Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №2

з дисципліни «Системи штучного інтелекту»

на тему

«Класичні методи пошуку рішень у просторах станів»

Виконав:

студент гр. ІС-91

Косюк Михайло

Київ – 2021

**Постановка задачі**

Реалізувати програму, яка розв’язує поставлену задачу 8-puzzle за допомогою алгоритму неінформативного пошуку BFS та алгоритму інформативного пошуку RBFS, що використовує задану евристичну функцію H1 - кількість фішок, які не стоять на своїх місцях .

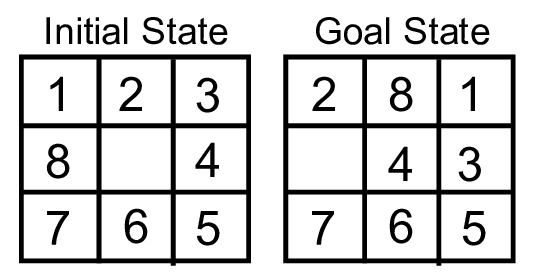
Провести серію експериментів для вивчення ефективності роботи алгоритмів. Кожний експеримент повинен відрізнятись початковим станом. Серія повинна містити не менше 20 експериментів для кожного алгоритму. За проведеними серіями необхідно визначити:

* середній час пошуку рішення у секундах
* середню кількість згенерованих станів під час пошуку
* середню кількість станів, що зберігаються в пам’яті під час роботи програми.

**Формулювання проблеми та опис простору станів**

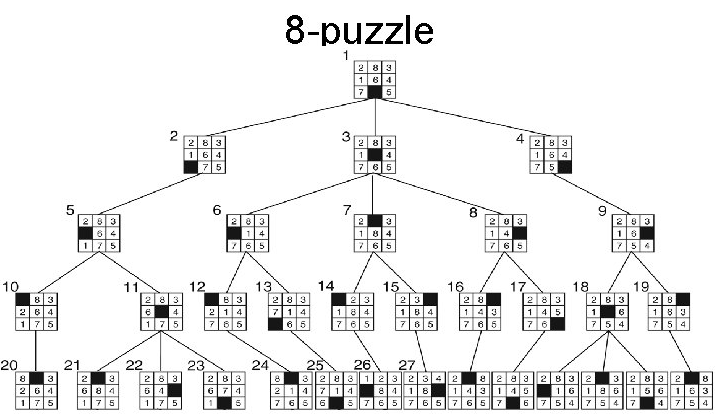
У даній роботі розв’язується задача **8-puzzle**, або простіше – п’ятнашки.

Дано дошку 3×3 з 8 плитками (кожна плитка має одне число від 1 до 8) і одне порожнє місце. Мета полягає в тому, щоб розмістити цифри на плитку так, щоб **відповідати цільовій конфігурації з урахуванням порожнього простору**, частіше за все це складання картинок. Ми можемо пересувати плитки в чотирьох напрямах **вліво, вправо, вниз, вгору**.

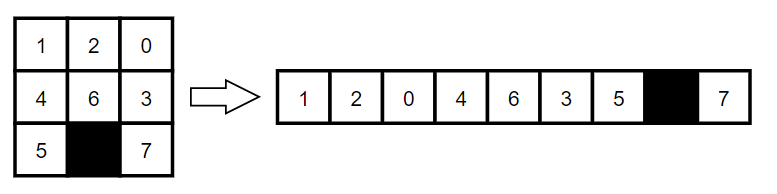


У реальній грі для того щоб перемістити положення будь-якої фішки на порожнє місце, насправді необхідно зробити декілька пересувань, але для спрощення вирішення даної задачі можна уявити, що рухається тільки пусте місце, шляхом зміни положень із будь-якою сусідньою плиткою, справді, якщо пересунути плитку на пусте місце, то пусте місце з’явиться на минулій позиції цієї плитки – отримуємо **уявне пересування пустого місця**.

Простором станів даної задачі є **дерево**, кожний з вузлів якого може мати 2, 3 або 4 дочірніх вузли, в залежності від позиції пустого місця: центр – 4 можливих нащадки (ліво, право, вниз вгору), центр грані – 3 можливих нащадки, кут – 2 можливих нащадки.



Останнім важливим фактом треба відмітити те, що в даному випадку можна провести **згортання простору** в рамках кожного стану, в реальному світі задача представляє із себе таблицю або матрицю, але уявно ніщо не заважає згорнути двовимірну реально структуру в одновимірний масив та пересувати пусте місце по ньому. Це дозволить спростити розробку та покращити швидкодію програми, не дивлячись на те, що розмір матриці мінімальний, для обходу двовимірного простору потребується час **О(n^2)**, тоді як для одновимірного лише **О(n)**.



Для опису простору станів в коді був створений клас Puzzle8, за допомогою якого можна зручно створювати нові вузли.

class Puzzle8 {

  static goalState = [1,2,3,8,0,4,7,6,5];

  static moves = {up: 'U', down: 'D', left: 'L', right: 'R'};

  static numberOfInstances = 0;

  state = [];

  parent = undefined;

  currentMove = '';

  usesHeuristic = false;

  heuristicValue = 0;

  pathCost = 0;

  eval = 0;

**Опис евристичної функції оцінки**

Інформований алгоритм задачі використовує евристичну функцію оцінки кількості плиток, що не стоять на своїх місцях порівняно із цільовим станом. Така функція дозволить давати точно **визначену чисельну оцінку кожному стану**. Прихід до рішення задачі, очевидно, повинен іти шляхом **мінімізації** цієї **функції**, оскільки значення даної евристичної функції для цільового стану – 0.

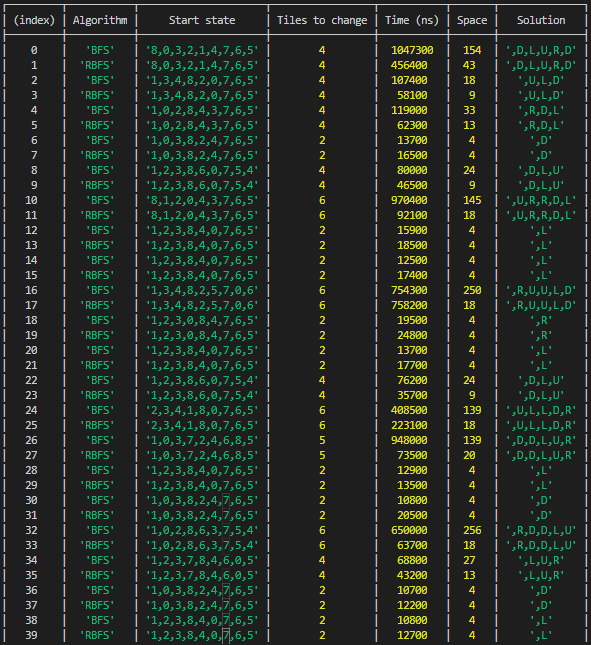
Також, завдяки проведеному згортанню простору станів, обчислення евристичної функції може бути проведеним за лінійний час та написане одним рядком коду.

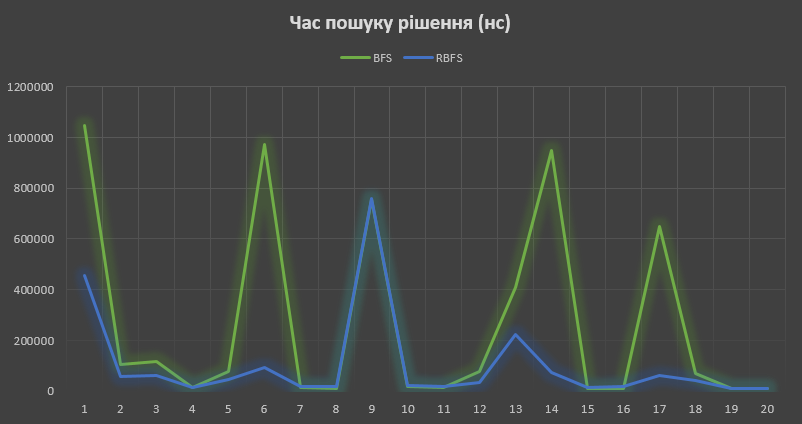
calcHeuristicValue () {

    this.heuristicValue = this.state.filter((tile, index) => tile != Puzzle8.goalState[index]).length;

  }

**Результати проведення серії експериментів у вигляді таблиць та графіків. Навести результати по всіх алгоритмах і дати їх розширене порівняння.**







**Аналіз отриманих результатів**

**BFS**: повний (якщо рішення існує, то воно буде знайдене), оптимальний (в даному випадку незваженого дерева)

**RBFS**: неповний (унаслідує від DFS), оптимальний (при допустимості евристичної функції).

Як бачимо за результатами RBFS є кращим вибором як за швидкодією так і за споживанням пам’яті, але тільки при **умові невеликої глибини залягання рішення в дереві**. Якщо рішення знаходиться глибоко в дереві RBFS стикнеться із проблемою великого часу роботи через збільшення середньої глибини рекурсії, а при замалому F- ліміті, навпаки, може не знайти рішення взагалі, хоча оцінити часову складність алгоритму в такому випадку буде важко. Суттєвою проблемою є неможливість визначення повторюваних станів, що може призвести до експоненціального росту складності. Також проблемою RBFS є дуже обмежене споживання пам’яті, при доступності більшого об’єму пам’яті, він просто не зможе нею скористатись.