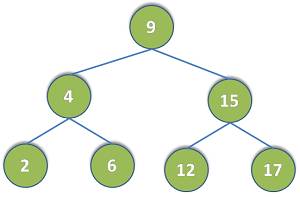
0Binary tree is basically tree in which each node can have two child nodes and each child node can itself be a small binary tree. To understand it, below is the example figure of binary tree.



Binary tree works on the rule that child nodes which are lesser than root node keep on the left side and child nodes which are greater than root node keep on the right side. Same rule is followed in child nodes as well that are itself sub-trees. Like in above figure, nodes (2, 4, 6) are on left side of root node (9) and nodes (12, 15, 17) are on right side of root node (9).

We will understand binary tree through its operations. We will cover following operations.

* Create binary tree
* Search into binary tree
* Delete binary tree
* Displaying binary tree

Creation of binary tree

void insert(node \*\* tree, int val) {

node \*temp = NULL;

if(!(\*tree)) {

temp = (node \*)malloc(sizeof(node));

temp->left = temp->right = NULL;

temp->data = val;

\*tree = temp;

return;

}

if(val < (\*tree)->data) {

insert(&(\*tree)->left, val);

} else if(val > (\*tree)->data) {

insert(&(\*tree)->right, val);

}

}

### Searching into binary tree

node\* search(node \*\* tree, int val) {

if(!(\*tree)) {

return NULL;

}

if(val == (\*tree)->data) {

return \*tree;

} else if(val < (\*tree)->data) {

search(&((\*tree)->left), val);

} else if(val > (\*tree)->data){

search(&((\*tree)->right), val);

}

}

### Deletion of binary tree

void deltree(node \* tree) {

if (tree) {

deltree(tree->left);

deltree(tree->right);

free(tree);

}

}

### Displaying binary tree

void print\_preorder(node \* tree) {

if (tree) {

printf("%d\n",tree->data);

print\_preorder(tree->left);

print\_preorder(tree->right);

}

}

void print\_inorder(node \* tree) {

if (tree) {

print\_inorder(tree->left);

printf("%d\n",tree->data);

print\_inorder(tree->right);

}

}

void print\_postorder(node \* tree) {

if (tree) {

print\_postorder(tree->left);

print\_postorder(tree->right);

printf("%d\n",tree->data);

}

}

**Graph**

https://cdncontribute.geeksforgeeks.org/wp-content/uploads/undirectedgraph.png

Adjacency Matrix Representation

Adjacency List Representation of Graph

**// A C Program to demonstrate adjacency list**

**// representation of graphs**

**#include <stdio.h>**

**#include <stdlib.h>**

**// A structure to represent an adjacency list node**

**struct AdjListNode**

**{**

**int dest;**

**struct AdjListNode\* next;**

**};**

**// A structure to represent an adjacency list**

**struct AdjList**

**{**

**struct AdjListNode \*head;**

**};**

**// A structure to represent a graph. A graph**

**// is an array of adjacency lists.**

**// Size of array will be V (number of vertices**

**// in graph)**

**struct Graph**

**{**

**int V;**

**struct AdjList\* array;**

**};**

**// A utility function to create a new adjacency list node**

**struct AdjListNode\* newAdjListNode(int dest)**

**{**

**struct AdjListNode\* newNode =**

**(struct AdjListNode\*) malloc(sizeof(struct AdjListNode));**

**newNode->dest = dest;**

**newNode->next = NULL;**

**return newNode;**

**}**

**// A utility function that creates a graph of V vertices**

**struct Graph\* createGraph(int V)**

**{**

**struct Graph\* graph =**

**(struct Graph\*) malloc(sizeof(struct Graph));**

**graph->V = V;**

**// Create an array of adjacency lists. Size of**

**// array will be V**

**graph->array =**

**(struct AdjList\*) malloc(V \* sizeof(struct AdjList));**

**// Initialize each adjacency list as empty by**

**// making head as NULL**

**int i;**

**for (i = 0; i < V; ++i)**

**graph->array[i].head = NULL;**

**return graph;**

**}**

**// Adds an edge to an undirected graph**

**void addEdge(struct Graph\* graph, int src, int dest)**

**{**

**// Add an edge from src to dest. A new node is**

**// added to the adjacency list of src. The node**

**// is added at the begining**

**struct AdjListNode\* newNode = newAdjListNode(dest);**

**newNode->next = graph->array[src].head;**

**graph->array[src].head = newNode;**

**// Since graph is undirected, add an edge from**

**// dest to src also**

**newNode = newAdjListNode(src);**

**newNode->next = graph->array[dest].head;**

**graph->array[dest].head = newNode;**

**}**

**// A utility function to print the adjacency list**

**// representation of graph**

**void printGraph(struct Graph\* graph)**

**{**

**int v;**

**for (v = 0; v < graph->V; ++v)**

**{**

**struct AdjListNode\* pCrawl = graph->array[v].head;**

**printf("\n Adjacency list of vertex %d\n head ", v);**

**while (pCrawl)**

**{**

**printf("-> %d", pCrawl->dest);**

**pCrawl = pCrawl->next;**

**}**

**printf("\n");**

**}**

**}**

**// Driver program to test above functions**

**int main()**

**{**

**// create the graph given in above fugure**

**int V = 5;**

**struct Graph\* graph = createGraph(V);**

**addEdge(graph, 0, 1);**

**addEdge(graph, 0, 4);**

**addEdge(graph, 1, 2);**

**addEdge(graph, 1, 3);**

**addEdge(graph, 1, 4);**

**addEdge(graph, 2, 3);**

**addEdge(graph, 3, 4);**

**// print the adjacency list representation of the above graph**

**printGraph(graph);**

**return 0;**

**}**