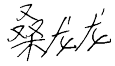
数据结构实验报告

<table>
<tr><td>学号-姓名</td><td>桑龙龙 20030540015</td><td>实验时间</td><td>2020 年 12 月 19 日</td></tr>
<tr><td>诚信声明</td><td colspan="3">本实验及实验报告所写内容为本人所作  </td></tr>
<tr><td>实验题目</td><td colspan="3">图的遍历
题目一 深度优先遍历
题目二 广度优先遍历</td></tr>
<tr><td>实验过程中遇到的主要问题</td><td colspan="3">无</td></tr>
<tr><td>实验小结</td><td colspan="3">本次试验进行了图的遍历的试验，在本次实验中分别用邻接矩阵和邻接链表的方法实现图的构造，并分别进行了图的深度优先遍历和广度优先遍历。</td></tr>
<tr><td>数据结构
（自定义数据类型）</td><td colspan="3">

```
struct adjMatrics{
    //邻接矩阵
    int** g;
    //矩阵
    int vertexNum;
    //节点数
    int edgeNum;
    //边数
    adjMatrics(int n){
        vertexNum=n;
        g=new int*[n+1];
        for(int i=1;i<=n;i++){
            g[i]=new int[n+1];
            memset(g[i],0,sizeof(int)*(n+1));
        }
    }
    ~adjMatrics(){
        for(int i=0;i<vertexNum;i++) delete []g[i];
        delete []g;
    }
    void add(int from,int to,int weight){
        //加入一条权重为 weight 由 from 到 to 的边
        g[from][to]=weight;
    }
};
```

</td></tr>
</table>

```
struct adjList{
    //邻接链表
    int vertexNum;
    int edgeNum;
    int* head;
    struct edge{
        int to;
        int weight;
        int next;
    }e[N];
    int tot;
    //静态链表实现的邻接链表
    adjList(int n){
        vertexNum=n;
        head=new int[n+1];
        memset(head,0,(n+1)*sizeof(int));
        memset(e,0,sizeof(e));
        tot=0;
    }
    adjList(){
        delete []head;
    }
    void add(int from,int to,int weight){
        //加入一条权重为 weight 由 from 到 to 的边
        e[++tot].to=to;
        e[tot].weight=weight;
        e[tot].next=head[from];
        head[from]=tot;
    }
};
```

主要算法
（或算法说明）

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <queue>
#define N 10000
struct adjMatrics{
    //邻接矩阵
    int** g;
    //矩阵
    int vertexNum;
    //节点数
    int edgeNum;
    //边数
    adjMatrics(int n){
        vertexNum=n;
        g=new int*[n+1];
        for(int i=1;i<=n;i++){
            g[i]=new int[n+1];
            memset(g[i],0,sizeof(int)*(n+1));
        }
    }
    ~adjMatrics(){
        for(int i=0;i<vertexNum;i++) delete []g[i];
        delete []g;
    }
    void add(int from,int to,int weight){
        //加入一条权重为 weight 由 from 到 to 的边
        g[from][to]=weight;
    }
};
struct adjList{
    //邻接链表
    int vertexNum;
    int edgeNum;
    int* head;
    struct edge{
        int to;
        int weight;
        int next;
    }e[N];
    int tot;
    //静态链表实现的邻接链表
    adjList(int n){
        vertexNum=n;
        head=new int[n+1];
```

```
        memset(head,0,(n+1)*sizeof(int));
        memset(e,0,sizeof(e));
        tot=0;
    }
    adjList(){
        delete []head;
    }
    void add(int from,int to,int weight){
        //加入一条权重为 weight 由 from 到 to 的边
        e[++tot].to=to;
        e[tot].weight=weight;
        e[tot].next=head[from];
        head[from]=tot;
    }
};
bool vis[N];
void bfsAdjMatrics(int cur,adjMatrics& graph){
    //广度优先遍历（邻接矩阵），由点 cur 初始
    memset(vis+1,0,graph.vertexNum);
    std::queue<int> Q;
    Q.push(cur);
    vis[cur]=true;
    printf("sAdjMatrics BFS:\n");
    while(!Q.empty()){
        cur=Q.front();
        Q.pop();
        printf("%d ",cur);
        for(int i=1;i<=graph.vertexNum;i++){
            if(vis[i] || !graph.g[cur][i]) continue;
            Q.push(i);
            vis[i]=true;
        }
    }
    printf("\n");
}
void dfsAdjMatrics(int cur,adjMatrics& graph){
    //深度优先遍历（邻接矩阵）
    if(vis[cur]) return;
    printf("AdjMatrics dfs vis: %d\n",cur);
    vis[cur]=true;//标记为访问过
    for(int i=1;i<=graph.vertexNum;i++){
        if(!graph.g[cur][i]) continue;
        dfsAdjMatrics(i,graph);
    }
    vis[cur]=false;//取消访问标记
```

```
}
void bfsAdjList(int cur,adjList& graph){
    //广度优先遍历（邻接链表），由点 cur 初始
    memset(vis+1,0,graph.vertexNum);
    std::queue<int> Q;
    Q.push(cur);
    vis[cur]=true;
    printf("adjList BFS:\n");
    while(!Q.empty()){
        cur=Q.front();
        Q.pop();
        printf("%d ",cur);
        for(int i=graph.head[cur];i;i=graph.e[i].next){
            if(vis[graph.e[i].to]) continue;
            Q.push(graph.e[i].to);
            vis[graph.e[i].to]=true;
        }
    }
    printf("\n");
}
void dfsAdjList(int cur,adjList& graph){
    //深度优先遍历（邻接链表）
    if(vis[cur]) return;
    printf("AdjList dfs vis: %d\n",cur);
    vis[cur]=true;
    for(int i=graph.head[cur];i;i=graph.e[i].next){
        dfsAdjList(graph.e[i].to,graph);
    }
    vis[cur]=false;
}
adjMatrics* buildAdjMatrics(){
    //初始化一个邻接矩阵
    int n,m,from,to;
    scanf("%d %d\n",&n,&m);
    adjMatrics* graph=new adjMatrics(n);
    graph->edgeNum=m;
    for(int i=0;i<m;i++){
        scanf("%d %d\n",&from,&to);
        graph->add(from,to,1);
    }
    return graph;
}
adjList* buildAdjList(){
    //初始化一个邻接链表
    int n,m,from,to;
```

```
    scanf("%d %d\n",&n,&m);
    adjList* graph=new adjList(n);
    graph->edgeNum=m;
    for(int i=0;i<m;i++){
        scanf("%d %d\n",&from,&to);
        graph->add(from,to,1);
    }
    return graph;
}
/*  输入说明：
    第一行两个数 n，m
    之后 m 行，每行 3 个数 from，to，weight
    解释：
    n 为节点个数（节点编号由 1 到 n），m 为边数
    from，to，weight 表示一条由 from 到 to 的权重为 weight 的边

*/
int main(){
    if(0){
        //邻接链表测试
        adjList* g=buildAdjList();
        dfsAdjList(1,*g);
        bfsAdjList(1,*g);
    }else{
        //邻接矩阵测试
        adjMatrics* g=buildAdjMatrics();
        dfsAdjMatrics(1,*g);
        bfsAdjMatrics(1,*g);
    }

}
```