**Graph Intro, BFS, DFS**

**From where I am going to study and practice Graph,**

* CLRS
* GFG
* Interview bit
* Codechef
* Hackerrank
* Edx competitive programming course
* Graph theory notes

Note : - Here I am not going to write the whole theory and complete explanation. This repository is just for quick revision and to remind me what I have studied till now and provide the link of the original resource from where I have actually studied that topic. Therefore if I want to check the original resource then I can easily check that.

Note:- the order of this file is in that order in which I have studied them.

**From GFG,**

1. **Graph and its Representation**

Theory link: <https://www.geeksforgeeks.org/graph-and-its-representations/>

Problem link: <https://practice.geeksforgeeks.org/problems/print-adjacency-list/0>

Problem Solution:

#include<bits/stdc++.h>

using namespace std;

void add\_edge(vector<int> \*arr, int a, int b){ //pointer to array of vectors

arr[a].push\_back(b);

arr[b].push\_back(a);

}

void print\_list(vector<int> arr,int i){ //int vector

cout<<i;

if(arr.size()>=1){

cout<<"-> ";

for(int i=0;i<arr.size()-1;i++){

cout<<arr[i]<<"-> ";

}

cout<<arr[arr.size()-1];

}

cout<<endl;

}

int main()

{

//code

// This code works well in othere ide

long long t;

cin>>t;

while(t--){

int v,e,a,b;

cin>>v>>e;

vector<int> arr[v]; // array(of size v) of int vectors

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

cin>>a>>b;

add\_edge(arr,a,b); // passing array of vectors

}

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

print\_list(arr[i],i); // passing a vector row

}

}

return 0;

}

Method 2: use STL linked list : see my another solution

#include<bits/stdc++.h>

using namespace std;

void add\_edge(list<int> \*lst, int a, int b){ //pointer to array of lists

lst[a].push\_back(b);

lst[b].push\_back(a);

}

void print\_list(list<int> lst,int i){ //int list

cout<<i;

if(lst.size()>=1){

cout<<"-> ";

// for(int i=0;i<lst.size()-1;i++){

// cout<< lst[i]<<"-> "; this will not work because list don't have indexing

// }

// cout<< lst[lst.size()-1];

int c=0,s=lst.size();

for(auto item:lst) {

c++;

if(c<s)

cout<<item<<"-> ";

}

cout<<lst.back();

}

cout<<endl;

}

int main()

{

//code

// stl list implementation

long long t;

cin>>t;

while(t--){

int v,e,a,b;

cin>>v>>e;

list<int> lst[v]; // array(of size v) of int list

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

cin>>a>>b;

add\_edge(lst,a,b); // passing array of lists

}

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

print\_list(lst[i],i); // passing a list row

}

}

return 0;

}

Method 3: this will work for any kind of node string etc. and using hashmap.

1. **Determine Sink node exists or not**

Sink node: node which have outdegree =0 and in degree=n-1 and there exists atmost one sink node

**Theory link:** <https://www.geeksforgeeks.org/determine-whether-universal-sink-exists-directed-graph/>

**Theory link:celebrity problem:** <https://www.geeksforgeeks.org/the-celebrity-problem/>

problem link: <https://practice.geeksforgeeks.org/problems/the-celebrity-problem/1>

video solution: <https://www.youtube.com/watch?time_continue=167&v=LtGnA5L6LIk>

there are three solutions with linear complexity based on same logic

sol1. Exercise sol 22.1.6

sol2. Stack implementation

sol3. Two pointers

# related theory : Number of sink nodes in a graph

# theory link: <https://www.geeksforgeeks.org/number-sink-nodes-graph/>

# definition is change for sink node here

1. **BFS**

Theory link: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/> for refrence not so much important

Problem link: <https://practice.geeksforgeeks.org/problems/bfs-traversal-of-graph/1>

Application: <https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/>

1. **DFS**

Theory link: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

Problem link: <https://practice.geeksforgeeks.org/problems/depth-first-traversal-for-a-graph/1>

Application: <https://www.geeksforgeeks.org/applications-of-depth-first-search/>

Iterative dfs: <https://www.geeksforgeeks.org/iterative-depth-first-traversal/>

1. Bfs shortest path problem on grid:

Nice video: <https://www.youtube.com/watch?v=KiCBXu4P-2Y&list=PLDV1Zeh2NRsDGO4--qE8yH72HFL1Km93P&index=5>

Shortest distance between two cell in matrix

Theory link: <https://www.geeksforgeeks.org/shortest-distance-two-cells-matrix-grid/>

Code link: <https://ide.geeksforgeeks.org/ZfdGveLSoq>

1. No. of connected components in undirected graph

Nice video: <https://www.youtube.com/watch?v=7fujbpJ0LB4&list=PLDV1Zeh2NRsDGO4--qE8yH72HFL1Km93P&index=3>

6.a No. of X shapes

Problem link: <https://practice.geeksforgeeks.org/problems/x-total-shapes/0/?ref=self>

Dynamic memory and vector implementation

6.b No. of islands

Similar to number of connected components in undirected graph

Theory link : <https://www.geeksforgeeks.org/find-number-of-islands/>

Problem link: <https://practice.geeksforgeeks.org/problems/find-the-number-of-islands/1>

1. Minimum time required to rot all oranges

Theory link: <https://www.geeksforgeeks.org/minimum-time-required-so-that-all-oranges-become-rotten/>

Problem link: <https://practice.geeksforgeeks.org/problems/rotten-oranges/0>

# Count number of islands where every island is row-wise and column-wise separated

# Theory link: <https://www.geeksforgeeks.org/count-number-islands-every-island-separated-line/>

1. Smallest sequence with given prime

Problem link: <https://www.interviewbit.com/problems/smallest-sequence-with-given-primes/>

1. Valid path between circles

TL: <https://www.geeksforgeeks.org/path-rectangle-containing-circles/>

Problem link: <https://www.interviewbit.com/problems/valid-path/>

1. Fire escape routes

Problem link: <https://www.codechef.com/problems/FIRESC>

1. Snake and Ladder

Based on shortest path using bfs

Theory link: <https://www.geeksforgeeks.org/snake-ladder-problem-2/>

Problem link: <https://practice.geeksforgeeks.org/problems/snake-and-ladder-problem/0>

# Graph implementation using STL for competitive programming | Set 1 (DFS of Unweighted and Undirected)

# Theory link: <https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-1-dfs-of-unweighted-and-undirected/>

# Graph implementation using STL for competitive programming | Set 2 (Weighted graph)

# Theory link: <https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-2-weighted-graph/>

# Graph representations using set and hash

# Tl: <https://www.geeksforgeeks.org/graph-representations-using-set-hash/>