**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет

имени Н.Э. Баумана»

(МГТУ им. Н.Э.Баумана)

Факультет «Робототехника и комплексная автоматизация» (РК)

Кафедра «Системы автоматизированного проектирования» (РК-6)

***Отчет***

***по лабораторной работе №3***

***по курсу***

***Проектирование интеллектуальных систем***

Выполнил: студент группы РК6-22М Лещев И.А.

Руководитель: Божко А.Н.

Вариант 11

Москва

2018

**Оглавление**

[**Описание алгоритма** 2](#_Toc515976537)

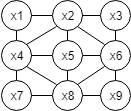
[**Результаты выполнения программы** 3](#_Toc515976538)

[**Описание программных переменных и методов** 4](#_Toc515976539)

[**Программная реализация** 5](#_Toc515976540)

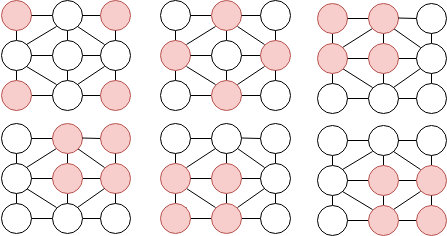
**Задача**

Все цифры от 1 до 9 впишите в кружочки так, чтобы сумма цифр в вершинах любого квадрата были одинаковы.



**Рисунок 1.** Вершины и связи образующие квадраты

Выделим все квадраты суммы вершин, которых необходимо оценивать:



**Рисунок 2.** Квадраты, образованные связями

Таким образом условием получаем следующую систему ограничений:

где некоторая константа.

# **Описание алгоритма**

«Поиск с возвратом для удовлетворения ограничений»

function Backtracking-Search(csp)

**return** решение result

или **return** индикатор отказа failure Recursive-Backtracking({}. сsр)

function Recursive-Backtracking(assignment. сsр)

**return** решение result

или **return** индикатор отказа failure

**if** присваивание assignment является полным

**then return** assignment

var = Select-Unassigned-Variable(Variables[csp]. assignment. сsр)

**for each** value in Order-Domain-Values(var, assignment, сsр)

**do**

**if** значение value является совместимым с присваиванием

assignment согласно ограничениям Constraints[csp]

**then** добавить {var = value} к присваиванию assignment

result = Recursive-Backtracking (assignment, сsр)

**if** result != failure **then return** result

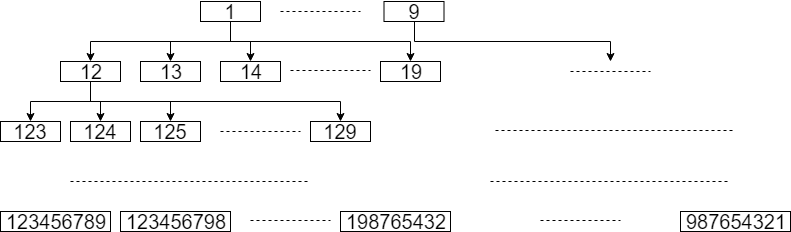
Удалить {var = value} из присваивания assignment

**return** failure

# **Результаты выполнения программы**



**Рисунок 3.** Результат выполнения программы

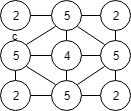


**Рисунок 4.** Схема дерева возможных комбинаций значений

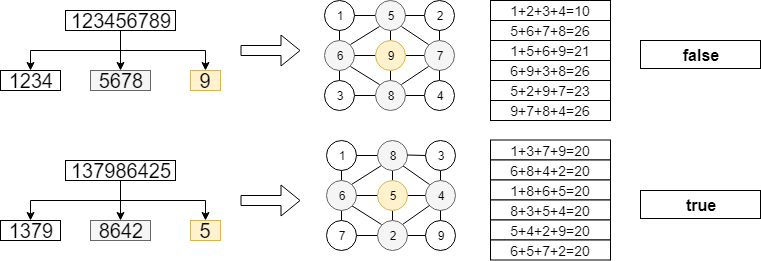
На рисунке 4 представлена схема рекурсивного поиска комбинаций цифр от 1 до 9. Каждая полученная комбинация проверяется на соответствие критерию поиска по схеме, представленной ниже.

Для более эффективного поиска решения используется эвристика с минимальным количеством оставшихся значений. Это позволяет избежать бессмысленных поисков среди переменных, которые всегда будут оканчиваться неудачей.

Для выбора следующей ячейки для подстановки значения используется степенная эвристика. На рисунке 5 представлены степени каждой из вершин графа.



**Рисунок 5.** Степени каждой из вершин графа



**Рисунок 6.** Проверка на соответствие критерию поиска

На рисунке 5 видно, что верхний набор значений не является удовлетворительным т.к. не все суммы вершин квадратов равны друг другу, и наоборот нижний набор значений является удовлетворяющим ограничения**.**

# **Описание программных переменных и методов**

**class csp** – класс, используемый для описания ограничений, накладываемых на узлы графа.

**static public bool Verify(string assignment)** – метод класса csp, возвращает значение «истина» если все ограничения удовлетворены и «ложь», если это не так.

**class Graph** – класс, используемый для описания графа и связей между его вершинами.

**int[,] graph** – представленный в виде массива граф.

**Dictionary<int, string[]> vertex\_connections** – структура соотносящая вершины с их рангами.

**Dictionary<int, string[]> vertex\_busy** – структура содержащая список вершин, который не рассматриваются при выборе следующей ячейки для заполнения.

**Void Update()** – метод обновляющий значения переменных в соответствии со значениями массива graph.

**public void Assign(int value, int row, int col)** – присваивает значение ячейки массива graph.

**public int SelectNextNode()** – выбор следующей вершины с наибольшим рангом.

**public int NextMaxRank()** – значение текущего наибольшего ранга.

**static void CutTheTree(string assignment,char[] set,Graph g)** – «обрезает» дерево поиска если была найдена тупиковая ветка.

**static void RecursiveBacktracking(string assignment, char[] set,Graph g)** – функция рекурсивно обрабатывающая дерево дзначений.

# **Программная реализация**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace backtracking

{

class main {

static void Main(string[] args)

{

char[] set = new char[] { '1', '2', '3', '4','5','6', '7', '8', '9', };

Graph g = new Graph();

RecursiveBacktracking(String.Empty, set,g);

Console.WriteLine("Failure");

Console.ReadLine();

}

public class csp

{

static public bool Verify(Graph g)

{

foreach (var i in g.graph)

if (i == 0)

return false;

int firstSumm = 0;

firstSumm = g.x1 + g.x3 + g.x7 + g.x9;

if (firstSumm != g.x4 + g.x2 + g.x6 + g.x8)

return false;

else if (firstSumm != g.x1 + g.x2 + g.x4 + g.x5)

return false;

else if (firstSumm != g.x2 + g.x3 + g.x5 + g.x6)

return false;

else if (firstSumm != g.x4 + g.x5 + g.x7 + g.x8)

return false;

else if (firstSumm != g.x5 + g.x6 + g.x8 + g.x9)

return false;

return true;

}

static public bool Verify(string assignment)

{

char[] temp = assignment.ToCharArray();

int summ = 0;

if (temp.Length == 6)

{

summ = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[2].ToString()) + int.Parse(temp[3].ToString());

int summ2 = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[5].ToString());

if (summ2 != summ) return false;

}

else if (temp.Length == 7)

{

summ = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[2].ToString()) + int.Parse(temp[3].ToString());

int summ2 = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[5].ToString());

int summ3 = int.Parse(temp[0].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[6].ToString()) + int.Parse(temp[2].ToString());

if (summ2 != summ || summ3 != summ) return false;

}

else if (temp.Length == 8)

{

summ = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[2].ToString()) + int.Parse(temp[3].ToString());

int summ2 = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[5].ToString());

int summ3 = int.Parse(temp[0].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[6].ToString()) + int.Parse(temp[2].ToString());

int summ4 = int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[7].ToString()) + int.Parse(temp[3].ToString());

if (summ2 != summ || summ3 != summ || summ4!=summ) return false;

}

else if(temp.Length == 9)

{

summ = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[2].ToString()) + int.Parse(temp[3].ToString());

int summ2 = int.Parse(temp[0].ToString()) + int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[5].ToString());

int summ3 = int.Parse(temp[0].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[6].ToString()) + int.Parse(temp[2].ToString());

int summ4 = int.Parse(temp[1].ToString()) + int.Parse(temp[4].ToString()) + int.Parse(temp[7].ToString()) + int.Parse(temp[3].ToString());

int summ5 = int.Parse(temp[4].ToString()) + int.Parse(temp[2].ToString()) + int.Parse(temp[8].ToString()) + int.Parse(temp[3].ToString());

int summ6 = int.Parse(temp[5].ToString()) + int.Parse(temp[6].ToString()) + int.Parse(temp[8].ToString()) + int.Parse(temp[7].ToString());

if (summ2 != summ || summ3 != summ || summ4 != summ ||

summ5!= summ||summ6!= summ) return false;

}

return true;

}

}

public class Graph

{

public int[,] graph = new int[,]

{{0,0,0},

{0,0,0},

{0,0,0}

};

public int x1;

public int x2;

public int x3;

public int x4;

public int x5;

public int x6;

public int x7;

public int x8;

public int x9;

public Dictionary<int, string[]> vertex\_connections = new Dictionary<int, string[]>();

private Dictionary<int, string[]> vertex\_busy = new Dictionary<int, string[]>();

public Graph()

{

Update();

vertex\_connections.Add(1, new string[] { "2", "4" });

vertex\_connections.Add(2, new string[] { "1", "3", "5", "4", "6" });

vertex\_connections.Add(3, new string[] { "2", "6" });

vertex\_connections.Add(4, new string[] { "1", "7", "5", "8", "2" });

vertex\_connections.Add(5, new string[] { "2", "4", "6", "8" });

vertex\_connections.Add(6, new string[] { "3", "9", "5", "8", "8", });

vertex\_connections.Add(7, new string[] { "8", "4" });

vertex\_connections.Add(8, new string[] { "7", "9", "5", "4", "6", });

vertex\_connections.Add(9, new string[] { "8", "6" });

}

public void Update()

{

x1 = graph[0, 0];

x2 = graph[0, 1];

x3 = graph[0, 2];

x4 = graph[1, 0];

x5 = graph[1, 1];

x6 = graph[1, 2];

x7 = graph[2, 0];

x8 = graph[2, 1];

x9 = graph[2, 2];

}

public void Assign(int value, int row, int col)

{

graph[row, col] = value;

Update();

}

public int SelectNextNode()

{

int result = 0;

int nextMax =NextMaxRank();

result = vertex\_connections.FirstOrDefault(x => x.Value.Count() == nextMax).Key;

return result;

}

public int NextMaxRank()

{

int res = 0;

foreach (var v in vertex\_connections)

if (vertex\_busy.Contains(v))

continue;

else

{

if (res < v.Value.Count())

res = v.Value.Count();

}

return res;

}

}

static void CutTheTree(string assignment,char[] set,Graph g)

{

char[] toLast = new char[] { set[set.Length - 1] };

for (int i = 0; i < set.Length - 1; i++)

assignment += set[i];

RecursiveBacktracking(assignment, toLast, g);

}

static void RecursiveBacktracking(string assignment, char[] set,Graph g)

{

int count = 0;

if (set.Length == 1)

{

//Console.WriteLine(permutation +"\_\_"+set[0]);

char[] chars = assignment.ToCharArray();

int sum1 = 0;

int sum2 = 0;

int sum3 = 0;

int sum4 = 0;

int sum5 = 0;

int sum6 = 0;

for (int i =0;i<chars.Length;i++)

{

//Bigger squares

if(i<chars.Length/2)

sum1 += int.Parse(chars[i].ToString());

else

sum2 += int.Parse(chars[i].ToString());

}

//Smaller squares

sum3 = int.Parse(chars[0].ToString()) + int.Parse(chars[4].ToString())

+ int.Parse(chars[5].ToString()) + int.Parse(set[0].ToString());

sum4 = int.Parse(chars[4].ToString()) + int.Parse(chars[1].ToString())

+ int.Parse(set[0].ToString()) + int.Parse(chars[6].ToString());

sum5 = int.Parse(chars[5].ToString()) + int.Parse(set[0].ToString())

+ int.Parse(chars[2].ToString()) + int.Parse(chars[7].ToString());

sum6 = int.Parse(set[0].ToString()) + int.Parse(chars[6].ToString())

+ int.Parse(chars[7].ToString()) + int.Parse(chars[3].ToString());

if (sum1==sum2 && sum1==sum3 && sum1 == sum4&& sum1 == sum5&& sum1 == sum6)

{

//Console.WriteLine(assignment + "\_\_" + set[0]);

Console.WriteLine("Success");

Console.WriteLine("{0} {1} {2}", chars[0], chars[4], chars[1]);

Console.WriteLine("{0} {1} {2}", chars[5], set[0], chars[6]);

Console.WriteLine("{0} {1} {2}", chars[2], chars[7], chars[3]);

Console.WriteLine();

Console.Read();

System.Environment.Exit(0);

}

return;

}

if (assignment.Length >= 6 && !csp.Verify(assignment))

{

char[] toLast = new char[] { set[set.Length - 1] };

for (int i = 0; i < set.Length-1; i++)

assignment += set[i];

if (assignment.Length<8)

//Console.WriteLine(assignment);

RecursiveBacktracking(assignment,toLast, g);

}

else

{

for (int i = 0; i < set.Length; i++)

{

char n = set[i];

string newAsssignment = assignment + n;

char[] subset = new char[set.Length - 1];

int j = 0;

for (int k = 0; k < set.Length; k++)

{

if (set[k] != n)

{

subset[j++] = set[k];

}

}

//Console.WriteLine(newAsssignment);

RecursiveBacktracking(newAsssignment, subset, g);

}

}

}

}

}