```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        """Add an edge to the graph."""
        self.graph[u].append(v)

    def dls(self, node, target, depth):
        """
        Perform Depth-Limited Search (DLS) from the current node.

        :param node: Current node
        :param target: Target node
        :param depth: Maximum depth to explore
        :return: True if target is found, False otherwise
        """
        if depth == 0:
            return node == target
        if depth > 0:
            for neighbor in self.graph[node]:
                if self.dls(neighbor, target, depth - 1):
                    return True
        return False

    def iddfs(self, start, target, max_depth):
        """
        Perform Iterative Deepening Depth-First Search (IDDFS).

        :param start: Starting node
        :param target: Target node to search for
        :param max_depth: Maximum depth limit for IDDFS
        :return: True if target is found, False otherwise
        """
        for depth in range(max_depth + 1):
            print(f"Searching at depth: {depth}")
            if self.dls(start, target, depth):
                return True
        return False


# Example Usage
if __name__ == "__main__":
    g = Graph()
    # Construct the graph
    g.add_edge(0, 1)
    g.add_edge(0, 2)
    g.add_edge(1, 3)
    g.add_edge(1, 4)
    g.add_edge(2, 5)
    g.add_edge(2, 6)

    start_node = 0
    target_node = 5
    max_depth = 3

    # Perform IDDFS
    if g.iddfs(start_node, target_node, max_depth):
        print(f"Target node {target_node} found within depth {max_depth}")
    else:
        print(f"Target node {target_node} NOT found within depth {max_depth}")
```

```
Searching at depth: 0
Searching at depth: 1
Searching at depth: 2
Target node 5 found within depth 3
```