**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

**КУРСОВА РАБОТА**

**по Изкуствен интелект**

Студент:

Факултетен номер:

Група:

Специалност:

Курс:

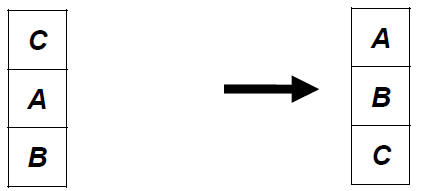
**Изготвил:**

**Дата: Проверил:**

# Задание.

**“Подредба на кубчета” (Block-world problem):**

Три кубчета A, B и C са подредени едно върху друго така както е показано на фиг.1. Целта е да се направи план за извършване на преподреждането им.



Фигура 1 Подредба на кубчета

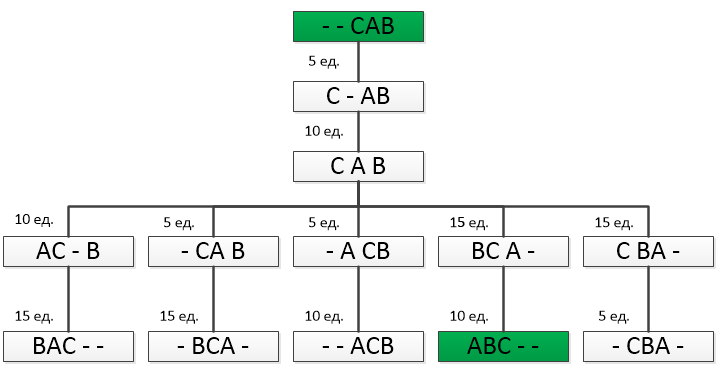
В даден момент е разрешено местенето само на едно кубче. Може да бъде премествано кубче, върху което няма друго такова. Кубчето може да бъде поставяно върху масата или върху друго такова. Нека кубчетата са с различна тежест, т.е. разходите за местенето им са различни.

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

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

* **Състояние**: Нека състоянието на тази задача да се представи чрез наредена тройка (α1, α2, α3), където α1, α2 и α3 са трите места, където могат да се сложат кубчетата.
* **Начално състояние**: (–,–, CAB).
* **Финално състояние**: (ABC,–, –).
* **Оператори и разходи на стрелките**:
* **A** – прехвърляне на кубчето А (10 ед.);
* **B** – прехвърляне на кубчето B (15 ед.);
* **C** – прехвърляне на кубчето C (5 ед.);

# Граф на ПС.



# 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

}

}

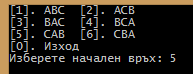
}

}

## 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 върха;
* **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 <stdlib.h>

#include <iostream>

using namespace std;

#define MAXN 20 /\* Максимален брой върхове в графа \*/

/\* Брой върхове в графа \*/

const unsigned n = 13;

const unsigned ev = 11; /\* Краен връх \*/

unsigned sv; /\* Начален връх \*/

/\* Матрица на съседство на графа \*/

const char A[MAXN][MAXN] = {

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, //0 - - CAB

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

{ 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 }, //2 C A B

{ 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 }, //3 AC - B

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 }, //4 - CA B

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 }, //5 - A CB

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 }, //6 BC A -

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 }, //7 C BA -

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

{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 }, //9 - BCA -

{ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 }, //10 - - ACB

{ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 }, //11 ABC - -

{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 } //12 - CBA -

};

const string States[n] = {

"- - CAB",

"C - AB",

"C A B",

"AC - B",

"- CA B",

"- A CB",

"BC A -",

"C BA -",

"BAC - -",

"- BCA -",

"- - ACB",

"ABC - -",

"- CBA -"

};

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]);

}

cout << States[j] << endl; /\* Oтпечатва поредния връх от намерения път \*/

return count;

}

void solve(unsigned start, unsigned end) {

unsigned k;

if (sv == ev) {

cout << "Кубчетата са подредени!" << endl;

}

else {

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

used[k] = 0;

pred[k] = -1;

}

BFS(start, end);

if (pred[end] > -1) {

cout << "Резултат:" << endl;

printPath(end);

}

}

}

int main(void) {

setlocale(LC\_ALL, "BGR");

unsigned ch;

do {

cout << "[1]. ABC";

cout << " [2]. ACB" << endl;

cout << "[3]. BAC";

cout << " [4]. BCA" << endl;

cout << "[5]. CAB";

cout << " [6]. CBA" << endl;

cout << "[0]. Изход" << endl;

cout << "Изберете начален връх: ";

cin >> ch;

system("cls");

switch(ch) {

case 1: sv = 11; break;

case 2: sv = 10; break;

case 3: sv = 8; break;

case 4: sv = 9; break;

case 5: sv = 0; break;

case 6: sv = 12; break;

case 0: ch = 0; break;

default: ch = -1; break;

}

if (ch != 0 && ch != -1) {

cout << endl;

solve(sv, ev);

cout << endl;

}

} while(ch != 0);

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.

Така както е представен графът обаче има само едно решение.