数据结构实验报告——实验八（2）

学号：__20201060330__姓名：___胡诚皓___得分：_____

一、实验目的

1. 复习图的逻辑结构、存储结构及基本操作；

2. 掌握邻接矩阵、邻接表及图的创建、遍历；

3. 了解图的应用。

二、实验内容

1. （必做题）图的基本操作

假设图中数据元素类型是字符型，请采用邻接矩阵实现图的以下基本操作：

(1) 构造图（包括有向图、有向网、无向图、无向网）；

(2) 根据深度优先遍历图；

(3) 根据广度优先遍历图。

2. （选做题）无向图双色填涂

给定一个无向图及两种颜色，请判定能否为这个无向图的相邻顶点着不同颜色。例如，对于有 4 个顶点（1、2、3、4）及 3 条边｛(1,2)、(1,3)、(2,4)｝的无向图，可以为相邻顶点着不同颜色；对于有 4 个顶点（1、2、3、4）及 4 条边｛(1,2)、(1,3)、(1,4)、(2,4)｝的无向图，不可以为相邻顶点着不同颜色。

三、数据结构及算法描述

1. （必做题）图的基本操作

数据结构

定义了枚举类型 GraphKind，可选值为 DG、DN、UDG、UDN，分别对应有向图、有向网、无向图、无向网。宏定义 MAX_VERTEX_NUM 为 20，作为最大支持的顶点个数。

此处不妨令顶点本身的类型 VertexType 为 int。以 char 为表示边信息的类型，可以为顶点之间的关系附加一些信息。为了更加直观的展示，在以下代码中，对于边，以“起点->终点”展示；对于弧，以“起点--权值-->终点”展示。

AcrNode 作为邻接表中每个普通元素的类型，包括了记录终点信息的 adjvex、指向下一条边或弧的指针 nextarc 以及指向边或弧相关信息的指针 info。

VNode 作为邻接表中头结点的类型，包含了自身的顶点信息 data 以及指向第一条以该顶点为起点的边或弧的指针。额外使用 typedef 定义了用于邻接表类型中的头顶点数组 AdjList。

ALGraph 为邻接表本身的类型，包括 AdjList 类型的头结点数组 vertices，代表顶点数、边或弧数 vexnum 和 arcnum，存储图具体类型的 kind。

另外，使用了和前几次相同的链队列，将队列相关的基本操作放在 queue.c 中，以 queue.h 为其头文件。

算法描述

int main()

定义 ALGraph 类型的变量 graph 用于存储要输入的图，code 为临时变量，用于临时存储图的类型。在读入用户要输入的图的类型后，调用对应的构建函数构建图，再分别调用 deepTraverseMap 和 breadthTraverseMap 输出图的深度优先遍历和广度优先遍历。

Boolean constructDG(ALGraph *graph)

此函数用于根据用户输入构建有向图。先读入图的顶点数和弧数，再按照“起点 终点”的格式读取有向图的各条弧，并且用头插法插入到邻接表的链表中。在输入弧的过程中，若起点或终点的顶点编号超过了之前输入的顶点数，将会要求用户重新输入。

在图构建完成后，会输出整个邻接矩阵供预览。每一行表示邻接表中的一个结点，冒号后是以当前结点为起点的各个边的信息。

Boolean constructDN(ALGraph *graph)

此函数用于根据用户输入构建有向网。先读入网的顶点数和弧数，再按照“起点 终点 权值”的格式读取有向网的各条弧，并且用头插法插入到邻接表的链表中。在输入弧的过程中，若起点或终点的顶点编号超过了之前输入的顶点数，将会要求用户重新输入。

在图构建完成后，会输出整个邻接矩阵供预览。每一行表示邻接表中的一个结点，冒号后是以当前结点为起点的各个边的信息。

Boolean constructUDG(ALGraph *graph)

此函数用于根据用户输入构建无向图。先读入图的顶点数和边数，再按照“起点 终点”的格式读取无向图的各条边，并且用头插法插入到邻接表的链表中。在输入边的过程中，若起点或终点的顶点编号超过了之前输入的顶点数，将会要求用户重新输入。

在图构建完成后，会输出整个邻接矩阵供预览。每一行表示邻接表中的一个结点，冒号后是以当前结点为起点的各个边的信息。

Boolean constructUDN(ALGraph *graph)

此函数用于根据用户输入构建无向网。先读入网的顶点数和边数，再按照“起点 终点 权值”的格式读取无向网的各条边，并且用头插法插入到邻接表的链表中。在输入边的过程中，若起点或终点的顶点编号超过了之前输入的顶点数，将会要求用户重新输入。

在图构建完成后，会输出整个邻接矩阵供预览。每一行表示邻接表中的一个结点，冒号后是以当前结点为起点的各个边的信息。

void dfs(Boolean visited[], ALGraph graph, int index)
作为深度优先遍历的递归函数，按以下步骤执行
①访问当前顶点 index，即将当前顶点在 visited 中标记为 TRUE，并输出当前顶点的编号。同时也初始化 pt 指向当前顶点 index 在邻接表中的第一条邻接边。
②使用 while 循环遍历 index 的所有邻接边，对于邻接边对应的未访问终点进行递归访问

void deepTraverseMap(ALGraph graph)
对 graph 进行深度优先遍历，先声明一个初始值均为 0（即 FALSE）的 visited 数组来记录各顶点是否被访问过，用 for 循环来保证每个顶点都被遍历到（在图非连通的情况下）。在 for 循环中调用 dfs 递归访问顶点

void breadthTraverseMap(ALGraph graph)
对 graph 进行广度优先遍历，同样先声明一个初始值均为 0（即 FALSE）的 visited 数组来记录各顶点是否被访问过。声明局部变量 cur 来存储当前访问的顶点编号以及 queue 来实现广度优先遍历，按以下步骤执行：
①将下标为 0 的顶点入队
②只要队列不为空，就出队一个顶点，保存在 cur 中。若 cur 已经访问过，就不再访问；若没有访问过，则访问之，即在 visited 中标注并输出
③将 pt 指向当前顶点 index 在邻接表中的第一条邻接边，使用 while 循环将当前顶点 index 的各个未访问邻接点入队
④若此时队列为空，用 for 循环找是否有未访问过的顶点，将未访问过的顶点入队（处理非连通图）。转到②

2. （选做题）无向图双色填涂

数据结构

无向图的数据结构与第一题中相同，同样使用 constructUDG 来构建无向图。与第一题中不同的是，在头结点类型的 VNode 中添加了两个域 color 和 colorStatus，分别表示当前结点填的颜色以及当前结点的邻接点的填色情况。
此处使用了二进制的思想，宏定义颜色 A 为 1（即$(01)_2$）、颜色 B 为 2（即$(10)_2$）。每个头结点的 colorStatus 域有如下四种情况：
• $(00)_2$（即十进制下的 0）：表示其邻接点目前都没有填色，可以任填一种颜色，代码中填了颜色 A
• $(01)_2$（即十进制下的 1）：表示其邻接点中只有填颜色 A 的，当前结点只能填颜色 B
• $(10)_2$（即十进制下的 2）：表示其邻接点中只有填颜色 B 的，当前结点只能填颜色 A
• $(11)_2$（即十进制下的 3）：表示其邻接点中既有填颜色 A 的，也有填颜色 B 的，当前结点无法填色

算法描述

void initALGraph(ALGraph *graph)
使用 for 循环将邻接表的头结点初始化，将结点的数据域直接设为与下标一致。color、colorStatus、firstarc 都置零

int main()
①声明存储图的变量 graph 并调用 initALGraph 进行初始化
②调用 constructUDG 构建无向图，并调用 bfsFilling 对图进行填色，用 res 接受填色的结果
③根据 res 的值判断是否成功填色，若填色成功，输出各个结点的填色情况

void printColor(ALGraph graph)
根据每个头结点的 color 域输出填色情况

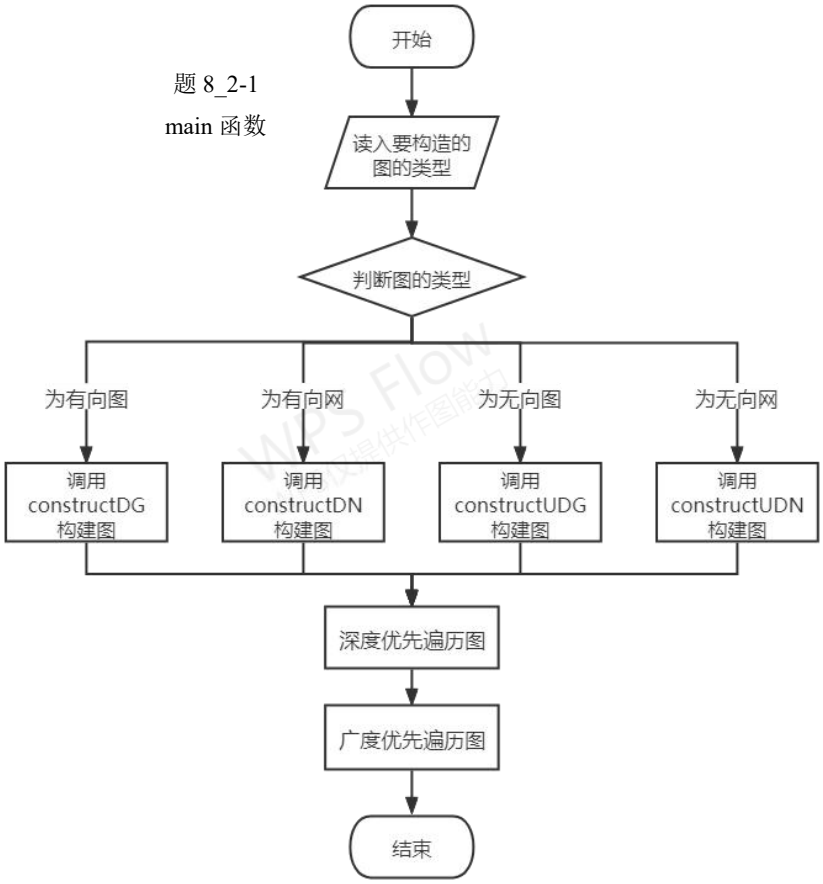
Boolean bfsFilling(ALGraph *graph)
使用广度优先遍历进行填色
①将下标为 0 的顶点入队
②只要队列不为空，就出队一个顶点，保存在 cur 中。若 cur 已经填色过，就不再对其进行操作；若没有填色过，则根据当前结点的 colorStatus 域按照上述“数据结构”中的规则进行填色
③根据邻接表，将当前结点未填色的邻接点都入队
④若此时队列为空，用 for 循环找是否有未填色过的顶点，将最先循环到的未填色顶点入队（处理非连通图），转到②

四、详细设计

1. （必做题）图的基本操作

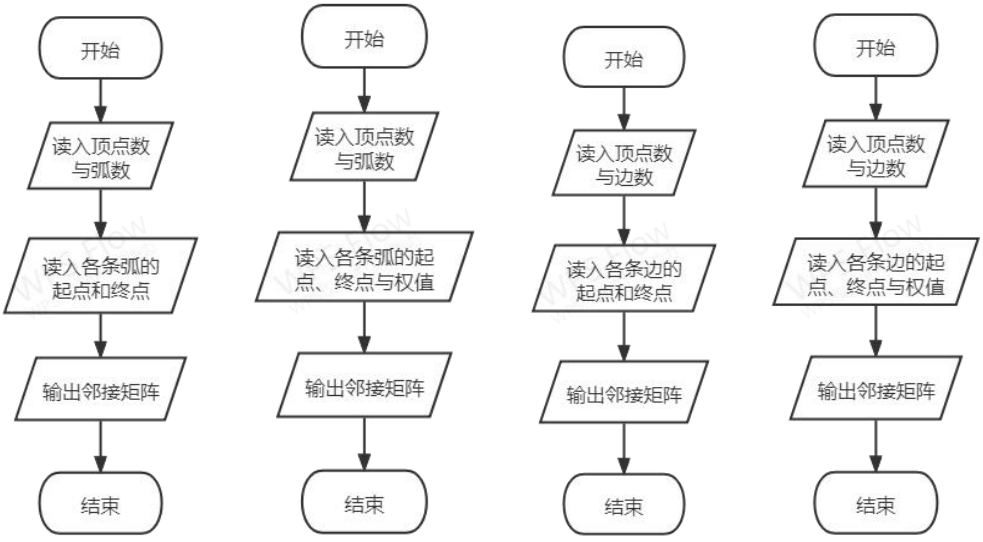
题 8_2-1

main 函数



题 8_2-1

constructDG、constructDN、constructUDG、constructUDN 函数

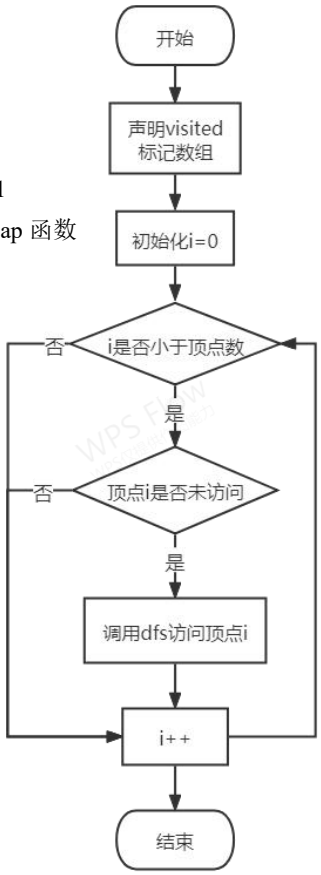


题 8_2-1

dfs 函数

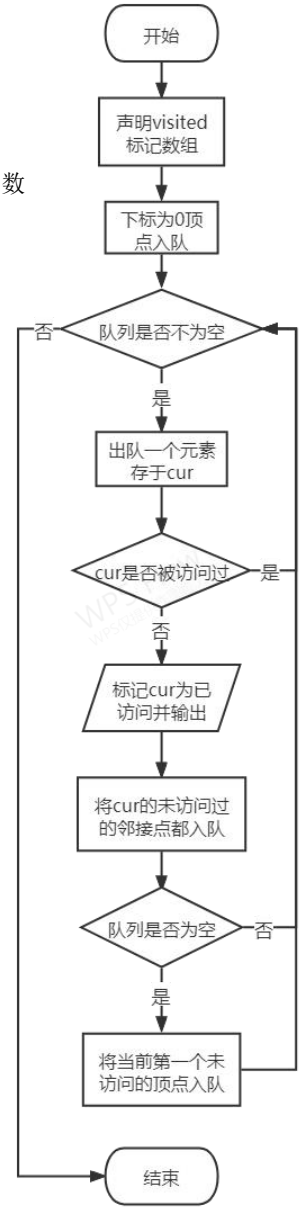
题 8_2-1

deepTraverseMap 函数



题 8_2-1

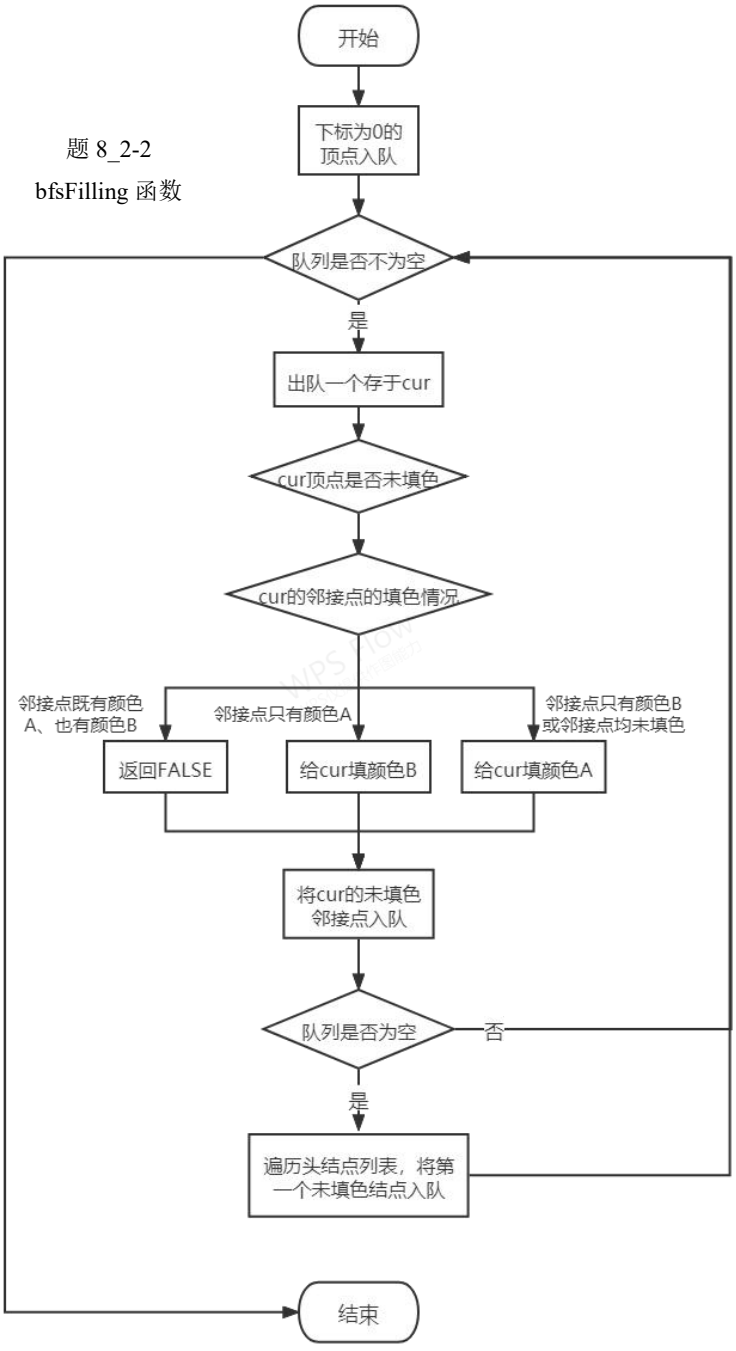
breadthTraverseMap 函数



2. （选做题）无向图双色填涂

题 8_2-2

bfsFilling 函数



五、程序代码

1. （必做题）图的基本操作

8_2-1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
```

```
#define MAX_VERTEX_NUM 20
#define TRUE 1
#define FALSE 0

typedef int Boolean;
//分别为有向图、有向网、无向图、无向网
typedef enum {
    DG, DN, UDG, UDN
} GraphKind;
//表示边信息的类型
typedef char InfoType;

//图/网中的顶点类型
typedef int VertexType;

//邻接表中普通元素的类型
typedef struct ArcNode {
    int adjvex;//边或弧所依附的顶点的位置，即边或弧的终点
    struct ArcNode *nextarc;//指向下一条依附顶点的边或弧的指针
    InfoType *info;//边或弧的相关信息指针
} ArcNode;

//邻接表中头结点类型
typedef struct {
    VertexType data;//顶点信息
    ArcNode *firstarc;//指向第一条依附本顶点的边或弧的指针
} VNode, AdjList[MAX_VERTEX_NUM];

//邻接表本身的类型
typedef struct {
    AdjList vertices;//图的头结点数组
    int vexnum, arcnum;//图的顶点数和边或弧数
    int kind;//图的类型
} ALGraph;

void initALGraph(ALGraph *graph) {
    for (int i = 0; i < MAX_VERTEX_NUM; i++) {
        graph->vertices[i].firstarc = NULL;
        graph->vertices[i].data = i;
    }
}

/* 构建有向图 */
Boolean constructDG(ALGraph *graph);
```

```
/* 构建有向网 */
Boolean constructDN(ALGraph *graph);
/* 构建无向图 */
Boolean constructUDG(ALGraph *graph);
/* 构建无向网 */
Boolean constructUDN(ALGraph *graph);
/* 深度优先遍历图 */
void deepTraverseMap(ALGraph graph);
/* 广度优先遍历图 */
void breadthTraverseMap(ALGraph graph);


int main() {
    ALGraph graph;
    initALGraph(&graph);
    int code;

    //读入要构造的图的类型
    while (1) {
        printf("-------------\n");
        printf("0:有向图\n1:有向网\n2:无向图\n3:无向网\n-1:退出\n 请选择要构造的类型\n");
        scanf("%d", &code);
        if (code == -1) {
            system("pause");
            return 0;
        } else if (code < -1 || code > 3) {
            printf("输入错误，请重新输入\n");
        } else {
            graph.kind = code;
            break;
        }
    }
    //根据不同图类型调用相应的构造函数
    if (graph.kind == DG) {
        constructDG(&graph);
    } else if (graph.kind == DN) {
        constructDN(&graph);
    } else if (graph.kind == UDG) {
        constructUDG(&graph);
    } else if (graph.kind == UDN) {
        constructUDN(&graph);
    }

    deepTraverseMap(graph);//深度优先遍历图 graph
```

```
    breadthTraverseMap(graph);//广度优先遍历图 graph

    printf("\n");

    system("pause");
    return 0;
}

Boolean constructDG(ALGraph *graph) {
    int start, end;
    ArcNode *newNode;
    ArcNode *pt;
    printf("-----------------\n");
    printf("开始构造有向图\n");
    printf("请输入顶点数（不能超过 20）：\n");
    scanf("%d", &graph->vexnum);
    printf("请输入弧数：\n");
    scanf("%d", &graph->arcnum);
    printf("请输入各条弧的起点和终点（起点终点之间以空格隔开）：\n");
    //读入各条弧
    for (int i = 0; i < graph->arcnum; i++) {
        scanf("%d %d", &start, &end);
        if (start >= graph->vexnum || start < 0 ||
            end >= graph->vexnum || end < 0) {
            printf("该条弧输入错误，请重新输入当前弧：\n");
            i--;
            continue;
        }
        //使用头插法将边插入到邻接表中
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        newNode->adjvex = end;
        newNode->info = (char *) malloc(sizeof(char)*15);
        sprintf(newNode->info, "%d->%d", start, end);
        newNode->nextarc = graph->vertices[start].firstarc;
        graph->vertices[start].firstarc = newNode;
    }
    printf("--------邻接表预览--------\n");
    for (int i = 0; i < graph->vexnum; i++) {
        printf("%d：", graph->vertices[i].data);
        pt = graph->vertices[i].firstarc;
        while (pt != NULL) {
            printf("%s ", pt->info);
            pt = pt->nextarc;
        }
```

```
        printf("\n");
    }

    return TRUE;
}


Boolean constructDN(ALGraph *graph) {
    int start, end, weight;
    ArcNode *newNode;
    ArcNode *pt;
    printf("-----------------\n");
    printf("开始构造有向网\n");
    printf("请输入顶点数（不能超过 20）：\n");
    scanf("%d", &graph->vexnum);
    printf("请输入弧数：\n");
    scanf("%d", &graph->arcnum);
    printf("请输入各条弧的起点、终点与权值（以空格隔开）：\n");
    //读入各条弧
    for (int i = 0; i < graph->arcnum; i++) {
        scanf("%d %d %d", &start, &end, &weight);
        if (start >= graph->vexnum || start < 0 ||
            end >= graph->vexnum || end < 0) {
            printf("该条弧输入错误，请重新输入当前弧：\n");
            i--;
            continue;
        }
        //使用头插法将边插入到邻接表中
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        newNode->adjvex = end;
        newNode->info = (char *) malloc(sizeof(char)*50);
        sprintf(newNode->info, "%d--%d-->%d", start, weight, end);
        newNode->nextarc = graph->vertices[start].firstarc;
        graph->vertices[start].firstarc = newNode;
    }
    printf("--------邻接表预览--------\n");
    for (int i = 0; i < graph->vexnum; i++) {
        printf("%d：", graph->vertices[i].data);
        pt = graph->vertices[i].firstarc;
        while (pt != NULL) {
            printf("%s  ", pt->info);
            pt = pt->nextarc;
        }
        printf("\n");
```

```
    }

    return TRUE;
}

Boolean constructUDG(ALGraph *graph) {
    int start, end;
    ArcNode *newNode;
    ArcNode *pt;
    printf("-----------------\n");
    printf("开始构造无向图\n");
    printf("请输入顶点数（不能超过 20）：\n");
    scanf("%d", &graph->vexnum);
    printf("请输入边数：\n");
    scanf("%d", &graph->arcnum);
    printf("请输入各条边的起点和终点（起点终点之间以空格隔开）：\n");
    //读入各条边
    for (int i = 0; i < graph->arcnum; i++) {
        scanf("%d %d", &start, &end);
        if (start >= graph->vexnum || start < 0 ||
            end >= graph->vexnum || end < 0) {
            printf("该条边输入错误，请重新输入当前弧：\n");
            i--;
            continue;
        }
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        //无向图，使用头插法将边插入到邻接表中
        newNode->adjvex = end;
        newNode->info = (char *) malloc(sizeof(char)*15);
        sprintf(newNode->info, "%d->%d", start, end);
        newNode->nextarc = graph->vertices[start].firstarc;
        graph->vertices[start].firstarc = newNode;
        //另一方向
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        newNode->adjvex = start;
        newNode->info = (char *) malloc(sizeof(char)*15);
        sprintf(newNode->info, "%d->%d", end, start);
        newNode->nextarc = graph->vertices[end].firstarc;
        graph->vertices[end].firstarc = newNode;
    }
    printf("--------邻接表预览--------\n");
    for (int i = 0; i < graph->vexnum; i++) {
        printf("%d: ", graph->vertices[i].data);
        pt = graph->vertices[i].firstarc;
```

```
        while (pt != NULL) {
            printf("%s  ", pt->info);
            pt = pt->nextarc;
        }
        printf("\n");
    }

    return TRUE;
}

Boolean constructUDN(ALGraph *graph) {
    int start, end, weight;
    ArcNode *newNode;
    ArcNode *pt;
    printf("-----------------\n");
    printf("开始构造无向网\n");
    printf("请输入顶点数（不能超过 20）：\n");
    scanf("%d", &graph->vexnum);
    printf("请输入边数：\n");
    scanf("%d", &graph->arcnum);
    printf("请输入各条边的起点、终点与权值（以空格隔开）：\n");
    //读入各条边
    for (int i = 0; i < graph->arcnum; i++) {
        scanf("%d %d %d", &start, &end, &weight);
        if (start >= graph->vexnum || start < 0 ||
            end >= graph->vexnum || end < 0) {
            printf("该条边输入错误，请重新输入当前弧：\n");
            i--;
            continue;
        }
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        //无向图，使用头插法将边插入到邻接表中
        newNode->adjvex = end;
        newNode->info = (char *) malloc(sizeof(char)*50);
        sprintf(newNode->info, "%d--%d-->%d", start, weight, end);
        newNode->nextarc = graph->vertices[start].firstarc;
        graph->vertices[start].firstarc = newNode;
        //另一方向
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        newNode->adjvex = start;
        newNode->info = (char *) malloc(sizeof(char)*50);
        sprintf(newNode->info, "%d--%d-->%d", end, weight, start);
        newNode->nextarc = graph->vertices[end].firstarc;
        graph->vertices[end].firstarc = newNode;
```

```
    }

    printf("--------邻接表预览--------\n");
    for (int i = 0; i < graph->vexnum; i++) {
        printf("%d: ", graph->vertices[i].data);
        pt = graph->vertices[i].firstarc;
        while (pt != NULL) {
            printf("%s  ", pt->info);
            pt = pt->nextarc;
        }
        printf("\n");
    }

    return TRUE;
}

void dfs(Boolean visited[], ALGraph graph, int index) {
    ArcNode *pt=graph.vertices[index].firstarc;
    //访问当前顶点
    visited[index] = TRUE;
    printf("%d ", index);
    //循环 index 结点的所有邻接点
    while (pt != NULL) {
        //找 index 的第一个未访问的邻接点
        while (pt != NULL && visited[pt->adjvex] == TRUE)
            pt = pt->nextarc;
        //找到则访问
        if (pt != NULL)
            dfs(visited, graph, pt->adjvex);
    }
}

void deepTraverseMap(ALGraph graph) {
    Boolean visited[MAX_VERTEX_NUM]={0,};
    //使遍历同时适用于非连通图
    printf("-------------\n 深度优先遍历: ");
    for (int i = 0; i < graph.vexnum; i++) {
        if (visited[i] == FALSE)
            dfs(visited, graph, i);
    }
    printf("\n");
}

void breadthTraverseMap(ALGraph graph) {
```

```
    Boolean visited[MAX_VERTEX_NUM]={0,};
    ArcNode *pt;
    QElemType cur;
    LinkQueue queue;
    Initqueue(&queue);
    printf("-------------\n广度优先遍历：");

    //从下标为0的顶点开始，将其入队
    Enqueue(&queue, 0);
    while (!Emptyqueue(queue)) {
        Dequeue(&queue, &cur);
        //访问过就不再访问，未访问过就访问
        if (!visited[cur]) {
            visited[cur] = TRUE;
            printf("%d ", cur);
            pt = graph.vertices[cur].firstarc;
            //访问cur的所有邻接点
            while (pt != NULL) {
                //将当前顶点的未访问邻接点入队
                if (visited[pt->adjvex] == FALSE) {
                    Enqueue(&queue, pt->adjvex);
                }
                pt = pt->nextarc;
            }
        }
        //处理非连通图
        if (Emptyqueue(queue)) {
            for (int i = 0; i < graph.vexnum; i++) {
                if (!visited[i]) {
                    Enqueue(&queue, i);
                    break;
                }
            }
        }
    }
    printf("\n");

}
```

queue.c

```
#include "queue.h"
#include <stdlib.h>
```

```
Status Emptyqueue(LinkQueue q) {
    if (q.head->next == NULL)
        return OK;
    return ERROR;
}

Status Enqueue(queuePtr q, QElemType elem) {
    QNode* tmp = (QNode *) malloc(sizeof(QNode));
    if (tmp == NULL)
        return ERROR;
    tmp->pt = elem;
    tmp->next = NULL;

    q->rear->next = tmp;
    q->rear = tmp;
    return OK;
}

Status Dequeue(queuePtr q, QElemType *out) {
    if (Emptyqueue(*q) == OK)
        return ERROR;
    QNode *tmp=q->head->next;
    *out = q->head->next->pt;
    q->head->next = tmp->next;
    //由于队列是有头结点的，对出队后变为空队列的情况做特殊处理
    if (q->head->next == NULL)
        q->rear = q->head;
    free(tmp);

    return OK;
}

Status Initqueue(queuePtr q) {
    if (q == NULL)
        return ERROR;
    q->rear = q->head = (QNode *) malloc(sizeof(QNode));
    if (q->rear == NULL)
        return ERROR;
    q->head->next = NULL;

    return OK;
}
```

queue.h

```
#ifndef UNTITLED3_QUEUE_H
#define UNTITLED3_QUEUE_H

#define Status int
#define OK 1
#define ERROR 0
typedef int QElemType;

typedef struct QNode{
    QElemType pt;
    struct QNode *next;
} QNode;

typedef struct LinkQueue {
    QNode *head, *rear;
} LinkQueue, *queuePtr;

/* 队列基本操作 */
Status Enqueue(queuePtr, QElemType);
Status Dequeue(queuePtr, QElemType *);
Status Emptyqueue(LinkQueue q);
Status Initqueue(queuePtr q);

#endif //UNTITLED3_QUEUE_H
```

2. （选做题）无向图双色填涂

8_2-2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

#define MAX_VERTEX_NUM 20
#define TRUE 1
#define FALSE 0
#define colorA 1
#define colorB 2

typedef int Boolean;
//分别为有向图、有向网、无向图、无向网
```

```
typedef enum {
    DG, DN, UDG, UDN
} GraphKind;
//表示顶点关系的类型
typedef int VRType;
//表示边信息的类型
typedef char InfoType;

//图/网中的顶点类型
typedef int VertexType;

//邻接表中普通元素的类型
typedef struct ArcNode {
    int adjvex;//边或弧所依附的顶点的位置，即边或弧的终点
    struct ArcNode *nextarc;//指向下一条依附顶点的边或弧的指针
    InfoType *info;//边或弧的相关信息指针
} ArcNode;

//邻接表中头结点类型
typedef struct {
    VertexType data;//顶点信息
    ArcNode *firstarc;//指向第一条依附本顶点的边或弧的指针
    int color, colorStatus;//color 为当前结点的颜色，colorStatus 为当前结点邻接点的填色情
况
} VNode, AdjList[MAX_VERTEX_NUM];

//邻接表本身的类型
typedef struct {
    AdjList vertices;//图的头结点数组
    int vexnum, arcnum;//图的顶点数和边或弧数
} ALGraph;

void initALGraph(ALGraph *graph) {
    for (int i = 0; i < MAX_VERTEX_NUM; i++) {
        graph->vertices[i].color = 0;
        graph->vertices[i].colorStatus = 0;
        graph->vertices[i].firstarc = NULL;
        graph->vertices[i].data = i;
    }
}

/* 构建无向图 */
Boolean constructUDG(ALGraph *graph);
/* 广度优先给图填色 */
```

```
Boolean bfsFilling(ALGraph *graph);
/* 输出填色情况 */
void printColor(ALGraph graph);

int main() {
    ALGraph graph;
    Boolean res;
    initALGraph(&graph);

    printf("构建无向图\n");
    constructUDG(&graph);

    res = bfsFilling(&graph);
    if (res == FALSE) {
        printf("该图无法填色");
    } else {
        printf("------------------------\n");
        printf("填色成功，填色情况如下\n");
        printColor(graph);
    }

    printf("\n");

    system("pause");
    return 0;
}

void printColor(ALGraph graph) {
    for (int i = 0; i < graph.vexnum; i++) {
        if (graph.vertices[i].color == colorA)
            printf("%d：colorA\n", i);
        else if(graph.vertices[i].color == colorB)
            printf("%d：colorB\n", i);
        else
            printf("%d：未填色\n", i);
    }
}

Boolean constructUDG(ALGraph *graph) {
    int start, end;
    ArcNode *newNode;
    ArcNode *pt;
    printf("-----------------\n");
    printf("开始构造无向图\n");
```

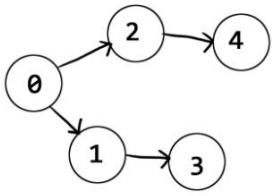
```
    printf("请输入顶点数（不能超过 20）：\n");
    scanf("%d", &graph->vexnum);
    printf("请输入边数：\n");
    scanf("%d", &graph->arcnum);
    printf("请输入各条边的起点和终点（起点终点之间以空格隔开）：\n");
    //读入各条边
    for (int i = 0; i < graph->arcnum; i++) {
        scanf("%d %d", &start, &end);
        if (start >= graph->vexnum || start < 0 ||
            end >= graph->vexnum || end < 0) {
            printf("该条边输入错误，请重新输入当前弧：\n");
            i--;
            continue;
        }
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        //无向图，头插法将边插入邻接表
        newNode->adjvex = end;
        newNode->info = (char *) malloc(sizeof(char)*15);
        sprintf(newNode->info, "%d->%d", start, end);
        newNode->nextarc = graph->vertices[start].firstarc;
        graph->vertices[start].firstarc = newNode;
        //另一方向
        newNode = (ArcNode *) malloc(sizeof(ArcNode));
        newNode->adjvex = start;
        newNode->info = (char *) malloc(sizeof(char)*15);
        sprintf(newNode->info, "%d->%d", end, start);
        newNode->nextarc = graph->vertices[end].firstarc;
        graph->vertices[end].firstarc = newNode;
    }
    printf("--------邻接表预览--------\n");
    for (int i = 0; i < graph->vexnum; i++) {
        printf("%d: ", graph->vertices[i].data);
        pt = graph->vertices[i].firstarc;
        while (pt != NULL) {
            printf("%s  ", pt->info);
            pt = pt->nextarc;
        }
        printf("\n");
    }

    return TRUE;
}

Boolean bfsFilling(ALGraph *graph) {
```

```
    ArcNode *pt;
    QElemType cur;
    LinkQueue queue;
    Initqueue(&queue);

    //从下标为0的顶点开始，将其入队
    Enqueue(&queue, 0);
    while (!Emptyqueue(queue)) {
        //出队一个元素
        Dequeue(&queue, &cur);
        //未填色过则根据情况填色
        if (graph->vertices[cur].color == 0) {
            //cur的邻接点中已经填有colorA和colorB，说明无法填色
            if (graph->vertices[cur].colorStatus == 3) {
                return FALSE;
            } else if ((graph->vertices[cur].colorStatus & colorA) == colorA) {//cur
邻接点中已经填有colorA，则填colorB并更新所有邻接点
                graph->vertices[cur].color = colorB;
                pt = graph->vertices[cur].firstarc;
                while (pt != NULL) {
                    graph->vertices[pt->adjvex].colorStatus |= colorB;
                    pt = pt->nextarc;
                }
            } else if ((graph->vertices[cur].color & colorB) == colorB) {//cur邻接点中
已经填有colorB，则天colorA并更新所有邻接点
                graph->vertices[cur].color = colorA;
                pt = graph->vertices[cur].firstarc;
                while (pt != NULL) {
                    graph->vertices[pt->adjvex].colorStatus |= colorA;
                    pt = pt->nextarc;
                }
            } else {//cur的邻接点中还没有填色过，就填colorA
                graph->vertices[cur].color = colorA;
                pt = graph->vertices[cur].firstarc;
                while (pt != NULL) {
                    graph->vertices[pt->adjvex].colorStatus |= colorA;
                    pt = pt->nextarc;
                }
            }
            pt = graph->vertices[cur].firstarc;
            //访问cur的所有邻接点
            while (pt != NULL) {
                //将当前顶点的未填色邻接点入队
                if (graph->vertices[pt->adjvex].color == 0) {
```

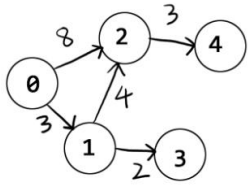
```
                    Enqueue(&queue, pt->adjvex);
                }
                pt = pt->nextarc;
            }
        }
        //处理非连通图
        if (Emptyqueue(queue)) {
            for (int i = 0; i < graph->vexnum; i++) {
                if (graph->vertices[i].color == 0) {
                    Enqueue(&queue, i);
                    break;
                }
            }
        }
    }
}
```

六、测试和结果

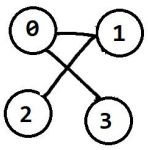
1. （必做题）图的基本操作



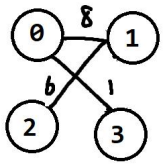

```
C:\Windows\System32\cmd.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-1.exe
-------------
0：有向图
1：有向网
2：无向图
3：无向网
-1：退出
请选择要构造的类型
0
-----------------
开始构造有向图
请输入顶点数（不能超过20）：
5
请输入弧数：
4
请输入各条弧的起点和终点（起点终点之间以空格隔开）：
0 2
0 1
2 4
1 3
--------邻接表预览--------
0：0->1 0->2
1：1->3
2：2->4
3：
4：
-------------
深度优先遍历：0 1 3 2 4
-------------
广度优先遍历：0 1 2 3 4

请按任意键继续. . .
```




```
C:\Windows\System32\cmd.exe - 8_2-1.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-1.exe
------------
0：有向图
1：有向网
2：无向图
3：无向网
-1：退出
请选择要构造的类型
1
-----------------
开始构造有向网
请输入顶点数（不能超过20）：
5
请输入弧数：
5
请输入各条弧的起点、终点与权值（以空格隔开）：
0 1 3
0 2 8
2 4 3
1 2 4
1 3 2
--------邻接表预览--------
0：0--8-->2  0--3-->1
1：1--2-->3  1--4-->2
2：2--3-->4
3：
4：
-------------
深度优先遍历：0 2 4 1 3
-------------
广度优先遍历：0 2 1 4 3

请按任意键继续. . .
```

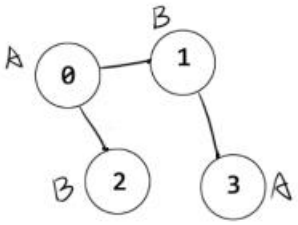



```
C:\Windows\System32\cmd.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-1.exe
------------
0：有向图
1：有向网
2：无向图
3：无向网
-1：退出
请选择要构造的类型
2
-----------------
开始构造无向图
请输入顶点数（不能超过20）：
4
请输入边数：
3
请输入各条边的起点和终点（起点终点之间以空格隔开）：
0 3
0 1
1 2
--------邻接表预览--------
0：0->1  0->3
1：1->2  1->0
2：2->1
3：3->0
-------------
深度优先遍历：0 1 2 3
-------------
广度优先遍历：0 1 3 2

请按任意键继续. . .

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>
```

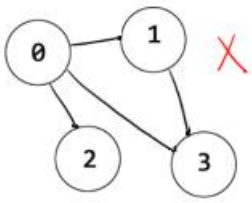



```
C:\Windows\System32\cmd.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-1.exe
-------------
0：有向图
1：有向网
2：无向图
3：无向网
-1：退出
请选择要构造的类型
3
-----------------
开始构造无向网
请输入顶点数（不能超过20）：
4
请输入边数：
3
请输入各条边的起点、终点与权值（以空格隔开）：
0 3 1
0 1 8
1 2 6
--------邻接表预览--------
0：0--8-->1  0--1-->3
1：1--6-->2  1--8-->0
2：2--6-->1
3：3--1-->0
-------------
深度优先遍历：0 1 2 3
-------------
广度优先遍历：0 1 3 2

请按任意键继续. . .

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>
```

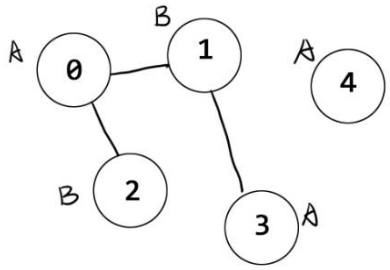
2. （选做题）无向图双色填涂



```
C:\Windows\System32\cmd.exe - 8_2-2.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-2.exe
构建无向图
-----------------
开始构造无向图
请输入顶点数（不能超过20）：
4
请输入边数：
3
请输入各条边的起点和终点（起点终点之间以空格隔开）：
0 1
0 2
1 3
--------邻接表预览--------
0：0->2  0->1
1：1->3  1->0
2：2->0
3：3->1
------------------------
填色成功，填色情况如下
0：colorA
1：colorB
2：colorB
3：colorA

请按任意键继续. . .
```




```
C:\Windows\System32\cmd.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>8_2-2.exe
构建无向图
-----------------
开始构造无向图
请输入顶点数（不能超过20）：
4
请输入边数：
4
请输入各条边的起点和终点（起点终点之间以空格隔开）：
0 1
0 2
0 3
1 3
--------邻接表预览--------
0：0->3  0->2  0->1
1：1->3  1->0
2：2->0
3：3->1  3->0
该图无法填色
请按任意键继续. . .

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 8_2>
```




```
C:\Windows\System32\cmd.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\d
ata-structure\experiment 8_2>8_2-2.exe
构建无向图
-----------------
开始构造无向图
请输入顶点数（不能超过20）：
5
请输入边数：
3
请输入各条边的起点和终点（起点终点之间以空格隔开）：
0 1
1 3
0 2
---------邻接表预览---------
0：0->2  0->1
1：1->3  1->0
2：2->0
3：3->1
4：
------------------------
填色成功，填色情况如下
0：colorA
1：colorB
2：colorB
3：colorA
4：colorA

请按任意键继续. . .
```

七、用户手册

1. （必做题）图的基本操作

图中边或弧的权值都以 int 存储，输入的值不能超过 int 的范围，调用的队列相关的函数都存在以 queue.h 为头文件的源代码 queue.c 中。此处假定各个顶点的索引值（下标）为其数据域的值，若要用其他数据域的值，建立从数据域到下标的散列表进行查询即可。

2. （选做题）无向图双色填涂

无向图的构建与第一题中相同。填色成功则输出各点填的颜色；填色失败则输出“该图无法填色”