Breadth First Search (BFS)

(Similar to level order of a binary tree)

↓ converted into

A Tree (A graph without any cycle / without any closed edges)

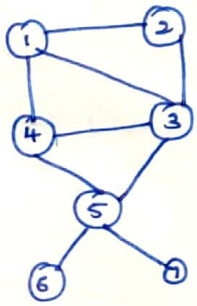
1) Visiting (visiting all the vertices)

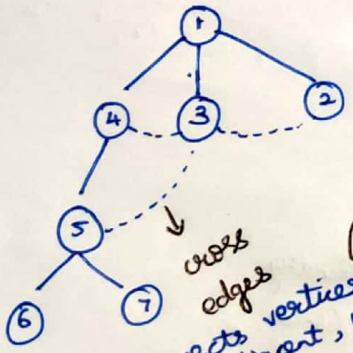
2) Exploration (Exploring all the vertices (i.e adjacent vertices)

BFS: 1, 4, 3, 2, 5, 6, 7

Queue: ~~1~~, ~~4~~, ~~3~~, ~~2~~, ~~5~~, 6, 7

} Visit the vertex & insert in queue





BFS spanning tree

cross edges (connects vertices are adjacent, not very far)

Result of BFS ⇒ 1, 4, 3, 2, 5, 6, 7

* Take out a vertex from queue and start exploring
* Explore in any order of the ~~of~~ vertices in queue.

Time ⇒ O(n)
Analytical Time

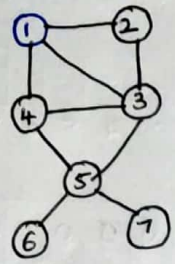
Note:

* Vertices can be explored in any order. For a given graph ~~there~~ can be various BFS.
* Importance is that NOT the order you are visiting, importance is that we are visiting all the vertices.

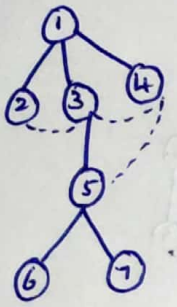
Scanned with CamScanner

BFS ~~trees~~ graphs

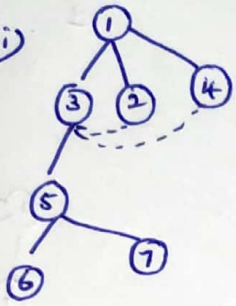
Eg) 1:



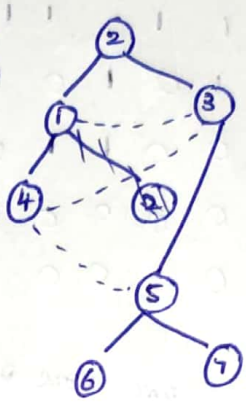
i)



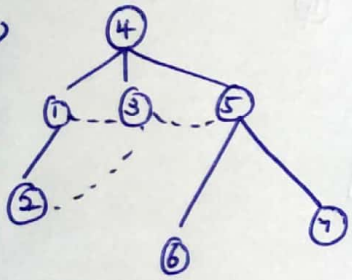
ii)



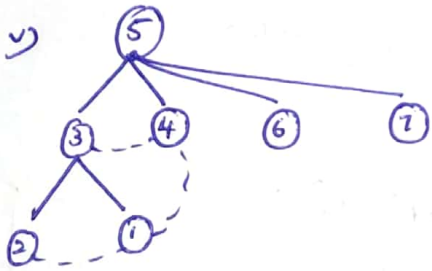
iii)



iv)



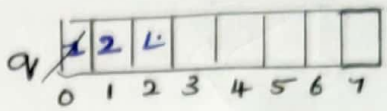
v)

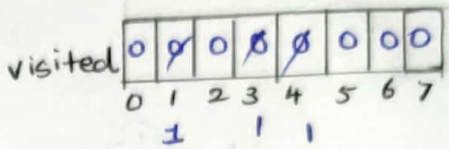


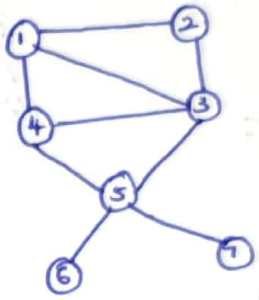
Scanned with CamScanner

Code for BFS graph

	0	1	2	3	4	5	6	7
0								
u → 1		0	1	1	1	0	0	0
2		1	0	1	0	0	0	0
3		1	1	0	1	1	0	0
A ⇒ 4		1	0	1	0	1	0	0
5		0	0	1	1	0	1	1
6		0	0	0	0	1	0	0
7		0	0	0	0	1	0	0







i) First, visited => 0, since None of the graph vertice is visited.

Starting vertice

```
void BFS (int i)
{
    printf ("%d", i);
    visited [i] = 1;
    enqueue(q, i);
    while (! isEmpty(q))
    {
        u = dequeue (q);
        for (v=1; v<=n; v++)
            { if (A[u][v]==1) && visited[v]==0)
                {
                    printf ("%d", v);
                    visited [v] = 1;
                    enqueue (q, v);
                }
            }
    }
```

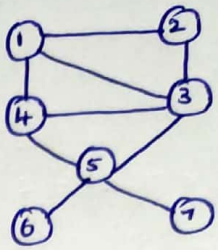
$O(n^2)$

If there is an edge and not visited

Scanned with CamScanner

1) visiting
2) Exploring

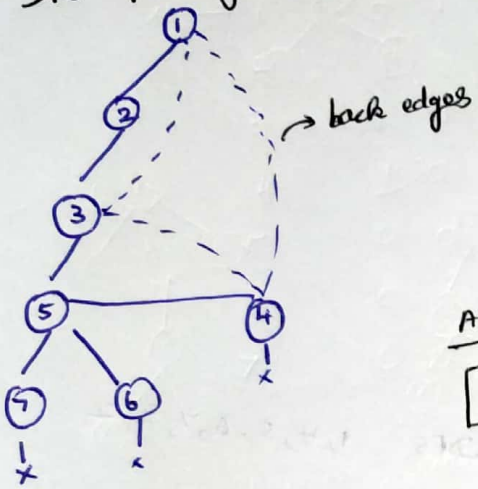
Depth first search



DFS: 1, 2, 3, 5, 7, 6

stack

DFS spanning tree



Analytical time

O(n)

*) Wherever we come to a New vertex, stop exploring the current vertex and explore the new vertex.

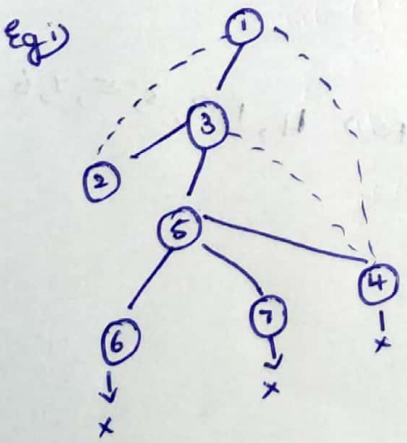
*) So put the current vertex into the stack so as to explore it in the future.

*) If we visited all adjacent / or NO adjacent vertice then it is completely explored

Exploring 7 is done so pop 5 from stack & explore the remaining vertice of 5

DFS: 1, 3, 5, 6, 7, 4, 2

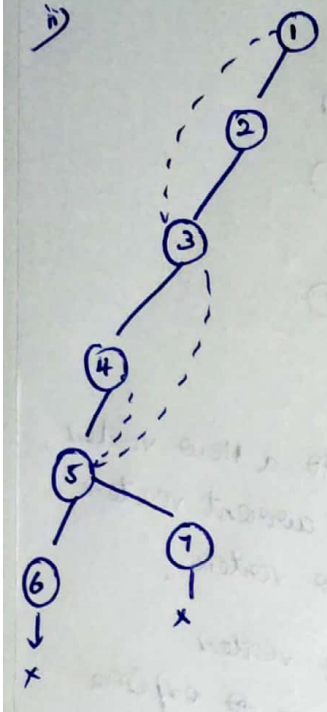
Eg:)



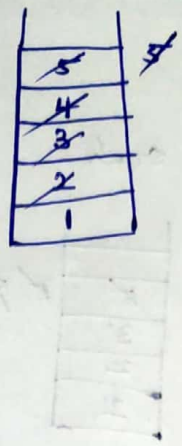
stack

Scanned with CamScanner

ii)

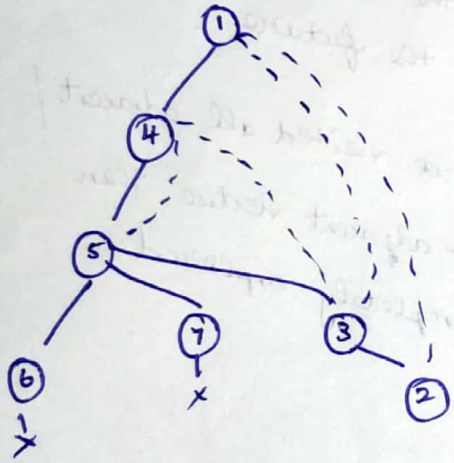


Stack

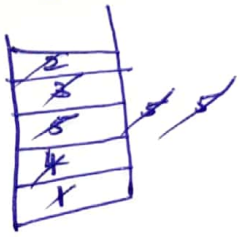


DFS: 1, 2, 3, 4, 5, 6, 7

iii)

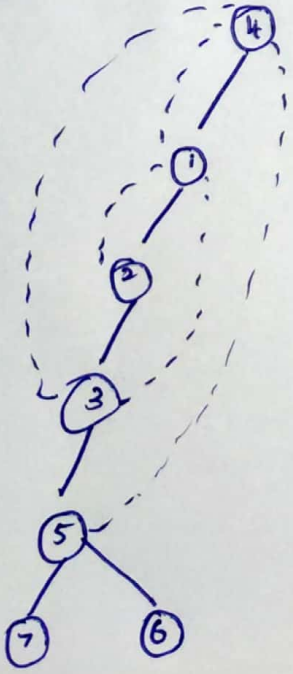


Stack

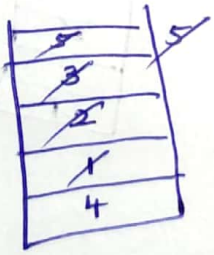


DFS: 1, 4, 5, 6, 7, 3, 2

iv)



Stack



DFS: 4, 1, 2, 3, 5, 7, 6

Scanned with CamScanner

Program for DFS

	0	1	2	3	4	5	6	7
0								
u → 1		0	1	1	1	0	0	0
2		1	0	1	0	0	0	0
3		1	1	0	1	1	0	0
A ⇒ 4		1	0	1	0	1	0	0
5		0	0	1	1	0	1	1
6		0	0	0	0	1	0	0
7		0	0	0	0	1	0	0

visited

0	1	2	3	4	5	6	7
0	1	1	0	0	0	0	0

```
void DFS (int u)
{
    if (visited[u] == 0)
    {
        printf ("%d", u);
        visited[u] = 1;
        for (v=1; v<=n; v++)
        {
            if (A[u][v] == 1 && visited[v] != 0)
            {   DFS (v);
            }
        }
    }
}
```

Scanned with CamScanner