# Computer Science 331

## Graph Search: Breadth-First Search

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #31



## Outline







## Breadth-First Search

Another way to search a connected component of a graph

Given a graph $G = (V, E)$ and *source vertex* $s$, the algorithm finds a *breadth-first* tree with root $s$, that is, a subgraph $\widehat{G} = (\widehat{V}, \widehat{E})$ such that

- $\widehat{G}$ is a tree
- for every vertex $v \in V$, $v \in \widehat{V}$ if and only if $c$ is reachable from $s$ (that is, there is a path from $s$ to $v$ in $G$ — so $\widehat{G}$ is a spanning tree for a connected component of $G$)
- for each vertex $v \in G$, the simple path from $v$ up to $s$ in $\widehat{G}$ (in which each edge is an edge from a node to its parent in the tree) is a *shortest* path from $v$ to $s$ in $G$.

The version of the algorithm presented here also returns the *distance* from $v$ to $s$ in $G$ (that is, the *length* of a shortest path from $v$ to $s$).





## Idea

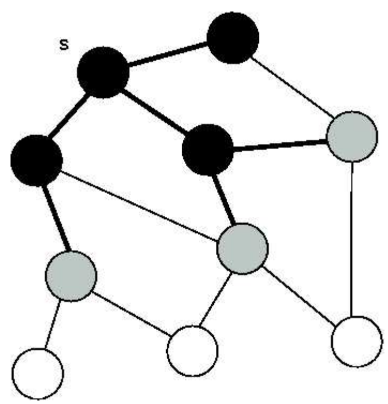
Begin with $s$; expand the boundary between "discovered" and "undiscovered" vertices uniformly across the breadth of the boundary

As in DFS, Vertices are coloured during the search

- All vertices are initially **white**, $s$ is almost immediately coloured **grey**.
- All white vertices are "undiscovered."
- "Discovered" vertices are either grey or black. Vertices on the boundary between discovered and undiscovered vertices are **grey**. Other discovered vertices are **black**.

Unlike DFS, when a grey vertex $t$ is processed, all white neighbours are recoloured grey; $t$ is then coloured black.

## Typical Search Pattern








## Specification of Requirements

**Precondition:** $G = (V, E)$ is a graph and $s \in V$

**Postcondition:**

- One value returned is a function $\pi : V \to V \cup \{NIL\}$ defining (with $s$) a *predecessor subgraph*, that is a spanning tree for the connected component of $G$ containing $s$
- For each vertex $v$ in the above spanning tree, the simple path from $v$ to $s$ in this tree is a *shortest path* from $v$ to $s$ in the graph $G$
- Another value returned is a function $d : V \to \mathbb{N} \cup \{+\infty\}$; for each vertex $v \in V$, $d[v]$ is the distance from $v$ to $s$ (so that $d[v] = +\infty$ if and only if there is no path from $v$ to $s$ in $G$).
- The graph $G$ has not been changed.





## Data and Data Structures

The following information is maintained for each $u \in V$ :

- $colour[u]$: Colour of $u$
- $d[u]$: Distance of $u$ from $s$
- $\pi[u]$: Parent of $u$ in tree being constructed

In order to ensure that the search is performed in a “breadth-first” way, a **queue** is used to store grey nodes





## Pseudocode

**BFS**$(G, s)$
  {Initialization}
  **for** each vertex $u \in V$ **do**
    $colour[u]$ = white {mark all vertices as undiscovered}
    $d[u] = +\infty$
    $\pi[u]$ = NIL
  **end for**
  $colour[s]$ = grey {start with source vertex $s$}
  $d[s] = 0$ {path from $s$ to itself has distance 0}
  $\pi[s]$ = NIL {$s$ is the root of the BFS tree (no parent)}
  Initialize queue $Q$ to be empty
  $Q$.add($s$) {add first grey node $s$ to the queue}

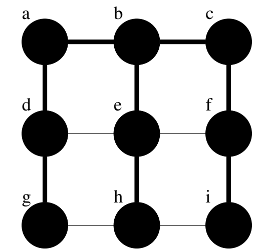
## Pseudocode, Continued

```
while (Q is not empty) do
  u = Q.remove()
  for each v ∈ Adj[u] do
    {examine neighbours of u}
    if colour[v] == white then
      colour[v] = grey   {discover each undiscovered neighbour}
      d[v] = d[u] + 1   {shortest path: s to u followed by (u, v)}
      π[v] = u   {u is the predecessor on the shortest path}
      Q.add(v)   {examine neighbours of v}
    end if
  end for
  colour[u] = black   {all neighbours of u have been discovered}
end while
return π, d
```





## Example




$Q$ (empty)

| | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| $d$ | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 4 |
| $\pi$ | NIL | a | b | a | b | c | d | e | f |





## Useful Properties Concerning Colours

Each of the following properties hold immediately before each execution of the outer `while` loop and before each execution of the inner `for` loop.

- All nodes that have never been added to the queue are `white`.
- All nodes that are currently on the queue are `grey`.
- All nodes that have been on the queue but later removed from it are `black`.
- All nodes that have been included in the predecessor subgraph (for $\pi$) are either `grey` or `black`. All other nodes are `white`.

It is also clear — by inspection of the code —that the colour of a node is never changed again once it becomes `black`.





## Useful Property of Distances

The *shortest-path distance* $\delta(s, v)$ from $s$ to $v$ is the minimum number of edges on a path from $s$ to $v$.

**Lemma 1**

*Let $G = (V, E)$ be an undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for every edge $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.*

**Proof.**

If $u$ is reachable from $s$ :

- one path from $s$ to $v$ : shortest path to $u$ followed by edge $(u, v)$
- shortest path to $v$ is at most as long as this path $(\delta(s, u) + 1)$

Otherwise, $\delta(s, u) = \infty$ and the inequality holds. □

## Lemma: Distance Inequality

### Lemma 2

*Let $G = (V, E)$ be an undirected graph, suppose BFS is run on $G$ from a given source vertex $s \in V$, and suppose that this algorithm terminates. Then (on termination of the algorithm), for each vertex $v \in V$ if $d[v]$ is a nonnegative integer then the sequence of edges*

$$(v, \pi[v]), (\pi[v], \pi[\pi[v]]), \ldots$$

*starting from $v$, and following edges from each vertex to its parent, forms a path of length $d[v]$ from $v$ to $s$ in $G$.*

### Method of Proof.

Prove that this property is satisfied at the both the beginning and end of each execution of the body of the `while` loop, using induction on the number of executions. □





## Lemma: Enqueued Vertices

### Lemma 3

*Suppose that, at the beginning of an execution of the body of the `while` loop, the queue $Q$ contains vertices $\langle v_1, v_2, \ldots, v_r\rangle$, where $v_1$ is the head of $Q$ and $v_r$ is the tail of $Q$. Then $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $1 \leq i \leq r - 1$.*

### Method of Proof.

Use induction on the number of nodes that have already been *removed* from the queue at this point. □

**Note:** One can show (indeed, one likely *does* show as part of the above proof) that this property is satisfied at the *end* of each execution of the body of the loop when the queue is nonempty, as well.





## Lemma: Distance and Queue Order

### Lemma 4

*Suppose that vertices $v_i$ and $v_j$ are added to the queue during the execution of BFS, and that $v_i$ is added before $v_j$. Then $d[v_i] \leq d[v_j]$ at the time $v_i$ is added to the queue.*

### Method of Proof.

Use induction on the number of vertices that were added to the queue *between* $v_i$ and $v_j$. Follows from Lemma 3, and the fact that each vertex only receives a finite $d$ value once. □





## Lemma: Correctness of Distance

### Lemma 5

*If a vertex $v$ is added to the queue at any point during the execution of the BFS algorithm, then $v$ is reachable from $s$. Furthermore, the value $d[v]$ that is set immediately before $v$ is added to the queue is equal to $\delta(s, v)$.*

See lecture supplement for complete proof.

## Lemma: Completeness of Predecessor Subgraph

### Lemma 6

*Suppose the BFS algorithm is run with an undirected graph $G = (V, E)$ and vertex $s \in V$ as input. If the algorithm terminates then, on termination, the predecessor subgraph for the function $\pi$ and vertex $s$ is a spanning tree for the connected component of $G$ that includes $s$.*

### Method of Proof.

Lemma 2 implies that every vertex in the predecessor subgraph (defined by $s$ and $\pi$) is reachable from $s$.

The fact that "if $v$ is reachable from $s$ then $v$ is included in the predecessor subgraph" — so that the predecessor subgraph is a spanning tree for the (entire) connected component containing $s$ — can be proved by induction on the distance from $v$ to $s$. □





## Partial Correctness of Breadth-First Search

### Theorem 7

*Let $G = (V, E)$ be a directed or undirected graph, and suppose BFS is run on $G$ from a given source vertex $s \in V$. Then each of the following properties is satisfied on termination of the algorithm (if it terminates):*

- *The predecessor subgraph $G_p = (V_p, E_p)$ for the function $\pi$ and vertex $s$ is a spanning tree for the connected component of $G$ that contains $s$.*
- *For all $v \in V$, $d[v]$ is the length of a shortest path from $v$ to $s$ in $G$, and $d[v] = +\infty$ if and only if $v$ is not reachable from $s$.*
- *For every $v \in V$ that is reachable from $s$, the path from $s$ to $v$ in $G_p$ is also a shortest path from $s$ to $v$ in $G$.*

### Proof.

Consequence of the previous lemmas. □





## Termination and Efficiency

### Theorem 8

*Let $G = (V, E)$ be a directed or undirected graph, and suppose BFS is run on $G$ from a given source vertex $s \in V$. Then the algorithm terminates after performing $O(|V| + |E|)$ operations.*

### Proof.

Exercise (can be completed by modifying the analysis of the DFS algorithm). □





## References

Text, Section 13.3

*Introduction to Algorithms*, Section 22.3: More details about the version of the algorithm presented here.