Breadth-First Search

Outline and Reading

- Breadth-first search (§6.3.3)
  - Algorithm
  - Example
  - Properties
  - Analysis
  - Applications
- DFS vs. BFS (§6.3.3)
  - Comparison of applications
  - Comparison of edge labels

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 $G$, and 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, such as finding and reporting a path with the minimum number of edges between two given vertices, and finding a simple cycle, if there is one.

Algorithm

**Algorithm BFS**

- **Input**: graph $G$
- **Output**: labeling of the edges and partition of the vertices of $G$

1. $L_0 \leftarrow$ new empty sequence
2. $L_0$.insertLast($s$)
3. setLabel($s$, VISITED)
4. $i \leftarrow 0$
5. while $L_i$.isEmpty() do
   6. $L_{i+1} \leftarrow$ new empty sequence
   7. for all $v \in L_i$.elements() do
      8. for all $e \in G$.incidentEdges($v$) do
         9. if getLabel($e$) = UNEXPLORED
            10. $w \leftarrow$ opposite($v$, $e$)
            11. if getLabel($w$) = UNEXPLORED
                12. setLabel($e$, DISCOVERY)
                13. setLabel($w$, VISITED)
                14. $L_{i+1}$.insertLast($w$)
            15. else
                16. setLabel($e$, CROSS)
         17. $i \leftarrow i + 1$

Example

- unexplored vertex
- visited vertex
- unexplored edge
- discovery edge
- cross edge

Example (cont.)
Example (cont.)

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

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 $\sum_v \deg(v) = 2m$

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning forest, connected components, paths, cycles</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Shortest paths</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Biconnected components</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

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 in the tree of discovery edges