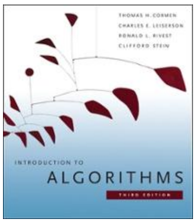
CS146 Data Structures and Algorithms



Chapter 22: Elementary Graph Algorithm

Graphs

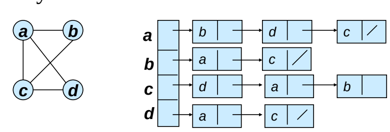
- *Graph* $G = (V, E)$
  - V = set of vertices
  - E = set of edges $\subseteq (V \times V)$
- Types of graphs
  - Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops.)
  - Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed.
  - Weighted: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

Graphs

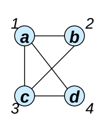
- If $(u, v) \in E$, then vertex v is adjacent to vertex u.
- Adjacency relationship is:
  - Symmetric if G is undirected.
  - Not necessarily so if G is directed.
- If G is connected:
  - There is a path between every pair of vertices.
  - $|E| \geq |V| - 1$.
  - Furthermore, if $|E| = |V| - 1$, then G is a tree.
- Other definitions in Appendix B (B.4 and B.5) as needed.

Representation of Graphs$_1$

- Two standard ways.
  - Adjacency Lists.



  - Adjacency Matrix.



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

Representation of Graphs$_2$

- Undirected graph

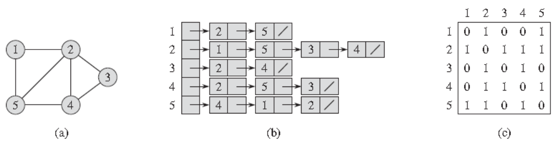


Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G with 5 vertices and 7 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Representation of Graphs$_3$

- Directed Graph

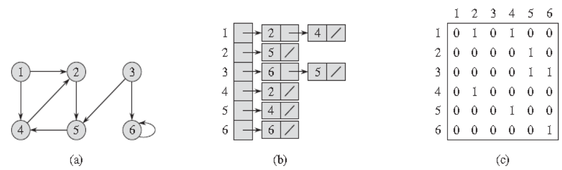
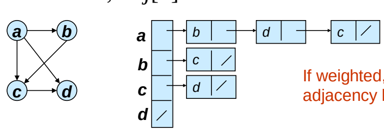


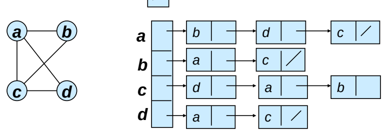
Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G with 6 vertices and 8 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Adjacency Lists

- Consists of an array *Adj* of $|V|$ lists.
- One list per vertex.
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u.

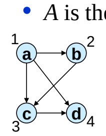


If weighted, store weights also in adjacency lists.

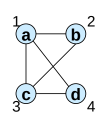


Adjacency Matrix

- $|V| \times |V|$ matrix A.
- Number vertices from 1 to $|V|$ in some arbitrary manner.
- A is then given by: $A[i,j] = a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$



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



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

$A = A^T$ for undirected graphs.

Graph-searching 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

- **Input:** Graph $G = (V, E)$, either directed or undirected, and ***source vertex*** $s \in V$.
- **Output:**
  - $d[v]$ = distance (smallest # of edges, or shortest path) from s to v, for all $v \in V$. $d[v] = \infty$ if v is not reachable from s.
  - $\pi[v] = u$ such that (u, v) is last edge on shortest path $s \rightsquigarrow v$.
    - u is v's predecessor.
  - Builds breadth-first tree with root s that contains all reachable vertices.

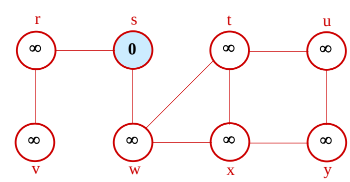
Definitions:
Path between vertices u and v: Sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $u=v_1$ and $v=v_k$, and $(v_i,v_{i+1}) \in E$, for all $1 \le i \le k-1$.
Length of the path: Number of edges in the path.
Path is simple if no vertex is repeated.

```
BFS(G,s)
for each vertex u in V[G] – {s}
     do color[u] ← white
          d[u] ← ∞
          π[u] ← nil
color[s] ← gray
d[s] ← 0
π[s] ← nil
Q ← Φ
enqueue(Q,s)
while Q ≠ Φ
     do u ← dequeue(Q)
          for each v in Adj[u]
               do if color[v] = white
                    then color[v] ← gray
                         d[v] ← d[u] + 1
                         π[v] ← u
                         enqueue(Q,v)
          color[u] ← black
```

white: undiscovered
gray: discovered
black: finished

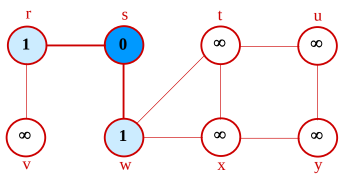
Q: a queue of discovered vertices
color[v]: color of v
d[v]: distance from s to v
π[u]: predecessor of v

Example (BFS)



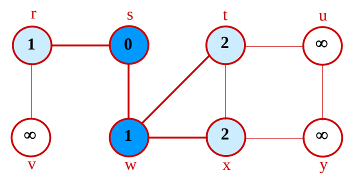
Q:	s
	0

Example (BFS)



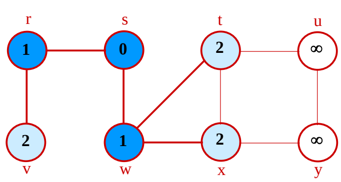
Q: w r
1 1

Example (BFS)



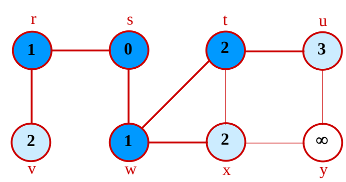
Q: r t x
1 2 2

Example (BFS)



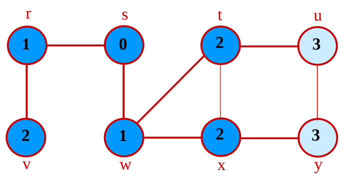
Q: t x v
2 2 2

Example (BFS)



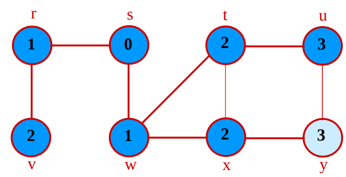
Q: x v u
2 2 3

Example (BFS)



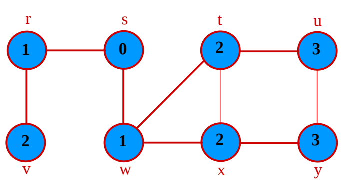
Q: u y
3 3

Example (BFS)



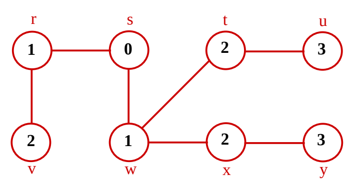
Q: y
3

Example (BFS)



Q: ∅

Example (BFS)



BF Tree

Analysis of BFS

- Initialization takes $O(V)$.
- Traversal Loop
  - After initialization, each vertex is enqueued and dequeued at most once, and each operation takes $O(1)$. So, total time for queuing is $O(V)$.
  - The adjacency list of each vertex is scanned at most once. The sum of lengths of all adjacency lists is $\Theta(E)$.
- Summing up over all vertices => total running time of BFS is $O(V+E)$, linear in the size of the adjacency list representation of graph.