**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 № 1 з дисципліни

«Проектування алгоритмів»

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

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Спаських Микола ІТ-04 18-варіант*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070688)

[2 ЗаВдання 4](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070689)

[3 Виконання 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070690)

[3.1 Псевдокод алгоритмів 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070691)

[3.2 Програмна реалізація 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070692)

[3.2.1 Вихідний код 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070693)

[3.2.2 Приклади роботи 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070694)

[3.3 Дослідження алгоритмів 8](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070695)

[Висновок 11](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_Toc81070696)

[Критерії оцінювання 12](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr1.ukr.docx#_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. 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 (512 Мб)

**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.
* **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.
* **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 | 8-ферзів | LDFS | A\* |  | F1 |
| 2 | 8-ферзів | LDFS | A\* |  | F2 |
| 3 | 8-ферзів | LDFS | RBFS |  | F1 |
| 4 | 8-ферзів | LDFS | RBFS |  | F2 |
| 5 | 8-ферзів | BFS | A\* |  | F1 |
| 6 | 8-ферзів | BFS | A\* |  | F2 |
| 7 | 8-ферзів | BFS | RBFS |  | F1 |
| 8 | 8-ферзів | BFS | RBFS |  | F2 |
| 9 | 8-ферзів | IDS | A\* |  | F1 |
| 10 | 8-ферзів | IDS | A\* |  | F2 |
| 11 | 8-ферзів | IDS | RBFS |  | F1 |
| 12 | 8-ферзів | IDS | RBFS |  | F2 |
| 13 | 8-puzzle | LDFS | A\* |  | H1 |
| 14 | 8-puzzle | LDFS | A\* |  | H2 |
| 15 | 8-puzzle | LDFS | RBFS |  | H1 |
| 16 | 8-puzzle | LDFS | RBFS |  | H2 |
| 17 | 8-puzzle | BFS | A\* |  | H1 |
| 18 | 8-puzzle | BFS | A\* |  | H2 |
| 19 | 8-puzzle | BFS | RBFS |  | H1 |
| 20 | 8-puzzle | BFS | RBFS |  | H2 |
| 21 | 8-puzzle | IDS | A\* |  | H1 |
| 22 | 8-puzzle | IDS | A\* |  | H2 |
| 23 | 8-puzzle | IDS | RBFS |  | H1 |
| 24 | 8-puzzle | IDS | RBFS |  | H2 |
| 25 | COLOR |  |  | HILL | MRV |
| 26 | COLOR |  |  | ANNEAL | MRV |
| 27 | COLOR |  |  | BEAM | MRV |
| 28 | COLOR |  |  | HILL | DGR |
| 29 | COLOR |  |  | ANNEAL | DGR |
| 30 | COLOR |  |  | BEAM | DGR |
| 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

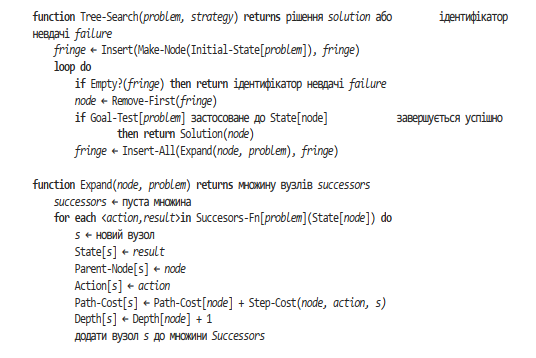
## Псевдокод алгоритмів

Структура вузла:

• State. Стан в просторі станів, якому відповідав би даний вузол.  
• Parent-Node. Вузол в дереві пошуку, що застосовувався для формування  
даного вузла (батьківський вузол).  
• Асtion. Дія, яка була застосована до батьківського вузла для формування  
даного вузла.  
• Path-Cost. Вартість шляху (від початкового стану до даного вузла),  
показаного за допомогою покажчиків батьківських вузлів, яку прийнято  
позначати як g(n).  
• Depth. Кількість етапів шляху від початкового cтану, що також  
називають глибиною.

• f=Path-Cost+h(Node, Goal); h – манхетенська відстань.

Для пошуку в ширину застосовував цей алгоритм:



Тільки замість солюшн він в мене повертав цільовий вузел в дереві, а потім по ссилкам на його предків я будував рішення.

Для А\*, я знайшов цей псевдокод в вікіпедії і реалізував його програмно:

## Реализация А\*

В приведённой реализации:

* Q — множество вершин, которые требуется рассмотреть,
* U — множество рассмотренных вершин,
* f[x] — значение эвристической функции "расстояние + стоимость" для вершины x,
* g[x] — стоимость пути от начальной вершины до xx,
* h(x) — эвристическая оценка расстояния от вершины xx до конечной вершины.

На каждом этапе работы алгоритма из множества Q выбирается вершина с наименьшим значением эвристической функции и просматриваются её соседи. Для каждого из соседей обновляется расстояние, значение эвристической функции и он добавляется в множество Q.  
Псевдокод:

**bool** A\*(start, goal)**:**

U = ∅

Q = ∅

Q.push(start)

g[start] = 0

f[start] = g[start] + h(start)

**while** Q.size() != 0

current = вершина из Q с минимальным значением f

**if** current == goal

**return** *true* // нашли путь до нужной вершины

Q.remove(current)

U.push(current)

**for** v : смежные с current вершины

tentativeScore = g[current] + d(current, v) // d(current, v) — стоимость пути между current и v

**if** v∈U **and** tentativeScore >= g[v]

**continue**

**if** v∉U **or** tentativeScore < g[v]

parent[v] = current

g[v] = tentativeScore

f[v] = g[v] + h(v)

**if** v∉Q

Q.push(v)

**return** *false*

тільки замість bool результату в мене він повертає Node, current коли рішення знайдено і null – коли не знайдено. Тобто в ньому я замінив bool на Node, true на current і false на null;

## Програмна реалізація

### Вихідний код

using System;

using System.Collections.Generic;

namespace lab1\_1

{

class Node

{

public int[,] State;

public Node Parent\_Node;

public string action;

public int Path\_Cost;

public int Depth;

public int f;

public Node(int[,] State, Node Parent)

{

this.State = new int[3, 3];

this.State = State;

Parent\_Node = Parent;

if (Parent == null)

{

this.Depth = 0;

this.Path\_Cost = 0;

}

else

{

this.Depth = Parent.Depth + 1;

this.Path\_Cost = Path\_Cost + 1;

}

}

}

class Program

{

public static int[,] matr = new int[3, 3]{

{7,3,6},

{1,4,8},

{5,2,0}

};

public static int kilk\_staniv;

public static List<Node> Qp;

public static int[,] Goal = new int[3, 3]{

{1,2,3},

{4,5,6},

{7,8,0}

};

public static Queue<Node> fringe = new Queue<Node>();

public static Node Up(Node per)

{

int l = -1, m = -1;

int[,] NewState = copy(per.State);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (NewState[i, j] == 0)

{

l = i;

m = j;

};

}

}

if (l == 0) return null;

NewState[l, m] = NewState[l - 1, m];

NewState[l - 1, m] = 0;

Node shild = new Node(NewState, per);

shild.action = "Up";

return shild;

}

public static Node Dawn(Node per)

{

int l = -1, m = -1;

int[,] NewState = copy(per.State);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (NewState[i, j] == 0)

{

l = i;

m = j;

};

}

}

if (l == 2) return null;

NewState[l, m] = NewState[l + 1, m];

NewState[l + 1, m]=0;

Node shild = new Node(NewState, per);

shild.action = "Dawn";

return shild;

}

public static Node Left(Node per)

{

int l = -1, m = -1;

int[,] NewState = copy(per.State);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (NewState[i, j] == 0)

{

l = i;

m = j;

};

}

}

if (m == 2) return null;

NewState[l, m] = NewState[l, m + 1];

NewState[l, m + 1] = 0;

Node shild = new Node(NewState, per);

shild.action = "Left";

return shild;

}

public static Node Right(Node per)

{

int l = -1, m = -1;

int[,] NewState = copy(per.State);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (NewState[i, j] == 0)

{

l = i;

m = j;

};

}

}

if (m == 0) return null;

NewState[l, m] = NewState[l, m - 1];

NewState[l, m - 1] = 0;

Node shild = new Node(NewState, per);

shild.action = "Right";

return shild;

}

public static List<Node> Expand(Node per )

{

List<Node> successors = new List<Node>();

Node shild;

shild=Up(per);

if (shild != null) successors.Add(shild);

shild = Left(per);

if (shild != null) successors.Add(shild);

shild = Dawn(per);

if (shild != null) successors.Add(shild);

shild = Right(per);

if (shild != null) successors.Add(shild);

return successors;

}

public static void vivod (Node n)

{

Console.WriteLine("matr:");

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

Console.Write(n.State[i,j]+"\t");

}

Console.WriteLine();

}

}

public static bool Goal\_Test(Node n)

{

for(int i=0; i < 3; i++)

{

for(int j=0; j < 3; j++)

{

if (n.State[i, j] != Goal[i, j]) return false;

}

}

return true;

}

public static Node tree\_search\_BFS(Node problem, int d=15)

{

kilk\_staniv = 1;

Node n;

fringe.Enqueue(problem);

while (fringe.Count != 0)

{

if (fringe.Count == 0) return null;

n = fringe.Dequeue();

if (Goal\_Test(n)) return n;

List<Node> shild = Expand(n);

if (shild!=null) kilk\_staniv += shild.Count;

if (shild != null) for (int i = 0; i < shild.Count; i++)

{

fringe.Enqueue(shild[i]);

}

if (n.Depth==d)

{

Console.WriteLine($"glubuna dereva poshuku BFS {d}, rishennia ne znaishov");

return null;

}

}

return null;

}

public static int[,] copy(int[,] mas)

{

int[,] rez = new int[3,3];

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

rez[i, j] = mas[i, j];

}

}

return rez;

}

public static void Vivod\_solution(Node solution)

{

if (solution == null) return;

Stack<Node> rez = new Stack<Node>();

while (solution.Parent\_Node != null)

{

rez.Push(solution);

solution = solution.Parent\_Node;

}

while (rez.Count!=0)

{

vivod(rez.Pop());

}

}

public static int h(Node n)

{

int h = 0;

for (int i=0; i<3; i++)

{

for (int j=0; j<3; j++)

{

int x = n.State[i, j];

switch (x)

{

case 1: h += Math.Abs(0 - i) + Math.Abs(0 - j); break;

case 2: h += Math.Abs(0 - i) + Math.Abs(1 - j); break;

case 3: h += Math.Abs(0 - i) + Math.Abs(2 - j); break;

case 4: h += Math.Abs(1 - i) + Math.Abs(0 - j); break;

case 5: h += Math.Abs(1 - i) + Math.Abs(1 - j); break;

case 6: h += Math.Abs(1 - i) + Math.Abs(2 - j); break;

case 7: h += Math.Abs(2 - i) + Math.Abs(0 - j); break;

case 8: h += Math.Abs(2 - i) + Math.Abs(1 - j); break;

case 0: h += Math.Abs(2 - i) + Math.Abs(2 - j); break;

default: break;

}

}

}

return h;

}

public static bool Contain (List<Node> l, Node n)

{

if (l == null) return false;

if (l.Count == 0) return false;

for(int k=0; k<l.Count; k++)

{

bool rez = true;

for (int i=0; i<3; i++)

{

for (int j=0; j<3; j++)

{

if (l[k].State[i, j] != n.State[i, j]) rez = false;

}

}

if (rez) return rez;

}

return false;

}

public static Node A\_star(Node Start)

{

// Q — множество вершин, которые требуется рассмотреть,

// U — множество рассмотренных вершин,

// f[x] — значение эвристической функции "расстояние + стоимость" для вершины x,

// g[x] — стоимость пути от начальной вершины до x,

// h(x) — эвристическая оценка расстояния от вершины x до конечной вершины.

kilk\_staniv = 1;

List<Node> U=new List<Node>();

List<Node> Q=new List<Node>();

Q.Add(Start);

Start.Path\_Cost = 0;

Start.f = Start.Path\_Cost + h(Start);

while (Q.Count != 0)

{

Q.Sort((x, y) => x.f - y.f);

Node current = Q[0];

if (Goal\_Test(current)) return current;

Q.Remove(current);

U.Add(current);

List<Node> vl = Expand(current);

if (vl != null) kilk\_staniv += vl.Count;

foreach (Node v in vl)

{

int tentativeScore = current.Path\_Cost + 1;

if ((Contain(U,v))&&(tentativeScore >= v.Path\_Cost)) continue;

if ((!(Contain(U,v))) || (tentativeScore < v.Path\_Cost)) {

v.Parent\_Node = current;

v.Path\_Cost = tentativeScore;

v.f = v.Path\_Cost + h(v);

if (!(Contain(Q,v))) Q.Add(v);

}

}

}

return null;

}

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

Node tree = new Node(matr, null);

Console.WriteLine("BFS:");

vivod(tree);

Vivod\_solution(tree\_search\_BFS(tree));

Console.WriteLine($"killist\_staniv={kilk\_staniv}");

Console.WriteLine("A\*:");

vivod(tree);

Vivod\_solution(A\_star(tree));

Console.WriteLine($"killist\_staniv={kilk\_staniv}");

}

}

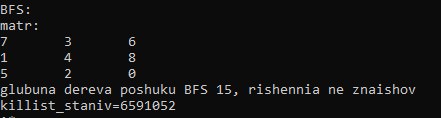
}

### Приклади роботи

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

Рисунок 3.1 – Алгоритм пошук в ширину

Пр1



Пр2

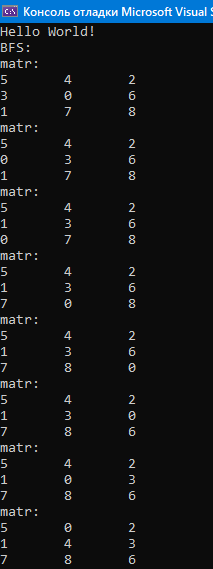
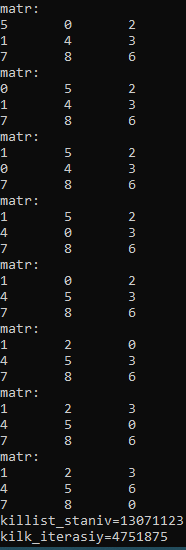
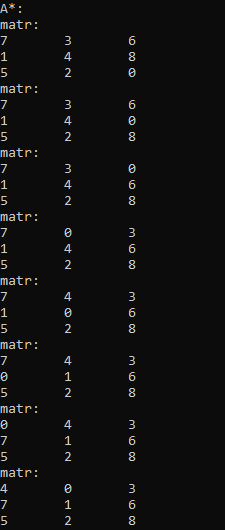
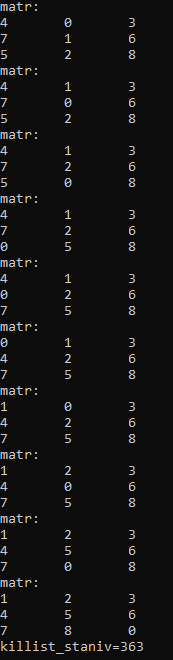
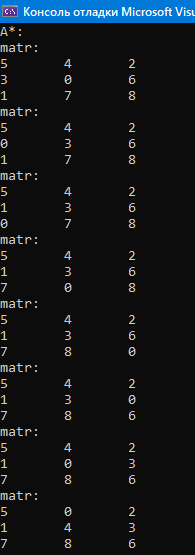
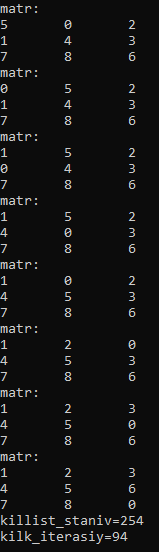
 

Рисунок 3.2 – А\* з манхетенською відстанню

Пр1:

ПР2:

Дослідження алгоритмів

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

Таблиця 3.1 – Характеристики оцінювання алгоритму пошук в ширину

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пaм’яті |
|  | 2396746 |  |  | 6591052  (2 гіга оперативки зайняло і не знайшло) |
|  | 4751875 |  |  | 13071123 |
|  | 36 |  |  | 99 |
| {1,2,3}, {7,4,5}, {8,0,6} | 255 |  |  | 702 |
|  | 23 |  |  | 62 |
|  | 10 |  |  | 27 |
|  | 3 |  |  | 6 |
|  | 17 |  |  | 46 |
|  | 54 |  |  | 147 |
|  | 177 |  |  | 494 |

Я брав стан з готовим розв'язком і переміщав в ньому цифри, а якби брав рандомний, то в половині випадків не знайшло б розв'язок. Інші 10 станів без розв'язку.

В таблиці 3.2 наведені характеристики оцінювання алгоритму А\*манхетен, задачі 8 пазл для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання алгоритму А\* з манхетенською відстанню

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
|  | 133 |  |  | 363 |
|  | 94 |  |  | 254 |
|  | 5 |  |  | 13 |
| {1,2,3}, {7,4,5}, {8,0,6} | 7 |  |  | 18 |
|  | 4 |  |  | 11 |
|  | 3 |  |  | 8 |
|  | 2 |  |  | 4 |
|  | 4 |  |  | 11 |
|  | 5 |  |  | 13 |
|  | 6 |  |  | 16 |

Висновок

При виконанні даної лабораторної роботи було розглянуто гру 8 пазл, написав неіформативний алгоритм пошуку розв’язку в шир, і інформативний А\* з манхетеською відстанню. В половині випадків 8-пазл не має розв’язку. Судячи по табличці А\* значно ефективніше ніж пошук в ширину використовує пам’ять і час комп’ютера.

Ссилка на гідхаб з лабами: https://github.com/kolya-sp/alg\_2