**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

**КУРСОВА РАБОТА**

**по Изкуствен интелект**

Студент: **Самет Онур**

Факултетен номер: **113901**

Група: **26Б**

Специалност: **КСТ**

Курс: **3**

**Изготвил:**

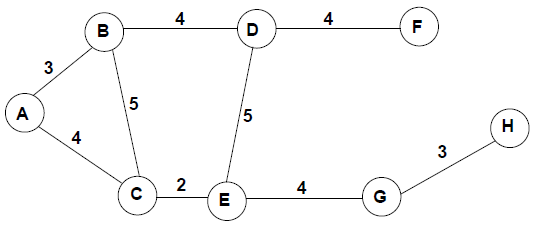
**Дата: Проверил:**

# Задание.

**“Търсене на път по карта” (Map traversal problem):**

Да се намери път между два града, използвайки пътната карта на *Фигура 1*

Алгоритъм: търсене в широчина [*Breadth-first search (BFS)*].



Фигура 1 Пътна карта

1. **Представяне на задачата като задача за търсене в ПС.**

Т.к могат да се изберат кои да са два върха, нека F е началният връх, а H е крайният. Разходите са описани на горната фигура на самите ребра на графа. Като разходът на стрелки в този случай ще бъде 4 + 5 + 4 + 3 = 16, т.к BFS открива минималният път между 2 върха, а това е F D E G H. Нека разгледаме постъпковото изпълнения на алгоритъма:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **X** | **Наследници на Х** | **"Оцелели" наследници на Х** | **OPEN** | **CLOSED** | **Коментар** |
| 0 | - | - | - | [F] | [] | Инициализация |
| 1 | F | [D] | [D] | [D] | [F] |  |
| 2 | D | [B,E] | [B,E] | [B,E] | [F,D] |  |
| 3 | B | [A,C] | [A,C] | [E,A,C] | [F,D,B] |  |
| 4 | E | [C,D,G] | [G] | [A,C,G] | [F,D,B,E] | C - OPEN; D - CLOSED |
| 5 | А | [B,C] | - | [C,G] | [F,D,B,E,A] | B - CLOSED; C - OPEN |
| 6 | C | [A,B,E] | - | [G] | [F,D,B,E,A,C] | A,B,E - CLOSED |
| 7 | G | [E,H] | [H] | [H] | [F,D,B,E,A,C,G] | E - CLOSED |
| 8 | H |  |  |  |  | **край** |

Вземаме H и гледаме в съдържанието на OPEN от предходната стъпка (6) дали H се съдържа в него, ако не – продължаваме с първият елемент (от ляво на дясно), в случая имаме само G, т.е този връх му е родител. По аналогичен начин продължаваме с G, но го има в предходните 2 стъпки (4 и 5), в този случай не правим нищо и продължаваме. Виждаме, че върхът G го няма в 3-та стъпка, а на първо място стои Е. Вземаме него и продължаваме. Е го има в 2-ра стъпка, затова преминаваме към 1-ва стъпка и вземаме D и след него F. По този начин получихме пътя (H G E D F), но трябва да го изведем в обратен ред (F D E G H).

# Програмна реализация.

## Псевдо-код на алгоритъма.

BFS ( start, end )

{

Създаваме празна опашка Queue;

Добавяме към опашката върха i;

Маркираме върха i като посетен

while (Опашката не е празна)

{

p = вземаме поредния елемент от опашката;

Анализираме върха p;

за всеки необходен наследник j на queue[p]

for (от j = 0 до j < n)

{

if (е наследник *И* не е посетен)

{

добавяме „оцелелият“ наследник в опашката

„Маркираме“ j като посетен

„Маркираме“ върха queue[p] като част от пътя

Ако сме стигнали до крайния връх – прекратяваме търсенето

}

}

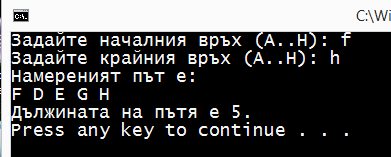
}

}

## Описание на разработениете модули.

* **Вид на функцията**: void BFS(unsigned i, unsigned end)
* **Действие**: Функцията служи за намиране на път между два върха в един граф, описан чрез матрица на съседство, чрез *Breadth-first search* алгоритъма;
* **Параметри**: i, end – номерата съответно на началния и крайния връх;
* **Използвани функции**: няма
* **Вид на функцията**: unsigned printPath(unsigned j)
* **Действие**: Функцията служи за отпечатване върховете от пътя и връща дължината му;
* **Параметри**: j – номер на крайния връх;
* **Използвани функции**: рекурсия
* **Вид на функцията**: void solve(unsigned start, unsigned end)
* **Действие**: Функцията служи за намиране и отпечатване на резултата
* **Параметри**: start, end – номерата съответно на началния и крайния връх;
* **Използвани функции**: BFS(), printPath().
* **Вид на функцията**: int main(void)
* **Действие**: Главна функция
* **Параметри**: няма;
* **Използвани функции**: solve().

## Тестови примери.



Фигура 2 Тестов пример

## Код на програмата.

#include <stdio.h>

#include <iostream>

#define MAXN 200 /\* Максимален брой върхове в графа \*/

/\* Брой върхове в графа \*/

const unsigned n = 8;

char sv; /\* Начален връх \*/

char ev; /\* Краен връх \*/

/\* Матрица на съседство на графа \*/

const char A[MAXN][MAXN] = {

{ 0, 1, 1, 0, 0, 0, 0, 0 }, // A - 1

{ 1, 0, 1, 1, 0, 0, 0, 0 }, // B - 2

{ 1, 1, 0, 0, 1, 0, 0, 0 }, // C - 3

{ 0, 1, 0, 0, 1, 1, 0, 0 }, // D - 4

{ 0, 0, 1, 1, 0, 0, 1, 0 }, // E - 5

{ 0, 0, 0, 1, 0, 0, 0, 0 }, // F - 6

{ 0, 0, 0, 0, 1, 0, 0, 1 }, // G - 7

{ 0, 0, 0, 0, 0, 0, 1, 0 }, // H - 8

};

int pred[MAXN];

char used[MAXN];

/\* Oбхождане в ширина от даден връх със запазване на предшественика \*/

void BFS(unsigned i, unsigned end) {

unsigned queue[MAXN];

unsigned currentVert, levelVertex, queueEnd, k, p, j;

for (k = 0; k < n; k++) {

queue[k] = 0;

}

queue[0] = i;

used[i] = 1;

currentVert = 0;

levelVertex = 1;

queueEnd = 1;

while (currentVert < queueEnd) { /\* докато опашката не е празна \*/

for (p = currentVert; p < levelVertex; p++) {

/\* p - вземаме поредния елемент от опашката \*/

currentVert++;

/\* за всеки необходен наследник j на queue[p] \*/

for (j = 0; j < n; j++) {

if (A[queue[p]][j] && !used[j]) {

queue[queueEnd++] = j;

used[j] = 1;

pred[j] = queue[p];

if (pred[end] > -1) {

// пътят е намерен

return;

}

}

}

}

levelVertex = queueEnd;

}

}

/\* Отпечатва върховете от минималния път и връща дължината му \*/

unsigned printPath(unsigned j) {

unsigned count = 1;

if (pred[j] > -1) {

count += printPath(pred[j]);

}

printf("%c ", j + 'A'); /\* Oтпечатва поредния връх от намерения път \*/

return count;

}

void solve(unsigned start, unsigned end) {

unsigned k;

for (k = 0; k < n; k++) {

used[k] = 0;

pred[k] = -1;

}

BFS(start, end);

if (pred[end] > -1) {

printf("Намереният път е: \n");

printf("\nДължината на пътя е %u.\n", printPath(end));

}

else

printf("Не съществува път между двата върха! \n");

}

int main(void) {

setlocale(LC\_ALL, "BGR");

printf("Задайте началния връх (A..%c): ", n + 64);

scanf("%c", &sv);

sv = toupper(sv);

std::cin.sync();

printf("Задайте крайния връх (A..%c): ", n + 64, sv);

scanf("%c", &ev);

ev = toupper(ev);

solve(sv - 'A', ev - 'A');

return 0;

}

# Творческа задача.

Необходимо е да се премахне логиката за прекратяване на търсенето след като е намерен крайният връх

if (pred[end] > -1) {

// пътят е намерен

return;

}

и да не се маркират вече посетените върхове. Също така трябва да се има в предвид и зациклянето.