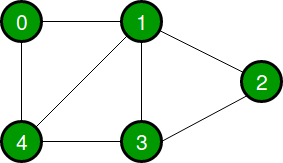
1.Graphs Implementation using STL:



#include<iostream>

#include<vector>

using namespace std;

void addedge(vector <int> V[],int u ,int v)

{

V[u].push\_back(v);

V[v].push\_back(u);

}

void printGraph(vector<int> adj[],int V)

{

for(int v=0;v<V;v++)

{

cout<<"\n Adjacency list of vertex: "

<<v<<" head ";

for(auto i:adj[v])

{

cout<<i;

}

cout<<"\n";}

}

int main()

{

vector <int> v[5];

addedge(v,0,1);

addedge(v,0,4);

addedge(v,1,2);

addedge(v,1,3);

addedge(v,1,4);

addedge(v,2,3);

addedge(v,3,4);

printGraph(v,5);

return 0;

}

Output:

Adjacency list of vertex 0

head -> 1-> 4

Adjacency list of vertex 1

head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2

head -> 1-> 3

Adjacency list of vertex 3

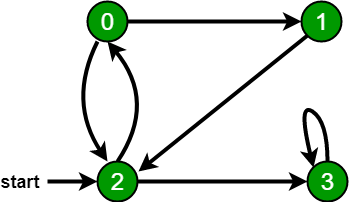
head -> 1-> 2-> 4

Adjacency list of vertex 4

head -> 0-> 1-> 3

2.GRAFH(BFS):(Directed Graph)

[Breadth First Traversal](https://practice.geeksforgeeks.org/problems/bfs-traversal-of-graph/1)



//Directed Graph

#include<iostream>

#include<vector>

#include <list>

using namespace std;

void graph(vector <int> adj[],int u,int v)

{

adj[u].push\_back(v);

}

void display(vector <int> adj[],int V)

{

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

{

for(auto x:adj[i])

cout<<x<<" ";

cout<<"\n";

}}

void BFS(vector <int> adj[],int v,int V)

{int s;

bool\* visited=new bool[V];

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

visited[i]=false;

list<int> q;

visited[v]=true;

q.push\_back(v);

while(!q.empty())

{

v=q.front();

cout<<v;

q.pop\_front();

for(auto x:adj[v])

{

if(!visited[x]){

visited[x]=true;

q.push\_back(x);

}

}

}

}

int main()

{

int t,v,e,e1,e2;

cin>>t;

while(t--){

cin>>v>>e;

vector <int> adj[v];

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

cin>>e1>>e2;

graph(adj,e1,e2);

}

BFS(adj,2,v) //BFS FROM NODE 2----------------

return 0;

}

3.Graph DFS

[Depth First Traversal](https://practice.geeksforgeeks.org/problems/depth-first-traversal-for-a-graph/1)

//Directed Graph

#include<iostream>

#include<stack>

#include<vector>

using namespace std;

stack<int> st;

void DFS(vector<int>[],int,int,bool[]);

void graph(vector <int> adj[],int u,int v)

{

adj[u].push\_back(v);

}

void display(vector <int> adj[],int V)

{

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

{

for(auto x:adj[i])

cout<<x<<" ";

cout<<"\n";

}}

void DFS(vector<int> adj[],int s,int V,bool visited[])

{

cout<<s;

st.push(s);

if(!st.empty())

{

for(auto x: adj[s])

{

if(!visited[x]){

visited[x]=true;

DFS(adj,x,V,visited);

}

}

st.pop();

}

}

int main()

{

int V=4,s=2;

vector <int> adj[4];

graph(adj,0,1);

graph(adj,0,2);

graph(adj,1,2);

graph(adj,2,0);

graph(adj,2,3);

graph(adj,3,3);

bool visited[V];

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

visited[i]=false;

visited[s]=true;

DFS(adj,s,V,visited);

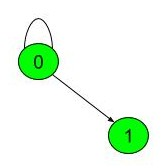
return 0;

}

4. Detect Cycle in a Graph

Additional--No. Of Cycles in a Graph

[Detect cycle in a directed graph](https://practice.geeksforgeeks.org/problems/detect-cycle-in-a-directed-graph/1)



#include<bits/stdc++.h>

using namespace std;

void graph(vector <int> adj[],int u,int v)

{

adj[u].push\_back(v);

}

bool BFS(vector <int> adj[],int v,int V,bool visited[])

{int s;

list<int> q;

visited[v]=true;

q.push\_back(v);

while(!q.empty())

{v=q.front();

q.pop\_front();

for(auto x:adj[v])

{

if(!visited[x])

{

visited[x]=true;

q.push\_back(x);

}

else{

return false;}

}}return true;}

int main()

{

int t,v,e,e1,e2,count=0;

cin>>t;

while(t--){

cin>>v>>e;

vector <int> adj[v];

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

cin>>e1>>e2;

graph(adj,e1,e2);

}

bool visited[v];

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

visited[i]=false;

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

{

if(!visited[i])

{

bool x=BFS(adj,i,v,visited);

if(!x)

count++;

}

}

if(count>0)

cout<<"CYCLE:"<<count;

else

cout<<"NO CYCLE";

count=0;

}

return 0;

}

return 0;

}

5.BFS (Disconnected Graph):---------------------------

#include<bits/stdc++.h>

using namespace std;

void BFS(vector <int> [],int,int,bool []);

void graph(vector <int> adj[],int u,int v)

{

adj[u].push\_back(v);

}

void display(vector <int> adj[],int V)

{

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

{

for(auto x:adj[i])

cout<<x<<" ";

cout<<"\n";}}

void BFS(vector <int> adj[],int v,int V,bool visited[]){

list<int> q;

visited[v]=true;

q.push\_back(v);

while(!q.empty())

{

v=q.front();

cout<<v;

q.pop\_front();

for(auto x:adj[v]){

if(!visited[x]){

visited[x]=true;

q.push\_back(x);}}

}

}

void start(vector<int> adj[],int V)

{

bool visited[V];

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

visited[i]=false;

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

{

if(!visited[i])

BFS(adj,i,V,visited);

}

}

int main()

{

int V=5;

vector <int> adj[V];

graph(adj, 0, 4);

graph(adj, 1, 2);

graph(adj, 1, 3);

graph(adj, 1, 4);

graph(adj, 2, 3);

graph(adj, 3, 4);

//display(adj,4);

start(adj,V);

return 0;

}

6.(Weighted Graph(Directed))

#include<bits/stdc++.h>

using namespace std;

void addedge(vector <pair<int,int>> adj[],int u,int v,int w)

{

adj[u].push\_back(make\_pair(v,w));

}

void printgraph(vector <pair<int,int>> adj[],int V)

{

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

for(auto itr = adj[i].begin(); itr!=adj[i].end(); itr++){

cout<<"Node : "<<itr->first<<"weight : "<<itr->second;

}

cout<<endl;

}

}

int main()

{

int V=5;

vector <pair<int,int>> adj[V];

addedge(adj,0,1,10);

addedge(adj, 0, 4, 20);

addedge(adj, 1, 2, 30);

addedge(adj, 1, 3, 40);

addedge(adj, 1, 4, 50);

addedge(adj, 2, 3, 60);

addedge(adj, 3, 4, 70);

printgraph(adj,V);

return 0;

}

7. **Minimum Swaps to Sort**(without Graph):-

[Minimum Swaps](https://practice.geeksforgeeks.org/problems/minimum-swaps/1)

#include<iostream>

#include<algorithm>

using namespace std;

int minswaps(int a[],int,int,int,int);

int main()

{

int t,n,i;

cin>>t;

while(t--)

{

cin>>n;

int a[n];

for(int i=0;i<n;i++)

cin>>a[i];

cout<<minswaps(a,n,n-1,0,0);

}

return 0;

}

int minswaps(int a[], int n,int high, int low,int count)

{

if(low!=n-1){

int min=100000,i,x;

bool y=false;

while(low<high){

x=a[high]-a[low];

if(x<0 && x<min)

{

min=x;

i=high;

y=true;}

high--;}

if(y==true){

swap(a[low],a[i]);

count++;}

minswaps(a,n,n-1,low+1,count);}

else

return count;}