Overview

⇒ Representation of Graphs
  • Breadth First Search
  • Depth-First Search
Storing a Graph

- Graph: set of vertices and edges between vertices:
  - $G = (V, E)$ where $V$ is a set of vertices and $E$ is a set of pairs $(u, v)$
    where $u, v \in V$
  - Undirected: $u$ and $v$ are connected
  - Directed: you can go from $u$ to $v$ (can also have edge $(v, u)$)

- Can store a graph:
  - Adjacency list for each vertex
    - Needs size $\Theta(|V| + |E|)$
  - Adjacency matrix:
    - Assume vertices numbered from 0 to $|V| - 1$
    - Needs size $\Theta(|V|^2)$

Examples

- Undirected Graph

- Directed Graph
Question

22.1-6

Most graph algorithms that take an adjacency-matrix representation as input require time $\Omega(V^2)$, but there are some exceptions. Show how to determine whether a directed graph $G$ contains a universal sink—a vertex with in-degree $|V| - 1$ and out-degree 0—in time $O(V)$, given an adjacency matrix for $G$. 

Which is Better: Adjacency lists or matrix?

- Which takes less space if matrix is sparse or dense?
  - If sparse, adjacency list might take less space
  - If dense, adjacency matrix takes less space

- Which is better if iterating over edges?
  - If edges are sparse, adjacency matrix takes $|V|^2 \gg |E|$

- If need to check if edge between two vertices
  - Adjacency matrix gives $O(1)$ time

- Either can be used for weighted graphs
Breadth-first Search

- Given a graph \( G = (V, E) \) and a source vertex \( s \)
  - BFS systematically explores the edges of \( G \) to discover every vertex that is reachable from \( s \)
  - Computes distance (smallest \# of edges) from \( s \) to each reachable vertex
  - Produces \textit{breadth-first tree} with root \( s \) containing all reachable vertices
    - For any vertex \( v \) reachable from \( s \), the simple path in the breadth-first tree from \( s \) to \( v \) corresponds to a “shortest path” from \( s \) to \( v \) in \( G \)
  - Works on both directed and undirected graphs
Overview of How it Works

- BFS is so named because it expands a frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier
- Discovers all vertices at distance $k$ from $s$ before discovering any vertices at distance $k + 1$
- Can be viewed as coloring the vertices:
  - gray on the frontier
  - black discovered
  - white undiscovered

How it Works

- Frontier of gray vertices kept as queue
- Initially put root node on queue
- Take top node $v$ off of queue
  - For each node $u$ adjacent to $v$ that is white
  - Add edge $(u, v)$ to tree (or set predecessor of $v$ to $u$)
  - Set $v.d$ to $u.d + 1$
  - Color $v$ gray and add it to queue
- Breadth-First Tree
  - Has all vertices reachable from root, and edges used in algorithm
  - Defines parent (or predecessor), ancestor and descendant relationship
Code

BFS(G, s)
1 for each vertex u ∈ G.V − {s}
2 \hspace{1em} u.color = \text{WHITE}
3 \hspace{1em} u.d = \infty
4 \hspace{1em} u.\pi = \text{NIL}
5 \hspace{1em} s.color = \text{GRAY}
6 \hspace{1em} s.d = 0
7 \hspace{1em} s.\pi = \text{NIL}
8 \hspace{1em} Q = \emptyset
9 \hspace{1em} \text{ENQUEUE}(Q, s)
10 while Q ≠ ∅
11 \hspace{1em} u = \text{DEQUEUE}(Q)
12 \hspace{1em} for each v ∈ G.Adj[u]
13 \hspace{1em} \hspace{1em} if v.color == \text{WHITE}
14 \hspace{1em} \hspace{1em} v.color = \text{GRAY}
15 \hspace{1em} \hspace{1em} v.d = u.d + 1
16 \hspace{1em} \hspace{1em} v.\pi = u
17 \hspace{1em} \hspace{1em} \text{ENQUEUE}(Q, v)
18 \hspace{1em} u.color = \text{BLACK}
Code Revisited

BFS($G, s$)

1. for each vertex $u \in G.V - \{s\}$
2. $u.color = \text{WHITE}$
3. $u.d = \infty$
4. $u.pi = \text{NIL}$
5. $s.color = \text{GRAY}$
6. $s.d = 0$
7. $s.pi = \text{NIL}$
8. $Q = \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. $u = \text{DEQUEUE}(Q)$
12. for each $v \in G.Adj[u]$
13. if $v.color == \text{WHITE}$
14. $v.color = \text{GRAY}$
15. $v.d = u.d + 1$
16. $v.pi = u$
17. ENQUEUE($Q, v$)
18. $u.color = \text{BLACK}$

What is wrong with version below?

for $u$ in $Q$
    if $v.color == \text{white}$:
        $v.color = \text{gray}$
        $v.d = u.d + 1$
        $v.pi = u$
        $Q.append(v)$
$u.color = \text{black}$
Time Analysis

• Each vertex will be processed at most once through main loop
  - Each edge will be processed at most once for each vertex
  - $O(|V| \times |E|)$?
• But ....
  - Each edge is just processed at most once altogether for a directed graph
  - Since each edge is in at most one adjacency list
  - For undirected, each edge processed at most twice
  - $O(|V| + |E|)$
  - Why is this not $\Theta$?
  - What is the $\Omega$?

Question

22.2-4
What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input?
Toward Proving BFS gives Shortest Paths

- Define \( \delta(s, v) \) as the shortest path distance from \( s \) to \( v \)
  - Minimum number of edges in any path from \( s \) to \( v \)
  - If no path, then \( \infty \)
  - A path of length \( \delta(s, v) \) from \( s \) to \( v \) is called a shortest path
- Following Proof is similar to textbook

Does BFS find all Reachable Vertices?

**Proof by contradiction:**
Say that \( u \) is reachable from \( s \) but \( u \) is not in breadth-first tree
Since \( u \) is reachable from \( s \) there is a path from \( s \) to \( u \):
\( v_0, v_1, v_2, ..., v_n \) where \( s = v_0 \) and \( u = v_n \)
Must be a first vertex that is not in breadth-first tree, say \( v_i \)
\( v_{i-1} \) is in breadth-first tree and edge \((v_{i-1}, v_i)\) is not in tree
The BFS algorithm would have been added it.
Contradiction
Does BFS find a shortest path?

**Proof by contradiction:**
Let \( u \) be a vertex in the breadth-first tree and there is a minimum path from \( s \) to \( u \) is \( v_0, v_1, v_2, ..., v_n \) where \( s = v_0 \) and \( u = v_n \)
So \( \delta(s, v_j) = j \) for \( 0 \leq j \leq n \)
Assume BFS did not find a shortest path from \( s \) to \( u \)
so \( u.d > \delta(s, u) \)  
Reminder: \( u.d \) is from BFS, \( \delta(s, u) \) is truth
Must be a first vertex in \( v_0, ..., v_n \) that BFS assigns wrong depth. Say \( v_i \)
So BFS is correct for \( v_j \) for \( j < i \)
so \( v_j.d = j = \delta(s, v_j) \) for \( j < i \)
But BFS is wrong for \( v_i \)
so \( v_i.d = i > \delta(s, v_i) \)
\( i \neq 0 \) since \( s \) has a path of 0 length, which is found by BFS
so \( 0 < i \leq n \)

---

Continued

How does BFS add \( v_i \) to tree?
Case 1: \( v_i \) was added to the queue via \( v_{i-1} \)'s adjacency list
so \( v_i.d = v_{i-1}.d + 1 = i \).
Contradiction
Case 2: \( v_i \) was added before \( v_{i-1} \)'s adjacency list is processed
BFS assigns \( d \) in increasing fashion (same or 1 more than previous node)
Since \( v_i \) processed before \( v_{i-1} \)'s children, \( v_i.d \leq v_{i-1}.d + 1 = i \)
Contradiction
Overview

- Representation of Graphs
- Breadth First Search
  ⇒ Depth-First Search

Depth-First Search

- Explore adjacency list with a stack
- Explore all nodes
  - Can create several trees
- Vertex properties
  - Predecessor
  - Time-stamps
Code

DFS(G)
1 for each vertex u ∈ G.V
2 u.color = WHITE
3 u.π = NIL
4 time = 0
5 for each vertex u ∈ G.V
6 if u.color == WHITE
7 DFS-VISIT(G, u)

DFS-VISIT(G, u)
1 time = time + 1 // white vertex u has just been discovered
2 u.d = time
3 u.color = GRAY
4 for each v ∈ G.Adj[u] // explore edge (u, v)
5 if v.color == WHITE
6 v.π = u
7 DFS-VISIT(G, v)
8 u.color = BLACK // blacken u; it is finished
9 time = time + 1
10 u.f = time

Illustration

- Timestamps
  - u.d discover time
  - u.f finish time
- Gray nodes
  - Which ones are gray versus BFS
- Running time
Parenthesis Theorem

**Theorem 22.7 (Parenthesis Theorem)**

In any depth-first search of a graph $G = (V, E)$, for any two vertices $u$ and $v$, exactly one of the following three conditions hold:

- $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint and neither $u$ nor $v$ is a descendant of the other in the depth-first search.
- $[v.d, v.f]$ is contained entirely within $[u.d, u.f]$, and $v$ is a descendant of $u$ in the depth-first tree.
- vice versa.

**Properties**

- Predecessor subgraph $G_\pi$
  - Is a forest of trees (might just be one tree).
  - Vertex $v$ is a descendant of vertex $u$ in the depth-first forest iff $v$ is discovered during the time in which $u$ is gray.
  - Discovery and finishing times have parenthesis structure.
More Theorems

Corollary 22.8 (Nesting of descendants’ intervals)
Vertex $v$ is a proper descendant of vertex $u$ in the depth-first forest for a (directed or undirected) graph $G$ iff $u.d < v.d < v.f < u.f$

Theorem 22.9 (White-path theorem)
In a depth-first forest of a (directed or undirected) graph, vertex $v$ is a descendant of vertex $u$ iff at the time $u.d$ that the search discovers $u$, there is a path from $u$ to $v$ consisting entirely of white vertices.

Proof of White Path Theorem

Part 1: $\Rightarrow$
$v$ is a descendant of $u$ $\Rightarrow$ there is a path of white nodes from $u$ to $v$
So there is a path in the predecessor subgraph from $v$ to $u$
All nodes on path must have been added when they were white
So must have been white when $u$ was discovered (time $u.d$)

Part 2: $\Leftarrow$
there is a path of white nodes from $u$ to $v$ $\Rightarrow$ $v$ is a descendant of $u$
Say that $v$ is not a descendant.
WLOG, assume $v$ is first node in path that is not a descendant
So $v$’s predecessor in the path, say $p$, is a descendant of $u$
$(p, v) \in E$, and $v$ was white when $p$ was explored
So $v$ would have been added while $u$ was still gray
Classification of Edges

\[ G = (V, E) \text{ and } G_\pi: \text{ depth-first forest produced by a depth-first search on } G \]

- **Tree edges:** edges in \( G_\pi \)
  - i.e., \((u, v)\): \( v \) was first discovered by exploring edge \((u, v)\) (so \( v.\pi = u \))

- **Back edge:** edges \((u,v)\) connecting a vertex \( u \) to an ancestor \( v \) in a depth first search (will include self loops in directed graph)

- **Forward edge:** non tree edge \((u,v)\) connecting a vertex \( u \) to a descendant \( v \)

- **Cross edges:** all other edges.
  - Can go between vertices in the same depth-first tree, as long as one ancestor is not an ancestor of the other
  - Can also be between trees

Example

Previous graph redrawn so tree edges and forward edges point down, and back edges point up
Theorem 22.10

In a depth-first search of an undirected graph, every edge is either a tree edge or back edge. In other words, no cross edges or forward edges.

Remember how edge \((u,v)\) is classified. When processing \(u\) and edge \((u,v)\) is seen for the first time, what color is \(v\)? Can \(v\) be black?
Question

Give a directed graph in which there is a path from $u$ to $v$, and there is a DFS in which $u$ is not an ancestor of $v$.

Question 22.3-9

Give a counterexample to the conjecture that if a directed graph $G$ contains a path from $u$ to $v$, then any depth-first search must result in $v.d \leq u.f$.

In other words, give a directed graph in which there is a path from $u$ to $v$, and there is a DFS in which $u$ is fully processed before $v$ is even discovered.