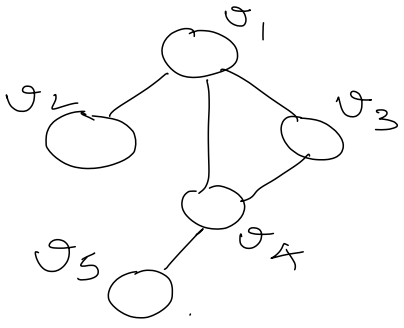


In tree → parent - child relationship

Vertex = Node

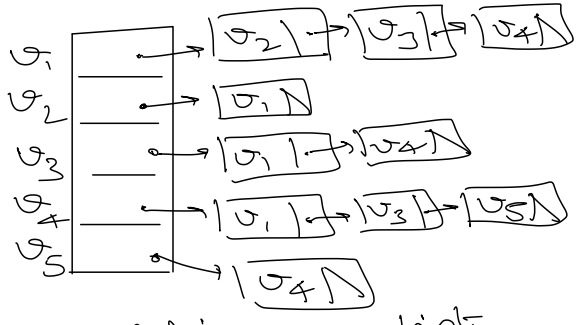
In Graph → adjacency information.

if two vertex are connected by an edge.

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	1	0
v_2	1	0	0	0	0
v_3	1	0	0	1	0
v_4	1	0	1	0	1
v_5	0	0	0	1	0

Adjacency Matrix

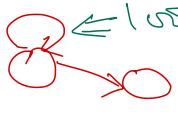
Space: $|V| \times |V| = O(V^2)$



Adjacency list

Space: $|V| + |E| = O(V+E)$

→ Graph with N vertices. There are no loops



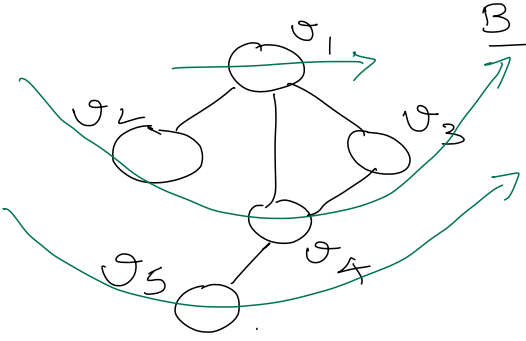
How many max edges can be there?

$$\frac{n(n-1)}{2}$$

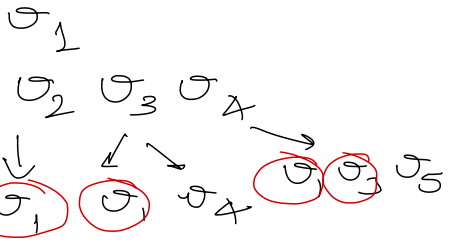
Graph Traversal

→ BFS : Breadth First Search

→ DFS : Depth First Search.



BFS



→ Traversal in tree starts from root.

→ Traversal in graph starts with any vertex.

→ Maintain visited information for each vertex.

	v_1	v_2	v_3	v_4	v_5
visited	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T	~~F~~ T

Start vertex : v_1

v_2 v_3 v_4

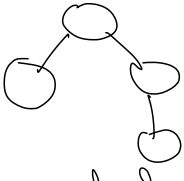
v_5

Queue

~~v_1~~ ~~v_2~~ ~~v_3~~ ~~v_4~~ ~~v_5~~

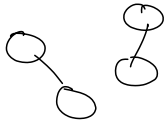
current → ~~v_1~~ ~~v_2~~ ~~v_3~~ ~~v_4~~ ~~v_5~~ v_5

→ Check if graph is connected.



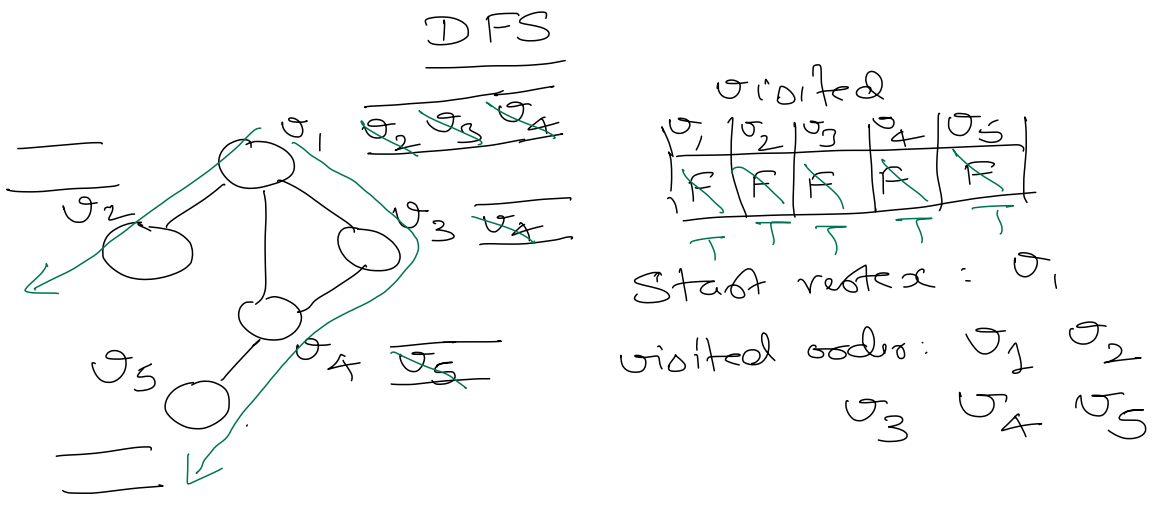
Connected Graph

↳ Path exist between any two pair of vertices.



Disconnected Graph.

After traversal if all vertex are visited then graph is connected.



DFS - For a vertex, visit one of its adjacent vertex and repeat DFS until we reach a vertex with no more adjacent vertex not yet visited then backtrack.

DFS(startVertex) - Mark every vertex as not visited.
- if startVertex is not visited then
  - Mark startVertex as visited.
  - Process startVerted.
- For every adjacent vertex to startVertex
  - if vertex is not yet visited then
   - DFS(vertex)

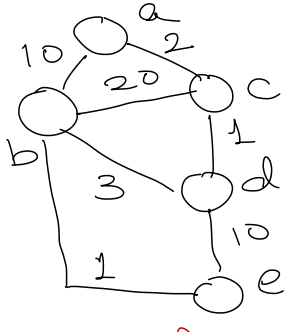
BFS - For a vertex, visit all its adjacent vertex not yet visited. Then repeat the same process for each of the adjacent vertex.

BFS(startVertex)
- Mark every vertex as not visited.
- Initiate the traversal with startVertex => Add startVertex to queue.
- While there are vertex to be visited => queue is not empty do
  - Get a vertex from queue.
  - if vertex is not yet visited then
    - Mark vertex as visited.
    - Process the vertex.
    - Add all vertices adjacent to this vertex that are not yet visited, to queue.

Shortest Path ⇒ weighted graph.

→ Map : Shortest distance from src → dest.

→ LinkedIn : 1st / 2nd / 3rd Contact ? ShortestPath.



Shortest Path between a & b

edge (a, b) = 10

a → c → d → b = 6
2 + 1 + 3

Brute Force approach ⇒ O(n!) → Number of vertices.

↳ Find all possible solutions. Then pick the most optimal.

a − b = 10

a − c − b = 22
2 + 20

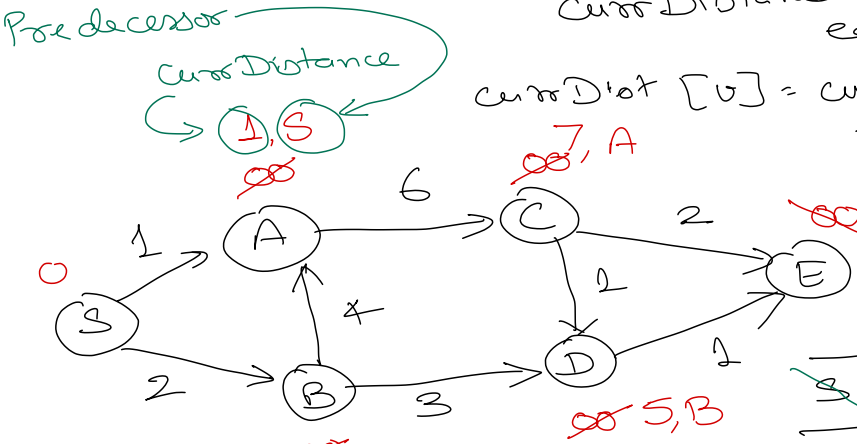
a − c − d − b = 6
2 + 1 + 3

a − c − d − e − b = 14
2 + 1 + 10 + 1

Dijkstra's shortest path ⇒ work if edge weight is non-negative.

Greedy approach

→ All vertex are very far away from source.
→ Set distance of source as 0.
→ Create a set of all vertices.
→ While there are vertex to be processed.
  → Find vertex u with smallest distance.
  → Find all its adjacent vertex v and update their distance.
    if currDistance[v] > currDistance[u] + edge(u, v) then
    currDist[v] = currDist[u] + edge(u, v)



u →	S	A	B	D	E
v →	A B	C	A D	E	D E

S A B C D E

Shortest Path: S → B → D → E weight: 6

Dijkstra's Shortest Path(startVertex, endVertex)

- Set currentDistance for all vertices as infinity. → O(V)
- Set currentDistance for startVertex as 0 → 1
- Create a list of all vertices in graph. → O(V)
- while list contains a vertex do → O(V)
  - Get the vertex, u, from list with smallest distance. → O(V)
  - For every vertex v, adjacent to u do
    // Update the distance of v, if required.
    - distanceToVviaU = currentDistance of u + weight of edge(u, v) → 1
    - if currentDistance of v > distanceToVviaU then
      - Set currentDistance of v to distanceToVviaU → 1
      - Set predecessor of v to u. → 1

→ MinHeap O(log V)

$V + 1 + V + V * (V + 1 + 1)$

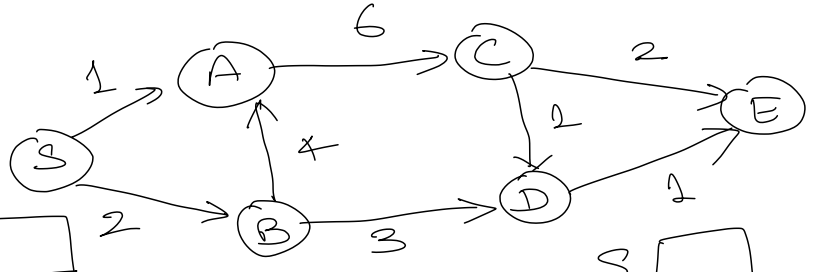
$= 2V + 1 + V * (2 + V)$

$= 2V + 1 + 2V + V^2$

$= V^2 + 4V + 1 \Rightarrow O(V^2)$

U → S
V → A B

A
C



Current Distance

S	0	
A	~~∞~~	1
B	~~∞~~	2
C	~~∞~~	7
D	∞	
E	∞	

S	
A	S
B	S
C	A
D	
E	

Predecessor

~~S~~ ~~A~~ B C D E

List of vertices

Bellman-Ford Shortest Path Algo

↳ It works if graph has negative edge weights.

O(V)

→ All vertex are very far away from source.

→ Set distance of source as 0. → 1

→ Create a list of all edges in graph. ↳ O(E)

→ For |V| - 1 times: O(V-1)

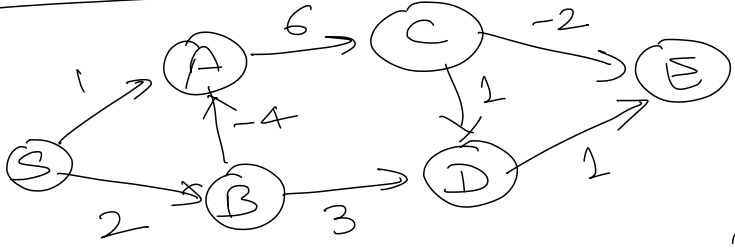
→ For each edge (u, v) E

→ distanceToV viaU = currDistance of u + edge(u, v) 2

→ if currDistance of v > distanceToV viaU then 1

→ currDistance of v = distanceToV viaU 1

→ Predecessor of v = u. 1



Edges

SA CD
SB CE
BA DE
AC
BD

current Distance

S	0	
A	∞	4 → 2
B	∞	2
C	∞	4
D	∞	5
E	∞	2

Predecessor

S	
A	~~B~~
B	S
C	A
D	B
E	C

$V + 1 + E + (V-1) * (E * (1++1))$

$V + 1 + E + (V-1) * (3E)$

$V + 1 + E + 3VE - 3E$

$3VE + V - 2E + 1 \Rightarrow O(VE)$

Max edges in graph $= \frac{V(V-1)}{2}$

$\approx V^2$

$\Rightarrow O(V^3)$