**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.

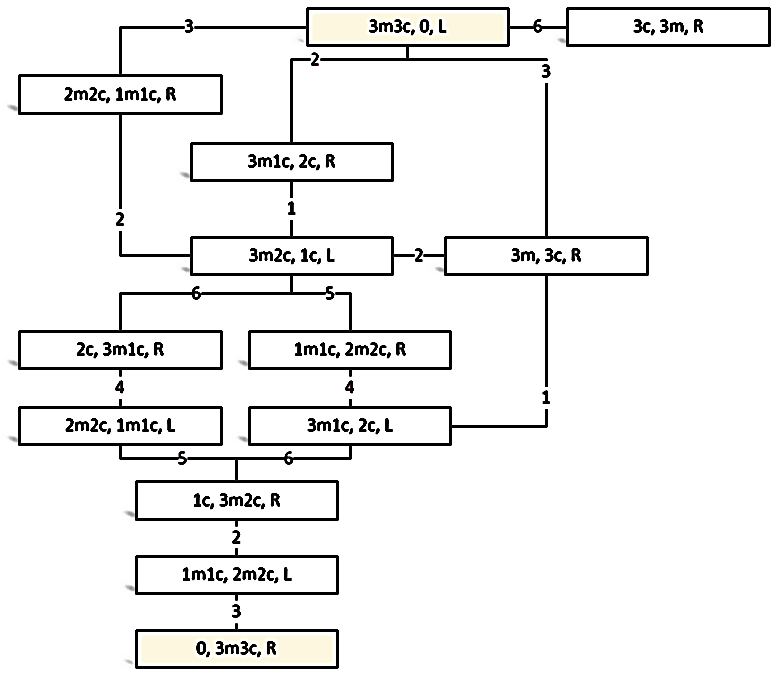
*Забележка:* Предложете решение на задачата за N=3.

Алгоритъм: Uniform Cost Search.

# 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.

procedure **uniform\_search**;

begin

OPEN:=[START];

CLOSED:=[ ];

while OPEN<>[ ] do

begin

*Изтрий най-лявото състояние на* OPEN *и го запиши в* X*;*

if X=GOAL then return(success)

else begin

*Намери наследниците на* X*;*

*Запиши* X *в* CLOSED*;*

for *всеки наследник* Y *на* X do

case

Y *не е в* OPEN *и не е в* CLOSED: *запиши* Y *в* OPEN;

*копие* Yc *на* Y *е в* OPEN:

if path\_cost(Y) < path\_cost(Yc) then begin

*изтрий* Yc;

*запиши* Y *в* OPEN; end

end*;*

end; {case}

end; {else}

*подреди състоянията от* OPEN *според разходите на пътищата до тях;*

end; {while}

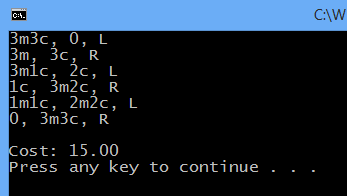
return(failure);

end; {uniform\_search}

## Описание на разработениете модули.

* **Вид на функцията**: void uniformCostSearch(int start, int goal)
* **Действие**: Функцията служи за намиране на път между два върха в един граф, описан чрез матрица на теглата, чрез *Uniform Cost Search* алгоритъма;
* **Параметри**: start – номерът на началния (започват от 0); end – номерт на крайния връх;
* **Използвани функции**: printPath(), removeNode(), contains(), sortNodes()
* **Вид на функцията**: void printPath(Node node)
* **Действие**: служи за печатане на намерения път;
* **Параметри**: node – върхът, към който път трябва да се отпечатва;
* **Използвани функции**: няма.
* **Вид на функцията:** bool contains(vector<Node> nodeList, Node node)
* **Действие**: функция за проверка дали node се съдържа в nodeList;
* **Параметри**: nodeList – списъкът с върховете; node – върхът за търсене;
* **Използвани функции**: няма.
* **Вид на функцията:** vector<Node> removeNode(vector<Node> nodeList, Node node)
* **Действие**: функция за „изтриване“ на даден връх;
* **Параметри**: nodeList – списъкът с върховете; node – върхът за изтриване;
* **Използвани функции**: няма.
* **Вид на функцията:** vector<Node> getChilds(Node node)
* **Действие**: функция за „генериране“ наследниците на node;
* **Параметри**: node – върхът, чийто наследници се търсят;
* **Използвани функции**: няма.
* **Вид на функцията**: int main(void)
* **Действие**: Главна функция
* **Параметри**: няма;
* **Използвани функции**: uniformCostSearch ().

## Тестови примери.



## Код на програмата.

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**const unsigned NODE\_COUNT = 13;**

**int graph[NODE\_COUNT][NODE\_COUNT] =**

**{**

**{ 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**

**};**

**string lblStates[NODE\_COUNT] =**

**{**

**"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",**

**};**

**struct Node {**

**int index;**

**double cost;**

**double pathCost;**

**vector<int> path;**

**};**

**vector<Node> getChilds(Node node)**

**{**

**vector<Node> childs;**

**for (int i = 0; i < NODE\_COUNT; i++)**

**{**

**if (graph[node.index][i] > 0)**

**{**

**Node child = { i, graph[node.index][i], graph[node.index][i] + node.pathCost };**

**child.path = node.path;**

**child.path.push\_back(node.index);**

**childs.push\_back(child);**

**}**

**}**

**return childs;**

**}**

**bool contains(vector<Node> nodeList, Node node)**

**{**

**bool result = false;**

**for (int i = 0; i < nodeList.size(); i++)**

**{**

**if (nodeList[i].index == node.index)**

**{**

**result = true;**

**break;**

**}**

**}**

**return result;**

**}**

**vector<Node> removeNode(vector<Node> nodeList, Node node)**

**{**

**vector<Node> result;**

**for (int i = 0; i < nodeList.size(); i++)**

**{**

**if (nodeList[i].index != node.index)**

**{**

**result.push\_back(nodeList[i]);**

**}**

**}**

**return result;**

**}**

**void printPath(Node node)**

**{**

**for (int i = 0; i < node.path.size(); i++)**

**{**

**printf("%s\n", lblStates[node.path[i]].c\_str());**

**}**

**printf("%s\n", lblStates[node.index].c\_str());**

**}**

**void uniformCostSearch(int start, int goal)**

**{**

**Node source = { start, 0, 0 };**

**vector<Node> open;**

**open.push\_back(source);**

**vector<Node> closed;**

**while (open.size() > 0)**

**{**

**Node x = open[0];**

**open = removeNode(open, x);**

**if (x.index == goal)**

**{**

**printPath(x);**

**printf("\nCost: %.2f\n", x.pathCost);**

**return;**

**}**

**else**

**{**

**vector<Node> childs = getChilds(x);**

**closed.push\_back(x);**

**for (int i = 0; i < childs.size(); i++)**

**{**

**Node y = childs[i];**

**if (!contains(open, y) &&**

**!contains(closed, y))**

**{**

**open.push\_back(y);**

**}**

**else**

**{**

**Node yc;**

**bool update = false;**

**for (int j = 0; j < open.size(); j++)**

**{**

**yc = open[j];**

**if (y.index == yc.index &&**

**y.pathCost < yc.pathCost)**

**{**

**update = true;**

**break;**

**}**

**}**

**if (update)**

**{**

**removeNode(open, yc);**

**open.push\_back(y);**

**}**

**}**

**}**

**}**

**// sort open**

**for (int k = 0; k < open.size() - 1; k++)**

**{**

**for (int h = 0; h < open.size() - k - 1; h++)**

**{**

**if (open[h].pathCost > open[h + 1].pathCost)**

**{**

**Node temp = open[h];**

**open[h] = open[h + 1];**

**open[h + 1] = temp;**

**}**

**}**

**}**

**}**

**}**

**int main()**

**{**

**uniformCostSearch(0, 12);**

**return 0;**

**}**

# Творческа задача.

В OPEN да се добавят и върховете, които вече се намират в него, но с различен vector<int> path и/или double pathCost. Освен това при достигане на целта да не се прекратява работата на функцията, а само да се печата текущото намерено решение и да се продължи, докато OPEN<>[].