Algorithm Design and Analysis

Lecture 5

- Exploring graphs

Adam Smith

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Puzzles

• Suppose an undirected graph $G$ is connected.
  – True or false? $G$ has at least $n-1$ edges.

• Suppose that an undirected graph $G$ has exactly $n-1$ edges (and no self-loops)
  – True or false? $G$ is connected.
  – What if $G$ has $n-1$ edges and no cycles?
Paths and Connectivity

Def. A path in an undirected graph $G = (V, E)$ is a sequence $P$ of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair $v_i, v_{i+1}$ is joined by an edge in $E$.

Def. A path is simple if all nodes are distinct.

Def. An undirected graph is connected if for every pair of nodes $u$ and $v$, there is a path between $u$ and $v$. 
Cycles

Def. A cycle is a path $v_1, v_2, ..., v_{k-1}, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k-1$ nodes are all distinct.

cycle $C = 1-2-4-5-3-1$
Trees

Def. An undirected graph is a tree if it is connected and does not contain a cycle.

Theorem. Let $G$ be an undirected graph on $n$ nodes. Any two of the following statements imply the third.

- $G$ is connected.
- $G$ does not contain a cycle.
- $G$ has $n-1$ edges.

![Tree Graph](image)
Rooted Trees

**Rooted tree.** Given a tree $T$, choose a root node $r$ and orient each edge away from $r$.

**Importance.** Models hierarchical structure.
Phylogeny Trees

Phylogeny trees. Describe evolutionary history of species.
Exploring a graph

Classic problem: Given vertices \( s, t \in V \), is there a path from \( s \) to \( t \)?

Idea: explore all vertices reachable from \( s \)

Two basic techniques:

• Breadth-first search (BFS)
  • Explore children in order of distance to start node

• Depth-first search (DFS)
  • Recursively explore vertex’s children before exploring siblings

9/5/2008

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Breadth First Search

**BFS intuition.** Explore outward from $s$ in all possible directions, adding nodes one "layer" at a time.

**BFS algorithm.**
- $L_0 = \{ s \}$.
- $L_1 =$ all neighbors of $L_0$.
- $L_2 =$ all nodes that do not belong to $L_0$ or $L_1$, and that have an edge to a node in $L_1$.
- $L_{i+1} =$ all nodes that do not belong to an earlier layer, and that have an edge to a node in $L_i$.

**Theorem.** For each $i$, $L_i$ consists of all nodes at distance exactly $i$ from $s$. There is a path from $s$ to $t$ iff $t$ appears in some layer.
**Breadth First Search**

**Property.** Let $T$ be a BFS tree of $G = (V, E)$, and let $(x, y)$ be an edge of $G$. Then the level of $x$ and $y$ differ by at most 1.
**Connected Component**

*Connected component.* Find all nodes reachable from *s*.

![Graph](image)

*Connected component containing node 1 = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}.*
Flood Fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
- **Node**: pixel.
- **Edge**: two neighboring lime pixels.
- **Blob**: connected component of lime pixels.
Flood Fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
- **Node:** pixel.
- **Edge:** two neighboring lime pixels.
- **Blob:** connected component of lime pixels.

recolor lime green blob to blue
Connected Component

Connected component. Find all nodes reachable from $s$.

---

$R$ will consist of nodes to which $s$ has a path
Initially $R = \{s\}$
While there is an edge $(u, v)$ where $u \in R$ and $v \notin R$
  Add $v$ to $R$
Endwhile

---

Theorem. Upon termination, $R$ is the connected component containing $s$.
- BFS = explore in order of distance from $s$.
- DFS = explore in a different way.
Generic traversal algorithm

1. \( R = \{s\} \)

2. While there is an edge \((u, v)\) where \(u \in R\) and \(v \not\in R\),
   - Add \(v\) to \(R\)

To implement this, need to choose…

• Graph representation

• Data structures to track…
  – Vertices already explored
  – Edge to be followed next

These choices affect the order of traversal
The **adjacency matrix** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 
1 & \text{if } (i, j) \in E, \\
0 & \text{if } (i, j) \notin E.
\end{cases}$$

Storage: $\Theta(V^2)$

Good for **dense** graphs.

- Lookup: $O(1)$ time
- List all neighbors: $O(|V|)$
An *adjacency list* of a vertex $v \in V$ is the list $\text{Adj}[v]$ of vertices adjacent to $v$.

- $\text{Adj}[1] = \{2, 3\}$
- $\text{Adj}[2] = \{3\}$
- $\text{Adj}[3] = \{}$
- $\text{Adj}[4] = \{3\}$

For undirected graphs, $|\text{Adj}[v]| = \text{degree}(v)$.
For digraphs, $|\text{Adj}[v]| = \text{out-degree}(v)$.

**How many entries in lists?** $2|E|$

**Total** $\Theta(V + E)$ storage — good for *sparse* graphs.
BFS with adjacency list rep.

- Discovered[1 .. n]: array of bits (explored or not)
  - initialized to all zeros
- Queue Q
  - initialized to empty
- Tree T
  - initialized to empty
BFS pseudocode

BFS(s):
1. Set Discovered[s]=1
2. Add s to Q
3. While (Q not empty)
   a) Dequeue (u)
   b) For each edge (u,v) adjacent to u
      a) IF Discovered[v]= false then
         a) Set Discovered[v] =true
         b) Add edge (u,v) to tree T
      c) Add v to Q
Theorem: BFS takes $O(m+n)$ time

BFS(s):
1. Set $\text{Discovered}[s]=1$
   \[O(1)\text{ time, run once overall.}\]
2. Add $s$ to $Q$
3. While (Q not empty)
   a) Dequeue $(u)$
   \[O(1)\text{ time, run once per vertex}\]
   b) For each edge $(u,v)$ adjacent to $u$
      a) IF $\text{Discovered}[v]=\text{false}$ then
         a) Set $\text{Discovered}[v]=\text{true}$
         \[O(1)\text{ time per execution, run at most twice per edge}\]
         b) Add edge $(u,v)$ to tree $T$
         c) Add $v$ to $Q$
   \[Total: O(m+n)\text{ time (linear in input size)}\]
• If $s$ is the roof of BFS tree
• For every vertex $u$,
  – path in BFS tree from $s$ to $u$ is a shortest path in $G$
  – depth in BFS tree = distance from $u$ to $s$
• Proof of BFS correctness: see KT, Chapter 3.