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;

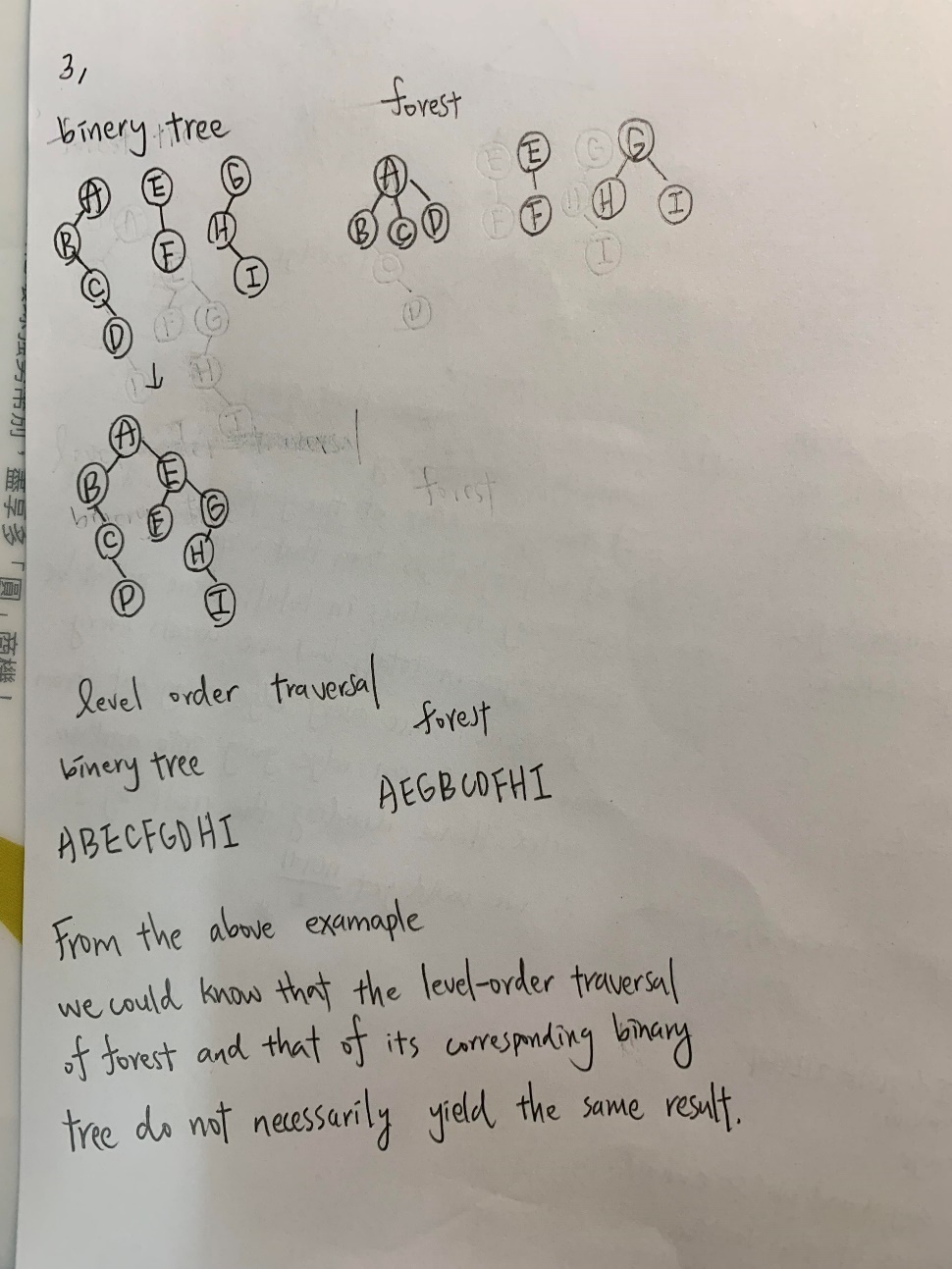
     cout << "After swap of the tree " << endl;

     printInorder(root);

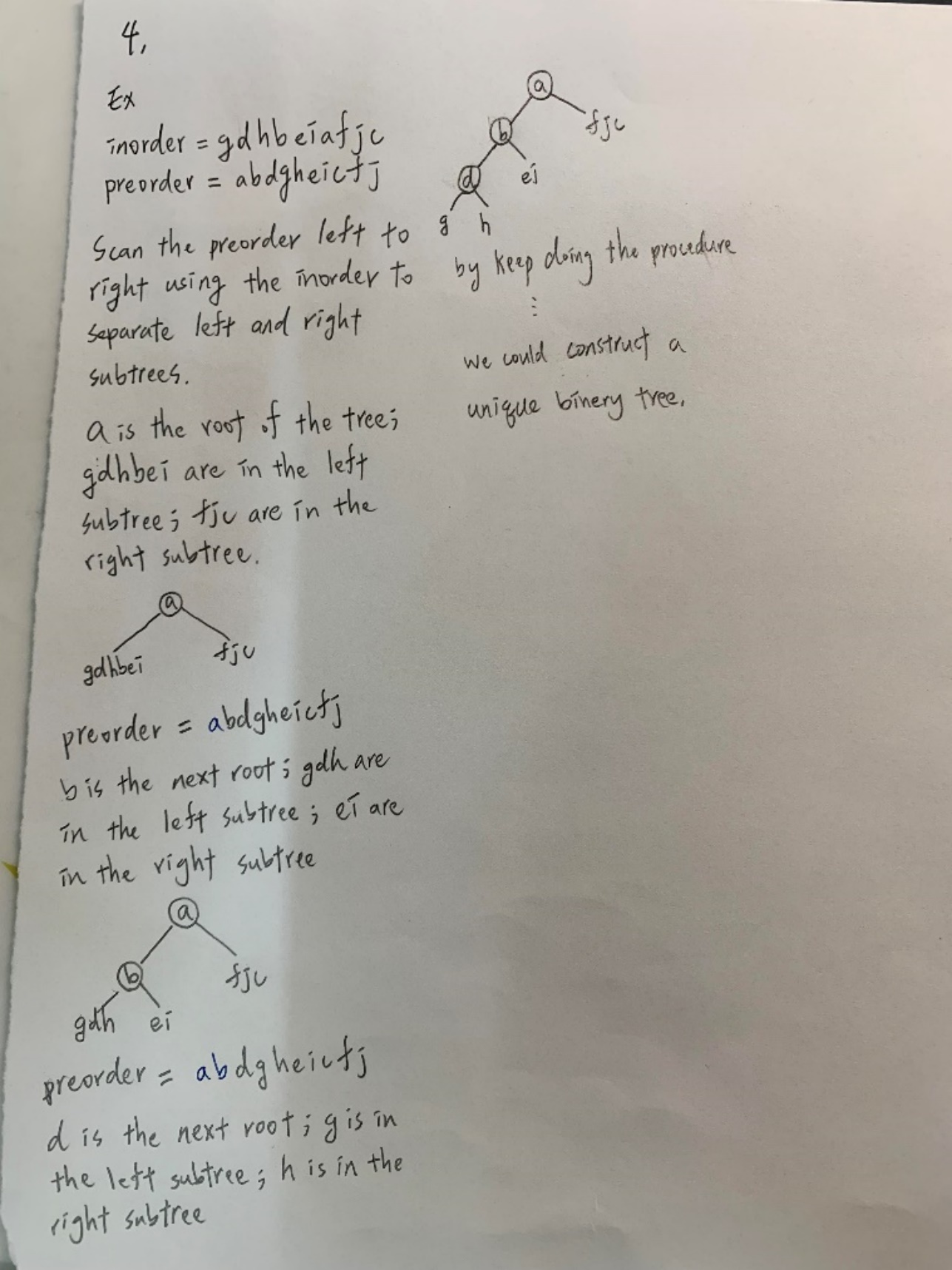
     return 0;

}

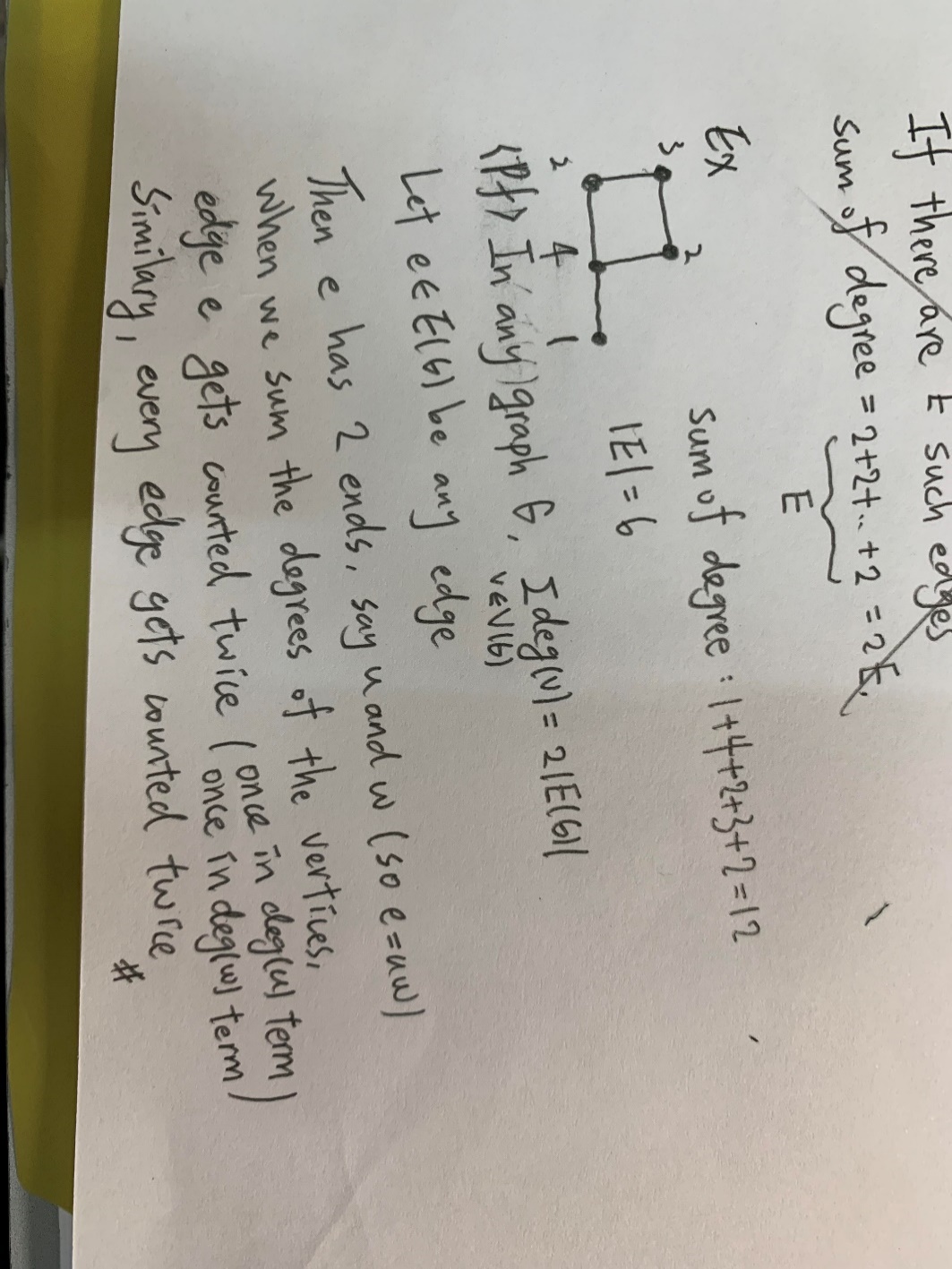
3.

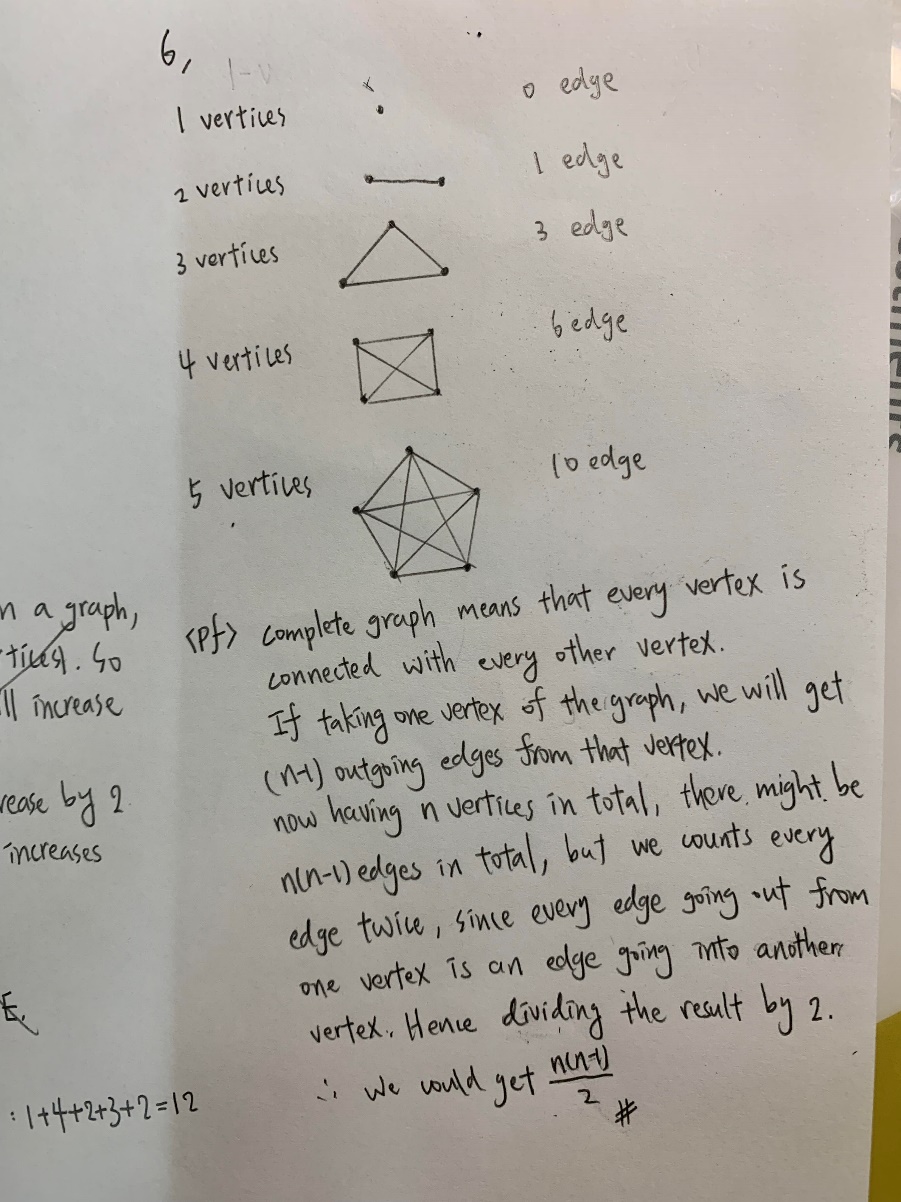


4.



5.

6.



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);

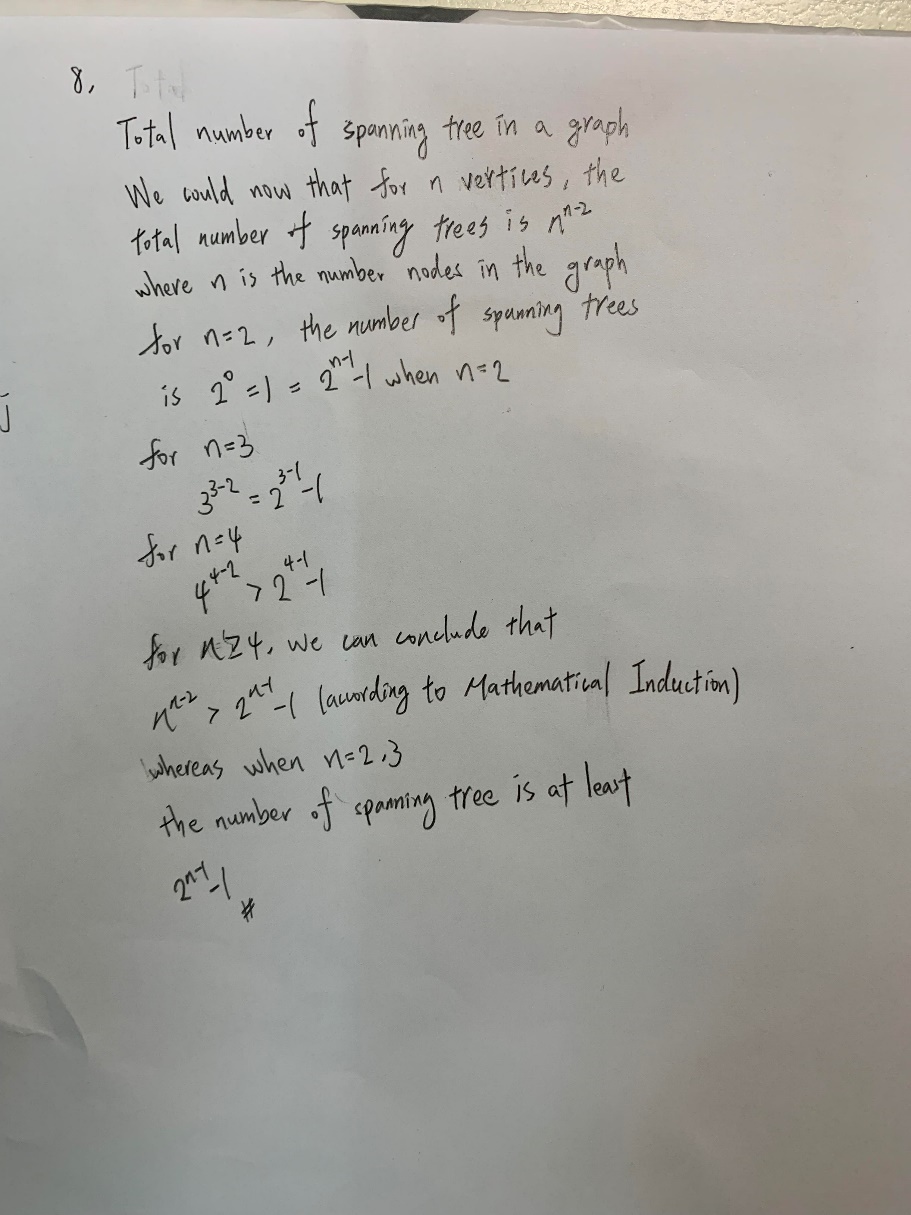
     g.insertEdge(4,1);

     g.insertEdge(4,5);

     g.BFS(1);

}

8.



9.

#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.

