\[ G = (V, E) \to \{(a, b), (a, d), (d, c), (e, b), (d, e), (c, e)\} \]

\[ \{c, b, c, d, e\} \]
1) adjacency matrix

- 1 row, 1 col for each vertex
- \( a_{ij} = \begin{cases} 
1 & \text{if edge } (v_i, v_j) \in E \\
0 & \text{otherwise} 
\end{cases} \)

- can check if \((u, v) \in E\) in \(O(1)\) time
- can iterate through neighbors of \(u\) in \(O(V)\)
- can iterate through all edges in \(O(V^2)\)
- \(\Theta(V^2)\) storage space

2) adjacency list

- for each vertex, maintain a linked list
- set \(v\) is on list if \((u, v) \in E\)
- for each \(v \in V\)
- for each \(u \in \text{Adj}[v]\)
- check \((u, v) \in E\) in \(O(V)\) time
- process \((v, u)\)
- iterate through all edges in \(O(E+V)\) time
-
Connect graph has
\[ E \in \Omega(\sqrt{V}) \]
so \( V \) is dominated by \( E \)
so \( O(V+E) = O(E) \)

Storage space is \( O(V+E) \)

Breadth 1st Search

1) figure out where we can get using 1 edge
2) figure out where we can get from verts found in 1)
3) figure 2)

Keep going until no new verts found
BFS(v)

let Q = ∅
Q.enqueue(v)
d[v] ← 0 < color is BLACK if done
GRAY if seen but not explored

∀ u ∈ V, color[u] ← WHITE WHITE not seen

color[v] = GRAY

while (Q ≠ ∅)

u ← Q.dequeue() executes at most once for each vertex
for each w ∈ Adj[u]

if (color[w] = WHITE)
Q.enqueue(w) executes at most once for each edge
O(V + E)
color[v] = GRAY O(E)
d[w] = d[u] + 1 considered linear for
color[w] = BLACK graph sizes

considered linear for graph sizes