Elementary Graph Algorithms

謝仁偉 副教授
jenwei@mail.ntust.edu.tw
國立台灣科技大學 資訊工程系
2017 Spring

Outline

- ***Representations of Graphs***
- Breadth-First Search
- Depth-First Search
- Topological Sort
- Strongly Connected Components

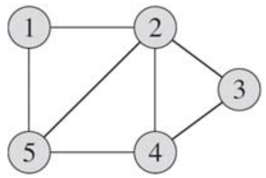
Representations of Graphs

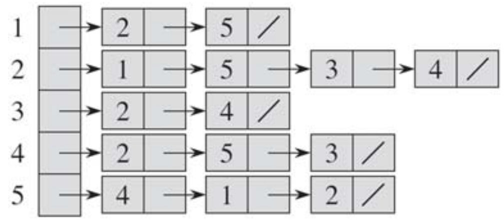
- We can choose between two standard ways to represent a graph $G = (V, E)$: as a collection of adjacency lists or as an adjacency matrix.
  - Either way applies to both directed and undirected graphs.
  - When the graph is sparse (those for which $|E|$ is much less than $|V|^2$) ➡ adjacency lists
  - When the graph is dense ($|E|$ is close to $|V|^2$) or when we need to be able to tell quickly if there is an edge connecting two given vertices ➡ adjacency matrix

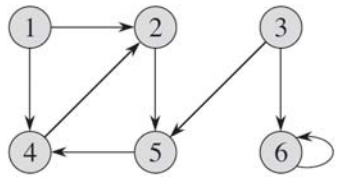
Adjacency-List Representation (1/2)

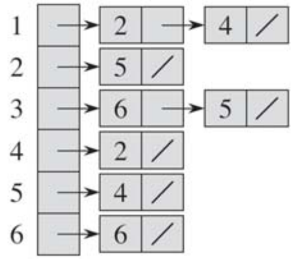
- The adjacency-list representation of a graph $G = (V, E)$ consists of an array Adj of $|V|$ lists, one for each vertex in V.
  - For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices v such that there is an edge $(u, v) \in E$.
  - If G is a directed graph, the sum of the lengths of all the adjacency lists is $|E|$, since an edge of the form (u, v) is represented by having v appear in $Adj[u]$.
  - If G is an undirected graph, the sum of the lengths of all the adjacency lists is $2|E|$, since if (u, v) is an undirected edge, then u appears in v's adjacency list and vice versa.

Adjacency-List Representation (2/2)









Weighted Graphs

- We can readily adapt adjacency lists to represent weighted graphs, that is, graphs for which each edge has an associated weight, typically given by a weight function $w : E \rightarrow \mathbb{R}$.
- We simply store the weight $w(u, v)$ of the edge $(u, v) \in E$ with vertex v in u's adjacency list.

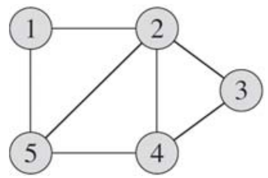
Adjacency-Matrix Representation (1/2)

- The adjacency-matrix representation of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

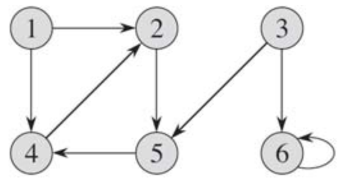
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E , \\ 0 & \text{otherwise} . \end{cases}$$

- If $G = (V, E)$ is a weighted graph with edge weight function w, we can simply store the weight $w(u, v)$ of the edge $(u, v) \in E$ as the entry in row u and column v of the adjacency matrix.

Adjacency-Matrix Representation (2/2)



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Outline

- Representations of Graphs
- ***Breadth-First Search***
- Depth-First Search
- Topological Sort
- Strongly Connected Components

Breadth-First Search (1/3)

- Given a graph $G = (V, E)$ and a distinguished source vertex s, breadth-first search systematically explores the edges of G to "discover" every vertex that is reachable from s.
  - It computes the distance (smallest number of edges) from s to each reachable vertex.
  - It also produces a "breadth-first tree" with root s that contains all reachable vertices.
- The algorithm discovers all vertices at distance k from s before discovering any vertices at distance $k + 1$.

Breadth-First Search (2/3)

- To keep track of progress, breadth-first search colors each vertex white, gray, or black.
  - All vertices start out white and may later become gray and then black.
  - A vertex is discovered the first time it is encountered during the search, at which time it becomes nonwhite.
  - Gray and black vertices, therefore, have been discovered, but breadth-first search distinguishes between them to ensure that the search proceeds in a breadth-first manner.

Breadth-First Search (3/3)

- If $(u, v) \in E$ and vertex u is black, then vertex v is either gray or black; that is, all vertices adjacent to black vertices have been discovered.
- Gray vertices may have some adjacent white vertices; they represent the frontier between discovered and undiscovered vertices.

Breadth-First Tree

- Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex s.
- Whenever the search discovers a white vertex v in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree.
  - We say that u is the predecessor or parent of v in the breadth-first tree.

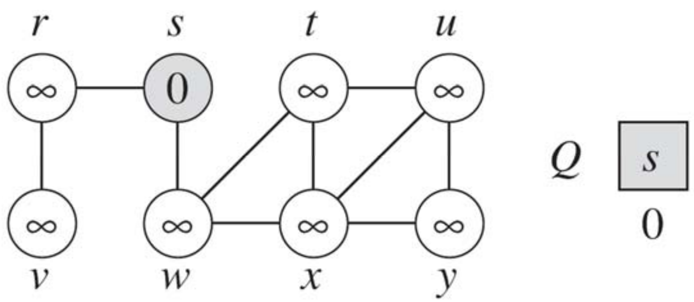

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

The distance from the source s to vertex u

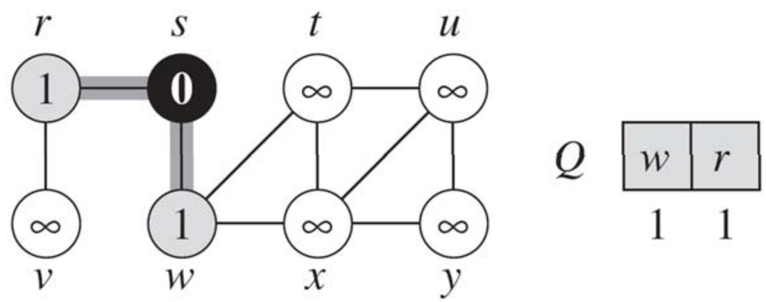
The predecessor of u
NIL: no predecessor (root or has not been discovered)

A first-in, first-out queue Q to manage the set of gray vertices

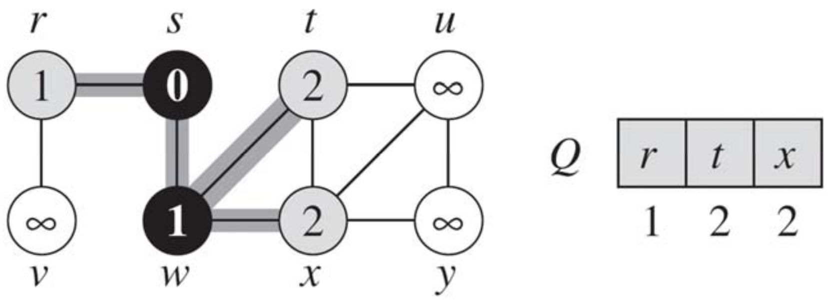
The Operation of BFS on an Undirected Graph (1/9)



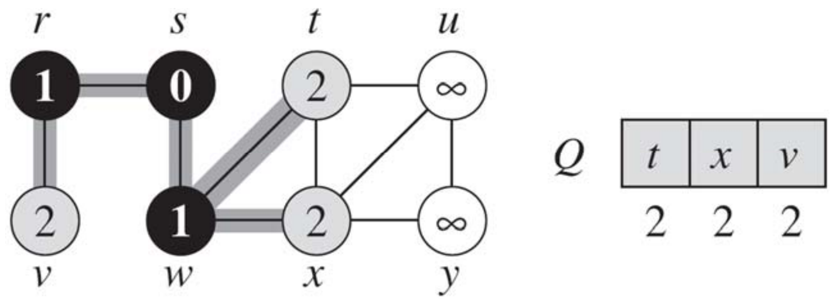
The Operation of BFS on an Undirected Graph (2/9)



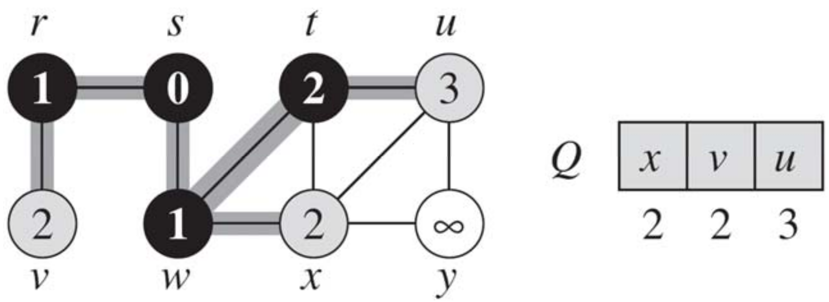
The Operation of BFS on an Undirected Graph (3/9)



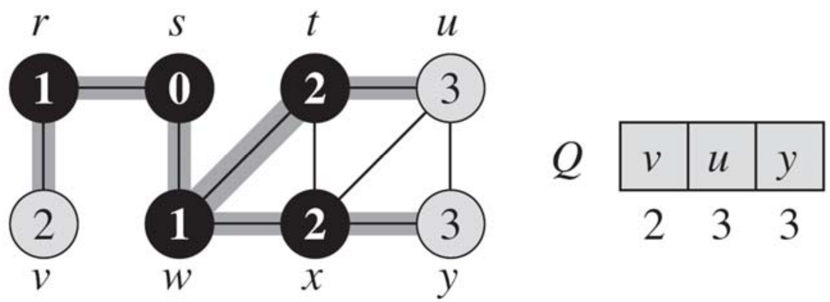
The Operation of BFS on an Undirected Graph (4/9)



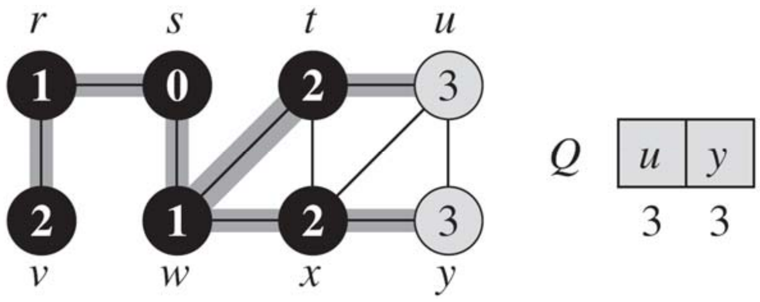
The Operation of BFS on an Undirected Graph (5/9)



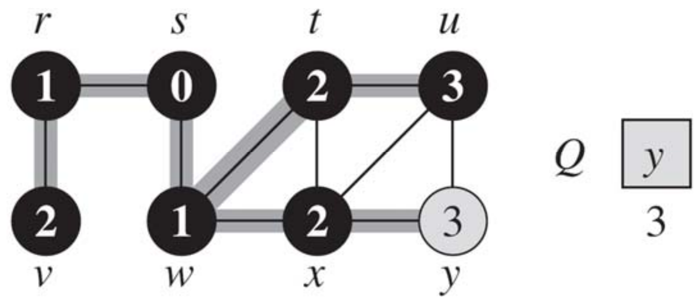
The Operation of BFS on an Undirected Graph (6/9)



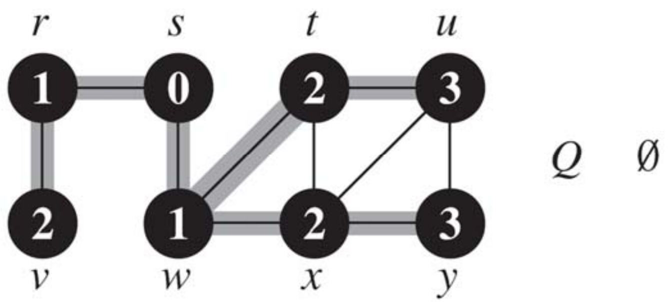
The Operation of BFS on an Undirected Graph (7/9)



The Operation of BFS on an Undirected Graph (8/9)



The Operation of BFS on an Undirected Graph (9/9)



Shortest Paths

- Define the shortest-path distance $\delta(s, v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v; if there is no path from s to v, then $\delta(s, v) = \infty$.
- We call a path of length $\delta(s, v)$ from s to v a shortest path from s to v.

Important Property (1/7)

- **Lemma 1.** Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

  $\delta(s, v) \le \delta(s, u) + 1.$

  Proof.

  If u is reachable from s, then so is v.

  If u is not reachable from s, then $\delta(s, u) = \infty$, and the inequality holds.

Important Property (2/7)

- **Lemma 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 $v.d \ge \delta(s, v)$.

  Proof. We use induction on the number of ENQUEUE operations.

  Basis: (after enqueueing s in line 9) $s.d = 0 = \delta(s, s)$ and $v.d = \infty \ge \delta(s, v)$, for all $v \in V - \{s\}$.

  Inductive Step: Consider a white vertex v that is discovered during the search from a vertex u.

Important Property (3/7)

From the assignment performed by line 15 and from **Lemma 1**, we obtain

$$\begin{aligned} v.d &= u.d + 1 \\ &\ge \delta(s, u) + 1 \\ &\ge \delta(s, v)\,. \end{aligned}$$

Vertex v is then enqueued, and it is never enqueued again because it is also grayed.

➡ The value of $v.d$ never changes again, and the inductive hypothesis is maintained.

Important Property (4/7)

- To prove that $v.d = \delta(s, v)$, we must first show more precisely how the queue Q operates during the course of BFS.
- **Lemma 3.** Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \le v_1.d + 1$ and $v_i.d \le v_{i+1}.d$ for $i = 1, 2, \ldots, r - 1$.

  Proof. We use induction on the number of queue operations.