МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО МГТУ «СТАНКИН»)

Институт
информационных технологий

Кафедра
информационных систем

Отчет по лабораторной работе №2

по дисциплине «**Интеллектуальные и экспертные системы**»
на тему: «Применение алгоритмов поиска для решения интеллектуальных задач»

Студент **Мустафаева П.М.**
группа ИДБ–22–06

подпись

Руководитель **Быстрикова В. А.**
старший преподаватель

подпись

Москва 2025 г.

ВВЕДЕНИЕ

Целью работы является изучение вопросов, связанных с формализацией представления интеллектуальных задач в пространстве состояний для дальнейшего поиска решений, выбор формы описания состояния, определяющий в общем случае сложность задания операторов задачи и, как следствие, размерность пространства состояний. Разработка программы, реализующей решение этих задач алгоритмами поиска в глубину и в ширину.

ХОД РАБОТЫ

Словесная постановка задачи о ханойских башнях:

Различные диски расположены на одном или нескольких стержнях, причем диски меньшего диаметра лежат на дисках большего диаметра. Требуется собрать пирамиду (переместить все диски на один стержень). Перемещать можно только самый верхний диск. Нельзя класть диск на диск меньшего диаметра.

Изменяемые параметры:

− количество дисков;

− количество стержней;

− исходная и заключительная конфигурация.

Состояния задачи описываются в виде списка массивов целых чисел.

Формализация задачи:

Задача – (Sн, Sц, O).

Состояние – (порядок дисков на первом стержне, порядок дисков на втором стержне, порядок дисков на третьем стержне).

Sн – (n…2,1; ;).

Sц – (; ; n…2,1).

O – Перемещение дисков (xyz; ;) -> (x ; yz ;).

Описание операторов и их воздействия на состояния: операторы – это правила перемещения дисков между стержнями, которые преобразуют одно состояние системы в другое. Каждый оператор:

1. Выбирает верхний диск с одного стержня (fromPole).
2. Помещает его на другой стержень (toPole), если это допустимо.
3. Создает новое состояние, отражающее изменение.

Каждый оператор можно представить в виде: Move(fromPole, toPole) (при условиях что fromPole не пуст; если fromPole не пуст, то верхний диск на нем должен быть больше перемещаемого диска; нельзя перемещать диск на тот же стержень).

Листинг 1 – Программный код

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace LR2_Hanoi_tower
{
   public partial class HanoiForm : Form
   {
      public HanoiForm()
      {
         InitializeComponent();
         btnUpdateGrid.Click += btnUpdateGrid_Click;
      }

      private void HanoiForm_Load(object sender, EventArgs e)
      {
         // Установите значения по умолчанию
         txtDiskCount.Text = "3";
         txtPoleCount.Text = "3";

         // Инициализация DataGridView
         if (int.TryParse(txtDiskCount.Text, out int diskCount) &&
            int.TryParse(txtPoleCount.Text, out int poleCount))
         {
            SetupDataGridViews(diskCount, poleCount);
            FillDefaultStartConfig(dgvStartConfig, diskCount); // Заполнение начальной
конфигурации
         }
      }
      private void FillDefaultStartConfig(DataGridView dgv, int diskCount)
      {
         // Очищаем все ячейки
         foreach (DataGridViewRow row in dgv.Rows)
         {
            foreach (DataGridViewCell cell in row.Cells)
            {
               cell.Value = null;
            }
         }

         // Заполняем первый стержень дисками (3, 2, 1 для diskCount=3)
         for (int row = 0; row < diskCount; row++)
         {
```

```
                dgv.Rows[row].Cells[0].Value = diskCount - row;
            }
        }

        private void SetupDataGridViews(int diskCount, int poleCount)
        {
            SetupDataGridView(dgvStartConfig, diskCount, poleCount);
            SetupDataGridView(dgvEndConfig, diskCount, poleCount);
        }

        private void SetupDataGridView(DataGridView dgv, int diskCount, int poleCount)
        {
            dgv.Rows.Clear();
            dgv.Columns.Clear();

            dgv.ColumnHeadersDefaultCellStyle = new DataGridViewCellStyle
            {
                Alignment = DataGridViewContentAlignment.MiddleCenter, // Выравнивание
по центру
            };

            // Добавляем столбцы (стержни)
            for (int i = 0; i < poleCount; i++)
            {
                dgv.Columns.Add(new DataGridViewTextBoxColumn
                {
                    Name = $"Pole{i}",
                    HeaderText = $"Стержень {i + 1}",
                    Width = 80,
                    DefaultCellStyle = new DataGridViewCellStyle
                    {
                        Alignment = DataGridViewContentAlignment.MiddleCenter
                    }
                });
            }

            // Добавляем строки (по количеству дисков)
            dgv.Rows.Add(diskCount);
        }

        private List<int[]> GetConfigFromDataGridView(DataGridView dgv, int diskCount)
        {
            List<int[]> config = new List<int[]>();

            foreach (DataGridViewColumn column in dgv.Columns)
            {
                List<int> disks = new List<int>();

                for (int row = 0; row < dgv.Rows.Count; row++)
                {
                    if (dgv[column.Index, row].Value != null &&
                        int.TryParse(dgv[column.Index, row].Value.ToString(), out int disk))
```

```
                    {
                        disks.Add(disk);
                    }
                }

                config.Add(disks.ToArray());
            }

            return config;
        }

        private void butBFS_Click(object sender, EventArgs e)
        {
            if (!int.TryParse(txtDiskCount.Text, out int diskCount) ||
        !int.TryParse(txtPoleCount.Text, out int poleCount) ||
        diskCount <= 0 || poleCount <= 0)
            {
                MessageBox.Show("Введите корректное количество дисков и стержней (положительные числа).");
                return;
            }

            List<int[]> startConfig = GetConfigFromDataGridView(dgvStartConfig, diskCount);
            List<int[]> targetConfig = GetConfigFromDataGridView(dgvEndConfig, diskCount);

            if (startConfig == null || targetConfig == null ||
                !ValidateConfiguration(startConfig, diskCount, poleCount) ||
                !ValidateConfiguration(targetConfig, diskCount, poleCount))
            {
                MessageBox.Show("Некорректные конфигурации.");
                return;
            }

            int steps;
            List<string> path = BFS(diskCount, poleCount, startConfig, targetConfig, out steps);

            listBoxPath.Items.Clear();

            if (path.Count > 0 && path[0] != "Путь не найден")
            {
                foreach (var step in path)
                {
                    listBoxPath.Items.Add(step);  // Просто добавляем строку как есть
                }
                txtSteps.Text = steps.ToString();
            }
            else
            {
                listBoxPath.Items.Add("Путь не найден");
```

```
                txtSteps.Text = "0";
            }
        }

        private void butDFS_Click(object sender, EventArgs e)
        {
            if (!int.TryParse(txtDiskCount.Text, out int diskCount) || !int.TryParse(txtPoleCount.Text, out int poleCount) || diskCount <= 0 || poleCount <= 0)
            {
                MessageBox.Show("Введите корректное количество дисков и стержней (положительные числа).");
                return;
            }

            // Получаем максимальную глубину (если задана)
            int maxDepth = -1;
            if (!string.IsNullOrEmpty(txtMaxDepth.Text))
            {
                if (!int.TryParse(txtMaxDepth.Text, out maxDepth) || maxDepth <= 0)
                {
                    MessageBox.Show("Максимальная глубина должна быть положительным числом");
                    return;
                }
            }

            List<int[]> startConfig = GetConfigFromDataGridView(dgvStartConfig, diskCount);
            List<int[]> targetConfig = GetConfigFromDataGridView(dgvEndConfig, diskCount);

            if (startConfig == null || targetConfig == null ||
                !ValidateConfiguration(startConfig, diskCount, poleCount) ||
                !ValidateConfiguration(targetConfig, diskCount, poleCount))
            {
                MessageBox.Show("Некорректные конфигурации.");
                return;
            }

            int steps;
            List<string> path = DFS(diskCount, poleCount, startConfig, targetConfig, out steps, maxDepth);

            listBoxPath.Items.Clear();

            if (path.Count > 0 && !path[0].StartsWith("Путь не найден"))
            {
                foreach (var state in path)
                {
                    listBoxPath.Items.Add(state);
                }
                txtSteps.Text = steps.ToString();
```

```
        }

        else
        {
            listBoxPath.Items.Add(path[0]);
            txtSteps.Text = "0";
        }
    }

    private bool ValidateConfiguration(List<int[]> config, int diskCount, int poleCount)
    {
        if (config == null || config.Count != poleCount)
            return false;

        // Проверка, что все диски присутствуют и нет лишних
        var allDisks = config.SelectMany(pole => pole).ToList();
        if (allDisks.Count != allDisks.Distinct().Count())
            return false; // Есть дубликаты

        if (allDisks.Any(d => d < 1 || d > diskCount))
            return false; // Диски вне допустимого диапазона

        // Проверка порядка дисков на каждом стержне
        foreach (var pole in config)
        {
            for (int i = 0; i < pole.Length - 1; i++)
            {
                if (pole[i] <= pole[i + 1])
                    return false; // Больший диск над меньшим
            }
        }

        return true;
    }

    // Преобразование состояния в строку для удобства сравнения и хранения
    private string StateToString(List<int[]> state)
    {
        return string.Join(" ; ", state.Select(s => string.Join(",", s)));
    }

    // Сравнение текущего состояния с целевым состоянием
    private bool CompareStates(List<int[]> state1, List<int[]> state2)
    {
        for (int i = 0; i < state1.Count; i++)
        {
            if (!state1[i].SequenceEqual(state2[i])) return false;
        }
        return true;
    }

    // Алгоритм поиска в ширину (BFS)
```

```
        private List<string> BFS(int diskCount, int poleCount, List<int[]> startConfig, List<int[]> targetConfig, out int steps)
        {
            Queue<List<int[]>> open = new Queue<List<int[]>>();
            HashSet<string> closed = new HashSet<string>();
            Dictionary<string, List<int[]>> parent = new Dictionary<string, List<int[]>>();

            open.Enqueue(startConfig);
            closed.Add(StateToString(startConfig));
            steps = 0;

            while (open.Count > 0)
            {
                List<int[]> currentState = open.Dequeue();
                steps++;

                if (CompareStates(currentState, targetConfig))
                {
                    return ConstructPath(parent, startConfig, currentState);
                }

                // Генерация новых состояний
                foreach (var nextState in GenerateNextStates(currentState, diskCount, poleCount))
                {
                    string nextStateStr = StateToString(nextState);
                    if (!closed.Contains(nextStateStr))
                    {
                        open.Enqueue(nextState);
                        closed.Add(nextStateStr);
                        parent[nextStateStr] = currentState;
                    }
                }
            }

            return new List<string> { "Путь не найден" };
        }

        // Генерация всех возможных новых состояний из текущего
        private List<List<int[]>> GenerateNextStates(List<int[]> currentState, int diskCount, int poleCount)
        {
            List<List<int[]>> nextStates = new List<List<int[]>>();

            // Логика для перемещения дисков и создания новых состояний
            for (int fromPole = 0; fromPole < poleCount; fromPole++)
            {
                if (currentState[fromPole].Length == 0) continue;  // Если стержень пустой, пропускаем

                for (int toPole = 0; toPole < poleCount; toPole++)
                {
```

```
                    if (fromPole == toPole) continue; // Если это тот же стержень, пропускаем

                    var newState = CloneState(currentState);
                    int disk = newState[fromPole].Last();
                    if (newState[toPole].Length == 0 || newState[toPole].Last() > disk)
                    {
                        newState[toPole] = newState[toPole].Concat(new[] { disk }).ToArray();
                        newState[fromPole] = newState[fromPole].Take(newState[fromPole].Length - 1).ToArray();
                        nextStates.Add(newState);
                    }
                }
            }

            return nextStates;
        }

        // Восстановление пути от начальной конфигурации до целевой
        private List<string> ConstructPath(Dictionary<string, List<int[]>> parent, List<int[]> startConfig, List<int[]> targetConfig)
        {
            List<string> path = new List<string>();
            string current = StateToString(targetConfig);

            while (current != StateToString(startConfig))
            {
                path.Add(current);
                current = StateToString(parent[current]);
            }

            path.Add(StateToString(startConfig));
            path.Reverse();
            return path;
        }

        // Клонирование состояния
        private List<int[]> CloneState(List<int[]> state)
        {
            return state.Select(pole => pole.ToArray()).ToList();
        }

        // Алгоритм поиска в глубину (DFS)
        private List<string> DFS(int diskCount, int poleCount, List<int[]> startConfig, List<int[]> targetConfig, out int steps, int maxDepth = -1)
        {
            // Используем стек с учетом глубины
            Stack<(List<int[]> state, int depth)> open = new Stack<(List<int[]>, int)>();
            HashSet<string> closed = new HashSet<string>();
            Dictionary<string, List<int[]>> parent = new Dictionary<string, List<int[]>>();

            open.Push((startConfig, 0));
            closed.Add(StateToString(startConfig));
```

```
            steps = 0;

            while (open.Count > 0)
            {
                var (currentState, currentDepth) = open.Pop();
                steps++;

                if (CompareStates(currentState, targetConfig))
                {
                    return ConstructPath(parent, startConfig, currentState);
                }

                // Проверяем ограничение глубины (если maxDepth != -1)
                if (maxDepth != -1 && currentDepth >= maxDepth)
                    continue;

                foreach (var nextState in GenerateNextStates(currentState, diskCount,
poleCount))
                {
                    string nextStateStr = StateToString(nextState);
                    if (!closed.Contains(nextStateStr))
                    {
                        open.Push((nextState, currentDepth + 1));
                        closed.Add(nextStateStr);
                        parent[nextStateStr] = currentState;
                    }
                }
            }

            return new List<string> { "Путь не найден (возможно, превышена
максимальная глубина)" };
        }

        private void btnUpdateGrid_Click(object sender, EventArgs e)
        {
            if (!int.TryParse(txtDiskCount.Text, out int diskCount) || diskCount <= 0)
            {
                MessageBox.Show("Введите корректное количество дисков
(положительное число)");
                return;
            }

            if (!int.TryParse(txtPoleCount.Text, out int poleCount) || poleCount <= 0)
            {
                MessageBox.Show("Введите корректное количество стержней
(положительное число)");
                return;
            }

            // Обновляем обе DataGridView
            SetupDataGridView(dgvStartConfig, diskCount, poleCount);
            SetupDataGridView(dgvEndConfig, diskCount, poleCount);
```

```
            // Заполняем начальную конфигурацию (первый стержень с дисками)
            FillDefaultStartConfig(dgvStartConfig, diskCount);
        }
    }
}
```

Результаты поиска в ширину, в глубину и в глубину с заданной максимальной глубиной представлены на рис. 1-3.

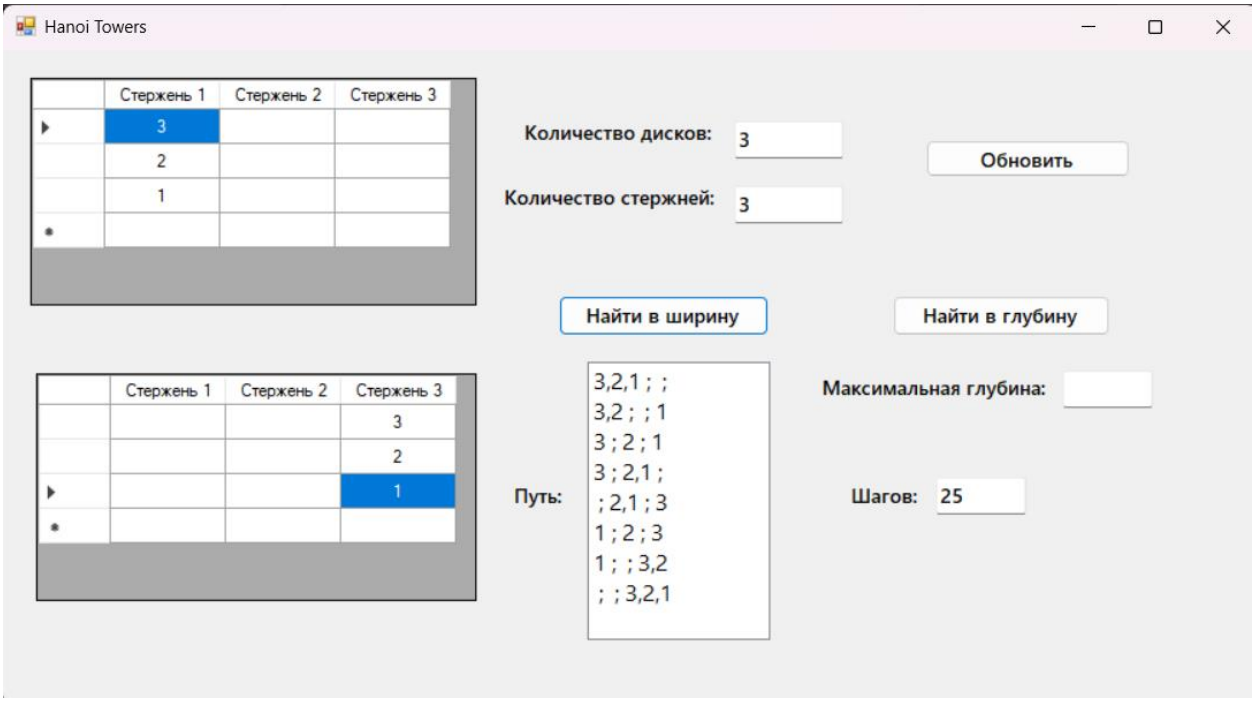

Рис. 1 Экранная форма с результатом работы алгоритма поиска в ширину

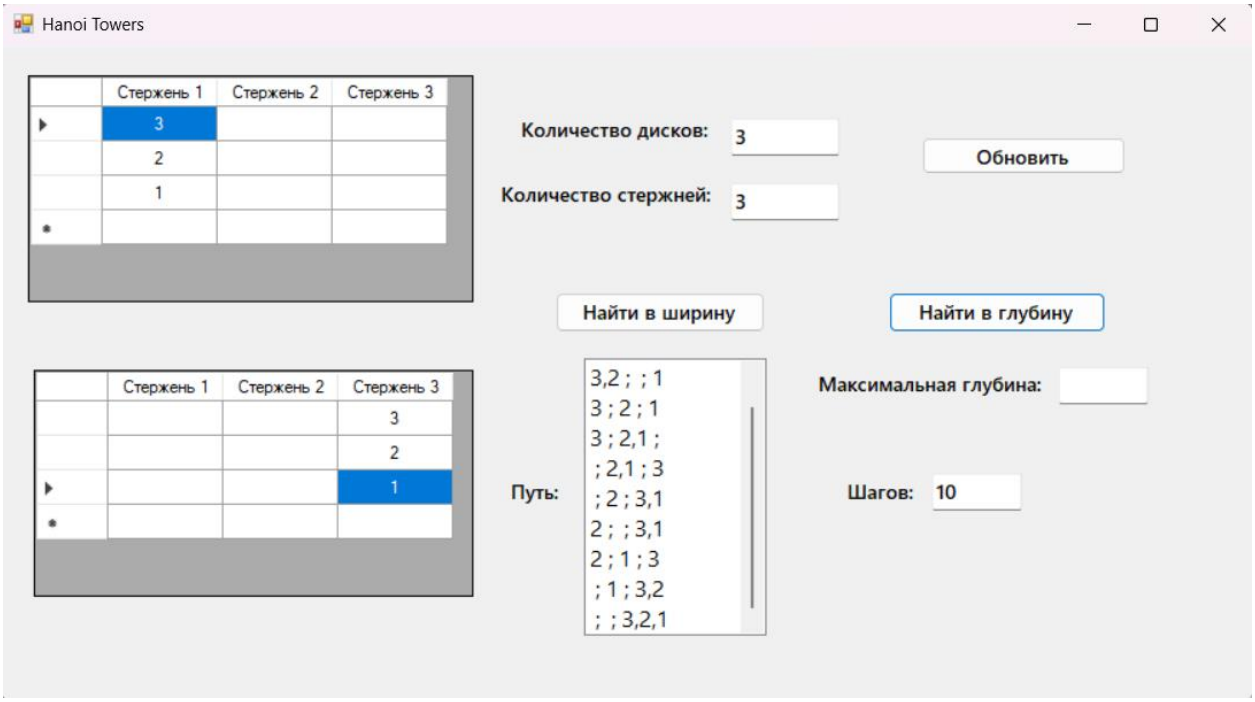

Рис. 2 Экранная форма с результатом работы алгоритма поиска в глубину

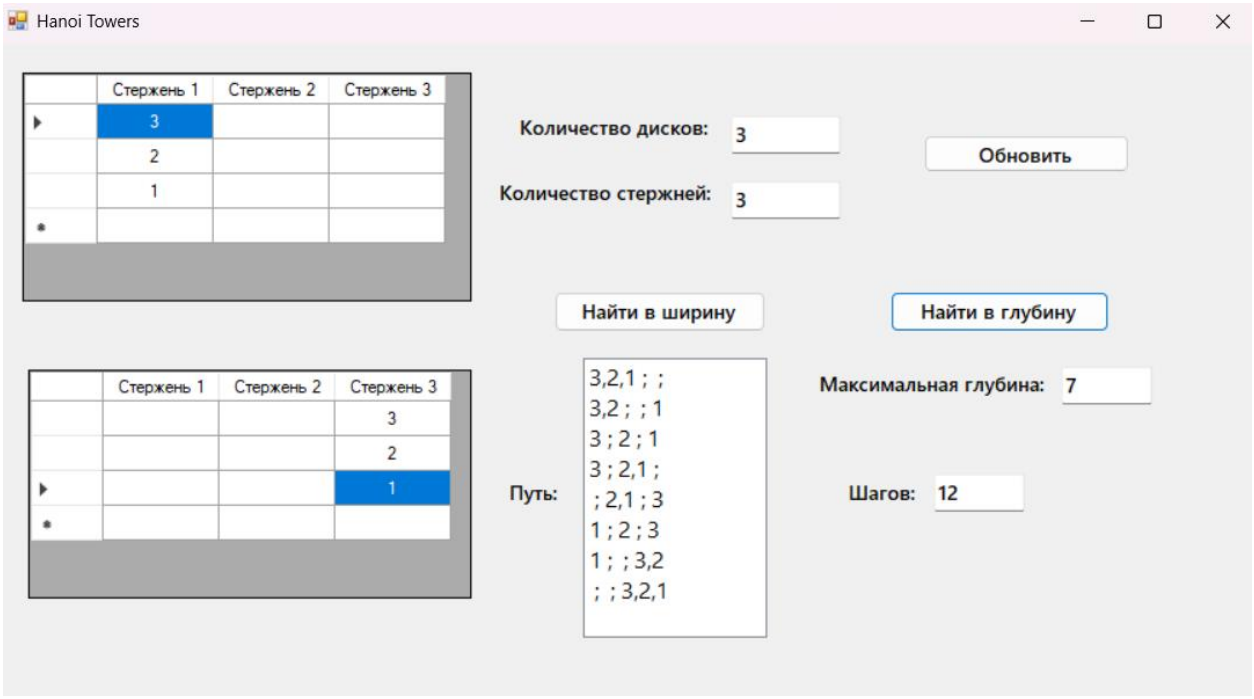

Рис. 3 Экранная форма с результатом работы алгоритма поиска в глубину с заданной максимальной глубиной

ВЫВОД

В ходе выполнения работы была разработана программа для реализации алгоритма поиска в глубину и в ширину для задачи о ханойских башнях. Оба алгоритма были визуализированы с помощью Windows Forms, что позволило наглядно продемонстрировать их работу и сравнить эффективность.