Graph : BFS

```
int fn( vector< vector<int>> &graph){
    queue<int> q;
    unordered_set<int> visited;
     // for stating from
     q.push(START_NODE); // please not you can use
                         // deque also here

      visited.insert(START_NODE);
      int ans =0; //your task here to just Collect all the
                  //node value

      while(q.empty()==false){
          int node = q.front(); q.pop();
           ans += node;
          for(int neighbor: graph[node]){
            if(visited.find(neighbour)== visited.end()){
               visited.Insert(neighbor);
               q.push(neighbor);
            }
          }
      }
       return ans;
   }
```

Starting node push

iterate to all child using adjacency List

18/03/2023

Binary Tree : DFS(recursive)

Code Template (Recursive Code)

```
// code logic to sum up all node value.
int dfs(TreeNode *root) {
    if (root == nullptr) return 0;
    return (root->value + dfs(root->left) + dfs(root->right));
}
```

Binary Tree : DFS (iterative)

```
int dfs(TreeNode *root) {
  Stack<TreeNode *> s;
   s.push(root);  ] start from
   Int ans=0;
   while (!s.empty()) {
     TreeNode* node = s.top(); s.pop();
      ans += node->value;   <- sum up of all node values.
      If(node->left) s.push(node->left);     ]  call
      if(node->right) s.push(node->right);   ]  for child operation.
   }
   return ans;  <- return
}
```

Binary Tree - BFS (Iterative)

Problem statement : Traverse the Binary Tree with BFS and Collect the sum of node values and return it.

```
int bfs(TreeNode* root) {
  deque<TreeNode*     > q;
   q.push_back(root);  } start node
   int ans=0;
   while(!q.empty()) {   <- iterate until
                            q is not empty
     int qsize = q.size();
     // do logic for current level
     // if you need something to process level by level
     for(int i=0; i<qsize; i++) {   <- level by level
                                        traversal
        TreeNode* node = q.front();
          q.pop_front();
          ans += node->value;
Child
data processing ->  if(node -> left) q.push_back(node->left)
                ->  if(node -> right) q.push_back(node->right)
       }
   }
   return ans;   <- Sum of all node
                    Value accumulated
                    in ans variable.
}
```