Overview

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

Chapter 22

Elementary Graph Algorithms
Examples

• Undirected Graph

(a) (b) (c)

• Directed Graph

(a) (b) (c)

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 \( u, v \in V \).

• Can store a graph:

1. Adjacency list for each vertex:

   - Needs size \( \Theta(|V| + |E|) \)

2. Adjacency matrix:

   - Needs size \( \Theta(|V|^2) \)
   - Assume vertices numbered from 0 to \(|V| - 1\)

   \[ A \in \{0, 1\}^{V \times V} \]

   \( A[u, v] = 1 \) if \( u \) is connected to \( v \)
   \( A[u, v] = 0 \) otherwise

\( n \) is a set of vertices and \( E \) is a set of edges between vertices.

Graph: set of vertices and edges between vertices.
Question

22.1-6

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

Which is Better?

• 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 edges are sparse, adjacency list takes less space
  - If dense, adjacency list might take less space
  - If sparse, adjacency list is more space-efficient

• Which takes less space if matrix is sparse or dense?
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 $s$.
- Produces a breadth-first tree with root $s$, containing all reachable vertices.
- Computes distance (smallest # of edges) from $s$ to each reachable vertex.
- 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.

Depth-First Search
How it Works

• Frontier of grey 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 \( v.d = u.d + 1 \)
      - Color \( v \) grey and add it to queue

Breadth-First Tree

- Has all vertices reachable from root and edges used in algorithm
- Defined by parent (or predecessor) relationship
- Defines parent (or predecessor), ancestor and descendent relationships
  - Has all vertices reachable from root and edges used in algorithm

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
  - Can be viewed as coloring the vertices:
    - Grey on the frontier
    - White undiscovered
    - Black discovered

- Any vertices at distance \( k + 1 \) from \( s \) before discovering those at distance \( k \) from \( s \)
- Discovered all vertices at distance \( k \) before discovering any vertices at distance \( k + 1 \)
- Expands a frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier

Illustration

Code

BFS(G,s):
for each vertex u ∈ V / NULf
    color D WHITE
    d D 1
    D NIL
s:
    color D GRAY
    d D 0
    D NIL
Q D ;
ENQUEUE.Q;s/
while Q ≠ ;
ENQUEUE.Q/
for each u ∈ AdjŒu
    if color == WHITE
        color D GRAY
        d D u: d C 1
        D u
        ENQUEUE.Q;
    u:
    color D BLACK

n color = BLACK
n = n
n + p n = p n
'color = GRAY
if 'color == WHITE
for each w ∈ G.AdjŒsŒ
    DEQUEUE(ŒsŒ)
    if ≠ then
        ENQUEUE(ŒsŒ)
        s
        ENQUEUE(ŒsŒ)
        s

for each vertex u ∈ G. V
    BFS(G,u)
Towards 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 $\delta(s, v)$ is called a shortest path.

Following proof is similar to textbook.

Time Analysis

- Each vertex will be processed at most once through main loop.
- Each edge will be processed at most once for each vertex.
- For undirected, each edge processed at most twice.
- For directed, each edge is in at most one adjacency list.
- Each edge is just processed at most once altogether for a directed graph.

- $|E| + |V|)$
- $|E| \times |V|$

• Why is the $\Omega$?
• Why is this not $\Theta$?
Finds Shortest Paths

Proof by contradiction:

Let \( u \) be a vertex in the breadth-first tree whose best path from \( s \) to \( u \) is \( s = v_0, v_1, v_2, \ldots, u = v_n \).

Assume that \( \delta(s, u) < u.d \).

There must be a first vertex in the path that goes astray. Say a vertex \( v_i \) such that \( \delta(s, v_i) = i < v_i.d \).

But for \( j < i \), \( \delta(s, v_j) = j = v_j.d \) since \( s \) has a path of 0 length, which is found by BFS.

So \( i \neq 0 \).

Since \( u \) is reachable, there is a path from \( s \) to \( u \) of length \( n \).

Proof by contradiction:

Finds all Reachable Vertices

Proof by contradiction:

The BFS algorithm would have added it:

Since \( n \) is reachable from \( s \), there is a path from \( s \) to \( u \).

Say that \( u \) is reachable from \( s \) but is not in the breadth-first tree, say \( u = n \).

Since \( u \) is reachable from \( s \), there is a path from \( s \) to \( u \) of length \( n \).
Question

What is the running time of BFS if we represent the input graph by an adjacency matrix and modify the algorithm to handle this form of input?

Continued

How does BFS add \( v_j \) to the tree?

Case 1:

If \( v_i \) 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:

If \( v_i \) was before \( v_{i-1} \)'s adjacency list is processed, \( v_i \)'s path length in the breadth-first tree can be at most \( v_i - 1 + 1 \) since depths are processed systematically.

Contradiction

Case 1:

\[ t = I + p \cdot I \]

If \( v_i \) was added to the queue via \( v_{i-1} \)'s adjacency list, so \( v_i \).

Case 1:

How does BFS add \( v_i \) to the tree?

Continued
Depth-First Search

- Explore adjacency list with a stack
- Explore all nodes
- Can create several trees

Vertex properties
- Predecessor
- Time-stamps

Overview

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

Code

Running time

Which ones are grey versus BFS

Grey nodes

$f \cdot n$ - finish time

discover time

Timeslamps
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:

1. \([u.d, u.f]\) and \([v.d, v.f]\) are entirely disjoint and neither \( u \) nor \( v \) is a descendant of the other in the depth-first search.
2. \([u.d, u.f]\) is contained entirely within \([v.d, v.f]\), and \( u \) is a descendant of \( v \) in the depth-first tree.
3. Vice versa.

Properties

- Predecessor subgraph \( G' \) have parenthesis structure.
- Discovery and finishing times are in the range of the following three conditions (for any two vertices \( u \) and \( v \) in the depth-first forest):

- \( u \) is a descendant of \( v \) in the depth-first forest.
- \( v \) is a descendant of \( u \) in the depth-first forest.
- Neither \( u \) nor \( v \) is a descendant of the other.
Proof of White Path Theorem

Part 1: $\Rightarrow$

- $v$ is a descendant of $u$.
- 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 $v$.
- $v$ 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 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 there is a path in the predecessor subgraph from $v$ to $u$.
- So there is a path in the predecessor subgraph from $v$ to $u$.
- So there is a path in the predecessor subgraph from $v$ to $u$.
- So must have been white when $u$ was discovered (time $u.d$).
- So $v$ would have been a descendant (time $u.d$).
Example

Classification of Edges

- **Tree edges**: edges in $G_\pi$, i.e., $(u, v)$: $v$ was first discovered by exploring edge $(u, v)$ or $(v.\pi, v)$.
- **Back edges**: edges $(u, v)$ connecting a vertex $u$ to an ancestor $v$ in a depth-first search (will include self-loops in directed graphs).
- **Forward edges**: non-tree edge $(u, v)$ connecting a vertex $u$ to a descendant of $v$.
- **Cross edges**: all other edges.

- Can also be between trees.
- Can also be between vertices in the same depth-first tree, as long as one ancestor is not an ancestor of the other.
- Can also be between vertices in different depth-first trees.

Can be between vertices in the same depth-first tree, as long as one ancestor is not an ancestor of the other.

Can also be between vertices in different depth-first trees.

Can also be between vertices in the same depth-first tree, as long as one ancestor is not an ancestor of the other.
In a depth-first search of an undirected graph, every edge is either a tree edge or back edge.

**Theorem 22.10**

When search over all edges, how does color of node reached:

- White?
- Gray?
- Black?

When search over all edges, how does color of node reached:

- White?
- Gray?
- Black?

Code snippet:

```c
#include <stdio.h>
#include <stdlib.h>

void DFS_VISIT(Gr, n)
{
    int time = time + 1;
    line = time + 1;
    n.color = 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 the ancestor of $v$.

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.

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