**Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS):**

There are two common ways to traverse a graph, BFS and DFS. Considering a Tree (or Graph) of huge height and width, both BFS and DFS are not very efficient due to following reasons.

DFS first traverses nodes going through one adjacent of root, then next adjacent. The problem with this approach is, if there is a node close to root, but not in first few subtrees explored by DFS, then DFS reaches that node very late. Also, DFS may not find shortest path to a node (in terms of number of edges).

BFS goes level by level, **but requires more space**. The space required by DFS is O(d) where d is depth of tree (So, in case you are using stack to traverse in DFS in iterative way, the stack will contain at most O(h) elements), but space required by BFS is O(n) where n is number of nodes in tree **(Why? Note that the last level of tree can have around n/2 nodes and second last level n/4 nodes and in BFS we need to have every level one by one in queue).**

IDDFS combines depth-first search’s space-efficiency and breadth-first search’s fast search (for nodes closer to root).

/ Returns true if target is reachable from

// src within max\_depth

bool IDDFS(src, target, max\_depth)

for limit from 0 to max\_depth

**//this looping is important**

**//DLS is depth limited Search**

if DLS(src, target, limit) == true

return true

return false

bool DLS(src, target, limit)

if (src == target)

return true;

// If reached the maximum depth,

// stop recursing.

if (limit <= 0)

return false;

foreach adjacent i of src

**//doing dfs here**

if DLS(i, target, limit?1)

return true

return false

**Implementation:**

// C++ program to search if a target node is reachable from

// a source with given max depth.

#include<bits/stdc++.h>

using namespace std;

// Graph class represents a directed graph using adjacency

// list representation.

class Graph

{

int V; // No. of vertices

// Pointer to an array containing

// adjacency lists

list<int> \*adj;

// A function used by IDDFS

bool DLS(int v, int target, int limit);

public:

Graph(int V); // Constructor

void addEdge(int v, int w);

// IDDFS traversal of the vertices reachable from v

bool IDDFS(int v, int target, int max\_depth);

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

// A function to perform a Depth-Limited search

// from given source 'src'

bool Graph::DLS(int src, int target, int limit)

{

if (src == target)

return true;

// If reached the maximum depth, stop recursing.

if (limit <= 0)

return false;

// Recur for all the vertices adjacent to source vertex

for (auto i = adj[src].begin(); i != adj[src].end(); ++i)

if (DLS(\*i, target, limit-1) == true)

return true;

return false;

}

// IDDFS to search if target is reachable from v.

// It uses recursive DFSUtil().

bool Graph::IDDFS(int src, int target, int max\_depth)

{

// Repeatedly depth-limit search till the

// maximum depth.

for (int i = 0; i <= max\_depth; i++)

if (DLS(src, target, i) == true)

return true;

return false;

}

// Driver code

int main()

{

// Let us create a Directed graph with 7 nodes

Graph g(7);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 5);

g.addEdge(2, 6);

int target = 6, maxDepth = 3, src = 0;

if (g.IDDFS(src, target, maxDepth) == true)

cout << "Target is reachable from source "

"within max depth";

else

cout << "Target is NOT reachable from source "

"within max depth";

return 0;

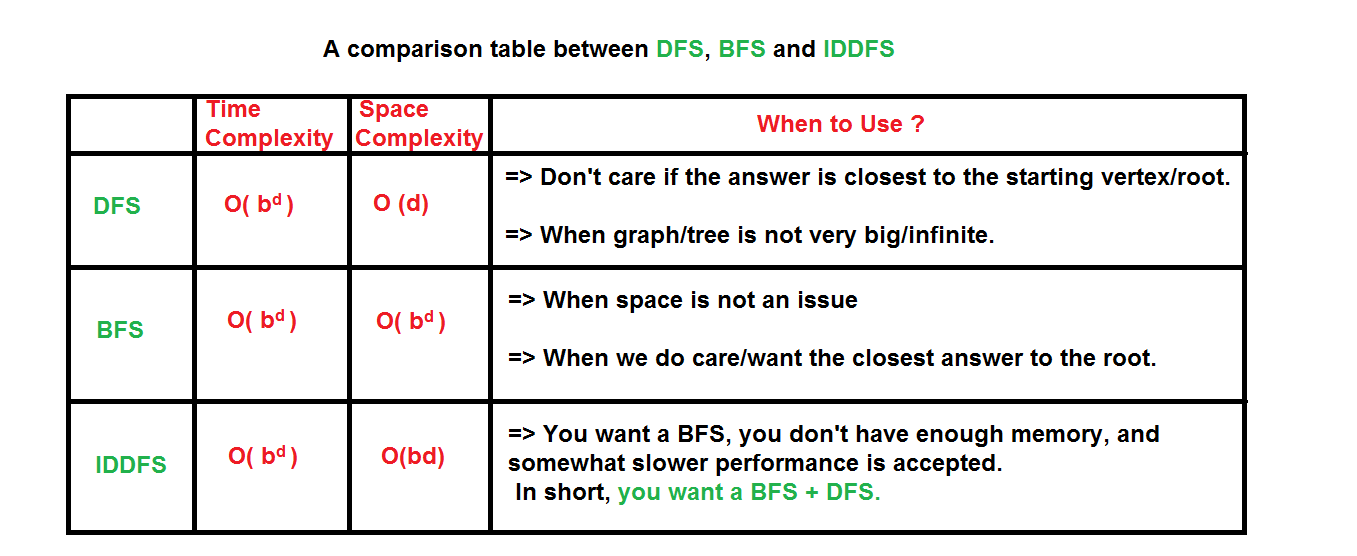
}

**Notes About This Implementation:** here, every vertex has a number associated with it. Now, soruce and target both will be given in form of numbers attached to it.

**Are not we visiting extra nodes:**

An important thing to note is, we visit top level nodes multiple times. The last (or max depth) level is visited once, second last level is visited twice, and so on. It may seem expensive, **but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level. So it does not matter much if the upper levels are visited multiple times. Now, in a graph, we need to check .**

**Comparison Between BFS, DFS and IDDFS:**

****

**Now, b is the branching factor of the tree. D is the depth of thee tree.**