Project4_图的遍历_实验报告

一. 实验目的

(1) 以邻接多重表为存储结构，实现联通无向图的深度优先和广度优先遍历。以指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

(2) 借助于栈类型（自行定义和实现），用非递归算法实现深度优先遍历。

(3) 以邻接表为存储结构，建立深度优先生成树和广度优先生成树，并以树形输出生成树。

二.实验环境

编程语言和开发工具

编程语言：C++

开发工具：Visual Studio Code\QT

三.分析与设计

3.1 需求分析:对输入的图进行深度优先遍历和广度优先遍历。

3.2 设计思路及细节:

1. 大概：两种遍历都有一个 visit 数组来判断该节点是否遍历过。

   深度：对图的深度优先遍历，通过自己设置的栈先将初始节点存入，再通过邻接表判断该节点的邻接结点，将其压入栈中，并通过循环查找邻接结点的邻接结点，也存入栈中，若已经没有邻接结点则出栈。

   广度：对图的广度优先遍历，用队列实现，将初始节点的邻接点全部押入队列，在压出本体。
2. 优点：有良好 ui 界面。

四.代码

1.图以及树结点的结构

```
vector<int> step(1000,0);
struct  Graph{
    vector<vector<int>> adj; //图的邻接表
    vector<string> name;     //顶点名称
    vector<int> val;     // 权
    int vertex;// 顶点数
    int edge;//边数
    Graph(int n=0, int m=0):vertex(n), edge(m){
        adj.resize(n);
    }
};
struct node{
    int index;
    int value;
    string name;
    vector<node*> next;
    node(){
        index=-1;
        value=0;
    }
};
```

2.自行定义的栈

```
template< class T >
class Stack{
    public:
        Stack():take(new T[capacity] ),size(0),capacity(1){};//构造函数
        Stack(const Stack&);//拷贝构造函数
        ~Stack();//析构函数
        Stack& operator=(const Stack&) ;//重载赋值运算符
        int Size() const;//返回栈中元素数目
        int Capacity() const;//返回当前栈的容量
        bool IsEmpty() const;//判断栈是否为空栈
        T& Top() const;//返回栈顶元素
        void Push(const T&);//将元素压入栈中，当元素数目超过栈的容量时重建栈
        void Pop();//弹出栈顶元素
    protected:
        int size;//栈中元素数目
        int capacity;//栈的容量
        void Expand();//扩充栈
        void Decrease();//压缩栈
    private:
        T* take;//保存栈中元素的数组
} ;
```

```
template< class T >
Stack<T>::Stack(const Stack<T> &s):take(new T[capacity]),size(s.size),capacity(s.capacity){
     for(int i=0;i<capacity;++i){
        take[i]=s.take[i];
    }
}

template< class T >
Stack< T >::~Stack(){
    delete [] take;
}

template< class T >
Stack< T >& Stack< T >::operator = (const Stack& s){
    take=new T[s.capacity];
    size=s.size;
    capacity=s.capacity;
    for(int i=0;i<capacity;++i){
        take[i]=s.take[i];
    }
}

template< class T >
int Stack< T >::Size() const{
    return size;
}

template< class T >
int Stack< T >::Capacity() const{
    return capacity;
}
```

```
bool Stack< T >::IsEmpty() const{
    if(size==0){
        return 1;
    }else{
        return 0;
    }
}

template< class T >
T& Stack< T >::Top() const{
    return take[size - 1];
}

template< class T >
void Stack< T >::Pop(){
    if(size == capacity/2)
        Decrease( );
        --size;
}

template< class T >
void Stack< T >::Push( const T& obj ){
    if( size == capacity )
        Expand();
        take[size++]=obj;
}
```

```
template< class T >
void Stack< T >::Expand(){
    capacity=4*capacity;
    T* temp= new T [capacity];
    for(int i = 0 ;i<size;i++){
        temp[i]=take[i];
    }
    delete [] take;
    take = temp;

}

template< class T >
void Stack< T >::Decrease(){
    capacity=capacity/4;
    T* temp= new T[ capacity ] ;
    for(int i = 0 ;i<size;++i){
        temp[i]=take[i];
    }
    delete [] take;
    take = temp;
}
```

3. 深度优先遍历和广度优先

```
void dfs(Graph &p,int put){ //深度优先遍历
    Stack<int> temp;
    temp.Push(put);
    cout<<temp.Top();
    cout<<p.name[temp.Top()]<<" ";
    step[put]=1;
    while(!temp.IsEmpty()){
        int s=temp.Top();
        int sizeofs=p.adj[s].size();
        int i;
        for(i=0;i<sizeofs;i++){
            int t=p.adj[s][i];
            if(step[t]!=1){
                temp.Push(t);
                cout<<t;
                cout<<p.name[t]<<" ";
                step[t]=1;
                dftree[s]->next.push_back(dftree[t]);
                break;
            }
        }
        if(i==p.adj[s].size()){     ▲ comparison of integers of
            temp.Pop();
        }
    }
    for(int j=0;j<=step.size();j++){ //最后将记录是否走过的数组置零
        step[j]=0;
    }
    cout<<endl;
    node *scantree=new node;     //建树并存入树的数组      ▲ Value s
    scantree=dftree[put];
    dotree(scantree);     ▲ Potential leak of memory pointed to
```

```
void bfs(Graph &p,int put){ //广度优先遍历
    queue<int> temp;
    temp.push(put);
    step[put]=1;
    while(!temp.empty()){
        int t=temp.front();
        cout<<t;
        cout<<p.name[t]<<" ";
        temp.pop();
        for(int i=0;i<=p.adj[t].size();i++){     ▲ comparison of
            if(step[p.adj[t][i]]!=1){
                step[p.adj[t][i]]=1;
                temp.push(p.adj[t][i]);
                bftree[t]->next.push_back(bftree[p.adj[t][i]]);
            }
        }
    }
    for(int i=0;i<=step.size();i++){ //最后将记录是否走过的数组置零
        step[i]=0;
    }
    cout<<endl;
    node *scantree=new node;     //建树并存入树的数组      ▲ Value st
    scantree=bftree[put];
    botree(scantree);     ▲ Potential leak of memory pointed to
}
```

4.建图函数和建树函数

```
void botree(node *&p){
    vector<string> temp;
    temp.push_back(p->name);
    for(int i=0;i<p->next.size();i++){
        temp.push_back(p->next[i]->name);
    }
    bop.push_back(temp);
    for(int i=0;i<p->next.size();i++){
        botree(p->next[i]);
    }
}
```

```
Graph makeGraph(int n,int s){  //建图的函
    Graph temp(n,s);
    int count=s,na=n;
    for(int i=0;i<na;i++){
        string p;
        cin>>p;
        temp.name.push_back(p);
        bftree[i]=new node;
        bftree[i]->index=i;
        bftree[i]->name=p;
        dftree[i]=new node;
        dftree[i]->index=i;
        dftree[i]->name=p;
    }
    while(count--){
        int fa,son;
        cin>>fa>>son;
        temp.adj[fa].push_back(son);
        temp.adj[son].push_back(fa);
    }
    return temp;
};
```

5.达成树形输出的函数

获取父节点

```
QStandardItem *Widget::getItem(QStandardItem *item, QString s)
{
    if(item == NULL)
        return NULL;
//    qDebug() << tr("fine %1").arg(item->text());  //检测是否找到
    QStandardItem *getitem = NULL;
    if(item->text().compare(s) == 0)
        return item;
    if(!item->hasChildren())//判断是否有孩子,没有则返回0
        return NULL;
    for(int i = 0;i < item->rowCount() && getitem == NULL;i++)//遍历item下所有子条目，若果getitme有获得对象，则退出循
    {
        QStandardItem * childitem = item->child(i);
        getitem = getItem(childitem,s);//寻找这个子条目的所有子条目是否存在文本为s的条目。
    }
    return getitem;
}
```

开始建树

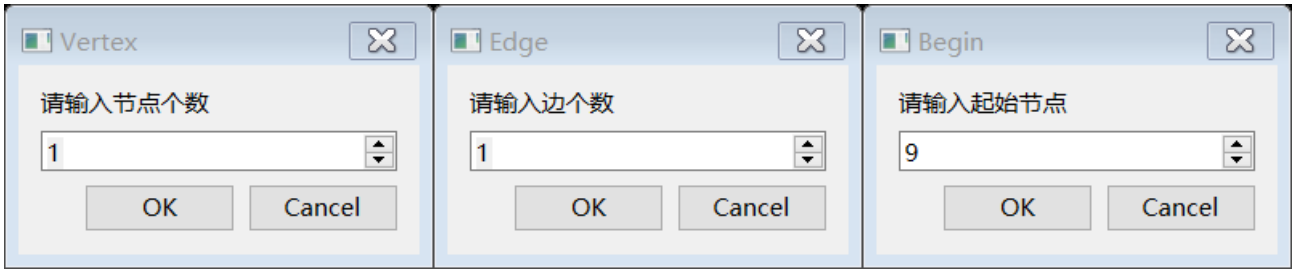
```
void Widget::remkt(vector<vector<QString>> &p){
    QStandardItemModel *modelt = static_cast<QStandardItemModel*>(ui->BFS->model());//创建模型指定父类
    ui->BFS->setModel(modelt);
    QString temp=p[0][0];
    QStandardItem* itemProject = new QStandardItem(temp);
    modelt->appendRow(itemProject);
    for(int l=1;l<p[0].size();l++){     ⚠ comparison of integers of different signs: 'int' and 'std::
        QString ty=p[0][l];
        QStandardItem* ip = new QStandardItem(ty);
        itemProject->appendRow(ip);
    }
    for(int i=1;i<p.size();i++){     ⚠ comparison of integers of different signs: 'int' and 'std::vec
//       QList<QStandardItem*> list= modelt->findItems(p[i][0]);
       QStandardItem * getitem = getItem(itemProject,p[i][0]);
//       QStandardItem* getre=list.at(0);
        for(int j=1;j<p[i].size();j++){     ⚠ comparison of integers of different signs: 'int' and 's
           QString tst=p[i][j];
          QStandardItem* it = new QStandardItem(tst);
           getitem->appendRow(it);
//           getitem->parent()->setChild(getitem->row(),0,new QStandardItem(tst));
        }
    }
}
```

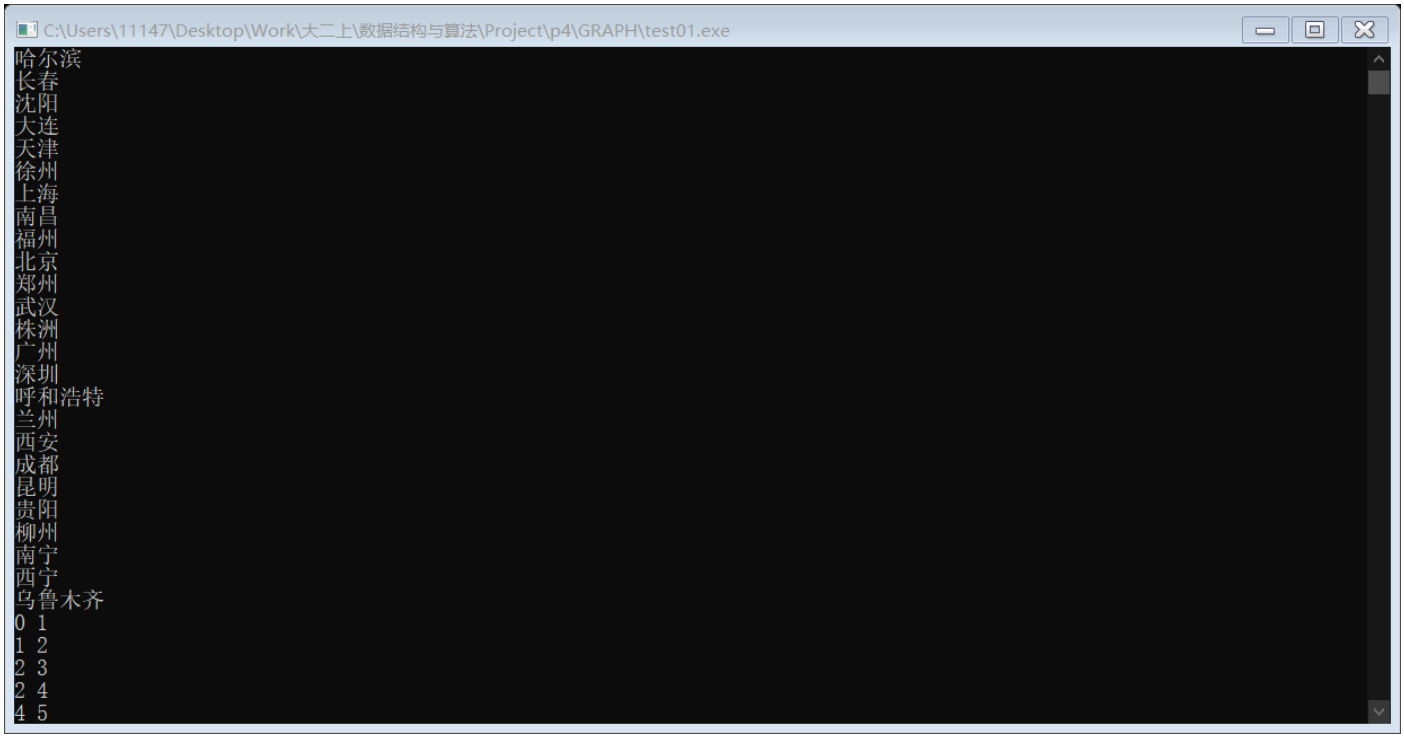
6.其他控件

```
        QImage *img=new QImage;
        img->load("pan.png");
        ui->label->setPixmap(QPixmap::fromImage(*img));

        const QString fileName = "jfu.png";
        QPalette pal = ui->BFS->palette();
        pal.setBrush(QPalette::Base, QPixmap(fileName).scaled(ui->BFS->size()));
        ui->BFS->setAutoFillBackground(true);
        ui->BFS->setPalette(pal);

        const QString fileName2 = "yfu.png";
        QPalette pal2 = ui->DFS->palette();
        pal2.setBrush(QPalette::Base, QPixmap(fileName2).scaled(ui->DFS->size()));
        ui->DFS->setAutoFillBackground(true);
        ui->DFS->setPalette(pal2);

    bool isok1;
    n = QInputDialog::getInt(nullptr, "Vertex", "请输入节点个数", 1, 1, 10000, 1, &isok1, Qt::Dialog | Qt::Wi
    if (!isok1)
        return;
    bool isok2;
    s = QInputDialog::getInt(nullptr, "Edge", "请输入边个数", 1, 1, 10000, 1, &isok2, Qt::Dialog | Qt::Window
    if (!isok2)
        return;
    fir= QInputDialog::getInt(nullptr, "Begin", "请输入起始节点", 1, 1, 10000, 1, &isok1, Qt::Dialog | Qt::Wi
    if (!isok1)
        return;
    QStandardItemModel *model = new QStandardItemModel(ui->BFS);//创建模型指定父类
    ui->BFS->setModel(model);
    model->setHorizontalHeaderLabels(QStringList()<<QStringLiteral("Breadth-first Search"));
    QStandardItemModel *model2 = new QStandardItemModel(ui->DFS);//创建模型指定父类
    ui->DFS->setModel(model2);
```

五．实验结果

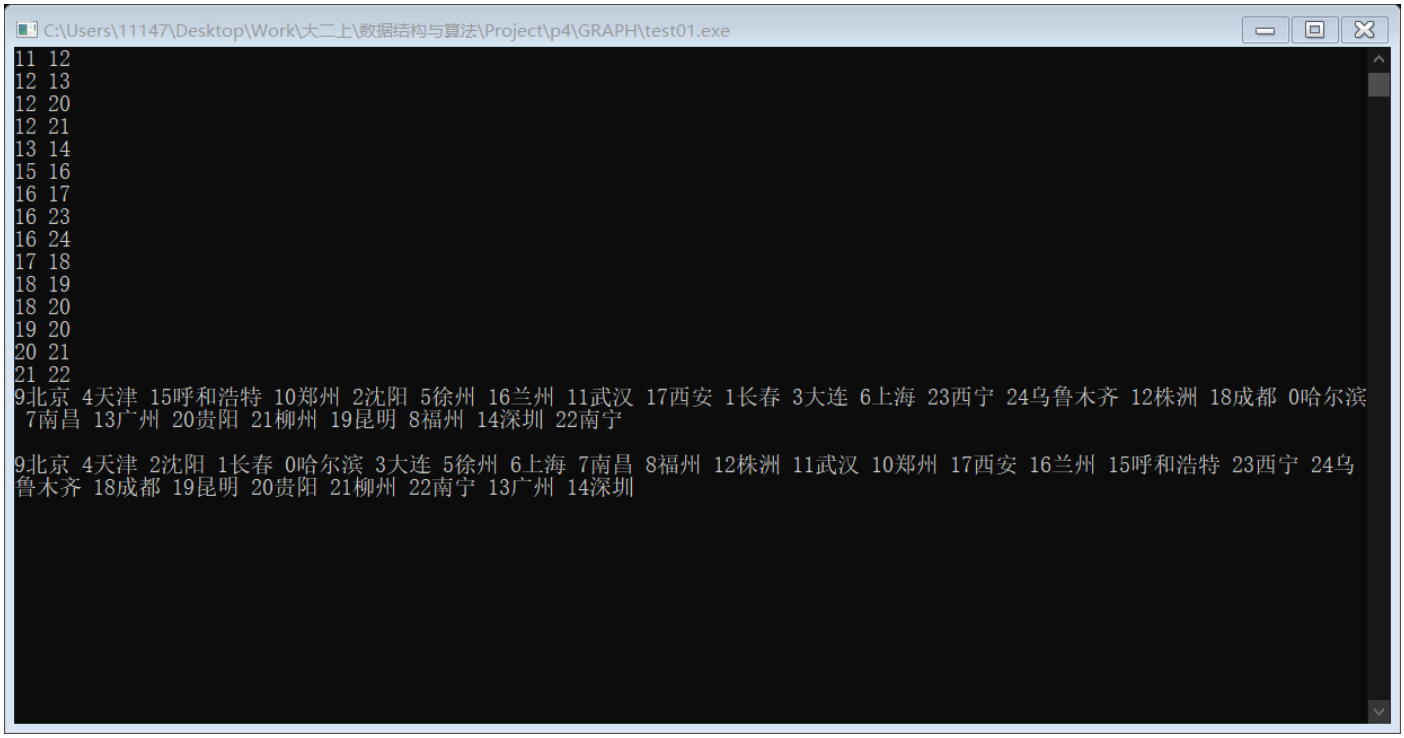
1.输入

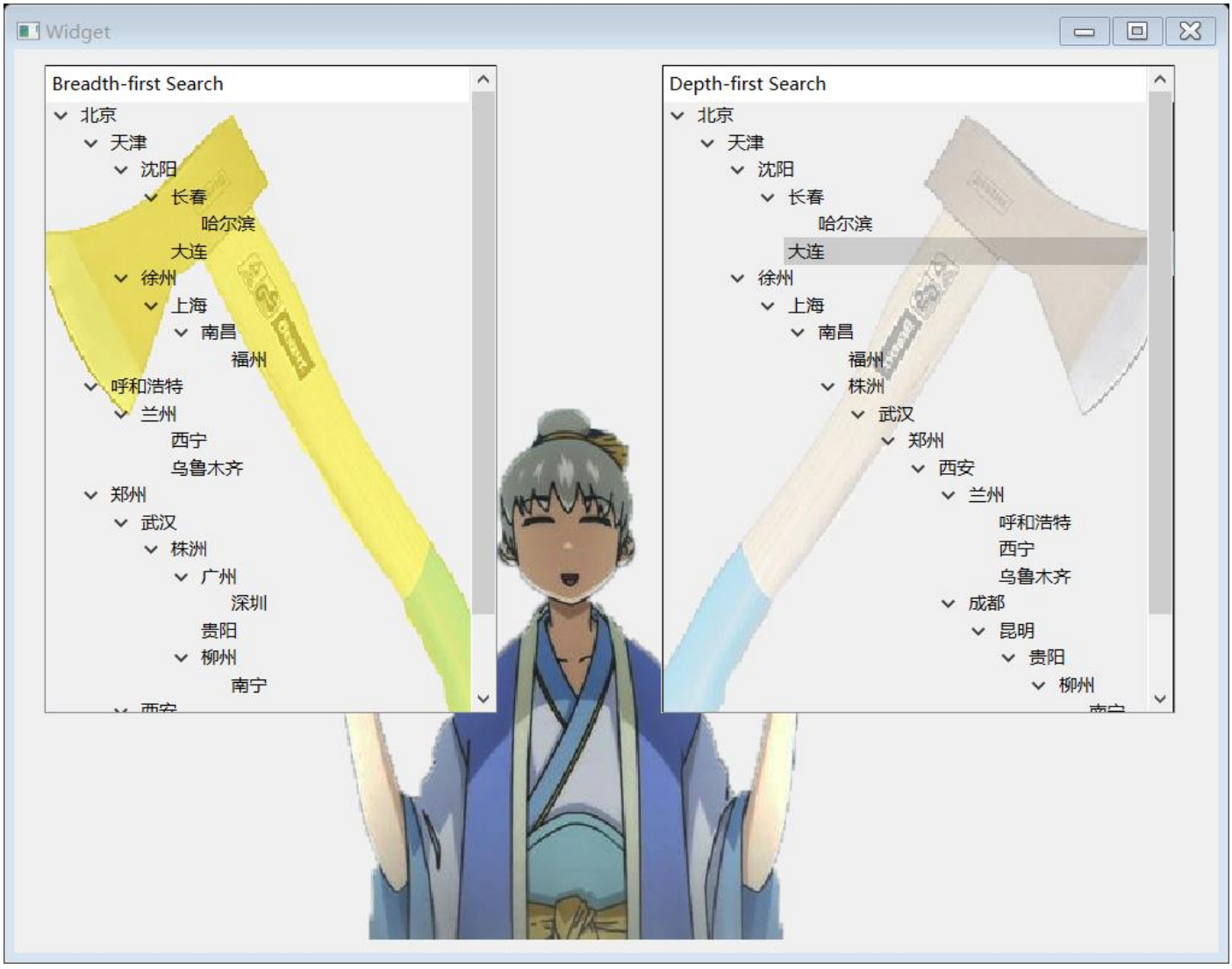




2.输出

黑框输出深度优先和广度优先遍历结构，之后有窗口带着树形结构输出





六．压缩包文件说明

1.该程序先在 VScode 上完成，后转移到 QT 进行 ui 设计。

3.test01 文件夹中为相应 QT 文件。

4.GRAPH 文件夹中为完成的 exe 程序。