**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-12 Авчаров Григорій*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc23720)

[2 ЗаВдання 4](#_Toc23276)

[3 Виконання 7](#_Toc29273)

[3.1 Псевдокод алгоритмів 7](#_Toc1172)

[3.1.1 Пошук з обмеженням глибини 7](#_Toc22223)

[3.1.2 A\* 8](#_Toc19983)

[3.2 Програмна реалізація 9](#_Toc17516)

[3.2.1 Вихідний код 9](#_Toc6953)

[3.2.2 Приклади роботи 11](#_Toc16095)

[3.3 Дослідження алгоритмів 12](#_Toc7081)

[Висновок 15](#_Toc11896)

[Критерії оцінювання 16](#_Toc14260)

# 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).

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

Варіант 1

Лабірин, пошук з обмеженням глибини, пошук А\* з евристикою Манхетенська відстань

## Псевдокод алгоритмів

### Пошук з обмеженням глибини

LDFSUtil(int v, bool[] visited, int limit)

{

if (limit >= 0)

if (!flag)

{

visited[v] = true;

Console.Write(v + " ");

List<int> vList = adj[v];

foreach (var n in vList)

{

if (!visited[n])

if (n != 99)

{

DFSUtil(n, visited, limit - 1);

}

else

{

flag = true;

break;

}

}

}

}

LDFS(int v, int limit)

{

bool[] visited = new bool[V];

LDFSUtil(v, visited, limit);

}

3.1.2 A\*

let the openList equal empty list of nodes

let the closedList equal empty list of nodes// Add the start node

put the startNode on the openList (leave it's f at zero)// Loop until you find the end

while the openList is not empty // Get the current node

let the currentNode equal the node with the least f value

remove the currentNode from the openList

add the currentNode to the closedList // Found the goal

if currentNode is the goal

Congratz! You've found the end! Backtrack to get path // Generate children

let the children of the currentNode equal the adjacent nodes

for each child in the children // Child is on the closedList

if child is in the closedList

continue to beginning of for loop // Create the f, g, and h values

child.g = currentNode.g + distance between child and current

child.h = distance from child to end

child.f = child.g + child.h // Child is already in openList

if child.position is in the openList's nodes positions

if the child.g is higher than the openList node's g

continue to beginning of for loop // Add the child to the openList

add the child to the openList

## Програмна реалізація

### Вихідний код

using System.Diagnostics;

using System.Drawing;

using System.Collections.Generic;

namespace System.Text.Encoding.Unicode;

enum CellState { Close, Open };

internal class Program

{

public static void Main(string[] args)

{

int \_Width, \_Height;

Cell[,] Cells;

\_Width = 10;

\_Height = 10;

Cells = new Cell[\_Width, \_Height];

#region Generation

for (int y = 0; y < \_Height; y++)

for (int x = 0; x < \_Width; x++)

Cells[x, y] = new Cell(new Point(x, y));

Random rand = new Random();

Int32 startX = rand.Next(0);

Int32 startY = rand.Next(0);

Stack<Cell> path = new Stack<Cell>();

Cells[startX, startY].Visited = true;

path.Push(Cells[startX, startY]);

#region Creation

while (path.Count > 0)

{

Cell \_cell = path.Peek();

List<Cell> nextStep = new List<Cell>();

if (\_cell.Position.X > 0 && !Cells[Convert.ToInt32(\_cell.Position.X - 1), Convert.ToInt32(\_cell.Position.Y)].Visited)

nextStep.Add(Cells[Convert.ToInt32(\_cell.Position.X) - 1, Convert.ToInt32(\_cell.Position.Y)]);

if (\_cell.Position.X < \_Width - 1 && !Cells[Convert.ToInt32(\_cell.Position.X) + 1, Convert.ToInt32(\_cell.Position.Y)].Visited)

nextStep.Add(Cells[Convert.ToInt32(\_cell.Position.X) + 1, Convert.ToInt32(\_cell.Position.Y)]);

if (\_cell.Position.Y > 0 && !Cells[Convert.ToInt32(\_cell.Position.X), Convert.ToInt32(\_cell.Position.Y) - 1].Visited)

nextStep.Add(Cells[Convert.ToInt32(\_cell.Position.X), Convert.ToInt32(\_cell.Position.Y) - 1]);

if (\_cell.Position.Y < \_Height - 1 && !Cells[Convert.ToInt32(\_cell.Position.X), Convert.ToInt32(\_cell.Position.Y) + 1].Visited)

nextStep.Add(Cells[Convert.ToInt32(\_cell.Position.X), Convert.ToInt32(\_cell.Position.Y) + 1]);

if (nextStep.Count() > 0)

{

Cell next = nextStep[rand.Next(nextStep.Count())];

if (next.Position.X != \_cell.Position.X)

{

if (\_cell.Position.X - next.Position.X > 0)

{

\_cell.Left = CellState.Open;

next.Right = CellState.Open;

}

else

{

\_cell.Right = CellState.Open;

next.Left = CellState.Open;

}

}

if (next.Position.Y != \_cell.Position.Y)

{

if (\_cell.Position.Y - next.Position.Y > 0)

{

\_cell.Top = CellState.Open;

next.Bottom = CellState.Open;

}

else

{

\_cell.Bottom = CellState.Open;

next.Top = CellState.Open;

}

}

next.Visited = true;

path.Push(next);

}

else

{

path.Pop();

}

}

#endregion

for (int i = 0; i < \_Height; i++)

for (int j = 0; j < \_Width; j++)

{

if ((Cells[i, j].Left == CellState.Open &&

Cells[i, j].Right == CellState.Open &&

Cells[i, j].Top == CellState.Open

) ||

(Cells[i, j].Bottom == CellState.Open &&

Cells[i, j].Right == CellState.Open &&

Cells[i, j].Top == CellState.Open

) ||

(Cells[i, j].Left == CellState.Open &&

Cells[i, j].Bottom == CellState.Open &&

Cells[i, j].Top == CellState.Open

) ||

(Cells[i, j].Left == CellState.Open &&

Cells[i, j].Right == CellState.Open &&

Cells[i, j].Bottom == CellState.Open

)

)

{

int x = rand.Next(1, 4);

switch (x)

{

case 1:

Cells[i, j].Left = CellState.Close;

break;

case 2:

Cells[i, j].Right = CellState.Close;

break;

case 3:

Cells[i, j].Bottom = CellState.Close;

break;

case 4:

Cells[i, j].Top = CellState.Close;

break;

}

}

}

string[,] matr = new string[\_Height \* 2 + 1, \_Width \* 2 + 1];

for (int i = 0; i < \_Height \* 2 + 1; i++)

for (int j = 0; j < \_Width \* 2 + 1; j++)

{

if ((i + j) % 2 != 0)

{

matr[i, j] = " ";

}

else

{

matr[i, j] = "1";

}

if (i % 2 == 0)

matr[i, j] = " ";

}

for (int i = 0; i < \_Height; i++)

for (int j = 0; j < \_Width; j++)

{

if (Cells[i, j].Left == CellState.Open)

{

matr[i \* 2 + 1, j \* 2 - 1 + 1] = "-";

}

if (Cells[i, j].Top == CellState.Open)

{

matr[i \* 2 - 1 + 1, j \* 2 + 1] = "|";

}

if (Cells[i, j].Right == CellState.Open)

{

matr[i \* 2 + 1, j \* 2 + 1 + 1] = "-";

}

if (Cells[i, j].Bottom == CellState.Open)

{

matr[i \* 2 + 1 + 1, j \* 2 + 1] = "|";

}

}

matr[0, 0] = @"\";

matr[\_Width \* 2, \_Height \* 2] = @"\";

for (int i = -1; i < \_Height \* 2 + 1 + 1; i++)

{

for (int j = -1; j < \_Width \* 2 + 1 + 1; j++)

{

if ((i == -1 || i == \_Width \* 2 + 1) && j < \_Width \* 2 + 1)

{

Console.Write(" \_");

}

else if ((j == -1 || j == \_Height \* 2 + 1))

{

Console.Write("|");

}

else

{

Console.Write(matr[i, j] + " ");

}

}

Console.WriteLine();

}

#endregion

Cell start = Cells[\_Width - 1, \_Height - 1];

Cell[,] aPath = new Cell[\_Width, \_Height];

Cell[,] fwdPath = new Cell[\_Width, \_Height];

float[,] g\_score = new float[\_Width, \_Height];

for (int p = 0; p < \_Height; p++)

for (int j = 0; j < \_Width; j++)

{

g\_score[p, j] = float.PositiveInfinity;

}

g\_score[start.Position.X, start.Position.Y] = 0;

float[,] f\_score = new float[\_Width, \_Height];

for (int k = 0; k < \_Height; k++)

for (int j = 0; j < \_Width; j++)

{

f\_score[k, j] = float.PositiveInfinity;

}

f\_score[start.Position.X, start.Position.Y] = h(start, Cells[0, 0]);

PriorityQueue<Cell, float> open = new PriorityQueue<Cell, float>();

open.Enqueue(start, h(start, Cells[0, 0]));

while (open.Count > 0)

{

Cell currCell = open.Dequeue();

string[] directions = new string[] { "Left", "Right", "Top", "Bottom" };

if (currCell.Position.X == 1 && currCell.Position.Y == 1)

{

Console.Write("(" + 1 + "," + 1 + ") ");

Console.Write("(" + 1 + "," + 1 + ") ");

Console.Write("(" + 0 + "," + 0 + ") ");

//Console.Write(String.Format("{0,10:D}", "(" + 1 + "," + 1 + ") "));

break;

}

foreach (string direction in directions)

{

Cell childCell = Cells[0, 0];

bool visited = false;

if (direction == "Left" && currCell.Left == CellState.Open && currCell.Position.Y != 0)

{

childCell = Cells[currCell.Position.X, currCell.Position.Y - 1];

visited = true;

}

if (direction == "Top" && currCell.Top == CellState.Open && currCell.Position.X != 0)

{

childCell = Cells[currCell.Position.X - 1, currCell.Position.Y];

visited = true;

}

if (direction == "Right" && currCell.Right == CellState.Open && currCell.Position.Y != \_Width - 1)

{

childCell = Cells[currCell.Position.X, currCell.Position.Y + 1];

visited = true;

}

if (direction == "Bottom" && currCell.Bottom == CellState.Open && currCell.Position.X != \_Height - 1)

{

childCell = Cells[currCell.Position.X + 1, currCell.Position.Y];

visited = true;

}

if (visited)

{

float temp\_g\_score = g\_score[currCell.Position.X, currCell.Position.Y] + 1;

float temp\_f\_score = temp\_g\_score + h(childCell, Cells[0, 0]);

if (temp\_f\_score < f\_score[childCell.Position.X, childCell.Position.Y])

{

g\_score[childCell.Position.X, childCell.Position.Y] = temp\_g\_score;

f\_score[childCell.Position.X, childCell.Position.Y] = temp\_f\_score;

open.Enqueue(childCell, temp\_f\_score);

aPath[childCell.Position.X, childCell.Position.Y] = currCell;

Console.Write("(" + currCell.Position.X + "," + currCell.Position.Y + ") ");

}

}

}

}

Console.WriteLine(g\_score);

Console.WriteLine(f\_score);

int q = 0;

Console.WriteLine();

Console.WriteLine();

Cell cell = Cells[1, 1];

while (cell != start)

{

if (cell != null && aPath[cell.Position.X, cell.Position.Y] != null)

{

fwdPath[aPath[cell.Position.X, cell.Position.Y].Position.X, aPath[cell.Position.X, cell.Position.Y].Position.Y] = cell;

cell = aPath[cell.Position.X, cell.Position.Y];

}

else

{

Console.WriteLine("Labyrith doesnt have solutions");

}

}

for (int z = 1; z < \_Height; z++)

{

for (int j = 0; j < \_Width; j++)

{

if (fwdPath[z, j] != null)

Console.Write(String.Format("{0,10:D}", "(" + fwdPath[z, j].Position.X + "," + fwdPath[z, j].Position.Y + ") "));

else Console.Write(String.Format("{0,10:D}", "(" + "/" + "," + "/" + ") "));

}

Console.WriteLine();

}

Console.WriteLine();

/\*

for (int a = 0; a < \_Height; a++)

{

for (int j = 0; j < \_Width; j++)

{

if (aPath[a, j] != null)

Console.Write(String.Format("{0,10:D}", "(" + f\_score[a, j] + "," + g\_score[a, j] + ") "));

else Console.Write(String.Format("{0,10:D}", "(" + "/" + "," + "/" + ") "));

}

Console.WriteLine();

}\*/

float h(Cell cell1, Cell cell2)

{

int x1 = cell1.Position.X;

int y1 = cell1.Position.Y;

int x2 = cell2.Position.X;

int y2 = cell2.Position.Y;

return Math.Abs(x1 - x2) + Math.Abs(y1 - y2);

}

}

}

class Cell

{

public Cell(Point currentPosition)

{

Visited = false;

Position = currentPosition;

}

public CellState Left { get; set; }

public CellState Right { get; set; }

public CellState Bottom { get; set; }

public CellState Top { get; set; }

public Boolean Visited { get; set; }

public Point Position { get; set; }

}

class Graph

{

private int V;

private List<int>[] adj;

bool flag = false;

public Graph(int v)

{

V = v;

adj = new List<int>[v];

for (int i = 0; i < v; ++i)

adj[i] = new List<int>();

}

public void AddEdge(int v, int w)

{

adj[v].Add(w);

}

public void DFSUtil(int v, bool[] visited, int limit)

{

if (limit >= 0)

if (!flag)

{

visited[v] = true;

Console.Write(v + " ");

List<int> vList = adj[v];

foreach (var n in vList)

{

if (!visited[n])

if (n != 99)

{

DFSUtil(n, visited, limit - 1);

}

else

{

flag = true;

Console.Write(n + " ");

break;

}

}

}

}

public void DFS(int v, int limit)

{

bool[] visited = new bool[V];

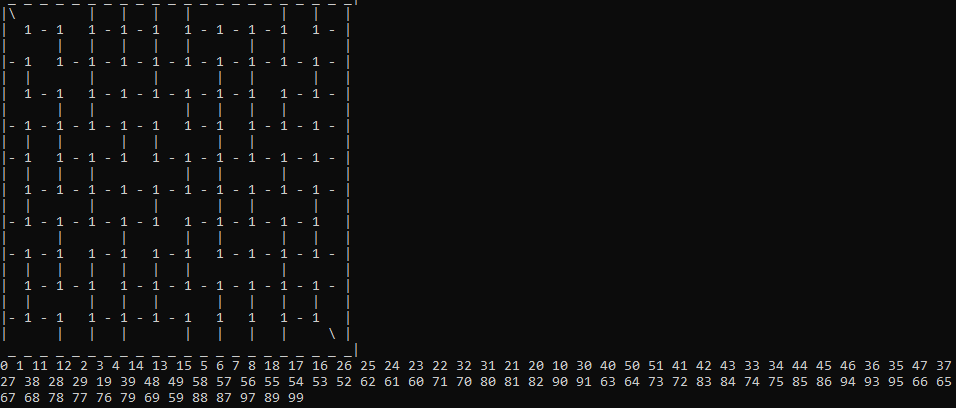
DFSUtil(v, visited, limit);

}

}

### Приклади роботи

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



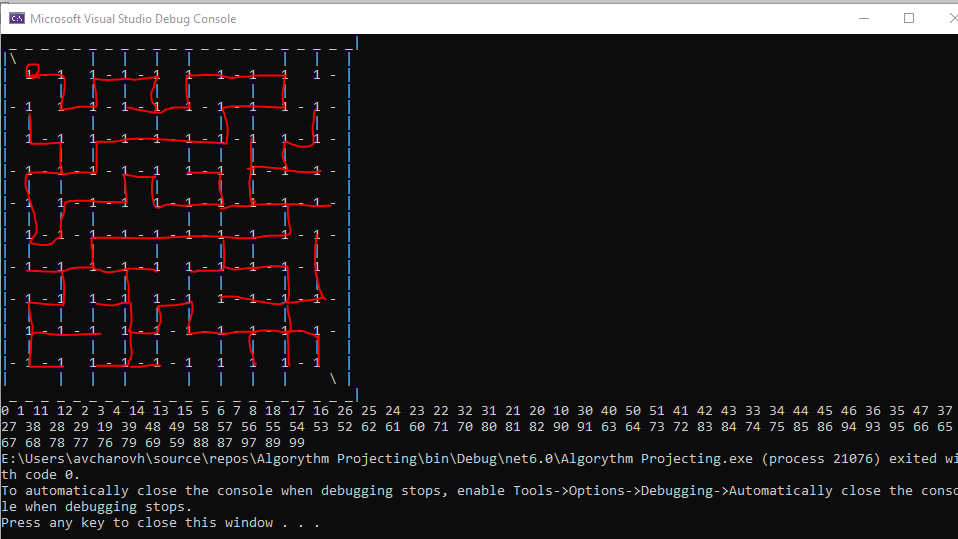


Рисунок 3.1 – Алгоритм пошуку з обмеженням глибини

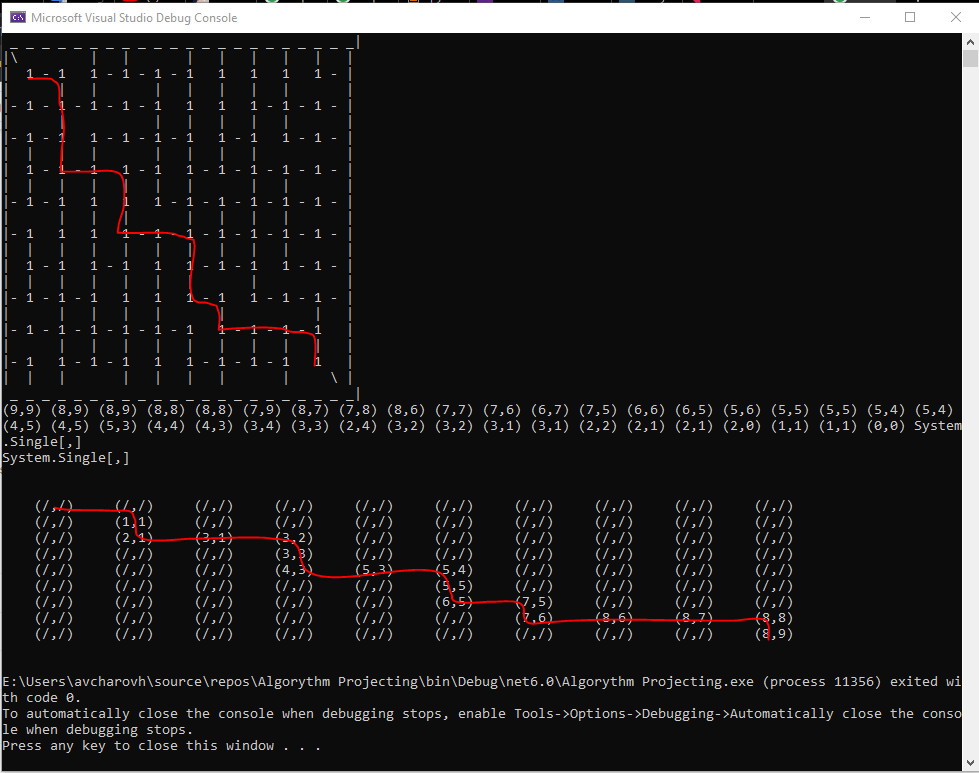


Рисунок 3.2 – Алгоритм пошуку А\*

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму пошуку з обмеженням глибини задачі лабіринту для 10 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму пошуку з обмеженням глибини

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пом’яті |
| Стан 1 | 856 | 583 | 272 |
| Стан 2 | 8691 | 4822 | 1218 |
| Стан 3 | 2588 | 1693 | 666 |
| Стан 4 | 521 | 450 | 234 |
| Стан 5 | 4115 | 2596 | 893 |
| Стан 6 | 1363 | 883 | 341 |
| Стан 7 | 679 | 483 | 267 |
| Стан 8 | 3032 | 1920 | 674 |
| Стан 9 | 897 | 611 | 280 |
| Стан 10 | 1047 | 691 | 288,5 |
| Average | 3239 | 1924 | 578 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму пошуку А\*, задачі Лабіринту для 10 початкових станів.

Таблиця 3.3 – Характеристики оцінювання пошуку А\*

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пом’яті |
| Стан 1 | 4422 | 4450 | 51 |
| Стан 2 | 4185 | 4221 | 51 |
| Стан 3 | 4079 | 4120 | 51 |
| Стан 4 | 4082 | 4124 | 59 |
| Стан 5 | 4539 | 4556 | 56 |
| Стан 6 | 4355 | 4382 | 64 |
| Стан 7 | 4298 | 4308 | 56 |
| Стан 8 | 4651 | 4677 | 59 |
| Стан 9 | 4320 | 4350 | 56 |
| Стан 10 | 4191 | 4218 | 59 |
| Average | 4312 | 4340 | 56 |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми пошуку в глибину з обмеженням глибини та А\* для задачі лабіринту, було здійснено програмну реалізацію цих алгоритмів. Було здійснено 10 експерементів для кожного із алгоритмів і зафіксовано кількість ітерацій, кількість пройдених станів та максимальну кількість станів у пам’яті.

Зроблено висновок, що пошук з обмеженням глибини є неповним і неоптимальним алгоритмом, коли пошук А\* є повним і оптимальним допоки еврестична функція є прийнятною. Було отримано, що пошук А\* має меншу кількість станів у пам’яті, але в середньому розгортає більше станів та здійснює більше ітерацій.

Критерії оцінювання

За умови здачі лабораторної роботи до 30.10.2022 включно максимальний бал дорівнює – 5. Після 30.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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