**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 2 з дисципліни

«Проектування алгоритмів»

**«Неінформативний, інформативний та локальний пошук»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-14 Громов Владислав*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[**1**](#_30j0zll) **Мета лабораторної роботи 3**

[**2**](#_3znysh7) **ЗаВдання 4**

[**3**](#_2et92p0) **Виконання 8**

[3.1](#_tyjcwt) Псевдокод алгоритмів 8

[3.2](#_3dy6vkm) Програмна реалізація 8

[*3.2.1*](#_1t3h5sf) *Вихідний код 8*

[*3.2.2*](#_4d34og8) *Приклади роботи 8*

[3.3](#_2s8eyo1) Дослідження алгоритмів 8

[**Висновок 11**](#_17dp8vu)

[**Критерії оцінювання 12**](#_3rdcrjn)

# Мета лабораторної роботи

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

# ЗаВдання

Записати алгоритм розв’язання задачі у вигляді псевдокоду, відповідно до варіанту (таблиця 2.1).

Реалізувати програму, яка розв’язує поставлену задачу згідно варіанту (таблиця 2.1) за допомогою алгоритму неінформативного пошуку **АНП**, алгоритму інформативного пошуку **АІП,** що використовує задану евристичну функцію Func, або алгоритму локального пошуку **АЛП та бектрекінгу,** що використовує задану евристичну функцію Func.

Програму реалізувати на довільній мові програмування.

**Увага!** Алгоритм неінформативного пошуку **АНП,** реалізовується за принципом «AS IS», тобто так, як є, без додаткових модифікацій (таких як перевірка циклів, наприклад).

Провести серію експериментів для вивчення ефективності роботи алгоритмів. Кожний експеримент повинен відрізнятись початковим станом. Серія повинна містити не менше 20 експериментів для кожного алгоритму. Початковий стан зафіксувати у таблиці експериментів. За проведеними серіями необхідно визначити:

* середню кількість етапів (кроків), які знадобилось для досягнення розв’язку (ітерації);
* середню кількість випадків, коли алгоритм потрапляв в глухий кут (не міг знайти оптимальний розв’язок) – якщо таке можливе;
* середню кількість згенерованих станів під час пошуку;
* середню кількість станів, що зберігаються в пам’яті під час роботи програми.

Передбачити можливість обмеження виконання програми за часом (30 хвилин) та використання пам’яті (1 Гб).

**Використані позначення:**

* **8-ферзів** – Задача про вісім ферзів полягає в такому розміщенні восьми ферзів на шахівниці, що жодна з них не ставить під удар один одного. Тобто, вони не повинні стояти в одній вертикалі, горизонталі чи діагоналі.
* **8-puzzle** – гра, що складається з 8 однакових квадратних пластинок з нанесеними числами від 1 до 8. Пластинки поміщаються в квадратну коробку, довжина сторони якої в три рази більша довжини сторони пластинок, відповідно в коробці залишається незаповненим одне квадратне поле. Мета гри – переміщаючи пластинки по коробці досягти впорядковування їх по номерах, бажано зробивши якомога менше переміщень.
* **Лабіринт** – задача пошуку шляху у довільному лабіринті від початкової точки до кінцевої з можливими випадками відсутності шляху. Структура лабіринту зчитується з файлу, або генерується програмою.
* **LDFS** – Пошук вглиб з обмеженням глибини.
* **BFS** – Пошук вшир.
* **IDS** – Пошук вглиб з ітеративним заглибленням.
* **A\*** – Пошук А\*.
* **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
* **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь А може стояти на одній лінії з ферзем В, проте між ними стоїть ферзь С; тому А не б’є В).
* **F2** – кількість пар ферзів, які б’ють один одного без урахування видимості.
* **H1** – кількість фішок, які не стоять на своїх місцях.
* **H2** – Манхетенська відстань.
* **H3** – Евклідова відстань.
* **COLOR** – Задача розфарбування карти самостійно обраної країни, не менше 20 регіонів (областей). Необхідно розфарбувати карту не більше ніж у 4 різні кольори. Мається на увазі приписування кожному регіону власного кольору так, щоб кольори сусідніх регіонів відрізнялись. Використовувати евристичну функцію, яка повертає кількість пар суміжних вузлів, що мають однаковий колір (тобто кількість конфліктів). Реалізувати алгоритм пошуку із поверненнями (backtracking) для розв’язання поставленої задачі. Для підвищення швидкодії роботи алгоритму використати евристичну функцію, а початковим станом вважати випадкову вершину.
* **HILL –** Пошук зі сходженням на вершину з використанням із використанням руху вбік (на 100 кроків) та випадковим перезапуском (кількість необхідних разів запуску визначити самостійно).
* **ANNEAL** – Локальний пошук із симуляцією відпалу. Робоча характеристика – залежність температури Т від часу роботи алгоритму t. Можна розглядати лінійну залежність: T = 1000 - k∙t, де k – змінний коефіцієнт.
* **BEAM** – Локальний променевий пошук. Робоча характеристика – кількість променів k. Експерименти проводи із кількістю променів від 2 до 21.
* **MRV** – евристика мінімальної кількості значень;
* **DGR** – ступенева евристика.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Задача** | **АНП** | **АІП** | **АЛП** | **Func** |
| 1 | Лабіринт | LDFS | A\* |  | H2 |
| 2 | Лабіринт | LDFS | RBFS |  | H3 |
| 3 | Лабіринт | BFS | A\* |  | H2 |
| 4 | Лабіринт | BFS | RBFS |  | H3 |
| 5 | Лабіринт | IDS | A\* |  | H2 |
| 6 | Лабіринт | IDS | RBFS |  | H3 |
| 7 | 8-ферзів | LDFS | A\* |  | F1 |
| 8 | 8-ферзів | LDFS | A\* |  | F2 |
| 9 | 8-ферзів | LDFS | RBFS |  | F1 |
| 10 | 8-ферзів | LDFS | RBFS |  | F2 |
| 11 | 8-ферзів | BFS | A\* |  | F1 |
| 12 | 8-ферзів | BFS | A\* |  | F2 |
| 13 | 8-ферзів | BFS | RBFS |  | F1 |
| 14 | 8-ферзів | BFS | RBFS |  | F2 |
| 15 | 8-ферзів | IDS | A\* |  | F1 |
| 16 | 8-ферзів | IDS | A\* |  | F2 |
| 17 | 8-ферзів | IDS | RBFS |  | F1 |
| 18 | Лабіринт | LDFS | A\* |  | H3 |
| 19 | 8-puzzle | LDFS | A\* |  | H1 |
| 20 | 8-puzzle | LDFS | A\* |  | H2 |
| 21 | 8-puzzle | LDFS | RBFS |  | H1 |
| 22 | 8-puzzle | LDFS | RBFS |  | H2 |
| 23 | 8-puzzle | BFS | A\* |  | H1 |
| 24 | 8-puzzle | BFS | A\* |  | H2 |
| 25 | 8-puzzle | BFS | RBFS |  | H1 |
| 26 | 8-puzzle | BFS | RBFS |  | H2 |
| 27 | Лабіринт | BFS | A\* |  | H3 |
| 28 | 8-puzzle | IDS | A\* |  | H2 |
| 29 | 8-puzzle | IDS | RBFS |  | H1 |
| 30 | 8-puzzle | IDS | RBFS |  | H2 |
| 31 | COLOR |  |  | HILL | MRV |
| 32 | COLOR |  |  | ANNEAL | MRV |
| 33 | COLOR |  |  | BEAM | MRV |
| 34 | COLOR |  |  | HILL | DGR |
| 35 | COLOR |  |  | ANNEAL | DGR |
| 36 | COLOR |  |  | BEAM | DGR |

# Виконання

## Псевдокод алгоритмів

**AStar**

public class Node

{

public Node? parent { get; }

public Cell cell { get; }

public int f { get; }

public int h { get; }

public int g { get; }

public Node(Node parent, Cell cell, int h, int g)

{

this.parent = parent;

this.cell = cell;

f = h + g;

this.h = h;

this.g = g;

}

}

private static bool IsCellHere(List<Node> l, Cell c)

{

return l.Any(n => n.cell.x == c.x && n.cell.y == c.y);

}

private static Node? FindWay(Cell[,] maze, int sX, int sY, int eX, int eY)

{

var open = new List<Node>();

var close = new List<Node>();

var ss = new Node(null, maze[sX, sY], 0, 0);

var deadEnds = 0;

var states = 0;

var iterations = 0;

do

{

if (open.Count > 0)

{

var min = Int32.MaxValue;

var inode = 0;

for (var i = 0; i < open.Count; i++)

{

if (open[i].f < min)

{

min = open[i].f;

inode = i;

}

}

//Console.WriteLine($"{ss.g} {ss.cell.x} {ss.cell.y}");

ss = open[inode];

open.RemoveAt(inode);

}

iterations++;

var c = false;

close.Add(ss);

if (!maze[ss.cell.x, ss.cell.y + 1].IsWall && !IsCellHere(close, maze[ss.cell.x, ss.cell.y + 1]))

{

c = true;

var node = new Node(ss, maze[ss.cell.x, ss.cell.y + 1], ss.h + 10,

Math.Abs(eX - ss.cell.x) + Math.Abs(eY - (ss.cell.y + 1)));

if (node.g == 0)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {close.Count}");

return node;

}

open.Add(node);

}

if (!maze[ss.cell.x, ss.cell.y - 1].IsWall && !IsCellHere(close, maze[ss.cell.x, ss.cell.y - 1]))

{

c = true;

var node = new Node(ss, maze[ss.cell.x, ss.cell.y - 1], ss.h + 10,

Math.Abs(eX - ss.cell.x) + Math.Abs(eY - (ss.cell.y - 1)));

if (node.g == 0)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {close.Count}");

return node;

}

open.Add(node);

}

if (!maze[ss.cell.x + 1, ss.cell.y].IsWall && !IsCellHere(close, maze[ss.cell.x + 1, ss.cell.y]))

{

c = true;

var node = new Node(ss, maze[ss.cell.x + 1, ss.cell.y], ss.h + 10,

Math.Abs(eX - (ss.cell.x + 1)) + Math.Abs(eY - ss.cell.y));

if (node.g == 0)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {close.Count}");

return node;

}

open.Add(node);

}

if (!maze[ss.cell.x - 1, ss.cell.y].IsWall && !IsCellHere(close, maze[ss.cell.x - 1, ss.cell.y]))

{

c = true;

var node = new Node(ss, maze[ss.cell.x - 1, ss.cell.y], ss.h + 10,

Math.Abs(eX - (ss.cell.x - 1)) + Math.Abs(eY - ss.cell.y));

if (node.g == 0)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {close.Count}");

return node;

}

open.Add(node);

}

if (!c) deadEnds++;

if (c) states++;

} while (open.Count != 0) ;

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {close.Count}");

Console.WriteLine("Лабиринт невозможно пройти!");

return null;

}

public static Cell[,] AStarAlg(Cell[,] maze, int sX, int sY, int eX, int eY)

{

var x = maze;

var n = FindWay(maze, sX, sY, eX, eY);

while (n != null)

{

n.cell.IsWay = true;

x[n.cell.x, n.cell.y] = n.cell;

n = n.parent;

}

return x;

}

**IDS**

public class Node

{

public Node? parent { get; }

public Cell cell { get; }

public Node(Node parent, Cell cell)

{

this.parent = parent;

this.cell = cell;

}

}

private static bool IsCellHere(List<Node> l, Cell c)

{

return l.Any(n => n.cell.x == c.x && n.cell.y == c.y);

}

private static Node? FindWay(Cell[,] maze, int sX, int sY, int eX, int eY)

{

var open = new List<Node>();

var node = new Node(null, maze[sX, sY]);

open.Add(node);

var deadEnds = 0;

var states = 0;

var iterations = 0;

var inMemoryStates = 0;

do

{

iterations++;

var c = false;

var temp = new List<Node>();

var num = open.Count;

foreach (var t in open)

{

var cc = false;

var x = t.cell.x;

var y = t.cell.y;

var xp = t.cell.x + 1;

var xm = t.cell.x - 1;

var yp = t.cell.y + 1;

var ym = t.cell.y - 1;

if (!maze[xp, y].IsWall && !IsCellHere(temp, maze[xp, y]))

{

c = true;

cc = true;

if (xp == eX && y == eY)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {inMemoryStates}");

return new Node(t, maze[xp, y]);

}

temp.Add(new Node(t, maze[xp, y]));

}

if (!maze[xm, y].IsWall && !IsCellHere(temp, maze[xm, y]))

{

c = true;

cc = true;

if (xm == eX && y == eY)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {inMemoryStates}");

return new Node(t, maze[xm, y]);

}

temp.Add(new Node(t, maze[xm, y]));

}

if (!maze[x, yp].IsWall && !IsCellHere(temp, maze[x, yp]))

{

c = true;

cc = true;

if (x == eX && yp == eY)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {inMemoryStates}");

return new Node(t, maze[x, yp]);

}

temp.Add(new Node(t, maze[x, yp]));

}

if (!maze[x, ym].IsWall && !IsCellHere(temp, maze[x, ym]))

{

c = true;

cc = true;

if (x == eX && ym == eY)

{

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {inMemoryStates}");

return new Node(t, maze[x, ym]);

}

temp.Add(new Node(t, maze[x, ym]));

}

open = new List<Node>(temp);

if (!cc) deadEnds++;

}

inMemoryStates += temp.Count;

if (c) states++;

iterations++;

} while (open.Count != 0);

Console.WriteLine($"Итераций: {iterations}");

Console.WriteLine($"Состояний: {states}");

Console.WriteLine($"Глухих углов: {deadEnds}");

Console.WriteLine($"Состояний в памяти: {inMemoryStates}");

Console.WriteLine("Лабиринт невозможно пройти!");

return null;

}

public static Cell[,] IdsAlg(Cell[,] maze, int sX, int sY, int eX, int eY)

{

var x = maze;

var n = FindWay(maze, sX, sY, eX, eY);

while (n != null)

{

n.cell.IsWay = true;

x[n.cell.x, n.cell.y] = n.cell;

n = n.parent;

}

return x;

}

## Програмна реалізація

### Вихідний код

**Program.cs**

using static ConsoleApp3.MazeGen;  
using static ConsoleApp3.AStar;  
using static ConsoleApp3.IDS;  
  
namespace ConsoleApp3  
{  
 class Program  
 {  
 public static void Main(string[] args)  
 {  
 var path = Directory  
 .GetParent(Directory.GetParent(Directory.GetParent(Environment.CurrentDirectory)  
 .ToString())  
 .ToString())  
 + @"\mazes\maze20.txt";  
   
 var maze = MatrixGen(path);  
  
 Console.WriteLine();  
 Console.WriteLine("Лабиринт без просчитангого пути:");  
 PrintMaze(maze);  
  
 var sX = 1; *//Convert.ToInt32(Console.ReadLine());* var sY = 1; *//Convert.ToInt32(Console.ReadLine());* var eX = 39; *//Convert.ToInt32(Console.ReadLine());* var eY = 39; *//Convert.ToInt32(Console.ReadLine());* var maze1 = AStarAlg(maze, sX, sY, eX, eY);  
 Console.WriteLine();  
 var maze2 = IdsAlg(maze, sX, sY, eX, eY);  
 if (maze1 != null)  
 {  
 Console.WriteLine("Лабиринт после нахождения пути IDS алгоритмом:");  
 PrintMaze(maze1);  
 }  
   
 if (maze2 != null)  
 {  
 Console.WriteLine("Лабиринт после нахождения пути A\* алгоритмом:");  
 PrintMaze(maze2);  
 }  
   
 }  
 }  
}

**MazeGen.cs**

namespace ConsoleApp3;  
  
public class MazeGen  
{  
 public static Cell[,] MatrixGen(string path)  
 {  
 var ss = ConvertMaze(path);  
 var maze = new Cell[ss[0].Length, ss.Length];  
 Console.WriteLine($"Размер лабиринта с учетом стен: {maze.GetLength(0)}x{maze.GetLength(1)}");  
 for (int i = 0; i < ss.Length; i++)  
 {  
 int k = 0;  
 for (int j = 0; j < ss[i].Length; j++)  
 {  
 if (ss[i][j].Equals('#')) maze[i, j] = new Cell(i, j, true);  
 if (ss[i][j].Equals('.')) maze[i, j] = new Cell(i, j, false);  
 }  
 }  
  
 return maze;  
 }  
  
 private static string[] ConvertMaze(string path)  
 {  
 var s = File.ReadAllLines(path);  
 var newStrings = new string[s.Length];  
 for (var i = 0; i < s.Length; i++)  
 {  
 var str = "";  
 var k = 0;  
 for (int j = 0; j < s[i].Length; j = j + k)  
 {  
 k = k == 2 ? 1 : 2;  
 str += s[i][j];  
 }  
  
 newStrings[i] = str;  
 }  
  
 return newStrings;  
 }  
 public static void PrintMaze(Cell[,] maze)  
 {  
 for (int i = 0; i < maze.GetLength(0); i++)  
 {  
 for (int j = 0; j < maze.GetLength(1); j++)  
 {  
 Console.Write(maze[i, j]);  
 }  
  
 Console.WriteLine();  
 }  
 }  
}

**Cell.cs**

namespace ConsoleApp3;  
  
public class Cell  
{  
 public int x, y;  
 public bool IsWall{get; set;}  
 public bool IsWay = false;  
  
  
 public Cell(int x, int y, bool isWall)  
 {  
 this.x = x;  
 this.y = y;  
 IsWall = isWall;  
 }  
   
   
  
  
 public override string ToString()  
 {  
 if (IsWay) return "##";  
 if (!IsWall) return " ";  
 return "██";  
 }  
}

**AStar.cs**

namespace ConsoleApp3;  
  
public class AStar  
{  
 public class Node  
 {  
 public Node? parent { get; }  
 public Cell cell { get; }  
 public int f { get; }  
 public int h { get; }  
 public int g { get; }  
  
 public Node(Node parent, Cell cell, int h, int g)  
 {  
 this.parent = parent;  
 this.cell = cell;  
 f = h + g;  
 this.h = h;  
 this.g = g;  
 }  
 }  
  
 private static bool IsCellHere(List<Node> l, Cell c)  
 {  
 return l.Any(n => n.cell.x == c.x && n.cell.y == c.y);  
 }  
  
 private static Node? FindWay(Cell[,] maze, int sX, int sY, int eX, int eY)  
 {  
 var open = new List<Node>();  
 var close = new List<Node>();  
 var ss = new Node(null, maze[sX, sY], 0, 0);  
 var deadEnds = 0;  
 var states = 0;  
 var iterations = 0;  
   
 do  
 {  
 if (open.Count > 0)  
 {  
 var min = Int32.**MaxValue**;  
 var inode = 0;  
 for (var i = 0; i < open.Count; i++)  
 {  
 if (open[i].f < min)  
 {  
 min = open[i].f;  
 inode = i;  
 }  
 }  
  
 *//Console.WriteLine($"{ss.g} {ss.cell.x} {ss.cell.y}");* ss = open[inode];  
 open.RemoveAt(inode);  
 }  
 iterations++;  
 var c = false;  
 close.Add(ss);  
 if (!maze[ss.cell.x, ss.cell.y + 1].IsWall && !IsCellHere(close, maze[ss.cell.x, ss.cell.y + 1]))  
 {  
 c = true;  
 var node = new Node(ss, maze[ss.cell.x, ss.cell.y + 1], ss.h + 10,  
 Math.Abs(eX - ss.cell.x) + Math.Abs(eY - (ss.cell.y + 1)));  
 if (node.g == 0)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {close.Count}");  
 return node;  
 }  
   
 open.Add(node);  
 }  
   
 if (!maze[ss.cell.x, ss.cell.y - 1].IsWall && !IsCellHere(close, maze[ss.cell.x, ss.cell.y - 1]))  
 {  
 c = true;  
 var node = new Node(ss, maze[ss.cell.x, ss.cell.y - 1], ss.h + 10,  
 Math.Abs(eX - ss.cell.x) + Math.Abs(eY - (ss.cell.y - 1)));  
   
 if (node.g == 0)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {close.Count}");  
 return node;  
 }  
 open.Add(node);  
 }  
   
 if (!maze[ss.cell.x + 1, ss.cell.y].IsWall && !IsCellHere(close, maze[ss.cell.x + 1, ss.cell.y]))  
 {  
 c = true;  
 var node = new Node(ss, maze[ss.cell.x + 1, ss.cell.y], ss.h + 10,  
 Math.Abs(eX - (ss.cell.x + 1)) + Math.Abs(eY - ss.cell.y));  
   
 if (node.g == 0)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {close.Count}");  
 return node;  
 }  
 open.Add(node);  
 }  
   
 if (!maze[ss.cell.x - 1, ss.cell.y].IsWall && !IsCellHere(close, maze[ss.cell.x - 1, ss.cell.y]))  
 {  
 c = true;  
 var node = new Node(ss, maze[ss.cell.x - 1, ss.cell.y], ss.h + 10,  
 Math.Abs(eX - (ss.cell.x - 1)) + Math.Abs(eY - ss.cell.y));  
 if (node.g == 0)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {close.Count}");  
 return node;  
 }  
 open.Add(node);  
 }  
   
 if (!c) deadEnds++;  
 if (c) states++;  
   
 } while (open.Count != 0) ;  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {close.Count}");  
 Console.WriteLine("Лабиринт невозможно пройти!");  
 return null;  
 }  
  
 public static Cell[,] AStarAlg(Cell[,] maze, int sX, int sY, int eX, int eY)  
 {  
 var x = maze;  
 var n = FindWay(maze, sX, sY, eX, eY);  
 while (n != null)  
 {  
 n.cell.IsWay = true;  
 x[n.cell.x, n.cell.y] = n.cell;  
 n = n.parent;  
 }  
  
 return x;  
 }  
}

**IDS.cs**

namespace ConsoleApp3;  
  
public class IDS  
{  
 public class Node  
 {  
 public Node? parent { get; }  
 public Cell cell { get; }  
  
 public Node(Node parent, Cell cell)  
 {  
 this.parent = parent;  
 this.cell = cell;  
 }  
 }  
 private static bool IsCellHere(List<Node> l, Cell c)  
 {  
 return l.Any(n => n.cell.x == c.x && n.cell.y == c.y);  
 }  
  
 private static Node? FindWay(Cell[,] maze, int sX, int sY, int eX, int eY)  
 {  
 var open = new List<Node>();  
 var node = new Node(null, maze[sX, sY]);  
 open.Add(node);  
 var deadEnds = 0;  
 var states = 0;  
 var iterations = 0;  
 var inMemoryStates = 0;  
 do  
 {  
 iterations++;  
 var c = false;  
   
 var temp = new List<Node>();  
 var num = open.Count;  
 foreach (var t in open)  
 {  
 var cc = false;  
 var x = t.cell.x;  
 var y = t.cell.y;  
 var xp = t.cell.x + 1;  
 var xm = t.cell.x - 1;  
 var yp = t.cell.y + 1;  
 var ym = t.cell.y - 1;  
   
 if (!maze[xp, y].IsWall && !IsCellHere(temp, maze[xp, y]))  
 {  
 c = true;  
 cc = true;  
 if (xp == eX && y == eY)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {inMemoryStates}");  
 return new Node(t, maze[xp, y]);  
 }  
 temp.Add(new Node(t, maze[xp, y]));  
 }  
 if (!maze[xm, y].IsWall && !IsCellHere(temp, maze[xm, y]))  
 {  
 c = true;  
 cc = true;  
 if (xm == eX && y == eY)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {inMemoryStates}");  
 return new Node(t, maze[xm, y]);  
 }  
 temp.Add(new Node(t, maze[xm, y]));  
 }  
 if (!maze[x, yp].IsWall && !IsCellHere(temp, maze[x, yp]))  
 {  
 c = true;  
 cc = true;  
 if (x == eX && yp == eY)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {inMemoryStates}");  
 return new Node(t, maze[x, yp]);  
 }  
 temp.Add(new Node(t, maze[x, yp]));  
 }  
 if (!maze[x, ym].IsWall && !IsCellHere(temp, maze[x, ym]))  
 {  
 c = true;  
 cc = true;  
 if (x == eX && ym == eY)  
 {  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {inMemoryStates}");  
 return new Node(t, maze[x, ym]);  
 }  
 temp.Add(new Node(t, maze[x, ym]));  
 }  
 open = new List<Node>(temp);  
 if (!cc) deadEnds++;  
 }  
  
 inMemoryStates += temp.Count;  
 if (c) states++;  
 iterations++;  
  
 } while (open.Count != 0);  
 Console.WriteLine($"Итераций: {iterations}");  
 Console.WriteLine($"Состояний: {states}");  
 Console.WriteLine($"Глухих углов: {deadEnds}");  
 Console.WriteLine($"Состояний в памяти: {inMemoryStates}");  
 Console.WriteLine("Лабиринт невозможно пройти!");  
 return null;  
 }  
   
 public static Cell[,] IdsAlg(Cell[,] maze, int sX, int sY, int eX, int eY)  
 {  
 var x = maze;  
 var n = FindWay(maze, sX, sY, eX, eY);  
 while (n != null)  
 {  
 n.cell.IsWay = true;  
 x[n.cell.x, n.cell.y] = n.cell;  
 n = n.parent;  
 }  
  
 return x;  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для різних алгоритмів пошуку.

Рисунок 3.1 – Алгоритм IDS

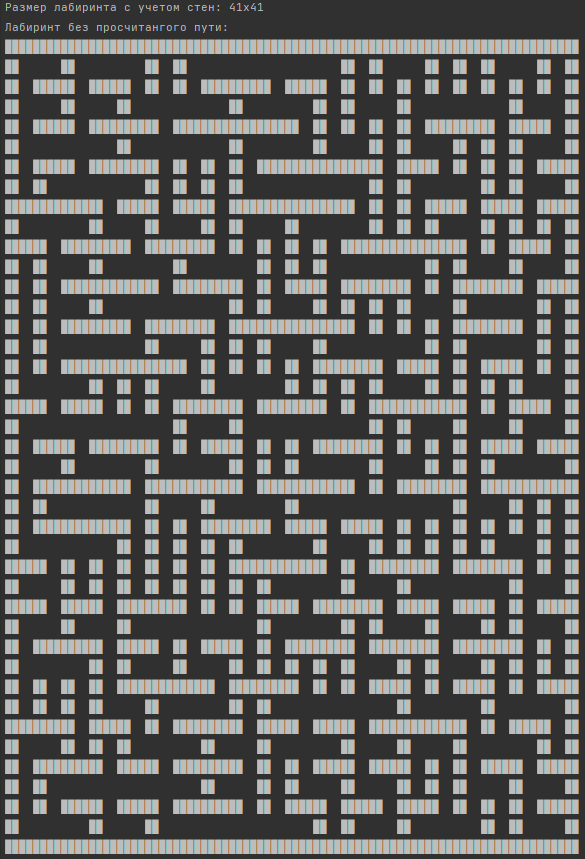
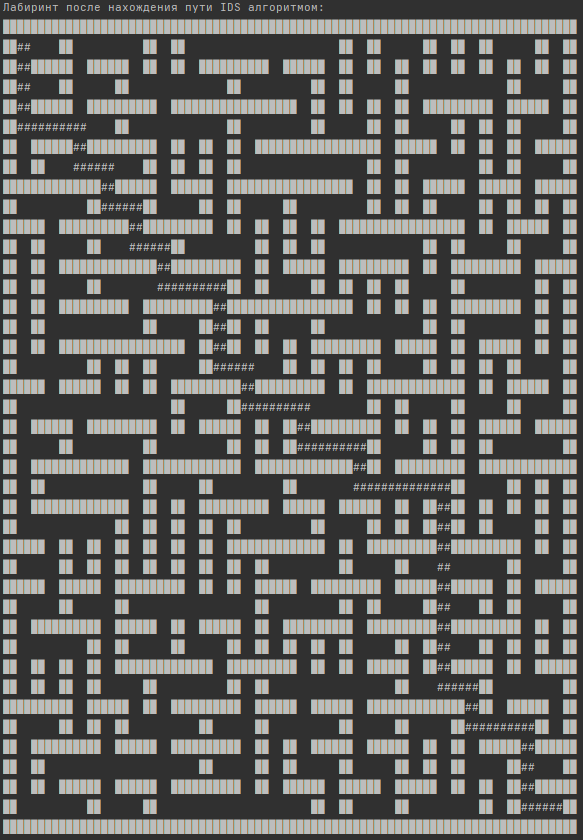
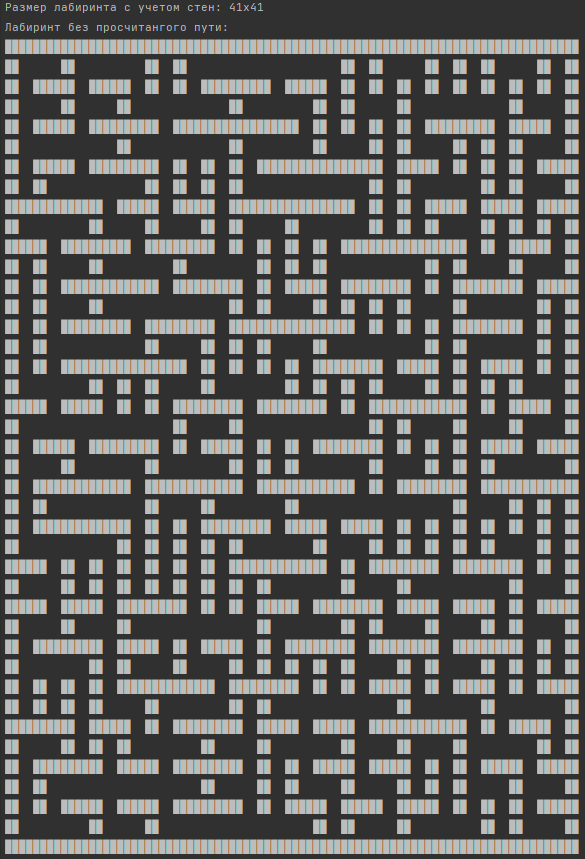
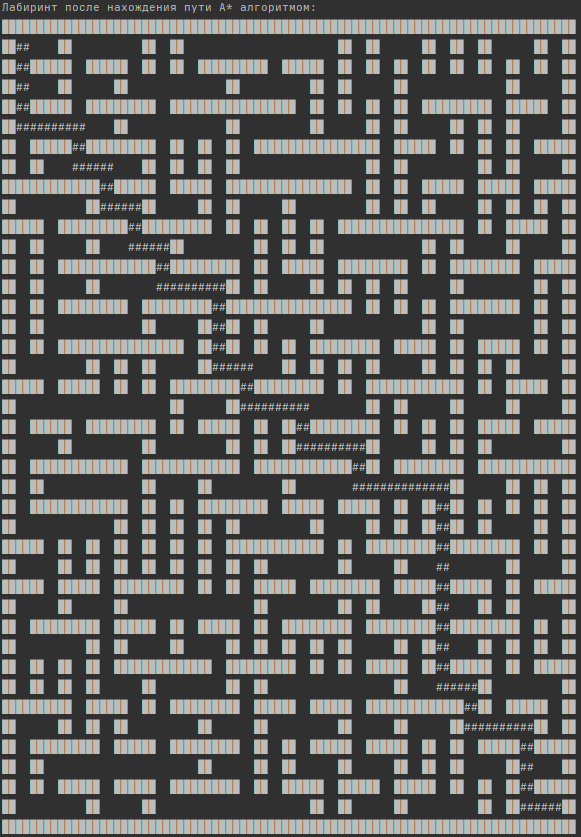
 

Рисунок 3.2 – Алгоритм A\*

## 

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму IDS, задачі Лабіринт для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму IDS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | 207 | 1664 | 103 | 12460 |
| Стан 2 | 223 | 2451 | 111 | 17474 |
| Стан 3 | 207 | 2165 | 103 | 17190 |
| Стан 4 | 199 | 2123 | 99 | 15355 |
| Стан 5 | 303 | 3053 | 151 | 22661 |
| Стан 6 | 223 | 2527 | 111 | 17687 |
| Стан 7 | 287 | 3290 | 143 | 25262 |
| Стан 8 | 183 | 903 | 91 | 7790 |
| Стан 9 | 183 | 1478 | 91 | 11143 |
| Стан 10 | 215 | 2607 | 107 | 18593 |
| Стан 11 | 207 | 1939 | 103 | 16122 |
| Стан 12 | 247 | 2351 | 123 | 16275 |
| Стан 13 | 271 | 3073 | 135 | 22821 |
| Стан 14 | 215 | 1599 | 107 | 10729 |
| Стан 15 | 279 | 3570 | 139 | 24456 |
| Стан 16 | 231 | 2048 | 115 | 15108 |
| Стан 17 | 167 | 1130 | 83 | 8458 |
| Стан 18 | 175 | 1191 | 87 | 8551 |
| Стан 19 | 199 | 1951 | 99 | 14345 |
| Стан 20 | 151 | 1164 | 75 | 595 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму A\*, задачі Лабіринт для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання алгоритму A\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | 702 | 106 | 595 | 702 |
| Стан 2 | 635 | 93 | 541 | 636 |
| Стан 3 | 751 | 97 | 653 | 751 |
| Стан 4 | 652 | 97 | 554 | 652 |
| Стан 5 | 749 | 107 | 641 | 749 |
| Стан 6 | 666 | 98 | 567 | 666 |
| Стан 7 | 708 | 99 | 608 | 708 |
| Стан 8 | 402 | 48 | 353 | 402 |
| Стан 9 | 630 | 81 | 548 | 630 |
| Стан 10 | 777 | 111 | 665 | 777 |
| Стан 11 | 732 | 100 | 631 | 732 |
| Стан 12 | 592 | 94 | 397 | 592 |
| Стан 13 | 764 | 108 | 655 | 764 |
| Стан 14 | 467 | 70 | 396 | 467 |
| Стан 15 | 684 | 105 | 578 | 684 |
| Стан 16 | 593 | 82 | 510 | 593 |
| Стан 17 | 561 | 75 | 485 | 561 |
| Стан 18 | 478 | 64 | 413 | 478 |
| Стан 19 | 637 | 92 | 533 | 637 |
| Стан 20 | 595 | 81 | 513 | 595 |

Висновок

При виконанні цієї лабораторної роботи було розглянуто два різні алгоритми знаходження найкоротшого шляху в лабіринті: A\* та IDS. Я навчився краще працювати з алгоритмами та розуміти їх. Також дуже добре розвинув логічне і просторове мислення з допомогою роботи з двовимірним простором.

Критерії оцінювання

За умови здачі лабораторної роботи до 23.10.2022 включно максимальний бал дорівнює – 5. Після 23.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* програмна реалізація алгоритму – 60%;
* дослідження алгоритмів – 25%;
* висновок – 5%.