Graph Representation

• Given a graph $G = (V, E)$
• The graph may be directed or undirected
• There are two common ways to represent for algorithms:
  • Adjacency lists.
    • Used when $|E| << |V|^2$ (graph is “sparse”)
  • Adjacency matrix.
    • Used when $|E| \approx |V|^2$ (graph is “dense”)

• Note: How many edges for a complete (has all edges possible) undirected graph?
Running Times

- We will be talking about the running time of graph algorithms in terms of both Vertices $|V|$ and Edges $|E|$.
- We can remove the cardinality when in asymptotic notation.
  - Example: $O(V + E)$

Adjacency Lists

- Array $\text{Adj}$ of $|V|$ (linked) lists, one per vertex.
- Vertex $u$’s list has all vertices $v$ such that $(u, v) \in E$.
- Example:

- Note: Book uses $G.\text{Adj}[u]$ in pseudo-code.
- For weighted graph, can attach weight attribute.
- Degree $\text{deg}(v)$: Number of vertices share edge with $v$. 
Adjacency Lists: Directed Example

- Space: $\theta(V+E)$
- Time to list all vertices adjacent to $u$: $O(1+\text{deg}(u))$
- Time to determine if $(u,v)$ is an edge: $O(1+\text{deg}(u))$

Adjacency Matrix

- $|V| \times |V|$ matrix $A = (a_{ij})$

\[
a_{ij} = \begin{cases} 
1 & \text{if } (i,j) \in E \\ 
0 & \text{otherwise} 
\end{cases}
\]

- Note: If undirected graph, what is a necessary property of Adjacency Matrix?
Google PageRank Algorithm

- Uses directed graph to show if one webpage has a link to another webpage:

  ![Google PageRank Graph](image)

- Uses this graph to create the link matrix: 
  \[
  A = \begin{bmatrix}
  0 & 0 & 1 & \frac{1}{2} \\
  \frac{1}{3} & 0 & 0 & 0 \\
  \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
  \frac{1}{3} & 0 & 0 & 0 \\
  \end{bmatrix}
  \]

- The PageRank vector is the vector \( x \) for which \( Ax = x \) (i.e. eigenvector corresponding to eigenvalue \( \lambda = 1 \)).

*The $25,000,000,000 Eigenvector: Linear Algebra behind Google* by Kurt Bryan and Tanaya Leise (see webpage for link)

Adjacency Matrix

- Space: \( O(V^2) \)
- Time to list all vertices adjacent to \( u \): \( O(V) \)
- Time to determine if \( (u,v) \) is an edge: \( O(1) \)

- What about weighted graphs?
  Store weight \( w(u,v) \) of edge \( (u,v) \) with vertex \( v \) in \( u \)'s adjacency list.
Breadth-First Search

- Input: Graph G = (V, E), either directed or undirected, adjacency list representation, source vertex s is in V.
- Output:
  - d[v] = distance (smallest # of edges) from s to v, for all v in V (or v.d)
  - π[v] = u such that (u,v) is last edge on shortest path s->v (or v.π=u)
  - u is v’s predecessor or parent (each v has at most one)
  - Set of edges {(π[v],v): v ≠ s} forms a tree

Breadth-First Search

- Idea: Send a wave out from s.
  - First hits all vertices 1 edge from s.
  - From there, hits all vertices 2 edges from s.
  - Etc.
- Use FIFO queue Q to maintain wavefront.
  - v is in Q if and only if wave has hit v but has not come out of v yet
**BFS(G, s)**

Each vertex $u$ has attributes:

- $u.color$:
  - WHITE: Not visited
  - GRAY: Visited but still considering
  - BLACK: Done

- $u.d$: distance from source vertex

- $u.\pi$: predecessor vertex (parent)

**Example**

```
BFS(G, s)
1   for each vertex u in G.V - {s}
2       u.color = WHITE
3       u.d = \infty
4       u.\pi = NIL
5   s.color = GRAY
6   s.d = 0
7   s.\pi = NIL
8   Q = \emptyset
9   ENQUEUE(Q, s)
10  while Q \neq \emptyset
11      u = DEQUEUE(Q)
12         for each v in G.Adj[u]
13             if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.\pi = u
17                     ENQUEUE(Q, v)
18               u.color = BLACK
```
Breadth-First Search

• Will breadth-first search reach all vertices?
• Time Complexity: O(V)+O(E)
  • Initialization: O(V) (Lines 1-8)
  • Enqueue/Dequeue each vertex at most once: O(V) total
  • Scan adjacency list when dequeued: O(E) total

• Space Complexity: O(V)

• Compared to DFS (next), this algorithm is complete in that it will always produce the v.d and v.π for the given connected component of a graph.

Depth-First Search

• Input: G = (V, E), directed or undirected. No source vertex given.
• Output: 2 timestamps on each vertex:
  • d[v] = discovery time (or v.d)
  • f[v] = finishing time (or v.f)
  • π[v] = u if DFS-VISIT(G,v) called when reviewing u's adjacency list (or v.π)

• Uses stack instead of a queue to maintain list of vertices
• Notation:

v.d/v.f

• Also parenthesis structure in distributed example
DFS(G)

DFS(G)
for each $u \in G.V$
    $u.color = \text{WHITE}$
    $time = 0$
for each $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-Visit($G, u$)

DFS-Visit($G, u$)
    $time = time + 1$
    $u.d = time$
    $u.color = \text{GRAY}$ // discover $u$
for each $v \in G.Adj[u]$ // explore $(u, v)$
    if $v.color == \text{WHITE}$
        DFS-Visit($v$)
    $u.color = \text{BLACK}$
    $time = time + 1$
    $u.f = time$ // finish $u$

Each vertex $u$ has attributes:

$u.color$:
WHITE: Not visited
GRAY: Visited but still considering
BLACK: Done

$u.d = time$ u discovered
$u.f = time$ u finished (done with adjacency list)

Example
Depth-First Search

• Time :
  \( \text{DFS}(G) : \theta(V) \) // Excluding call to DFS-VISIT\((G,v)\)
  \( \text{DFS-VISIT}(G,v) : \text{Total } \theta(E) \ (|\text{Adj}(v)| \text{ for each vertex}) \)

  Total: \( \theta(V+E) \)

• Space: \( O(V) \)

Your Turn

• Solve exercise 22.3-2 on page 610
Classification of Edges

- Tree edge: Edge traversed during search
- Back edge: edge \((u,v)\) from vertex \(u\) to ancestor \(v\) in DFS tree
- Forward edge: edge \((u,v)\) from vertex \(u\) to descendant \(v\) in DFS tree
- Cross edge: any other edge