**РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”**

**КУРСОВА РАБОТА**

**№1**

**по Изкуствен интелект**

Студент:

Факултетен номер:

Група:

Специалност:

Курс:

**Изготвил:**

**Дата: Проверил:**

# Задание.

“Мисионери и канибали” (Missionary and cannibal problem) - I I -ри вариант:

N мисионери (N>=1) и N канибали се намират на левия бряг на една река. Целта на задачата е всички мисионери и канибали да бъдат прехвърлени на десния бряг на реката като се използва лодка. Ограниченията са следните:

1. Лодката е най-много за трима човека (единият, от които гребе).
2. На кой да е бряг или в лодката, броят на мисионерите трябва да е по-голям или равен на броя на канибалите.

Нека разходите за превозването на канибал са 1 единица, а на мисионер - 2 единици.

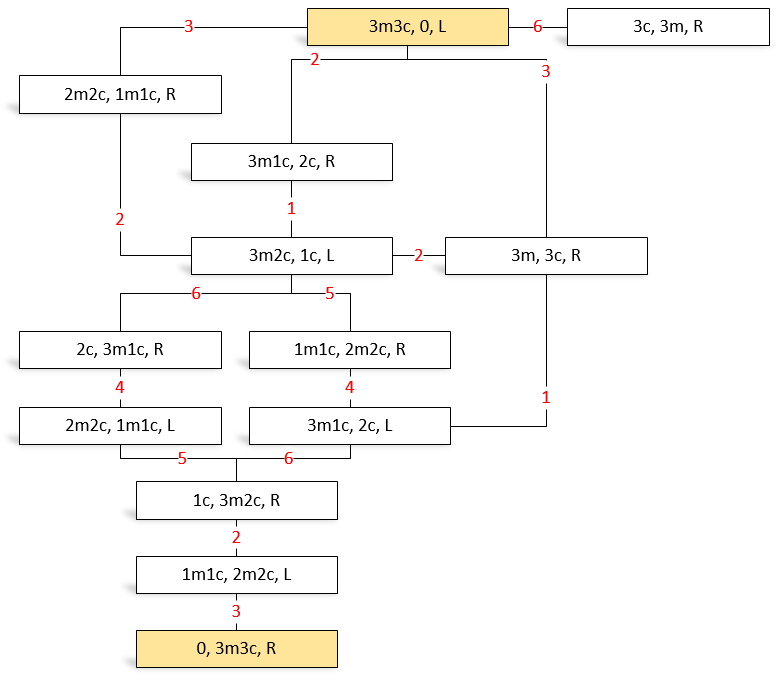
*Забележка:* Предложете решение на задачата за N=3.

Алгоритъм: търсене с връщане [*Backtracking*].

# 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 и α2 описват кои хора са съответно на левия и десния бряг, а α3 показва къде се намира лодката. Ако със символа m се означат мисионерите, с c – канибалите , а с L и R – съответно ляв и десен бряг.
* **Начално състояние** – (3m3c, 0, L).
* **Финално състояние** – (0, 3m3c, R).
* **Оператори и разходи на стрелките** –
* **1m** – прехвърляне на 1 мисионер (2 ед.);
* **2m** – прехвърляне на 2 мисионери (4 ед.);
* **3m** – прехвърляне на 2 мисионери (6 ед.);
* **1m1c** – прехвърляне на 1 мисионер и 1 канибал (3 ед.);
* **1c** – прехвърляне на 1 канибал (1 ед.);
* **2c** – прехвърляне на 2 канибала (2 ед.);
* **3c** – прехвърляне на 2 канибала (3 ед.);
* **2m1c** – прехвърляне на 2 мисионери и 1 канибал (5 ед.);

# Граф на ПС.



# Програмна реализация.

## Псевдо-код на алгоритъма.

void опит(стъпка i, depth) {

if (i == GOAL) {

печатаме\_решението;

exit() – прекратяваме\_търсенето\_на\_други\_решения

}

If (depth > 0)

/\* разширяване на частичното решение по всички възможни начини \*/

for (k = 1; k <= n; k++)

if (k-тият\_кандидат\_е\_приемлив) {

регистрация\_на\_кандидата;

depth--;

опит(i+1, depth);

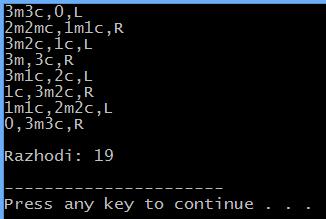
премахване\_на\_регистрацията;

}

## Описание на разработениете модули.

* **Вид на функцията**: backtracking(unsigned i)
* **Действие**: Функцията служи за намиране на път между два върха в един граф, описан чрез матрица на съседство (теглата), чрез *Backtracking* алгоритъма;
* **Параметри**: i – номерът на началния връх и на всеки следващ, който се разглежда;
* **Използвани функции**: рекурсия
* **Вид на функцията**: int main(void)
* **Действие**: Главна функция
* **Параметри**: няма;
* **Използвани функции**: backtracking ().

## Тестови примери.



## Код на програмата.

**#include <stdlib.h>**

**#include <iostream>**

**using namespace std;**

**#define MAXN 20**

**const unsigned n = 13; // broi vurhove**

**const unsigned ev = 12; // kraen vruh**

**unsigned sv = 0; // startov vruh**

**const char A[MAXN][MAXN] = {**

**{0,3,2,3,6,0,0,0,0,0,0,0,0}, //0 | 3m3c,0,L**

**{3,0,0,0,0,2,0,0,0,0,0,0,0}, //1 | 2m2mc,1m1c,R**

**{2,0,0,0,0,1,0,0,0,0,0,0,0}, //2 | 3m1c,2c,R**

**{3,0,0,0,0,2,0,0,0,1,0,0,0}, //3 | 3m,3c,R**

**{6,0,0,0,0,0,0,0,0,0,0,0,0}, //4 | 3c,3m,R**

**{0,2,1,2,0,0,6,5,0,0,0,0,0}, //5 | 3m2c,1c,L**

**{0,0,0,0,0,6,0,0,4,0,0,0,0}, //6 | 2c,3m1c,R**

**{0,0,0,0,0,5,0,0,0,4,0,0,0}, //7 | 1m1c,2m2c,R**

**{0,0,0,0,0,0,4,0,0,0,5,0,0}, //8 | 2m2c,1m1c,L**

**{0,0,0,1,0,0,0,4,0,0,6,0,0}, //9 | 3m1c,2c,L**

**{0,0,0,0,0,0,0,0,5,6,0,2,0}, //10 | 1c,3m2c,R**

**{0,0,0,0,0,0,0,0,0,0,2,0,3}, //11 | 1m1c,2m2c,L**

**{0,0,0,0,0,0,0,0,0,0,0,3,0}, //12 | 0,3m3c,R**

**};**

**const string StatesStr[n] = {**

**"3m3c,0,L", "2m2mc,1m1c,R", "3m1c,2c,R",**

**"3m,3c,R", "3c,3m,R", "3m2c,1c,L",**

**"2c,3m1c,R", "1m1c,2m2c,R", "2m2c,1m1c,L",**

**"3m1c,2c,L", "1c,3m2c,R", "1m1c,2m2c,L",**

**"0,3m3c,R"**

**};**

**unsigned vertex[MAXN];**

**char used[MAXN];**

**int tempLen, ti;**

**void addVertex(unsigned i, int depth) {**

**unsigned j, k;**

**if (i == ev) { // putqt e nameren**

**for (j = 0; j < ti; j++)**

**{**

**cout << StatesStr[vertex[j]].c\_str() << endl;**

**}**

**cout << endl << "Razhodi: " << tempLen << endl;**

**cout << endl << "----------------------" << endl;**

**system("pause");**

**exit(1);**

**}**

**if (depth > 0) {**

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

**if (!used[k]) { /\* ako vurhut k ne uchastva v putq do momenta \*/**

**if (A[i][k] > 0) {**

**/\* ako vurhut koito se dobavq e suseden na posledniq ot putq\*/**

**tempLen += A[i][k];**

**used[k] = 1; /\* markirane na k kato uchastvasht v putq \*/**

**vertex[ti++] = k; /\* dobavqne na vyrha k kym putq \*/**

**depth--;**

**addVertex(k, depth);**

**used[k] = 0; /\* demarkirane na k \*/**

**tempLen -= A[i][k];**

**ti--;**

**}**

**}**

**}**

**}**

**}**

**int main(void) {**

**setlocale(LC\_ALL, "BGR");**

**unsigned i;**

**tempLen = 0, ti = 1;**

**for (i = 0; i < n; i++) used[i] = 0;**

**used[sv] = 1; vertex[0] = sv;**

**addVertex(sv, 2\*n);**

**return 0;**

**}**

# Творческа задача.

Достатъчно е да се премахне оператора за прекратяване работата на програмата – exit(1). Псевдо-кодът на алгоритъма става:

void опит(стъпка i, depth) {

if (i == GOAL) {

печатаме\_решението;

}

If (depth > 0)

/\* разширяване на частичното решение по всички възможни начини \*/

for (k = 1; k <= n; k++)

if (k-тият\_кандидат\_е\_приемлив) {

регистрация\_на\_кандидата;

опит(i+1, depth);

премахване\_на\_регистрацията;

}