

Домашнє завдання №16.1

Скласти програму (C/C++), яка дозволяє знаходити у графі мінімальний шлях від заданої вершини до інших вершин за допомогою алгоритму Дейкстри.

Вибір варіанту

$(N_{\text{ж}} + N_{\text{г}} + 1) \% 2 + 1$
де: $N_{\text{ж}}$ – порядковий номер студента в групі, а $N_{\text{г}}$ – номер групи(1,2,3,4,5,6,7 або 8)

Варіанти завдань

Варіант	Кількість вершин графу
1	4
2	5

Приклад коду

Програма відображає заданий граф у вигляді матриці суміжності (*англ.* adjacency matrix), в якій замість чисел 0 і 1(відсутність або присутність ребра), містяться ваги ребер(на відсутність ребра вказує значення NE – not exist).

Знайдені мінімальні шляхи від заданої вершини до інших вершин графу відображаються у вигляді послідовності проміжних і кінцевих вершин.

Кількість вершин графу у прикладі	7
Макровизначення	<code>#define VERTEX_COUNT 7</code>

Лістинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define VERTEX_COUNT 7
#define MAX_VERTEX_COUNT 8

#define UNDIRECT_BEHAVIOR

#define NE (~0) // NOT EXIST
#define NA NE // NOT AVAILABLE

#define EDGE_VALUES {\
/*****V0**V1**V2**V3**V4**V5**V6**V7*/\
/*V0*/{NA, 7, NE, 5, NE, NE, NE},\
/*V1*/{NA, NA, 8, 9, 7, NE, NE},\
/*V2*/{NA, NA, NA, NE, 5, NE, NE},\
/*V3*/{NA, NA, NA, NA, 15, 6, NE},\
/*V4*/{NA, NA, NA, NA, NA, 8, 9},\
/*V5*/{NA, NA, NA, NA, NA, NA, 11},\
/*V6*/{NA, NA, NA, NA, NA, NA, NA},\
/*V7*/{NA, NA, NA, NA, NA, NA, NA}\
}
```

```

}

#define COMPLETE 0
#define NOT_COMPLETE (~COMPLETE)
#define INFINITY (~0)

#define TITLE_MAX_SIZE 256

typedef struct VertexStruct {
    unsigned int prevIndex;
    unsigned int value;
    unsigned int state;
} Vertex;

typedef struct VerticesStruct {
    unsigned int vertexCount;
    Vertex* items;
} Vertices;

void destroyVertices(Vertexes* vertexes) {
    if (vertexes) {
        free(vertexes->items);
        free(vertexes);
    }
}

Vertexes* runDijkstrasAlgorithm(int edgeValues[MAX_VERTEX_COUNT][MAX_VERTEX_COUNT],
    unsigned int vertexCount, unsigned int sourceVertexIndex) {
    unsigned int vertexIndex;
    unsigned int baseVertexIndex, neighborVertexIndex;
    unsigned int distance, distanceAddon, tryNewDistance;

    Vertexes* vertexes = (Vertexes*)malloc(sizeof(Vertexes));
    vertexes->vertexCount = vertexCount;

    vertexes->items = (Vertex*)malloc(vertexes->vertexCount * sizeof(Vertex));

    for (vertexIndex = 0; vertexIndex < vertexCount; ++vertexIndex) {
        vertexes->items[vertexIndex].prevIndex = NE;
        vertexes->items[vertexIndex].value = INFINITY;
        vertexes->items[vertexIndex].state = NOT_COMPLETE;
    }
    vertexes->items[sourceVertexIndex].value = 0;

    for (distance = INFINITY;; distance = INFINITY) {
        for (baseVertexIndex = 0, vertexIndex = 0; vertexIndex < vertexCount;
            ++vertexIndex) {
            if (distance > vertexes->items[vertexIndex].value && vertexes->items[vertexIndex].state != COMPLETE) {
                distance = vertexes->items[vertexIndex].value;
                baseVertexIndex = vertexIndex;
            }
        }
        if (distance == INFINITY) {
            break;
        }

        vertexes->items[baseVertexIndex].state = COMPLETE;

        for (neighborVertexIndex = 0; neighborVertexIndex < vertexCount;
            ++neighborVertexIndex) {
            distanceAddon = edgeValues[baseVertexIndex][neighborVertexIndex];
#ifdef UNDIRECT_BEHAVIOR
            if (distanceAddon == NE) {
                distanceAddon =

```

```

edgeValues[neighborVertexIndex][baseVertexIndex];
    }
#endif
    if (distanceAddon != NE && vertexes->items[neighborVertexIndex].state
!= COMPLETE) {
        tryNewDistance = distance + distanceAddon;
        if (tryNewDistance < vertexes-
>items[neighborVertexIndex].value) {
            vertexes->items[neighborVertexIndex].value =
tryNewDistance;
            vertexes->items[neighborVertexIndex].prevIndex =
baseVertexIndex;
        }
    }
}
return vertexes;
}

void printGraphEdgeValues(const char* title, int
edgeValues[MAX_VERTEX_COUNT][MAX_VERTEX_COUNT], unsigned int vertexCount) {
    unsigned int iIndex, jIndex;

    printf("%s\r\n ", title);
    for (jIndex = 0; jIndex < vertexCount; ++jIndex) {
        printf(" V%-2d", jIndex);
    }
    printf("\r\n");
    for (iIndex = 0; iIndex < vertexCount; ++iIndex) {
        printf("V%-2d", iIndex);
        for (jIndex = 0; jIndex < vertexCount; ++jIndex) {
            if (jIndex) {
                printf(",");
            }
            printf(" ");
#ifdef UNDIRECT_BEHAVIOR
            if (iIndex < jIndex) {
#endif
                if (edgeValues[iIndex][jIndex] != NE) {
                    printf("%-2d", edgeValues[iIndex][jIndex]);
                }
                else {
                    printf("NE");
                }
#ifdef UNDIRECT_BEHAVIOR
            }
            else {
                printf("NA");
            }
        }
        printf("\r\n");
    }
    printf("\r\n");
}

void printPathToVertex_(Vertexes* vertexes, unsigned int vertexIndex) {
    if (vertexIndex == NE || !vertexes || !vertexes->items) {
        return;
    }

    printPathToVertex_(vertexes, vertexes->items[vertexIndex].prevIndex);

    if (vertexes->items[vertexIndex].prevIndex == NE) {

```

```

        printf("%d", vertexIndex);
    }
    else {
        printf(" => %d", vertexIndex);
    }
}

void printPathToVertex(const char* title, Vertexes* vertexes, unsigned int
destinationVertexIndex) {
    printf("%s ", title);
    printPathToVertex_(vertexes, destinationVertexIndex);
    printf("\r\n");
}

int main() {
    char title[TITLE_MAX_SIZE] = { '\0' };
    unsigned int sourceVertexIndex = 0;
    unsigned int destinationVertexIndex;

    int edgeValues[MAX_VERTEX_COUNT][MAX_VERTEX_COUNT] = EDGE_VALUES;

    Vertexes* vertexes = runDijkstrasAlgorithm(edgeValues, VERTEX_COUNT,
sourceVertexIndex);
    if (!vertexes) {
        return 1;
    }

    printGraphEdgeValues("Graph:", edgeValues, VERTEX_COUNT);

    for (destinationVertexIndex = 0; destinationVertexIndex < VERTEX_COUNT;
++destinationVertexIndex) {
        if (vertexes->items[destinationVertexIndex].state == COMPLETE) {
            sprintf(title, "Patch from %d vertex to %d vertex:",
sourceVertexIndex, destinationVertexIndex);
            printPathToVertex(title, vertexes, destinationVertexIndex);
        }
    }

    destroyVertexes(vertexes);

#ifdef __linux__
    (void)getchar();
#elif defined(_WIN32)
    system("pause");
#else
#endif

    return 0;
}

```