EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

Graphs: Exploration and Searching

Method to explore many key properties of a graph

- Nodes that are reachable from a specific node v
- Detection of cycles
- Extraction of strongly connected components
- Topological sorts
- Find a path with the minimum number of edges between two given vertices

Note: Some slides in this presentation have been adapted from the author's and Prof Elder's slides.

Graph Search Algorithms

- Depth-first Search (DFS)
- Breadth-first Search (BFS)

Breadth First Search

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 $|V|$ vertices and $|E|$ edges takes $\Theta(|V| + |E|)$ 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
    - Cycle detection

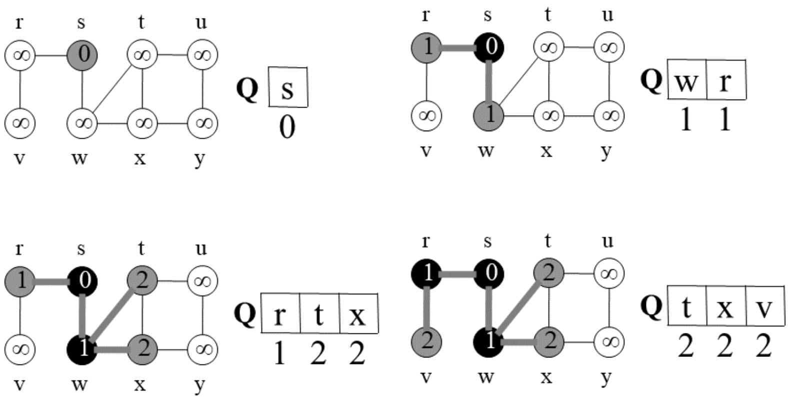
Breadth First Search - 2

- In BFS exploration takes place on a level or wavefront consisting of nodes that are all the same distance from the source s
- We can label these successive wavefronts by their distance: $L_0, L_1, \ldots$

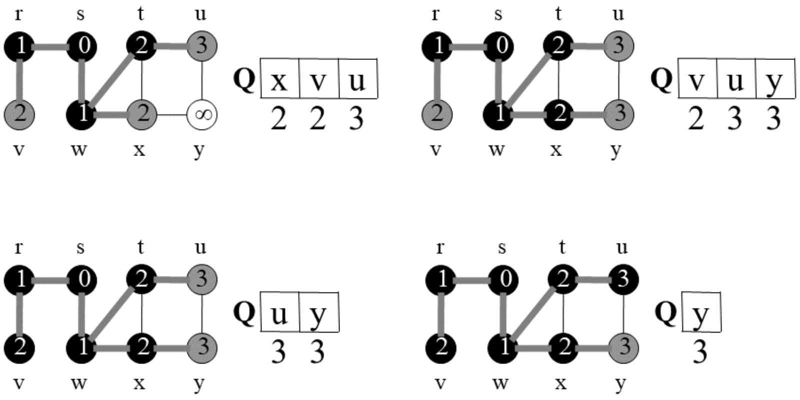
Breadth First Search - 3

- Input: directed or undirected graph $G = (V, E)$, source vertex $s \in V$
- Output: for all $v \in V$
  - $d[v]$, the shortest distance from s to v
  - $\pi[v] = u$, such that (u, v) is the last edge on the shortest distance from s to v
- Idea: send out search 'wave' from s
- Keep track of progress by colouring vertices:
  - Undiscovered vertices are coloured **white**
  - Just discovered vertices (on the wavefront) are coloured **grey**
  - Previously discovered vertices (behind wavefront) are coloured **black**

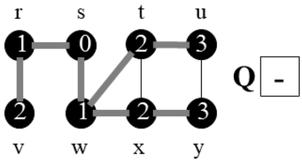
Breadth First Search - Example



Breadth First Search - Example



Breadth First Search - Example



Breadth First Search - Algorithm

```
BFS(G,s)
for each vertex u ∈ V[G]-{s}
    color[u] ← white
    d[u] ← ∞
    π[u] ← NIL
color[s] ← gray
d[s] ← 0
π[u] ← NIL
Q ← {s}
while Q ≠ ∅ do
    u ← head[Q]
    for each v ∈ Adj[u] do
        if color[v] = white then
            color[v] ← gray
            d[v] ← d[u] + 1
            π[v] ← u
            Enqueue(Q,v)
    Dequeue(Q)
    color[u] ← black
```

BFS: Properties

Notation: G_s: connected component containing 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 any vertex v reachable from s, the path in the breadth first tree from s to v corresponds to a shortest path in G

BFS: Analysis

- Setting/getting a vertex/edge label takes $O(1)$ time
- Vertices are enqueued if there color is white
- Assuming that en- and dequeuing takes $O(1)$ time the total cost of this operation is $O(|V|)$
- Adjacency list of a vertex is scanned when the vertex is dequeued (and only then ...)
- The sum of the lengths of all lists is $O(|E|)$. Consequently, $O(|E|)$ time is spent on scanning them
- Initializing the algorithm takes $O(|V|)$
- Thus BFS runs in $\Theta(|V| + |E|)$ time provided the graph is represented by an adjacency list structure

BFS Application: Shortest Unweighted Paths

- Goal: To recover the shortest paths from a source node s to all other reachable nodes v in a graph
  - The length of each path and the paths themselves are returned
- Notes:
  - There are an exponential number of possible paths
  - Analogous to level order traversal for trees
  - This problem is harder for general graphs than trees because of cycles!

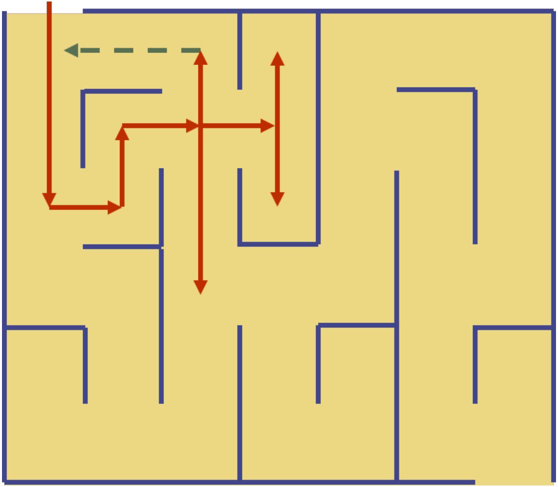
Depth-first Search

A DFS 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
- Find a cycle in the graph

Depth-first Search - 2

DFS: similar to a classic strategy for exploring a maze



Depth-first Search - Steps

- We start at vertex s, tying the end of our string to the point and painting s "visited (discovered)". Next we label s as our current vertex called u
- Now, we travel along an arbitrary edge (u, v)
- If edge (u, v) leads us to an already visited vertex v we return to u
- If vertex v is unvisited, we unroll our string, move to v, paint v "visited", set v as our current vertex, and repeat the previous steps

Depth-first Search - Steps

- Eventually, we will get to a point where *all incident edges on u lead to visited vertices*
- We then backtrack by unrolling our string to a previously visited vertex v. Then v becomes our current vertex and we repeat the previous steps
- Then, if all incident edges on v lead to visited vertices, we backtrack as we did before. We *continue to backtrack along the path we have traveled*, finding and exploring unexplored edges, and repeating the procedure

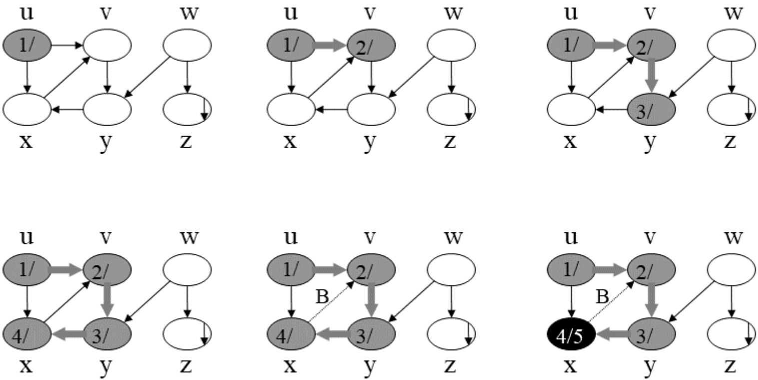
Depth-first Search - Algorithm

- Initialize – color all vertices white
- Visit each and every white vertex using $DFS - Visit$
- Each call to $DFS - Visit(u)$ roots a new tree of the depth-first forest at vertex u
- A vertex is **white** if it is undiscovered
- A vertex is **gray** if it has been discovered but not all of its edges have been discovered
- A vertex is **black** after all of its adjacent vertices have been discovered (the adj. list was examined completely)
- In addition to, or instead of labeling vertices with colours, they can be labeled with discovery and finishing times.

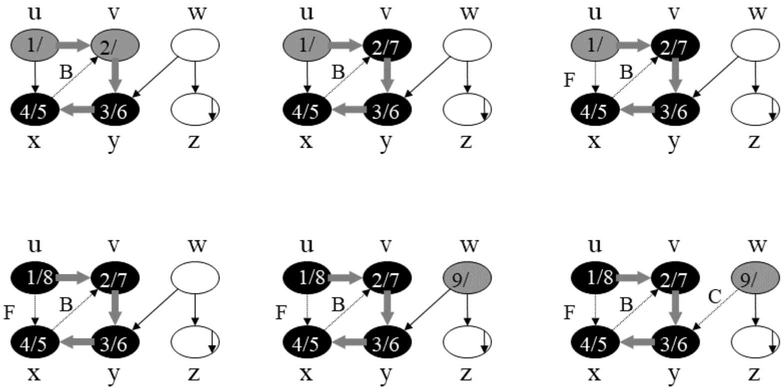
Depth-first Search - Algorithm

- Time is an integer that is incremented whenever a vertex changes state
  - from unexplored to discovered
  - from discovered to finished
- These discovery and finishing times can then be used to solve other graph problems (e.g., computing strongly-connected components)
- Two timestamps put on every vertex:
  - discovery time $d(v) \geq 1$
  - finish time $1 < f(v) \leq 2n$

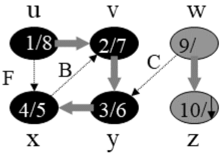
DFS - Example

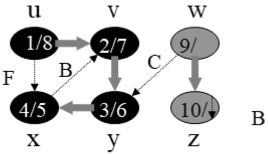

u
v
w
1/
x
y
z
u
v
w
1/
2/
x
y
z
u
v
w
1/
2/
3/
x
y
z
u
v
w
1/
2/
4/
3/
x
y
z
u
v
w
1/
2/
B
4/
3/
x
y
z
u
v
w
1/
2/
B
4/5
3/
x
y
z

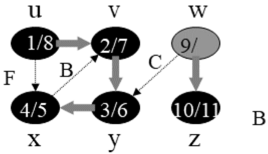
DFS - Example

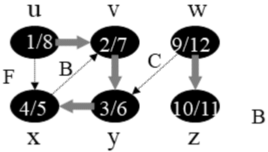

u
v
w
1/
2/
B
4/5
3/6
x
y
z
1/
2/7
B
4/5
3/6
1/
2/7
F
B
4/5
3/6
1/8
2/7
F
B
4/5
3/6
1/8
2/7
9/
F
B
4/5
3/6
1/8
2/7
9/
C
F
B
4/5
3/6

DFS - Example


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
B
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/11
B
x
y
z


u
v
w
1/8
2/7
9/12
F
B
C
4/5
3/6
10/11
B
x
y
z

DFS - Algorithm

```
DFS(G)
for each vertex u ∈ V[G]
        do color[u] ← WHITE
time ← 0
for each vertex u ∈ V[G]
        do if color[u] = WHITE
              then DFS-VISIT(u)

DFS-VISIT(u)
color[u] ← GRAY              ▷ White vertex u discovered.
d[u] ← time                  ▷ Mark with discovery time.
time ← time + 1              ▷ Tick global time.
for each v ∈ Adj[u]          ▷ Explore all edges (u, v).
        do if color[v] = WHITE
              then DFS-VISIT(v)
color[u] ← BLACK             ▷ Blacken u; it is finished.
f[u] ← time                  ▷ Mark with finishing time.
time ← time + 1              ▷ Tick global time.
```