Lecture 10:
• Breadth-First Search (cont)
• Shortest Paths
• Connected Components

Announcements
• HW-1 due
• Exam 1, Tues, Oct 4
• PS-3 out, due Thur, Oct 6
• HW-2 out (soon), due Tues, Oct 11

Breadth-First Search (cont)

Algorithm: BFS
Input: A graph $G = (V, E)$ and a source vertex $s \in V$.
Result: Dictionary mapping discovered vertices to their parent vertices.

1 begin
2 mark all vertices $v \in V$ as not visited
3 mark vertex $s$ as visited
4 set parent of $s$ to -1 (denoting the root)
5 $Q \leftarrow$ a queue data structure initialized with $s$
6 while $Q$ is not empty do
7 $u \leftarrow$ the vertex at the front of $Q$ (dequeue)
8 for $v$ adjacent to $u$ in $E$ do
9 if $v$ is not yet visited then
10 mark $v$ as visited
11 set $v$’s parent to $u$
12 add $v$ to the end of $Q$
Breadth-First Search (cont)

Analysis of BFS:

(1). initialization of array for visited nodes is $O(V)$

(2). each directed out-edge from a “frontier” node is visited once

(3). each undirected edge from a “frontier” node is visited twice

(4). thus, $O(E)$ edges visited

(5). making BFS $O(V + E)$ ... which is linear in the size of the graph!

Check in: is there a best case versus a worst case?
  • e.g., what happens when nothing is reachable from $s$?

BFS Observations

1. The layers partition the vertices reachable from $s$
   • thus, each reachable vertex is in exactly one layer (L1, L2, ...)

2. The layer denotes the minimum number of edges from $s$ to layer vertices
   all vertices 1 edge from $s$ are in Layer 1
   all vertices 2 edges from $s$ are in Layer 2
   all vertices 3 edges from $s$ are in Layer 3
   ...
   all vertices $d$ edges from $s$ are in Layer $d$

   • minimum since existence of shorter path would put vertex in earlier layer
BFS Observations (cont)

3. **BFS computes the shortest paths from** $s$
   - the **length** of a path is the number of path edges
   - the **shortest** path is the path with minimal length
   - the layer of $v$ denotes the shortest path length from $s$ to $v$
   - path length is one way to define "distance" between vertices

BFS is the "go to" algorithm for computing shortest paths ...
   - simple reachability queries
   - computing diameter of a graph (e.g., max moves needed to solve a game)
   - network routing tables
   - computing "Bacon" numbers

Connected Components

Let $G = (V, E)$ be a graph
- A **connected component** of $G$ is the largest (maximal) subset $S \subseteq V$ with a path between each pair of vertices in $S$
- Alternatively: an equivalence relation where the relation is reachability

**Check in:** How many connected components are in the following graph?

```
0 2 4 6
1 3 5 7
```

**Check in:** What are min-max number of components in an undirected graph?
- 1 (all vertices) to $|V|$ (no edges)

**Applications:** check if graph is “disconnected” (e.g., computer network), find all components ("clusters")
Connected Components (cont)

We can find connected components in an undirected graph using BFS ...

Basic Idea:
• start with a vertex $u$ and determine vertices reachable from $u$ (via BFS)
• find a vertex $v$ not reachable from $u$ (i.e., not yet discovered)
• determine vertices reachable from $v$ (via BFS)
• repeat until all vertices are in a connected component

Comments:
• this only works for undirected graphs (see below)
• the connected component formed by $u$ is disjoint from the one formed by $v$
• Q: why? ... otherwise, $v$ would be reachable from $u$!

Analysis:
• Q: What is the cost? ... $\Theta(V + E)$
• unlike BFS from a vertex, we must visit every edge

HW-2 asks you to implement this basic algorithm
• assume connected components are numbered from 0, 1, 2, ···
• maintain mapping from vertex to its connected component
• use the mapping to check if a vertex is discovered or not
• must implement as augmented BFS algo (not just calling BFS)

In a directed graph, this algorithm finds "weakly" connected components
• treats the directed edges as undirected (ignoring the direction)
• we'll discuss approaches later to find "strongly" connected components