Clone an Undirected Graph

The idea is to do a BFS traversal of the graph and while visiting a node make a clone node of it (a copy of original node). If a node is encountered which is already visited then it already has a clone node.

How to keep track of the visited/cloned nodes? A HashMap/Map is required in order to maintain all the nodes which have already been created. *Key stores*: Reference/Address of original Node *Value stores*: Reference/Address of cloned Node A copy of all the graph nodes has been made,

how to connect clone nodes? While visiting the neighboring vertices of a node *u* get the corresponding cloned node for u , let's call that *cloneNodeU* , now visit all the neighboring nodes for *u* and for each neighbor find the corresponding clone node(if not found create one) and then push into the neighboring vector of *cloneNodeU* node.

ow to verify if the cloned graph is a correct? Do a BFS traversal before and after the cloning of graph. In BFS traversal display the value of a node along with its address/reference. Compare the order in which nodes are displayed, if the values are same but the address/reference is different for both the traversals then the cloned graph is correct.

Implementation:

C++

```
// A C++ program to Clone an Undirected Graph

#include<bits/stdc++.h>

using namespace std;


struct GraphNode

{

    int val;


    //A neighbour vector which contains addresses to

    //all the neighbours of a GraphNode

    vector<GraphNode*> neighbours;
```

```
};

// A function which clones a Graph and
// returns the address to the cloned
// src node
GraphNode *cloneGraph(GraphNode *src)
{
    //A Map to keep track of all the
    //nodes which have already been created
    map<GraphNode*, GraphNode*> m;
    queue<GraphNode*> q;

    // Enqueue src node
    q.push(src);
    GraphNode *node;

    // Make a clone Node
    node = new GraphNode();
    node->val = src->val;

    // Put the clone node into the Map
    m[src] = node;
    while (!q.empty())
    {
        //Get the front node from the queue
        //and then visit all its neighbours
        GraphNode *u = q.front();
```

```
        q.pop();
        vector<GraphNode *> v = u->neighbours;
        int n = v.size();
        for (int i = 0; i < n; i++)
        {
            // Check if this node has already been created
            if (m[v[i]] == NULL)
            {
                // If not then create a new Node and
                // put into the HashMap
                node = new GraphNode();
                node->val = v[i]->val;
                m[v[i]] = node;
                q.push(v[i]);
            }

            // add these neighbours to the cloned graph node
            m[u]->neighbours.push_back(m[v[i]]);
        }
    }

    // Return the address of cloned src Node
    return m[src];
}

// Build the desired graph
GraphNode *buildGraph()
```

```
{
    /*
        Note : All the edges are Undirected
        Given Graph:
        1--2
        | |
        4--3
    */
    GraphNode *node1 = new GraphNode();
    node1->val = 1;
    GraphNode *node2 = new GraphNode();
    node2->val = 2;
    GraphNode *node3 = new GraphNode();
    node3->val = 3;
    GraphNode *node4 = new GraphNode();
    node4->val = 4;
    vector<GraphNode *> v;
    v.push_back(node2);
    v.push_back(node4);
    node1->neighbours = v;
    v.clear();
    v.push_back(node1);
    v.push_back(node3);
    node2->neighbours = v;
    v.clear();
    v.push_back(node2);
    v.push_back(node4);
```

```
    node3->neighbours = v;
    v.clear();
    v.push_back(node3);
    v.push_back(node1);
    node4->neighbours = v;
    return node1;
}

// A simple bfs traversal of a graph to
// check for proper cloning of the graph
void bfs(GraphNode *src)
{
    map<GraphNode*, bool> visit;
    queue<GraphNode*> q;
    q.push(src);
    visit[src] = true;
    while (!q.empty())
    {
        GraphNode *u = q.front();
        cout << "Value of Node " << u->val << "\n";
        cout << "Address of Node " <<u << "\n";
        q.pop();
        vector<GraphNode *> v = u->neighbours;
        int n = v.size();
        for (int i = 0; i < n; i++)
        {
            if (!visit[v[i]])
```

```
            {
                visit[v[i]] = true;
                q.push(v[i]);
            }
        }
    }
    cout << endl;
}

// Driver program to test above function
int main()
{
    GraphNode *src = buildGraph();
    cout << "BFS Traversal before cloning\n";
    bfs(src);
    GraphNode *newsrc = cloneGraph(src);
    cout << "BFS Traversal after cloning\n";
    bfs(newsrc);
    return 0;
}
```

Output

```
BFS Traversal before cloning
Value of Node 1
Address of Node 0x1b6ce70
Value of Node 2
Address of Node 0x1b6cea0
Value of Node 4
Address of Node 0x1b6cf00
```

```
Value of Node 3
Address of Node 0x1b6ced0

BFS Traversal after cloning
Value of Node 1
Address of Node 0x1b6e5a0
Value of Node 2
Address of Node 0x1b6e5d0
Value of Node 4
Address of Node 0x1b6e620
Value of Node 3
Address of Node 0x1b6e670
```

Time Complexity: O(V+E) where V is the number of vertices and E is the number of edges in the graph.
Auxiliary Space: O(V), since a map is used to store the graph nodes which can grow upto V.