1 Breadth-First Search: Motivation

Idea: From a start vertex $s$, try to reach any vertex of an unweighted graph by a path traversing the fewest possible edges.

- **Example:** airline travel planning
- Starting from home (e.g., St. Louis), how many connecting flights does it take to reach each other city in the U.S.?
- **Abstraction:** vertices = airports, edges = flights
- **unweighted:** counting flights, not miles traveled

How might we try to solve this problem?

- First, find every city you can reach from home $s$ in one flight
- In graph, these cities correspond to what? ... vertices in Adj[$s$]
- For each city reachable in one step, find all the cities reachable in one more step by the same rule, etc.
- For each city, remember
  - How many flights did you need to get there?
  - How did you get there (most recent flight)?

2 More Ideas, and an Example

In a graph that is not a tree, how can we avoid traversing a vertex multiple times?

- **mark** vertices as we see them
- process vertices in FIFO order
How do we implement marking and FIFO order?

- FIFO: use an ordinary queue (*not* a priority queue)
- marking: each vertex has a “visited” field
- Set visited field, distance, and a *parent* pointer of each new vertex as it is enqueued
3 Pseudocode

Given graph $G = (V, E)$, starting vertex $s$. Use FIFO queue $Q$

BFS($G$, $s$)

for $u \in V - \{s\}$ do

  $u$.distance $\leftarrow \infty$
  $u$.visited $\leftarrow$ false
  $u$.parent $\leftarrow$ null

$s$.distance $\leftarrow 0$
$s$.visited $\leftarrow$ true
$Q$.enqueue($s$)

while $Q$ is not empty do

  $u \leftarrow Q$.dequeue()

  for $v \in \text{Adj}[u]$ do

    if not $v$.visited

      $v$.distance $\leftarrow u$.distance + 1
      $v$.visited $\leftarrow$ true
      $v$.parent $\leftarrow u$
      $Q$.enqueue($v$)

4 Correctness

- Visited fields ensure that each vertex is assigned distance only once.
- But is it assigned minimum distance from $s$?
- **Claim**: Every vertex of $G$ is enqueued in strict order by its distance from $s$, with its correct distance set at the time of enqueueing.
- **Pf**: by induction on distance from $s$.
  - **Bas**: Vertex $s$ is enqueued first with correct distance 0.
  - **Ind**: Suppose the claim holds for vertices up to distance $d - 1$.

- Then all vertices at distance $\leq d - 1$ are enqueued before any vertex at distance $d$, with correct distances.
- By FIFO property of $Q$, all vertices at distance $d - 1$ will be dequeued before any vertex at distance $d$.
- Each vertex $v$ at distance $d$ is adjacent to some vertex $w$ at distance $d - 1$; hence, $v$ will be discovered and assigned its correct distance when $w$ is dequeued.
- Finally, since no vertex at distance $d$ is dequeued until all vertices at distance $d - 1$ are dequeued, all vertices at distance $d$ will have been enqueued by the time the last vertex at distance $d - 1$ is processed. QED
5 Efficiency

- Initialization: $\Theta(1)$ per vertex = $\Theta(n)$ total
- No vertex added to queue more than once (visited field prevents it)
- Conclude $O(n)$ passes through outer `while` loop (Why not just $n$? graph may not be connected)
- What about time spent in inner `for` loop?
- For each vertex dequeued, all its edges are inspected.
- Since each vertex dequeued at most once, total cost of inner loop at most sum of adjacency list lengths = $O(m)$
- Hence, total cost is $O(n + m)$.

6 Breadth-First Tree

Parent pointers of each vertex after BFS form a tree rooted at $s$.

Is this tree unique? **No**: could enqueue two vertices of equal distance in either order, depending on adjacency list ordering