Data Structures and Algorithms

Prof. Ganesh Ramakrishnan,
Prof. Ajit Diwan,
Prof. D.B. Phatak

Department of Computer Science and Engineering
IIT Bombay

Session: Graph Traversal Algorithm (BFS)

Techniques for Graph Traversal starting at source s

- Breadth First Search (BFS)
  - Discovers all vertices at distance d from s before discovering any vertices at distance $d+1$
- Depth First Search (DFS)
  - Search deeper in the graph whenever possible

Breadth-First Search (BFS)

- $UNEXP \equiv$ Unexplored
- $VIST \equiv$ Visited
- $DISC \equiv$ Discovery
- $SIB \equiv$ edge to a sibling or to a child shared with sibling.

Algorithm BFS(G)
Input: Graph G
Output: Labeling of the edges and partition of vertices of G
for each vertex $u \in V[G]$ **do**
    $setLabel(u, UNEXP)$
end for
for each edge $e \in E[G]$ **do**
    $setLabel(e, UNEXP)$
end for
for each vertex $v \in V[G]$ **do**
    if $getLabel(v) = UNEXP)$ **then**
        BFS(G,v)
    end if
end for

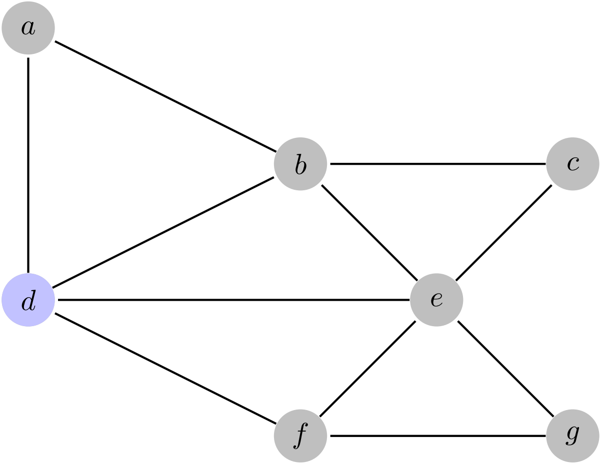
Figure: Breadth-First Search: BFS(G)

Breadth-First Search (BFS contd.)

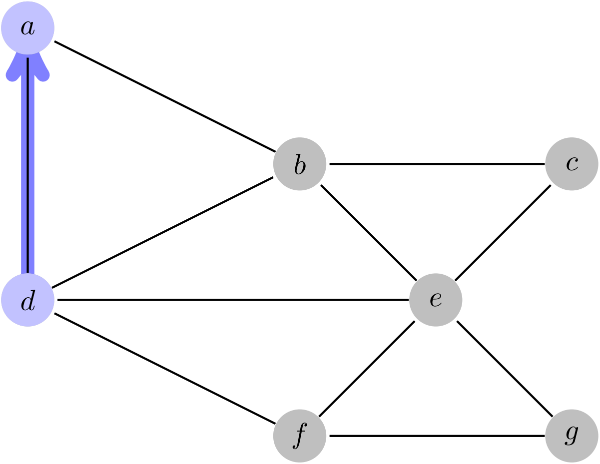
```
Algorithm BFS(G,v)
S_0 ← new empty sequence
S_0.apppend(v)
setLabel(v, VIS)
i ← 0
while ¬S_i.isEmpty() do
    S_{i+1} = newemptysequence
    for each u ∈ S_i.elements() do
        for each e ∈ G.incidentEdges(u) do
            if getLabel(e) == UNEXP then
                w ← e.getOtherVertex(u)
                if getLabel(w) = UNEXP then
                    setLabel(e, DISC)
                    setLabel(w, VIS)
                    S_{i+1}.apppend(w)
                else
                    setLabel(e, SIB)
                end if
            end if
        end for
    end for
    i = i + 1
end while
```

Figure: Breadth-First Search: BFS(G,s)

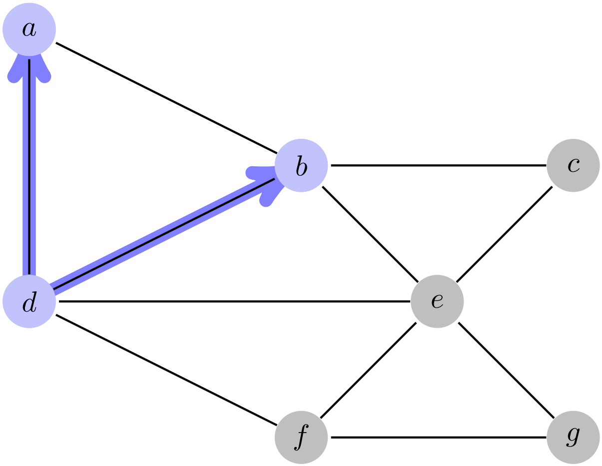
Breadth First Search



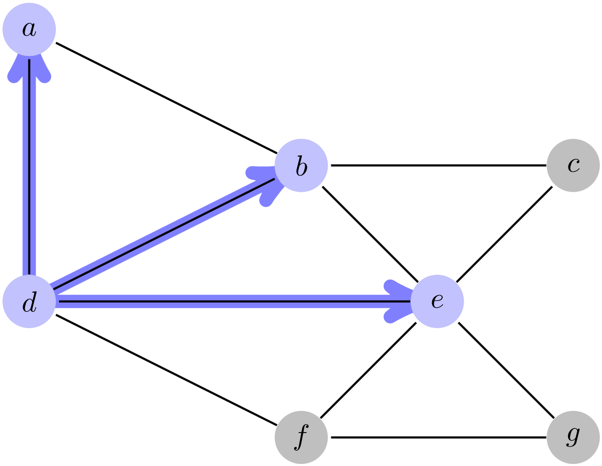
Breadth First Search



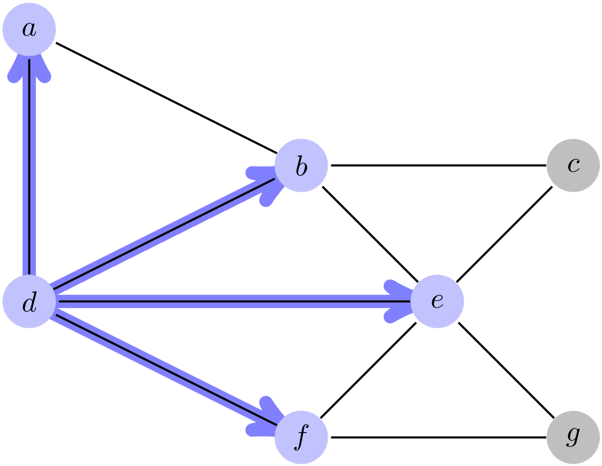
Breadth First Search



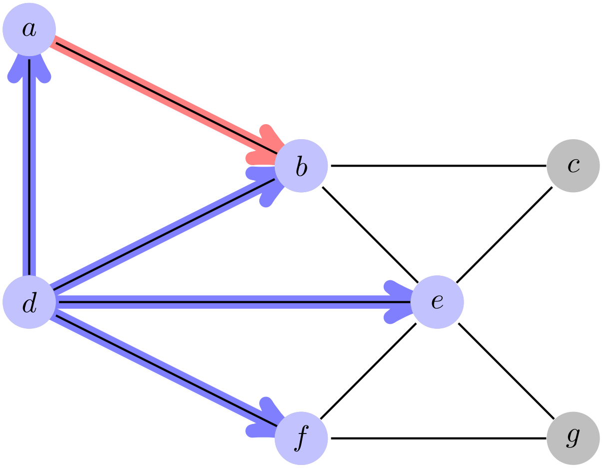
Breadth First Search



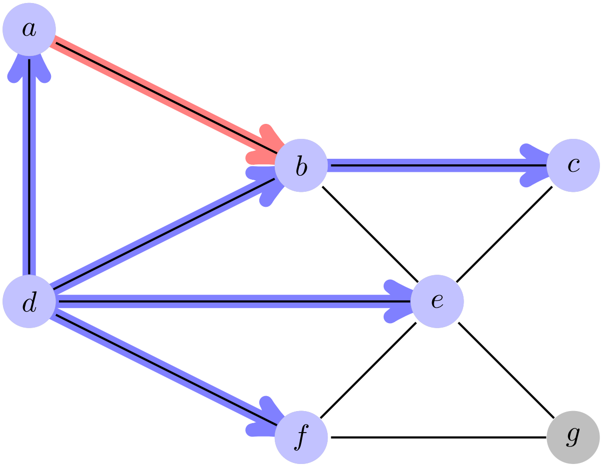
Breadth First Search



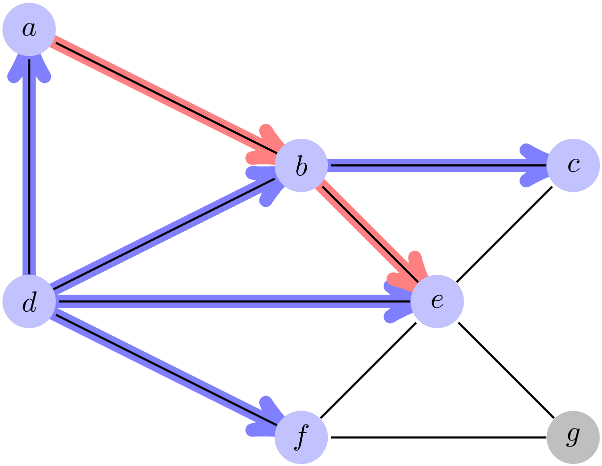
Breadth First Search



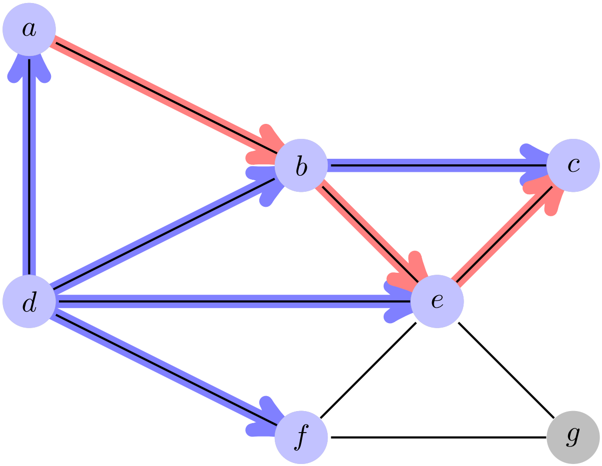
Breadth First Search



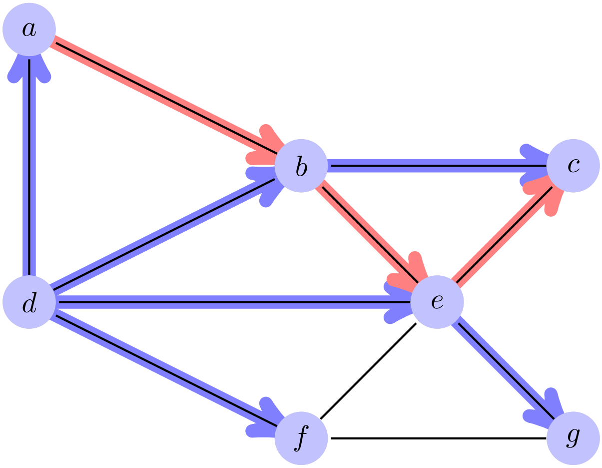
Breadth First Search



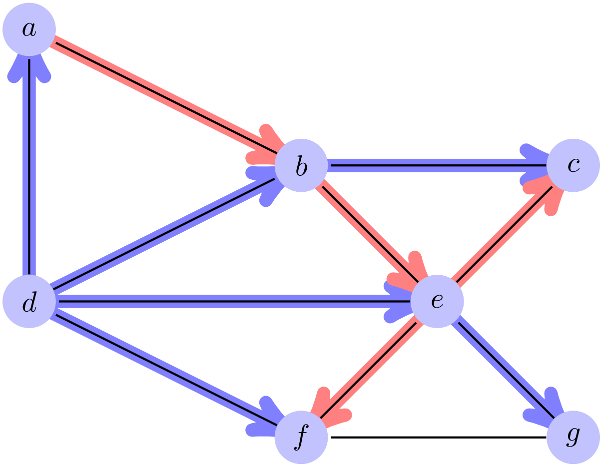
Breadth First Search



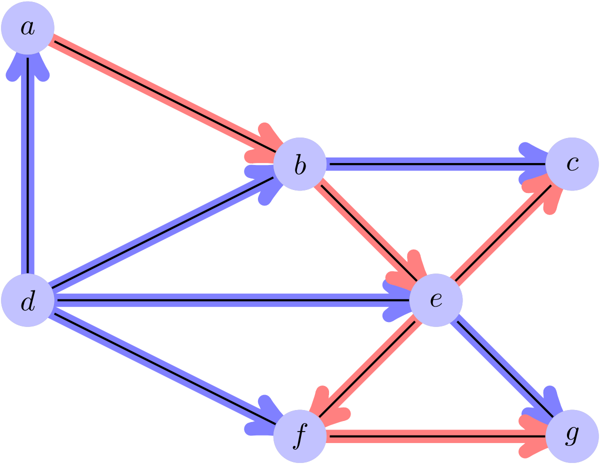
Breadth First Search



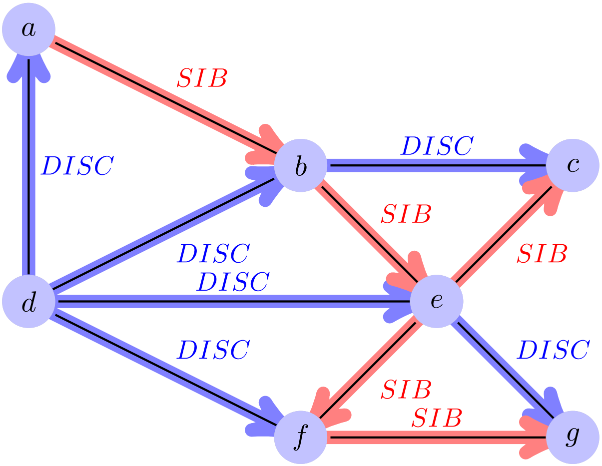
Breadth First Search



Breadth First Search



Breadth First Search



Properties of BFS

1. If G_v is a connected component of a graph G containing vertex v then $BFS(G, v)$ visits all the vertices and edges of G_v
2. The edges labeled $DISC$ by $BFS(G, v)$ form a spanning tree T_v of G_v
3. For each vertex $u \in S_i$,
   - The path from v to u in T_v has i edges
   - Every path from v to u in G_v has atleast i edges

Analysis of BFS

1. Each vertex is labeled twice
   - once as $UNEXP$ (Unexplored)
   - once as VIS (Visited)
2. Each edge is labeled twice
   - once as $UNEXP$ (Unexplored)
   - once as $DISC$ (Discovered) or as $INTR$ (an Intra edge)
3. Each vertex is inserted once into a S_i for some i
4. $G.incidentEdges(u)$ is called once for each vertex $u \implies O(E)$
5. $\implies$ BFS runs in $O(V+E)$ time

Applications of of BFS

BFS traversal of a graph G can be used to solve the following problems in $O(V+E)$ time

- Compute the connected components of G
- Compute a spanning forest of G
- Determine if G is a forest or find a simple cycle in G if there exists one
- Given two vertices of G, find a path (if there exists one) in G between them with the minimum number of edges
- BFS is the classic strategy for determining a solution to the rubix cube

Thank you