Breadth-First Search

CS 624 — Analysis of Algorithms

November 18, 2024

Definitions

A **graph** $G = (V, E)$ contains a set V of **vertices** and a set E of **edges**.

A **directed graph** has $E \subseteq V \times V$. An edge (u, v) is an edge from u to v, also written $u \to v$. Self loops such as (u, u) are allowed.

An **undirected graph** has $E \subseteq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$. An edge $\{u, v\}$ connects u and v. It is also written (u, v), but we consider $(u, v) = (v, u)$. Self loops are not allowed.

A **weighted graph** (either directed or undirected) also associates a weight with each edge, given by a weight function $w : E \to \mathbb{R}$.

More Definitions

- A graph is called **dense** if $|E| \approx |V|^2$, or **sparse** if $|E| \ll |V|^2$. In any case, $|E| = O(|V|^2)$.
- If $(u, v) \in E$, then vertex v is **adjacent** to vertex u.
- Adjacency relationship is symmetric if G is undirected, not necessarily so if G is directed.

More Definitions

For an undirected graph $G = (V, E)$:

- G is **connected** if there is a path between every pair of vertices.
- If G is connected, then $|E| \geq |V| - 1$.
- Furthermore, if G is connected and $|E| = |V| - 1$, then G is a tree.
- Other definitions in Appendix B (B.4 and B.5) as needed.

Graph Representations

One way to represent a graph is as a list of vertices, where each vertex has an **adjacency list** represnting its edges.

- For each vertex $v \in V$, we have a list $\mathrm{Adj}[v]$ consisting of those vertices u such that $(v, u) \in E$.
- It is actually a set, but usually implemented as a list.
- This works for both directed and undirected graphs.

  Directed graph: an edge (v, u) is represented by $u \in \mathrm{Adj}[v]$.

  Undirected graph: an edge (v, u) is represented by $u \in \mathrm{Adj}[v]$ and $v \in \mathrm{Adj}[u]$.

Another representation uses a single **adjacency matrix**.

Graph Search Algorithms

Searching a graph:

- Systematically follow the edges of a graph to visit all of the vertices of the graph.
- Used to discover the structure of a graph.
- Standard graph-searching algorithms:
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)

Breadth-First Search (BFS)

- BFS scans the graph G, starting from some given node s.
- BFS expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
- The key mechanism in this algorithm is the use of a queue, denoted by Q.

The BFS Algorithm

Algorithm 1 BFS(G, s)

for each vertex $u \in V[G] - \{s\}$ **do**
  $Color[u] \leftarrow$ White
  $d[u] \leftarrow \infty$
  $\pi[u] \leftarrow$ NIL
end for
$Color[s] \leftarrow$ Gray — discover s
$d[s] \leftarrow 0$
$\pi[s] \leftarrow$ NIL
$Q \leftarrow \varnothing$
$\mathrm{Enqueue}(Q, s)$
while $Q \neq \varnothing$ **do**
  $u \leftarrow \mathrm{Dequeue}(Q)$ — process u
  for each $v \in \mathrm{Adj}[u]$ **do**
    if $Color[v] =$ White **then**
      $Color[v] \leftarrow$ Gray — discover v
      $d[v] \leftarrow d[u] + 1$
      $\pi[v] \leftarrow u$ — (u, v) is a "tree edge"
      $\mathrm{Enqueue}(Q, v)$
    end if
  end for
  $Color[u] \leftarrow$ Black — finish u
end while

A vertex is "**discovered**" the first time it is encountered during the search.

A vertex is "**finished**" if all vertices adjacent to it have been discovered.

Colors indicate progress:

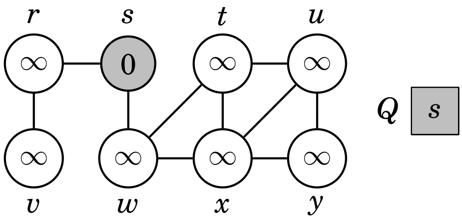
- White means undiscovered.
- Gray means discovered, not processed.
- Black means fully processed.

Colors are helpful for reasoning about the algorithm. Not necessary for implementation.

$d[u]$ is length of shortest path from s to u.

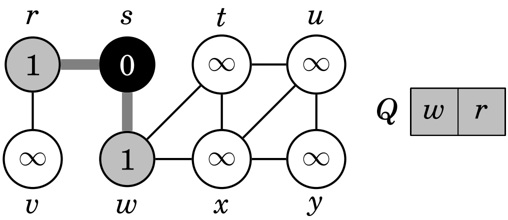
$\pi[u]$ is previous node on shortest path from s to u.

BFS Example



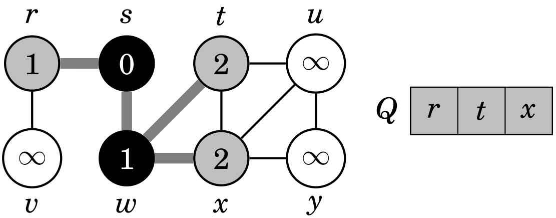
- Note that all nodes are initially colored white.
- A node is colored gray when it is placed on the queue.

BFS Example



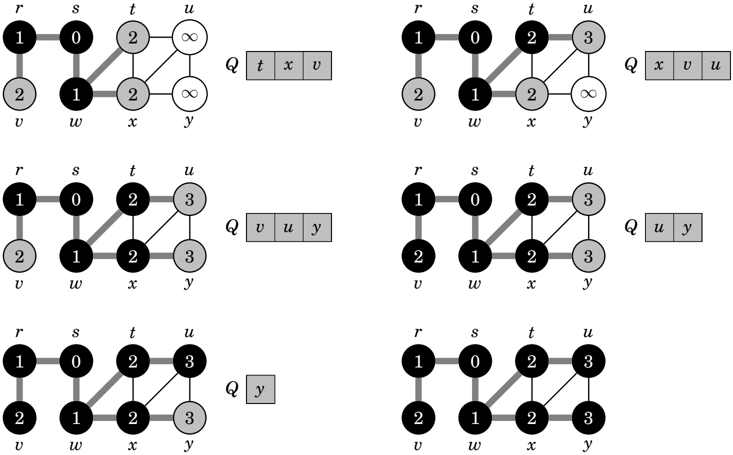
- A node is colored black when taken off the queue.
- Nodes colored white have not yet been visited. The nodes colored black are “finished” and the nodes colored gray are still being processed.

BFS Example



- When a node is placed on the queue, the edge from the first node in the queue (which is being taken off the queue) to that node is marked as a *tree edge* in the breadth-first tree.
- These edges actually do form a tree (called the breadth-first tree) whose root is the start node s.

BFS Example


r s t u
1 0 2 ∞
2 1 2 ∞
v w x y
Q t x v
r s t u
1 0 2 3
2 1 2 ∞
v w x y
Q x v u
r s t u
1 0 2 3
2 1 2 3
v w x y
Q v u y
r s t u
1 0 2 3
2 1 2 3
v w x y
Q u y
r s t u
1 0 2 3
2 1 2 3
v w x y
Q y
r s t u
1 0 2 3
2 1 2 3
v w x y

Running Time

Each node is visited once and each edge is examined at most twice.

Therefore the cost is $O(|V| + |E|)$.

The BFS Algorithm — Proof of Correctness

Lemma

If G is connected, then the breadth-first tree constructed by this algorithm

- *really is a tree, and*
- *contains all the nodes in the graph.*

The BFS Algorithm — Proof of Correctness

Proof.

- A node becomes the target of a tree edge when it is placed on the queue. Since that only happens once, no node is the target of two tree edges.
- Next, let us show that every node that is processed by the algorithm is reachable by a chain of tree edges from the root. It is enough to prove the following statement:
- When a node is placed on the queue, it is reachable by a chain of tree edges from the root.
- It is clearly true at the beginning: There is only one node in the queue and it is the root. The rest can be shown by induction.

The BFS Algorithm – Proof of Correctness

Proof (Cont.)

- Suppose it is true up to some point.
- When the next node v is placed on the queue, v is an endpoint of an edge whose other endpoint is the node at the head of the queue, and that edge is made a tree edge.
- By the inductive assumption, the node at the head of the queue is reachable by a path of tree edges from the root.
- Appending the new edge to the path gives a path of tree edges from the root to v.

The BFS Algorithm – Proof of Correctness

Proof (Cont.)

- Every node that is processed by the algorithm is reachable by a chain of edges from the root – so the edges form a tree.
- Suppose there was one node v that was not reached by this process.
- Since G is connected, there would have to be a path from the root to v.
- On that path there is a *first* node (w) which was not in the tree.
- That node might be v, or it might come earlier in the path.
- That means that the edge in the path leading to that node starts from a node in the tree.
- At some point, that node in the tree was at the head of the queue.
- Therefore, w would have been placed in the queue by the algorithm, and the edge to w would have been a tree edge — a contradiction.

□

Lemma

If at any point in the execution of the BFS algorithm the queue consists of the vertices $\{v_1, v_2, \ldots, v_n\}$, where v_1 is at the head of the queue, then $d[v_i] \leq d[v_{i+1}]$ for $1 \leq i \leq n-1$, and $d[v_n] \leq d[v_1| + 1$.

- In other words, the assigned depth numbers increase as one walks down the queue, and there are at most two different depths in the queue at any one time.
- If there are two, they are consecutive.

The BFS Algorithm – Proof of Correctness

Proof.

- The result is true trivially at the start of the program, since there is only one element in the queue. The rest by induction.
- At any step, a vertex is added to the tail of the queue only when it is reachable from the vertex at the head (which is being taken off).
- The depth assigned to the new vertex at the tail is 1 more than that of the vertex at the head.
- By the inductive hypothesis it is greater than or equal to the depths of any other vertex on the queue, and no more than 1 greater than any of them.

□

The BFS Algorithm – Proof of Correctness

Lemma

If two nodes in G are joined by an edge in the graph (which might or might not be a tree edge), their d values differ by at most 1.

Proof.

- Let the nodes be v and u. One of them is reached first in the breadth-first walk.
- w.l.o.g, say v is reached first. So v is put on the queue first, and reaches the head of the queue before u does. When v reaches the head of the queue, there are two possibilities:
  - u has not yet been reached. In that case, when we take v off the queue, since there is an edge from v to u, u will be put on the queue and we will have $d[u] = d[v] + 1$.
  - u has been reached and therefore is on the queue. In this case, we know from the previous lemma that $d[v] \leq d[u] \leq d[v] + 1$. □

The BFS Algorithm – Proof of Correctness

Theorem

If G is connected, then the breadth-first search tree gives the shortest path from the root to any node.

Proof.

- We know there is a path in the tree from the root to any node.
- The depth of any node in the tree is the length of the path in the tree from the root to that node.
- So for each node v in the tree, we have

  $$d[v] = \text{the length of the path in the tree from the root to } v$$

  and let us set

  $$s[v] = \text{the length of the shortest path in } G \text{ from the root to } v$$

The BFS Algorithm – Proof of Correctness

Proof (Cont.)

- We are trying to prove that $d[v] = s[v]$ for all $v \in G$.
- We know just by the definition of $s[v]$ that $s[v] \leq d[v]$ for all v.
- Suppose there is at least one node for which the theorem is not true.
- All the nodes w for which the statement of the theorem is not true satisfy $s[w] < d[w]$.
- Among all those nodes, pick one — call it v — for which $s[v]$ is smallest.

The BFS Algorithm – Proof of Correctness

Cont.

- Let u be the node preceding v on a shortest path from the root to v.
- We have

$$d[v] > s[v]$$
$$s[v] = s[u] + 1$$
$$s[u] = d[u]$$

- Hence $d[v] > s[v] = s[u] + 1 = d[u] + 1$.
- But by former lemma, this is impossible.

□

Print Shortest Path

We assume that $\mathrm{BFS}(G, s)$ has already been run, so that each node x has been assigned its depth $d[x]$.

Algorithm 2 $\mathrm{PrintPath}(G, s, v)$

```
if v = s then
    PRINT s
else
    if π[v] = NIL then
        PRINT "no path from" s "to" v "exists"
    else
        PrintPath(G, s, π[v])
        PRINT v
    end if
end if
```

The cost of this algorithm is proportional to the number of vertices in the path, so it is $O(d[v])$.