Graph Basics

- Terminologies
- Representations
- Traversals

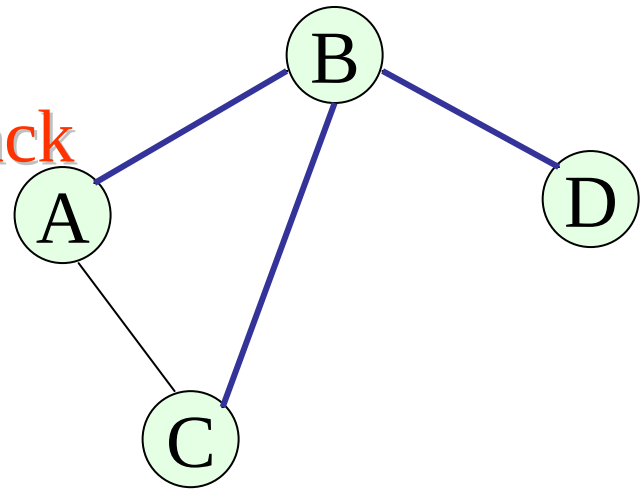
Graph Traversals

- Visits all the vertices that it can reach
- Visits all vertices of the graph if and only if the graph is connected
  - A connected component
    - The subset of vertices visited during a traversal that begins at a given vertex
- To prevent indefinite loops (break the cycles)
  - Mark each vertex during a *visit*, and
  - Never visit a vertex more than once

DFS and BFS Traversals

Depth-First Search (DFS) Traversal

- Proceeds along a path from a vertex *v* as deeply into the graph as possible before backing up
- A "last visited, first explored" strategy
- Has a simple recursive form
- Has an iterative form that uses a stack

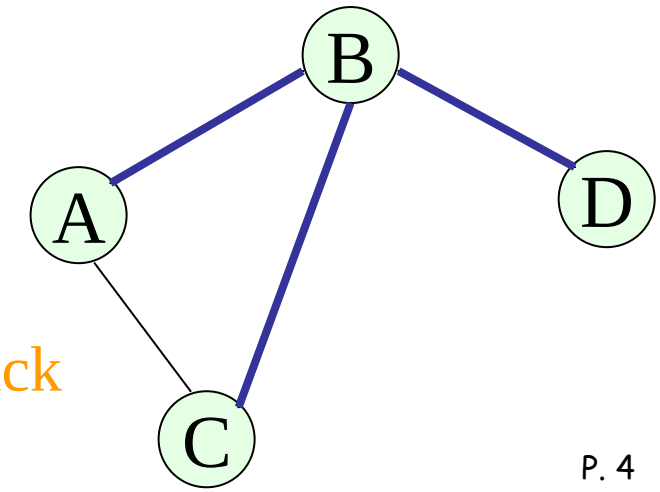


recursiveDFS(Vertex v)
Mark v as visited;
for (*each unvisited vertex u adjacent to v*)
recursiveDFS(u);

DFS in *iterative* form (stack)

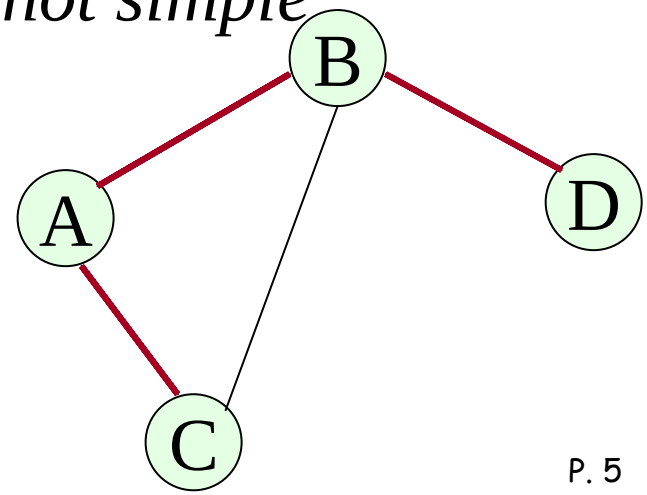
iterativeDFS(Vertex v) **DFS traversal sequence**: **AB** BC B**D**

```
s.createStack();
s.push(v);
Mark v as visited;
while (!s.isEmpty())
{     u = s.getTop();                    // at top of the stack
      if (unvisited vertex w is adjacent to u)
      {       s.push(w);
              Mark w as visited;         // output
      }
      else    s.pop();                   // backtrack
}
```



DFS and BFS Traversals

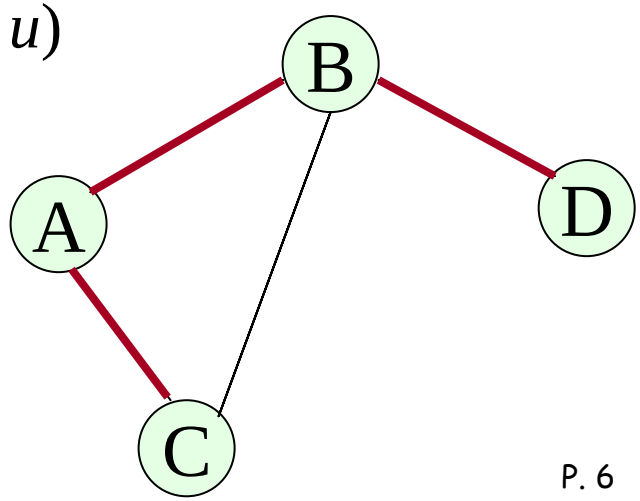
- **Breadth-First Search (BFS) Traversal**
  - Visit every vertex adjacent to a vertex *v* before visiting any other vertex
  - A "first visited, first explored" strategy
  - An iterative form uses a queue
  - A recursive form is possible, but *not simple*



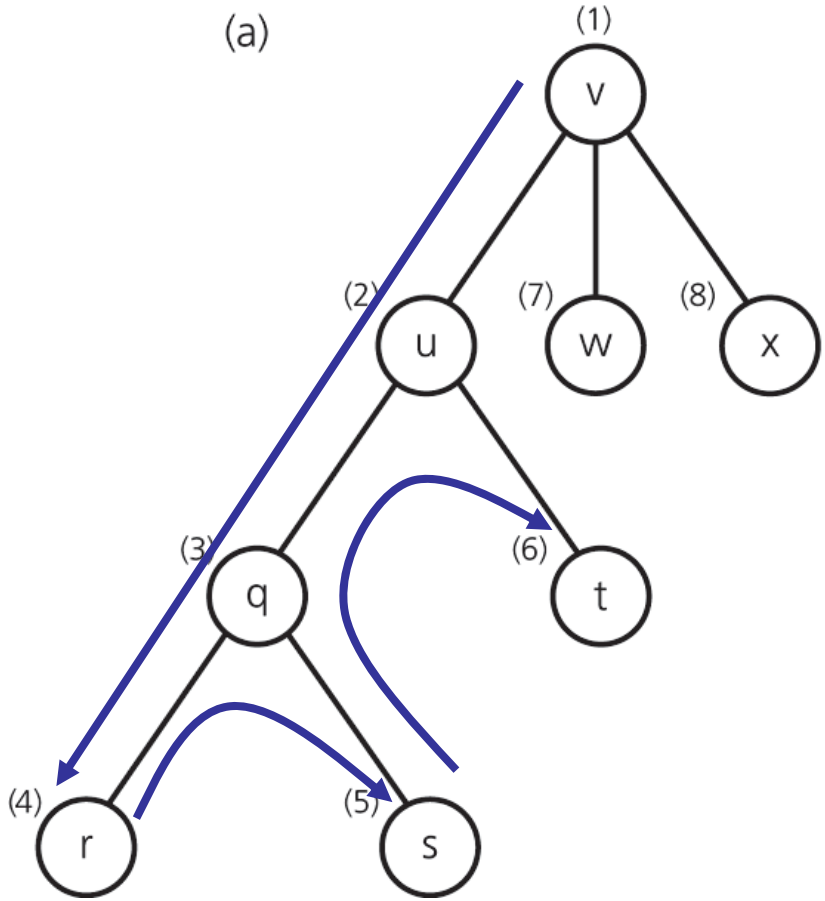
BFS in *iterative* form (queue)

BFS traversal sequence: **AB** AC B**D**

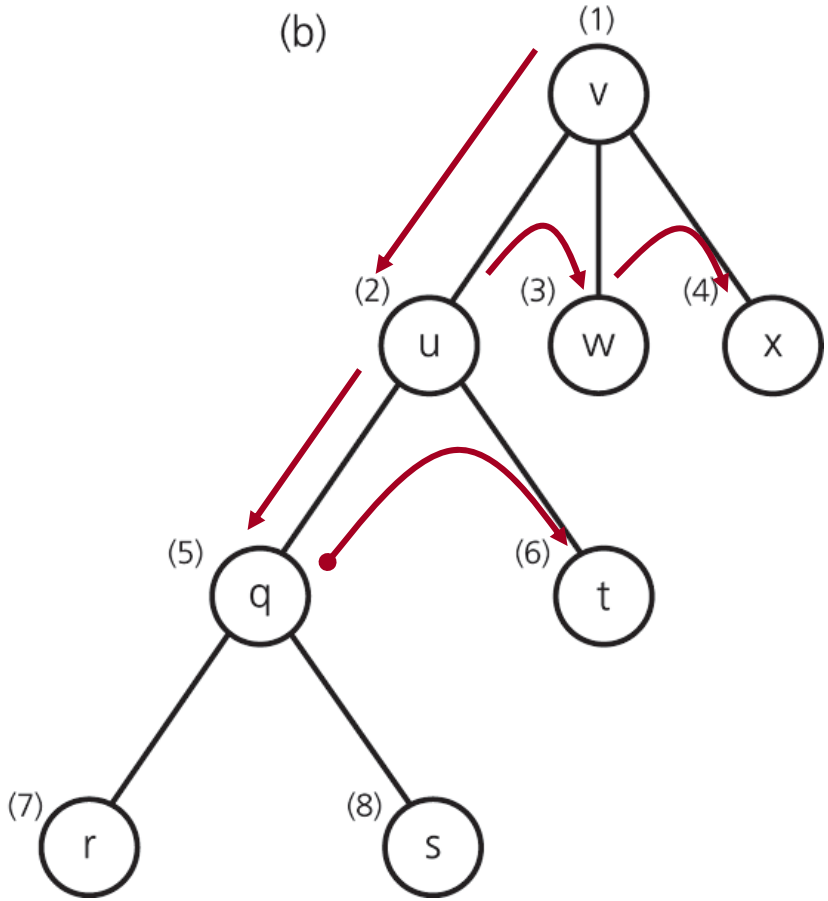
```
iterativeBFS(Vertex v)
    q.createQueue();
    q.enqueue(v);
    Mark v as visited;
    while (!q.isEmpty())
    {       q.dequeue(u);
            for (each unvisited vertex w adjacent to u)
            {           Mark w as visited;      // output
                        q.enqueue(w);
            }
    }
```



DFS and BFS Traversals



DFS

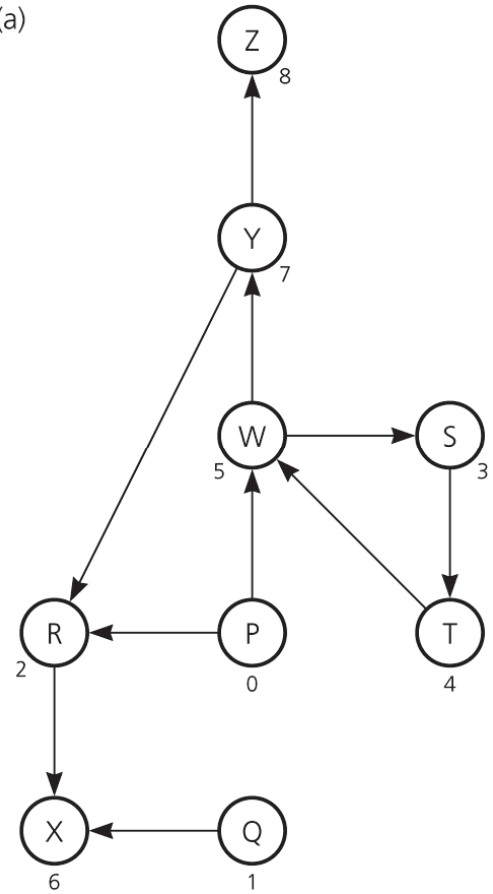


BFS

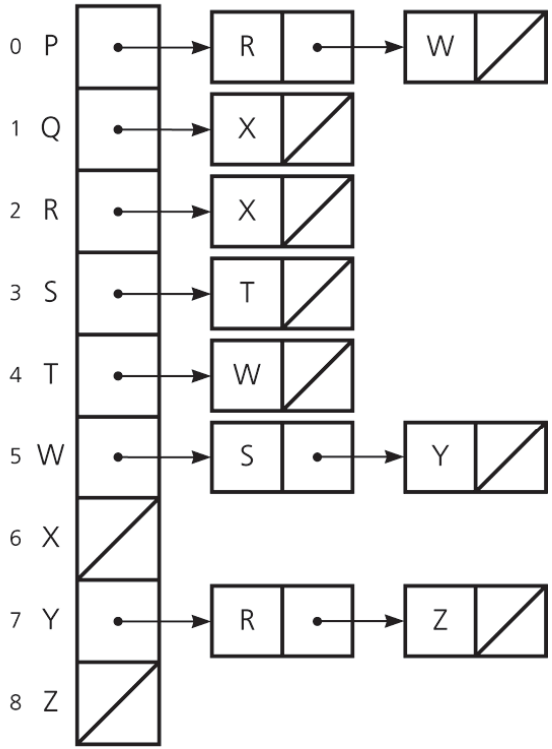
Practice 2: *Graph Traversal Sequences*

- **Starting at P, write down DFS and BFS traversal sequences, if the adjacent vertices are selected in *alphabetical* order.**

(a)



(b)



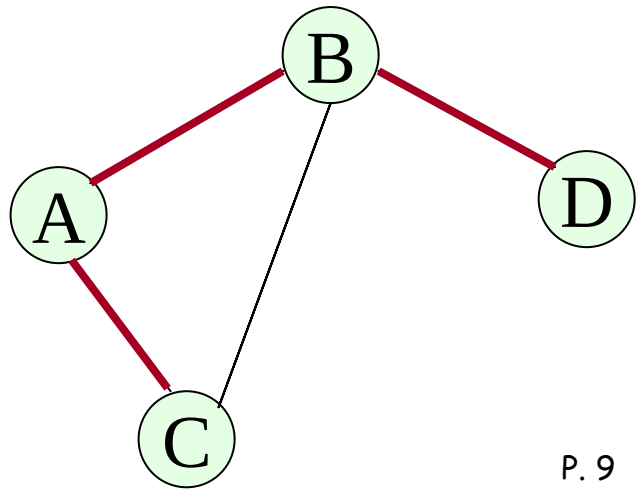
DFS: PR RX PW WS ST WY YZ

PRXWSTYZ

BFS in *recursive* form (queue)

BFS traversal sequence: **AB** AC B**D**

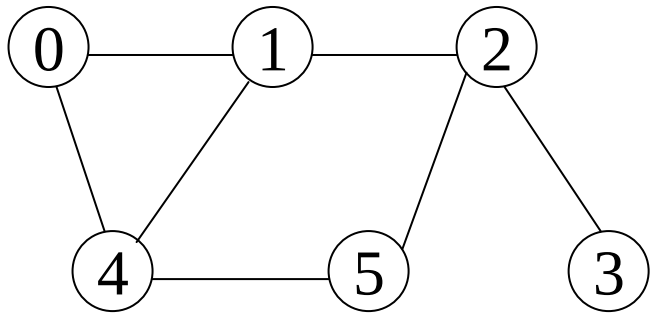
```
    q.createQueue();
    Mark v as visited;
     recursiveBFS(v);
    …
recursiveBFS(Vertex v)
    for (each unvisited vertex u adjacent to v)
    {     Mark v as visited;                    // output
          q.enqueue(u); }
    while (!q.isEmpty())
    {     q.dequeue(w);
          recursiveBFS(w);      }
```



Self-exercise 2

1. Use both **DFS** and **BFS** to traverse the following graph, beginning with vertex 0. List the vertices in the order in which each traversal visits them.

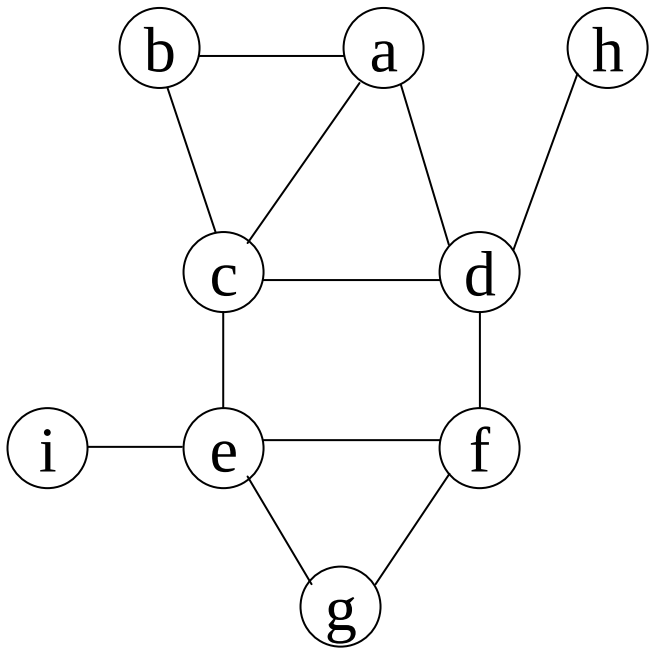
   PS. *If you have multiple choices, always visit the vertex with the smallest label first.*



Self-exercise 2

2. Use both **DFS** and **BFS** to traverse the following graph, beginning with vertex a. List the vertices in the order in which each traversal visits them.

   PS. *If you have multiple choices, always visit the vertex with the smallest label first.*



Summary

- **The most common implementations of a graph use either an adjacency matrix or adjacency list**
- **Graph searching**
  - Depth-first search goes as deep into the graph as it can before backtracking
    - **Uses a stack**
  - Bread-first search visits all possible adjacent vertices before traversing further into the graph
    - **Uses a queue**