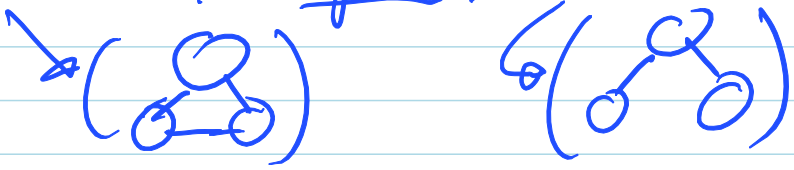
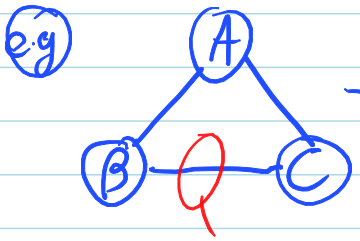
Graph Traversal Algorithms

- 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 veiticies connected by a set of edges



assume BI-DIRECTIONALITY without (→ or ↔)

$G = G(V, E)$

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

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

- with (n) nodes in a SIMPLE UNDIRECTED GRAPH

$$\text{edges} \leq {}^{n}C_{2}$$

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

- vertex's degree = number edges coming into it
- PATH: succession of edges between nodes
  - ∴ Cycle = path + SAME $\begin{cases} \text{start} \\ \text{end} \end{cases}$ 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
  - HashTable!
  - weights of all edges
    - 1 = one edge (>1) → >1 edges

(?) 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)        //with n nodes, e edges,
                                complexity = O(n+e)
  V=explored
  ∀(e incident to V)
    if (e = ¬explored)
      go to w along e
      if w is unexplored then
        label e as discovery edge
        DFS(G,w)
      else
        label e as backtrack edge
        go back along it to V
      endif
    endif
  end for
```

- BFS has the same complexity