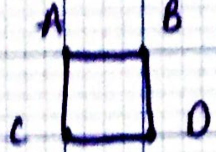
DINH BẢO DUY ITITIU19107

10. Graph 3 is simple.

Graph 4:



Remove 1 (a,b) edge and 2 (b,d) edge

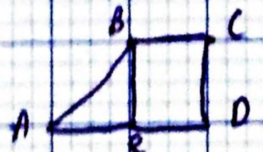
Graph 5:

Remove loops on vertex a, b, d
Remove 1 (a,b) edge and 1 (b,d) edge

Graph 6:

Remove 1 (A,C) edge and 1 (b,d) edge.

Graph 7:

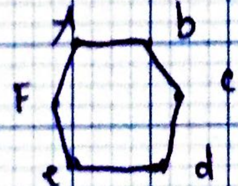


Remove loop at c, e, 1 edge (c,d)

Graph 8:

Remove loops at e, d 2 edge (c,d)
1 edge (a,b), (a,e), (b,c)

Graph 9:



Remove loop at f, d, 1 edge (a,b),
(b,c), (e,d)

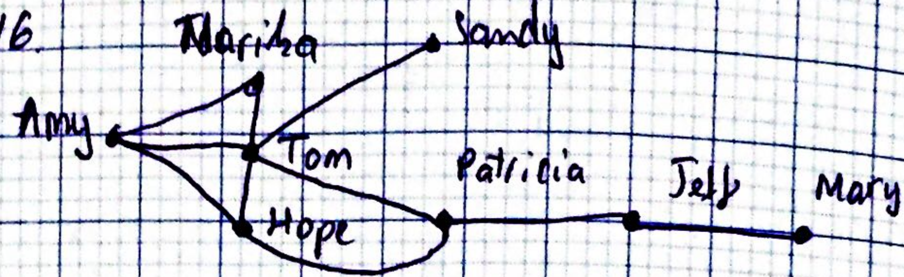
12.

If uRv then there is edge $\{u, v\}$, and graph is undirected
So $\{u, v\} = \{v, u\} \Rightarrow$ Symmetric. And since loops exist at every vertex $\Rightarrow uRu$ exist $\forall u \in G$. $\Rightarrow$ reflexive relation.

LHP

14 Hawk is connected to owl, racoon & crow in the graph so it competes with those animals

16.



6.

We solve this using graph with each person is a vertex and edge between 2 vertices represent a hand shake between those people ⇒ Degree of a vertex is the number of people that person shakes hands with. We have:

$\sum_v \deg(v) = 2e$ and e is positive int ⇒ Sum of number of handshakes is even.

10.

Graph 7:

a has ~~1~~ 1 out, 3 in b has 2 out, 1 in c has 1 out, 2 in

d has 3 out, 1 in ⇒ ins = outs = edges = 7.

Graph 8:

a has 2 in, 2 out b has 3 in, 4 out c has 2 in, 1 out

d has 1 in, 1 out ⇒ ins = outs = edges = 8.

Graph 9:

a has 6 in 1 out b has 1 in, 5 out. c has 2 in 5 out.
d has 4 in 2 out. $\Rightarrow$ ins = outs = edges = 13.

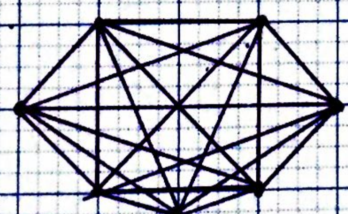
18.

~~Simple graph with 2 vertex has 2 possibilities~~:

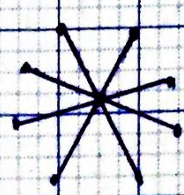
We have n vertices in graph $(n \geq 2)$, so all possible degrees a vertex could have is $0, \dots, n-1$ degrees so for n vertices to have all unique number of degrees ~~ai = 0, for~~ each vertex have $0, \dots, n-1$ degrees.
This is impossible because $n-1$ degree means it connects with every other vertex $\Rightarrow$ 0 degree not possible
$\Rightarrow$ At least 1 repeat of degree number in simple graph. with $n \geq 2$.

20.

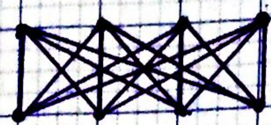
a) K_7



b) $K_{1,8}$



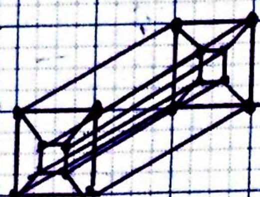
c) $K_{4,4}$



d) C_7:

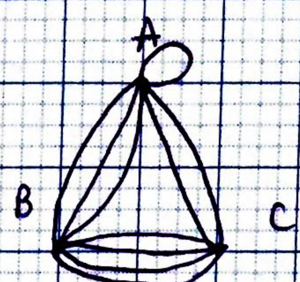
e) W_7:

f) Q_4:

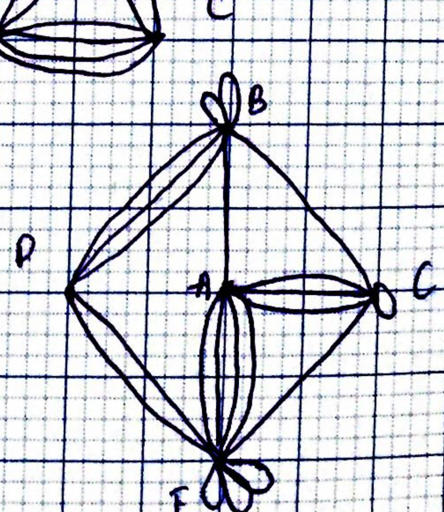


10.
$$\begin{array}{c|ccc|} & & & \\ A & 0 & 1 & 0 \\ B & 1 & 0 & 1 \\ C & 0 & 1 & 0 \\ & A & B & C \end{array}$$

16.
$$\begin{array}{c|ccc|} & & & \\ A & 1 & 3 & 2 \\ B & 3 & 0 & 4 \\ C & 2 & 4 & 0 \\ & A & B & C \end{array}$$

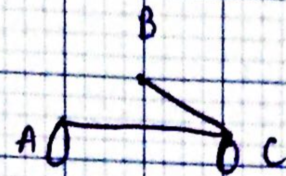


18.
$$\begin{array}{c|ccccc|} & & & & & \\ A & 0 & 1 & 3 & 0 & 4 \\ B & 1 & 2 & 1 & 3 & 0 \\ C & 3 & 1 & 1 & 0 & 1 \\ D & 0 & 3 & 0 & 0 & 2 \\ E & 4 & 0 & 1 & 2 & 3 \\ & A & B & C & D & E \end{array}$$



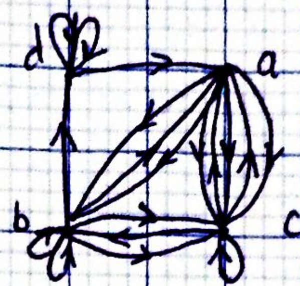
22.

$$\begin{array}{c|c}
 & \begin{array}{ccc} & & \end{array} \\
\begin{array}{c} A \\ B \\ C \end{array} & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\
 & \begin{array}{ccc} A & B & C \end{array}
\end{array}$$



24.

$$\begin{array}{cc}
\begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 2 & 3 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix} \\
 & \begin{array}{cccc} A & B & C & D \end{array}
\end{array}$$



```
#include<iostream>
#include<vector>
#include<stack>
#include<unordered_map>
using namespace std;
class Graph{

    int vertex;
    vector<unordered_map<int,int>> adj;

    public:
        Graph(int v){
            vertex = v;
            adj = vector<unordered_map<int,int>>(v+1);
        }
        void addEdge(int u, int v){
            adj[u][v] = 1;
            adj[v][u] = 1;
        }
        void removeEdge(int v,int u){
            adj[v].erase(u);
            adj[u].erase(v);
        }

        // function checks if the graph contains a euler
path/circuit or not
        void printEulerPathCircuit(){

            int odd = 0; // number of vertices with odd degree
            int oddVertex = 0; // it stores vertex with odd
degree if it exists

            for(int i=1;i<=vertex;++i){
                if(adj[i].size()%2==1){
                    ++odd;
                    oddVertex = i;
                }
```

```
        }

        if(odd==0){
            cout<<"Euler Circuit: ";
            printEuler(1);
        }
        else if(odd==2){
            cout<<"Euler Path: ";
            printEuler(oddVertex);
        }
        else{
            cout<<"Euler Path/Circuit Doesn't Exist"<<endl;
        }

    }
    void printEuler(int v){

        stack<int> cpath;    // current path
        stack<int> epath;    // euler path

        cpath.push(v);        // euler path starts from v

        while(!cpath.empty()){
            int u = cpath.top();

            if(adj[u].size()==0){
                epath.push(u);
                cpath.pop();
            }
            else{
                cpath.push(adj[u].begin()->first);
                removeEdge(u,adj[u].begin()->first);
            }
        }

        while(!epath.empty()){
            cout<<" "<<epath.top()<<" ";
```

```
                epath.pop();
            }

        }

};
int main()
{
    int v=0;
    cout << "Enter number of vertices: ";
    cin >> v;
    Graph G(v);
    // G.addEdge(1, 6);
    // G.addEdge(6, 3);
    // G.addEdge(3, 2);
    // G.addEdge(2, 1);
    // G.addEdge(2, 5);
    // G.addEdge(5, 4);
    // G.addEdge(4, 2);
    int i = 0;
    int j =0;
    cout << "Input edges, input -1 to either to finish input
(only input in range" << endl;
    while(1){
      cout << "Input from vertex: ";
      cin >> i;
      cout << "Input to vertext: ";
      cin >> j;
      if(i == -1 || j==-1){
        break;
      }
      cout << "Added edge from " << i << " to " << j << endl;
      G.addEdge(i,j);
    }
    G.printEulerPathCircuit();
}
```

Output:

```
Enter number of vertices: 6
Input edges, input -1 to either to finish input
Input from vertex: 1
Input to vertext: 6
Added edge from 1 to 6
Input from vertex: 6
Input to vertext: 3
Added edge from 6 to 3
Input from vertex: 3
Input to vertext: 2
Added edge from 3 to 2
Input from vertex: 2
Input to vertext: 1
Added edge from 2 to 1
Input from vertex: 2
Input to vertext: 5
Input from vertex: 5
Input to vertext: 4
Added edge from 5 to 4
Input from vertex: 4
Input to vertext: 2
Added edge from 4 to 2
Input from vertex: -1
Input to vertext: 0
Euler Circuit:  1  2  4  5  2  3  6  1
```