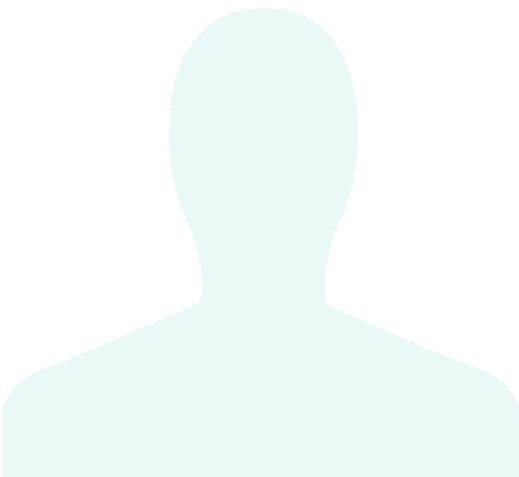


[Coderust 2.0: Faster Coding Interview Preparation using Interactive Visualizations](#)

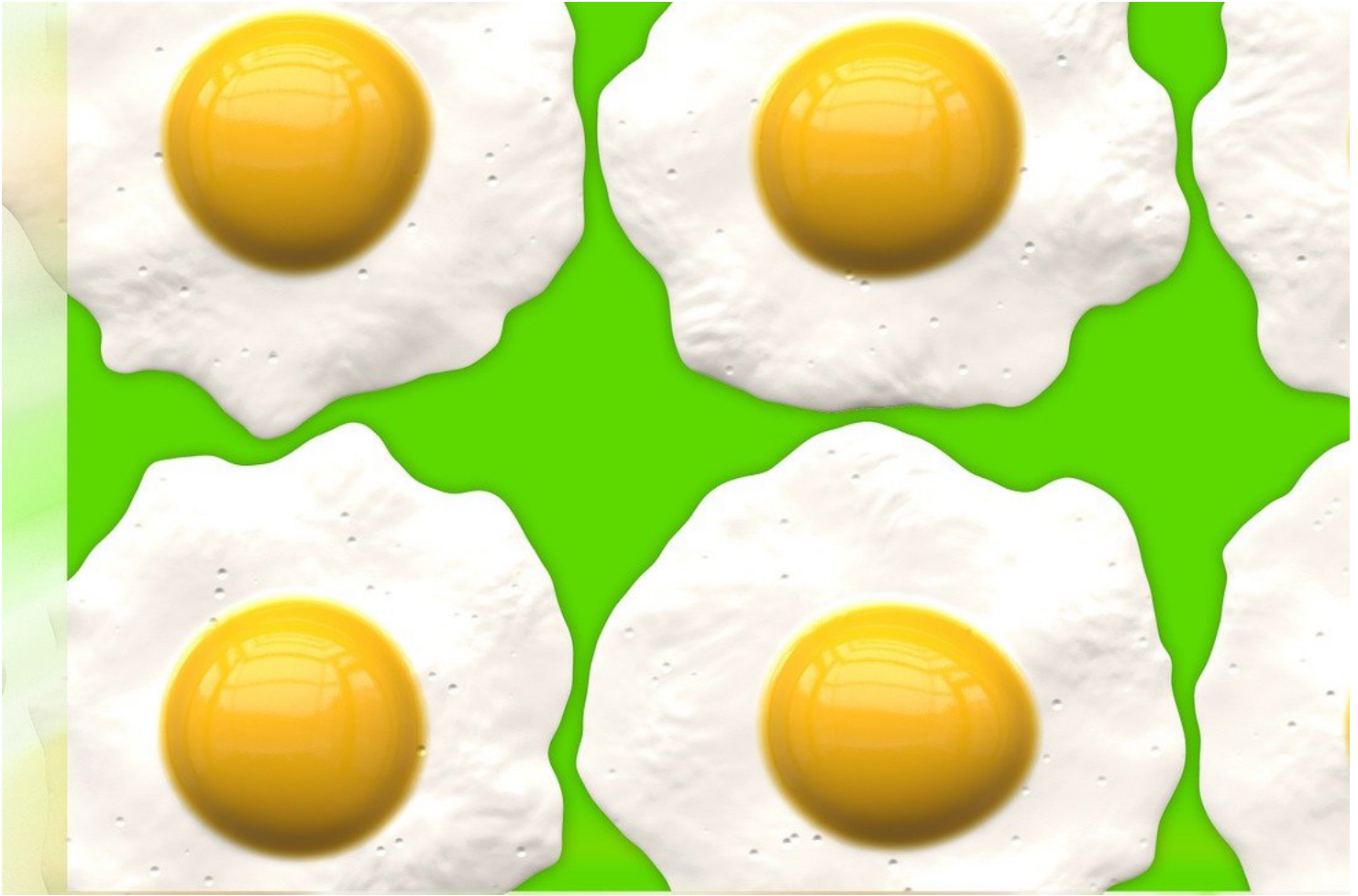
- Arrays
  - [Binary Search](#)  
Preview
  - [Find Maximum in Sliding Window](#)
  - [Search rotated array](#)
  - [Find smallest common number](#)
  - [Rotate Array](#)
  - [Find low/high index](#)
  - [Move zeros to left](#)
  - [Find maximum single sell profit](#)
  - [Implement Quicksort](#)
  - [Merge Overlapping Intervals](#)
  - [Sum of Two Values](#)  
Preview
- Linked List
  - [Reverse a singly linked list](#)  
Preview
  - [Remove Duplicates from a Linked List](#)
  - [Delete node with a given key](#)
  - [Insertion Sort of a Linked List](#)
  - [Intersection Point of Two Lists](#)  
Preview
  - [Nth from last node](#)
  - [Swap Nth Node with Head](#)
  - [Merge Two Sorted Linked Lists](#)
  - [Merge Sort](#)
  - [Reverse even nodes](#)
  - [Rotate a Linked List](#)
  - [Reverse k Elements](#)
  - [Add Two Integers](#)
  - [Copy linked list with arbitrary pointer](#)
- Math & Stats
  - [Find kth permutation](#)  
Preview
  - [Integer Division](#)
  - [Pythagorean Triplets](#)
  - [All Sum Combinations](#)
  - [Find Missing Number](#)
  - [Permute String](#)
  - [All Subsets](#)
  - [Is Number Valid?](#)
  - [Power of a Number](#)
  - [Calculate Square Root](#)
- String
  - [Reverse words in a sentence](#)  
Preview
  - [Remove Duplicates](#)
  - [Remove white spaces](#)
  - [String Segmentation](#)
  - [XML to Tree](#)
  - [Find all palindrome substrings](#)
  - [Regular Expression](#)
- Trees
  - [Check if two binary trees are identical](#)  
Preview
  - [Write an Inorder Iterator for a Binary Tree](#)
  - [Iterative Inorder Traversal](#)
  - [Inorder Successor BST](#)
  - [Level Order Traversal of Binary Tree](#)
  - [Is Binary Search Tree?](#)
  - [Convert Binary Tree To Doubly Linked List](#)
  - [Print Tree Perimeter](#)
  - [Connect Same Level Siblings](#)

- [Serialize/Deserialize Binary Tree](#)
- [Connect All Siblings](#)
- [Inorder Successor BST with parent pointers](#)
- [Nth Highest in BST](#)
- [Mirror binary tree nodes](#)
- [Delete zero sum sub-trees](#)
- [N-ary Tree to Binary Tree](#)
- Stacks and Queues
  - [Stack using Queues](#)  
Preview
  - [Queue using Stacks](#)
  - [Expression Evaluation](#)
- Graphs
  - [Clone a Directed Graph](#)  
Preview
  - [Minimum Spanning Tree](#)
  - [Word Chaining](#)
- Back Tracking
  - [Boggle](#)  
Preview
  - [All Possible Parentheses](#)
  - [Solve N-Queens problem](#)
  - [Find K-sum subsets](#)
- Dynamic Programming
  - [Fibonacci numbers](#)  
Preview
  - [Largest Sum Subarray](#)
  - [MaxSum Subsequence - Nonadjacent Elements](#)
  - [Game Scoring: Find number of ways a player can score 'n' runs](#)
  - [Coin Changing Problem](#)
  - [Levenshtein Distance](#)
- Miscellaneous
  - [Sum of Three Values](#)  
Preview
  - [Make Columns and Rows Zeros](#)
  - [Search in a Matrix](#)
  - [Implement LRU Cache](#)
  - [Host Endianness](#)
  - [Closest Meeting Point](#)
- Appendix
  - [Testimonials](#)
  - [Frequently Asked Questions](#)
  - [Other Resources](#)

[LearnTeach](#)

- [My Profile](#)

- [View](#)
- [Edit](#)
- Logout



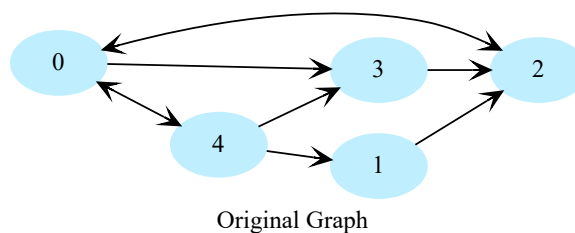
## Clone a Directed Graph

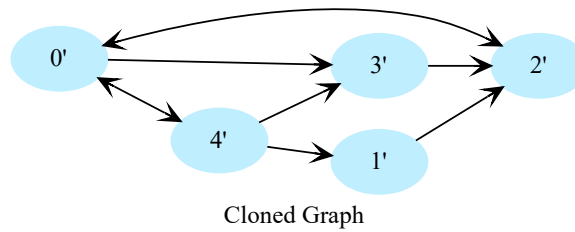
Given root node of a directed graph, clone this graph by creating its deep copy such that cloned graph has same vertices and edges as original graph.

### Description

Given root node of a directed graph, clone this graph by creating its deep copy such that cloned graph has same vertices and edges as the original graph.

Let's look at below graphs as an example. If input graph is  $G = (V, E)$  where  $V$  is set of vertices and  $E$  is set of edges, then output graph (cloned graph)  $G' = (V', E')$  such that  $V = V'$  and  $E = E'$ .





## Hints

- Use hash table
- Depth first traversal

## Solution

### Runtime Complexity

Linear,  $O(n)$ .

### Memory Complexity

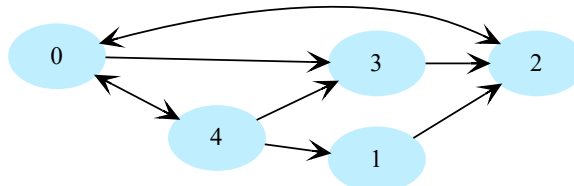
Logarithmic,  $O(n)$ . 'n' is number of vertices in graph.

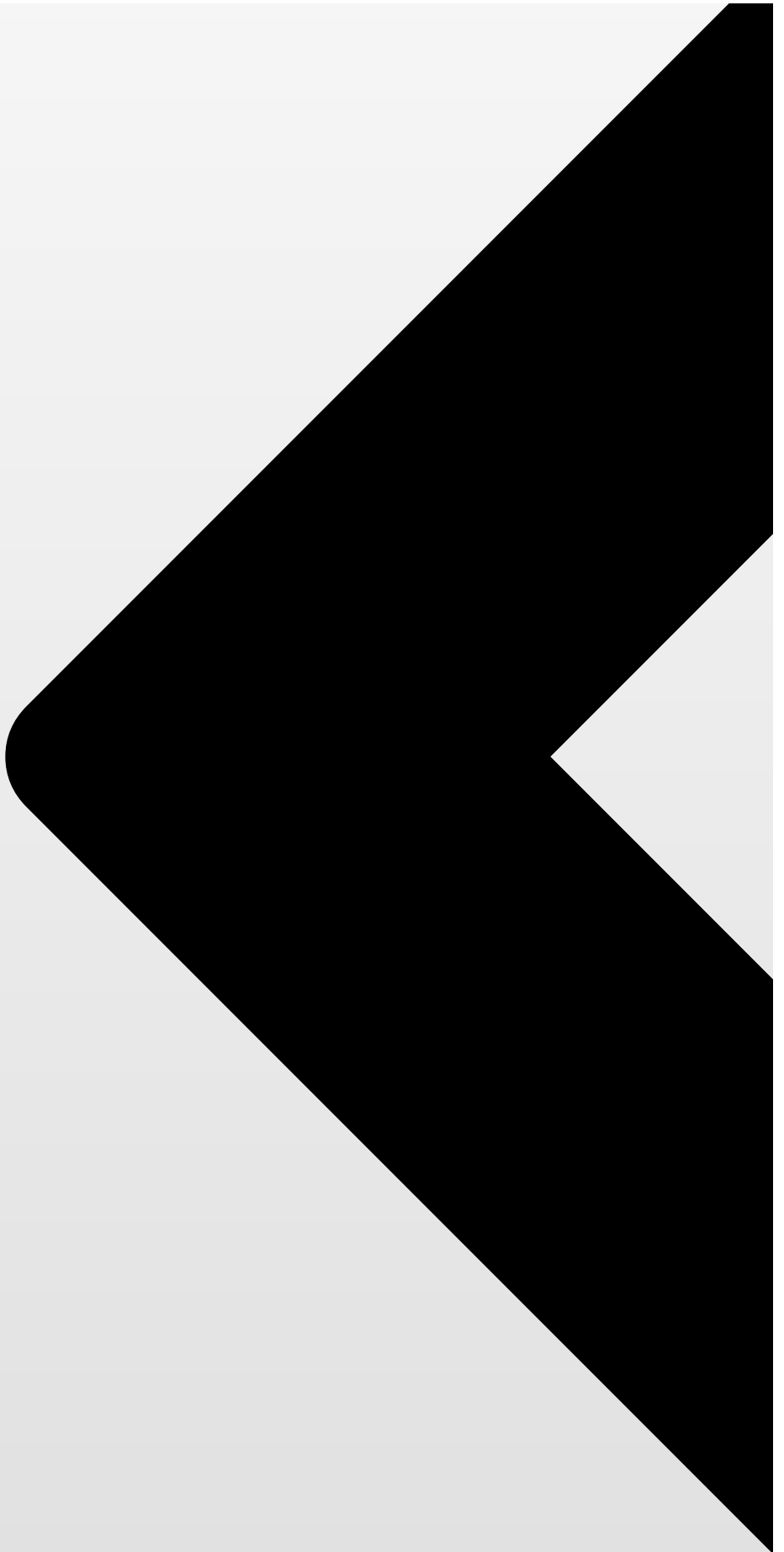
*We can have most  $n$  entries in hash table, so worst case space complexity is  $O(n)$ .*

We use depth first traversal and create a copy of each node while traversing the graph. To avoid getting stuck in cycles, we'll use a hashtable to store each completed node and will not revisit nodes that exist in the hashtable. Hashtable key will be a node in the original graph, and its value will be the corresponding node in cloned graph.

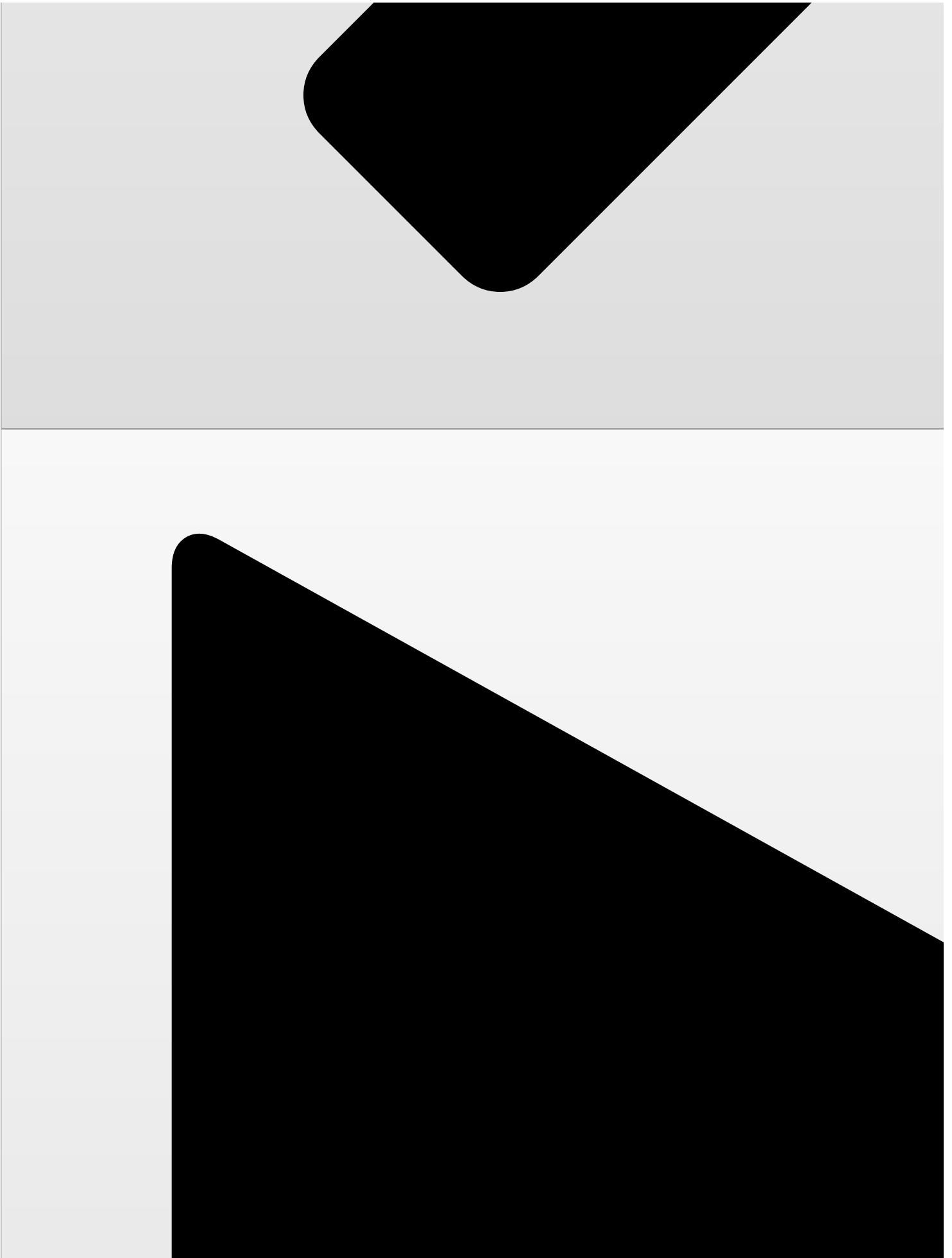
For above graph lets assume root is node '0'. We'll start with the root node i.e. '0'.

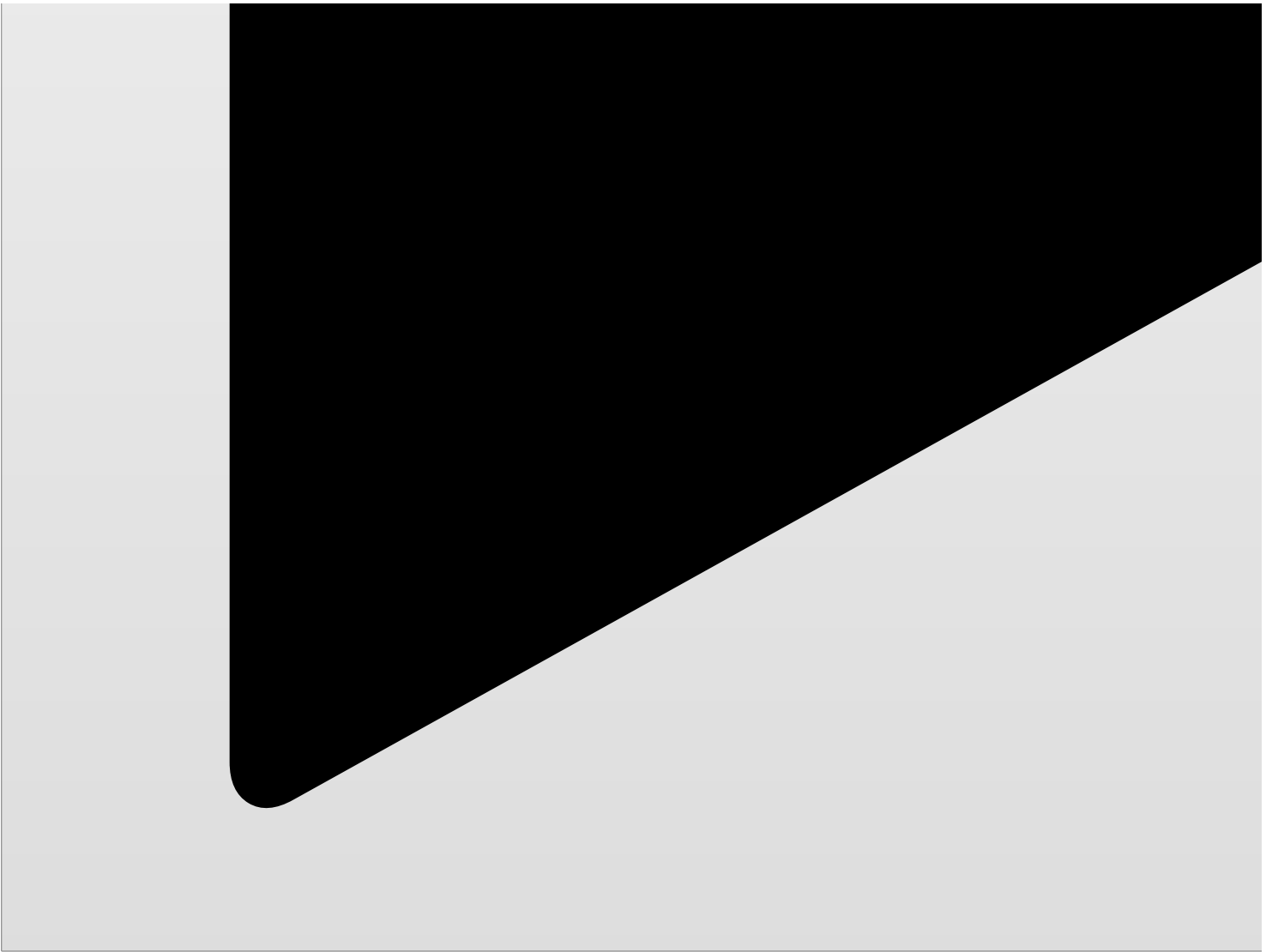
Suppose the original graph looks like this. We will now clone the exact same graph by performing a depth first traversal.



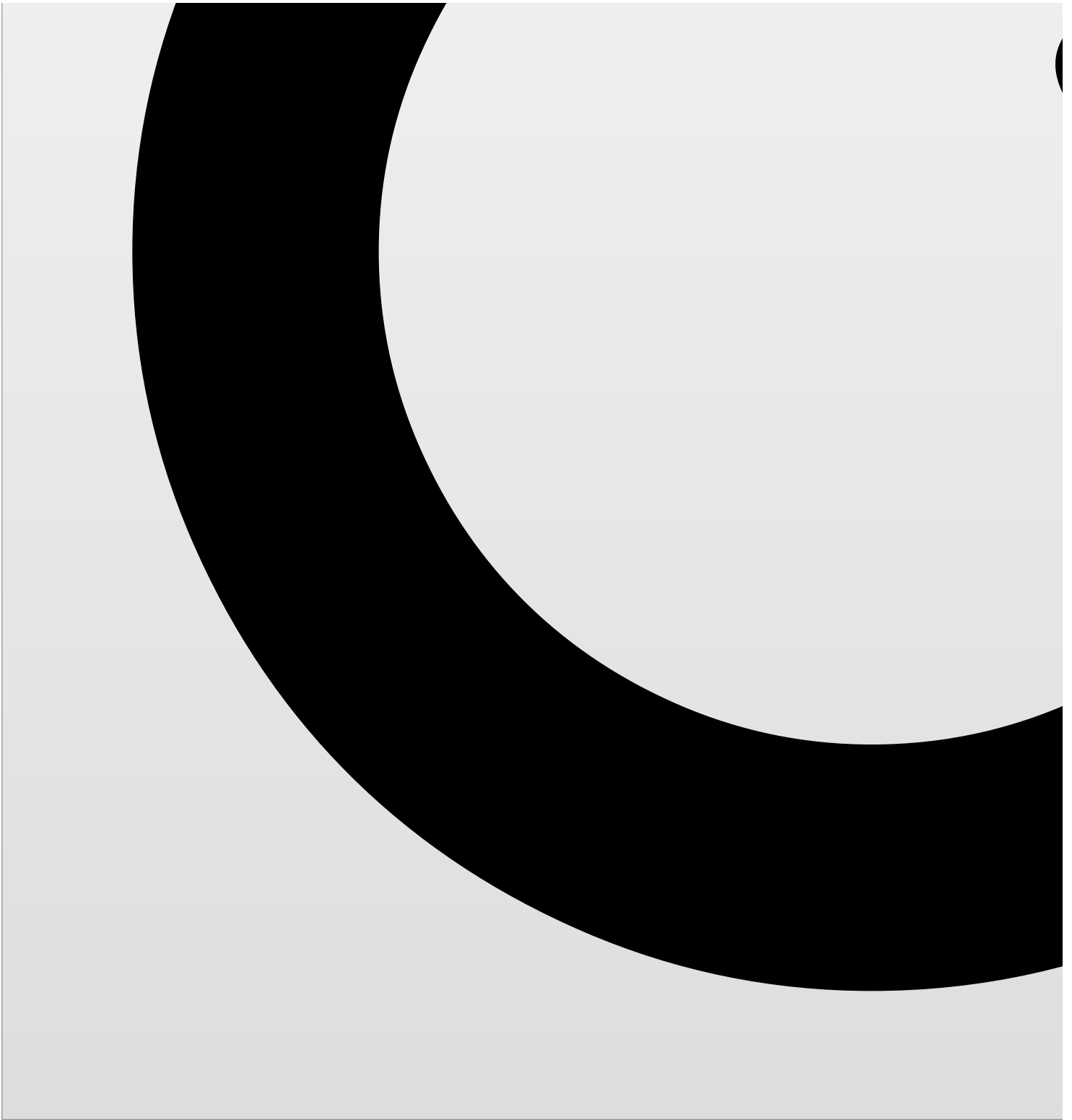


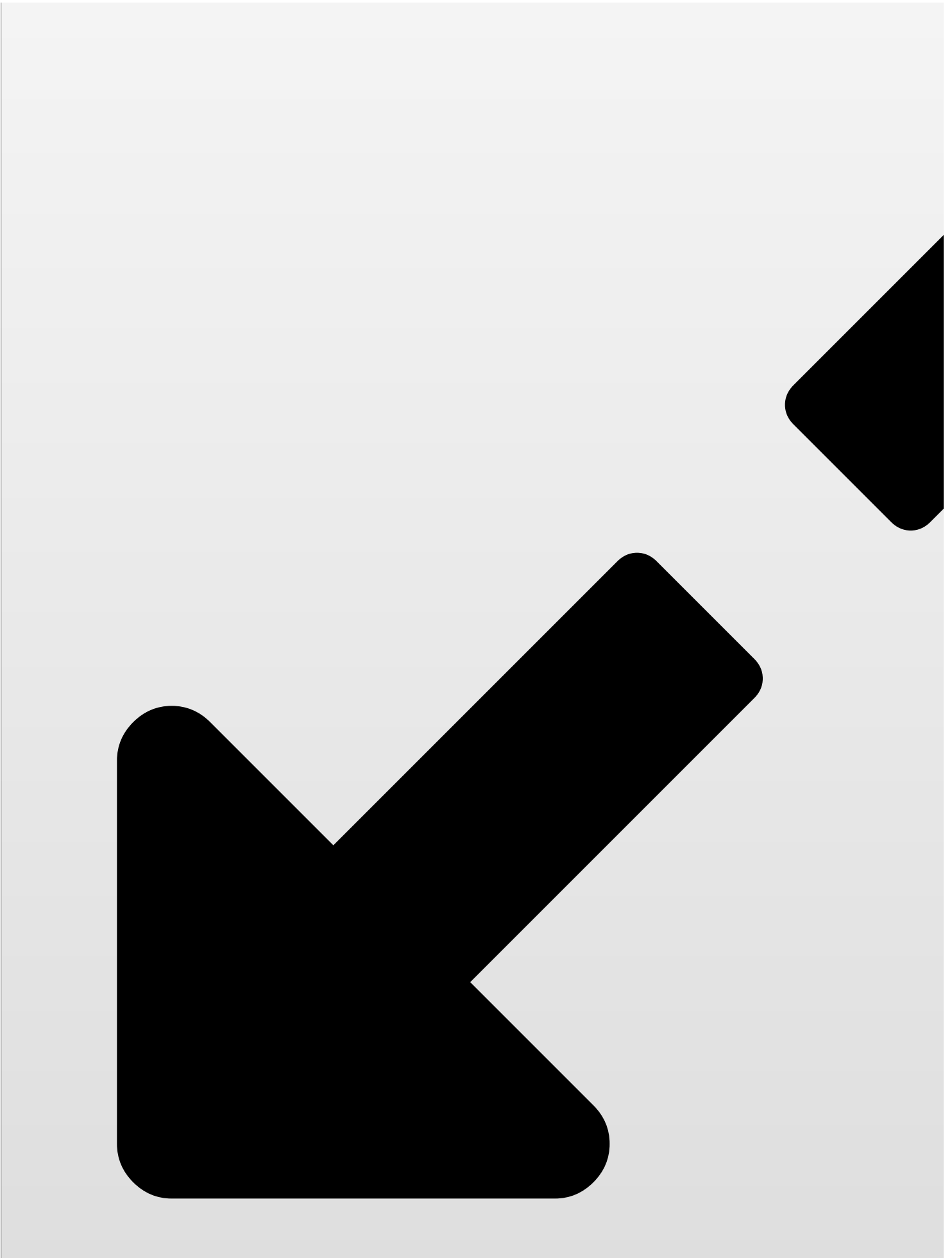












- [C++](#)
- [Java](#)
- [Python](#)
- [JS](#)
- [Ruby](#)

35

1

```
struct Node {
```

2

```
    int data;
```

3

```
    list<Node*> neighbors;
```

4

```
    Node(int d) : data(d) {}
```

5

```
};
```

6

7

```
Node* clone_rec(Node* root,
```

8

```
    unordered_map<Node*,
```

9

```
    Node*>& nodes_completed) {
```

10

11

```
    if (root == nullptr) {
```

12

```
        return nullptr;
```

13

```
    }
```

14

15

```
    Node* pNew = new Node(root->data);
```

16

```
    nodes_completed[root] = pNew;
```

17

```
18
    for (Node* p : root->neighbors) {
19
20
        auto x = nodes_completed.find(p);
21
22
        if (x == nodes_completed.end()){
23
            pNew->neighbors.push_back(clone_rec(p, nodes_completed));
24
        } else {
25
            pNew->neighbors.push_back(x->second /*value*/);
26
        }
27
    }
28
29
    return pNew;
30
}
31
32
Node* clone(Node* root) {
33
    unordered_map<Node*, Node*> nodes_completed;
34
    return clone_rec(root, nodes_completed);
35
}
```

Cover Image courtesy of [Queensland Museum](#)

[Mark as completed](#)

[← Previous](#) [Expression Evaluation](#) [Next →](#) [Minimum Spanning Tree](#)

Send feedback or ask a question





7 recommendations

- [Home](#)
- [Featured](#)
- [Team](#)
- [Blog](#)
- [FAQ](#)
- [Terms of Service](#)
- [Contact Us](#)