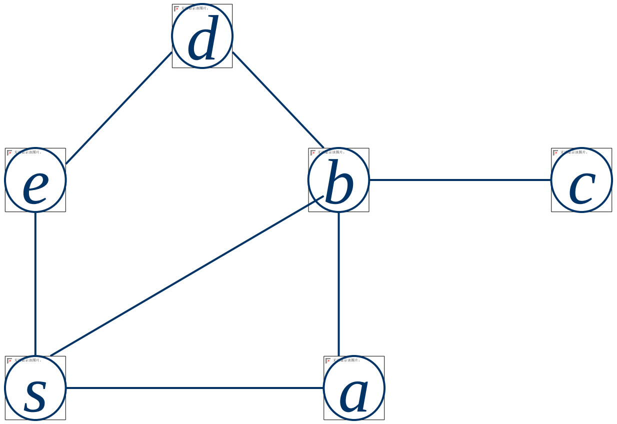
Lecture 234
Breadth-First Search

Breadth-first search (outline)

1. The single source shortest-paths problem for unweighted graph
2. The Breadth-first search algorithm
3. The correctness proof
4. The running time of BFS algorithm

- Note: *We only introduce BFS for undirected graphs. But it also works for directed graphs.*

Shortest paths



Example: There are three simple paths (distinct vertices) from source *s* to *b*: <*s*, *b*>, <*s*, *a*, *b*>, <*s*, *e*, *d*, *b*> of length 1, 2, 3, respectively.

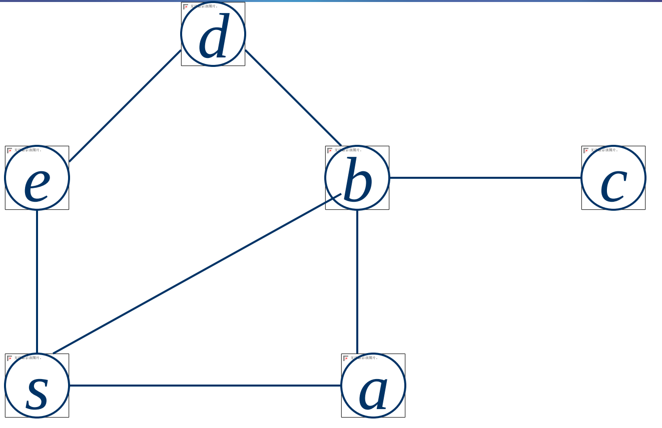
So the shortest path (a path containing the smallest number of edges) from *s* to *b* is <*s*, *b*>.

The shortest paths from *s* to other vertices *a*, *e*, *d*, *c* are:

<*s*, *a*>, <*s*, *e*>, <*s*, *e*, *d*>, <*s*, *b*, *c*>.

There are two shortest paths from *s* to *d*.

The shortest-paths problem



- Distance $\delta(s, v)$: The length of the shortest path from s to v. For example $\delta(s, c)=2$.
- The problem:
  - Input: A graph $G = (V, E)$ and a source vertex $s \in V$
  - Question: A shortest path from s to each vertex $v \in V$ and the distance $\delta(s, v)$.

The Breadth-First Search

- The idea of the BFS:
  - Each time, search as many vertices as possible.
  - 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
    4. … …
- Initially, *s* is made GRAY, others are colored WHITE.
- After a gray vertex is processed, **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 Breadth-First Search (more details)

- *G* is given by its adjacency-lists.
- Initialization:
  - First Part: lines 1 – 4
  - Second Part: lines 5 - 9
- Main Part: lines 10 – 18

- Enqueue(*Q*, *v*): add a vertex *v* to the end of the queue *Q*
- Dequeue(*Q*): Extract the first vertex in the queue *Q*

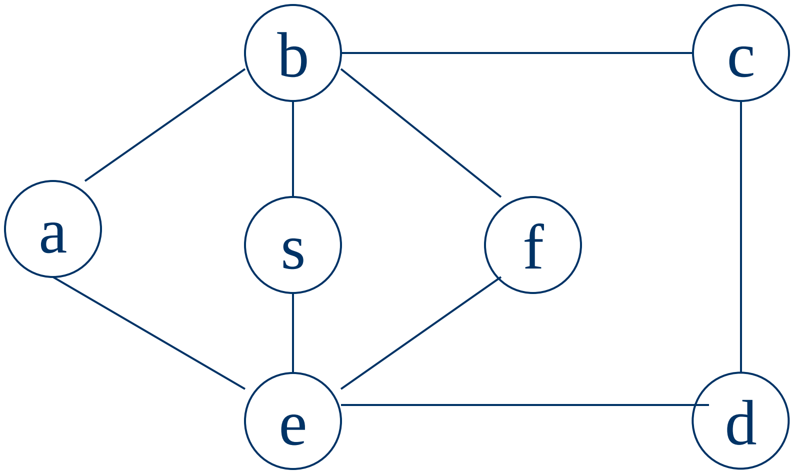
```
BFS(G, s)
1   for each vertex u∈V[G]-{s}
2          do u.color<--WHITE
3               u.d<--∞
4               u.π<--NIL
5    s.color<--GRAY
6    s.d<--0
7    s.π<--NIL
8    Q<--ϕ
9     ENQUEUE(Q, s)
10   while Q≠ϕ
11       do u<--DEQUEUE(Q)
12            for each v∈Adj[u]
13                do if v.color=WHITE
14                    then v.color<--GRAY
15                         v.d<--u.d+1
16                         v.π<--u
17                         ENQUEUE(Q, v)
18           u.color<--BLACK
```

What does the BFS do?

- Given a graph $G = (V, E)$, the BFS returns:
  - $v.d$, proved to be the shortest distance from s to v
  - $v.\pi$, the predecessor of v in the search, which can be used to derive a shortest path from s to vertex v.
- BFS actually returns a **shortest path tree** in which the unique simple path from s to node v is a shortest path from s to v in the original graph.
- In addition to the two arrays $v.d$ and $v.\pi$, BFS also uses another array *v.color*, which has three possible values:
  - WHITE: represented "undiscovered" vertices;
  - GRAY: represented "discovered" but not "processed" vertices;
  - BLACK: represented "processed" vertices.

Example of Breadth-First Search

source vertex: *s*

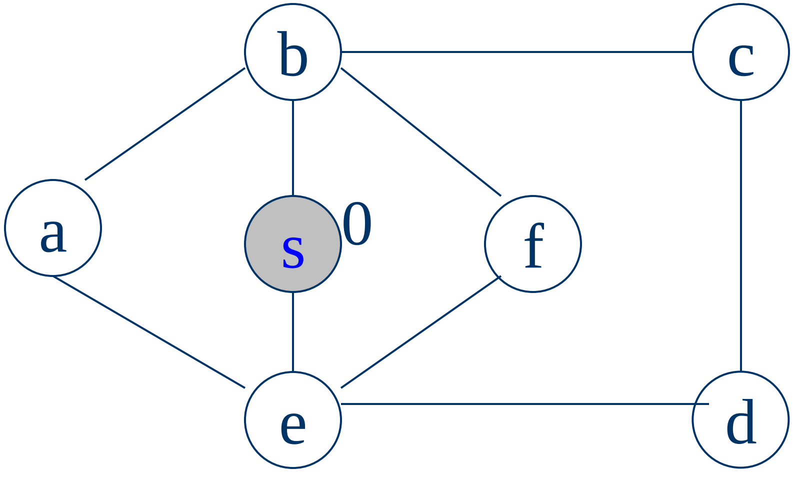


Example(Continued)

- Initialization

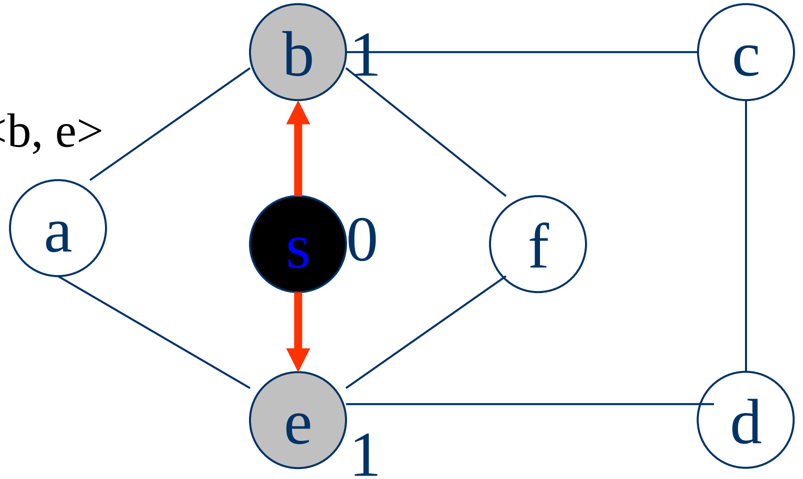
vertex u	s	a	b	c	d	e	f
color	G	W	W	W	W	W	W
d	0	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Q = <s>
(put s into Q (discovered),
mark s gray (unprocessed))



Example(continued)

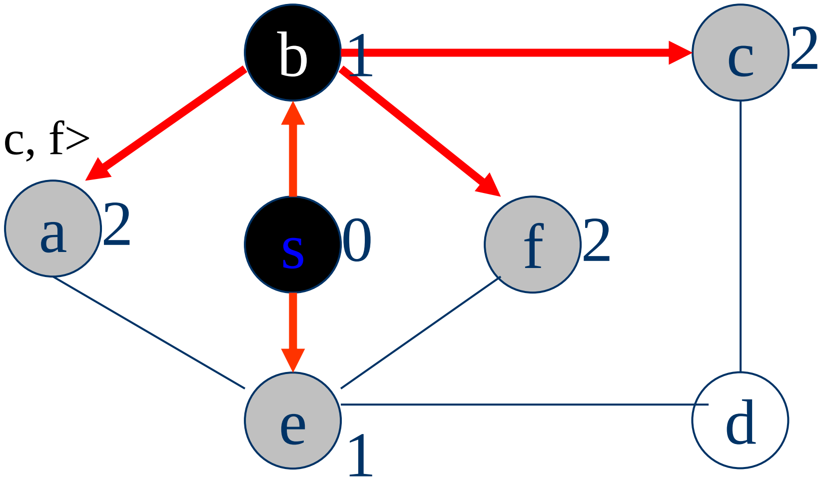
- While loop, first iteration
  - Dequeue s from Q. Find Adj[s]=<b, e>
  - Mark b,e as "G"
  - Update b.d, e.d, b.π, e.π
  - Put b, e into Q
  - Mark s as "B"
- Q=<b, e>



vertex u	s	a	b	c	d	e	f
color	B	W	G	W	W	G	W
d	0	∞	1	∞	∞	1	∞
π	NIL	NIL	s	NIL	NIL	s	NIL

Example(continued)

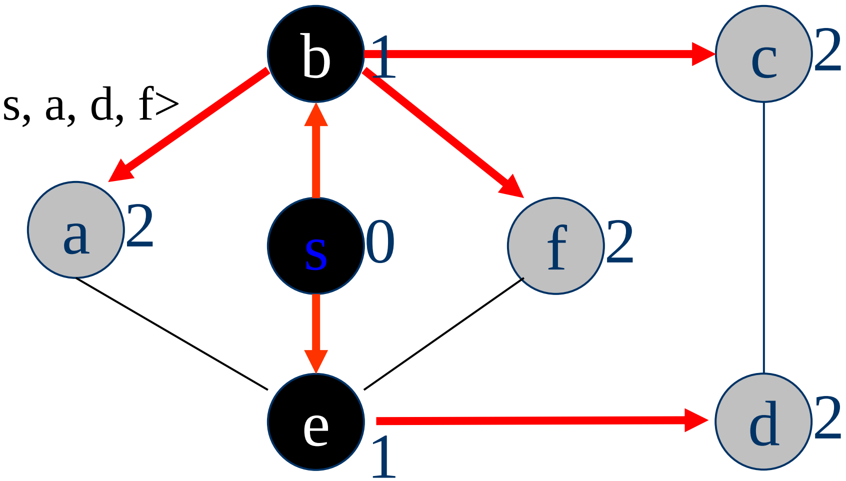
- While loop, second iteration
  - Dequeue b from Q, find Adj[b]=<s, a, c, f>
  - Mark a, c, f as “G”,
  - Update a.d, c.d, f.d, a.π, c.π,f.π
  - Put a, c, f into Q
  - Mark b as “B”
- Q=<e, a, c, f>



vertex u	s	a	b	c	d	e	f
color	B	G	B	G	W	G	G
d	0	2	1	2	∞	1	2
π	NIL	b	s	b	NIL	s	b

Example(continued)

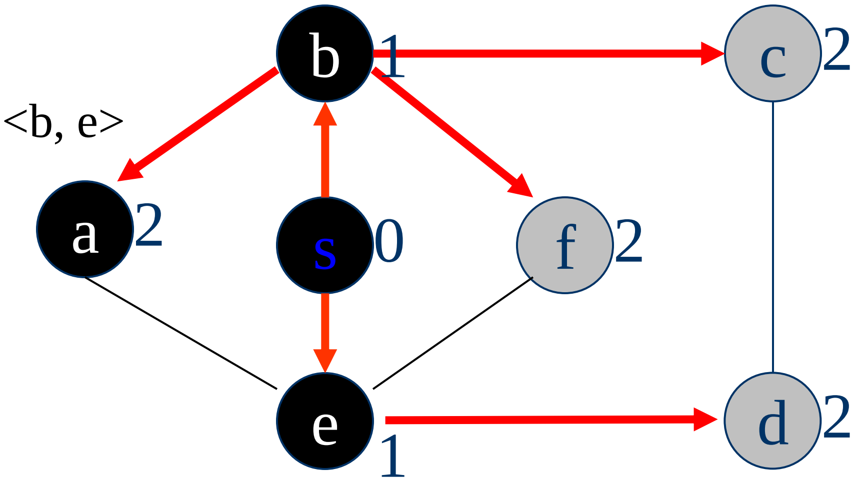
- While loop, third iteration
  - Dequeue e from Q, find Adj[e]=<s, a, d, f>
  - Mark d as “G”, mark e as “B”
  - Update d.d, d.π,
  - Put d into Q
- Q=<a,c,f,d>



vertex u	S	a	b	c	d	e	f
color	B	G	B	G	G	B	G
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

Example(continued)

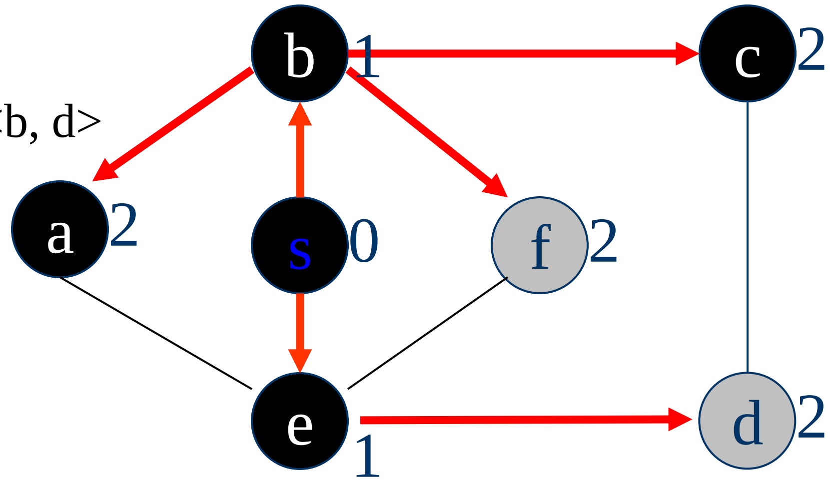
- While loop, fourth iteration
  - Dequeue a from Q, find Adj[a]=<b, e>
  - mark a as “B”
- Q=<c, f, d>



vertex u	s	a	b	c	d	e	f
color	B	B	B	G	G	B	G
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

Example(continued)

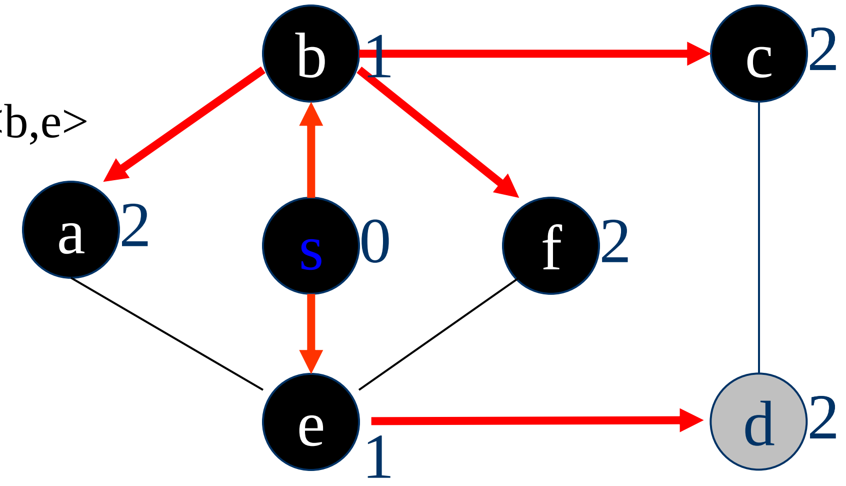
- While loop, fifth iteration
  - Dequeue c from Q, find Adj[c]=<b, d>
  - mark c as “B”
- Q=<f, d>



vertex u	s	a	b	c	d	e	f
color	B	B	B	B	G	B	G
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

Example(continued)

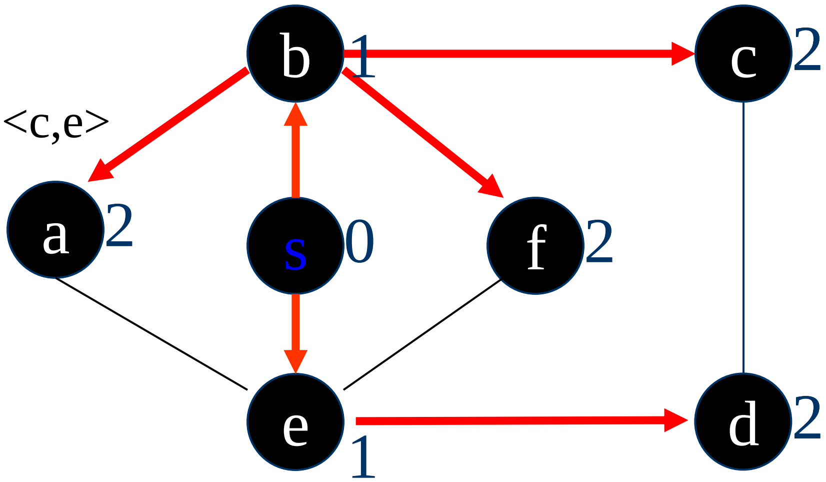
- While loop, sixth iteration
  - Dequeue f from Q, find Adj[f]=<b,e>
  - mark f as "B"
- Q=<d>



vertex u	s	a	b	c	d	e	f
color	B	B	B	B	G	B	B
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

Example(continued)

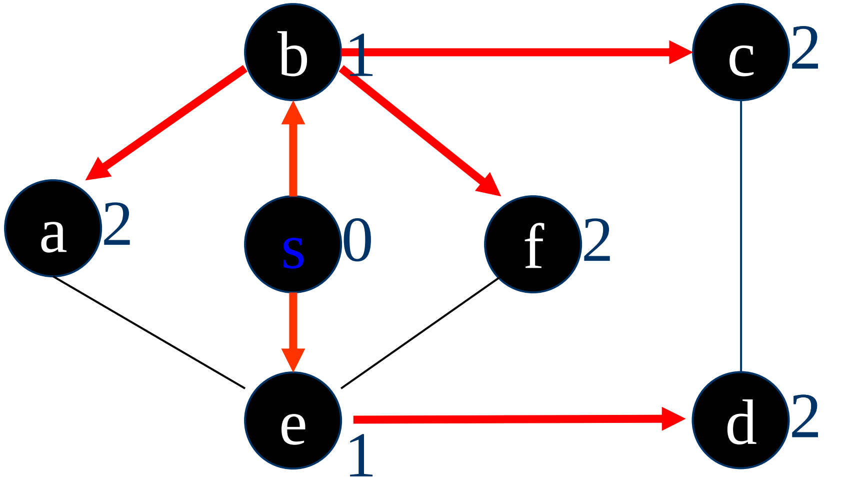
- While loop, seventh iteration
  - Dequeque d from Q, find Adj[d]=<c,e>
  - mark d as “B”
- Q is empty



vertex u	s	a	b	c	d	e	f
color	B	B	B	B	B	B	B
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

Example(continued)

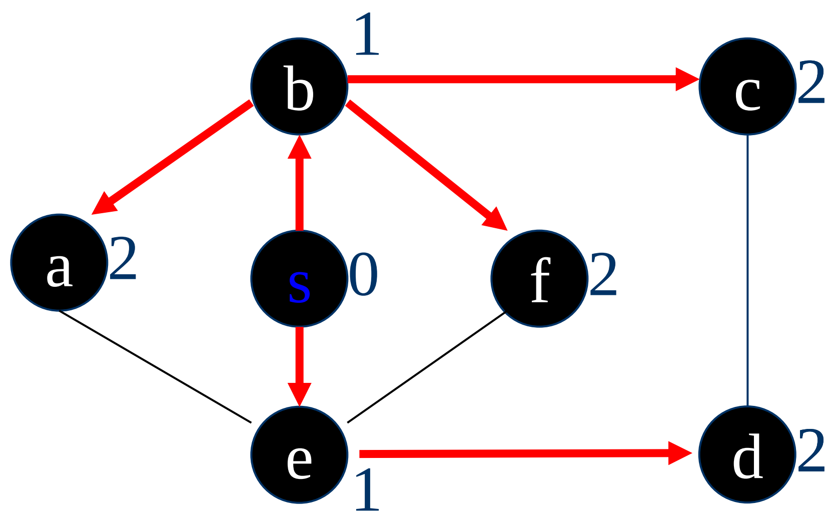
- While loop, eighth iteration
- Since Q is empty, stop



vertex u	s	a	b	c	d	e	f
color	B	B	B	B	B	B	B
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

What does BFS produce

- BFS creates a prodecessor subgraph $G_\pi = (V_\pi, E_\pi)$, where
  - $V_\pi = \{u \in V \mid u.\pi \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(u.\pi, u) \mid$ where $u.\pi$ is computed in the BFS(G, s) calls$\}$
  - prodecessor subgraph is a tree since it is connect and $|E_\pi| = |V_\pi| - 1$



The red edges form a tree   //note there are |V|-1 edges

Our Goal

- Shortest-path distance $\delta(s, v)$:
  - the minimum number of edges in any path from vertex s to vertex v; if there is no path from s to v, then $\delta(s, v)=\infty$.
- Shortest path:
  - A path of length $\delta(s, v)$: $s \to v$.

Recall what does the BFS return?

- The BFS returns:
  - $v.d$, proved to be the shortest distance from s to v
  - $v.\pi$, the predecessor of v in the search, which can be used to derive a shortest path from s to vertex v.
- What we need to prove:
  - $\delta(s, v) = v.d$
  - $s \rightarrow v = s \rightarrow v.\pi + (v.\pi, v)$

An Important Property

- **Lemma 22.1**: Let $G=(V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u)+1$.
- Proof:
  - If u is reachable from s, then so is v. In this case, the shortest path from s to v can not be longer than the shortest path from s to u followed by the edge (u, v) and thus the inequality holds.
  - If u is not reachable from s, then $\delta(s, u)=\infty$, and the inequality holds.

A Property of the BFS

- **Lemma 22.2:**
- Let $G=(V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$.
- Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof of Lemma 22.2(1)

- We use induction on the number of ENQUEUE operations. Our inductive hypothesis is that $v.d \geq \delta(s, v)$ for all $v \in V$.
- The basis of the induction is the situation immediately after enqueuing s in line 9 of BFS. The inductive hypothesis holds here, because $s.d = 0 = \delta(s, s)$ and

  $v.d = \infty \geq \delta(s, v)$, $v \in V - \{s\}$.
- For the inductive step, consider a white vertex v that is discovered during the search from a vertex u.

Proof of Lemma 22.2(2)

- We obtain

$$\begin{aligned} v.d &= u.d + 1 \quad \text{(line 15)} \\ &\geq \delta(s, u) + 1 \quad \text{(inductive hypothesis)} \\ &\geq \delta(s, v) \quad \text{(Lemma 22.1)} \end{aligned}$$

- Vertex *v* is then enqueued, and it is never enqueued again because it is also grayed and the **then** clause of lines 14–17 is executed only once for white vertices.
- Thus, the value of *v.d* never changes again, and the inductive hypothesis is maintained.

A property of the Queue

- **Lemma 22.3:**
- Suppose that during the execution of BFS on a graph G=(V, E) , the queue Q=<v_1, v_2, …, v_r >, where v_1 is the head of Q and v_r is the tail.
- Then $v_r.d \leq v_1.d$ +1 and $v_i.d \leq v_{i+1}.d$, for $1 \leq i \leq r$-1.

Proof of Lemma 22.3(1)

- The proof is by induction on the number of queue operations. Initially, when the queue contains only s, the lemma certainly holds.
- For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex. If the head v_1 of the queue is dequeued, v_2 becomes the new head. (If the queue becomes empty, then the lemma holds vacuously.)
- By the inductive hypothesis, $v_1.d \le v_2.d$,

  $v_r.d \le v_1.d+1 \le v_2.d+1$, $v_i.d \le v_{i+1}.d$, $2 \le i \le r-1$.
- Thus, the lemma follows with v_2 as the head.

Proof of Lemma 22.3(2)

- In order to understand what happens upon enqueuing a vertex, we need to examine the code more closely. When we enqueue a vertex v in line 17 of BFS, it becomes v_{r+1} . At that time, we have already removed vertex u, whose adjacency list is currently being scanned, from the queue Q.
- By the inductive hypothesis, the new head v_1 has $v_1.d \geq u.d$. Thus, $v_{r+1}.d = v.d = u.d+1 \leq v_1.d+1$.
- From the inductive hypothesis, we also have $v_r.d \leq u.d + 1$,

  or from line 15 we have $v_r.d = u.d+1$,

  and so $v_r.d \leq u.d + 1 = v.d = d[v_{r+1}]$.
- The remaining inequalities are unaffected. Thus, the lemma follows when v is enqueued.

Lemma 22.3中文证明思路

- Proof:
  - 对入队和出队操作次数进行归纳证明。我们证明入队或出队操作之后，引理总成立。
  - 当Q=<s>时, 引理中的性质成立。
  - v_1出队，v_2成为队首。则依据归纳假设

    $d[v_1] \le d[v_2]$, $d[v_r] \le d[v_1]+1 \le d[v_2]+1$, $d[v_i] \le d[v_{i+1}]$, $2 \le i \le r-1$。
  - v入队，成为队尾v_{r+1}。之前存在u出队，此时算法正在搜索u的邻接链表, 发现白色顶点v。

    $d[v_1] \ge d[u]$，$d[v_r] \le d[u]+1$ （归纳假设）

    $d[v_{r+1}] = d[v] = d[u]+1$ （算法第15行）

    $d[v_{r+1}] = d[u]+1 \le d[v_1]+1$; $d[v_r] \le d[u]+1 = d[v_{r+1}]$。

A Corollary（推论）

- **Corollary 22.4:**
- Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j.

- Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

- v_i先于v_k入队，则 $d[v_i] \leq d[v_k]$ （先入队的距离小）

- **Proof:** Immediate from Lemma 22.3 and the property that each vertex receives a finite d value at most once during the course of BFS.

Correctness of BFS (Theorem 22.5)

- Let $G=(V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$.
- Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = \delta(s, v)$, for all $v \in V$.
- Moreover, for any vertex $v \neq$ s that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

Proof of Theorem 22.5(1)

- Assume, for the purpose of contradiction, that some vertex receives a d value not equal to its shortest-path distance. Let v be the vertex with minimum $\delta(s, v)$, that receives such an incorrect d value; clearly $v \neq s$. By Lemma 22.2, $v.d \geq \delta(s, v)$, and thus we have that $v.d > \delta(s, v)$ .Vertex v must be reachable from s, for if it is not, then $\delta(s, v) = \infty \geq v.d$.
- Let u be the vertex immediately preceding v on a shortest path from s to v , so that $\delta(s, v) = \delta(s, u)+1$. Because $\delta(s, u) < \delta(s, v)$, and because of how we chose v, we have $u.d = \delta(s, u)$.
- Putting these properties together, we have

$$v.d > \delta(s, v) = \delta(s, u)+1 = u.d + 1 \quad \text{--------------} \quad (22.1).$$

Proof of Theorem 22.5(2)

- Now consider the time when BFS chooses to dequeue vertex *u* from Q in line 11. At this time, vertex *v* is either white, gray, or black. We shall show that in each of these cases, we derive a contradiction to inequality (22.1).
- If *v* is white, then line 15 sets $v.d = u.d + 1$, contradicting inequality (22.1).
- If *v* is black, then it was already removed from the queue and, by Corollary 22.4, we have $v.d \le u.d$, a contradiction.
- If *v* is gray, then it was painted gray upon dequeuing some vertex *w*, which was removed from Q earlier than *u* and for which : $v.d = w.d+1$. By Corollary 22.4, $w.d \le u.d$, and so we have: $v.d \le u.d + 1$, once again , a contradiction .

Proof of Theorem 22.5(3)

- Thus we conclude that : $v.d = \delta(s, v)$, for all $v \in V$.

  All vertices reachable from s must be discovered, for otherwise, they would have $\infty = v.d \geq \delta(s, v)$.
- To conclude the proof of the theorem, observe that if $v.\pi = u$, then $v.d = u.d + 1$.
- Thus, we can obtain a shortest path from s to v by taking a shortest path from s to $v.\pi$: and then traversing the edge $(v.\pi, v)$.

Correctness of BFS中文版（1）

- 定理22.5： $\delta(s, v)=d[v]$

  $s \to v = s \to \pi[v] + (\pi[v], v)$

- 证明： （part 1）
  - 反证法。假设存在一些顶点，它们的$d[]$值不等于最短路径距离。设v是这些顶点中，最短路径距离最小的。（$v \neq s$）
  - $d[v] \geq \delta(s, v)$ （引理22.2），因此$d[v] > \delta(s, v)$ 。v和s之间 必然存在一条最短路径 $s \to v$， 否则 $\delta(s, v) = \infty \geq d[v]$ 。设u是这条最短路径上v的前驱，即$s \to u — v$。因此, $\delta(s, v) = \delta(s, u) +1$。这意味着$\delta(s, u) < \delta(s, v)$, 根据v的选择方式， $d[u] = \delta(s, u)$

    $d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$。
  - 接下来分析当u出队时，v可能的三种颜色：W，G，B。

    我们证明每一种情况下,上式都不可能成立。

Correctness of BFS中文版（2）

- 定理22.5：$\delta(s, v) = d[v]$

  $s \to v = s \to \pi[v] + (\pi[v], v)$

- 证明：（part 2）
  - v是白色的, $d[v] = d[u] + 1$。(算法15行)
  - v是黑色的, v已经出队, $d[v] \le d[u]$ 。(推论22.4)
  - v是灰色的, v在顶点w的邻接链表里, w先于u出队.

    $d[v] = d[w]+1$, $d[w] \le d[u]$, 从而 $d[v] \le d[u] + 1$。
  - 综上，假设的v是不可能存在的。因此$d[v] = \delta(s, v)$.
  - 所有s不可达的顶点的$d[]$值都是∞。

  （part 3）
  - 如果$\pi[v]=u$，则$d[v] = d[u] + 1$，$s \to u + (u, v)$ 是s到v的最短路径。

Predecessor subgraph

- BFS creates a predecessor subgraph $G_\pi = (V_\pi, E_\pi)$, where
  - $V_\pi = \{u \in V |\ u.\pi \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(u.\pi, u)|$ where $u.\pi$ is computed in the BFS(G, s) calls}
- The predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a breadth-first tree if
  - V_π consists of vertices reachable from s。
  - For all $v \in V_\pi$, there is a unique simple path form s to v in G_π that is also a shortest path from s to v in G.
- The edges in E_π are called tree edges (树边)。

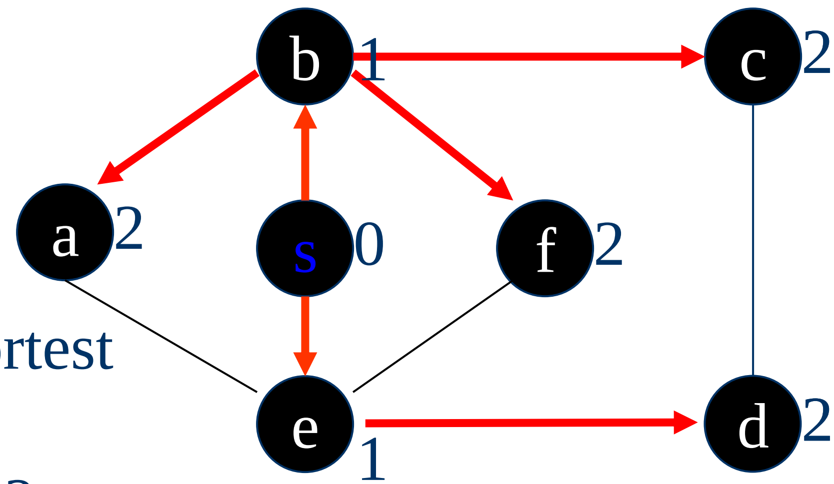
Breadth-first Tree (Lemma 22.6)

- When applied to a directed or undirected graph G $G=(V, E)$, procedure BFS constructs π, so that the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$, is a breadth-first tree.

Proof of Lemma 22.6

- Line 16 of BFS sets $v.\pi = u$ if and only if $(u, v) \in E$ and $\delta(s, v) < \infty$ — that is, if v is reachable from s— and thus V_π consists of the vertices in V reachable from s.
- Since G_π forms a tree, it contains a unique simple path from s to each vertex in V_π.
- By applying Theorem 22.5 inductively, we conclude that every such path is a shortest path in G.

Example of Breadth-first Tree



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	B	B	B	B	B	B	B
d	0	2	1	2	2	1	2
π	NIL	b	s	b	e	s	b

The Answer

PRINT_PATH(G,s,v)

```
if v=s
    then print s
    else if v.π=NIL;
            then print "no path from"s "to" v;
            else PRINT_PATH(G,s, v.π)
                    print v
```

BFS for disconnected graph

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

BFS for disconnected graph

```
BFS_GENERAL(G)
for each vertex u ∈ V
    u.color ← WHITE;
    u.d ← ∞;
    u.π ← NIL;
for each vertex u ∈ V
    if u.d = ∞
        then BFS(G, u);
```

Analysis of the Breadth-First Search Algorithm

- We assume that it takes one unit time to test the color of a vertex, or to update the color of a vertex, or to compute $v.d = u.d + 1$, or to set $v.\pi = u$, or to en-queue, or to de-queue.
- The following analysis is valid for connected graphs.
  - The initialization requires $O(V)$ time units.
  - In the while loop, each vertex u is en-queued and de-queued **exactly once**. So, each adjacency list Adj[u] is scanned **exactly once** after u is de-queued. So scanning the adjacency lists needs time $O(E)$.
- Total time is: $T(V, E) = O(V+E)$.

Conclusion

- Content
  - BFS：algorithm，proof of its correctness，cases，analysis of its time complexity
- Homework
  - 22.2-2 and 22.2-8

Nest Class

- The DFS algorithm
- The time complexity of DFS algorithm
- Properties of the DFS algorithm