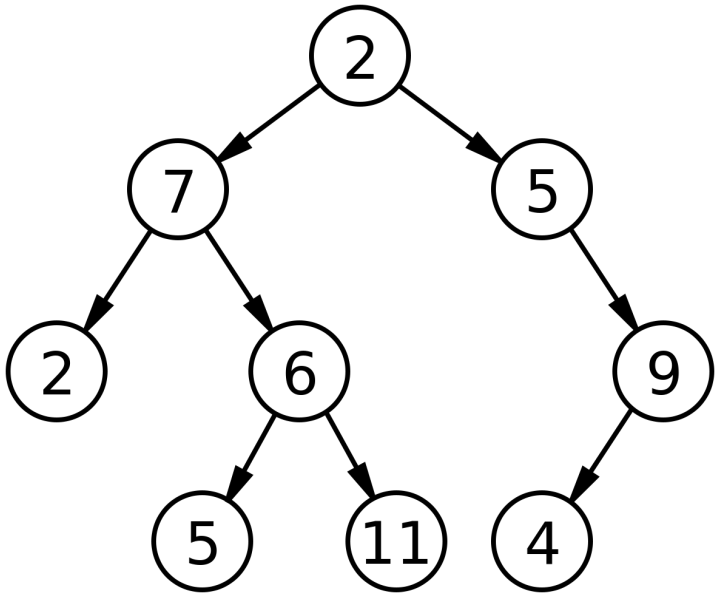
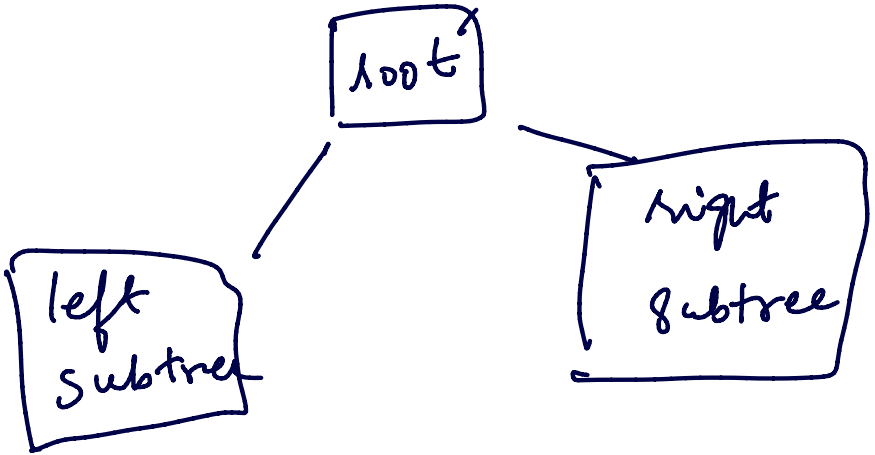
Midterm Revision

what is the sum of elements

1- Is 11 present in the tree?





```c
int addBT(Node* root)
{
    if (root == NULL)
        return 0;
    return (root->data + addBT(root->left) + addBT(root->right));
}
```

One Approach

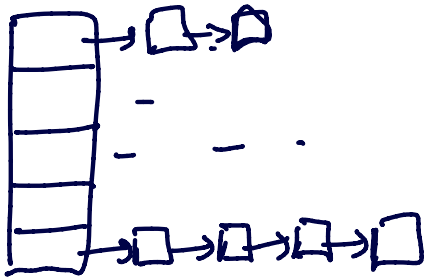
```c
void addBT(Node* root,int *sum){
  if (root!=NULL){

  *sum+=root->data;

  addBT(root->left,sum);
  addBT(root->right,sum);
}
}
```

Another approach
(not recommended)

5

Consider a Hash Table of size 10. Collisions in this Hash Table are resolved using chaining. After insertion of a certain number of elements, the length of the chain in each of the Hash Table slots are as follows:

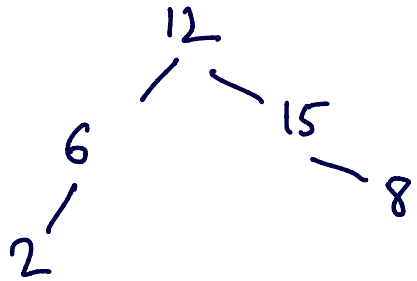
Slot-1: 3
Slot-2: 5
Slot-3: 6
Slot-4: 0
Slot-5: 5



What is the number of collisions caused due to the insertion of the elements?

2 + 4 + 5 + 4

```
#include <iostream>
struct Node {
        int key;
        Node* left;
        Node* right;
        Node(int k) {
                this->key = k;
                this->right = this->left = NULL;
        }

};

void print(Node* n, int k) {

        if (n == NULL) return;
        if ( k%2 == 0) {
                std::cout << n->key << " ";
                k++;
        }

        print(n->left, k);
        print(n->right, k);
}
```

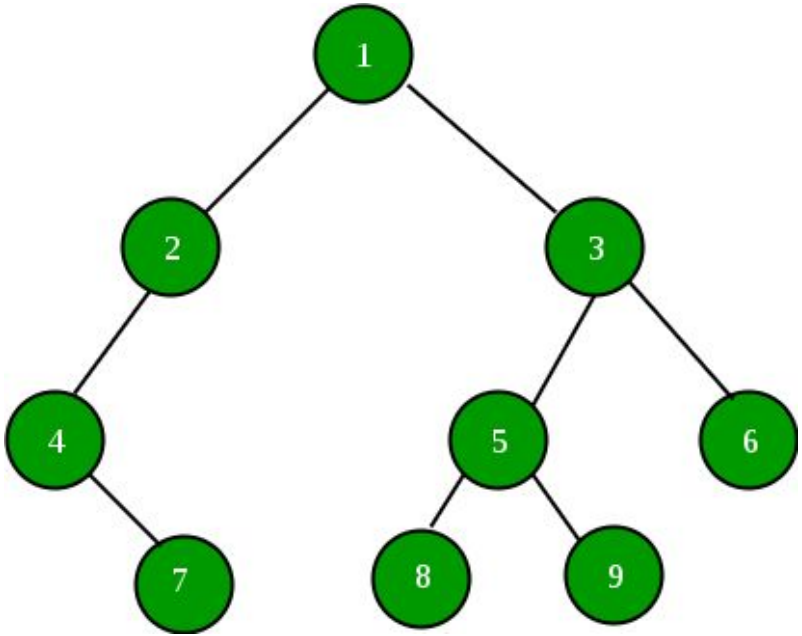



```
int main() {

        Node* root = new Node(12);

        root->left = new Node(6);

        root->right = new Node(15);

        root->left->left = new Node(2);

        root->right->right = new Node(8);

        print(root, 0);

}
```

What will be the output?

12

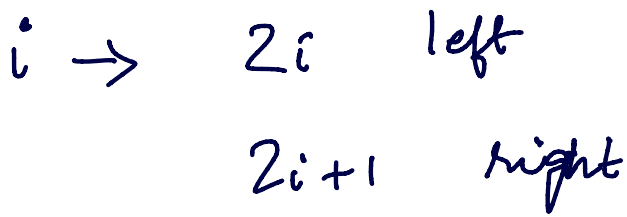
2- Is this a min heap?

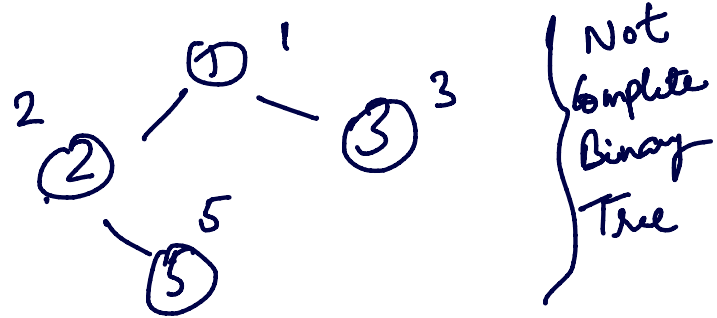


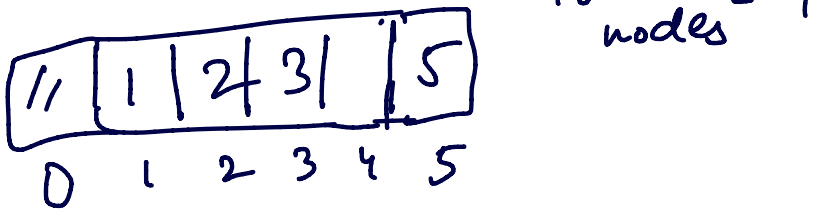
No

A Min-Heap is a complete binary tree!

Since it's a complete binary tree, let's try making an array representation of this tree (we don't do this will normally other binary trees because of huge empty spaces)





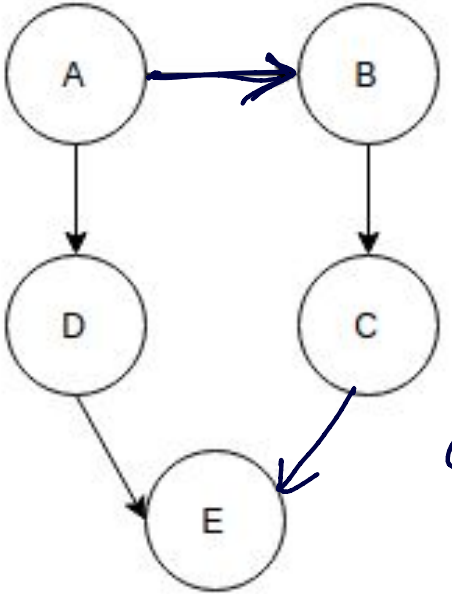


3: Check if the tree is complete?

```
bool isComplete (struct Node* root, unsigned int index,
                 unsigned int number_nodes)
{
    // An empty tree is complete
    if (root == NULL)
        return (true);

    // If index assigned to current node is more than
    // number of nodes in tree, then tree is not complete
    if (index > number_nodes)
        return (false);

    // Recur for left and right subtrees
    return (isComplete(root->left, 2*index, number_nodes) &&
            isComplete(root->right, 2*index + 1, number_nodes));
}
```

If you start with index 1

4. Given a graph, find the vertices with maximum incoming edges (indegree)



Source

How would you code it up?
(easy way with adjacency matrix)

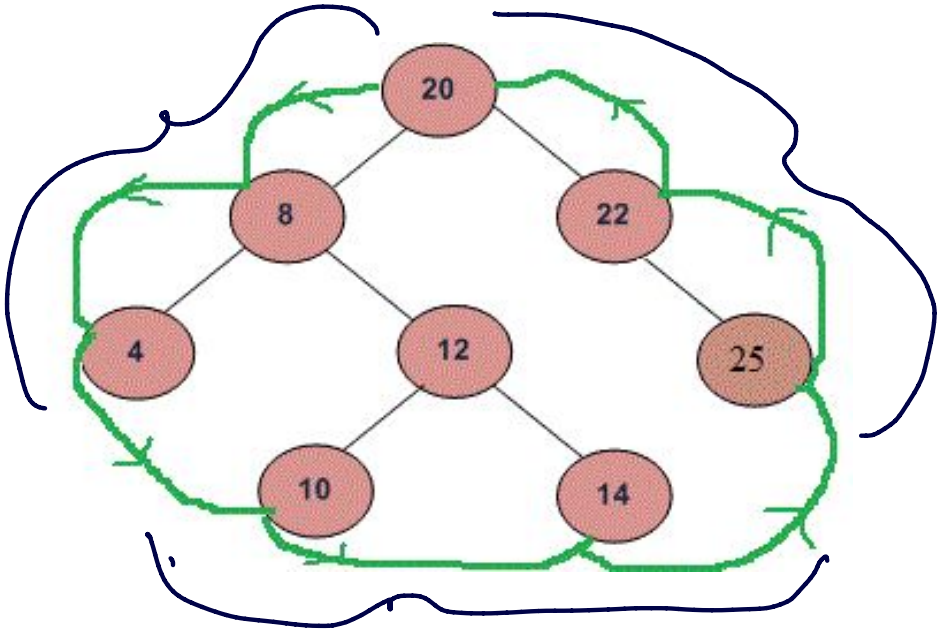
	A	B	C	D	E
A	0	1	0	1	0
B			1		
C					1
D					1
E	0	0	0	0	0

← destination.

Choose vertex whose column has max 1's

You can do this by adjacency list too since you have used it more

5: Print boundary of a tree



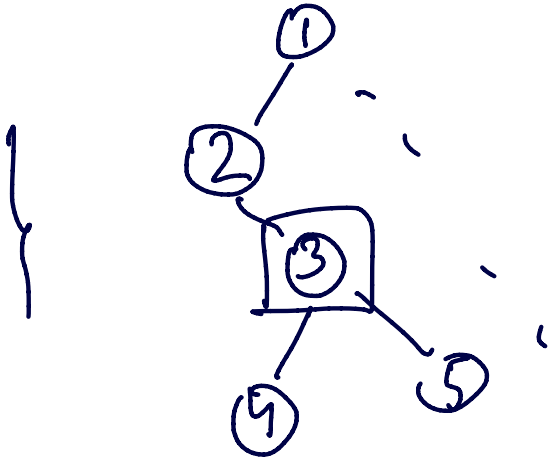
20, 8, 4, 10, 14, 25, 22

1) Print left boundary top-down
2) Print all leaves from left to right
3) Print right boundary bottom up

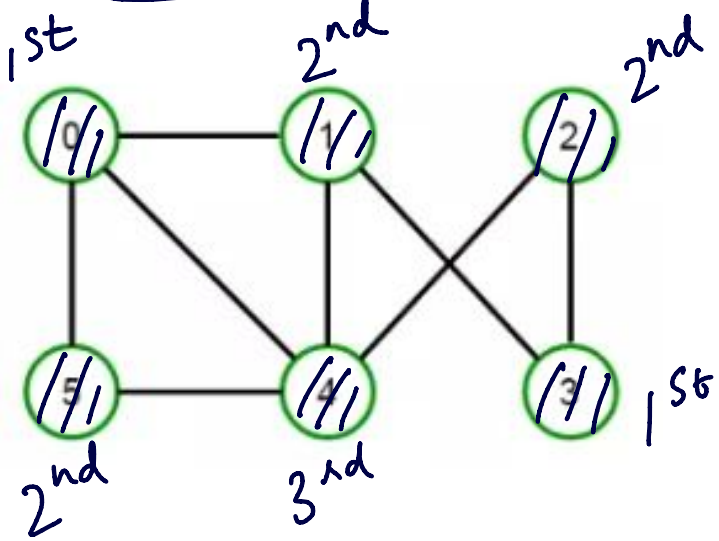
```
void printBoundary(struct node* root)
{
    if (root) {
        printf("%d ", root->data);

        // Print the left boundary in top-down manner.
        printBoundaryLeft(root->left);

        // Print all leaf nodes
        printLeaves(root->left);
        printLeaves(root->right);

        // Print the right boundary in bottom-up manner
        printBoundaryRight(root->right);
    }
}
```

```c
void printBoundaryLeft(struct node* root)
{
    if (root) {
        if (root->left) {
            // to ensure top down order, print the node
            // before calling itself for left subtree
            printf("%d ", root->data);
            printBoundaryLeft(root->left);
        }
        else if (root->right) {
            printf("%d ", root->data);
            printBoundaryLeft(root->right);
        }
        // do nothing if it is a leaf node, this way we avoid
        // duplicates in output
    }
}
```

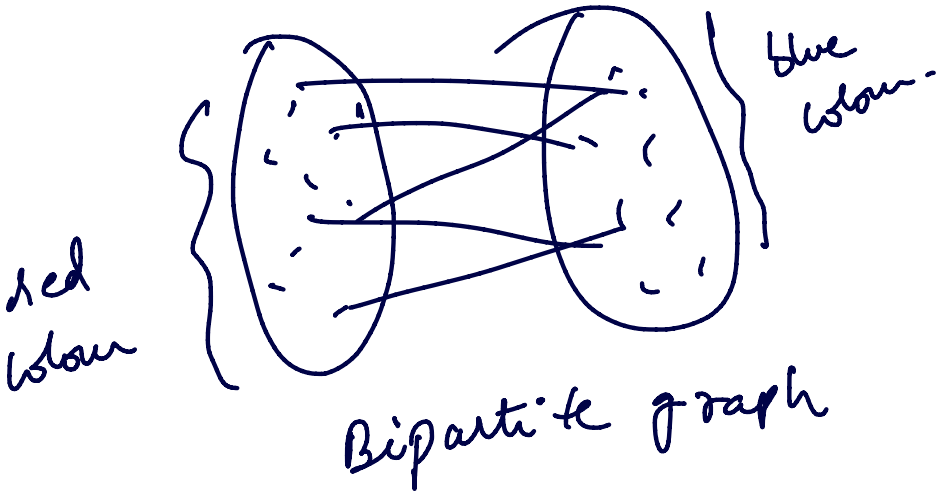
How will printBoundaryRight() look like?



6. K-colorable graph, what is k?



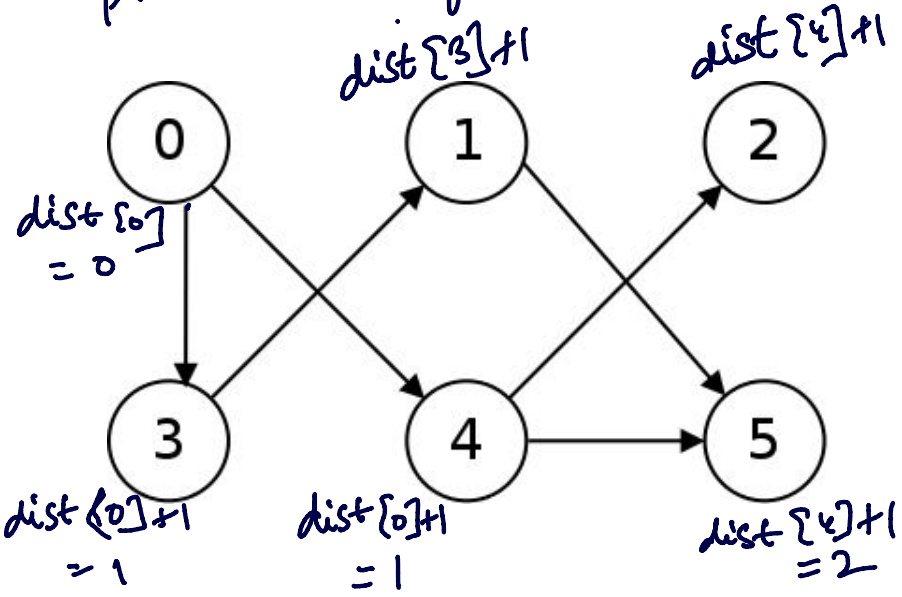
Color the vertices of a graph using k colors in such a way that vertices of the same color are never adjacent



7. Print the shortest path

```
for (int i = 0; i < n; i++) {
    visited[i] = false;
    dist[i] = INT_MAX;
    pred[i] = -1;
}


visited[src] = true;
dist[src] = 0;
queue.push_back(src);

// standard BFS algorithm
while (!queue.empty()) {
    int u = queue.front();
    queue.pop_front();
    for (int i = 0; i < adj[u].size(); i++) {
        if (visited[adj[u][i]] == false) {
            visited[adj[u][i]] = true;
            dist[adj[u][i]] = dist[u] + 1;
            pred[adj[u][i]] = u;
            queue.push_back(adj[u][i]);

            // We stop BFS when we find
            // destination.
            if (adj[u][i] == dest)
               return true;
        }
    }
}

return false;
```

distance array
pred array
src = 0
dest = 5



Draw the queue to understand better!

```
// vector path stores the shortest path
vector<int> path;
int crawl = dest;
path.push_back(crawl);
while (pred[crawl] != -1) {
    path.push_back(pred[crawl]);
    crawl = pred[crawl];
}

// distance from source is in distance array
cout << "Shortest path length is : "
    << dist[dest];

// printing path from source to destination
cout << "\nPath is::\n";
for (int i = path.size() - 1; i >= 0; i--)
    cout << path[i] << " ";
```

loop from 5 to pred[5] to pred[pred[5]] and so on until you reach the Src

path

5, 4, 0

reverse print it.