

Breadth-First Search

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

1

Breadth-First Search

□ Breadth-first search (BFS) is a general technique for traversing a graph

□ A BFS traversal of a graph G

- Visits all the vertices and edges of G
- Determines whether G is connected
- Computes the connected components of G
- Computes a spanning forest of G

□ BFS on a graph with  $n$  vertices and  $m$  edges takes  $O(n + m)$  time

□ BFS can be further extended to solve other graph problems

- Find and report a path with the minimum number of edges between two given vertices
- Find a simple cycle, if there is one

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

2

BFS Algorithm

□ The algorithm uses a mechanism for setting and getting “labels” of vertices and edges

Algorithm **BFS**( $G$ )

Input graph  $G$

Output labeling of the edges and partition of the vertices of  $G$

for all  $u \in G.vertices()$   
   $setLabel(u, UNEXPLORED)$

for all  $e \in G.edges()$   
   $setLabel(e, UNEXPLORED)$

for all  $v \in G.vertices()$   
  if  $getLabel(v)$   
  =  $UNEXPLORED$   
     $BFS(G, v)$

Algorithm **BFS**( $G, s$ )

$L_0 \leftarrow$  new empty sequence

$L_0.addLast(s)$

$setLabel(s, VISITED)$

$i \leftarrow 0$

while  $\neg L_i.isEmpty()$   
   $L_{i+1} \leftarrow$  new empty sequence

for all  $v \in L_i.elements()$   
    for all  $e \in G.incidentEdges(v)$   
      if  $getLabel(e) = UNEXPLORED$   
         $w \leftarrow opposite(v, e)$   
        if  $getLabel(w)$   
        =  $UNEXPLORED$   
           $setLabel(e, DISCOVERY)$   
           $setLabel(w, VISITED)$   
           $L_{i+1}.addLast(w)$

else  
     $setLabel(e, CROSS)$

$i \leftarrow i + 1$

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

3

Python Implementation

1 def BFS(g, s, discovered):

2     """Perform BFS of the undiscovered portion of Graph g starting at Vertex s.

3

4     discovered is a dictionary mapping each vertex to the edge that was used to

5     discover it during the BFS (s should be mapped to None prior to the call).

6     Newly discovered vertices will be added to the dictionary as a result.

7     """

8     level = [s] # first level includes only s

9     while len(level) > 0:

10       next\_level = [] # prepare to gather newly found vertices

11       for u in level:

12          for e in g.incident\_edges(u): # for every outgoing edge from u

13           v = e.opposite(u)

14           if v not in discovered: # v is an unvisited vertex

15              discovered[v] = e # e is the tree edge that discovered v

16              next\_level.append(v) # v will be further considered in next pass

17       level = next\_level # relabel 'next' level to become current

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

4

Example

A

A

unexplored vertex

visited vertex

unexplored edge

discovery edge

cross edge

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

5

Example (cont.)

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

6

Example (cont.)

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

7

Properties

Notation

$G_s$ : connected component of  $s$

Property 1

$BFS(G, s)$  visits all the vertices and edges of  $G_s$

Property 2

The discovery edges labeled by  $BFS(G, s)$  form a spanning tree  $T_s$  of  $G_s$

Property 3

For each vertex  $v$  in  $L_i$ 

- The path of  $T_s$  from  $s$  to  $v$  has  $i$  edges
- Every path from  $s$  to  $v$  in  $G_s$  has at least  $i$  edges

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

L<sub>0</sub>

A

L<sub>1</sub>

B

C

D

E

F

© 2013 Goodrich, Tamassia, Goldwasser

Breadth-First Search

8

## Analysis

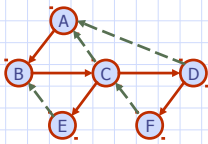
- Setting/getting a vertex/edge label takes  $O(1)$  time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence  $L_i$
- Method incidentEdges is called once for each vertex
- BFS runs in  $O(n + m)$  time provided the graph is represented by the adjacency list structure
  - Recall that  $\sum \deg(v) = 2m$

## Applications

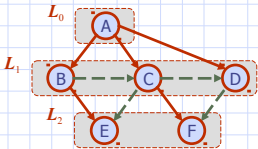
- Using the **template method pattern**, we can specialize the BFS traversal of a graph  $G$  to solve the following problems in  $O(n + m)$  time
  - Compute the connected components of  $G$
  - Compute a spanning forest of  $G$
  - Find a simple cycle in  $G$ , or report that  $G$  is a forest
  - Given two vertices of  $G$ , find a path in  $G$  between them with the minimum number of edges, or report that no such path exists

## DFS vs. BFS

Applications	DFS	BFS
Spanning forest, connected components, paths, cycles	✓	✓
Shortest paths		✓
Biconnected components	✓	



DFS



BFS

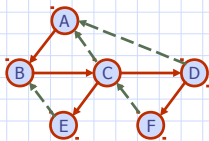
## DFS vs. BFS (cont.)

### Back edge ( $v, w$ )

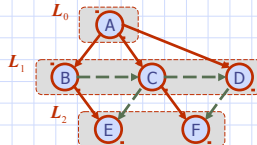
- $w$  is an ancestor of  $v$  in the tree of discovery edges

### Cross edge ( $v, w$ )

- $w$  is in the same level as  $v$  or in the next level



DFS



BFS