图与树

图模板

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

#define MAX(a, b) ((a) > (b) ? (a) : (b) )

//定义图的定点
typedef struct Vertex {
    int id;
    vector<int> connectors;     //存储节点的后续连接顶点编号
    Vertex() : id(-1) {}
    Vertex(int nid) : id(nid) {}
} Vertex;

//定义Graph的邻接表表示
typedef struct Graph {
    vector<Vertex> vertexs;    //存储定点信息
    int nVertexs;              //计数：邻接数
    bool isDAG;                //标志：是有向图吗

    Graph(int n, bool isDAG) : nVertexs(n), isDAG(isDAG) {
vertexs.resize(n); }
```

```
    //向图中添加边
    bool addEdge(int id1, int id2) {
        if (!(MAX(id1, id2) < vertexs.size())) return false;

        if (isDAG) {
            vertexs[id1].connectors.push_back(id2);
        }
        else {
            vertexs[id1].connectors.push_back(id2);
            vertexs[id2].connectors.push_back(id1);
        }
        return true;
    }

    //广度优先搜索
    vector<int> BFS(int start) {
        set<int> visited;
        vector<int> g, rst;
        g.push_back(start);
        visited.insert(start);
        while(g.size() > 0) {
            int id = g[0];
            g.erase(g.begin());
            rst.push_back(id);
            for(int i = 0; i < vertexs[id].connectors.size(); i++)
{
                int id1 = vertexs[id].connectors[i];
                if (visited.count(id1) == 0) {
                    g.push_back(id1);
                    visited.insert(id1);
                }
            }
        }
        return rst;
    }

    //深度优先搜索
    vector<int> DFS(int start) {
        set<int> visited;
        vector<int> g, rst;
        g.push_back(start);
        //cout << "push " << start << " ";
```

```
        visited.insert(start);
        rst.push_back(start);
        bool found;
        while(g.size() > 0) {
            int id = g[g.size()-1];
            found = false;
            for(int i = 0; i < vertexs[id].connectors.size(); i++)
{
                int id1 = vertexs[id].connectors[i];
                if (visited.count(id1) == 0) {
                    g.push_back(id1);
                    rst.push_back(id1);
                    visited.insert(id1);
                    //cout << "push " << id1 << " ";
                    found = true;
                    break;
                }
            }
            if (!found) {
                int id2 = g[g.size()-1];
                rst.push_back(-1 * id2);
                //cout << "pop " << id2 << " ";
                g.pop_back();
            }
        }
        //cout << endl;
        return rst;
    }
} Graph;

int main() {
    Graph g(8, false);
    g.addEdge(0, 1);
    g.addEdge(0, 3);
    g.addEdge(1, 2);
    g.addEdge(3, 4);
    g.addEdge(3, 5);
    g.addEdge(4, 5);
    g.addEdge(4, 6);
    g.addEdge(5, 6);
    g.addEdge(5, 7);
    g.addEdge(6, 7);
```

```
    vector<int> bv = g.BFS(0);
    cout << "宽度优先搜索节点顺序: ";
    for(int j = 0; j < bv.size(); j++)
        cout << bv[j] << " ";
    cout << endl;

    cout << "深度优先搜索节点顺序: ";
    Graph g1(6, false);
    g1.addEdge(0, 1);
    g1.addEdge(0, 4);
    g1.addEdge(0, 5);
    g1.addEdge(1, 5);
    g1.addEdge(4, 5);
    g1.addEdge(5, 2);
    g1.addEdge(5, 3);
    g1.addEdge(2, 3);
    vector<int> route = g1.DFS(0);
    for(int i = 0; i < route.size(); i++)
        cout << route[i] << " ";
    cout << endl;

    char ch;
    cin >> ch;
    return 0;
}
```

2019-1

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <queue>
#define MAX(a, b) ((a) > (b) ? (a) : (b) )
using namespace std;
int n,m;
vector<int> inDegreelist,outDegreelist;

//定义图的定点
```

```
typedef struct Vertex {
    int id,inDegree,outDegree;
    vector<int> connectors;     //存储节点的后续连接顶点编号
    Vertex() : id(-1),inDegree(0),outDegree(0) {}
    Vertex(int nid) : id(nid),inDegree(0),outDegree(0) {}
} Vertex;

//定义Graph的邻接表表示
typedef struct Graph {
    vector<Vertex> vertexs;    //存储定点信息
    int nVertexs;              //计数：邻接数
    bool isDAG;                //标志：是有向图吗

    Graph(int n, bool isDAG) : nVertexs(n), isDAG(isDAG) {
vertexs.resize(n); }
    Graph() : nVertexs(1), isDAG(1) { vertexs.resize(1); }
    //向图中添加边
    bool addEdge(int id1, int id2) {
        if (!(MAX(id1, id2) < vertexs.size())) return false;

        if (isDAG) {
            vertexs[id1].connectors.push_back(id2);
            vertexs[id1].outDegree++;
            vertexs[id2].inDegree++;
        }
        else {
            vertexs[id1].connectors.push_back(id2);
            vertexs[id2].connectors.push_back(id1);

            vertexs[id1].outDegree++;
            vertexs[id1].inDegree++;

            vertexs[id2].outDegree++;
            vertexs[id2].inDegree++;

        }
        return true;
    }
} Graph;

Graph g;
```

```
void init(){
    cin>>n>>m;
    g=Graph(n, true);
    int src,dst;
    while(m--){
        cin>>src>>dst;
        g.addEdge(src,dst);
    }
    vector<Vertex>::iterator it = g.vertexs.begin();
    while(it!=g.vertexs.end()){
        inDegreelist.push_back(it->inDegree);
        outDegreelist.push_back(it->outDegree);
        it++;
    }
}
int countin(int n){
    return count(inDegreelist.begin(),inDegreelist.end(),n);
}
int countout(int n){
    return count(outDegreelist.begin(),outDegreelist.end(),n);
}

bool Is_List(){
    //有一个inDegree为0的头和一个outDegree为0的尾，且其余节点入度与出度都为
1;
    return (countin(0)==1)&&(countout(0)==1)&&(countin(1)==n-1)&&
(countout(1)==n-1);
}

bool Is_Tree(){
    //有一个inDegree为0的头且其余节点inDegree均为1，且不是链表;
    return (countin(0)==1)&&(countin(1)==n-1);
}

bool topologicalSort(){//拓扑排序判断有环无环
    int num=0;//记录加入拓扑排序的顶点数
    queue<int> q;
    for(int i=0;i<n;i++){
        if(inDegreelist[i]==0){
            q.push(i);//将所有入度为0的顶点入队
        }
    }
```

```
    while(!q.empty()){
        int u=q.front();//取队首顶点u
        q.pop();
        for(int i=0;i<g.vertexs[u].connectors.size();i++){
            int v=g.vertexs[u].connectors[i];//u的后继节点v
            inDegreelist[v]--;//v的入度减1
            if(inDegreelist[v]==0){//顶点v的入度减为0则入队
                q.push(v);
            }
        }
        g.vertexs[u].connectors.clear();//清空u的所有出边
        num++;//加入拓扑排序的顶点数加1
    }
    if(num==n) return true;//加入拓扑排序的顶点为n，则拓扑排序成功，图无环
    else return false;//否则拓扑排序失败，图有环
}


int main(){
    init();
    if(n==0||m==0){
        cout<<"error"<<endl;
    }
    if(Is_List()){
        cout<<"list"<<endl;
    }

    else if(Is_Tree()){
        cout<<"tree"<<endl;
    }
    else if(topologicalSort()){
        cout<<"no ring"<<endl;
    }
    else{
    cout<<"have ring"<<endl;
    }
    return 0;
}
```

树模板

注释版

```
#include<bits/stdc++.h>
#include<cmath>

#define mem(a,b) memset(a,b,sizeof a);

using namespace std;

typedef long long ll;

const int maxn=50;
int mid[maxn],po[maxn],pr[maxn];
int first;

struct node
{
    int l,r;
}T[maxn];

// 中序+先序=>二叉树
int mid_pr_build(int la,int ra,int lb,int rb) // la,ra：表示中序遍历 lb,rb：表示先序遍历
{
    // 这里不能等于，因为假设：len==1，则la==ra，直接返回，但是实际上是有一个 rt 的，却没被建立
    if(la>ra) return 0;
    int rt=pr[lb]; // 因为先序遍历第一个是根节点
    int p1=la,p2;

    while(mid[p1]!=rt) p1++; // 在中序遍历中找到根节点
    p2=p1-la;
    T[rt].l=mid_pr_build(la,p1-1,lb+1,lb+p2); // 左子树（锁定左子树范围的下标）
    T[rt].r=mid_pr_build(p1+1,ra,lb+p2+1,rb); // 右子树（锁定右子树范围的下标）

    return rt;
}

// 中序+后序=>二叉树
```

```
int mid_po_build(int la,int ra,int lb,int rb) // la,ra：表示中序遍历 lb,rb：表示后序遍历
{
    if(la>ra) return 0;
    int rt=po[rb]; // 因为后序遍历最后一个是根节点
    int p1=la,p2;

    while(mid[p1]!=rt) p1++; // 在中序遍历中找到根节点
    p2=p1-la;
    T[rt].l=mid_po_build(la,p1-1,lb,lb+p2-1); // 左子树（锁定左子树范围的下标）
    T[rt].r=mid_po_build(p1+1,ra,lb+p2,rb-1); // 右子树（锁定右子树范围的下标）

    return rt;
}

// 求树高
int getHeight(int rt)
{
    if(rt==0) return 0;
    return 1+max(getHeight(T[rt].l),getHeight(T[rt].r));
}

// 层序遍历
void bfs(int rt)
{
    queue<int> q;
    vector<int> v;
    q.push(rt);

    while(!q.empty())
    {
        int w=q.front();
        q.pop();
        v.push_back(w);
        if(T[w].l!=0) q.push(T[w].l);
        if(T[w].r!=0) q.push(T[w].r);
    }

    int len=v.size();
    for(int i=0;i<len;i++)
```

```
        printf("%d%c",v[i],i==len-1?'\n':' '); // 推荐这种写法，简洁
}

// 先序遍历
void preT(int rt)
{
    if(rt==0) return;
    printf(first?first=0,"%d":" %d",rt);
    preT(T[rt].l);
    preT(T[rt].r);
}

// 中序遍历
void midT(int rt)
{
    if(rt==0) return;
    midT(T[rt].l);
    printf(first?first=0,"%d":" %d",rt);
    midT(T[rt].r);
}

// 后序遍历
void postT(int rt)
{
    if(rt==0) return;
    postT(T[rt].l);
    postT(T[rt].r);
    printf(first?first=0,"%d":" %d",rt);
}

int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        first=1;
        for(int i=0;i<n;i++) scanf("%d",&po[i]); // 后序结点
//        for(int i=0;i<n;i++) scanf("%d",&pr[i]); // 先序结点
        for(int i=0;i<n;i++) scanf("%d",&mid[i]); // 中序结点

        int rt=mid_po_build(0,n-1,0,n-1); // 中+后，返回根节点
//        int rt=mid_pr_build(0,n-1,0,n-1); // 中+先，返回根节点
```

```
        bfs(rt); // 层序遍历
//        preT(rt); // 先序遍历
//        puts("");
//        postT(rt); // 后序遍历
//        puts("");
//        midT(rt); // 中序遍历
//        puts("");
    }

    return 0;
}
```

简化版（Val As Index，若数据不在1~N内，则可能越界）

```
#include<bits/stdc++.h>
#include<cmath>

#define mem(a,b) memset(a,b,sizeof a);

using namespace std;

typedef long long ll;

const int maxn=50;
int mid[maxn],po[maxn],pr[maxn];
int first;

struct node
{
    int l,r;
}T[maxn];

int mid_pr_build(int la,int ra,int lb,int rb)
{
    if(la>ra) return 0;
    int rt=pr[lb];
    int p1=la,p2;

    while(mid[p1]!=rt) p1++;
```

```
    p2=p1-la;
    T[rt].l=mid_pr_build(la,p1-1,lb+1,lb+p2);
    T[rt].r=mid_pr_build(p1+1,ra,lb+p2+1,rb);

    return rt;
}

int mid_po_build(int la,int ra,int lb,int rb)
{
    if(la>ra) return 0;
    int rt=po[rb];
    int p1=la,p2;

    while(mid[p1]!=rt) p1++;
    p2=p1-la;
    T[rt].l=mid_po_build(la,p1-1,lb,lb+p2-1);
    T[rt].r=mid_po_build(p1+1,ra,lb+p2,rb-1);

    return rt;
}

int getHeight(int rt)
{
    if(rt==0) return 0;
    return 1+max(getHeight(T[rt].l),getHeight(T[rt].r));
}

void bfs(int rt)
{
    queue<int> q;
    vector<int> v;
    q.push(rt);

    while(!q.empty())
    {
        int w=q.front();
        q.pop();
        v.push_back(w);
        if(T[w].l!=0) q.push(T[w].l);
        if(T[w].r!=0) q.push(T[w].r);
    }
```

```
    int len=v.size();
    for(int i=0;i<len;i++)
        printf("%d%c",v[i],i==len-1?'\n':' ');
}

void preT(int rt)
{
    if(rt==0) return;
    printf(first?first=0,"%d":" %d",rt);
    preT(T[rt].l);
    preT(T[rt].r);
}

void midT(int rt)
{
    if(rt==0) return;
    midT(T[rt].l);
    printf(first?first=0,"%d":" %d",rt);
    midT(T[rt].r);
}

void postT(int rt)
{
    if(rt==0) return;
    postT(T[rt].l);
    postT(T[rt].r);
    printf(first?first=0,"%d":" %d",rt);
}

int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        first=1;
        for(int i=0;i<n;i++) scanf("%d",&po[i]);
//        for(int i=0;i<n;i++) scanf("%d",&pr[i]);
        for(int i=0;i<n;i++) scanf("%d",&mid[i]);

        int rt=mid_po_build(0,n-1,0,n-1);
//        int rt=mid_pr_build(0,n-1,0,n-1);
```

```
        bfs(rt);
//        preT(rt);
//        postT(rt);
//        midT(rt);
    }

    return 0;
}
```

简化版（Val Not As Index，可以存任意的 Val）

```
#include<bits/stdc++.h>
#include<cmath>

#define mem(a,b) memset(a,b,sizeof a)
#define ssclr(ss) ss.clear(), ss.str("")
#define INF 0x3f3f3f3f
#define MOD 1000000007

using namespace std;

typedef long long ll;

const int maxn=5e4+1000;

int f;
int pre[maxn], in[maxn];

struct node
{
    int l,r,d;
}T[maxn];

int create(int l1,int r1,int l2,int r2) // in pre
{
    if(l2>r2) return -1;
    int rt=l2;
    int p1=l1,p2;

    while(in[p1]!=pre[rt]) p1++;
    p2=p1-l1;
```

```
    T[rt].d=pre[rt];
    T[rt].l=create(l1,p1-1,l2+1,l2+p2);
    T[rt].r=create(p1+1,r1,l2+p2+1,r2);

    return rt;
}

void postT(int rt)
{
    if(rt==-1 || !f) return;
    postT(T[rt].l);
    postT(T[rt].r);
    if(f) f=0, printf("%d\n",T[rt].d);
}

int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++) scanf("%d",&pre[i]);
    for(int i=0;i<n;i++) scanf("%d",&in[i]);
    int rt=create(0,n-1,0,n-1);
    f=1, postT(rt);

    return 0;
}
```