

CS624 - Analysis of Algorithms

BFS

November 5, 2025

Graphs – Basic Definitions

- Graph $G = (V, E)$
- V = set of vertices, E = set of edges $\subseteq (V \times V)$
- Undirected graph: edge $(u, v) = (v, u)$; for all v , $(v, v) \notin E$
(No self loops.)
- Directed graph: (u, v) is edge from u to v , denoted as $u \rightarrow v$.
Self loops are allowed.
- Weighted graph: each edge has an associated weight, given by a weight function $w : E \rightarrow \mathbb{R}$.
- Dense graph: $|E| \approx |V|^2$.
- Sparse: $|E| \ll |V|^2$.
- $|E| = O(|V|^2)$

Graphs – Basic Definitions

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

Graph Search Algorithms

- Searching a graph: Systematically follow the edges of a graph to visit 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)

- Let G be an undirected graph.
- One way to represent a graph is by a set *adjacency lists*, one for each vertex.
- For each vertex $v \in V$, we have a list $Adj[v]$ consisting of those vertices u such that $(v, u) \in E$.
- It is actually a set, but usually implemented as a list.
- This representation works just as well for directed graphs.
- In this case, the edge (v, u) means the edge starting from v and ending at u .
- BFS scans the graph G , starting from some arbitrary node s .
- The key mechanism in this algorithm is the use of a queue, denoted by Q .

The BFS Algorithm

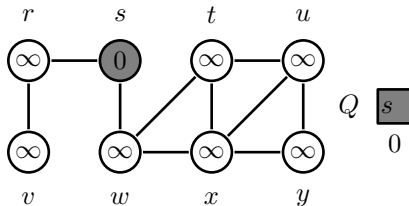
Algorithm 1 $BFS(G, s)$

```
1: for each vertex  $u \in V[G] \setminus s$  do
2:    $Color[u] \leftarrow White$ 
3:    $d[u] \leftarrow \infty$ 
4:    $\pi[u] \leftarrow nil$ 
5: end for
6:  $Color[s] \leftarrow Gray$ 
7:  $d[s] \leftarrow 0$ 
8:  $\pi[s] \leftarrow nil$ 
9:  $Q \leftarrow \emptyset$ 
10:  $Enqueue(Q, s)$ 
11: while  $Q \neq \emptyset$  do
12:    $u \leftarrow Dequeue(Q)$ 
13:   for each  $v \in Adj[u]$  do
14:     if  $Color[v] == White$  then
15:        $Color[v] \leftarrow Gray$ 
16:        $d[v] \leftarrow d[u] + 1$ 
17:        $\pi[v] \leftarrow u$ 
18:       Mark the edge from  $\pi[v]$  to  $u$  as a "tree edge".
19:        $Enqueue(Q, v)$ 
20:     end if
21:   end for
22:    $Color[u] \leftarrow Black$ 
23: end while
```

The BFS Algorithm

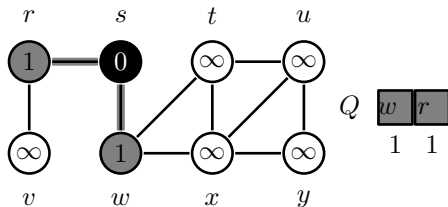
- Expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
- A vertex is “discovered” the first time it is encountered during the search.
- A vertex is “finished” if all vertices adjacent to it have been discovered.
- Colors the vertices to keep track of progress.
- White – Undiscovered.
- Gray – Discovered but not finished.
- Black – Finished.
- Colors are required only to reason about the algorithm. Can be implemented without colors.

The BFS Algorithm – Example



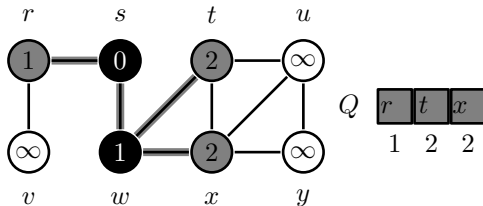
- Note that all nodes are initially colored white.
- A node is colored gray when it is placed on the queue.

The BFS Algorithm – Example



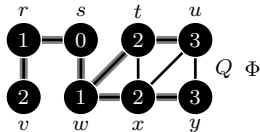
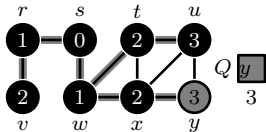
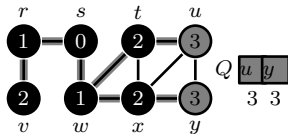
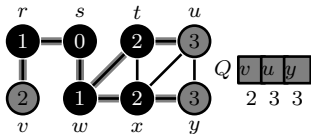
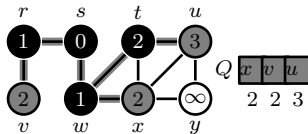
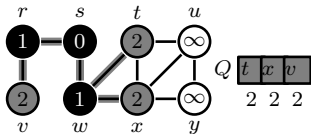
- A node is colored black when taken off the queue.
- Nodes colored white have not yet been visited. The nodes colored black are “finished” and the nodes colored gray are still being processed.

The BFS Algorithm – 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 .

The BFS Algorithm – Example



The BFS Algorithm – Runtime and Properties

- Each node is visited once and each edge is examined at most twice.
- Therefore the cost is $O(|V| + |E|)$.
- Proof of correctness:

Lemma

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

- *Really is a tree*
- *It 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

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

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.



The BFS Algorithm – Proof of Correctness

Lemma

If at any point in the execution of the BFS algorithm the queue consists of the vertices $\{v_1, v_2, \dots, v_n\}$, 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

$d[v]$ = the length of the path in the tree from the root to v

and let us set

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



The BFS Algorithm – Proof of Correctness

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 $BFS(G, s)$ has already been run, so that each node x has been assigned its depth $d[x]$.

Algorithm 2 *PrintPath*(G, s, v)

```
1: if  $v = s$  then
2:   PRINT  $s$ 
3: else
4:   if  $\pi[v] == nil$  then
5:     PRINT “no path from”  $s$  “to”  $v$  “exists”
6:   else
7:     PrintPath( $G, s, \pi[v]$ )
8:     PRINT  $v$ 
9:   end if
10: end if
```

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