Program Name/Purpose- Iterative Deepening Search(IDS)/ Iterative Deepening Depth First Search(IDFFS)

Project Category- Graphs, Trees

Programming Paradigm/Algorithm Used- Depth Limited Search, a variation of Depth First Search

Data Structures Used- Adjacency List to store graph

Motivation-

To get the best of both the worlds-
a) Depth First Search (DFS)
b) Breadth First Search(BFS)

In short, **DFS + BFS = IDDFS**

IDDFS combines depth-first search's **space-efficiency** and breadth-first search's **completeness**

What does the program do?

An iterative deepening search simply does DFS in a BFS fashion. To simplify, just like BFS we increment the depth/level from the starting node/root at each step and at each step we perform a DFS until that level. Thus, DFS is restricted from going beyond that level in each step. Hence this variation is given a special term- Depth Limited Search (DLS). The heart of the program is DLS itself, i.e- we use DLS as a sub-routine to Iterative Deepening Depth First Search(IDDFS).

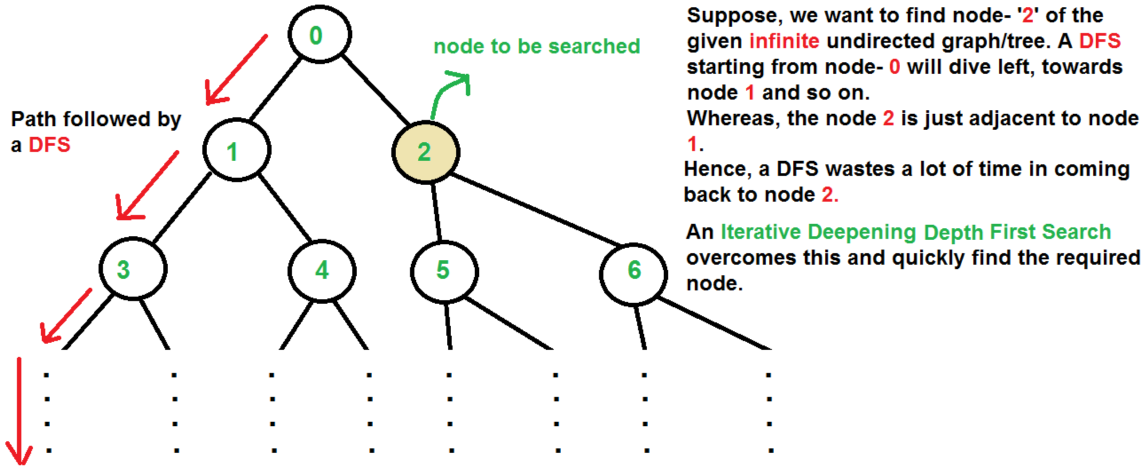
Why the need for a new search algorithm(IDDFS) when there are efficient DFS and BFS already in the business ?

We have already mentioned that IDDFS combines depth-first search's **space-efficiency** and breadth-first search's **completeness**

To elaborate on this consider the comparison of IDDFS with DFS and BFS -

a) IDDFS vs DFS-

Suppose you have an *infinite graph/tree* shown below and you want to search node- 2. Would you choose DFS to search ?



The above picture shows why using **DFS** is a bad idea.
Infact, a DFS on any big graph/tree(can be infinite) to search for a nodes is a bad idea. The reason, is that a DFS might loop through all the nodes of the graph before coming to the node to be searched, even if that node would be right next to the start node !

Thus, one big disadvantage of **DFS** is that it is **not complete** (a thing which is very important in Artificial Intelligence)

b) IDDFS vs BFS-

BFS, uses a lot of memory. Suppose we have a tree having branching factor- **b** (number of children of each node), and its depth- **d.**

A BFS requires a queue to hold the whole level of a tree at each level. Hence at the last level, it stores b^d nodes. Hence, the space complexity of BFS is $O(b^d)$.

An IDDFS overcomes this disadvantage . It has modest space requirements- **O(bd)** **,**to be precise.

Hence, an Iterative deepening depth first search (IDDFS) combines the benefits of breadth-

*first and depth-first search: it has modest memory requirements (linear space complexity), and it is complete and optimal when the branching factor **b** is finite, with asymptotically the same time complexity as DFS and BFS (In reality, IDDFS is somewhat slower than both DFS and BFS as it has a larger constant factor in the asymptotic time complexity expression. This will be discussed in more detail in Time Complexity Section below).*

c) Bonus:-
Does IDDFS finds the shortest distance just like BFS and which data structure IDDFS uses ?

Yes, IDDFS also finds the shortest distance just like BFS.
One more speciality of IDDFS is that this doesn't require any additional data structure, whereas
DFS uses an auxiliary array to keep track of the visited nodes and BFS uses a queue data structure.

A comparison table is shown below for reference-

A comparison table between DFS, BFS and IDDFS

	Time Complexity	Space Complexity	When to Use ?
DFS	$O(b^d)$	O (d)	=> Don't care if the answer is closest to the starting vertex/root. => When graph/tree is not very big/infinite.
BFS	$O(b^d)$	$O(b^d)$	=> When space is not an issue => When we do care/want the closest answer to the root.
IDDFS	$O(b^d)$	O(bd)	=> You want a BFS, you don't have enough memory, and somewhat slower performance is accepted. In short, you want a BFS + DFS.

Algorithm-

For each depth starting from 0 and a given maximum depth and a starting vertex/root

1) If the node's current depth is less than or equal to the maximum depth, then return back
2) Else if the node's current depth is larger than the maximum depth, then perform a DFS on the children of that node.

Explanation of the Algorithm-

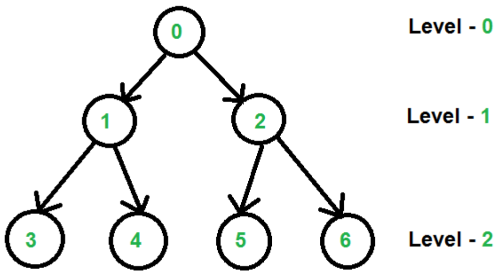
There can be two cases-

a) When the graph has cycles
b) When the graph has no cycles

The latter case gets somewhat trickier as we go deeper and deeper.
Hence we will firstly discuss the first case.

<u>The algorithm for both the cases are same.</u>

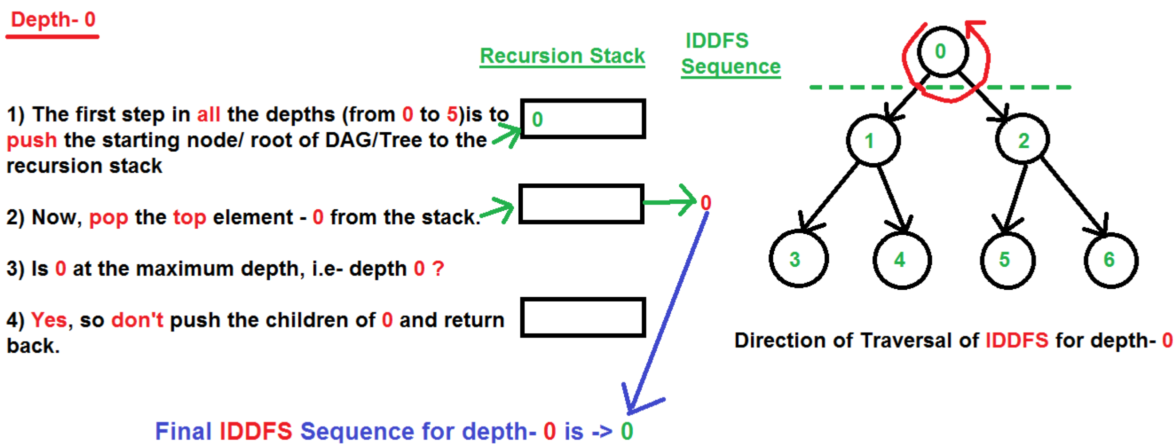
Consider the graph/tree having no cycle as shown below.



We will perform an Iterative Deepening Depth First Search(IDDFS) on the above DAG/ Tree from depth 0 to depth 5. You will see that in DAGs/ Trees, IDDFS prints the same output after depth 3, i.e- the output for depth 3, 4, 5...and so on will be same. In the last part of this article, we will also see another graph having a cycle in which the output of IDDFS for each depth will vary.

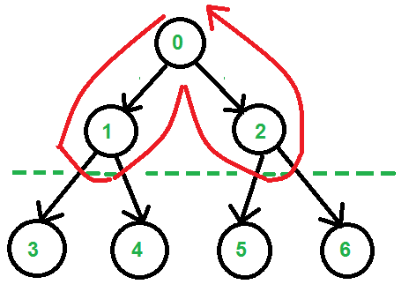
Since IDDFS is a recursive function, we will show what is happening in the recursion stack step by step.

We will show the steps of IDDFS by showing what is happening inside the **recursion stack** in the following figures for each depth from 0 to 5(given).



Depth- 1

	Recursion Stack	IDDFS Sequence
1) The first step in all the depths (from 0 to 5)is to push the starting node/ root of DAG/Tree to the recursion stack	0	
2) Now, pop the top element - 0 from the stack.		0
3) Is 0 at the maximum depth, i.e- depth 0 ?		
4) No, so push the children of 0 in the stack	1 2	
5) Now, pop the top element - 1 from the stack.	2	0,1
6) Is 1 at the maximum depth, i.e- depth 1 ?		
7) Yes, so don't push the children of 1 and return		
5) Now, pop the top element - 2 from the stack.		0,1,2
6) Is 2 at the maximum depth, i.e- depth 1 ?		
7) Yes, so don't push the children of 2 and return		

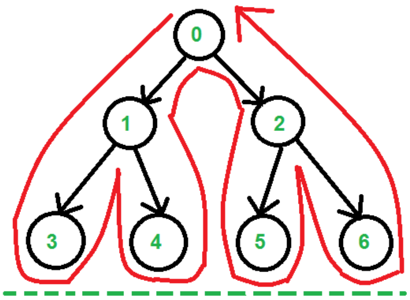


Direction of Traversal of IDDFS for depth- 1

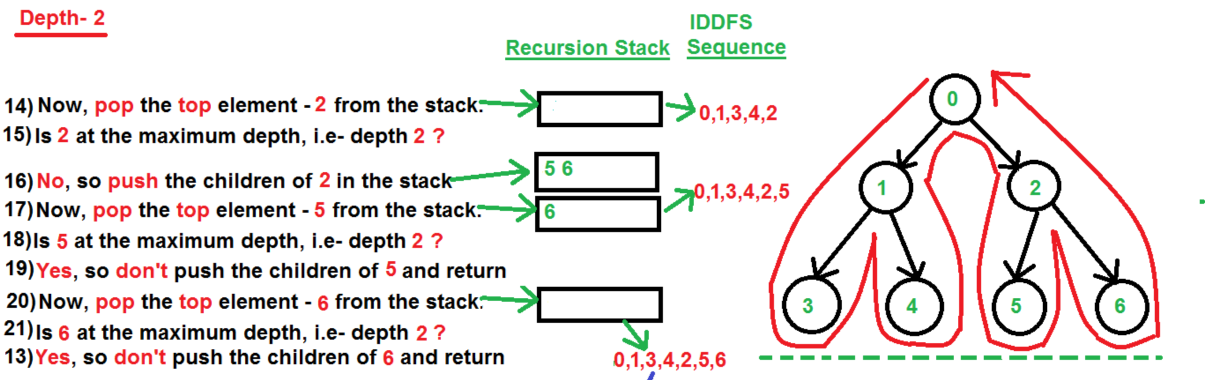
Final IDDFS Sequence for depth- 1 is -> 0, 1, 2

Depth- 2

	Recursion Stack	IDDFS Sequence
1) The first step in all the depths (from 0 to 5)is to push the starting node/ root of DAG/Tree to the recursion stack	0	
2) Now, pop the top element - 0 from the stack.		0
3) Is 0 at the maximum depth, i.e- depth 2 ?		
4) No, so push the children of 0 in the stack	1 2	
5) Now, pop the top element - 1 from the stack.	2	0,1
6) Is 1 at the maximum depth, i.e- depth 2 ?		
7) No, so push the children of 1 in the stack	3 4 2	
8) Now, pop the top element - 3 from the stack.	4 2	0,1,3
9) Is 3 at the maximum depth, i.e- depth 2 ?		
10) Yes, so don't push the children of 3 and return		
11) Now, pop the top element - 4 from the stack.	2	0,1,3,4
12) Is 4 at the maximum depth, i.e- depth 2 ?		
13) Yes, so don't push the children of 4 and return		



Direction of Traversal of IDDFS for depth- 2



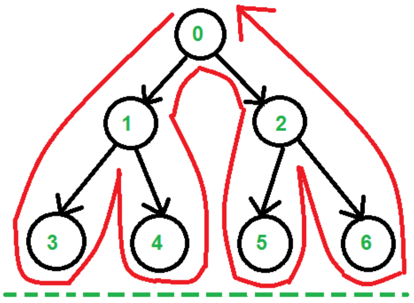
Direction of Traversal of IDDFS for depth- 2

Final IDDFS Sequence for depth- 2 is -> 0,1,3,4,2,5,6

For Depth- 3, 4, 5..and so on, the Iterative Deepening Depth First Search gives the same output as for the depth- 2.

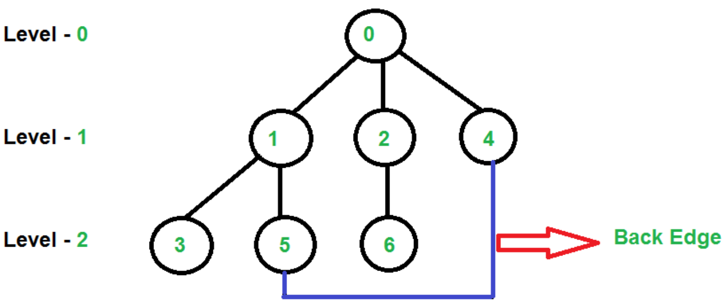
This is because the given DAG/Tree has no cycles and its maximum depth is - 2.

In our next example we will see a case of a graph having cycles which is quite trickier than this case



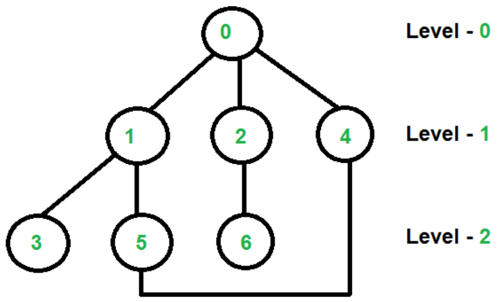
Direction of Traversal of IDDFS for depth >2

Now, we will give an example when the graph has cycles.



Although, at first sight, it may seem that since there are only 3 levels, so we might think that Iterative Deepening Depth First Search of level 3, 4, 5,...and so on will remain same. But, this is not the case. You can see that there is a cycle in the above graph, hence IDDFS will change for level- 3,4,5..and so on.

Depth	Iterative Deepening Depth First Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1



The explanation of the above pattern is left to the readers.

Time Complexity- $O(b^d)$

Suppose we have a tree having branching factor- **b** (number of children of each node), and its depth- **d.**

In an iterative deepening search, the nodes on the bottom level are expanded **once**, those on the next to bottom level are expanded **twice**, and so on, up to the root of the search tree, which is expanded **d+1** times. So the total number of expansions in an iterative deepening search is-

$$(d)b + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + b^d$$

Summation [(d + 1 - i) b^i], from i = 0 to i = d

After evaluating the above expression, we find that ***asymptotically IDDFS takes the same time as that of DFS and BFS***, but it is indeed slower than both of them as it has a higher constant factor in its time complexity expression.

IDDFS is best suited for a complete infinite tree, as iterative deepening visits states multiple times and it may seem wasteful, 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

Space Complexity- $O(bd)$

Points to Note- **Iterative Deepening Depth First Search** is also popular as **Iterative Deepening Search**

References-

https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search