CENTRAL SOUTH UNIVERSITY

算法分析与实验设计
实验报告（三）

学　　院：计算机学院

专业班级：大数据 234

学生姓名：LukiRyan

学　　号：234

指导教师：

年 5 月 12 日

LukiRyan

目录

LukiRyan

一 BFS 遍历

1. 问题

以邻接表的存储方式，实现图的 BFS 和 DFS 遍历，并分析复杂度。（100 分）

输入：

第一行输入两个数 m, n，表示图有 m 个顶点（所有顶点的字母各不相同），n 条边；

接下来 n 行每行输入两个顶点，表示这两个顶点之间有边相连；

最后一行输入遍历开始的顶点

输出:

从遍历开始的顶点出发，分别输出图的 BFS 和 DFS 遍历的结果（若某个节点存在多种遍历方式，则按照字母表顺序来进行遍历，即输出只有一种结果）

例子：

输入：

3,3

A,B

A,C

B,C

A

输出：

A,B,C

A,B,C

LukiRyan

2. 代码

我用的是 C 语言（本次实验用的都是 C 语言，并且需要用到 dev）：

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#define N 510

int n, m;                    // 节点数和边数
int g[N][N];                 // 邻接矩阵表示图的连接关系
bool vis[N];                 // 记录节点是否被访问过
int queue[N], front = 0, rear = 0;   // 队列用于广度优先搜索

void bfs() {
    while (front < rear) {              // 队列不为空时进行循环
        int x = queue[front++];         // 取出队首节点
        vis[x] = 1;                     // 标记当前节点为已访问
        char s = 'A' + x - 1;           // 将节点编号转换为对应的字母表示
        printf("%c ", s);               // 输出当前节点的字母表示
        for (int j = 1; j <= n; j++) {
            if (vis[j] != 1 && g[x][j]) {   // 若节点未被访问且与当前节点相邻
                queue[rear++] = j;          // 将节点入队
                vis[j] = 1;                 // 标记节点为已访问
            }
        }
    }
}

int main() {
    int a, b, c;
    char s1, s2, s3;
    scanf("%d%d", &n, &m);                  // 输入节点数和边数
    for (int i = 1; i <= m; i++) {
        scanf(" %c %c", &s1, &s2);          // 输入边的两个节点
        a = s1 - 'A' + 1;                   // 将节点的字母表示转换为对应的编号
        b = s2 - 'A' + 1;
        g[a][b] = g[b][a] = 1;              // 在邻接矩阵中标记边的存在
    }
    scanf(" %c", &s3);                      // 输入遍历起点的节点字母
    c = s3 - 'A' + 1;                       // 将起点字母转换为编号
    queue[rear++] = c;                      // 将起点节点入队
```

```
    bfs();                                  // 调用广度优先搜索函数进行遍历
    return 0;
}
//时间复杂度为 O(n+m)，空间复杂度为 O(n)
```

LukiRyan

3. 分析

在这段代码中，广度优先搜索（BFS）算法通过使用队列来实现节点的访问顺序。下面是广度优先搜索算法在这段代码中的体现：

1、创建队列：
定义 queue 数组作为队列，用于存储待访问的节点。
使用 front 和 rear 分别表示队列的前端和后端。

2、入队操作：
在主函数中，将起点节点 c 入队，即 queue[rear++] = c;。
在广度优先搜索函数 bfs() 中，当发现与当前节点相邻的未访问节点时，将其入队，即 queue[rear++] = j;。

3、出队操作：
在 bfs() 函数的循环中，通过 int x = queue[front++]; 操作，取出队列的头部节点 x。
队列的头部节点出队后，front 指针自增，表示队列的前端向后移动一位。

4、节点访问和标记：
当节点从队列中出队时，将其标记为已访问，即 vis[x] = 1;。
在出队操作后，输出当前节点的字母表示，即 printf("%c ", s);。

5、遍历相邻节点：
在 bfs() 函数的循环中，使用 for 循环遍历与当前节点相邻的所有节点。
当发现一个未被访问过的与当前节点相邻的节点时，将其入队，并标记为已访问，即 queue[rear++] = j; vis[j] = 1;。

通过以上操作，广度优先搜索算法在这段代码中实现了按照广度优先的顺序遍历图的节点。使用队列作为辅助数据结构，保证了节点的访问顺序是按照距离起点的距离逐层扩展的。

复杂度分析：

时间复杂度：

输入部分：输入节点数和边数的时间复杂度为 O(1)。
构建邻接矩阵部分：通过循环读取边的信息并在邻接矩阵中标记边的存在，时间复杂度为 O(m)，其中 m 是边数。
广度优先搜索部分：对于每个节点，最多会访问与其相邻的所有节点一次，因此时间复杂度为 O(n + m)，其中 n 是节点数，m 是边数。

总体时间复杂度为 O(m + n)。

空间复杂度：

邻接矩阵：使用二维数组 g[N][N] 表示图的连接关系，需要 O(n^2) 的空间。
访问状态数组：使用布尔数组 vis[N] 记录节点是否被访问过，需要 O(n) 的空间。
队列：使用数组 queue[N] 作为队列，最多需要存储 n 个节点，因此需要 O(n) 的空间。
其他变量和输入部分所需的空间可忽略不计。
总体空间复杂度为 O(n^2)。

综上所述，该代码的时间复杂度为 O(m + n)，空间复杂度为 O(n^2)。其中，m 是边数，n 是节点数。

LukiRyan

4. 测试

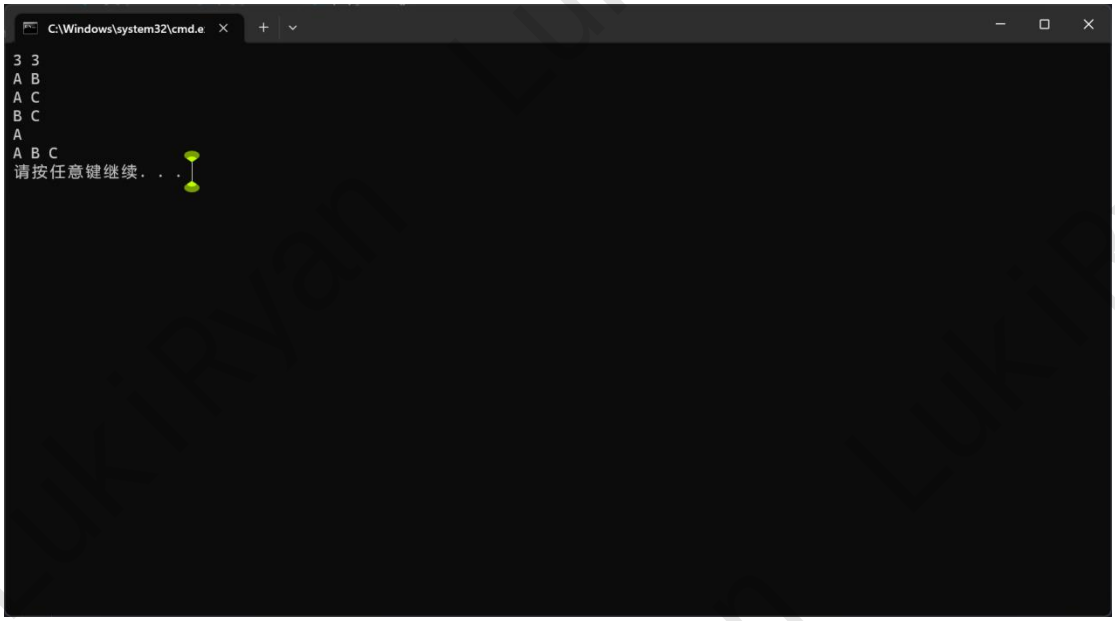

图 1：按样例输入

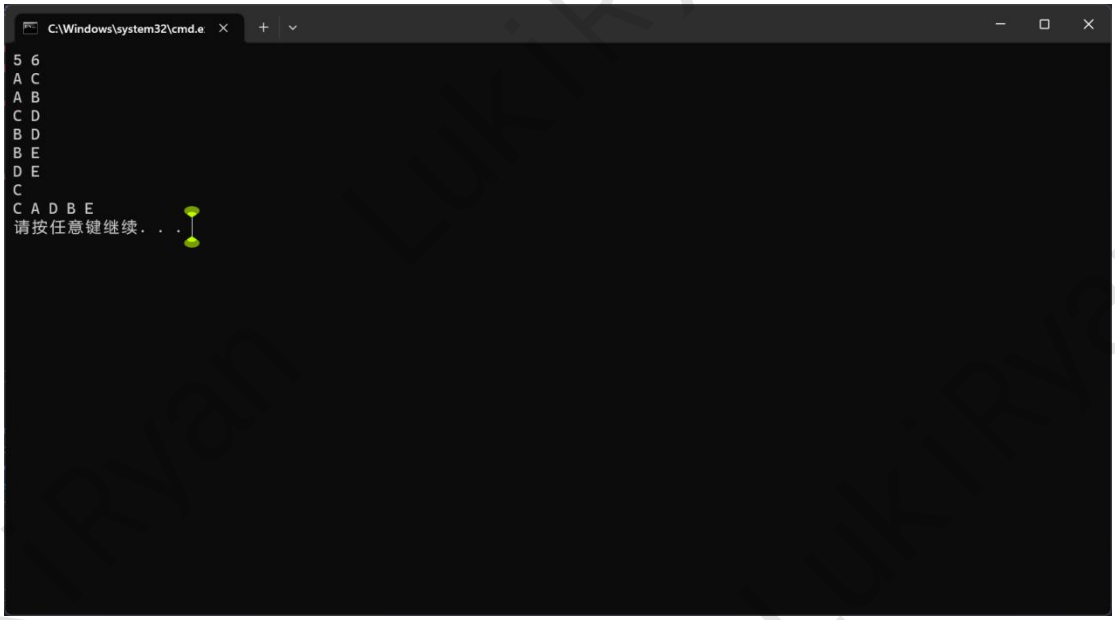

图 2：随机测试

Luki Ryan

二 DFS 遍历

1. 问题

以邻接表的存储方式，实现图的 BFS 和 DFS 遍历，并分析复杂度。（100 分）

输入：

第一行输入两个数 m, n，表示图有 m 个顶点（所有顶点的字母各不相同），n 条边；

接下来 n 行每行输入两个顶点，表示这两个顶点之间有边相连；

最后一行输入遍历开始的顶点

输出:

从遍历开始的顶点出发，分别输出图的 BFS 和 DFS 遍历的结果（若某个节点存在多种遍历方式，则按照字母表顺序来进行遍历，即输出只有一种结果）

例子：

输入：

3,3

A,B

A,C

B,C

A

输出：

A,B,C

A,B,C

LukiRyan

2. 代码

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#define N 510

int n, m;              // 节点数和边数
int g[N][N];           // 邻接矩阵表示图的连接关系
bool vis[N];           // 记录节点是否被访问过

// 深度优先搜索函数
void dfs(int x) {
    char s = 'A' + x - 1;   // 将节点编号转换为对应的字母表示
    printf("%c ", s);         // 输出当前节点的字母表示
    vis[x] = 1;                 // 标记当前节点为已访问
    for (int j = 1; j <= n; j++) {
        if (g[x][j] && vis[j] != 1) {
            dfs(j);             // 递归调用 dfs 函数继续遍历与当前节点相邻且未
访问过的节点
        }
    }
}

int main() {
    int a, b, c;
    char s1, s2, s3;
    scanf("%d%d", &n, &m);   // 输入节点数和边数
    for (int i = 1; i <= m; i++) {
        scanf(" %c %c", &s1, &s2);         // 输入边的两个节点
        a = s1 - 'A' + 1;                  // 将节点的字母表示转换为对应的编号
        b = s2 - 'A' + 1;
        g[a][b] = g[b][a] = 1;             // 在邻接矩阵中标记边的存在
    }
    scanf(" %c", &s3);                       // 输入遍历起点的节点字母
    c = s3 - 'A' + 1;                      // 将起点字母转换为编号
    dfs(c);                                  // 调用深度优先搜索函数进行遍历
    return 0;
}
//时间复杂度为 O(n^2)，空间复杂度为 O(n)。
```

3. 分析

这段代码中 DFS 算法的流程如下：

1、首先，在 main 函数中，从输入中获取起始节点 c。
2、接下来，调用 dfs(c) 函数，开始 DFS 遍历。
3、在 dfs 函数中，首先将当前节点 x 标记为已访问，即 vis[x] = 1。
4、然后，根据当前节点 x 计算对应的字母 s，表示字母表中的相应字母。
5、使用 printf 函数打印字符 s，表示正在访问的节点。
6、接下来，通过一个循环遍历当前节点 x 的邻居节点。
7、对于每个邻居节点 j，如果它未被访问过（即 vis[j] != 1），则递归调用 dfs(j)，以访问该邻居节点。
8、递归调用的过程中，会继续深入访问当前节点 x 的邻居节点的邻居，形成一个深度优先的搜索路径。
9、当没有未访问的邻居节点时，返回到上一级递归调用的节点，继续遍历其他未访问的邻居节点。
10、这个过程会一直持续下去，直到所有节点都被访问过。
11、最终，当所有的节点都被访问过后，DFS 遍历结束。

在整个 DFS 遍历的过程中，通过递归调用 dfs 函数，可以实现从起始节点开始，沿着一个路径尽可能深入地访问未访问过的节点。当无法继续深入时，回溯到上一个节点，继续遍历其他未访问的邻居节点。这样就能够遍历整个图的节点，并按照深度优先的顺序打印出节点访问的顺序。

这段代码是基于邻接矩阵表示的无向图进行 DFS 遍历，其中 g[N][N] 是用于存储图的邻接矩阵。在每一次递归调用 dfs 函数时，根据邻接矩阵确定当前节点 x 的邻居节点，并通过 g[x][j] 的值判断两个节点之间是否存在边。

复杂度分析：

时间复杂度分析：

预处理部分和主函数中的输入操作的时间复杂度可以忽略不计。
dfs() 函数的时间复杂度取决于图的结构。在最坏情况下，每个节点都需要被访问一次，每次访问需要遍历所有的节点。因此，时间复杂度为 O(n^2)，其中 n 是节点数。
所以，总体的时间复杂度为 O(n^2)。

空间复杂度分析：

全局变量 g[N][N] 和 vis[N] 占用的空间是固定的，与输入规模无关，因此空间复杂度为 O(1)。

dfs() 函数的递归调用会占用一定的栈空间，其最大深度取决于图的结构。在最坏情况下，栈空间的最大深度为 n。因此，空间复杂度为 O(n)。

所以，总体的空间复杂度为 O(n)。

LukiRyan

4. 测试

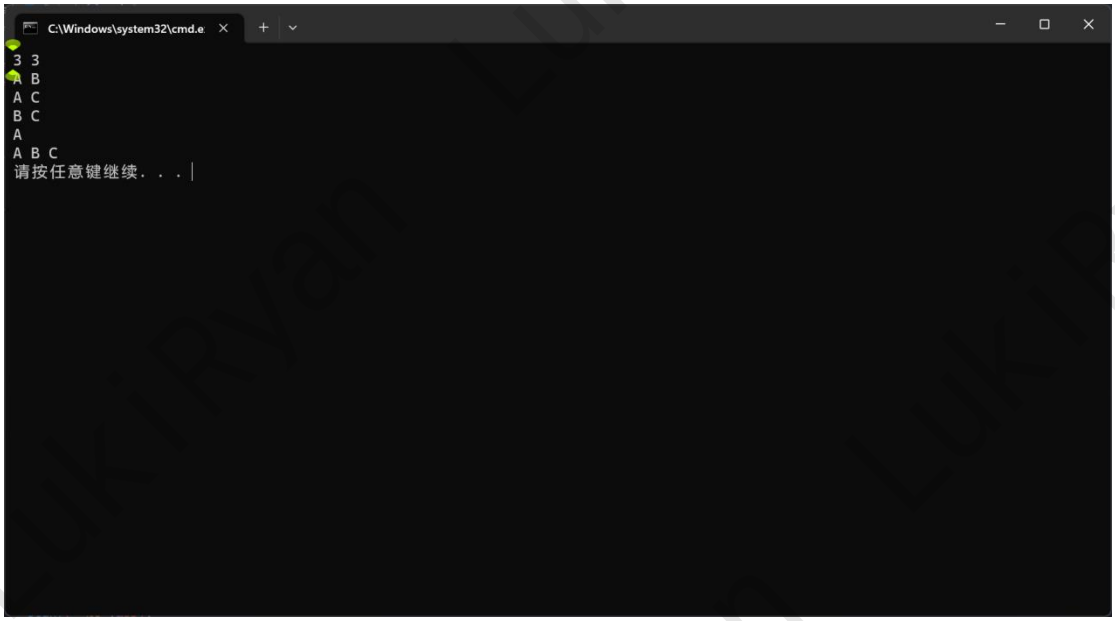

图 3：按样例测试

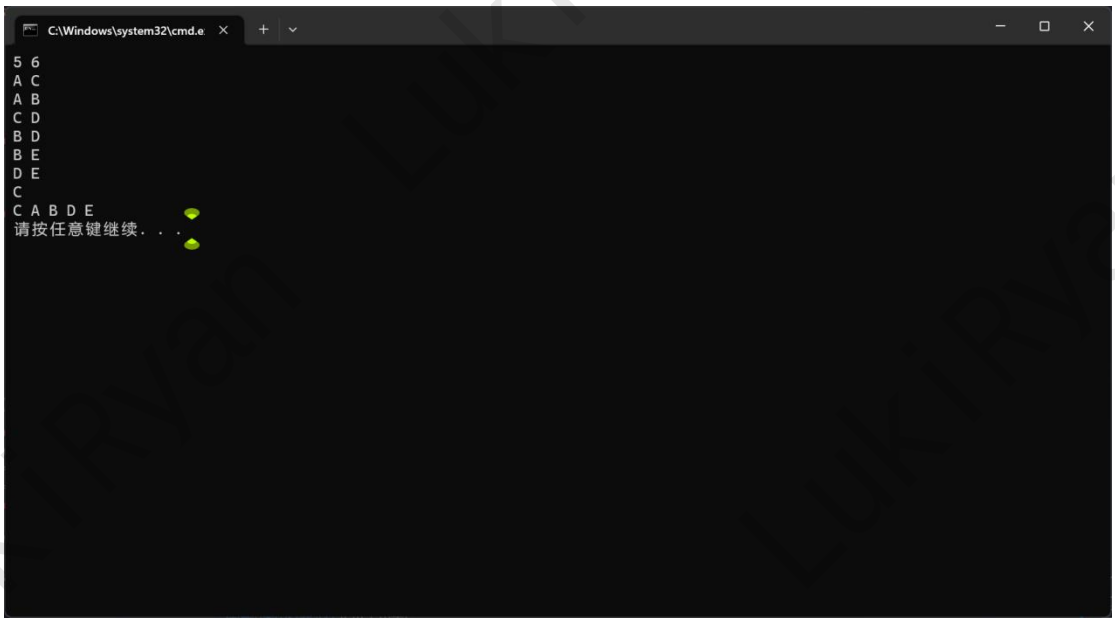

图 4：随机进行测试