COMP 182 Algorithmic Thinking

Breadth-first Search

Luay Nakhleh
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, distances, clustering coefficients,…
- Quantifying these properties provides an alternative to "inspection by visualization."

Graph Exploration

- Furthermore, when the graph is large, exhaustive analysis of the entire graph is infeasible.
- The alternative: Explore a region (or, regions) of the graph, and report the results based on the explored part(s).

Graph Exploration

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

Breadth-first Search (BFS)

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

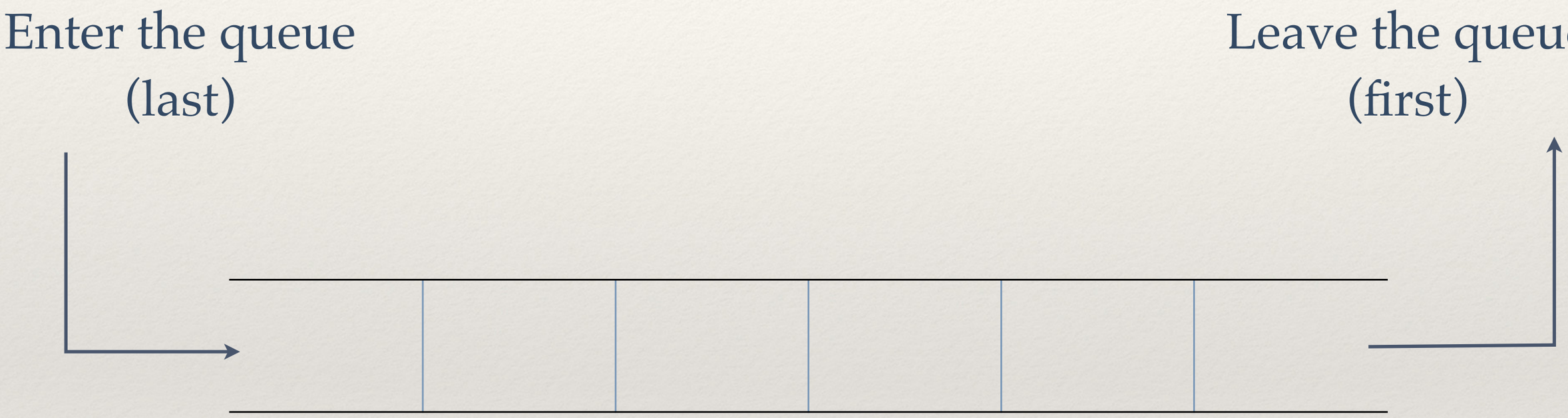
Breadth-first Search (BFS)

- The question is: How do we ensure that all the neighbors of a give node are visited before any of their neighbors are?
- 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.
- Think of how a queue works at a cafeteria: 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 the trays at the cafeteria; the last tray added to the stack would be the first one picked up for use.

Queues



Queues

- Two main operations are defined on queue *Q*:
  - *enqueue(Q,x)*: add element *x* to the end of the queue
  - *dequeue(Q)*: remove the first element that was enqueued into Q and return it

BFS and Queues

- When a node *u* is visited during the exploration of the graph, all of its neighbors are enqueued.
- When done visiting all of *u*'s siblings, *u*'s neighbors are visited by getting them out of the queue.
- So, queues are used in BFS to ensure a proper implementation of the algorithm.

Breadth-First Search (BFS)

Algorithm 1: BFS.

Input: Graph $g = (V, E)$; source node $i \in V$.
Output: $v_j \in \{\text{True}, \text{False}\}\ \forall j \in V$.

```
1  foreach j ∈ V do
2      v_j ← False;                          // Node j has not been visited yet
3  v_i ← True;                               // Start by visiting the source node i
4  Initialize Q to an empty queue;
5  enqueue(Q, i);
6  while Q is not empty do
7      j ← dequeue(Q);
8      foreach neighbor h of j do
9          if v_h = False then
10             v_h ← True;
11             enqueue(Q, h);
12 return v;
```

Breadth-First Search (BFS)

Algorithm 1: BFS.

Input: Graph $g = (V, E)$; source node $i \in V$.
Output: $v_j \in \{\text{True}, \text{False}\}\ \forall j \in V$.

```
1  foreach j ∈ V do
2      v_j ← False;                          // Node j has not been visited yet
3  v_i ← True;                               // Start by visiting the source node i
4  Initialize Q to an empty queue;
5  enqueue(Q, i);
6  while Q is not empty do
7      j ← dequeue(Q);
8      foreach neighbor h of j do
9          if v_h = False then
10             v_h ← True;
11             enqueue(Q, h);
12 return v;
```

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

Breadth-First Search (BFS)

Algorithm 1: BFS.

Input: Graph $g = (V, E)$; source node $i \in V$.
Output: $v_j \in \{\text{True}, \text{False}\}\ \forall j \in V$.

```
1  foreach j ∈ V do
2      v_j ← False;                          // Node j has not been visited yet
3  v_i ← True;                               // Start by visiting the source node i
4  Initialize Q to an empty queue;
5  enqueue(Q, i);
6  while Q is not empty do
7      j ← dequeue(Q);
8      foreach neighbor h of j do
9          if v_h = False then
10             v_h ← True;
11             enqueue(Q, h);
12 return v;
```

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

place all neighbors of the currently visited node in the queue

BFS and Queues

- If the input graph g has n nodes and m edges, how many operations does BFS perform?
- Key idea: Think of how many times each edge is traversed during the execution of the algorithm.

Questions?