**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-13 Крупосій Вадим Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070688)

[2 ЗаВдання 4](#_Toc81070689)

[3 Виконання 8](#_Toc81070690)

[3.1 Псевдокод алгоритмів 8](#_Toc81070691)

[3.2 Програмна реалізація 8](#_Toc81070692)

[3.2.1 Вихідний код 8](#_Toc81070693)

[3.2.2 Приклади роботи 8](#_Toc81070694)

[3.3 Дослідження алгоритмів 8](#_Toc81070695)

[Висновок 11](#_Toc81070696)

[Критерії оцінювання 12](#_Toc81070697)

# 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

BFS

ФУНКЦІЯ BFS

Поточний стан = дошка()

Черга = нова.черга()

ПОВТОРИТИ ПОКИ не черга.порожня()

стан = черга.вилучити()

для нащадок в поточний\_стан.розкрити\_стан

ЯКЩО не конфлікт(стан)

ПОВЕРНУТИ стан

ВСЕ ЯКЩО

ДЛЯ стан В отримати\_нащадків(стан)

черга.додати(стан)

ВСЕ ПОВТОРИТИ

RBFS

**function** Recursive-Best-First-Search(problem) **returns** рішення result

або індикатор невдачі *failure*

RBFS(*problem*, Make-Node(Initial-State[*problem*] ) , ∞)

**function** RBFS(problem, node, f\_limit) **returns** рішення result

або індикатор невдачі failure і новий межа f-вартості f\_limit

**if** Goal-Test[*problem*](State[*node*]) **then return** вузел *node*

*successors* ← Expand(*node*, *problem*)

**if** множина вузлів-наступників successors пуста

**then return** failure, ∞

**for each** s **in** *successors* **do**

f[s] ← *max*(g(s)+h(s) , f[*node*] )

**repeat**

best ← вузол з найменшим f-значенням в множині *successors*

**if** f[best] > *f\_limit* **then return** *failure*, f[*best*]

alternative ← друге після найменшого f-значення в множині successors

result, f[best] ← RBFS(problem, best, min{f\_limit, alternative))

**if** result ≠ failure **then return** result

## Програмна реалізація

### Вихідний код

**Board.cs:**

using System;

using System.Collections.Generic;

using System.Dynamic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

public class Board

{

public int[] State { get; private set; }

public int f\_coef = 0;

public Board()

{

State = new int[8];

}

public Board(int[] board)

{

State = new int[8];

for (int i = 0; i < 8; i++)

{

State[i] = board[i];

}

}

public bool QueensHits()

{

for (int i = 0; i < 8; i++)

{

for (int j = i; j < 8; j++)

{

if (i != j)

{

if (State[i] == State[j] || j - i == State[j] - State[i] || j - i == -(State[j] - State[i]))

{

return true;

}

}

}

}

return false;

}

public Board[] ExpandState()

{

List<Board> expandedStates = new List<Board>();

for (int i = 0; i < 8; i++)

{

int y = State[i];

for (int j = 0; j < 7; j++)

{

Board tempState = new Board(State);

if (y <= j)

{

tempState.State[i] = j + 1;

}

else

{

tempState.State[i] = j;

}

expandedStates.Add(tempState);

}

}

return expandedStates.ToArray();

}

public int f2()

{

int result = 0;

for (int i = 1; i < 8; ++i)

{

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

{

if (State[i] + i == State[j] + j)

{

result++;

}

if (State[i] - i == State[j] - j)

{

result++;

}

if (State[i] == State[j])

{

result++;

}

}

}

return result;

}

public void DisplayState()

{

Console.Write(" ");

for (int c = 0; c < 8; c++)

{

Console.Write($" {c + 1} ");

}

Console.WriteLine();

for (int i = 0; i < 8; i++)

{

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

for (int j = 0; j < 8; j++)

{

if (State[i] == j)

{

Console.Write("[Q]");

}

else

{

Console.Write("[\_]");

}

}

Console.WriteLine();

}

}

}

**BFS.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace lab2

{

public static class BFS

{

public static Board Execute(Board root)

{

DateTime dt = DateTime.Now;

Board currentState = new Board();

Queue<Board> queue = new Queue<Board>();

queue.Enqueue(root);

while (queue.TryPeek(out \_))

{

currentState = queue.Dequeue();

foreach (Board child in currentState.ExpandState())

{

if (!child.QueensHits())

{

Console.WriteLine("Time: " + (DateTime.Now - dt).TotalSeconds + " s");

return child;

}

queue.Enqueue(child);

}

}

return null;

}

}

}

**RBFS.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab2

{

internal class RBFS

{

private static int limit = 10;

public static Board Execute(Board root)

{

DateTime dt = DateTime.Now;

Board result = RBFS\_algo(root, 30, 0);

Console.WriteLine("Time: " + (DateTime.Now - dt).TotalSeconds + " s");

return result;

}

private static Board RBFS\_algo(Board node, int f\_limit, int depth)

{

if (isSolved(node))

{

return node;

}

if (depth >= limit)

{

return null;

}

List<Board> childStates = node.ExpandState().ToList();

List<int> f = new List<int>();

foreach (var child in childStates)

{

f.Add(child.f2());

}

while (true)

{

int bestValue = f.Min();

int bestIndex = f.IndexOf(bestValue);

Board bestState = childStates[bestIndex];

if (bestValue > f\_limit)

{

return null;

}

childStates.Remove(bestState);

f.Remove(bestValue);

int altValue = f.Min();

Board result = RBFS\_algo(bestState, Math.Min(f\_limit, altValue), depth + 1);

if (result != null)

{

return result;

}

}

}

private static bool isSolved(Board state)

{

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

{

for (int j = 0; j < state.State.Length; ++j)

{

if (i == j) continue;

if (state.State[i] == state.State[j] || i + state.State[i] == j + state.State[j] || i - state.State[i] == j - state.State[j])

{

return false;

}

}

}

return true;

}

}

}

**Rrogram.cs:**

using Microsoft.CodeAnalysis;

using System;

using System.Collections.Generic;

using System.Linq;

namespace lab2

{

internal class Program

{

static void Main(string[] args)

{

int[] initialQueens = new int[8];

Console.WriteLine("Generate State:");

Random rnd = new Random();

for (int i = 0; i < 8; i++)

{

initialQueens[i] = rnd.Next(8);

}

Console.WriteLine();

Board root = new Board(initialQueens);

root.DisplayState();

Console.WriteLine();

Console.WriteLine("BFS algorithm find solution:");

Board result = BFS.Execute(root);

if (result == null)

{

Console.WriteLine("Can't Find Any Solution");

}

else

{

result.DisplayState();

}

Console.WriteLine();

Console.WriteLine("RBFS algorithm find solution:");

result = RBFS.Execute(root);

if (result == null)

{

Console.WriteLine("Can't Find Any Solution");

}

else

{

result.DisplayState();

}

Console.WriteLine();

}

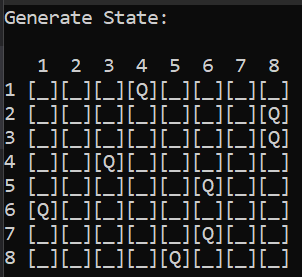
}

}

### Приклади роботи

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

Початковий стан:



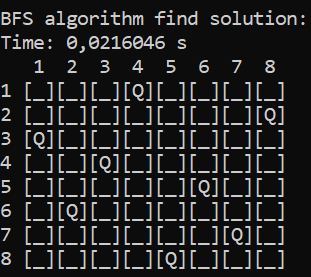


Рисунок 3.1 – Алгоритм BFS

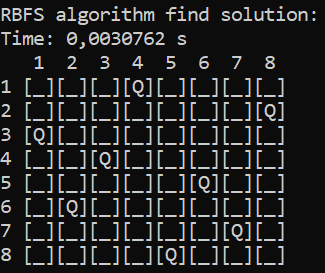


Рисунок 3.2 – Алгоритм RBFS

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму bfs, задачі 8-ферзів для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму bfs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
| Стан 1 | 5390 | 0 | 43125 | 40215 |
| Стан 2 | 123154 | 0 | 985209 | 984301 |
| Стан 3 | 1986 | 0 | 15881 | 14900 |
| Стан 4 | 58345 | 0 | 466750 | 465708 |
| Стан 5 | 142386 | 0 | 1139078 | 1129064 |
| Стан 6 | 306381 | 0 | 2451041 | 2444069 |
| Стан 7 | 160463 | 0 | 1283694 | 1282789 |
| Стан 8 | 700709 | 0 | 5605686 | 5595673 |
| Стан 9 | 91111 | 0 | 728894 | 718754 |
| Стан 10 | 162480 | 0 | 1299846 | 1289654 |
| Стан 11 | 17883 | 0 | 143082 | 135667 |
| Стан 12 | 174010 | 0 | 1392142 | 1287543 |
| Стан 13 | 511516 | 0 | 4092164 | 4084657 |
| Стан 14 | 162435 | 0 | 1298527 | 1283953 |
| Стан 15 | 77583 | 0 | 624082 | 619586 |
| Стан 16 | 198457 | 0 | 1590001 | 1573456 |
| Стан 17 | 635 | 0 | 5055 | 4927 |
| Стан 18 | 91123 | 0 | 728879 | 715647 |
| Стан 19 | 455456 | 0 | 3464314 | 33796852 |
| Стан 20 | 316947 | 0 | 2536573 | 2528641 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму rbfs, задачі 8-ферзів для 20 початкових станів.

Таблиця 3.2 – Характеристики оцінювання алгоритму RBFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | 3447 | 3040 | 22456 | 65 |
| Стан 2 | 6 | 0 | 333 | 61 |
| Стан 3 | 9 | 0 | 506 | 64 |
| Стан 4 | 155 | 113 | 2343 | 64 |
| Стан 5 | 34 | 17 | 841 | 65 |
| Стан 6 | 4824 | 4480 | 19264 | 65 |
| Стан 7 | 862 | 675 | 10472 | 65 |
| Стан 8 | 104 | 73 | 1730 | 65 |
| Стан 9 | 14 | 3 | 616 | 65 |
| Стан 10 | 90 | 72 | 1174 | 65 |
| Стан 11 | 6 | 0 | 336 | 61 |
| Стан 12 | 1514 | 1320 | 10868 | 65 |
| Стан 13 | 6378 | 5693 | 38360 | 65 |
| Стан 14 | 5 | 0 | 280 | 60 |
| Стан 15 | 60 | 40 | 1120 | 65 |
| Стан 16 | 4 | 0 | 224 | 59 |
| Стан 17 | 1214 | 1070 | 8125 | 65 |
| Стан 18 | 6 | 0 | 336 | 61 |
| Стан 19 | 310 | 267 | 2466 | 65 |
| Стан 20 | 53 | 34 | 1060 | 65 |

Висновок

При виконанні даної лабораторної роботи було розглянуто два алгоритми: алгоритм неінформативного пошуку **АНП - BFS**, алгоритм локального пошуку **АЛП та бектрекінгу - RBFS,** що використовує задану евристичну функцію Func – F2. Тестування проводились на задачі 8-ферзів. Було проведено серію експериментів для вивчення ефективності роботи алгоритмів. Кожний експеримент відрізняється початковим станом. Початковий стан фіксувався у таблиці експериментів. За проведеними серіями було визначено середню кількість етапів (кроків), які знадобились 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. Було передбачити можливість обмеження виконання програми за часом (30 хвилин) та використання пам’яті (1 Гб).

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%.