Graph Representation

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

from

Adjacency List

Y0:

Col 1: Y0 D2

D 2: Y0

Pr 3: Y0 D2 H4

n: 5
m: 9
Pr 3: $A \subseteq B^2$

H 4: \[ A \subseteq B^2 \]
Graph Implementation Time/Space Complexity

\[ \begin{array}{ccc}
\text{Adj Matrix} & \text{Adj List} & \text{Adj Set (Hash)} \\
\text{Space} & \Theta(n^2) & \Theta(n+m) \\
& & \text{worst case } \Theta(n^2) \text{ (dense)} \\
& & \text{best case } \Theta(n) \text{ (no edges or connected)} \\
\text{has_edge} & \Theta(1) & O(n) \\
& & O(\text{outdegree}(v)) \\
\text{add_edge} & \Theta(1) & \Theta(1) \\
& & \Theta(1) \text{ expected} \\
& & \text{ (even w/o presort)} \\
\text{for_each_neighbor} & \Theta(n) & \Theta(\text{outdegree}(v)) \\
& & O(n) \\
\text{for_each_edge} & \Theta(n^2) & \Theta(n+m) \\
& & \sum_{i=0}^{n-1} (1 + \text{outdegree(vert-i)}) \\
& & = \sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1} \text{outdegree(vert-i)} \quad \text{(directed graph)} \\
& & = n + m
\end{array} \]
Depth-first search: Keep following edges from current vertex. Backtrack when no edges to unvisited vertices.
DFS-VISIT(G, u)

for each v adjacent to u
    if (color[v] = WHITE)
        pred[v] = u
        dist[v] = dist[u] + 1
        color[v] ← GRAY
        DFS-VISIT(G, v)

    color[u] ← BLACK

so just like for each vertex u
    for each edge (u, v)
        do something

O(n + m)

DFS(G)

    for each u in G.V
        color[u] ← WHITE

    time ← 0

    for each u in G.V
        if color[u] = WHITE
            DFS-VISIT(G, u)
Breadth-First Search

starting from s
find verts 1 edge from s
find verts 2 edges from s
BFS(V,E,s)

for each vertex u in V
  color[u] ← WHITE
  d[u] ← infinity
  pred[u] ← NULL

F ← []

color[s] ← GRAY
d[s] ← 0
pred[s] ← NULL
Q ← [s]

while not Q.isEmpty()
  u ← Q.dequeue() \(\Theta(1)\)
  for each v adjacent to u
    pred[v] ← u
    d[v] ← d[u] + 1
    color[v] ← GRAY
    Q.enqueue(v) \(\Theta(1)\)
  color[u] = BLACK
  F = F + u

*< iterates at most once for each vertex* 
*so just like for each vertex u, for each edge \((u,v)\) do something* 
*\(O(n + m)\)*