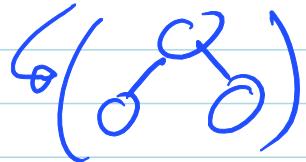


Graph Traversal Algorithms

→ Difference between graphs and trees

→ Trees ⊂ Graphs!

→ Graphs can have cycles, trees don't



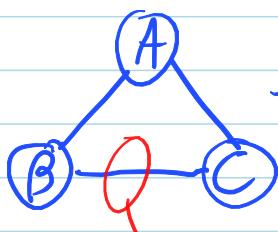
→ Why are graphs useful?

→ representation of complex networks

→ PATHS

→ GRAPH definition: set of vertices connected by a set of edges

e.g.



$$G = G(V, E)$$

$$E = \{A, B, C\}$$

$$V = \{(A, B), (B, C), (C, A)\}$$

assume
BI-DIRECTIONALITY
without (\rightarrow or \leftarrow)

→ with n nodes → edges ≤
in a
SIMPLE
UNDIRECTED
GRAPH.

$$nC_2$$

$$\therefore \text{edges} \leq \frac{n(n-1)}{2}$$

→ vertex's degree = number edges coming into it

→ PATH: succession of edges between nodes
Cycle = path + SAME $\begin{matrix} \leftarrow \\ \text{start} \end{matrix} \rightarrow \begin{matrix} \rightarrow \\ \text{end} \end{matrix}$ indexes

- SUBGRAPH: a subset of (nodes / edges) in a graph
- Spanning SUBGRAPH: All nodes, but subset of edges
- Forest: graph without cycles ∵ Tree = connected forest

→ Adjacency list, adjacency matrix

list of all nodes, where each node x has
a list of all its adjacent nodes
→ directed graphs: target nodes with X = source

weights of all edges

1 = one edge ($> 1 \rightarrow > 1$ edges)

③ Can we create adjacency matrix
for sparse graphs ?? → Yes, but it'll be a

sparse matrix

↓
Hash Table!

→ Depth First Search (undirected graph G)

→ backtracking technique

DFS (graph G , vertex V)

$V = \text{explored}$

E (incident to V)

if ($e = \text{explored}$)

go to w along e

if w is unexplored then

label e as discovery edge

DFS (G, w)

else

label e as back track edge

go back along it to V

endif

end if

end for

→ BFS has the same complexity

// with n nodes, e edges,
complexity = $O(n + e)$

BFS(Graph g, Vertex v)