**Звіт до комп’ютерного практикуму №2.**

**Класичні методи пошуку рішень у просторах станів**

**ПІБ:** Шляхтун Денис Михайлович.

**Група:** ІП-14.

**Мета:** Розглянути та дослідити алгоритми неінформативного та інформативного пошуку в моделях на основі станів. Провести порівняльний аналіз ефективності використання алгоритмів.

**Завдання:** Реалізувати програму, яка розв’язує поставлену задачу Task за допомогою алгоритму неінформативного пошуку AlgNoInf та алгоритму інформативного пошуку AlgInf, що використовує задану евристичну функцію Func. Провести серію експериментів для вивчення ефективності роботи алгоритмів.

**Номер варіанту:** 22.

**Завдання для варіанту:** 8-puzzle, BFS, A\*, H2.

**Формалізація задачі:**

Стан – це позиції фішок від 1 до 8 та порожнє місце, що позначається як 0, на матриці 3х3.

Дії, які можуть виконуватися – переміщення фішки на порожнє місце, якщо ця фішка межує з нею.

Цільовий стан:

Зображення, що містить текст, знімок екрана, Шрифт, чорно-білий

Автоматично згенерований опис

Початковий стан генерується шляхом випадкового переміщення фішок кінцевого стану.

**Дерево пошуку:** У кожного вузла наступник – це усі можливі переміщення фішок по ігровій дошці на даному стані:

Зображення, що містить схема, ряд, План, Паралель

Автоматично згенерований опис

Також для виконання роботи було розроблено метод створення нащадків з перевіркою усіх предків поточного вузла, щоб не створювалися повторні нащадки. Надалі алгоритми, у яких застосовуватиметься цей метод, будуть підписані «+М».

**Алгоритм неінформованого пошуку:** Пошук у ширину (BFS) – алгоритм пошуку на графі. Здійснюється за допомогою звичайної FIFO черги. Вартість переходу – константа. Вважається повним оптимальним алгоритмом, у пам’яті зберігає вузли поточного рівня та відкриті дочірні.

**Програмна реалізація алгоритму неінформованого пошуку:** тільки основна(-ні) процедури

def BFS(start, is\_history\_used):

start\_timer = timer()

max\_state\_num\_at\_once = 0

queue = list()

num\_of\_generated\_nodes = 1

queue.append(start)

while queue:

if timer() - start\_timer < MAX\_TIME and len(queue) < MAX\_ELEMENTS\_AT\_ONCE:

max\_state\_num\_at\_once = max(max\_state\_num\_at\_once, len(queue))

node = queue.pop(0)

if node.is\_puzzle\_solved():

break

for child in node.get\_children(is\_history\_used):

num\_of\_generated\_nodes+=1

queue.append(child)

else:

break

if not node.is\_puzzle\_solved():

node = None

return node, num\_of\_generated\_nodes, max\_state\_num\_at\_once

**Алгоритм інформованого пошуку:** A\* – евристичний алгоритм пошуку на графі. Працює за допомогою черги пріоритетів, що впорядковує вузли за функцією оцінки. Функція оцінки складається з вартості досягнення поточного вузла з початкового та оціненої вартості досягнення цільового вузла з поточного. Вважається повним оптимальним алгоритмом, у пам’яті зберігає усі нерозгорнуті вузли.

**Евристична функція оцінки:** сума відстаней всіх фішок до їх цільових станів, Манхетенська відстань.



**Програмна реалізація алгоритму інформованого пошуку:**

def a\_star\_search(start, is\_history\_used):

start\_timer = timer()

frontier = PriorityQueue()

frontier.put((0, start))

came\_from = {}

cost\_so\_far = {}

came\_from[start] = None

cost\_so\_far[start] = 0

counter = 1

current = None

max\_queue\_size = 1

while not frontier.empty():

if timer() - start\_timer < MAX\_TIME and len(frontier.elements) < MAX\_ELEMENTS\_AT\_ONCE:

max\_queue\_size = max(max\_queue\_size, len(frontier.elements))

current = frontier.get()

if current.is\_puzzle\_solved():

break

for next in current.get\_children(is\_history\_used):

new\_cost = cost\_so\_far[current] + 1

if next not in cost\_so\_far or new\_cost < cost\_so\_far[next]:

cost\_so\_far[next] = new\_cost

priority = new\_cost + next.heuristic()

frontier.put((priority, next))

came\_from[next] = current

counter += 1

else:

break

if not current.is\_puzzle\_solved():

current = None

return current, counter, max\_queue\_size

**Приклад роботи розробленої програми:**

Зображення, що містить текст, знімок екрана, Шрифт, чорно-білий

Автоматично згенерований опис

**Результати використання алгоритмів:**

Зображення, що містить текст, знімок екрана, меню, чорно-білий

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, меню, чорно-білий

Автоматично згенерований опис

Зображення, що містить текст, меню, знімок екрана, чорно-білий

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, меню

Автоматично згенерований опис

Зображення, що містить текст, Графік, ряд, схема

Автоматично згенерований опис

Зображення, що містить Графік, ряд, текст, схема

Автоматично згенерований опис

Зображення, що містить текст, Графік, ряд, схема

Автоматично згенерований опис

Середні значення показників:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

**Оцінка результатів навчання:**

Обидва алгоритми є повними та оптимальними, адже в усіх випадках (крім експерименту 16 для BFS через переповнення пам’яті) знаходився найкоротший шлях від початкового до кінцевого стану, у цьому можна засвідчитися, порівнявши шляхи усіх чотирьох проходів над одним початковим станом.

Порівнюючи середні значення показників, кількість згенерованих вузлів та максимальна кількість одночасних вузлів у BFS майже у 10 разів більша за A\*, по графіку порівняння часу видно, що BFS часто займає набагато більше часу для виконання, але через переповнення пам’яті в одному з експериментів чисельно час порівняти не вдалося. Менша кількість вузлів для А\* зумовлена функцією оцінки, яка «направляє» до кінцевого результату, коли BFS перебирає усі можливі вершини.

Цікаво порівняти алгоритми, доповнені створенням нащадків з перевіркою попередників поточної вершини. Алгоритм BFS з такою перевіркою займає майже у 5 разів менше часу за А\*, згенерував у середньому таку ж кількість вузлів, а одночасно існувало менше на 25% вузлів. Алгоритм А\* з перевіркою попередників поточної вершини взагалі виявився дуже хорошим, зменшивши середній час більш ніж у 100 разів у порівнянні зі звичайним створенням нащадків, у середньому генерував у 70 разів менше вузлів та одночасно мав у 100 разів менше вузлів.

Практичні розрахунки довели теоретичні відомості, а додавання перевірки історії нащадків перевершила усі очікування, особливо для алгоритму А\*, адже він використовує функцію оцінки, яка і так перешкоджає проходу на менш вигідний шлях.