1.

```
#include<bits/stdc++.h>
using namespace std;
template< class T>
class node{
private:
    T data;
    node<T>* next;
/*     node(int _data):data(_data),next(0){}
    node():data(0),next(0){}  */
    template <class TT> friend class Queue;
};
template < class T>
class Queue{
public:
    Queue():front(NULL),rear(NULL),size(0){};
    void enqueue(T p){
    node<T>* ptr = new node<T>[1];
    ptr->data = p;
    if(IsEmpty()){
        front = ptr;
        rear = ptr;
    }else{
        rear->next = ptr;
        rear = rear->next;
        size++;
    }
}
    void dequeue(){
    if(IsEmpty()){
        cout << "queue is empty";
    }else{
        node<T> *deleteNode = front;
        front = front->next;
        delete deleteNode;
        deleteNode = NULL;
        size--;
    }
```

```
}
    bool IsEmpty(){
    if(front == NULL && rear == NULL){
        return true;
    }else return false;
}
    void top(){
    if(IsEmpty()){
        cout << "queue is empty";
    }else{
        cout << front->data;
    }
}
private:
    node<T>* front;
    node<T>* rear;
    int size;
};
int main(){
    Queue<int> q;
    q.enqueue(5);
    q.enqueue(2);
    q.enqueue(13);
    q.top();
    q.dequeue();
    cout << endl;
    q.top();
}
```

2.

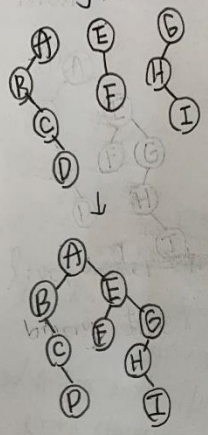
```
#include<bits/stdc++.h>
using namespace std;
struct node{
     int data;
     struct node* left;
     struct node* right;
};
struct node* newNode(int data){
```

```
    node* node = new struct node();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return(node);
}
void printInorder(node* node){
    if (node == NULL) return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}
void swapTree(node* node){
    if(node==NULL) return;
    struct node* temp=node->left;
    node->left=node->right;
    node->right=temp;
    swapTree(node->left);
    swapTree(node->right);
}
int main(){
    struct node *root  = newNode(1);
    root->left         = newNode(2);
    root->right        = newNode(3);
    root->left->left   = newNode(4);
    root->left->right  = newNode(5);
    cout << endl;
    cout << "Inorder of the binary tree "<< endl;
    printInorder(root);
    swapTree(root);
    cout << endl;
    cout << "After swap of the tree " << endl;
    printInorder(root);
    return 0;
}
```

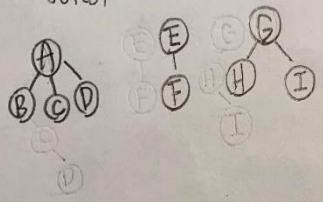
3.

3,

binery tree



forest





level order traversal

binery tree

ABECFGDHI

forest

AEGBCDFHI

From the above examaple we could know that the level-order traversal of forest and that of its corresponding binary tree do not necessarily yield the same result.

4.

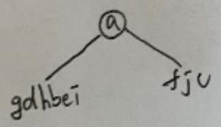
4.

Ex

inorder = gdhbeiafjc
preorder = abdgheicfj

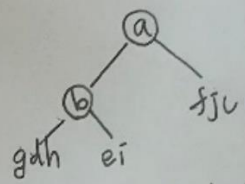
Scan the preorder left to right using the inorder to separate left and right subtrees.

a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



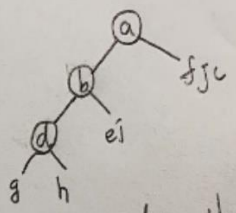

preorder = abdgheicfj

b is the next root; gdh are in the left subtree; ei are in the right subtree




preorder = abdgheicfj

d is the next root; g is in the left subtree; h is in the right subtree



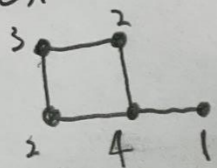

by keep doing the procedure

⋮

we could construct a unique binery tree.

5.

Ex




Sum of degree : $1+4+2+3+2=12$

$|E|=6$

⟨Pf⟩ In any graph $G$, $\sum_{v\in V(G)} \deg(v) = 2|E(G)|$

Let $e \in E(G)$ be any edge

Then $e$ has 2 ends, say $u$ and $w$ (so $e=uw$)

when we sum the degrees of the vertices,

edge $e$ gets counted twice ( once in $\deg(u)$ term, once in $\deg(w)$ term )

Similary, every edge gets counted twice

#

6.

6,

1 vertices — 0 edge

2 vertices — 1 edge

3 vertices — 3 edge

4 vertices — 6 edge

5 vertices — 10 edge

⟨Pf⟩ complete graph means that every vertex is connected with every other vertex.

If taking one vertex of the graph, we will get $(n-1)$ outgoing edges from that vertex.

now having $n$ vertices in total, there might be $n(n-1)$ edges in total, but we counts every edge twice, since every edge going out from one vertex is an edge going into another vertex. Hence dividing the result by 2.

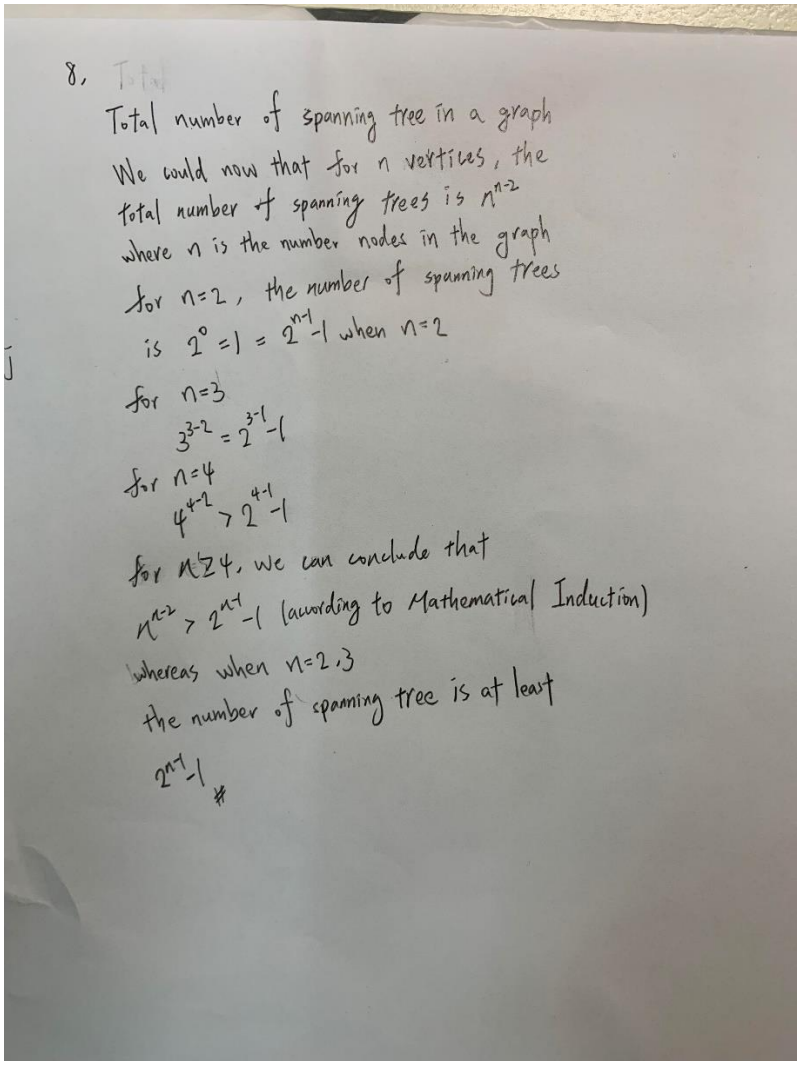
∴ we would get $\frac{n(n-1)}{2}$ #

7.

```
#include<iostream>
#include<list>
using namespace std;

class graph{
private:
     int V;
     list<int> *adj;
public:
     graph(int V);
     void insertEdge(int v,int x);
     void BFS(int s);
};

graph::graph(int V){
     this->V = V;
     adj = new list<int>[V];
}

void graph::insertEdge(int v,int x){
     adj[v].push_back(x);
}

void graph::BFS(int s){
     bool *visit = new bool[V];
     for(int i=0;i<V;i++){
          visit[i] = false;
     }
     list<int> queue;
     visit[s] = true;
     queue.push_back(s);
     list<int>::iterator i;
     while(!queue.empty()){
          s = queue.front();
          cout << s << " ";
          queue.pop_front();
```

```
          for(i = adj[s].begin();i!=adj[s].end();++i){
               if(!visit[*i]){
                    visit[*i] = true;
                    queue.push_back(*i);
               }
          }
     }
}
int main(){
     graph g(6);
     g.insertEdge(0,1);
     g.insertEdge(0,2);
     g.insertEdge(1,2);
     g.insertEdge(1,3);
     g.insertEdge(2,3);
     g.insertEdge(3,4);
     g.insertEdge(4,0);
     g.insertEdge(4,1);
     g.insertEdge(4,5);

     g.BFS(1);
}
```

8.



8.

Total number of spanning tree in a graph

We could now that for $n$ vertices, the total number of spanning trees is $n^{n-2}$ where $n$ is the number nodes in the graph

for $n=2$, the number of spanning trees is $2^0 = 1 = 2^{n-1} - 1$ when $n=2$

for $n=3$

$3^{3-2} = 2^{3-1} - 1$

for $n=4$

$4^{4-2} > 2^{4-1} - 1$

for $n \geq 4$, we can conclude that

$n^{n-2} > 2^{n-1} - 1$ (according to Mathematical Induction)

whereas when $n=2,3$

the number of spanning tree is at least

$2^{n-1} - 1$ #

9.

```cpp
#include<bits/stdc++.h>
using namespace std;

int globalTime = 0;
struct node{
    int num;
    vector<int> neighbor;
    int color;
    int fininshTime;
};

class graph{
public:
    graph(int n){
        G.resize(n);
```

```
        for(int i=0;i<G.size();i++){
            G[i].num = i;
        }
    }
    vector<node> G;
    void DFS(int now);
    void sortNum();
};

void graph::DFS(int now){
    G[now].color = 1;
    for(int nb: G[now].neighbor){
        if(G[nb].color == 0) DFS(nb);
    }
    G[now].fininshTime = globalTime;
    globalTime++;
    return;    //上一層
}

bool compare(node a, node b){
    return a.fininshTime > b.fininshTime;
}
void graph::sortNum(){
    sort(G.begin(),G.end(),compare);
}

int main(){
    graph T(6);
    int e = 8;
    for(int i=0;i<e;i++){
        int s,g;
        cin >> s >> g;
        T.G[s].neighbor.push_back(g);
    }
    for(int i=0;i<6;i++){
        T.G[i].color = 0;
    }
    for(int i=0;i<T.G.size();i++){
```

```
        if(T.G[i].color==0) T.DFS(i);
    }
    T.sortNum();
    for(int i=0;i<T.G.size();i++){
        cout << T.G[i].num << " ";
    }
}
```

10.