ANSWER - 01

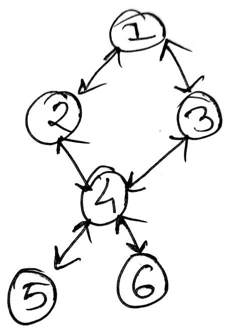
Adjacency Matrix :

$$\begin{array}{c|ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}$$

Adjacency List :

1 = {2, 3}
2 = {1, 4}
3 = {1, 4}
4 = {2, 3, 5, 6}
5 = {4}
6 = {4}

Graph Representation :



ANSWER - 02

```
#include<bits/stdc++.h>
using namespace std;
```

```
int power(int x, int n)
{
   if(n == 0)
      return 1;
   return x * power(x, n-1);
}


int main()
{
   int x, n;cin>>x>>n;
   cout<<power(x, n)<<endl;
   return 0;
}
```

ANSWER - 03

The difference between BFS and DFS algorithms is :

1. BFS searches level wise, DFS searches recursively.
2. BFS uses a Queue to find the shortest path, DFS uses a Stack to find the shortest path
3. BFS requires more memory space, DFS requires less memory space.
4. BFS is slower than DFS, DFS is faster than BFS.
5. BFS is better when target node is closer to source node. DFS is better when target node is far from source node.

ANSWER - 04

BFS and DFS are two popular algorithms used for traversing/searching a graph or a tree. Here's how each of these algorithms works :

BFS and how it works :

BFS means Breadth First Search. BFS starts at the root node of the graph or tree and visits all the nodes at the current level before moving on to the next level. It uses a queue data structure to keep track of the nodes to be visited.

The steps of BFS are as follows:

1. Enqueue the starting node into a queue.
2. Mark the starting node as visited.
3. Dequeue the starting node from the queue.
4. For each adjacent node of the dequeued node, if it has not been visited yet, mark it as visited and enqueue it into the queue.
5. Repeat steps 3-4 until the queue is empty.

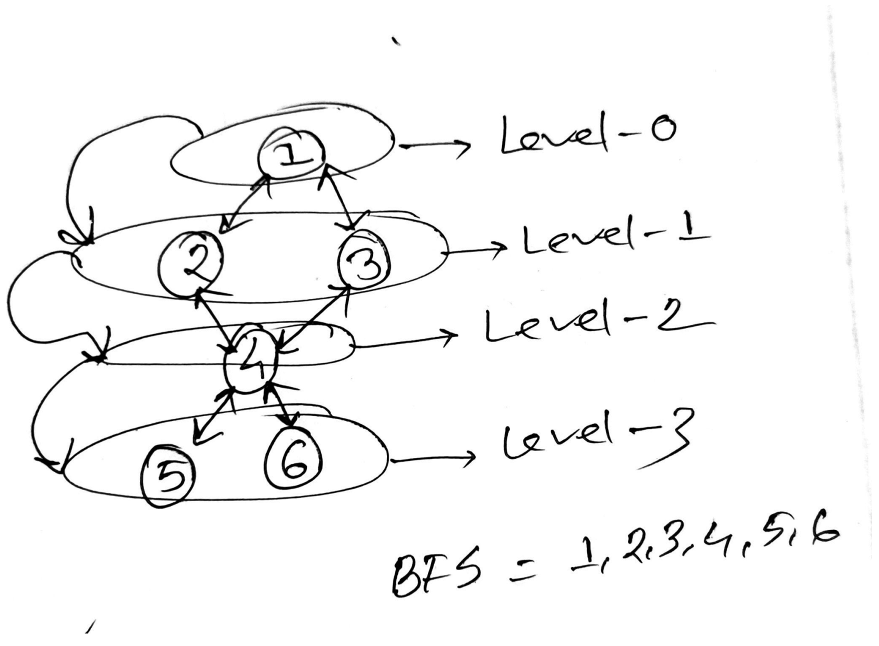


Fig : BFS Traversal

DFS and how it works :

DFS means Depth First Search. DFS starts at the root node of the graph or tree and explores as far as possible along each branch before backtracking.

The steps of DFS are as follows:

1. Select a node
2. Explore non visited node.
3. Pause current node
4. Move to non visited node while exploring.
5. While exploring if any visited node found, we will ignore them
6. While exploring if no non-visited node found, we will go back to the previous node where we came from and active that node which was paused previously.
7. Repeat 2-6 until all node is visited.

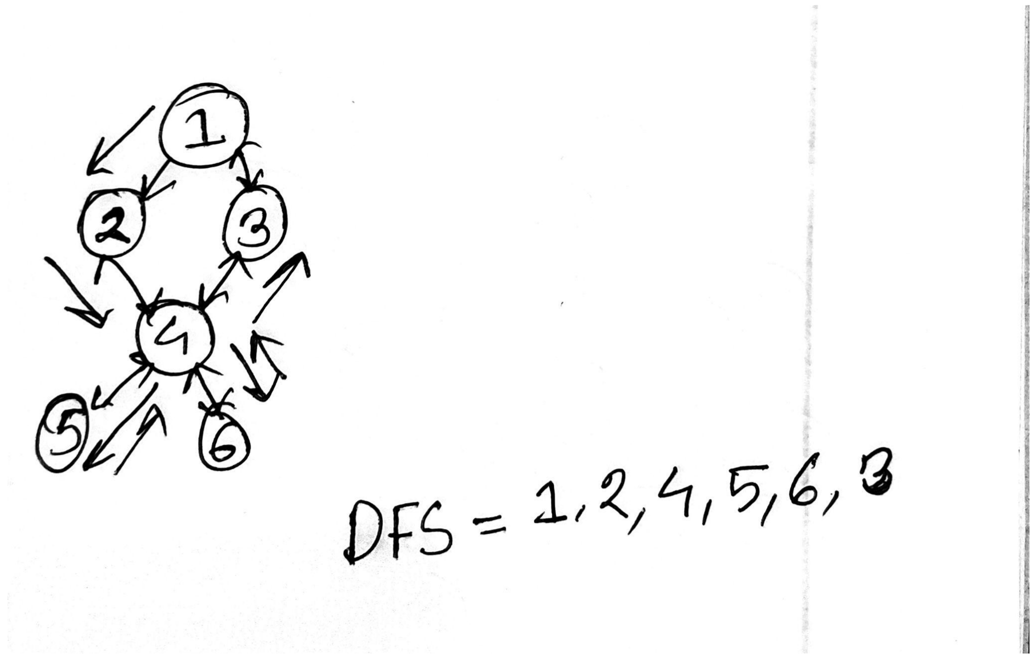
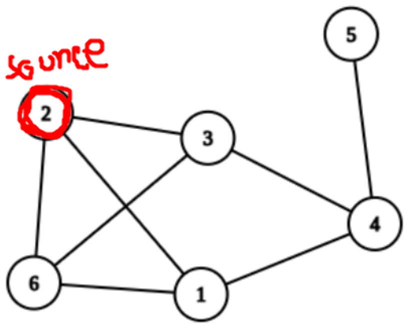


Fig : DFS Traversal

ANSWER - 05

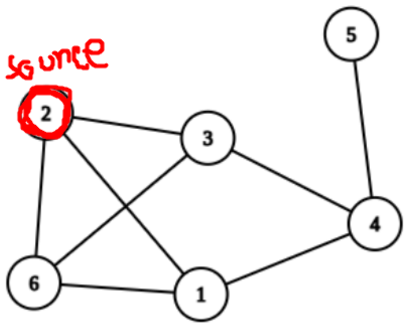


BFS Traversal :

Steps :

1. Select 2 as source node.
2. Initiate a queue and visited array.
3. Enqueue 2 as selected node. and mark selected node as visited in visited arry.
4. Dequeue selected node from the queue and print in output.
5. Explore selected nodes's all adjacent node. which are : 6, 1, 3
6. Enqueue adjacent nodes in the queue. Queue will be [6, 1, 3]
7. Selected node will be queue's top element. Selected node is 6.
8. Dequeue 6 from queue and print it the output.
9. Explore 6 node's adjacent nodes. But if any visited node found we will ignore them. So 1 is ignored.
10. 1 will be selected and its adjacent node is 4.so 4 will be enqueued. 1 will be dequeued and print in the output.
11. Exploring 3. We will find a alread visited node 4. So we will ignore it. Dequeue 3 and pint in the output.
12. Queue top node will be 4 and its adjacent node is 5.Enqueue 5
13. 4 will be dequeued and print in the output.
14. Queue top will be 5 and will be dequeued and print int the output.

OUTPUT : 2, 6, 1, 3, 4, 5



DFS Traversal :
Steps :

1. Initiate visited array .
2. Source node 2,
3. Mark 2 as visited and print it
4. we will find 6.
5. We will pause 2 and move on its adjacent node 6.
6. Mark 6 as visited and print it
7. While exploring 6 we will find 1.
8. Pause 6 and move on its adjacent node 1.
9. Mark 1 as visited and print it
10. While exploring 1, we will find 4.
11. Pause 1and move its adjacent node 4.
12. Mark 4 as visited and print it.
13. While exploring 4, we will find 5.
14. Pause 4 and move to 5.
15. Mark 5 as visited and print it.
16. While exploring 5, there will be no non visited node to explore.
17. Come back to previous node and active it.
18. While exploring 4 again we will find 3 as non visited node.
19. Pause 4 and move to 3.
20. Mark 3 as visited and print it.

OUTPUT : 2, 6, 1, 4, 5, 3