ALG LAB8

王世炟 PB20151796

2022/12/3

图搜索 BFS 算法及存储优化

实验内容

设计实现图存储方式（邻接矩阵，邻接表中或 CSR，至少实现两种），并根据给定的图数据选择合适的存储方式进行存储（存储方式选择也是实验的检查内容之一），进行图的广度优先遍历。

算法设计思路

图存储方式

本次实验选择的两种图存储方式是邻接矩阵和邻接表。对于图 $G(V,E)$ 来说，其邻接表由一个包含 $|V|$ 条链表的数组 Adj 所构成，每个节点有一条链表。对于每个节点 $u \in V$,邻接表 $Adj[u]$ 包含所有与结点u之间有边相连的结点 v 。对于邻接矩阵表示来说，通常将图 G 中的结点编为 $1,2,\cdots,|V|$ 。图 G 由一个 $|V| \times |V|$ 的矩阵 $A=(a_{ij})$ 予以表示，矩阵满足以下条件：

$$a_{ij}=\begin{cases}1 & 若(i,j)\in E\\ 0 & 其他\end{cases}$$

空间复杂度：

存储方式	空间复杂度
邻接表	$\Theta(V+E)$
邻接矩阵	$\Theta(V^2)$

由空间复杂度可见，如果结点数较少且边比较密集，那么使用邻接矩阵存储图优于邻接表；如果结点数较多且边比较稀疏，那么使用邻接表存储图优于邻接矩阵。

BFS广度优先遍历

广度优先搜索首先将所有结点赋为白色，然后维护一个队列，首先将源节点加入队列，然后每次出队一个节点，将节点赋为灰色，然后将所有与该节点相连的节点入队，之后将当前节点赋为黑色。重复上述步骤，直至队空，遍历完成。

源码+注释

邻接矩阵：

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

using namespace std;


//边上权类型
typedef int EdgeType;

//最大顶点数
#define MaxVEX 100

// 颜色
enum Color
{
    White,
    Black,
    Grey
};

//邻接矩阵
typedef struct MGraph
{
    //顶点表
    Color vex[MaxVEX];
    //邻接矩阵，边表
    EdgeType edge[MaxVEX][MaxVEX];

    //图中当前的顶点数和边数
    int numVertexes, numEdges;
} MGraph;

MGraph G;


void BFS(int v)
{
    if (G.vex[v] != White)
    {
        return;
    }
    queue<int> q;
    q.push(v);
    G.vex[v] = Grey;
    while (!q.empty())
    {
        int temp = q.front();
        q.pop();
        cout << temp + 1 << "   ";
        for (int i = 0; i < G.numVertexes; i++)
        {
            if ((G.edge[temp][i] == 1) && (G.vex[i] == White))
            {
                q.push(i);
```

```
                G.vex[i] = Grey;
            }
        }
        G.vex[temp] = Black;
    }
    return;
}


int main()
{
    int p, q;
    int numEdges = 0, numVertexes = 0;
    string str;
    set<string> s;
    ifstream infile("data.txt", ios_base::in);
    while (infile >> str)
    {
        s.insert(str);
        numEdges++;
    }
    infile.close();
    infile.clear(ios::goodbit);  // 恢复流的状态
    numEdges /= 2;
    numVertexes = s.size();
    G.numEdges = numEdges;
    G.numVertexes = numVertexes;
    for (int i = 0; i < G.numVertexes; i++) {
        G.vex[i] = White;       // 结点颜色初始化
        for (int j = 0; j < G.numVertexes; j++) {
            //邻接矩阵初始化
            G.edge[i][j] = 0;
        }
    }
    infile.open("data.txt", ios_base::in);
    for (int i = 0; i < G.numEdges; i++)
    {
        infile >> p;
        infile >> q;
        G.edge[p - 1][q - 1] = 1;
        G.edge[q - 1][p - 1] = 1;
    }
    // for (int i = 0; i < G.numVertexes; i++)
    // {
    //     for (int j = 0; j < G.numVertexes; j++)
    //     {
    //         printf("%4d", G.edge[i][j]);
    //     }
    //     printf("\n");
    // }
    for (int i = 0; i < G.numVertexes; i++)
    {
        BFS(i);
```

```
    }
    return 0;
}
```

邻接表：

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

using namespace std;

// 颜色
enum Color
{
    White,
    Black,
    Grey
};

// 边上权类型
typedef int EdgeType;

// 顶点类型
typedef string VertexType;

// 最大顶点数
#define MaxVEX 1000000

// 边表结点
typedef struct EdgeNode
{
    // 邻接点域，存储该顶点对应的下标
    int adjVex;

    // 链域，指向下一个邻接点
    EdgeNode *next;
} EdgeNode;

//顶点表结点
typedef struct VertexNode
{
    VertexType data;     // 顶点域，存储顶点信息
    EdgeNode *firstEdge; // 边表头指针
    Color color;         // 颜色
} VertexNode, AdjList[MaxVEX];

typedef struct GraphAdjList
{
    AdjList adjList;
    //图中当前顶点数和边数
    int numVertexes, numEdges;
} GraphAdjList;

GraphAdjList G;

void BFS(int v, ofstream &outfile)
{
    int count = 0;
    if (G.adjList[v].color != White)
    {
```

```
        return;
    }
    queue<int> q;
    q.push(v);
    G.adjList[v].color = Grey;

    while (!q.empty())
    {
        int temp = q.front();
        q.pop();
        // cout << G.adjList[temp].data << "   ";
        outfile << G.adjList[temp].data << endl;
        count++;
        EdgeNode *temp1 = G.adjList[temp].firstEdge;
        while (temp1)
        {
            if (G.adjList[temp1->adjVex].color == White)
            {
                q.push(temp1->adjVex);
                G.adjList[temp1->adjVex].color = Grey;
            }
            temp1 = temp1->next;
        }
        G.adjList[temp].color = Black;
    }
    cout << "BFS遍历到的顶点数为：" << count << endl;
    return;
}

int main()
{
    int numEdges = 0, numVertexes = 0;
    string p;
    set<string> s;
    map<string, int> m;
    clock_t begin, end;

    ifstream infile("D:\\USR\\dataset\\twitter_small.txt", ios_base::in);
    while (infile >> p)
    {
        s.insert(p);
        numEdges++;
    }
    infile.close();
    infile.clear(ios::goodbit); // 恢复流的状态
    numEdges /= 2;
    numVertexes = s.size();
    G.numEdges = numEdges;
    G.numVertexes = numVertexes;
    // 根据顶点信息，建立顶点表
    int i = 0;
    for (auto it = s.begin(); it != s.end(); it++, i++)
    {
```

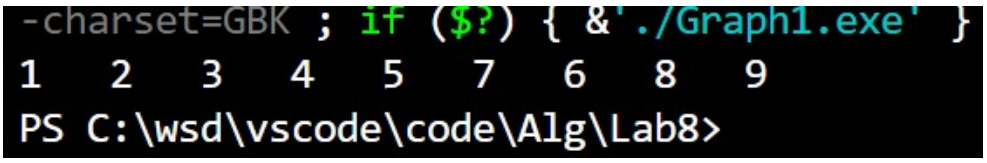
```
        G.adjList[i].data = *it;
        m.insert(pair<string, int>(*it, i)); // 编码
        G.adjList[i].firstEdge = NULL;       // 将边表置为空表
        G.adjList[i].color = White;
    }
    infile.open("D:\\USR\\dataset\\twitter_small.txt", ios_base::in);
    EdgeNode *e;
    //建立边表
    string str1, str2;
    for (int k = 0; k < G.numEdges; k++)
    {
        // 结点插入采用：头插法
        infile >> str1;
        infile >> str2;
        e = (EdgeNode *)malloc(sizeof(EdgeNode));// 向内存申请空间
        //邻接序号为j
        e->adjVex = m.find(str2)->second;
        //将e指针指向当前顶点指向的结点
        e->next = G.adjList[m.find(str1)->second].firstEdge;
        //将当前顶点的指针指向e
        G.adjList[m.find(str1)->second].firstEdge = e;
    }
    string str;
    cout << "请输入编号：" << endl;
    cin >> str;
    if (!s.count(str))
    {
        cout << "顶点不存在";
        return 0;
    }

    begin = clock();
    ofstream outfile("small.txt", ios_base::out);
    BFS(m[str], outfile);
    end = clock();
    cout << "耗时 " << (double)(end - begin) / CLK_TCK << "s" << endl;
    return 0;
}
```

算法测试结果

data.txt:

```
1 2
1 3
2 4
2 5
2 7
3 8
3 7
3 6
4 5
4 7
5 8
4 8
4 9
5 9
6 8
6 9
8 7
8 9
```

使用邻接矩阵，运行结果：



twitter_small.txt:

使用邻接表，运行结果：