

「Project 6 图的遍历」

21307035 邓栩瀛

1、程序功能简要说明。

- (1) 实现联通无向图的深度优先和广度优先遍历。
- (2) 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。
- (3) 自定义栈类型，用非递归算法实现深度优先遍历。
- (4) 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。

2、程序运行截图，包括计算功能演示、部分实际运行结果展示、命令行或交互式界面效果等。

(1) 创建图

```
Please enter your choice:1
请输入地点数和路径数
25 30
请按顺序输入地点名称
北京
天津
沈阳
长春
哈尔滨
大连
徐州
上海
南昌
福州
株洲
广州
深圳
武汉
郑州
西安
成都
昆明
贵阳
柳州
南宁
呼和浩特
兰州
西宁
乌鲁木齐
```

```
输入每条路径上的两个地点的序号和距离：
1 2 137
1 15 695
1 22 668
2 3 704
2 7 674
3 4 305
3 6 397
4 5 242
5 8 651
7 15 349
8 9 825
9 10 622
9 11 367
11 12 675
11 14 409
11 19 902
11 20 672
12 13 140
14 15 534
15 16 511
16 17 842
16 23 676
17 18 1100
17 19 967
18 19 639
19 20 607
20 21 255
22 23 1145
23 24 216
23 25 1892
```

(2) 输出深度优先遍历序列及边集

```
Please enter your choice:2
Please enter the source point
1
北京 呼和浩特 兰州 乌鲁木齐 西宁 西安 成都 贵阳 柳州 南宁 株洲 武汉 郑州 徐州 天津 沈阳 大连 长春 哈尔滨 上海 南昌 福州 广州 深圳 昆明
Please enter your choice:3
Please enter the source point
1
北京——>呼和浩特:668
呼和浩特——>兰州:1145
兰州——>乌鲁木齐:1892
兰州——>西宁:216
兰州——>西安:676
西安——>成都:842
成都——>贵阳:967
贵阳——>柳州:607
柳州——>南宁:255
柳州——>株洲:672
株洲——>武汉:409
武汉——>郑州:534
郑州——>徐州:349
徐州——>天津:674
天津——>沈阳:704
沈阳——>大连:397
沈阳——>长春:305
长春——>哈尔滨:242
哈尔滨——>上海:651
上海——>南昌:825
南昌——>福州:622
株洲——>广州:675
广州——>深圳:140
贵阳——>昆明:639
```

(3) 输出广度优先遍历序列及边集

```
Please enter your choice:4
Please enter the source point
1
北京 呼和浩特 郑州 天津 兰州 西安 武汉 徐州 沈阳 乌鲁木齐 西宁 成都 株洲 大连 长春 贵阳 昆明 柳州 广州 南昌 哈尔滨 南宁 深圳 福州 上海
```

```
Please enter your choice:5
Please enter the source point
1
北京--->呼和浩特:668
北京--->郑州:695
北京--->天津:137
呼和浩特--->兰州:1145
郑州--->西安:511
郑州--->武汉:534
郑州--->徐州:349
天津--->沈阳:704
兰州--->乌鲁木齐:1892
兰州--->西宁:216
西安--->成都:842
武汉--->株洲:409
沈阳--->大连:397
沈阳--->长春:305
成都--->贵阳:967
成都--->昆明:1100
株洲--->柳州:672
株洲--->广州:675
株洲--->南昌:367
长春--->哈尔滨:242
柳州--->南宁:255
广州--->深圳:140
南昌--->福州:622
南昌--->上海:825
```

(4) 输出深度优先生成树

```
Please enter your choice:6
Please enter the source point
1
(北京, 呼和浩特) (呼和浩特, 兰州) (兰州, 乌鲁木齐) (兰州, 西宁) (兰州, 西安) (西安, 成都) (成都, 贵阳) (贵阳, 柳州) (柳州, 南宁) (柳州, 株洲) (株洲, 武汉) (武汉, 郑州) (郑州, 徐州) (
徐州, 天津) (天津, 沈阳) (沈阳, 大连) (沈阳, 长春) (长春, 哈尔滨) (哈尔滨, 上海) (上海, 南昌) (南昌, 福州) (株洲, 广州) (广州, 深圳) (贵阳, 昆明)
北京
呼和浩特
兰州
西安 西宁 乌鲁木齐
成都
贵阳
昆明 柳州
株洲 南宁
广州 武汉
深圳 郑州
徐州
天津 上海
沈阳 南昌
长春 大连 福州
哈尔滨
```

(5) 输出广度优先生成树

```
Please enter your choice:7
Please enter the source point
1
(北京, 呼和浩特) (北京, 郑州) (北京, 天津) (呼和浩特, 兰州) (郑州, 西安) (郑州, 武汉) (郑州, 徐州) (天津, 沈阳) (兰州, 乌鲁木齐) (兰州, 西宁) (西安, 成都) (武汉, 株洲) (沈阳, 大连) (
沈阳, 长春) (成都, 贵阳) (成都, 昆明) (株洲, 柳州) (株洲, 广州) (株洲, 南昌) (长春, 哈尔滨) (柳州, 南宁) (广州, 深圳) (南昌, 福州) (南昌, 上海)
天津 郑州 呼和浩特
沈阳 徐州 武汉 西安 兰州 西宁 乌鲁木齐
长春 大连 上海 株洲 成都
哈尔滨 南昌 广州 柳州 昆明 贵阳
福州 深圳 南宁
```

3、部分关键代码及其说明。

(1) 深度优先遍历

```
1 void DFS(ALGraph &G, int source)
2 {
3     visited.assign(visited.size(), 0);
4     Stack s;
5     s.push(source);
6     visited[source] = 1;
7     cout << G.vertices[source].location << " ";
8     while (!s.empty())
9     {
10         int current;
11         current = s.Top();
12         ArcNode *p = G.vertices[current].firstarc;
13         while (p)
```

```

14     {
15         if (visited[p->adjvex])
16         {
17             p = p->nextarc;
18         }
19         else
20         {
21             cout << G.vertices[p->adjvex].location << " ";
22             s.push(p->adjvex);
23             visited[p->adjvex] = 1;
24             break;
25         }
26         if (p == NULL)
27         {
28             s.pop();
29         }
30     }
31 }
32 cout << endl;
33 }

```

(2)广度优先遍历

```

1 void BFS(ALGraph G, int source)
2 {
3     visited.assign(visited.size(), 0);
4     int u;
5     queue<int> Q;
6     if (visited[source] == 0)
7     {
8         visited[source] = 1;
9         cout << G.vertices[source].location << " ";
10        Q.push(source);
11        while (!Q.empty())
12        {
13            u = Q.front();
14            Q.pop();
15            ArcNode *z;
16            z = (ArcNode *)malloc(sizeof(ArcNode));
17            if (!z)
18                exit(-1);
19            z = G.vertices[u].firstarc;
20            int w;
21            for (; z; z = z->nextarc)
22            {
23                w = z->adjvex;
24                if (visited[w] == 0)
25                {
26                    visited[w] = 1;
27                    cout << G.vertices[w].location << " ";
28                    Q.push(w);
29                }
30            }
31        }
32    }
33    cout << endl;
34 }

```

(3)栈的定义与实现

```

1  class node
2  {
3  public:
4      int data;
5      node *next;
6  };
7
8  class Stack
9  {
10 public:
11     node *top;
12     Stack()
13     { //利用构造函数将栈指针设置为NULL,否则top指针就成了野指针,因为没有指向一块内存
14         top = NULL;
15     }
16     void push(int x) //进栈操作
17     {
18         node *s = new node;
19         s->data = x;
20         s->next = top; //结点的next域指向头指针
21         top = s;       //头指针再指向新结点
22     }
23     int pop() //出栈操作
24     {
25         if (top == NULL)
26         {
27             cout << "Error" << endl;
28             return 0;
29         }
30         else
31         {
32             node *p = top;
33             int x = top->data;
34             top = top->next;
35             delete p;
36             return x;
37         }
38     }
39     bool empty()
40     {
41         if (top == NULL)
42         {
43             return true;
44         }
45         else
46             return false;
47     }
48     int Top()
49     {
50         return top->data;
51     }
52 };

```

(4) 图的创建

```

1  void CreateGraph(ALGraph &G)
2  {
3      cout << "请输入地点数和路径数" << endl;
4      cin >> G.vexnum >> G.arcnum;
5      cout << "请按顺序输入地点名称" << endl;

```

```

6   for (k = 0; k < G.vexnum; k++)
7   {
8       cin >> G.vertices[k].location;
9       G.vertices[k].firstarc = NULL;
10  }
11  cout << "===== " << endl;
12  cout << "输入每条路径上的两个地点的序号和距离: " << endl;
13  for (k = 0; k < G.arcnum; k++)
14  {
15      cin >> vi >> vj >> w;
16      G.weight[vi-1][vj-1] = w;
17      G.weight[vj-1][vi-1] = w;
18      ArcNode *pArcNode = (ArcNode *)malloc(sizeof(ArcNode));
19      pArcNode->adjvex = vj-1;
20      pArcNode->nextarc = G.vertices[vi-1].firstarc;
21      G.vertices[vi-1].firstarc = pArcNode;
22
23      pArcNode = (ArcNode *)malloc(sizeof(ArcNode));
24      pArcNode->adjvex = vi-1;
25      pArcNode->nextarc = G.vertices[vj-1].firstarc;
26      G.vertices[vj-1].firstarc = pArcNode;
27  }
28  }

```

4、程序运行方式简要说明。

通过键入图中的各信息（点、边、带权值）创建图，并能实现图的广度优先遍历和深度优先遍历，可输出遍历序列及其边集；同时可以生成深度优先生成树和广度优先生成树，并以树形结构输出。

5、测试数据集

```

1 //25个地点:
2 北京
3 天津
4 沈阳
5 长春
6 哈尔滨
7 大连
8 徐州
9 上海
10 南昌
11 福州
12 株洲
13 广州
14 深圳
15 武汉
16 郑州
17 西安
18 成都
19 昆明
20 贵阳
21 柳州
22 南宁
23 呼和浩特
24 兰州
25 西宁
26 乌鲁木齐
27 //30条边:

```

28	1 2 137
29	1 15 695
30	1 22 668
31	2 3 704
32	2 7 674
33	3 4 305
34	3 6 397
35	4 5 242
36	5 8 651
37	7 15 349
38	8 9 825
39	9 10 622
40	9 11 367
41	11 12 675
42	11 14 409
43	11 19 902
44	11 20 672
45	12 13 140
46	14 15 534
47	15 16 511
48	16 17 842
49	16 23 676
50	17 18 1100
51	17 19 967
52	18 19 639
53	19 20 607
54	20 21 255
55	22 23 1145
56	23 24 216
57	23 25 1892