로고, 등록 상표, 폰트, 상징이(가) 표시된 사진

자동 생성된 설명

**데이터구조2**

**과제#3\_1\_dfs(list)**

|  |  |
| --- | --- |
| 과목명 | 데이터구조2 |
| 담당교수 | 남윤영 |
| 학과 | 컴퓨터공학과 |
| 학년 | 2 |
| 학번 | 20204101 |
| 이름 | 신동욱 |
| 제출일 | 09.26 |

**1. 소스코드**

**2. 실행 과정 분석**

**3. 실행 결과**

**1. 소스코드**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_VERTICES 50

#define TRUE 1

#define FALSE 0

int visted\_mat[MAX\_VERTICES];

int visted\_list[MAX\_VERTICES];

typedef struct {

    int n;

    int adj\_mat[MAX\_VERTICES][MAX\_VERTICES];

} GraphType\_mat;

void init\_mat(GraphType\_mat\* g) {

    int r, c;

    g->n = 0;

    for (r = 0; r < MAX\_VERTICES; r++) {

        for (c = 0; c < MAX\_VERTICES; c++) {

            g->adj\_mat[r][c] = 0;

        }

    }

}

void insert\_mat\_vertex(GraphType\_mat\* g, int v) {

    if ((g->n) > MAX\_VERTICES) {

        fprintf(stderr, "Exceeding the number of vertices");

        return;

    }

    g->n++;

}

int insert\_mat\_edge(GraphType\_mat\* g, int start, int end) {

    if (start == end) {

        return FALSE;

    }

    if (start >= g->n || end >= g->n) {

        fprintf(stderr, "Error\n");

        return FALSE;

    }

    if (g->adj\_mat[start][end] == 1 || g->adj\_mat[end][start] == 1) {

        return FALSE;

    }

    g->adj\_mat[start][end] = 1;

    g->adj\_mat[end][start] = 1;

    return TRUE;

}

void print\_adj\_mat(GraphType\_mat\* g) {

    char a = 'A';

    for (int i = 0; i < g->n; i++) {

        printf("%c  ", a);

        a++;

        for (int j = 0; j < g->n; j++) {

            printf("%2d ", g->adj\_mat[i][j]);

        }

        printf("\n");

    }

}

void dfs\_mat(GraphType\_mat\* g, int v) {

    int w;

    char a = 'A';

    a = a + v;

    visted\_mat[v] = TRUE;

    printf("%c ", a);

    for (w = 0; w < g->n; w++) {

        if (g->adj\_mat[v][w] && !visted\_mat[w])

            dfs\_mat(g, w);

    }

}

typedef struct GraphNode {

    int vertex;

    struct GraphNode\* link;

}GraphNode;

typedef struct GraphType\_list {

    int n;

    GraphNode\* adj\_list[MAX\_VERTICES];

}GraphType\_list;

void init\_list(GraphType\_list\* g) {

    int v;

    g->n = 0;

    for (v = 0; v < MAX\_VERTICES; v++) {

        g->adj\_list[v] = NULL;

    }

}

void dfs\_list(GraphType\_list\* g, int v) {

    GraphNode\* w;

    char c = 'A';

    c = c + v;

    visted\_list[v] = TRUE;

    printf("%c ", c);

    w = g->adj\_list[v];

    for (w; w; w = w->link) {

        if (!visted\_list[w->vertex])

            dfs\_list(g, w->vertex);

    }

}

void insert\_list\_vertex(GraphType\_list\* g, int v) {

    if (g->n >= MAX\_VERTICES) {

        fprintf(stderr, "Exceeding the number of vertices");

        return;

    }

    g->n++;

}

int insert\_list\_edge(GraphType\_list\* g, int u, int v) {

    if (u >= g->n || v >= g->n) {

        fprintf(stderr, "Error");

        return FALSE;

    }

    GraphNode\* node = (GraphNode\*)malloc(sizeof(GraphNode));

    if (!node) {

        fprintf(stderr, "Error");

        return FALSE;

    }

    node->vertex = v;

    node->link = g->adj\_list[u];

    g->adj\_list[u] = node;

    return TRUE;

}

void print\_adj\_list(GraphType\_list\* g) {

    for (int i = 0; i < g->n; i++) {

        GraphNode\* p = g->adj\_list[i];

        printf("Adjacency list of vertex %d:", i);

        while (p != NULL) {

            printf(" -> %d", p->vertex);

            p = p->link;

        }

        printf("\n");

    }

}

int main() {

    GraphType\_mat\* g\_m;

    g\_m = (GraphType\_mat\*)malloc(sizeof(GraphType\_mat));

    init\_mat(g\_m);

    srand(time(NULL));

    printf("Please enter the number of vertices: ");

    int ver;

    scanf("%d", &ver);

    for (int i = 0; i < ver; i++) {

        insert\_mat\_vertex(g\_m, i);

    }

    printf("Generate a random connected graph\n");

    int max\_edge = ver \* (ver - 1) / 2;

    printf("Number of edges (less then %d): ", max\_edge);

    int edge;

    scanf("%d", &edge);

    int i = 0;

    while (i < edge) {

        int randomX = rand() % ver;

        int randomY = rand() % ver;

        if (!insert\_mat\_edge(g\_m, randomX, randomY)) {

            continue;

        }

        i++;

    }

    print\_adj\_mat(g\_m);

    printf("adjacency matrix DFS: ");

    dfs\_mat(g\_m, 0);

    GraphType\_list\* g\_l;

    g\_l = (GraphType\_list\*)malloc(sizeof(GraphType\_list));

    init\_list(g\_l);

    for (int i = 0; i < g\_m->n; i++) {

        insert\_list\_vertex(g\_l, i);

    }

    for (int i = 0; i < g\_m->n; i++) {

        for (int j = 0; j < g\_m->n; j++) {

            if (g\_m->adj\_mat[i][j] == 1) {

                insert\_list\_edge(g\_l, i, j);

            }

        }

    }

    print\_adj\_list(g\_l);

    printf("\nadjacency list DFS: ");

    dfs\_list(g\_l, 0);

    printf("\n");

    free(g\_m);

    free(g\_l);

    return 0;

}

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명2. 실행 과정**

**텍스트, 도표, 폰트, 친필이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 라인, 친필이(가) 표시된 사진

자동 생성된 설명**

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**3. 실행 결과**

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

인접 행렬 DFS와  
인접 리스트 DFS의 탐색 경로가   
다른 모습을 볼 수 있다.