NOTES:

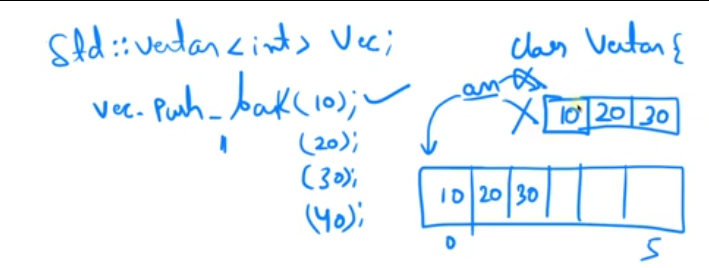
STL – Standard Template Library

1. Containers: Data Types- store the data
2. Iterators: move back and forth in the containers
3. Algorithms:

Array: Random Access Iterators

* STD:: Array – size is needed at the compile time, pass by value – Size known
* Syntax: array <type, Size> name;

Vector: Dynamic Array – Sequence Container

1. At(), front(), back(), data() --- Elem Access
2. Modifiers : insert(), emplace(), push\_back(), emplace\_back(), pop\_back(), resize(),swap(), erase(), clear()
3. 
4. A computer screen shot of a code

   Description automatically generated
5. Modifiers: insert – to insert at an specific spot,
6. Access Time – O(1), capacity doubles when size = capacity
7. Creating a Tree using vector inside vector

SETS : sorted set of unique obj – associative containers

* Red -black Tree – Self balancing Tree

MAPS: map<T1, T2>obj; T1 – key, T2 value type- store in sorted order

* Looping through map : for( auto &el1: Map) : el1.first and el1 second

Question Solution :  
  
  
1. Rat In a maze:  
//{ Driver Code Starts

// Initial template for C++

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Solution {

private:

    void func(vector<vector<char>> &path, vector<vector<int>> &maze, vector<char> &multi, int x, int y) {

        int n = maze.size();

        // Check if reached the bottom-right corner

        if (x == n - 1 && y == n - 1) {

            path.push\_back(multi);

            return;

        }

        // Mark the current cell as visited

        maze[x][y] = 0;

        // Explore all possible directions

        if (x + 1 < n && maze[x + 1][y] == 1) {

            multi.push\_back('D');

            func(path, maze, multi, x + 1, y);

            multi.pop\_back(); // backtrack

        }

        if (x - 1 >= 0 && maze[x - 1][y] == 1) {

            multi.push\_back('U');

            func(path, maze, multi, x - 1, y);

            multi.pop\_back(); // backtrack

        }

        if (y + 1 < n && maze[x][y + 1] == 1) {

            multi.push\_back('R');

            func(path, maze, multi, x, y + 1);

            multi.pop\_back(); // backtrack

        }

        if (y - 1 >= 0 && maze[x][y - 1] == 1) {

            multi.push\_back('L');

            func(path, maze, multi, x, y - 1);

            multi.pop\_back(); // backtrack

        }

        // Unmark the current cell as visited (backtrack)

        maze[x][y] = 1;

    }

public:

    vector<string> findPath(vector<vector<int>> &m, int n) {

        vector<char> multi;

        vector<vector<char>> path;

        vector<vector<int>> maze = m; // Make a copy of maze

        // Check if the starting point is accessible

        if (maze[0][0] == 1) {

            int x = 0;

            int y = 0;

            func(path, maze, multi, x, y);

        }

        // Convert vector<vector<char>> path to vector<string> result

        vector<string> result;

        for (auto &vec : path) {

            string s(vec.begin(), vec.end());

            result.push\_back(s);

        }

        return result;

    }

};

2. Permutations of the String:

class Solution {

private:

    void solve(vector<vector<int>>&ans,vector<int>nums,  int index){

        if(index >= nums.size()){

            ans.push\_back(nums);

            return;

        }

        else{

            for(int j= index; j<nums.size(); j++){

                swap(nums[index],nums[j]);

                solve( ans,nums, index+1);

                    swap(nums[index],nums[j]);

            }

        }

    }

public:

    vector<vector<int>> permute(vector<int>& nums) {

        vector<vector<int>> ans;

        int index =0;

        solve(ans,nums,index);

        return ans;

    }

};

3.Singly Linked List

#include <iostream>

using namespace std;

class Node {

    public:

    int data;

    Node \*next;

    Node(int d) : data(d), next(nullptr) {}

    void print\_list(Node\* &head);

    void push\_back(Node\* &head, int data);

    void addToFront(Node\* &head, int data);

    void insertAt(Node\* &head, int data, int index);

    void deleteAt(Node\* &head, int index);

};

Node\* findMid(Node\* &head){

    Node\* fast= head;

    Node\* slow = head;

    while(fast != NULL){

        fast = fast->next;

        if(fast != NULL){

            fast = fast->next;

        }

        slow = slow ->next;

    }

    return slow;

}

void reverseLL(Node\* &head){

    Node\* previous =NULL;

    Node\* current = head;

    Node\* temp = NULL;

    while(current != NULL){

        temp = current->next;

        current->next = previous;

        previous = current;

        current = temp;

    }

    head = previous;

}

void Node::print\_list(Node\* &head) {

    Node\* temp = head;

    while(temp != nullptr) {

        cout << temp->data << ", ";

        temp = temp->next;

    }

    cout << endl;

}

void Node::push\_back(Node\* &head, int d) {

    Node\* temp = head;

    Node\* newNode = new Node(d);

    if (head == nullptr) {

        head = newNode;

        return;

    }

    while (temp->next != nullptr) {

        temp = temp->next;

    }

    temp->next = newNode;

}

void Node::addToFront(Node\* &head, int d) {

    Node\* newNode = new Node(d);

    newNode->next = head;

    head = newNode;

}

void Node::insertAt(Node\* &head, int data, int index) {

    if (index == 0) {

        addToFront(head, data);

        return;

    }

    Node\* temp = head;

    int count = 0;

    while (count < index - 1 && temp != nullptr) {

        count++;

        temp = temp->next;

    }

    if (temp == nullptr) {

        push\_back(head, data);

        return;

    }

    Node\* newNode = new Node(data);

    newNode->next = temp->next;

    temp->next = newNode;

}

void Node::deleteAt(Node\* &head, int index) {

    Node\* temp = head;

    if (index == 0) {

        head = head->next;

        delete temp;

        return;

    }

    int count = 0;

    while (count < index - 1 && temp != nullptr) {

        count++;

        temp = temp->next;

    }

    if (temp == nullptr || temp->next == nullptr) {

        cout << "OUT OF BOUNDS" << endl;

        return;

    }

    Node\* nodeToDelete = temp->next;

    temp->next = temp->next->next;

    delete nodeToDelete;

}

bool checkPalindrome( Node\* head){

    Node\* temp = head;

    Node\* mid = findMid(head);

    reverseLL(mid);

    while(mid != NULL){

        if(temp->data == mid->data){

            temp = temp->next;

            mid = mid ->next;

        }

        else{

            return false;

        }

    }

    return true;

}

int main() {

    Node\* n1 = new Node(5);

    Node\* head = n1;

    //n1->push\_back(head, 5);

    n1->push\_back(head, 4);

    n1->push\_back(head, 4);

    n1->push\_back(head, 5);

    //n1->push\_back(head, 3);

    n1->print\_list(head);

    Node\* mid = findMid(head);

    cout << mid->data << endl;

    cout << checkPalindrome(head) << endl;

    return 0;

}

4.Doubly Linked List

#include <iostream>

using namespace std;

class Node {

    int data;

    Node \*next;

    Node \*previous;

    public:

        Node(int d) : data(d), next(nullptr), previous(nullptr) {}

        void print\_list(Node\* &head);

        void push\_back(Node\* &head, int data);

        void addToFront(Node\* &head, int data);

        void insertAt(Node\* &head, int data, int index);

        void deleteAt(Node\* &head, int index);

};

void Node:: print\_list(Node\* &head){

    Node\* temp = head;

    while(temp != NULL){

        cout << temp->data << ", ";

        temp = temp->next;

    }

    cout <<endl;

}

void Node::push\_back(Node\* &head, int d){

    Node\* temp = head;

    Node\* newNode = new Node(d);

    if (head == nullptr) {

        head = newNode;

        return;

    }

    while ( temp->next != NULL){

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->previous = temp;

}

void Node :: addToFront(Node\* &head,int d){

    Node\* temp = head;

    Node\* newNode = new Node(d);

    newNode->next = temp;

    temp->previous = newNode;

    head = newNode;

}

void Node :: insertAt(Node\* &head, int data , int index){

    if(index == 0){

        addToFront(head,data);

        return;

    }

    Node\* temp = head;

    int count = 0;

    while(count < index-1 && temp->next != NULL){

        count++;

        temp = temp->next;

    }

    if(temp->next == NULL){

        push\_back(head,data);

        return;

    }

    if(temp->next!= NULL){

        Node\* newNode = new Node(data);

        newNode->next = temp->next;

        newNode->previous = temp;

        temp->next->previous = newNode;

        temp->next = newNode;

    }

}

void Node :: deleteAt(Node\* &head, int index){

    Node\* temp = head;

    int count = 0;

    if(index == 0){

        temp->next->previous = NULL;

        delete temp;

        return;

    }

    while(count < index && temp != NULL){

        count ++;

        temp = temp->next;

    }

    if(temp == NULL){

        cout <<"OUT OF BOUNDS"<< endl;

        return;

    }

    temp->previous->next= temp->next;

    if(temp->next != NULL)

        temp->next->previous = temp->previous;

    delete temp;

}

int main(){

    Node\* n1 = new Node(6);

    Node\* head = n1;

    n1->push\_back(head,5);

    n1->print\_list(head);

    n1->addToFront(head, 4);

    n1->print\_list(head);

    n1->insertAt(head, 45, 3);

    n1->print\_list(head);

    n1->insertAt(head, 56, 3);

    n1->print\_list(head);

    n1->deleteAt(head, 5);

    n1->print\_list(head);

    return 0;

}

5.Reverse a Linked list

void Node :: reverseLL(Node\* &head){

    Node\* previous =NULL;

    Node\* current = head;

    Node\* temp = NULL;

    while(current != NULL){

        temp = current->next;

        current->next = previous;

        previous = current;

        current = temp;

    }

    head = previous;

}

6.kReverse a Linked List

/\*\*

 \* Definition for singly-linked list.

 \* class Node {

 \* public:

 \*     int data;

 \*     Node \*next;

 \*     Node() : data(0), next(nullptr) {}

 \*     Node(int x) : data(x), next(nullptr) {}

 \*     Node(int x, Node \*next) : data(x), next(next) {}

 \* };

 \*/

Node\* kReverse(Node\* head, int k) {

    // Edge case: empty list or no need to reverse

    if (head == NULL || k <= 1) {

        return head;

    }

    // Step 1: Reverse the first k nodes

    Node\* next = NULL;

    Node\* curr = head;

    Node\* prev = NULL;

    int count = 0;

    // Reverse first k nodes

    while (curr != NULL && count < k) {

        next = curr->next;

        curr->next = prev;

        prev = curr;

        curr = next;

        count++;

    }

    // Step 2: If we have reversed k nodes, recursively reverse the rest of the list

    if (count == k) {

        head->next = kReverse(next, k);

    } else {

        // If less than k nodes were left, we need to reverse back to original order

        curr = prev;

        prev = NULL;

        while (curr != NULL) {

            next = curr->next;

            curr->next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

    // Return the new head of the reversed sublist

    return prev;

}

7.Find Mid Of a Linked List

Node \*findMiddle(Node \*head) {

    // Write your code here

    Node\* temp = head;

    int count = 1;

    while(temp->next){

        temp = temp ->next;

        count++;

    }

    int mid = 1;

    temp = head;

    while(mid<(count/2 +1)){

        mid++;

        temp = temp->next;

    }

    //cout << count;

    head = temp;

    return head;

}

8.Floyd’s Cycle detection algorithm -https://youtu.be/PvrxZaH\_eZ4?si=Kzw1RSAAHEWjOb\_A

<https://youtu.be/PvrxZaH_eZ4?si=Kzw1RSAAHEWjOb_A>

bool detectloop(Node\* head);

Node\* startNode(Node\* head){

    if(!detectloop(head))

        return NULL;

    Node\* slow= head;

    Node\* fast = head;

    Node\* intersection= NULL;

    while(fast!=NULL && slow!=NULL){

        fast = fast->next;

        if(fast!= NULL){

            fast = fast->next;

        }

        slow= slow->next;

        if(slow==fast){

            intersection = slow;

            break;

        }

    }

    slow = head;

    fast = intersection;

    while(slow!=fast ){

        slow = slow->next;

        fast = fast->next;

    }

    return slow;

}

bool detectloop(Node\*head){

    if(head == NULL)

        return true;

    if(head->next == NULL)

        return false;

    Node\* slow = head;

    Node\* fast = head;

    while(fast != NULL && slow != NULL){

        fast = fast -> next;

        if(fast != NULL){

            fast = fast->next;

        }

        slow = slow->next;

        if(slow == fast)

            return true;

    }

    return false;

}

Node \*removeLoop(Node \*head)

{

    // Write your code here.

    if(!detectloop(head))

        return head;

    Node\* startingNode = startNode(head);

    Node\* fast =startingNode;

    while(fast->next != startingNode){

        fast = fast->next;

    }

    fast->next = NULL;

    return head;

}

9.Removing Duplicates From a Sorted Linked List

Node \* removeDuplicates(Node \*head)

{

    // Write your code here

    if(head == NULL)

        return head;

    Node\* temp= head;

    Node\* dupli;

    while(temp->next != NULL){

        if(temp->data != temp->next->data)

            temp = temp->next;

        else{

            dupli = temp->next;

            temp->next= temp->next->next;

            if(temp->next != NULL)

                temp->next->prev = temp;

            delete( dupli);

        }

    }

    return head;

}

10.Removing Duplicates from an unsorted Linked List

Node \*removeDuplicates(Node \*head)

{

    unordered\_map <int, bool> arr;

    Node\* temp = head;

    Node\* prev = NULL;

    while(temp != NULL){

        int val = temp->data;

        if(arr[val]){

            prev->next = temp->next;

            delete(temp);

            temp = prev->next;

        }

        else{

            arr[val] = true;

            prev = temp;

            temp = temp->next;

        }

    }

    return head;

}

11.Sorting 0,1,2

Node\* sortList(Node \*head){

    // Write your code here.

    if(head == NULL)

        return head;

    map <int, int> hash;

    Node\* temp = head;

    while(temp != NULL){

        hash[temp->data] += 1;

        temp = temp->next;

    }

    temp = head;

    for (auto it = hash.begin(); it != hash.end(); ++it) {

        int data = it->first;

        int count = it->second;

        for (int i = 0; i < count; ++i) {

            temp->data = data;

            temp = temp->next;

        }

    }

    return head;

}

12.Merging two sorted Linked List

Node<int>\* sortTwoLists(Node<int>\* first, Node<int>\* second)

{

    // Write your code here.

    if(first == NULL)

        return second;

    if(second == NULL)

        return first;

    Node<int>\* curr = first;

    Node<int>\* previous = NULL;

    Node<int>\* temp2 = second;

    if(temp2->data < curr->data ){

           previous = temp2;

           temp2 = temp2->next;

           previous->next = curr;

           first = previous;

        }

    while(curr != NULL && temp2 != NULL){

        if(temp2->data < curr->data ){

           previous->next = temp2;

           temp2 = temp2->next;

           previous->next->next = curr;

           previous = previous->next;

        }

        else{

            previous = curr;

            curr = curr->next;

        }

    }

    if(temp2 !=NULL){

        previous->next = temp2;

    }

    return first;

}

13.Checking Palindrome for a Linked List

Node\* findMid(Node\* &head){

    Node\* fast= head;

    Node\* slow = head;

    while(fast != NULL){

        fast = fast->next;

        if(fast != NULL){

            fast = fast->next;

        }

        slow = slow ->next;

    }

    return slow;

}

void reverseLL(Node\* &head){

    Node\* previous =NULL;

    Node\* current = head;

    Node\* temp = NULL;

    while(current != NULL){

        temp = current->next;

        current->next = previous;

        previous = current;

        current = temp;

    }

    head = previous;

}

class Solution{

  public:

    //Function to check whether the list is palindrome.

    bool isPalindrome( Node\* head){

    Node\* temp = head;

    Node\* mid = findMid(head);

    reverseLL(mid);

    while(mid != NULL){

        if(temp->data == mid->data){

            temp = temp->next;

            mid = mid ->next;

        }

        else{

            return false;

        }

    }

    return true;

}

};

14.Addiing Two Numbers using Linked List

\*/

void reverseLL(Node\* &head){

    Node\* previous = NULL;

    Node\* curr = head;

    Node\* next= NULL;

    while( curr != NULL){

        next = curr->next;

        curr->next = previous;

        previous = curr;

        curr = next;

    }

    head = previous;

}

class Solution

{

    public:

    //Function to add two numbers represented by linked list.

    // Function to add two numbers represented by linked list.

    struct Node\* addTwoLists(struct Node\* num1, struct Node\* num2)

    {

        // Reverse both input lists

        reverseLL(num1);

        reverseLL(num2);

        Node\* ans = NULL;

        Node\* head = NULL;

        int carry = 0;

        // Traverse both lists

        while (num1 != NULL || num2 != NULL || carry != 0) {

            int sum = carry;

            if (num1 != NULL) {

                sum += num1->data;

                num1 = num1->next;  // Advance pointer

            }

            if (num2 != NULL) {

                sum += num2->data;

                num2 = num2->next;  // Advance pointer

            }

            int digit = sum % 10;

            carry = sum / 10;

            Node\* newNode = new Node(digit);

            if (ans == NULL) {

                ans = newNode;

                head = ans;

            } else {

                ans->next = newNode;

                ans = ans->next;

            }

        }

        // Reverse the result list before returning

        reverseLL(head);

        while(head->data == 0 && head->next != NULL){

            head = head->next;

        }

        if(head->next == NULL){

            head->data = 0;

        }

        return head;

    }

};

15.Merge Sort in a Linked List

node\* sortTwoLists(node\* first, node\* second) {

    if (first == nullptr)

        return second;

    if (second == nullptr)

        return first;

    node\* ans = new node(-1); // Dummy node

    node\* temp = ans;

    while (first != nullptr && second != nullptr) {

        if (first->data < second->data) {

            temp->next = first;

            temp = first;

            first = first->next;

        } else {

            temp->next = second;

            temp = second;

            second = second->next;

        }

    }

    // Attach remaining nodes

    while (first != nullptr) {

        temp->next = first;

        temp = first;

        first = first->next;

    }

    while (second != nullptr) {

        temp->next = second;

        temp = second;

        second = second->next;

    }

    node\* sortedHead = ans->next;

    delete ans; // Free dummy node

    return sortedHead;

}

node\* findMid(node\* head) {

    if (head == nullptr || head->next == nullptr)

        return head;

    node\* slow = head;

    node\* fast = head->next;

    while (fast != nullptr && fast->next != nullptr) {

        slow = slow->next;

        fast = fast->next->next;

    }

    return slow;

}

node\* mergeSort(node\* head) {

    if (head == nullptr || head->next == nullptr)

        return head;

    node\* mid = findMid(head);

    node\* left = head;

    node\* right = mid->next;

    mid->next = nullptr;

    left = mergeSort(left);

    right = mergeSort(right);

    return sortTwoLists(left, right);

}

## Stacks

Que: Check valid parenthesis

bool isValidParenthesis(string s)

{

    // Write your code here.

    if(s.size() % 2 !=0){

        return false;

    }

    stack <char> valid;

    for(int i=0; i<s.size(); i++){

        if(s[i]== '(' || s[i] == '{' || s[i] == '['){

            valid.push(s[i]);

        } else{

            if (valid.empty()) {

                return false;

            }

            if ((valid.top() == '(' && s[i] == ')') ||

                (valid.top() == '{' && s[i] == '}') ||

                (valid.top() == '[' && s[i] == ']')) {

                valid.pop();

            }

        }

    }

    if(valid.empty()){

        return true;

    }

    return false;

}

Que: Pushing at the bottom of the stack

#include <bits/stdc++.h>

void solve(stack<int>& myStack,stack<int>& temp){

    if(myStack.empty()){

        return;

    }

    int num = myStack.top();

    myStack.pop();

    temp.push(num);

    solve(myStack,temp);

}

stack<int> pushAtBottom(stack<int>& myStack, int x)

{

    // Write your code here.

    stack<int> temp;

    solve(myStack,temp);

    myStack.push(x);

    solve(temp,myStack);

    return myStack;

}

Que 3: Reversing a Stack Using recursion

void insertAtBottom(stack<int> &s, int num) {

    if (s.empty()) {

        s.push(num);

        return;

    }

    int n = s.top();

    s.pop();

    insertAtBottom(s, num);

    s.push(n);

}

void reverseStack(stack<int> &stack) {

    if (stack.empty()) {

        return;

    }

    int num = stack.top();

    stack.pop();

    reverseStack(stack);

    insertAtBottom(stack, num);

}

Que 4 : NEXTSMALLER ELEMENT:

For O(n) complexity start from the back and then iterate through the stack and array

#include <stack>

vector<int> nextSmallerElement(vector<int> &arr, int n)

{

    // Write your code here.

    stack<int> elem;

    int temp;

    elem.push(-1);

    for(int i=n-1 ; i>= 0; i--){

        if(arr[i] > elem.top()){

            temp = arr[i];

            arr[i] = elem.top();

            elem.push(temp);

        }

        else{

            while(elem.top()>= arr[i]){

                elem.pop();

            }

            temp = arr[i];

            arr[i] = elem.top();

            elem.push(temp);

        }

    }

    return arr;

}

Que 5: Largest Rectangle in Histogram

Note: Consider finding nextSmaller and previous Smaller element

class Solution {

    vector<int> prevSmallerElement(vector<int> &arr, int n)

{

    // Write your code here.

    stack<int> elem;

    vector <int> result(n);

    int temp;

    int count;

    elem.push(-1);

    for(int i=0 ; i<= n-1; i++){

            temp = arr[i];

            while(elem.top()!=-1 &&arr[elem.top()]>= temp){

                elem.pop();

            }

            result[i] = elem.top();

            elem.push(i);

    }

    return result;

}

vector<int> nextSmallerElement(vector<int> &arr, int n)

{

    // Write your code here.

    stack<int> elem;

    vector <int> result(n);

    int temp;

    elem.push(-1);

    for(int i=n-1 ; i>= 0; i--){

            temp = arr[i];

            while(elem.top() != -1 && arr[elem.top()]>= temp){

                elem.pop();

            }

            result[i] = elem.top();

            elem.push(i);

    }

    return result;

}

public:

    int largestRectangleArea(vector<int>& heights) {

        vector <int> prevIndex;

        vector <int> nextIndex;

        nextIndex = nextSmallerElement(heights, heights.size());

        prevIndex = prevSmallerElement(heights, heights.size());

        int width;

        int length;

        int maxArea=INT\_MIN;

        int newArea;

        for(int i=0; i<heights.size(); i++){

        if(nextIndex[i] ==-1){

            nextIndex[i] = heights.size();

        }

        width=nextIndex[i] - prevIndex[i] - 1;

        length = heights[i];

        newArea = length \* width;

        if(newArea > maxArea){

            maxArea = newArea;

        }

    }

    return maxArea;

    }

};

Que 6: Celebrity Problem

//{ Driver Code Starts

// Initial template for C++

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

  public:

    // Function to find if there is a celebrity in the party or not.

    int celebrity(vector<vector<int> >& mat) {

        // code here

    int colIndex;

    int checkcol;

    vector <int> possibleceleb;

    for(int i=0; i<mat.size(); i++){

        colIndex=0;

        for(int j=0; j<mat[i].size(); j++){

            if(mat[i][j]==1){

                break;

            }

            colIndex++;

        }

        if(colIndex == mat[i].size()){

            possibleceleb.push\_back(i);

        }

    }

    for(int i=0 ; i<possibleceleb.size(); i++){

        checkcol =possibleceleb[i];

        colIndex=0;

        for(int j = 0; j<mat.size();j++){

            if(mat[j][checkcol] == 0 && j != checkcol){

                break;

            }

            colIndex++;

        }

        if(colIndex == mat.size()){

            return checkcol;

        }

    }

    return -1;

    }

};

//{ Driver Code Starts.

int main() {

    int t;

    cin >> t;

    while (t--) {

        int n;

        cin >> n;

        vector<vector<int> > M(n, vector<int>(n, 0));

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                cin >> M[i][j];

            }

        }

        Solution ob;

        cout << ob.celebrity(M) << endl;

    }

}

// } Driver Code Ends

Que 7: Maximum Area in a matrix containing 0 and 1

//{ Driver Code Starts

// Initial template for C++

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

  public:

    // Function to find if there is a celebrity in the party or not.

    int celebrity(vector<vector<int> >& mat) {

        // code here

    int colIndex;

    int checkcol;

    vector <int> possibleceleb;

    for(int i=0; i<mat.size(); i++){

        colIndex=0;

        for(int j=0; j<mat[i].size(); j++){

            if(mat[i][j]==1){

                break;

            }

            colIndex++;

        }

        if(colIndex == mat[i].size()){

            possibleceleb.push\_back(i);

        }

    }

    for(int i=0 ; i<possibleceleb.size(); i++){

        checkcol =possibleceleb[i];

        colIndex=0;

        for(int j = 0; j<mat.size();j++){

            if(mat[j][checkcol] == 0 && j != checkcol){

                break;

            }

            colIndex++;

        }

        if(colIndex == mat.size()){

            return checkcol;

        }

    }

    return -1;

    }

};

//{ Driver Code Starts.

int main() {

    int t;

    cin >> t;

    while (t--) {

        int n;

        cin >> n;

        vector<vector<int> > M(n, vector<int>(n, 0));

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                cin >> M[i][j];

            }

        }

        Solution ob;

        cout << ob.celebrity(M) << endl;

    }

}

// } Driver Code Ends

TOP 150:

Q1. Shift a string by k times or shift an array k times

A screen shot of a computer program

Description automatically generated

Trees: Non-linear Data structures

Binary Tree <= 2 children

Question : Level Order Traversal - Hint : Use Queue – Same as BFS

void levelOrder(node\* root){

    queue<node\*> fr;

    fr.push(root);

    fr.push(NULL);

    while(!fr.empty()){

        node\* temp = fr.front();

        fr.pop();

        if(temp == NULL){

            cout << endl;

            if(!fr.empty()){

                fr.push(NULL);

            }

        }

        else{

            cout << temp->data << " " ;

            if(temp->left){

                fr.push(temp->left);

            }

            if(temp->right){

                fr.push(temp->right);

            }

        }

    }

}

Inorder : LNR

void inOrder(node \* root){

    if(root == NULL){

        return;

    }

    inOrder(root->left);

    cout << root->data << " , ";

    inOrder(root->right);

}

Follow :

PreOrder : NLR

PostOrder : LRN

**Note: Build A tree from Level Order Traversal: whenever Level Order Traversal Use Queue**

**Que: Depth of a Binary Tree:**

Longest path between your root node and a leaf node

Start the Height for root to be 1 and then dive into the left and right and select the max out of them.

// Online C++ compiler to run C++ program online

#include <iostream>

#include <queue>

using namespace std;

class node{

public:

int data;

node\* left;

node\* right;

node(int d){

data = d;

left = nullptr;

right = nullptr;

}

node(){}

};

node \* buildTree(node\* root){

cout << "Enter the Data " << endl;

int d;

cin >> d;

if(d == -1){

return NULL;

}

root = new node(d);

cout << "enter the data for the left child of the " << root->data << endl;

root->left = buildTree(root->left);

cout << "enter the data for the right child of the " << root->data << endl;

root->right = buildTree(root->right);

return root;

}

void levelOrder(node\* root){

queue<node\*> fr;

fr.push(root);

fr.push(NULL);

while(!fr.empty()){

node\* temp = fr.front();

fr.pop();

if(temp == NULL){

cout << endl;

if(!fr.empty()){

fr.push(NULL);

}

}

else{

cout << temp->data << " " ;

if(temp->left){

fr.push(temp->left);

}

if(temp->right){

fr.push(temp->right);

}

}

}

}

void inOrder(node \* root){

if(root == NULL){

return;

}

inOrder(root->left);

cout << root->data << " , ";

inOrder(root->right);

}

int main() {

// Write C++ code here

node\* tree1;

tree1 = buildTree(tree1);

// 1 2 7 -1 -1 -1 3 4 -1 -1 -1

cout << "Level Order Traversal "<< endl;

levelOrder(tree1);

cout << endl;

cout << "Inorder Traversal" << endl;

inOrder(tree1);

return 0;

}

Zig Zag Traversal: Maintain Two vectors one final and one temp inside , then maintain a flag to switch the traversal direction

vector <int> zigZagTraversal(Node\* root)

    {

        // Code here

        vector <int> result;

        queue <Node\*> nodes;

        if(root == NULL){

            return result;

        }

        bool flag = true;

        nodes.push(root);

        while(!nodes.empty()){

           int size = nodes.size();

           vector <int> temp(size);

           for(int i=0; i<size; i++){

                Node\* frontNode = nodes.front();

                nodes.pop();

                int index;

                if(flag){

                    index = i;

                }else{

                    index = size-i-1;

                }

                temp[index] = frontNode->data;

                if(frontNode->left){

                    nodes.push(frontNode->left);

                }

                if(frontNode->right){

                    nodes.push(frontNode->right);

                }

           }

           flag = !flag;

           for(auto i:temp){

               result.push\_back(i);

           }

        }

        return result;

    }