### Graph Implementation Time/Space Complexity

<table>
<thead>
<tr>
<th></th>
<th>Adj Matrix</th>
<th>Adj List</th>
<th>Adj Set (Hash)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>$O(n^2)$</td>
<td>$O(n^2)$ worst case</td>
<td>$O(n+m)$</td>
</tr>
<tr>
<td><code>has_edge</code></td>
<td>$O(1)$</td>
<td>$O(n)$ worst case</td>
<td>$O(1)$ expected</td>
</tr>
<tr>
<td><code>add_edge</code></td>
<td>$O(1)$</td>
<td>$O(1)$ amortized</td>
<td>$O(1)$ expected</td>
</tr>
<tr>
<td>precondition: edge doesn't exist</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>for_each_out_neighbor</code></td>
<td>$O(n)$</td>
<td>$O(n)$ worst case</td>
<td>$O(n)$ worst-case</td>
</tr>
<tr>
<td><code>for each vertex for each out-neighbor</code></td>
<td>$O(n^2)$</td>
<td>$O(n+m)$</td>
<td>$O(n+m)$</td>
</tr>
</tbody>
</table>

$$0 \leq m \leq n \cdot (n-1)$$

dense space: $m$ is $O(n)$
**Breadth-First Search**

- Finds shortest paths from starting point in unweighted graph

```
Q ← [start]
color[start] ← in queue
d[start] ← 0
pred[start] ← NULL

while Q not empty
  u ← dequeue(Q)
  for each outneighbor v of u
    if color[v] = unseen
      enqueue(Q, v)
      color[v] ← in queue
      if v ← u
        d[v] ← d[u] + 1
      color[u] ← done
```

for each vertex (in order of increasing d)
for each outneighbor

$\mathcal{O}(n+m)$ using adj. list
Depth-first search: backtracking graph exploration
minimum spanning tree (MST)

Depth-first search: backtracking graph exploration