# Graphs
# Breadth First Search
# &
# Depth First Search

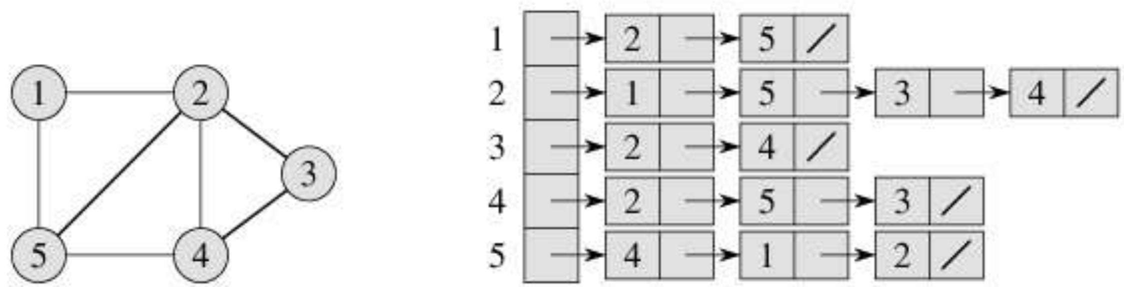
# Contents

- Overview of Graph terminology.
- Graph representation.
- Breadth first search.
- Depth first search. – if time permits
- Pseudocode walkthrough using sample graphs.
- Applications of BFS and DFS.
- References.
- Q & A
- Working example for BFS.

# Graph terminology - overview

- A **graph** consists of
  - set of **vertices** V = {$v_1$, $v_2$, ….. $v_n$}
  - set of **edges** that connect the vertices E ={$e_1$, $e_2$, …. $e_m$}
- Two vertices in a graph are **adjacent** if there is an edge connecting the vertices.
- Two vertices are on a **path** if there is a sequences of vertices beginning with the first one and ending with the second one
- Graphs with ordered edges are **directed**. For directed graphs, vertices have **in and out degrees**.
- **Weighted** Graphs have values associated with edges.

# Graph representation – undirected




(a) (b)

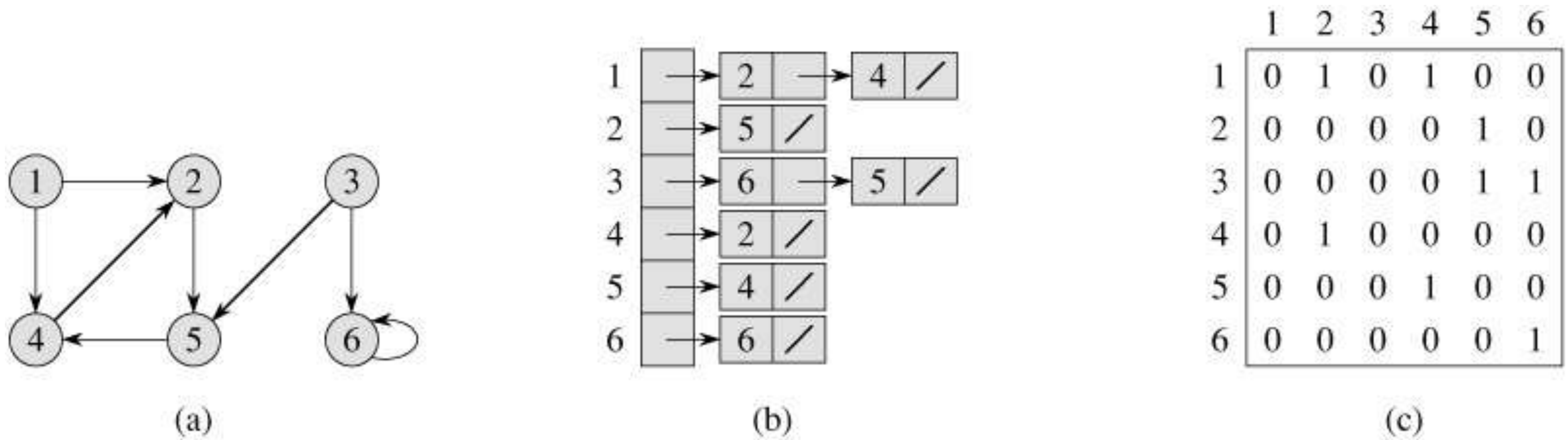
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

(c)

graph Adjacency list Adjacency matrix

ref. Introduction to Algorithms by Thomas Cormen

# Graph representation – directed




graph Adjacency list Adjacency matrix

ref. Introduction to Algorithms by Thomas Cormen

# Some notes

- Adjacency list representation is usually preferred since it is more efficient in representing sparse graphs.
  - Graphs for which $|E|$ is much less than $|V|^2$
- Adjacency list requires memory of the order of $\theta(V+E)$
- Searching a graph means systematically following the edges of the graph so as to visit the vertices.

# Breadth first search

- Given
  - a graph G=(V,E) – set of vertices and edges
  - a distinguished source vertex s
- Breadth first search systematically explores the edges of G to discover every vertex that is reachable from s.
- It also produces a ‘breadth first tree’ with root s that contains all the vertices reachable from s.
- For any vertex v reachable from s, the path in the breadth first tree corresponds to the shortest path in graph G from s to v.
- It works on both directed and undirected graphs. However, we will explore only directed graphs.

ref. Introduction to Algorithms by Thomas Cormen

# Breadth first search

It is so named because

It discovers all vertices at distance k from s before discovering vertices at distance k+1.

Animation:

http://en.wikipedia.org/wiki/Image:Animated_BFS.gif

ref. Introduction to Algorithms by Thomas Cormen

# Breadth first search - concepts

- To keep track of progress, it colors each vertex - white, gray or black.
- All vertices start white.
- A vertex discovered first time during the search becomes nonwhite.
- All vertices adjacent to black ones are discovered. Whereas, gray ones may have some white adjacent vertices.
- Gray represent the frontier between discovered and undiscovered vertices.

# BFS – How it produces a Breadth first tree

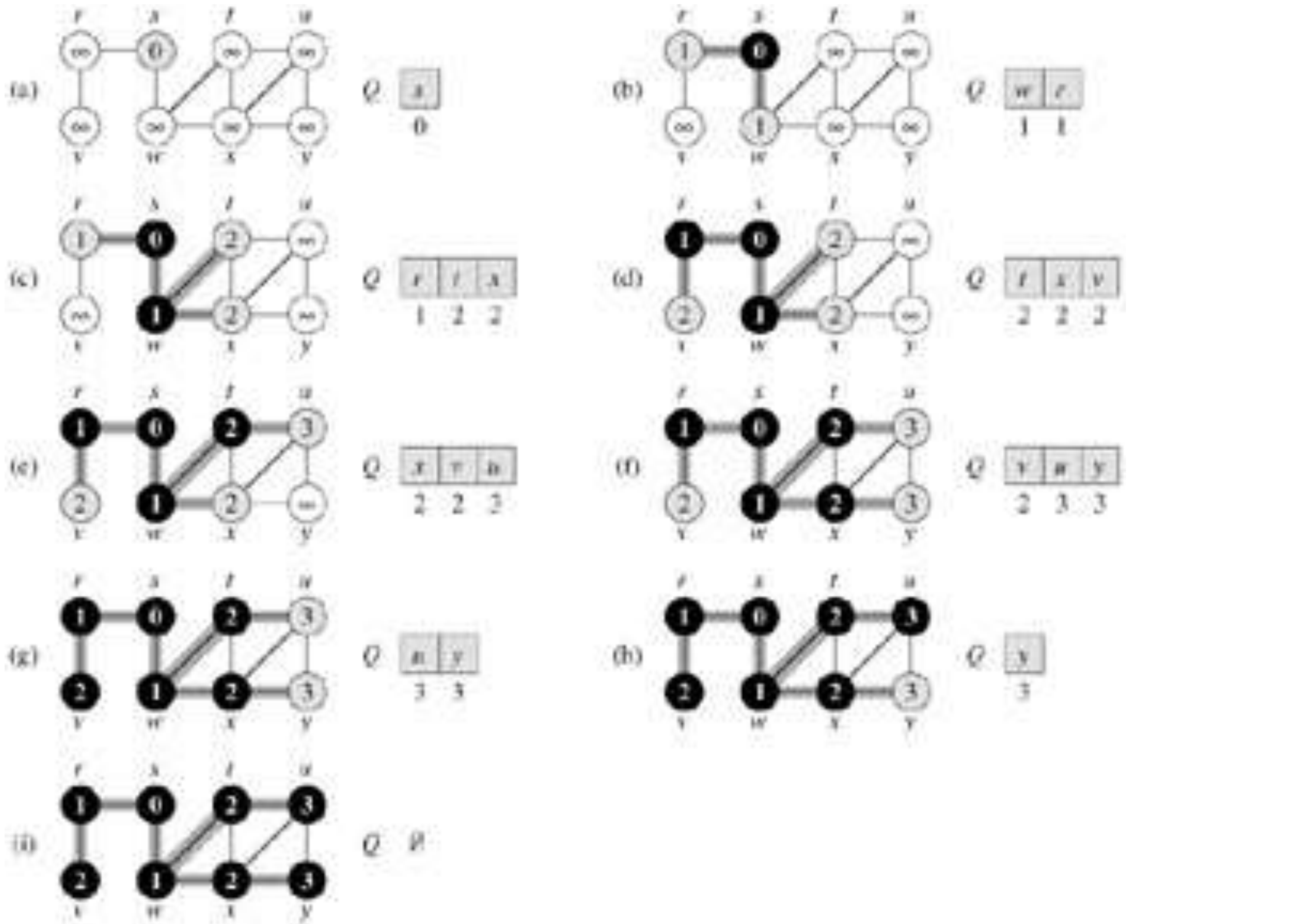
- The tree initially contains only root. – s
- Whenever a vertex v is discovered in scanning adjacency list of vertex u
  - Vertex v and edge (u,v) are added to the tree.

# BFS - algorithm

```
BFS(G, s)                               // G is the graph and s is the starting node
 1  for each vertex u ∈ V [G] - {s}
 2      do color[u] ← WHITE             // color of vertex u
 3         d[u] ← ∞                     // distance from source s to vertex u
 4         π[u] ← NIL                   // predecessor of u
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← Ø                               // Q is a FIFO - queue
 9  ENQUEUE(Q, s)
10   while Q ≠ Ø                        // iterates as long as there are gray vertices. Lines 10-18
11      do u ← DEQUEUE(Q)
12         for each v ∈ Adj[u]
13            do if color[v] = WHITE              // discover the undiscovered adjacent vertices
14                  then color[v] ← GRAY          // enqueued whenever painted gray
15                       d[v] ← d[u] + 1
16                       π[v] ← u
17                       ENQUEUE(Q, v)
18         color[u] ← BLACK             // painted black whenever dequeued
```

ref. Introduction to Algorithms by Thomas Cormen

# Breadth First Search - example



ref: Introduction to
Algorithms by Thomas
Cormen

# Breadth first search - analysis

- Enqueue and Dequeue happen only once for each node. - O(V).
- Sum of the lengths of adjacency lists – θ(E) (for a directed graph)
- Initialization overhead O(V)

Total runtime O(V+E)

ref. Introduction to Algorithms by Thomas Cormen

# Depth first search

- It searches ‘deeper’ the graph when possible.
- Starts at the selected node and explores as far as possible along each branch before backtracking.
- Vertices go through white, gray and black stages of color.
  - White – initially
  - Gray – when discovered first
  - Black – when finished i.e. the adjacency list of the vertex is completely examined.
- Also records timestamps for each vertex
  - d[v] when the vertex is first discovered
  - f[v] when the vertex is finished

ref. Introduction to Algorithms by Thomas Cormen

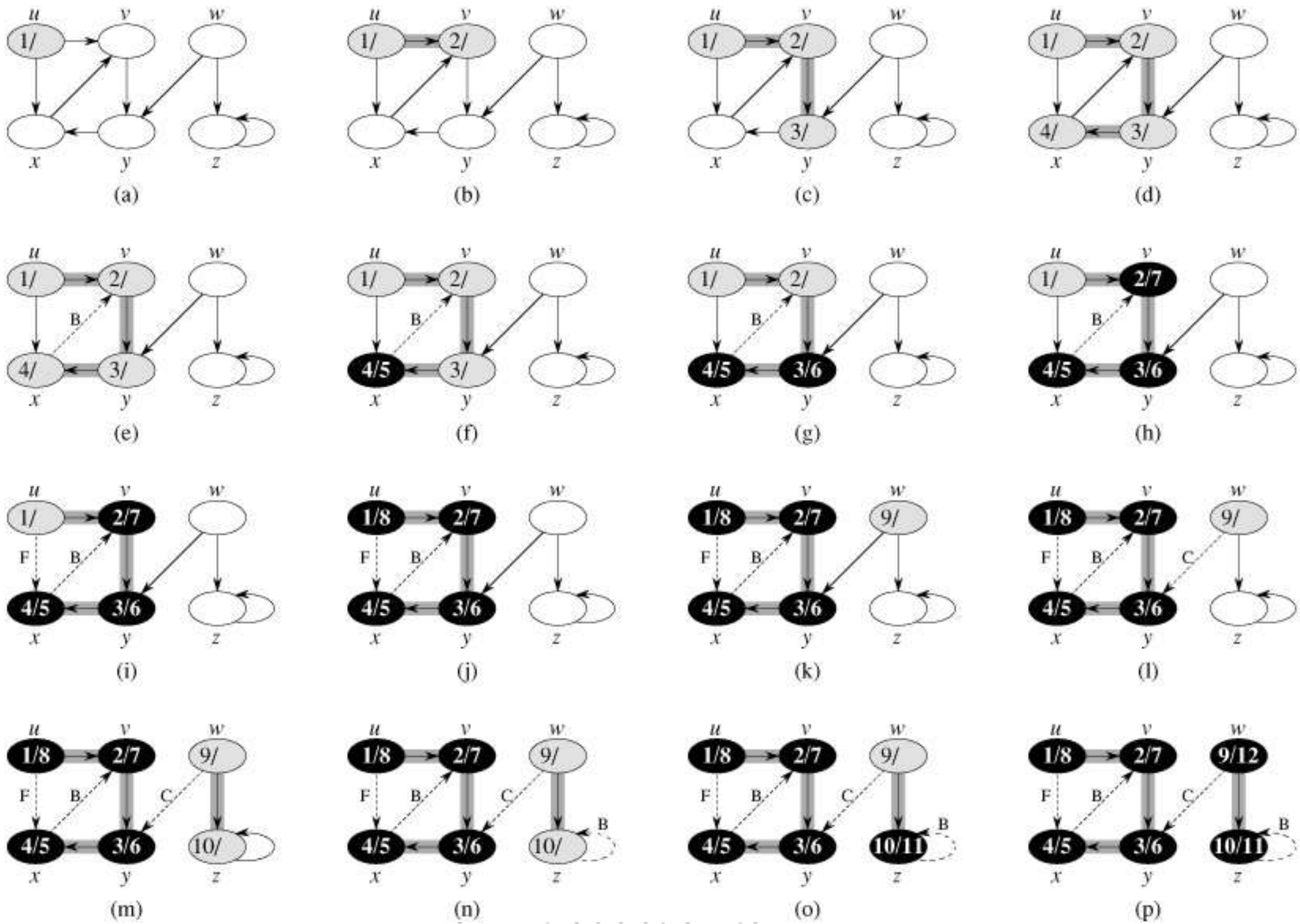
# Depth first search - algorithm

```
DFS(G)
for each vertex u ∈ V [G]
     do color[u] ← WHITE                // color all vertices white, set their parents NIL
        π[u] ← NIL
time ← 0                                // zero out time
for each vertex u ∈ V [G]               // call only for unexplored vertices
     do if color[u] = WHITE             // this may result in multiple sources
           then DFS-VISIT(u)

DFS-VISIT(u)
color[u] ← GRAY     ▹White vertex u has just been discovered.
time ← time +1
d[u] time                  // record the discovery time
for each v ∈ Adj[u]        ▹Explore edge(u, v).
     do if color[v] = WHITE
           then π[v] ← u   // set the parent value
                  DFS-VISIT(v)          // recursive call
color[u] BLACK     ▹ Blacken u; it is finished.
f [u] ▹ time ← time +1
```

ref. Introduction to Algorithms by Thomas Cormen

# Depth first search – example




Algorithms by Thomas Cormen

# Depth first search - analysis

- Lines 1-3, initialization take time $\Theta(V)$.
- Lines 5-7 take time $\Theta(V)$, excluding the time to call the DFS-VISIT.
- DFS-VISIT is called only once for each node (since it's called only for white nodes and the first step in it is to paint the node gray).
- Loop on line 4-7 is executed |Adj(v)| times. Since, $\sum_{v \in V} |Adj(v)| = \Theta(E)$, the total cost of DFS-VISIT it $\theta(E)$

The total cost of DFS is $\theta(V+E)$

ref. Introduction to Algorithms by Thomas Cormen

# BFS and DFS - comparison

- Space complexity of DFS is lower than that of BFS.
- Time complexity of both is same – $O(|V|+|E|)$.
- The behavior differs for graphs where not all the vertices can be reached from the given vertex s.
- Predecessor subgraphs produced by DFS may be different than those produced by BFS. The BFS product is just one tree whereas the DFS product may be multiple trees.

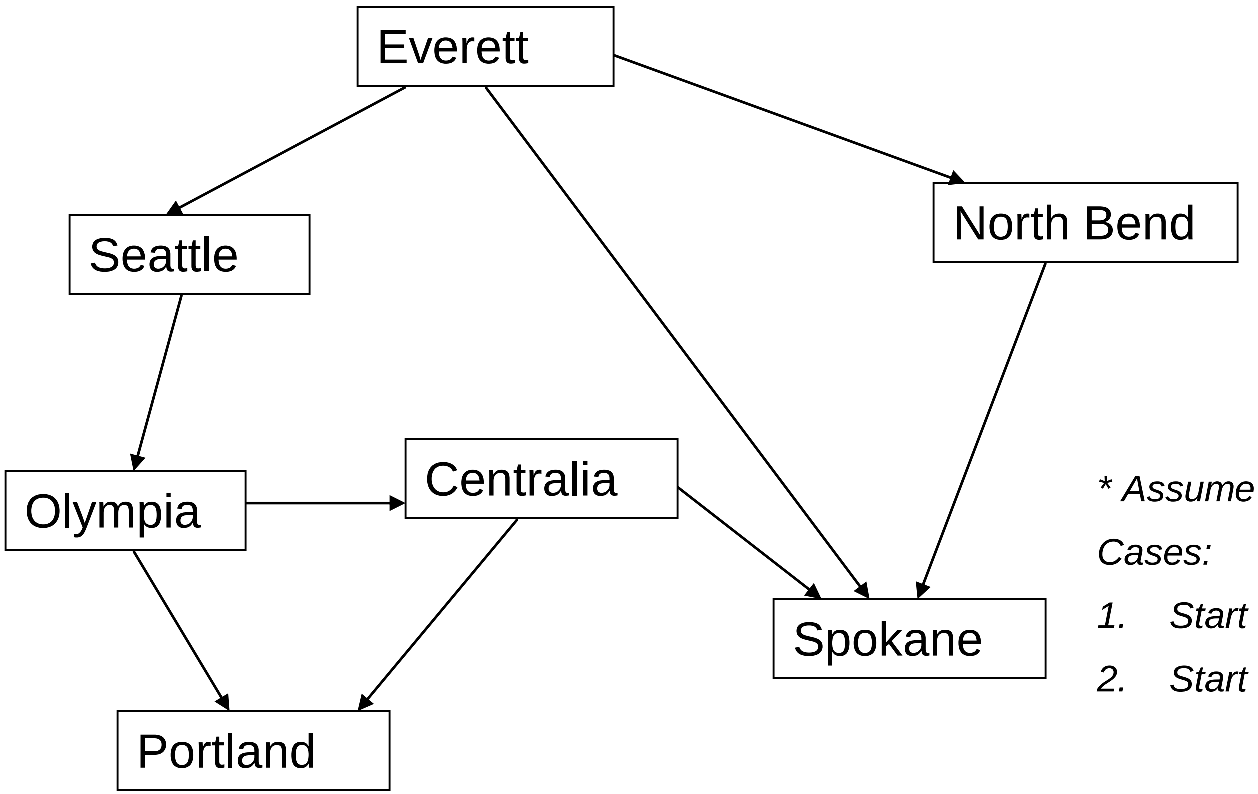
# BFS and DFS – possible applications

- Exploration algorithms in Artificial Intelligence
- Possible to use in routing / exploration wherever travel is involved. E.g.,
  - I want to explore all the nearest pizza places and want to go to the nearest one with only two intersections.
  - Find distance from my factory to every delivery center.
  - Most of the mapping software (GOOGLE maps, YAHOO(?) maps) should be using these algorithms.
  - Companies like Waste Management, UPS and FedEx?
- Applications of DFS
  - Topologically sorting a directed acyclic graph.
    - List the graph elements in such an order that all the nodes are listed before nodes to which they have outgoing edges.
  - Finding the strongly connected components of a directed graph.
    - List all the subgraphs of a strongly connected graph which themselves are strongly connected.

# References


- Data structures with C++ using STL by Ford, William; Topp, William; Prentice Hall.
- Introduction to Algorithms by Cormen, Thomas et. al., The MIT press.
- http://en.wikipedia.org/wiki/Graph_theory
- http://en.wikipedia.org/wiki/Depth_first_search

# Working example for BFS




** Assume – edge value 1 for all*

*Cases:*

1. *Start with Everett*
2. *Start with Olympia*