

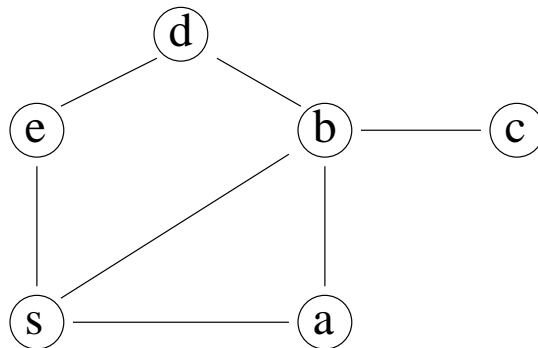
Lecture 6: Breadth-First Search

Outline of this Lecture

- The shortest path problem.
- The breadth-first search algorithm.
- The running time of BFS.
- The correctness proof.

Note: We introduce BFS for *undirected* graphs, but the same algorithm will also work for directed graphs.

Shortest Paths



Example: 3 simple paths from the source s to b :

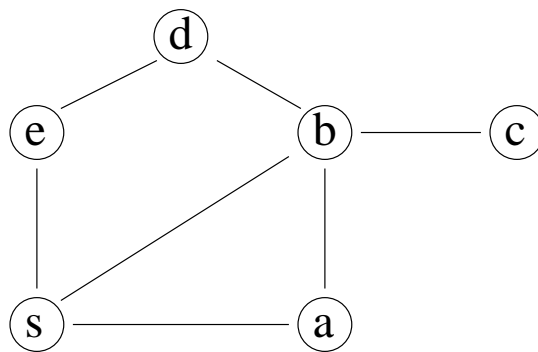
$$\langle s, b \rangle, \langle s, a, b \rangle, \langle s, e, d, b \rangle$$

of length 1, 2, 3 respectively. So the shortest path from s to b is $\langle s, b \rangle$. The shortest paths from s to other vertices are

$$\langle s, a \rangle, \langle s, b \rangle, \langle s, b, c \rangle, \langle s, b, d \rangle, \langle s, e \rangle.$$

There are two shortest paths from s to d .

The Shortest Path Problem



Distance $d[v]$: The **length** of a shortest path from s to v .
For example $d[c] = 2$. We define $d[s] = 0$.

The Problem: Given a graph $G = (V, E)$ and a **source** vertex $s \in V$, find the **distances** $d[v]$ and a **shortest path** from s to each other vertex in G .

What Does the Breadth-First Search Do

Given a graph $G = (V, E)$, the BFS returns

- the distances $d[v]$ from s to v ;
- the predecessors $\text{pred}[v]$, which is used to derive a shortest path from s to every other vertex v .

BFS is actually returning a *shortest path tree* in which the unique path from s to node u is a shortest path from s to u in the original graph.

Remarks: In addition to the two arrays $d[v]$ and $\text{pred}[v]$, the BFS also uses another auxiliary array $\text{color}[v]$, which has three possible values:

- white (W, “undiscovered”),
- gray (G, “discovered” but not “processed”),
- black (B, “discovered” and “processed”).

The Breath-First Search

The Idea of the BFS:

Visit the vertices as follows:

1. visit all vertices at distance 1
 2. visit all vertices at distance 2
 3. visit all vertices at distance 3
- etc.

Initially, s is made gray.

When a gray vertex is visited, its color is changed to black, and the color of all white neighbors is changed to gray.

Gray vertices are kept in a queue Q .

The Breath-First Search

More details.

G given by its adjacency list.

Initialization, first part:

For each vertex $u \in V$,
 $color[u] = W$; $d[u] = \infty$; $pred[u] = NIL$;

Initialization, second part:

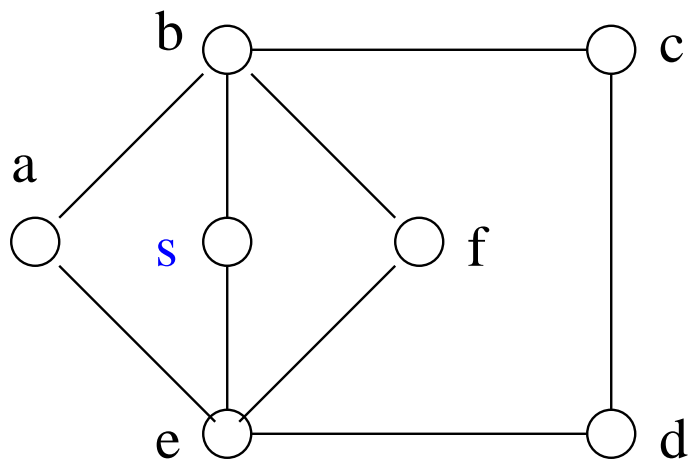
$color[s] = G$, $d[s] = 0$, $Q = \langle s \rangle$.

Main loop:

if Q is nonempty,
 $u = dequeue(Q)$
 for each $v \in adj[u]$,
 if ($color[v] == W$), do
 $color[v] = G$
 $d[v] = d[u] + 1$
 $pred[v] = u$
 put v in Q
 $color[u] = B$.

Example of the Breadth-First Search

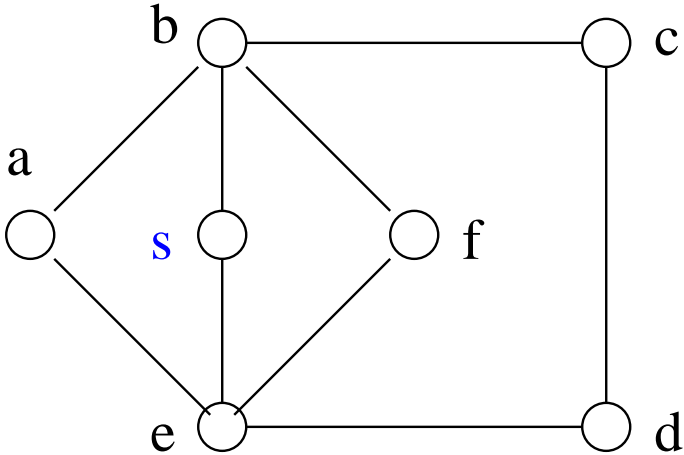
Problem: Given the following undirected graph and source vertex, find the distance from s to each vertex $u \in V$ and the predecessor $\text{pred}[u]$ along a shortest path by following the algorithm described earlier.



Example – Continued

Initialization, first part

vertex u	s	a	b	c	d	e	f
$color[u]$	W	W	W	W	W	W	W
$d[u]$	∞	∞	∞	∞	∞	∞	∞
$pred[u]$	NIL	NIL	NIL	NIL	NIL	NIL	NIL



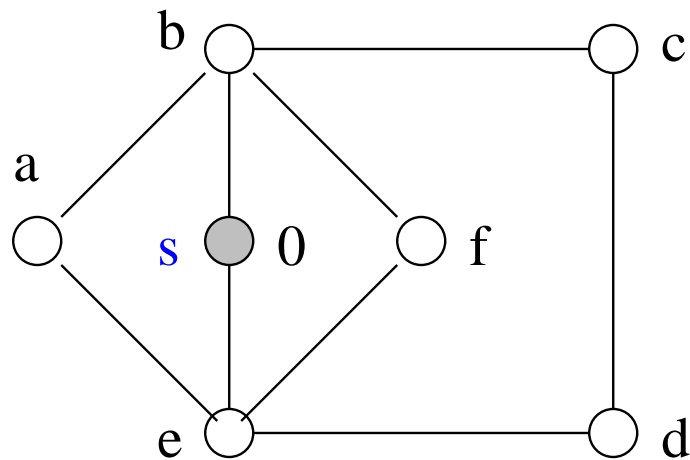
Example – Continued

Initialization, second part

vertex u	s	a	b	c	d	e	f
$color[u]$	G	W	W	W	W	W	W
$d[u]$	0	∞	∞	∞	∞	∞	∞
$pred[u]$	NIL	NIL	NIL	NIL	NIL	NIL	NIL

$Q = \langle s \rangle$

(Put s into Q (discovered) & mark “G”, meaning “unprocessed”)



Example – Continued

While loop, first iteration

Dequeue s from Q . Find $adj[s] = \langle b, e \rangle$.

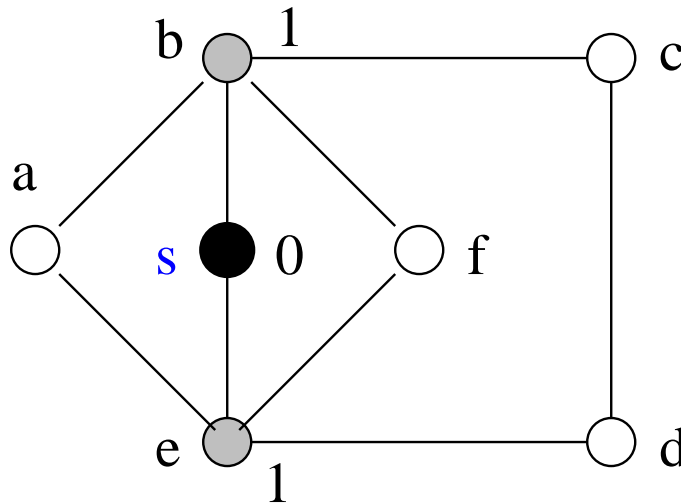
Mark b and e “G”, mark s “B”.

Update $d[b]$, $d[e]$, $pred[b]$, $pred[e]$.

Put b, e in Q .

vertex u	s	a	b	c	d	e	f
$color[u]$	B	W	G	W	W	G	W
$d[u]$	0	∞	1	∞	∞	1	∞
$pred[u]$	NIL	NIL	s	NIL	NIL	s	NIL

$Q = \langle b, e \rangle$



Example – Continued

While loop, second iteration

Dequeue b from Q . Find $adj[b] = \langle s, a, c, f \rangle$.

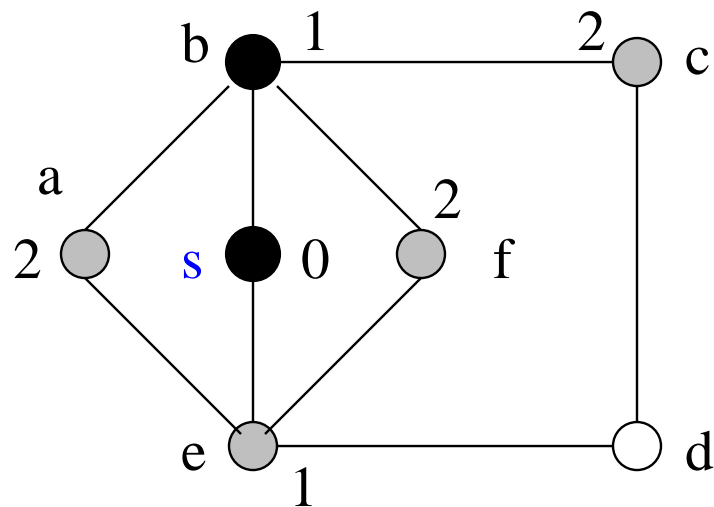
Mark a, c, f "G", mark b "B".

Update $d[a], d[c], d[f]$, $pred[a], pred[d], pred[f]$.

Put a, c, f in Q .

vertex u	s	a	b	c	d	e	f
$color[u]$	B	G	B	G	W	G	G
$d[u]$	0	2	1	2	∞	1	2
$pred[u]$	NIL	b	s	b	NIL	s	b

$Q = \langle e, a, c, f \rangle$



Example – Continued

While loop, third iteration

Dequeue e from Q . Find $adj[e] = \langle s, a, d, f \rangle$.

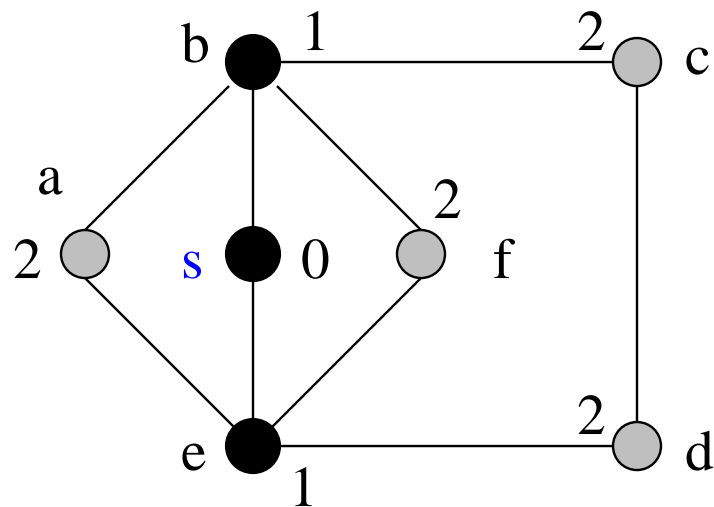
Mark d “gray”, mark e “B”.

Update $d[d]$, $pred[d]$.

Put d in Q .

vertex u	s	a	b	c	d	e	f
$color[u]$	B	G	B	G	G	B	G
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \langle a, c, f, d \rangle$



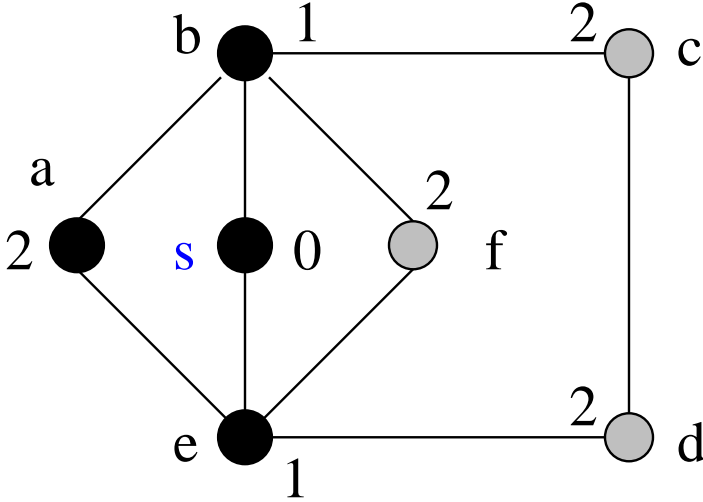
Example – Continued

While loop, forth iteration

Dequeue a from Q . Find $adj[a] = \langle b, e \rangle$.
 Mark a “B”.

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	G	G	B	G
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \langle c, f, d \rangle$



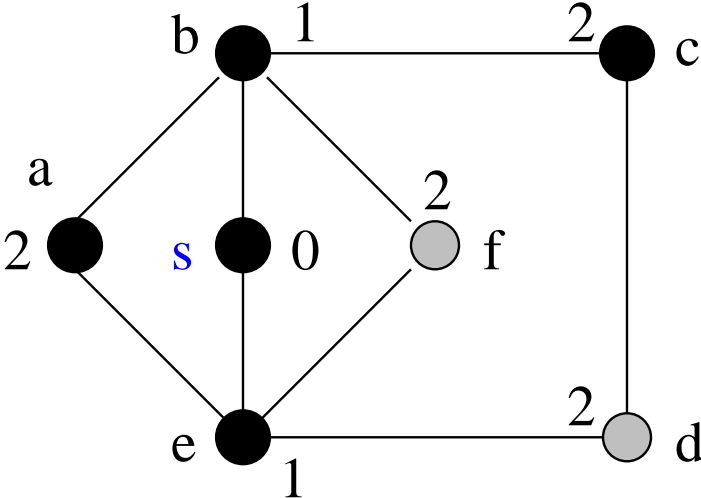
Example – Continued

While loop, fifth iteration

Dequeue c from Q . Find $adj[c] = \langle b, d \rangle$.
 Mark c "B".

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	B	G	B	G
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \langle f, d \rangle$



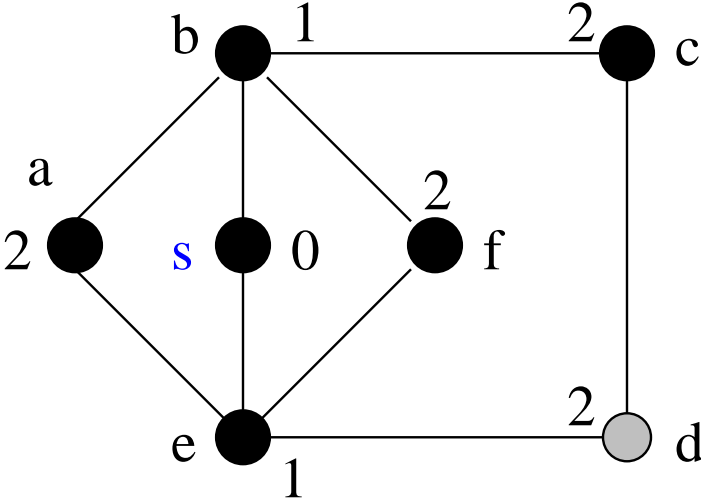
Example – Continued

While loop, sixth iteration

Dequeue f from Q . Find $adj[f] = \langle b, e \rangle$.
Mark f "B".

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	B	G	B	B
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \langle d \rangle$



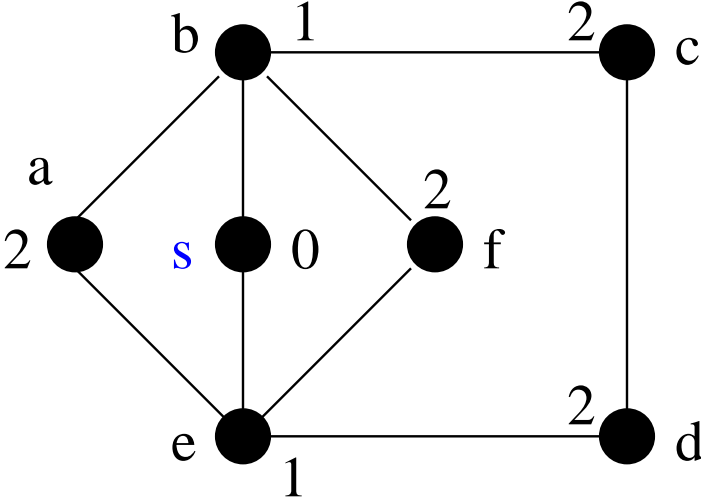
Example – Continued

While loop, seventh iteration

Dequeue d from Q . Find $adj[d] = \langle c, e \rangle$.
 Mark d “B”.

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	B	B	B	B
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \emptyset$



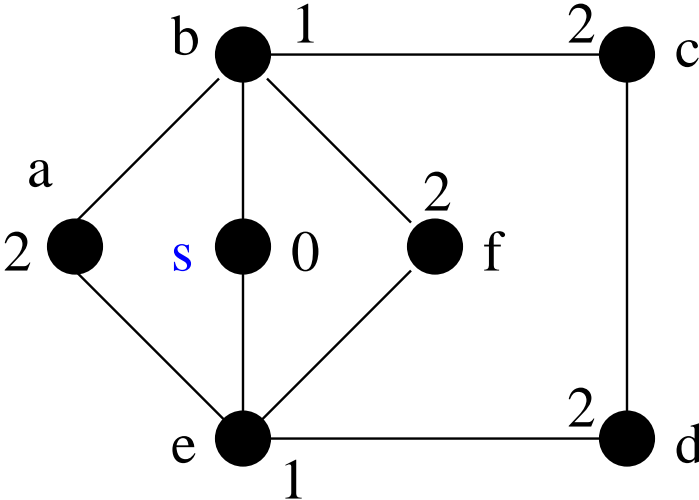
Example – Continued

While loop, eighth iteration

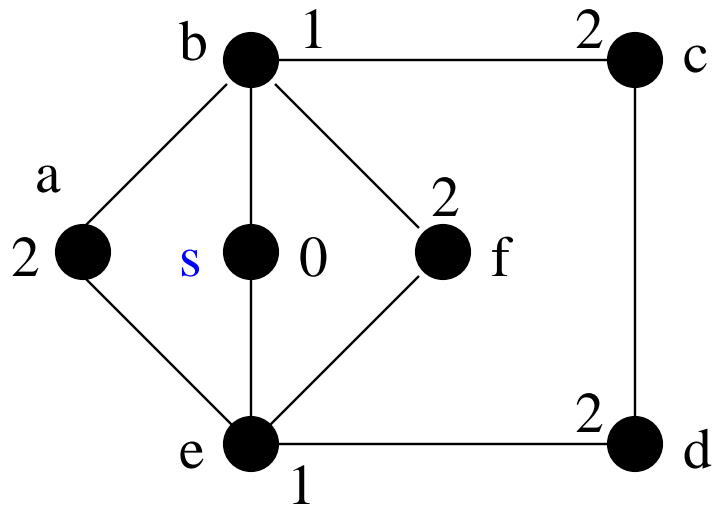
Since Q is empty, stop.

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	B	B	B	B
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

$Q = \emptyset$



Example – Continued



Question: How do you construct a shortest path from s to any vertex by using the following table?

vertex u	s	a	b	c	d	e	f
$color[u]$	B	B	B	B	B	B	B
$d[u]$	0	2	1	2	2	1	2
$pred[u]$	NIL	b	s	b	e	s	b

The Breadth-First Search Algorithm

```
for (each vertex  $u \in V$ )
{
     $color[u] = W$ ;
     $d[u] = \infty$ ;
     $pred[u] = NIL$ ;
}
 $color[s] = G$ ;  $d[s] = 0$ ; enqueue( $Q, s$ );
if ( $Q$  is nonempty)
{
     $u = dequeue(Q)$ ;
    for (each  $v \in adj[u]$ )
        if ( $color[v] == W$ )
        {
             $color[v] = G$ ;
             $d[v] = d[u] + 1$ ;
             $pred[v] = u$ ;
            enqueue( $C, v$ );
        }
     $color[u] = B$ ;
}
}
```

Analysis of the Breadth-First Search Algorithm

Let $n = |V|$ and $e = |E|$. We assume that it takes one time unit to test the color of a vertex, or to update the color of a vertex, or to compute $d[v] = d[u] + 1$, or to set $\text{pred}[v] = u$, or to enqueue, or to dequeue.

The following analysis is valid for **connected** graphs.

- The initialization requires $3n + 3$ time units.
- Each vertex u must be processed and is processed once. What is the total amount of time for processing u ?

For each $v \in \text{adj}[u]$, if $\text{color}[v] = W$, the inner loop takes 4 time units. Otherwise the inner loop will not be carried out.

$\text{dequeue}[u]$ and set $\text{color}[u] = B$ take 2 time units. Hence the total amount of time needed for processing u is at most

$$5 \deg(u) + 2$$

Hence the total amount of time needed for processing all the vertices is at most

$$\sum_{u \in V} (5 \deg(u) + 2) = 10e + 2n.$$

- Hence

$$\begin{aligned} T(n, e) &\leq (3n + 3) + (10e + 2n) \\ &= 5n + 10e + 3 = O(n + e). \end{aligned}$$

Analysis of the Breadth-First Search Algorithm

The analysis can be improved:

Each vertex is colored G **exactly once**.

Therefore, the inner loop is executed **exactly** $(n - 1)$ times.

Hence

$$\begin{aligned} T(n, e) &= (3n + 3) + 4(n - 1) + \sum_{u \in V} (\deg(u) + 2) \\ &= (3n + 3) + (4n - 4) + (2e + 2n) \\ &= 9n + 2e - 1. \end{aligned}$$

Compare to

$$T(n, e) \leq 5n + 10e + 3.$$

Remark: Note that $e \leq n(n - 1)/2$.

Since the graph is connected, $e \geq n - 1$.

If $e = \Theta(n)$, then $T(n, e) = \Theta(n)$.

Graphs that are not connected

The BFS algorithm also works for graphs that are not connected. For such graphs, only the vertices v that are in the same component as s will get a value $d[v] \neq \infty$.

In particular, we can use the array $d[]$ at the end of the computation to decide if the graph is connected.

Alternatively, we can use the array $color[]$ or the array $pred[]$. Explain why.

How is the analysis of the BFS algorithm changed if we do not assume that the graph is connected?

We can actually modify BFS so that it returns a forest. More specifically, if the original graph is composed of connected components C_1, C_2, \dots, C_k then BFS will return a tree corresponding to each C_i .

```

BFS(s) Start BFS
color[s] = G; d[s] = 0; enqueue(Q, s);
if (Q is nonempty)
{
    u = dequeue(Q);
    for (each v ∈ adj[u])
        if (color[v] == W)
        {
            color[v] = G;
            d[v] = d[u] + 1;
            pred[v] = u;
            enqueue(Q, v);
        }
    color[u] = B;
} End BFS
for (each vertex u ∈ V) Initialize
{
    color[u] = W;
    d[u] = ∞;
    pred[u] = NIL;
}
for (each vertex u ∈ V) Start Connected Component
    if d[u] ≠ ∞)
        BFS(u);

```

Correctness of the BFS Algorithm

The correctness of the BFS algorithm consists of the following two parts.

1. Prove that the BFS algorithm outputs the correct distance $d[v]$.
2. Prove that the paths obtained by using the array $\text{pred}[v]$ are the shortest.

Since the path constructed with the array $\text{pred}[v]$ has length exactly $d[v]$, we need to prove only the first part!

Correctness of the BFS Algorithm

Observations: Any vertex v in Q has a real value $d[v] \neq \infty$.

For $u, v \in Q$ at any time, if $d[u] < d[v]$ then u was discovered earlier than v and (will be processed) earlier than v .

Proof: No proof is given here. You are encouraged to come up with your own proof.

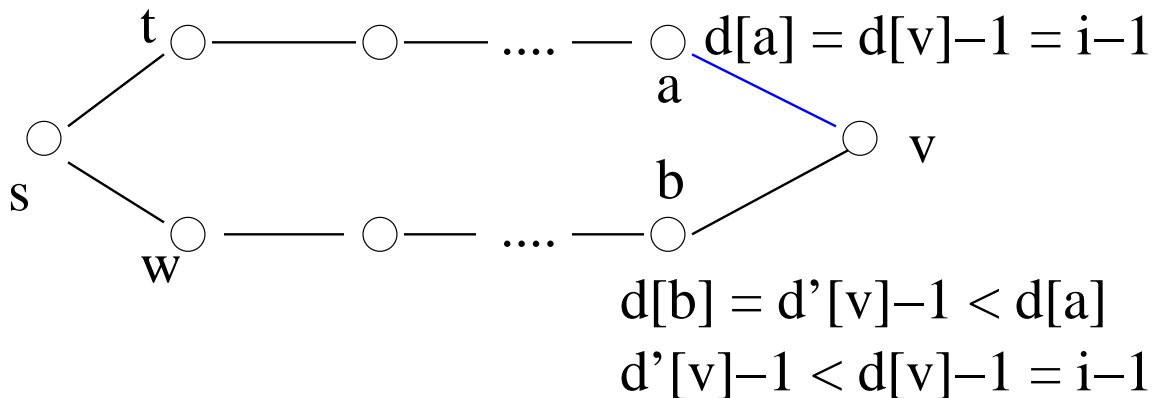
Theorem: The BFS algorithm outputs the correct distance $d[v]$.

Proof: See next page.

Correctness of the BFS Algorithm

Proof: By induction on $d[v]$. If $d[v] = 0$, then $v = s$. The conclusion is true.

Assume that $d[v]$ is the correct distance for all $d[v] < i$. Consider the case $d[v] = i$. If $d[v]$ were not the correct distance, then the true distance $d'[v] < d[v]$. We then have two paths:



- (1) $st\dots a$ must be a shortest path by induction hypothesis as $d[a] = d[v] - 1 = i - 1 < i$
- (2) $sw\dots b$ must be a shortest path as it is a subpath of the shortest path $sw\dots bv$
- (3) a distinct from b because $d[b] < d[a]$, while both $d[a]$ and $d[b]$ are true distance

Since $d[b] < d[a]$, b should be processed earlier than a , and should discover v . This is contrary to that a discovered v .