

# 重 庆 大 学

## 学 生 实 验 报 告

实验课程名称 数据结构与算法

开课实验室 DS1501

学 院 软件学院 年级 2021 专业班 软件 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 1 学期

总 成 绩	
教师签名	XXX

软件学院制

# 《数据结构与算法》实验报告

开课实验室：DS1501

2022 年 11 月 17 日

学院	软件学院	年级、专业、班	2021 级软件工 程 X 班	姓名	XXX	成绩	
课程 名称	数据结构与算法	实验项目 名 称	2022-2023 学年第一学期数 据结构与算法上机练习 005	指导教师	XXX		
教 师 评 语	教师签名：  年 月 日						

## 一、实验目的

- 请认真学习第 11 章“图”的各项基本知识和算法，掌握图的基本概念，主要算法的基本思想和实现方式；
- 在已经实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）的基础上，完成以下任务：
- 图的遍历算法：深度优先搜索（DFS）和广度优先搜索（BFS）；
- 拓扑排序算法：TopSort（包括递归版和队列版）；
- 请完成以上实验任务，撰写实验报告，包括算法关键代码和简单的算法运行时间代价分析。

## 二、使用仪器、材料

PC 微机；  
Windows 操作系统，VS2022 编译环境；

## 三、实验步骤

- 1、实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）；
- 2、实现图的遍历算法：深度优先搜索（DFS）和广度优先搜索（BFS）；
- 3、实现拓扑排序算法：TopSort（包括递归版和队列版）；
- 4、完成以上实验任务，撰写实验报告，包括算法关键代码和简单的算法运行时间代价分析。

#### 四、实验过程原始记录(数据、图表、计算等)

1、实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）；

关键代码：

```
//书P251 图11.6 图的相邻矩阵实现
class Graphm :public Graph {
private:
    int numVertex, numEdge;
    int** matrix;
    int* mark;
public:
    Graphm(int numVert) { Init(numVert); }
    ~Graphm() {
        delete[] mark;
        for (int i = 0; i < numVertex; i++)
            delete[] matrix[i];
        delete[] matrix;
    }
    void Init(int n) {
        int i;
        numVertex = n;
        numEdge = 0;
        mark = new int[n];
        for (i = 0; i < numVertex; i++) {
            mark[i] = 0;
            matrix = (int**)new int* [numVertex];
        }
        for (int i = 0; i < numVertex; i++) {
            matrix[i] = new int[numVertex];
        }
        for (int i = 0; i < numVertex; i++)
        {
            for (int j = 0; j < numVertex; j++) {
                matrix[i][j] = 0;
            }
        }
    }
}
```

```

int n() { return numVertex; }
int e() { return numEdge; }
int first(int v) {
    for (int i = 0; i < numVertex; i++) {
        if (matrix[v][i] != 0) {
            return i;
        }
    }
    return numVertex;
}
int next(int v, int w)
{
    for (int i = w + 1; i < numVertex; i++)
    {
        if (matrix[v][i] != 0)
            return i;
    }
    return numVertex;
}
void setEdge(int v1, int v2, int wght) {
    if (matrix[v1][v2] == 0) {
        numEdge++;
        matrix[v1][v2] = wght;
    }
}
void delEdge(int v1, int v2) {
    if (matrix[v1][v2] != 0) {
        matrix[v1][v2] = 0;
    }
}
bool isEdge(int i, int j) {
    return matrix[i][j] != 0;
}
int weight(int v1, int v2) {
    return matrix[v1][v2];
}
int getMark(int v) {
    return mark[v];
}
void setMark(int v, int val) {
    mark[v] = val;
}
};

```

## 2、实现图的遍历算法：深度优先搜索（DFS）和广度优先搜索（BFS）；

### （1）关键代码：

#### ①深度优先搜索（DFS）

```
//书P254 深度优先搜索
void DFS(Graph* G, int v)
{
    G->setMark(v, 1);
    cout << v << ' ';
    for (int w = G->first(v); w < G->n(); w = G->next(v, w))
        if (G->getMark(w) == 0)
        {
            DFS(G, w);
        }
}
```

#### ②广度优先搜索（BFS）

```
//书P256 图11.10 广度优先图遍历算法的实现
void BFS(Graph* G, int start, AQueue* Q)
{
    int v, w;
    Q->enqueue(start);
    G->setMark(start, 1);
    cout << start << ' ';
    while (Q->length() != 0)
    {
        v = Q->dequeue();
        for (w = G->first(v); w < G->n(); w = G->next(v, w))
            if (G->getMark(w) == 0)
            {
                G->setMark(w, 1);
                cout << w << ' ';
                Q->enqueue(w);
            }
    }
}
```

(2) 验证程序:

①深度优先搜索 (DFS)

```
int main()
{
    Graphm *G;
    G = new Graphm(8);
    G->Init(8);
    G->setEdge(0, 1, 1);
    G->setEdge(1, 3, 1);
    G->setEdge(1, 4, 1);
    G->setEdge(3, 7, 1);
    G->setEdge(4, 6, 1);
    G->setEdge(0, 2, 1);
    G->setEdge(2, 5, 1);
    G->setEdge(5, 6, 1);
    G->setEdge(6, 2, 1);
    DFS(G, 0);
    cout << endl;
    return 0;
}
```

②广度优先搜索 (BFS)

```
int main()
{
    Graphm *G;
    G = new Graphm(8);
    G->Init(8);
    G->setEdge(0, 1, 1);
    G->setEdge(1, 3, 1);
    G->setEdge(1, 4, 1);
    G->setEdge(3, 7, 1);
    G->setEdge(4, 6, 1);
    G->setEdge(0, 2, 1);
    G->setEdge(2, 5, 1);
    G->setEdge(5, 6, 1);
    G->setEdge(6, 2, 1);
    AQueue* Q = new AQueue(12);
    BFS(G, 0, Q);
    cout << endl;
    return 0;
}
```

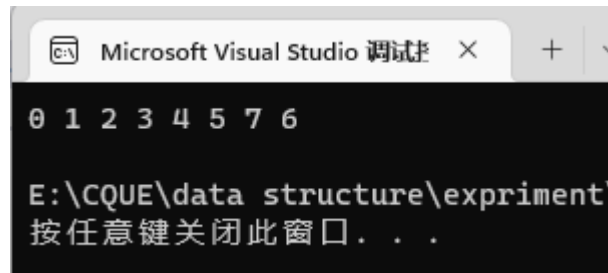
(3) 结果:

①深度优先搜索 (DFS)



```
0 1 3 7 4 6 2 5  
E:\CQUE\data structure\expriment\  
按任意键关闭此窗口...
```

②广度优先搜索 (BFS)



```
0 1 2 3 4 5 7 6  
E:\CQUE\data structure\expriment\  
按任意键关闭此窗口...
```

(4) 运行时间代价分析:

①深度优先搜索 (DFS)

在有向图中, DFS 对每一条边处理一次; 在无向图中, DFS 对每一条边都从两个方向处理; 每个顶点一定会访问到, 而且只访问一次, 因此总代价是  $\Theta(|V|+|E|)$ 。

②广度优先搜索 (BFS)

遍历顶点的每个相邻边的时间复杂度是  $\Theta(N)$ , 其中  $N$  是相邻边的数量。因此, 对于  $V$  个顶点, 时间复杂度变  $\Theta(V*N)=\Theta(E)$ , 其中  $E$  是图中边的总数。由于从队列中删除和添加顶点是  $\Theta(1)$ , 为什么将它添加到 BFS 的总运行时间代价为  $\Theta(V+E)$ 。

### 3、实现拓扑排序算法：TopSort（包括递归版和队列版）；

(1) 关键代码：

#### ①递归版

//书P257 图11.13 递归拓扑排序的实现

```
void tophelp(Graph* G, int v)
{
    G->setMark(v, 1);
    for (int w = G->first(v); w < G->n(); w = G->next(v, w))
        if (G->getMark(w) == 0)
            tophelp(G, w);
    cout << v << ' ';
}

void topsort1(Graph* G)
{
    int i;
    for (i = 0; i < G->n(); i++)
        G->setMark(i, 0);
    for (i = 0; i < G->n(); i++)
        if (G->getMark(i) == 0)
            tophelp(G, i);
}
```

#### ②队列版

//书P258 图11.15 基于队列的拓扑排序算法

```
void topsort2(Graph* G, AQueue* Q)
{
    int Count[12] = { 0 };
    int v, w;
    for (v = 0; v < G->n(); v++)
        Count[v] = 0;
    for (v = 0; v < G->n(); v++)
        for (w = G->first(v); w < G->n(); w = G->next(v, w))
            Count[w]++;
    for (v = 0; v < G->n(); v++)
        if (Count[v] == 0)
            Q->enqueue(v);
    while (Q->length() != 0)
    {
        v = Q->dequeue();
        cout << v << ' ';
        for (w = G->first(v); w < G->n(); w = G->next(v, w))
        {
            Count[w]--;
            if (Count[w] == 0)
                Q->enqueue(w);
        }
    }
}
```



(2) 验证程序:

①递归版

```
int main()
{
    Graphm *G;
    G = new Graphm(9);
    G->Init(9);
    G->setEdge(0, 2, 1);
    G->setEdge(0, 7, 1);
    G->setEdge(1, 2, 1);
    G->setEdge(1, 4, 1);
    G->setEdge(1, 3, 1);
    G->setEdge(2, 3, 1);
    G->setEdge(3, 5, 1);
    G->setEdge(3, 6, 1);
    G->setEdge(4, 5, 1);
    G->setEdge(7, 8, 1);
    G->setEdge(8, 6, 1);
    topsort1(G);
    cout << endl;
    return 0;
}
```

②队列版

```
int main()
{
    Graphm *G;
    G = new Graphm(9);
    G->Init(9);
    G->setEdge(0, 2, 1);
    G->setEdge(0, 7, 1);
    G->setEdge(1, 2, 1);
    G->setEdge(1, 4, 1);
    G->setEdge(1, 3, 1);
    G->setEdge(2, 3, 1);
    G->setEdge(3, 5, 1);
    G->setEdge(3, 6, 1);
    G->setEdge(4, 5, 1);
    G->setEdge(7, 8, 1);
    G->setEdge(8, 6, 1);
    AQueue* Q = new AQueue(12);
    topsort2(G, Q);
    cout << endl;
    return 0;
}
```

(3) 结果:

①递归版



```
Microsoft Visual Studio 调试 X + v
5 6 3 2 8 7 0 4 1
E:\CQUE\data structure\expriment\
按任意键关闭此窗口...
```

②队列版



```
Microsoft Visual Studio 调试 X + v
0 1 7 2 4 8 3 5 6
E:\CQUE\data structure\expriment\
按任意键关闭此窗口...
```

(4) 运行时间代价分析:

如果 AOV 网络有  $V$  个顶点,  $E$  条边; 在拓扑排序的过程中, 搜索入度为零的顶点所需的时间是  $\Theta(V)$ 。在正常情况下, 每个顶点进一次栈, 出一次栈, 所需时间  $\Theta(V)$ ; 每个顶点入度减 1 的运算共执行了  $E$  次。所以总的运行时间代价为  $\Theta(V+E)$ 。

## 五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。