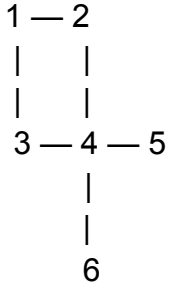
Answer Script

Question No. 01
Convert the following Adjacency Matrix into an Adjacency List and draw the graph. **20** (no need to code)   0 1 2 3 4 5 6 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
Answer No. 01
Adjacency Matrix to an Adjacency list: 0: [ ] 1: [2, 3] 2: [1, 4] 3: [1, 4] 4: [2, 3, 5, 6] 5: [4] 6: [4] **Graph:**   

Question No. 02

You are given two positive integers n and m . Now calculate the value of n to the power m using recursion.Write a C++ program for it.
20

Sample Input	**Sample Output**
2 4	16

Answer No. 02

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

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

int main() {
        int n, m;
        cin >> n >> m;
        cout << power(n, m) << "\n";
        return 0;
}
```

Question No. 03

What is the difference between BFS and DFS algorithms? (At least Five) **20**

Answer No. 03

BFS	DFS
BFS stands for Breadth First Search.	DFS stands for Depth First Search.
BFS uses Queue data structure for finding the shortest path.	DFS uses Stack data structure.
BFS is slower than DFS.	DFS is faster than BFS.
BFS requires more memory space.	DFS requires less memory space.
BFS is implemented using FIFO (First In First Out) principle.	DFS is implemented using LIFO (Last In First Out) principle.

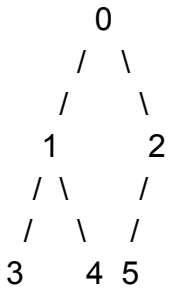
Question No. 04
What is BFS and how does it work? What is DFS and how does it work? (With Figure) **20**

Answer No. 04

BFS: BFS means **Breadth First Search**. BFS is an algorithm for traversing or searching tree or graph data structures. It explores all the nodes at the present depth before moving on to the nodes at the next depth level.

Working procedure:
Step 1: Initializing a queue and marking the source vertex as visited.
Step 2: Adding the source vertex to the queue.
Step 3: While the queue is not empty, remove the first vertex from the queue and add its unvisited neighbors to the queue and mark them as visited.
Step 4: Repeat step 3 until the queue is empty.

Example:



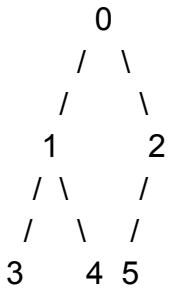
Initialized queue	Visited array
{ 0 }	[ ]
{ 1, 2 }	[0]
{ 2, 3, 4 }	[0, 1]
{ 3, 4, 5 }	[0, 1, 2]
{ 4, 5 }	[0, 1, 2, 3]
{ 5 }	[0, 1, 2, 3, 4]
{ }	[0, 1, 2, 3, 4, 5]

DFS: DFS means **Depth First Search.** DFS is an algorithm for traversing or searching tree or graph data structures which uses the idea of backtracking. It explores all the nodes by going forward if possible or uses backtracking.

Working procedure:

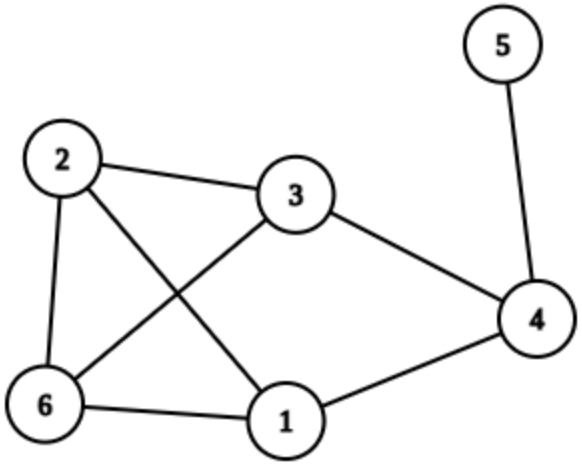
Step 1: Initialize a stack and mark the source vertex as visited.
Step 2: Push the source vertex onto the stack.
Step 3: While the stack is not empty, pop a vertex from the stack and explore its unvisited neighbors.
Step 4: Mark each unvisited neighbor as visited and push it onto the stack.
Step 5: Repeat step 3 and 4 until the stack is empty.

Example:



Initialized Stack	Visited array
{ 0 }	[ ]
{ 1, 2 }	[0]
{ 3, 4, 2 }	[0, 1]
{ 4, 2 }	[0, 1, 3]
{ 2 }	[0, 1, 3, 4]
{ 5 }	[0, 1, 3, 4, 2]
{ }	[0, 1, 3, 4, 2, 5]

Question No. 05



Perform BFS and DFS Traversal on the following graph and write the traversal output. Choose node 2 as the source. You must write all steps you perform for doing BFS and DFS Traversal on the following graph.

Answer No. 05

BFS Traversal:

Steps	Queue	Visited Array	Output
1	[2]	[2]	[ ]
2	[1, 3, 6]	[2, 1, 3, 6]	[2]
3	[3, 6, 4]	[2, 1, 3, 6, 4]	[2, 1]
4	[6, 4]	[2, 1, 3, 6, 4]	[2, 1, 3]
5	[4]	[2, 1, 3, 6, 4]	[2, 1, 3, 6]
6	[5]	[2, 1, 3, 6, 4, 5]	[2, 1, 3, 6, 4]
7	[ ]	[2, 1, 3, 6, 4, 5]	**[2, 1, 3, 6, 4, 5]**

BFS Traversal Output : [2, 1, 3, 6, 4, 5]

DFS Traversal:

Steps	Stack	Visited Array	Output
1	[2]	[2]	[2]
2	[2, 1]	[2, 1]	[2, 1]
3	[2, 1, 4]	[2, 1, 4]	[2, 1, 4]
4	[2, 1, 4, 5]	[2, 1, 4, 5]	[2, 1, 4, 5]
5	[2, 1, 4]	[2, 1, 4, 5]	[2, 1, 4, 5]
6	[2, 1, 4, 3]	[2, 1, 4, 5, 3]	[2, 1, 4, 5, 3]
7	[2, 1, 4, 3, 6]	[2, 1, 4, 5, 3, 6]	[2, 1, 4, 5, 3, 6]
8	[2, 1, 4, 3]	[2, 1, 4, 5, 3, 6]	[2, 1, 4, 5, 3, 6]
9	[2, 1, 4]	[2, 1, 4, 5, 3, 6]	[2, 1, 4, 5, 3, 6]
10	[2, 1]	[2, 1, 4, 5, 3, 6]	[2, 1, 4, 5, 3, 6]
11	[2]	[2, 1, 4, 5, 3, 6]	[2, 1, 4, 5, 3, 6]
12	[ ]	[2, 1, 4, 5, 3, 6]	**[2, 1, 4, 5, 3, 6]**

DFS Traversal Output : [2, 1, 4, 5, 3, 6]