Breadth-first Search

Algorithmic Thinking

Luay Nakhleh

Department of Computer Science

Rice University

Graph Exploration

- Elucidating graph properties provides a powerful tool for understanding networks and their emergent properties.
- Some properties of interest include: degree distribution, community structure, node centrality, clustering coefficients,...
- When the graph is huge, exhaustive analysis of the entire graph is infeasible.
- The alternative: Explore a region (or, regions) of the graph, and report the results based on this exploration.

Graph Exploration

- Graph exploration can be done
  - deterministically: using, for example, breadth-first search (BFS) or depth-first search (DFS)
  - nondeterminstically: using random walks.
- Here, we will focus on BFS

Breadth-First Search (BFS)

- The main idea behind BFS is very simple:
  - Starting from some pre-specified node, explore the node and its neighbors; then, for each neighbor, explore its neighbors; and so on until no more nodes can be explored!

Breadth-First Search (BFS)

- The question is: How do we ensure that all the neighbors of a given node are explored before any of their neighbors are?

- That is, suppose we are exploring node u and its neighbors v, w, and x. How do we make sure v, w, and x are explored before a neighbor of, say, w, is explored (because if the neighbor of w gets explored before, say, x, this would <u>not</u> be BFS)

  - The answer: By using an appropriate data structure!

Queues

- A *queue* is a data structure that implements a first-in first-out (FIFO) data access model.

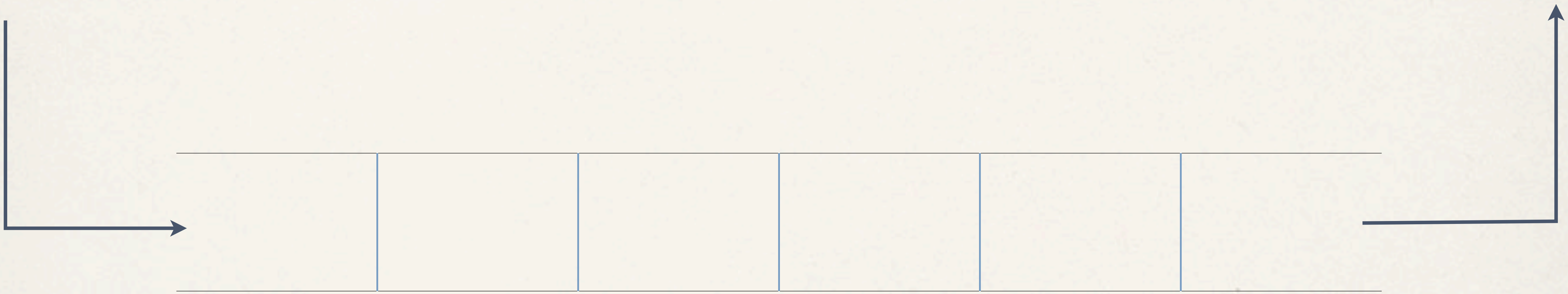
- Just think how a queue works when you go to a post office: the first person to enter the queue would be the first person served and the first person to leave the queue.

- Contrast this with a last-in first-out (LIFO) model: think of trays at a cafeteria; the last tray put on the stack of trays would be the first one picked up for use (the data structure that implements this model is called a *stack*).

Queues

Enter the queue
(last)

Leave the queue
(first)



Queues

- Two main operations are defined on queue Q:
  - enqueue(Q,x): add element x to the queue (at the end)
  - dequeue(Q): remove the first element that was enqueued into Q and return it
  - Queues can be implemented so that each of the two operations takes O(1) time.

BFS and Queues

- When a node u is reached during the exploration of the graph, enqueue all the neighbors of u, and when done with all of u's siblings, go to u's neighbors (by getting them out of the queue).

Breadth-First Search (BFS)

- Graph exploration via BFS (or any other exploration method) is done to compute some statistic (e.g., distances) or test a property of the graph (e.g., acyclicity).

- Therefore, it is typical to see a BFS algorithm coupled with additional statements to conduct such computations and/or tests.

- Here, we will illustrate the use of BFS to compute distances (from the start node) while exploring the graph.

Breadth-First Search (BFS)

Algorithm 1: BFS.

Input: Undirected graph $g = (V, E)$; source node i.
Output: $d_j, \forall j \in V$: the distance between nodes i and j.

Initialize Q to an empty queue;
foreach $j \in V$ **do**
  $d_j \leftarrow \infty$;
$d_i \leftarrow 0$;
$enqueue(Q, i)$;
while Q *is not empty* **do**
  $j \leftarrow dequeue(Q)$;
  foreach *neighbor* h *of* j **do**
    if $d_h = \infty$ **then**
      $d_h \leftarrow d_j + 1$;
      $enqueue(Q, h)$;
return d;

before the exploration begins, distances to all nodes are infinite

the first time the algorithm enters this loop, only node i is in the queue!

place all neighbors (that haven't been explored) of the currently explored node in the queue

Efficiency

- If input graph g is represented as an adjacency matrix, BFS takes $\Theta(n^2)$, where n is the number of nodes.
- The two key issues that you should observe to arrive at this running time are:
  - Every node gets added to the queue only once and in every iteration of the loop at Line 6, a node is removed from the queue.
  - To find the neighbors at Line 8, the algorithm has to inspect O(n) entries in the matrix.

Efficiency

- If input graph g is represented as an adjacency list, BFS takes $\Theta(n+m)$, where n is the number of nodes and m is the number of edges.
- To see this, think of two factors:
    - The loop at Line 2 takes O(n) to initialize the distances.
    - Don't think about the loop of Line 6 in terms of the number of nodes; instead, convince yourself that in that loop every edge in the graph gets traversed exactly twice; hence, the loop performs O(n+m) work.