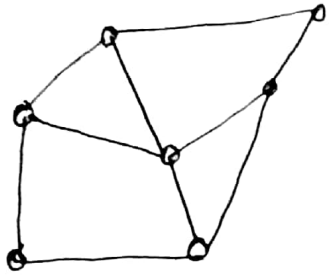
Sparse VS Dense Graphs

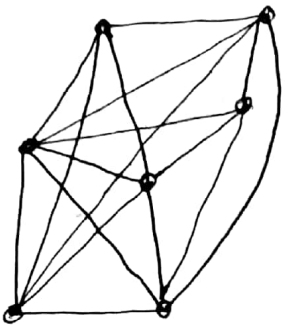
Sparse Graph:

Sparse graph is a graph in which the number of edges is close to the minimal number of edges.

- Graphs are sparse when only a small fraction of the possible number of vertex pairs actually have edges defined between them.
- Graphs are usually sparse due to application-specific constraints.
- Road networks must be sparse because of road junctions



Sparse



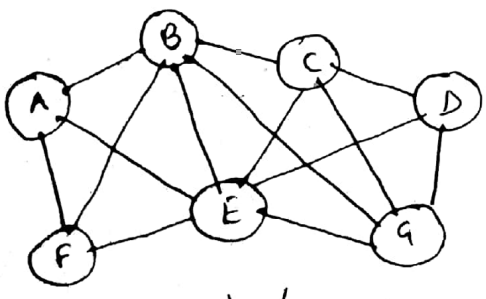
Dense

Dense Graph

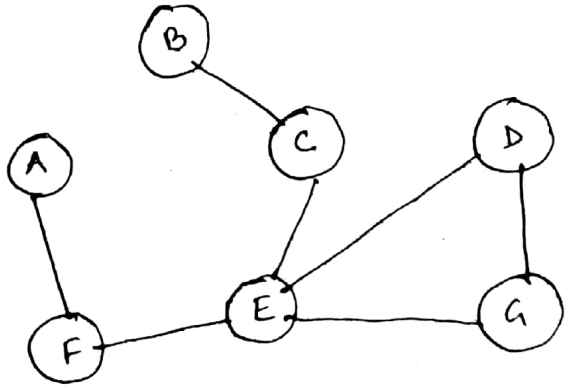
Dense graph is a graph in which the number of edges is close to the maximal number of edges.

Types of Graphs

Dense VS. Sparse



Dense graphs (many edges between nodes)



Sparse graphs (few edges between nodes)

- If the graph has n vertices (nodes)
  - Maximum no. of edges is n^2.
- In dense graphs number of edges is close to n^2.
- In sparse graphs number of edges is close to n.

* A dense graph is a graph where the number of edges is relatively large compared to number of nodes.

When is a graph a dense graph?

We could use

- $|E| = O(|V|^2)$ – Dense
- $|E| = O(|V|)$ – Sparse

* Examples of

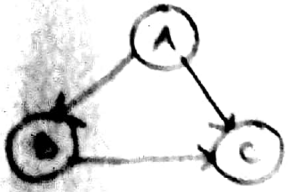
– Dense Graphs:

- Each node is connected to at least 25% of other nodes

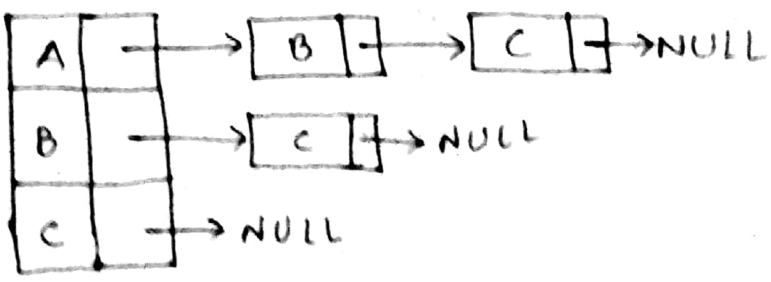
– Sparse Graphs:

- Each node is connected to only a constant number of other nodes.

Memory Requirements or space complexity

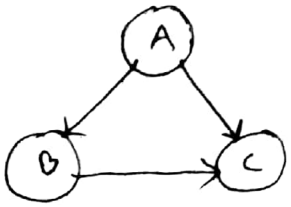


Adjacency List – $O(|V|+|E|)$



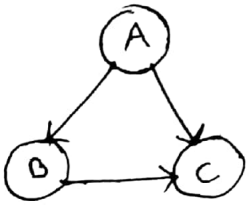
Adjacency Matrix

Adjacency matrix — $O(|V|^2)$



	A	B	C
A		1	1
B			1
C			

Reachability



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

Given a node u in V, find all the nodes v in V that are reachable from u.

Example:

$u = A$

Reachable(u) = Reachable $(A) = \{B, C\}$

Reachability Algorithms

Two algorithms

- BFS (Breadth First Search (BFS))
  - Explore the nodes in an orderly manner
  - Look at the nodes that are closest to source
  - Then look at their neighbors.

  (Uses queue)

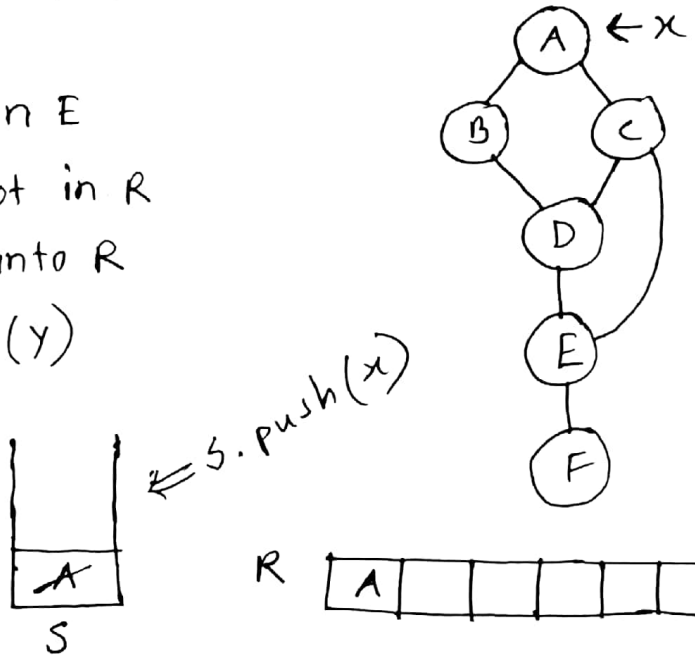
- Depth First Search (DFS)
  - Explore the nodes by going deeper and deeper into the graph
  - Use backtracking to try different paths (stack)

DFS Algorithm (Iterative)

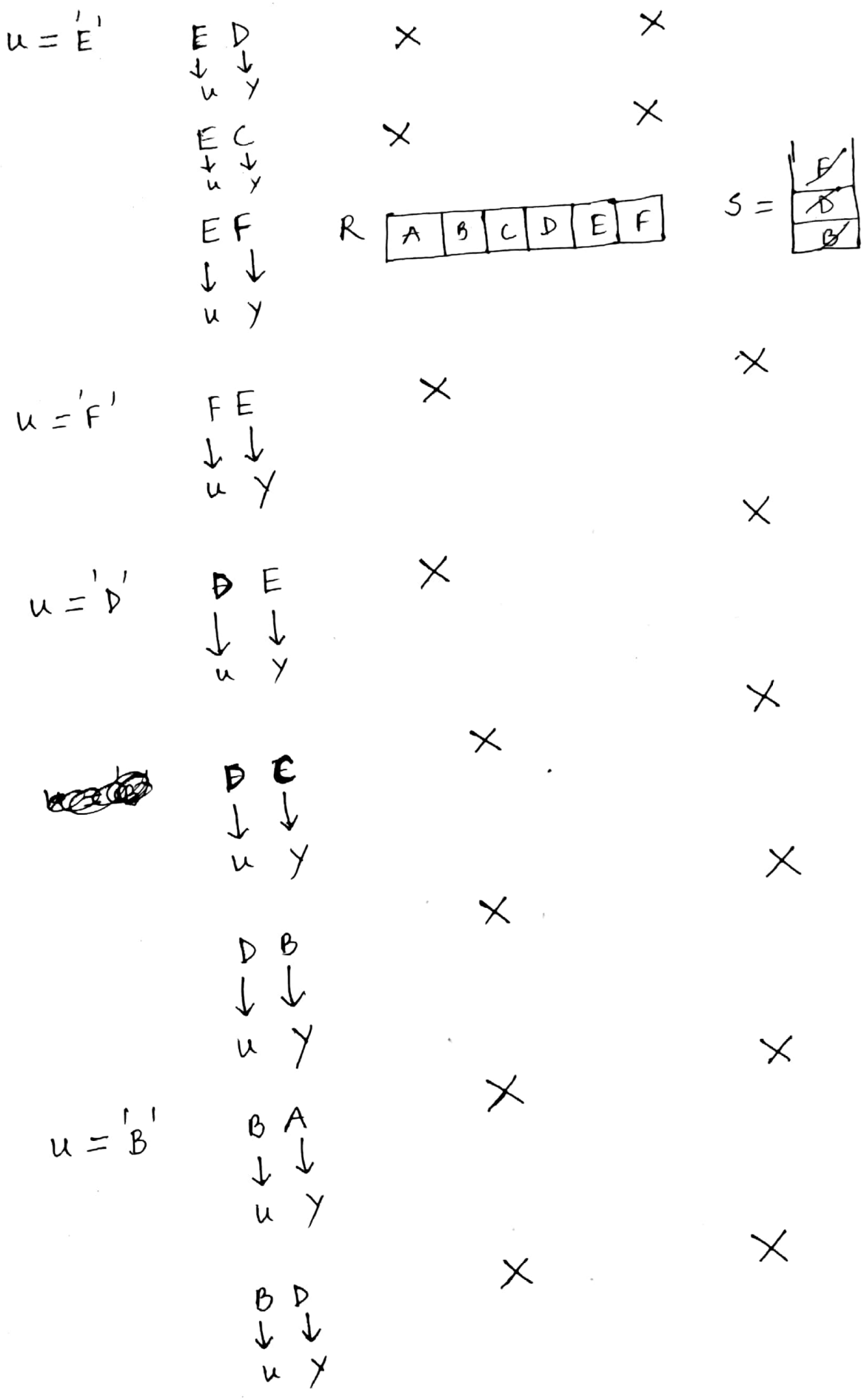
☐ Let R be the sets of vertices reachable from starting node x, Let S be a stack

Algorithm

```
DFS(vertex x){
    S.push(x)
    put x into R
    while(S is not empty){
        u = S.pop()
        for all (u,y) in E
         { if y is not in R
            { put y into R
              S.push(y)
            }
         }
    }
}
```



u = S.pop()	Find (u,y)	Put y into R	S.push(y)
u = 'A'	A B ↓ ↓ u y	R A B	S= B
	A C ↓ ↓ u y	R A B C	S= E B
u = 'C'	C A ↓ ↓ u y	×	×
	C D ↓ ↓ u y	R A B C D	S = D B
	C E ↓ ↓ u y	R A B C D E	S = E D B

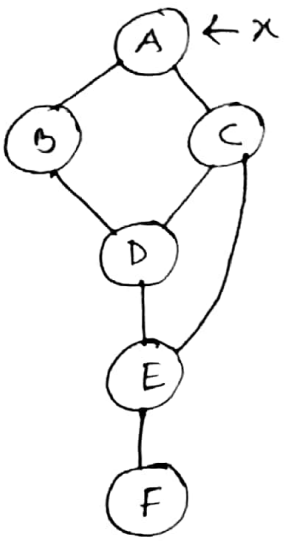


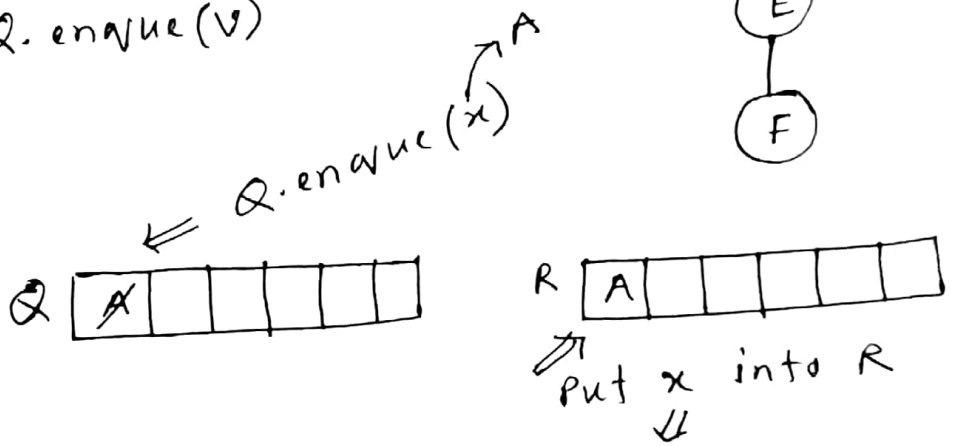
DFS Sequence = ACEFDB

Breadth First Search (BFS)

```
BFS(node x){
  Q.enque(x) // Assume Q is a Queue
  put x into R // R is the set of vertices visited in BFS
  while (Q is not empty){
    u = Q.deque()
    for all neighbors v of u {
      if v is not in R{
        put v into R
        Q.enque(v)
      }
    }
  }
}
```

BFS sequence = R = ABCDEF





u = Q.deque()	Find v	Put v in R	Q.enque(v)
u = 'A'	B	R A B	Q B
	C	R A B C	Q B C
u = 'B'	A	×	×
	D	R A B C D	Q D
u = 'C'	A	×	×
	D	×	×
	E	R A B C D E	Q E
u = 'D'	D	×	×
u = 'E'	C	×	×
	D	×	×
	F	R A B C D E F	F
u = 'F'	F	×	×

DFS