and mark s. All other vertices are unmarked.

For $i = 0, 1, 2, 3, \ldots$

- Go over all $u \in L_i$.
- For every $u \in L_i$, go over all edges (u,v) incident on u. If v hasn't been marked, then mark it and add it to list L_{i+1}. Also add the edge (u,v) to "BFS tree".

"BFS tree"

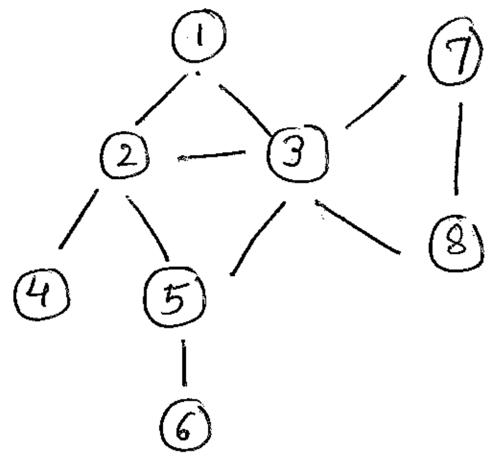


The shortest paths from s are given by the unique paths in the BFS tree.

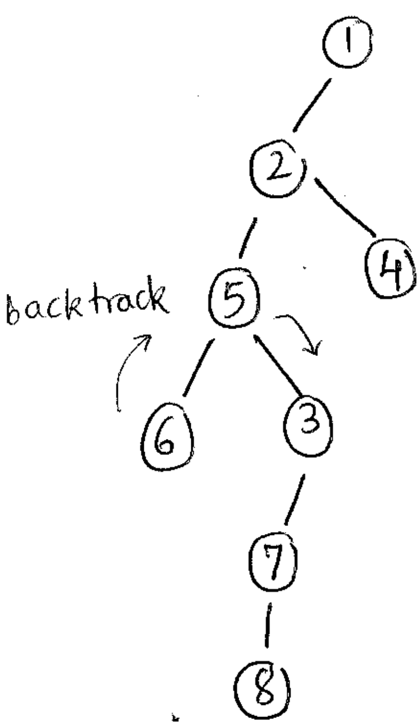
Depth first search $G(V,E)$, s.

- Attempt to visit a new/undiscovered node if possible.
- Backtrack if necessary.

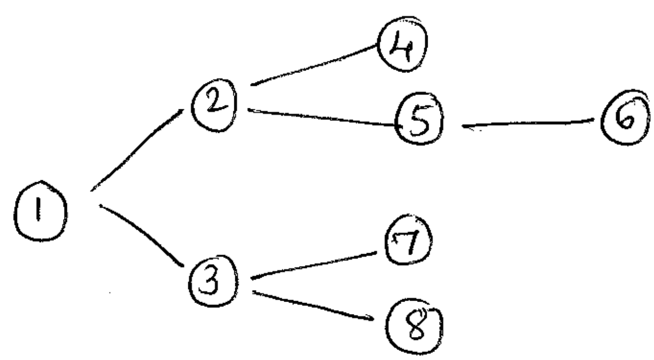
Example



DFS tree



BFS tree



Algorithm

Explored $(v) = F \quad \forall v \in V$.

DFS (s).

DFS (v) {

Explored $(v) = T$.

Let $u_1, u_2, \dots, u_k$ be neighbors of v.

For $i = 1, 2, \dots, k$ {

If Explored $(u_i) = F$, { Add ege (v, u_i) to DFS tree. DFS (u_i). }

}

}

Theorem DFS runs in $O(m+n)$ time.

Proof $\because$ each edge is examined $O(1)$ times.

Stack implementation of DFS

Explored (v) = F $\forall v \in V$.

Initialize Stack to $\{s\}$.

While { stack $\neq \phi$ } {

- Take topmost node u from the stack.
- If Explored (u) = F then {

    set Explored (u) = T.

    $\forall (u,v) \in E$, push v to stack.

  }

}.

Applications of BFS

Testing bipartiteness of graphs-

Def $G(V,E)$ is bipartite if $V = U \cup W$ and every edge has one endpoint each in U and W.

Fact G is bipartite iff G has no odd cycle.

Problem Given $G(V, E)$ in adj. list rep., decide if G is bipartite.

Algorithm $O(m+n)$ time.

- Do BFS starting at any node.
- $L_0, L_1, L_2, \ldots, L_K$ be layers.
- If there is no edge inside any layer L_i, declare G to be bipartite.

  Else declare G to be non-bipartite.

Proof of correctness

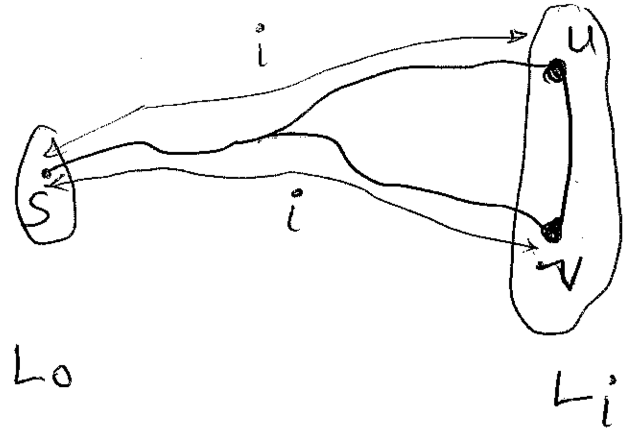
① If no edge inside any layer L_i then let

$U = L_0 \cup L_2 \cup L_4 \cup L_6 \cdots$

$W = L_1 \cup L_3 \cup L_5 \cup L_7 \cdots$

$U \cup W$ is a bipartition as all edges are across layers L_i and L_{i+1}.

② Suppose there is an edge (u,v) inside L_i.



Then $s \overset{i}{\rightsquigarrow} v \xrightarrow{1} u \overset{i}{\rightsquigarrow} s$ is a closed walk of length $2i+1$ (odd). Hence G is not bipartite.

> Note. BFS, DFS can also be performed on directed graphs.

i.e. - Given directed graph $G(V,E)$, node s, all nodes reachable from s can be found in $O(m+n)$ time.

- Let $G^{reverse}$ be the graph G with all edge directions reversed. By performing BFS/DFS on $G^{reverse}$, one can find in $O(m+n)$ time, all nodes from which s can be reached.

<u>Def</u> Let $G(V,E)$ be a directed graph. G is called <u>strongly connected</u> if $\forall u,v \in V$, there is $u \leadsto v$ path (and also $v \leadsto u$ path).

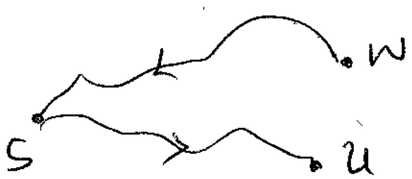
<u>Problem</u> Given dir. $G(V,E)$, decide if G is strongly connected.

<u>Algorithm</u> $O(m+n)$ time.

Fix some $s \in V$. Note that $G(V,E)$ is strongly connected <u>iff</u>

(1) All nodes are reachable from s.

(2) s can be reached from all nodes.



$w \leadsto s$

Both tasks (1), (2) can be checked in $O(m+n)$ time by BFS/DFS as described.