МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Кафедра дискретної математики

Звіт

про виконання індивідуального завдання

Підготував

студент групи ПМІ-41

*Михайлович Орест*

Львів – 2011Задача «П’ятнашки»

1. **Опис задачі**

Задача полягає у розташуванні п’ятнадцяти пронумерованих комірок у порядку спадання. Комірки розташовані на дошці 4х4. Головоломка вважається розв’язаною, коли усі комірки від 1 до 4 розташовані у першому ряді, 5-8 у другому, 9-12 у третьому і 13-15 у четвертому ряді, порожня комірка займає останню позицію. За один крок можна поміняти місцями порожню комірку з довільною коміркою, яка має спільну сторону з нею. Задача може бути узагальнена до NxN випадку з N2-1 пронумерованою коміркою. У початковому стані комірки розташовані довільно.

1. **Умова розв’язності**

Кількість можливих станів для задачі NxN є N! але лише половина з них є розв’язними.

Нехай комірка з числом *і* розташована перед *ni* комірками з числами меншими за *і.* Позначимо номер рядка де розташована пуста комірка через *e.*



Розв’язок існує тоді і тільки тоді, коли M- парне.

1. **Постановка задачі**
2. Початковий стан – довільний
3. Операції над станом:
   1. перемістити порожню комірку вверх
   2. перемістити порожню комірку вниз
   3. перемістити порожню комірку вправо
   4. перемістити порожню комірку вліво

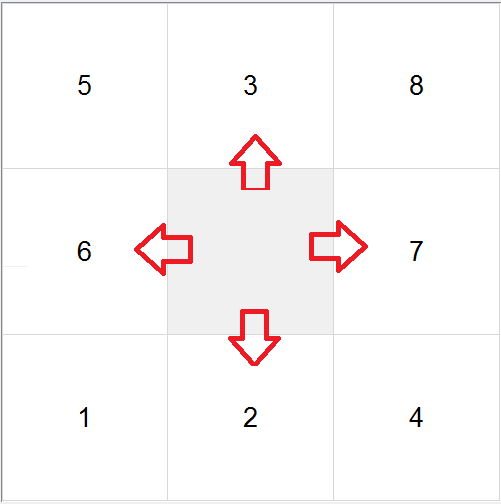


Рис. 1. Операції над станом

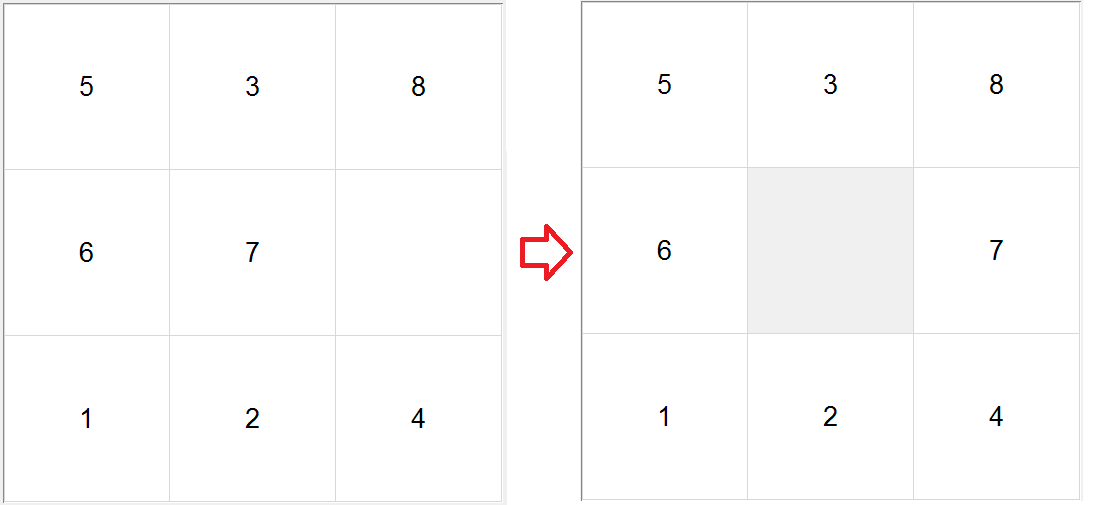
1. Модель переходу: операція «вверх», «вниз», «вправо», «вліво» задається відносно пустої комірки. В результаті виконання операції порожня комірка міняється місцями з сусідньою у вказаному напрямку  
   

Рис. 2. Модель переходу між станми

1. Перевірка цільового стану – усі комірки розташовані у порядку спадання, порожня комірка розташована у останній позиції.

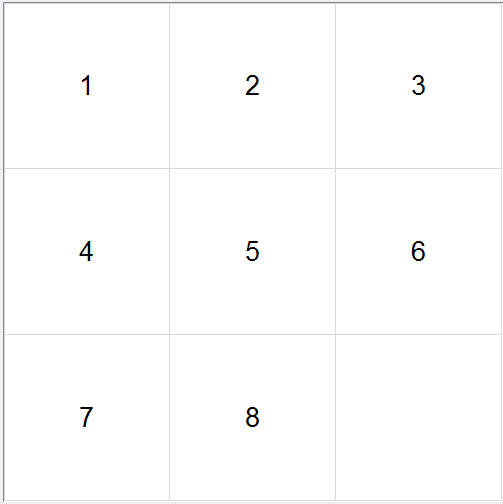


Рис. 3. Цільовий стан

1. Ціна операції - *С*(*S*, *A*, *S'*) = 1
2. **Реалізація**

Для реалізації пошукових алгоритмів та візуалізації результату я обрав технологію Qt (v. 4.7.1), мова програмування – С++, середовище – Qt Creator (2.1.0).

Для генерації документації використав doxygen (1.7.4).

Код доступний на:  
svn checkout [**http**://sm-lnu.googlecode.com/svn/trunk/branches/Qt/AI](http://sm-lnu.googlecode.com/svn/trunk/branches/Qt/AI)

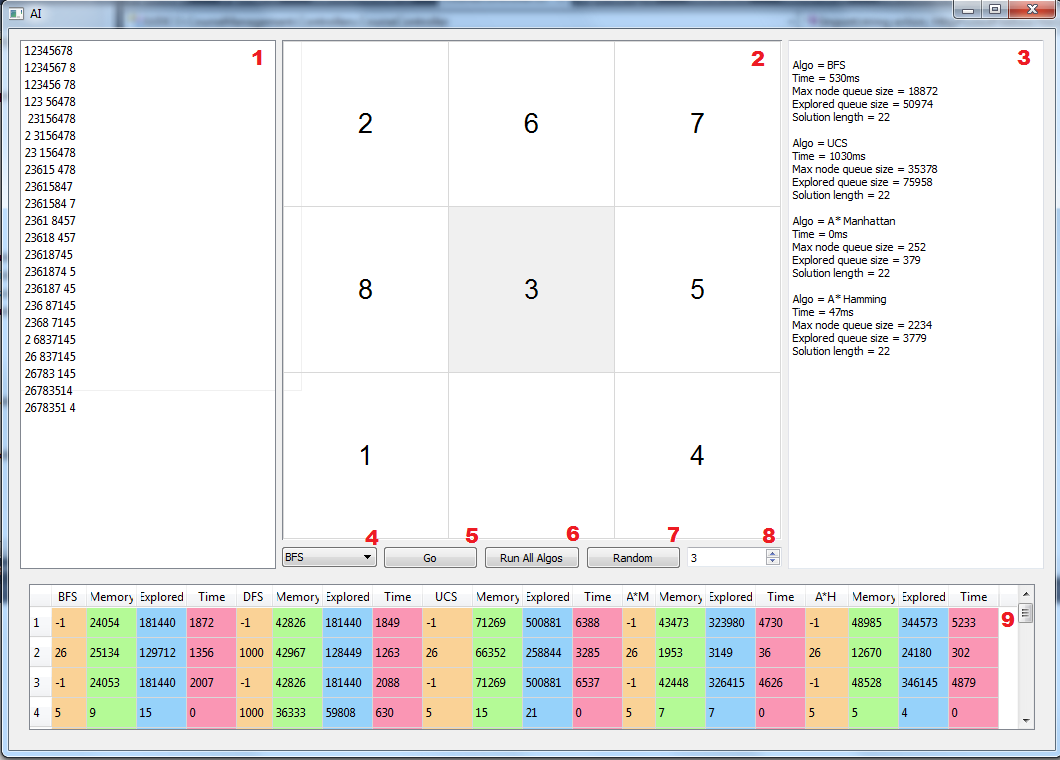


Рис. 4. Інтерфейс програми

**1** – список станів, які увійшли в розв’язок;

**2** – таблиця, для задання початкового стану і відображення розв’язку;

**3** – результат виконання окремого алгоритму;

**4** – вибір алгоритму (BFS, DFS, UCS, A\* Hemming, A\* Manhattan);

**5** – пошук вибраним алгоритмом і занесення результатів у **3**;

**6** – пошук усіма алгоритмами і занесення результатів у **9;**

**7** – створити початковий сан **2** випадковим чином;

**8** – розмір гри;

**9** – таблиця з результатами пошуків, містить такі дані:

1. Довжина шляху, який увійшов у розв’язок або -1, якщо розв’язок відсутній;
2. **Memory** – оцінка алгоритму за використаною пам’яттю – максимальна довжина черги Frontier за час виконання алгоритму;
3. **Explored** – оцінка алгоритму за кількістю відвіданих вершин – кількість вершин у контейнері Explored на момент завершення пошуку;
4. **Time** – час виконання пошуку у мілісекундах.
5. **Алгоритми**

Усі алгоритми пошуку використовують метод SearchRun у базовому класі BaseSearch:

template<class TState>

Solution<TState> BaseSearch<TState>::SearchRun()

{

const BaseNode<TState>\* node

= new BaseNode<TState>(NULL, new TState(mProblem->GetInitState()), 0);

mFrontier->*Add*(node, *F*(node));

while(!mFrontier->*IsEmpty*())

{

node = mFrontier->*Pop*();

mExplored->*Add*(node, NULL);

if(mProblem->IsGoalState(node->GetState()))

{

return GetSolution(node);

}

List<BaseAction<TState>\*> actions = mProblem->*GetActions*(node->GetState());

for(int i = 0; i < actions.Size(); ++i)

{

const BaseNode<TState>\* child = node->ChildNode(mProblem, actions[i]);

bool frontierContains = mFrontier->Contains(child->GetState());

if(!mExplored->Contains(child->GetState()) && !frontierContains)

{

mFrontier->*Add*(child, *F*(child));

if(mProblem->IsGoalState(child->GetState()))

{

return GetSolution(child);

}

}

else if(frontierContains)

{

mFrontier->*Update*(child, *F*(child));

}

}

}

return GetFailure();

}

Похідні класи BreadthFirstSearch, DepthFirstSearch, UniformCostSearch, AStarSearch перевизначають лише mFrontier та функцію:

int virtual *F*(const BaseNode<TState>\* node) = 0

* 1. **Breadth First Search (BFS)**

Визначає список Frontier як First In, First Out (FIFO) чергу, Значення функції F при вставленні в чергу не враховується.

* 1. **Deep First Search (DFS)**

Визначає список Frontier як Last In, First Out (LIFO) чергу, Значення функції F при вставленні в чергу не враховується.

* 1. **Uniform Cost Search (UCS)**

Визначає список Frontier як чергу з пріоритетом, значення функції F відіграє роль пріоритету вузла і набуває значення довжини шляху від початкового стану до поточного. F(n) = g(n)

* 1. **A\* (Hemming heuristic)**

Визначає список Frontier як чергу з пріоритетом, значення функції F відіграє роль пріоритету вузла і набуває значення довжини шляху від початкового стану до поточного плюс значення евристики. F(n) = g(n) + h(n). Евристика визначається як кількість комірок, які займають не своє місце.

* 1. **A\* (Manhattan heuristic)**

Аналогічно до пункту d. F(n) = g(n) + h(n). Евристика визначається як сума довжин найкоротших шляхів від кожної комірки до її позиції у цільовому стані.

1. **Результати експериментів**

Пошук відбувався на довільних початкових станах, які мають розв'язок. Гра розміром 3х3. Результати пошуку представлені у Табл. 1 та на Рис. 5.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Максимальна кількість вузлів у черзі Frontier | | | | | Загальна кількість елементів у Explored | | | | |
| № | BFS | DFS | UCS | A\*M | A\*H | BFS | DFS | UCS | A\*M | A\*H |
| 1 | 477 | 35593 | 545 | 33 | 47 | 718 | 57729 | 781 | 45 | 64 |
| 2 | 17289 | 42826 | 43926 | 909 | 4273 | 55070 | 111925 | 102480 | 1431 | 7440 |
| 3 | 8349 | 42389 | 8487 | 218 | 553 | 15507 | 86868 | 15457 | 332 | 896 |
| 4 | 15648 | 8650 | 28882 | 661 | 2341 | 44300 | 11304 | 64146 | 1058 | 4030 |
| 5 | 9353 | 25736 | 10442 | 178 | 864 | 17095 | 36945 | 17909 | 273 | 1392 |
| 6 | 24048 | 151 | 71269 | 3842 | 30128 | 174387 | 187 | 427244 | 6675 | 70249 |
| 7 | 13866 | 35442 | 18110 | 490 | 1695 | 31729 | 57784 | 37337 | 753 | 2825 |
| 8 | 3136 | 31023 | 3304 | 55 | 220 | 4991 | 47014 | 5101 | 74 | 356 |
| 9 | 23191 | 42913 | 53043 | 1340 | 6344 | 89965 | 124871 | 147335 | 2190 | 11175 |
| 10 | 21811 | 42913 | 48463 | 690 | 6985 | 83794 | 110994 | 135377 | 1056 | 12250 |
| 11 | 19287 | 42833 | 36137 | 554 | 3054 | 51931 | 93139 | 77151 | 863 | 5051 |
| 12 | 24983 | 42912 | 59802 | 1088 | 9385 | 114047 | 146510 | 199315 | 1782 | 16912 |
| AVG | 15119,83 | 32781,75 | 31867,5 | 838,1667 | 5490,75 | 56961,17 | 73772,5 | 102469,4 | 1377,667 | 11053,33 |

Табл. 1. Результати експериментів. Гра 3х3

Рис. 5. Порівняння алгоритмів за пам’яттю та відвіданими вузлами

Для станів які не мають розв’язку, очевидно, що у Explored попали усі можливі стани тому розмір його постійний і рівний 9!/2 = 181440.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | BFS | DFS | UCS | A\*M | A\*H |
| 1 | 1794 | 1800 | 6303 | 4181 | 4446 |
| 2 | 1856 | 1841 | 6568 | 4414 | 4633 |
| 3 | 1903 | 1872 | 6147 | 4383 | 4540 |
| 4 | 1934 | 1872 | 6458 | 4384 | 4508 |
| 5 | 1919 | 1872 | 6443 | 4446 | 4680 |
| 6 | 1935 | 1856 | 6131 | 4524 | 4477 |
| 7 | 1935 | 1857 | 6506 | 4352 | 4633 |
| 8 | 1934 | 1872 | 6240 | 4275 | 4446 |
| AVG | 1901,25 | 1855,25 | 6349,5 | 4369,875 | 4545,375 |

Табл. 2. Середній час пошуку з початковим станом, який не має розв'язку

1. **Висновок**

З результату експериментів можемо зробити висновок, що найефективнішими алгоритмами пошуку є алгоритми з інформованим пошуком A\*. Серед алгоритмів з інформованим пошуком ефективнішим є алгоритм, який використовував манхетенську віддаль в якості евристики, на другому місці – алгоритм, який використовував віддаль Хемінга. Серед алгоритмів не інформованого пошуку найефективніший виявився пошук вшир, який до того ж дає оптимальний розв’язок, далі – пошук вглиб, недоліком якого є дуже не оптимальний розв’язок, останній – пошук з урахуванням ціни пройденого шляху. У випадку, коли початковий стан завідома не мав роз'явзку алгоритми A\* виявились не ефективними. Це зумовлено використанням складнішої структури даних для зберігання станів, яка потребувала організації черги з пріоритетом та швидким доступом до елементів за їх значенням.