Міністерство освіти і науки України

Національний лісотехнічний університет України

Кафедра інформаційних технологій

**ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №6**

з навчальної дисципліни

**«Системи штучного інтелекту»**

на тему:

«**Алгоритм А\***»

**Виконав:**

студент групи КНС-11/2

Тимчак Д. О.

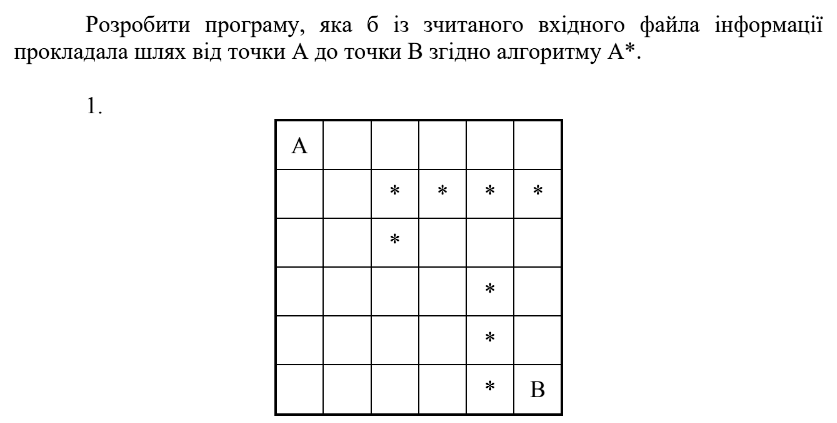
**Перевірив:**

Сторожук О.Л.

Львів - 2023

**МЕТА РОБОТИ**: ознайомитись із алгоритмом А\* та використати його для

пошуку оптимального шляху.

**Варіант 1** ****

**Хід виконання:**

Лістинг програми:  
  
#include <iostream>

#include <fstream>

#include <vector>

#include <queue>

#include <cmath>

#include <stack>

#include <Windows.h>

using namespace std;

const int ROWS = 6;

const int COLS = 6;

struct Node {

int row, col, g, h, f;

Node\* parent;

Node(int r, int c, int g\_val, int h\_val, Node\* p) : row(r), col(c), g(g\_val), h(h\_val), parent(p) {

f = g + h;

}

};

bool isValid(int row, int col) {

return (row >= 0) && (row < ROWS) && (col >= 0) && (col < COLS);

}

bool isObstacle(const vector<vector<int>>& matrix, int row, int col) {

return matrix[row][col] == 1;

}

bool isDestination(int row, int col, const pair<int, int>& dest) {

return row == dest.first && col == dest.second;

}

int calculateHValue(int row, int col, const pair<int, int>& dest) {

return abs(row - dest.first) + abs(col - dest.second);

}

void tracePath(Node\* endNode) {

stack<pair<int, int>> path;

while (endNode != nullptr) {

path.push({ endNode->row, endNode->col });

endNode = endNode->parent;

}

cout << "Шлях: ";

while (!path.empty()) {

auto point = path.top();

path.pop();

cout << "[" << point.first << ", " << point.second << "] ";

}

cout << endl;

}

bool isPartOfPath(Node\* currentNode, int row, int col);

void printMatrixWithHighlightedPath(const vector<vector<int>>& matrix, const pair<int, int>& start,

const pair<int, int>& dest, Node\* currentNode) {

system("cls"); // Очистка консолі (Windows)

cout << "Матриця:" << endl;

for (int i = 0; i < ROWS; ++i) {

for (int j = 0; j < COLS; ++j) {

if (make\_pair(i, j) == start) {

cout << "A ";

}

else if (make\_pair(i, j) == dest) {

cout << "B ";

}

else {

if (matrix[i][j] == 1) {

cout << "\* ";

}

else if (currentNode != nullptr && isPartOfPath(currentNode, i, j)) {

cout << ". "; // Виділяємо шлях символом "."

}

else {

cout << "0 ";

}

}

}

cout << endl;

}

Sleep(500); // Пауза для візуалізації кроку алгоритму (500 мілісекунд)

}

bool isPartOfPath(Node\* currentNode, int row, int col) {

while (currentNode != nullptr) {

if (currentNode->row == row && currentNode->col == col) {

return true;

}

currentNode = currentNode->parent;

}

return false;

}

void aStar(const vector<vector<int>>& matrix, const pair<int, int>& start, const pair<int, int>& dest) {

auto cmp = [](Node\* a, Node\* b) {

return (a->f) > (b->f);

};

priority\_queue<Node\*, vector<Node\*>, decltype(cmp)> openSet(cmp);

vector<vector<bool>> closedSet(ROWS, vector<bool>(COLS, false));

Node\* startNode = new Node(start.first, start.second, 0, calculateHValue(start.first, start.second, dest), nullptr);

openSet.push(startNode);

while (!openSet.empty()) {

Node\* currentNode = openSet.top();

openSet.pop();

int row = currentNode->row;

int col = currentNode->col;

closedSet[row][col] = true;

if (isDestination(row, col, dest)) {

printMatrixWithHighlightedPath(matrix, start, dest, currentNode);

tracePath(currentNode);

return;

}

vector<pair<int, int>> successors = {

{row - 1, col}, {row + 1, col}, {row, col - 1}, {row, col + 1}

};

for (const auto& successor : successors) {

int nextRow = successor.first;

int nextCol = successor.second;

if (isValid(nextRow, nextCol) && !closedSet[nextRow][nextCol] && !isObstacle(matrix, nextRow, nextCol)) {

int gValue = currentNode->g + 1;

int hValue = calculateHValue(nextRow, nextCol, dest);

Node\* successorNode = new Node(nextRow, nextCol, gValue, hValue, currentNode);

openSet.push(successorNode);

}

}

// Виводимо матрицю з виділеним шляхом

printMatrixWithHighlightedPath(matrix, start, dest, currentNode);

}

cout << "Шлях не знайдено." << endl;

}

int main() {

SetConsoleOutputCP(1251);

ifstream inputFile("Area.txt");

if (!inputFile.is\_open()) {

cout << "Помилка відкриття файлу." << endl;

return 1;

}

vector<vector<int>> matrix(ROWS, vector<int>(COLS));

for (int i = 0; i < ROWS; ++i) {

for (int j = 0; j < COLS; ++j) {

inputFile >> matrix[i][j];

}

}

inputFile.close();

pair<int, int> start, dest;

cout << "Введіть початкову точку (рядок колонка): ";

cin >> start.first >> start.second;

cout << "Введіть кінцеву точку (рядок колонка): ";

cin >> dest.first >> dest.second;

if (!isValid(start.first, start.second) || !isValid(dest.first, dest.second) ||

isObstacle(matrix, start.first, start.second) || isObstacle(matrix, dest.first, dest.second)) {

cout << "Неправильна початкова або кінцева точка." << endl;

system("pause");

return 1;

}

if (matrix[dest.first][dest.second] == 1) {

cout << "Кінцева точка знаходиться на перешкоді. До неї не можна дістатися." << endl;

system("pause");

return 1;

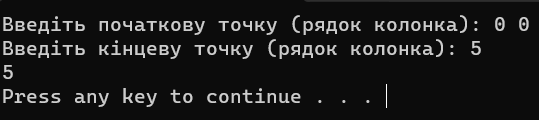
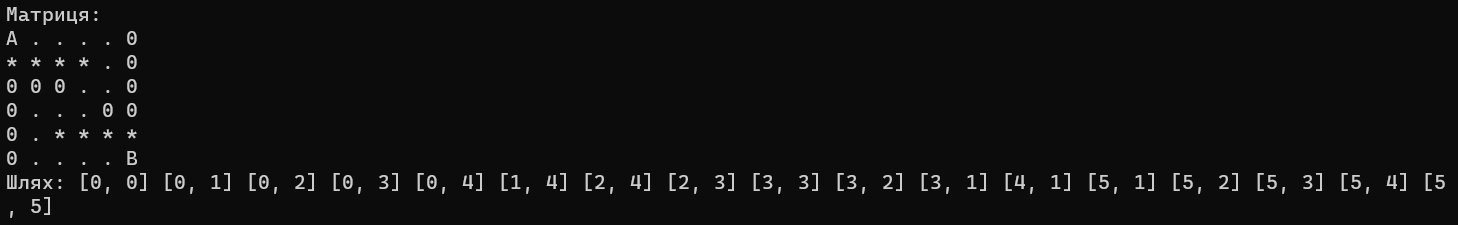
}

system("pause");

aStar(matrix, start, dest);

return 0;

}

Результат виконання програми:  
  
 

Пояснення коду:

Цей код реалізує алгоритм A\* для пошуку найкоротшого шляху в лабіринті. Давайте детально розглянемо кожен етап роботи програми:

1. Оголошення структури Node

struct Node {

int row, col, g, h, f;

Node\* parent;

Node(int r, int c, int g\_val, int h\_val, Node\* p) : row(r), col(c), g(g\_val), h(h\_val), parent(p) {

f = g + h;

}

};

Структура Node представляє вузол у графі. Кожен вузол має координати (row, col), вартість шляху від початкового вузла до поточного (g), евристичну вартість від поточного вузла до кінцевого (h), загальну вартість (f), та посилання на батьківський вузол (parent).

2. Оголошення допоміжних функцій

bool isValid(int row, int col);

bool isObstacle(const vector<vector<int>>& matrix, int row, int col);

bool isDestination(int row, int col, const pair<int, int>& dest);

int calculateHValue(int row, int col, const pair<int, int>& dest);

void tracePath(Node\* endNode);

bool isPartOfPath(Node\* currentNode, int row, int col);

Ці функції виконують наступні завдання:

isValid: Перевіряє, чи координати (row, col) знаходяться в межах лабіринту.

isObstacle: Перевіряє, чи вказана клітина лабіринту є перешкодою.

isDestination: Перевіряє, чи поточні координати є кінцевою точкою.

calculateHValue: Обчислює евристичну вартість (відстань Мангеттена) від поточної точки до кінцевої.

tracePath: Виводить знайдений шлях, переходячи від кінцевого вузла до початкового.

isPartOfPath: Перевіряє, чи вказана клітина є частиною шляху, побудованого від початкового до поточного вузла.

3. Оголошення функції printMatrixWithHighlightedPath

void printMatrixWithHighlightedPath(const vector<vector<int>>& matrix, const pair<int, int>& start,

const pair<int, int>& dest, Node\* currentNode);

Ця функція виводить матрицю лабіринту, при цьому виділяючи шлях, побудований від початкового вузла до поточного.

4. Оголошення основної функції aStar

void aStar(const vector<vector<int>>& matrix, const pair<int, int>& start, const pair<int, int>& dest);

Основна функція алгоритму A\* має такий хід роботи:

4.1. Ініціалізація

auto cmp = [](Node\* a, Node\* b) {

return (a->f) > (b->f);

};

priority\_queue<Node\*, vector<Node\*>, decltype(cmp)> openSet(cmp);

vector<vector<bool>> closedSet(ROWS, vector<bool>(COLS, false));

Node\* startNode = new Node(start.first, start.second, 0, calculateHValue(start.first, start.second, dest), nullptr);

openSet.push(startNode);

Створюється пріоритетна черга openSet для вибору вузла з мінімальною вартістю f. Також ініціалізуються матриця closedSet для відстеження вже оброблених вузлів та початковий вузол.

4.2. Цикл пошуку

while (!openSet.empty()) {

Node\* currentNode = openSet.top();

openSet.pop();

int row = currentNode->row;

int col = currentNode->col;

closedSet[row][col] = true;

if (isDestination(row, col, dest)) {

printMatrixWithHighlightedPath(matrix, start, dest, currentNode);

tracePath(currentNode);

return;

}

vector<pair<int, int>> successors = {

{row - 1, col}, {row + 1, col}, {row, col - 1}, {row, col + 1}

};

Виконується цикл пошуку, доки черга openSet не стає порожньою. Обирається вузол з мінімальною вартістю f. Якщо це кінцевий вузол, алгоритм завершується.

4.3. Генерація наступників

for (const auto& successor : successors) {

int nextRow = successor.first;

int nextCol = successor.second;

if (isValid(nextRow, nextCol) && !closedSet[nextRow][nextCol] && !isObstacle(matrix, nextRow, nextCol)) {

int gValue = currentNode->g + 1;

int hValue = calculateHValue(nextRow, nextCol, dest);

Node\* successorNode = new Node(nextRow, nextCol, gValue, hValue, currentNode);

openSet.push(successorNode);

}

}

// Виводимо матрицю з виділеним шляхом

printMatrixWithHighlightedPath(matrix, start, dest, currentNode);

}

Для кожного можливого наступника поточного вузла обчислюється його вартість g та h. Якщо вузол вже закритий, він проігноровується. Якщо вартість g для наступника менше, ніж раніше збережена в пріоритетній черзі, вартість оновлюється, або додається новий вузол в чергу.

4.4. Виведення результату

cout << "Шлях не знайдено." << endl;

Якщо алгоритм завершено (досягнута ціль), виводиться шлях від початкового до кінцевого вузла. Це відбувається у функціях printMatrixWithHighlightedPath та tracePath.

5. Оголошення функції main

int main() {

// ... (зчитування матриці та координат початкової та кінцевої точок)

aStar(matrix, start, dest);

return 0;

}

У функції main зчитується матриця лабіринту з файлу, вводяться координати початкової та кінцевої точок, і запускається алгоритм A\* для пошуку шляху.

**Висновок:** ознайомлено із алгоритмом А\* та використано його для

пошуку оптимального шляху. Створено програму на мові програмування С++

Яка зчитує поле з перешкодами та за допомогою даного алгоритму шукає найкоротший шлях до кінцевої точки