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

**Класичні методи пошуку рішень у просторах станів**

**ПІБ: Мєшков Андрій Ігорович, Ткач Владислав Анатолійович**

**Група:** **ІП-15**

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

***З*авдання:** Реалізувати програму, яка розв’язує поставлену задачу 8-ферзів за допомогою алгоритму неінформативного пошуку BFS та алгоритму інформативного пошуку RBFS, що використовує задану евристичну функцію F2 – кількість пар ферзів, які б’ють один одного без урахування видимості. Провести серію експериментів для вивчення ефективності роботи алгоритмів.

Програму реалізувати на довільній мові програмування.

**Номер варіанту: 23**

**Завдання для варіанту:**

****

**Формалізація задачі:**

**Стани**: Станом задачі є конфігурація розміщення 8 ферзів на шаховій дошці. Кожен ферзь розміщується на одній із 64 клітинок дошки, тому можливі 64^8 різних станів.

**Дії**: Дії включають переміщення ферзя з однієї клітини на іншу. Однак, так як ферзі не можуть рухатись по горизонталі, вертикалі та діагоналям без перешкод, деякі дії можуть бути неприпустимими. Такі дії потрібно виключити з множини можливих дій.

**Початковий стан**: Усі 8 ферзів розміщені на дошці.

**Цільовий стан**: Цільовим станом є така конфігурація ферзів, коли жоден з них не б'ється іншого. Це означає, що жоден ферзь не може перебувати на тій самій горизонталі, вертикалі або діагоналі з іншим ферзем.

**Алгоритм неінформативного пошуку**: BFS, перша позиція за списком.

**Алгоритм інформативного пошуку**: RBFS, евристична функція.

**Евристична функція**: кількість пар ферзів, які б’ють один одного без урахування видимості.

**Дерево пошуку:** Функція `GenerateSuccessors` генерує наступників для заданого стану дошки з ферзями. Кожен наступник є новою конфігурацією ферзів на дошці, яка відрізняється від початкової лише однією дією - переміщенням одного ферзя на нове місце.

Отже, основна ідея полягає в переборі всіх можливих комбінацій переміщень ферзя і визначенні конфігурацій з найменшою кількістю конфліктів. Для цього для кожного ферзя перебирається кожна пуста клітина на дошці, і для кожної такої комбінації обчислюється кількість конфліктів. Обрана та додана до списку наступників та конфігурація з найменшою кількістю конфліктів.

**Алгоритм неінформованого пошуку:**

Алгоритм BFS (Breadth-First Search) - це метод пошуку в ширину, що застосовується для визначення найкоротшого шляху у незваженому графі. Він працює, обробляючи спочатку вузли, що знаходяться на відстані 1 від початкового вузла, потім на відстані 2 і так далі, до досягнення цільового вузла або перебору всіх доступних вузлів. Для збереження станів, які потрібно обробити, використовується черга. Порівняно з DFS (Depth-First Search), BFS зазвичай не проводить повного перебору всіх шляхів до досягнення цільового вузла, оскільки спрямований на знаходження найкоротшого шляху. Однак у BFS може бути значна витрата пам'яті у випадку, якщо граф містить багато вузлів. Узагальнюючи, BFS - це метод пошуку в ширину, що використовує чергу для зберігання станів, які потрібно обробити, та визначає найкоротший шлях у незваженому графі.

Характеристики алгоритму:

1) Коректність: у випадку наявності шляху BFS завжди знаходить найкоротший маршрут від стартового вузла до цільового в невзваженому графі.

2) Повнота: якщо маршрут існує, BFS завжди досягне цільового вузла.

3) Часова складність: ефективність алгоритму залежить від кількості ребер та вузлів у графі. У найгіршому випадку, коли граф є повним, складність становить O(n^2). У найкращому випадку, коли цільовий вузол знаходиться на першому кроці, складність - O(1).

4) Використання пам'яті: для роботи BFS необхідна пам'ять для черги, яка зберігає стани, які потрібно опрацювати. У найгіршому випадку, коли граф має максимальну кількість вузлів, складність пам'яті становить O(n).

5) Розширення: BFS легко може бути розширений для розв'язання задач у вагованому графі або для пошуку всіх можливих маршрутів, а також для визначення діаметра графа.

**Програмна реалізація алгоритму неінформованого пошуку:**

public static Board? SearchBfs(Board initialBoard, CancellationToken cancellationToken)

{

var visitedCount = 0;

var queue = new Queue<Board>();

var visited = new HashSet<Board>();

queue.Enqueue(initialBoard);

visited.Add(initialBoard);

*while* (queue.Count > 0)

{

*try*

{

cancellationToken.ThrowIfCancellationRequested();

}

*catch* (OperationCanceledException e)

{

*return* null;

}

var currentBoard = queue.Dequeue();

visitedCount++;

*if* (currentBoard.GetConflicts() == 0)

{

*return* currentBoard;

}

GenerateSuccessors(currentBoard);

*foreach* (var successor *in* currentBoard.Successors.Where(successor => !visited.Contains(successor)))

{

queue.Enqueue(successor);

visited.Add(successor);

}

}

*return* null;

}

**Алгоритм інформованого пошуку:**

Алгоритм інформованого пошуку - це метод пошуку, що використовує додаткову інформацію про простір пошуку для ефективного наведення. Принцип його роботи полягає у використанні додаткових даних, таких як оцінка вартості або відстані до цільового стану, для прийняття рішення про напрямок пошуку та вибір наступного кроку.

Алгоритм обмеженого інформованого пошуку (RBFS - Recursive Best-First Search) є одним із варіантів інформованого пошуку. Він використовує принцип пошуку "найкращий перший" (Best-First Search), який вибирає наступний вузол для розгляду на основі оцінки його відстані до цільового вузла. Проте RBFS використовує рекурсивну стратегію, щоб уникнути повного перебору всіх можливих шляхів.

Принцип роботи RBFS полягає у використанні рекурсивного підходу до пошуку найкращого вузла. На кожному кроці він вибирає наступний вузол для розгляду, оцінюючи його вартість, та продовжує пошук в цьому напрямку, доки не досягне кінцевого стану. При досягненні межі обмеження, він використовує збережені інформацію про оцінки вартості для повернення до іншого вузла та продовження пошуку.

Отже, RBFS поєднує принципи пошуку "найкращий перший" з рекурсивною стратегією, що дозволяє ефективно використовувати інформацію про оцінки вартості для керування пошуковим процесом.

**Евристична функція оцінки:** кількість пар ферзів, які б’ють один одного без урахування видимості.

public int GetConflicts()

{

\_conflicts ??= (from queenX in Queens

from queenY in Queens

where queenX != queenY

where queenX.X == queenY.X || queenX.Y == queenY.Y ||

Math.Abs(queenX.X - queenY.X) == Math.Abs(queenX.Y - queenY.Y)

select queenX).Count();

*return* \_conflicts.Value;

}

**Програмна реалізація алгоритму інформованого пошуку:**

public static Board? SearchRbfs(Board board, CancellationToken cancellationToken)

{

var states = new List<Board>();

var visited = 0;

*try*

{

var finalBoard = Search(board, states, cancellationToken, ref visited);

*return* finalBoard;

}

*catch* (Exception e)

{

*return* null;

}

}

private static Board? Search(Board board, List<Board> states, CancellationToken cancellationToken, ref int visited)

{

cancellationToken.ThrowIfCancellationRequested();

*if* (board.GetConflicts() == 0)

{

*return* board;

}

GenerateSuccessors(board);

states.AddRange(board.Successors);

var nextState = states.MinBy(s => s.GetConflicts());

*if* (nextState is null)

{

*return* null;

}

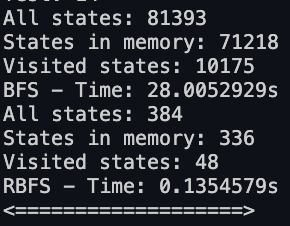
states.Remove(nextState);

visited++;

*return* Search(nextState, states, cancellationToken, ref visited);

}

**Приклад роботи розробленої програми:**



**Результати використання алгоритмів:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Експеримент** | **Алгоритм** | **Час, с** | **Загальна кількість відвіданих вузлів** | **Максимальна кількість вузлів, що одночасно зберігалися в пам'яті** | **Загальна кількість згенерованих вузлів** |
| **1** | **BFS** | 0.6082313 | 108 | 749 | 857 |
| **RBFS** | 0.016597 | 3 | 21 | 24 |
| **2** | **BFS** | 24.675997 | 7908 | 55349 | 63257 |
| **RBFS** | 13.0157831 | 2979 | 20853 | 23832 |
| **3** | **BFS** | 13.9630486 | 5087 | 35602 | 40689 |
| **RBFS** | 0.0508582 | 17 | 119 | 136 |
| **4** | **BFS** | 2.1789911 | 733 | 5124 | 5857 |
| **RBFS** | 0.0110129 | 4 | 28 | 32 |
| **5** | **BFS** | 1.8350518 | 540 | 3773 | 4313 |
| **RBFS** | 1.9813313 | 509 | 3563 | 4072 |
| **6** | **BFS** | 78.952643 | 27804 | 194621 | 222425 |
| **RBFS** | 0.7441512 | 245 | 1715 | 1960 |
| **7** | **BFS** | 5.0454895 | 1888 | 13209 | 15097 |
| **RBFS** | 0.0111107 | 4 | 28 | 32 |
| **8** | **BFS** | 0.316292 | 116 | 805 | 921 |
| **RBFS** | 0.0141717 | 5 | 35 | 40 |
| **9** | **BFS** | 6.9315522 | 2619 | 18326 | 20945 |
| **RBFS** | 0.0178698 | 5 | 35 | 40 |
| **10** | **BFS** | 13.2206794 | 4935 | 34538 | 39473 |
| **RBFS** | 0.6914035 | 244 | 1708 | 1952 |
| **11** | **BFS** | 35.9216072 | 13583 | 95074 | 108657 |
| **RBFS** | 2.0022692 | 671 | 4697 | 5368 |
| **12** | **BFS** | 17.6617757 | 6665 | 46648 | 53313 |
| **RBFS** | 0.2718692 | 98 | 686 | 784 |
| **13** | **BFS** | 15.7263039 | 5955 | 41678 | 47633 |
| **RBFS** | 0.0181858 | 6 | 42 | 48 |
| **14** | **BFS** | 28.0052929 | 10175 | 71218 | 81393 |
| **RBFS** | 0.1354579 | 48 | 336 | 384 |
| **15** | **BFS** | 18.4800403 | 6550 | 45843 | 52393 |
| **RBFS** | 0.0691742 | 20 | 140 | 160 |
| **16** | **BFS** | 1.0684742 | 389 | 2716 | 3105 |
| **RBFS** | 0.0114169 | 4 | 28 | 32 |
| **17** | **BFS** | 0.2800772 | 104 | 721 | 825 |
| **RBFS** | 0.0083262 | 3 | 21 | 24 |
| **18** | **BFS** | 4.5918898 | 1703 | 11914 | 13617 |
| **RBFS** | 0.0177599 | 6 | 42 | 48 |
| **19** | **BFS** | 35.0852678 | 12977 | 90832 | 103809 |
| **RBFS** | 0.0173382 | 6 | 42 | 48 |
| **20** | **BFS** | 8.8926628 | 3130 | 21903 | 25033 |
| **RBFS** | 0.0894131 | 22 | 154 | 176 |

**Оцінка результатів навчання:**

Аналізуючи наведені результати, можна зробити кілька висновків:

Час виконання: RBFS продемонстрував значно кращі показники часу виконання у порівнянні з BFS у більшості випадків. У експериментах час виконання RBFS був відчутно меншим, ніж у BFS. Це підтверджує теоретичні передбачення про те, що RBFS може бути ефективнішим алгоритмом за рахунок використання інформованого пошуку.

Просторова складність: RBFS виявився ефективнішим у використанні пам'яті у порівнянні з BFS. У багатьох випадках, RBFS зберігав у пам'яті меншу кількість вузлів, що може бути важливим у великих або складних просторах пошуку.

Обчислювальна складність: Оцінюючи загальну кількість відвіданих та згенерованих вузлів, можна побачити, що RBFS також ефективно працює у відношенні обчислювальної складності. Він здатний досягати бажаних результатів за меншу кількість кроків, що може значно зменшити обчислювальні витрати.

Повнота: Обидва алгоритми виявилися повними, адже вони знаходять розв'язок, якщо такий існує. Таким чином, результати підтверджують теоретичні передбачення щодо повноти обох алгоритмів.

Отже, практичні результати підтверджують теоретичні твердження про ефективність RBFS порівняно з BFS у багатьох випадках. RBFS виявився оптимальним за часом виконання, просторовою та обчислювальною складностями, що робить його привабливим вибором для задач, де важлива швидкість та оптимальність виконання.