Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

**Лабораторна робота №5**

з дисципліни «Основи штучного інтелекту»

на тему:

*«***Інформативний пошук***»*

Варіант -13

Виконав:

студент групи ІС-72

Кривохижа Роман Андрійович

Київ - 2020

**Мета роботи**: розв’язати логічну задачу з використанням алгоритму інформативного пошуку.

**Завдання**: RBFS – Рекурсивний пошук за першим найкращим збігом

**Умова**: На ігровому полі 3 × 3 розставлені фішки з цифрами від 1 до 8. Є одна вільна клітина, на яку можуть бути перенесені фішки з сусідніх позицій. При цьому не допускається переміщення фішок через одну, а також по діагоналі. Завдання полягає в тому, щоб перетворити початкову конфігурацію в цільову конфігурацію

**Варіант:**

C:\Users\andrey\Downloads\Telegram Desktop\image_2020-05-07_00-40-04.png

**GitHub:** <https://github.com/kryvokhyzha/Courses/tree/master/AI>

**Лістинг програми**:

* task.pl

bestfirst( Start, Solution) :-

expand( [], l( Start, 0/0), 9999, \_, yes, Solution).

% Assume 9999 is greater than any f-value

% --------- 1

expand( P, l( N, \_), \_, \_, yes, [N|P]) :-

goal(N).

% --------- 2

expand( P, l(N,F/G), Bound, Tree1, Solved, Sol) :-

F =< Bound,

( bagof( M/C, (s(N,M,C), not(member(M,P))), Succ),

!, % Node N has successors

succlist( G, Succ, Ts), % Make subtrees Ts

bestf( Ts, F1), % f-value of best successor

expand( P, t(N,F1/G,Ts), Bound, Tree1, Solved, Sol)

;

Solved = never % N has no successors - dead end

) .

% --------- 3

expand( P, t(N,F/G,[T|Ts]), Bound, Tree1, Solved, Sol) :-

F =< Bound,

bestf( Ts, BF), min( Bound, BF, Bound1), % Bound1 = min(Bound,BF)

expand( [N|P], T, Bound1, T1, Solved1, Sol),

continue( P, t(N,F/G,[T1|Ts]), Bound, Tree1, Solved1, Solved, Sol).

% --------- 4

expand( \_, t(\_,\_,[]), \_, \_, never, \_) :- !.

% --------- 5

expand( \_, Tree, Bound, Tree, no, \_) :-

f( Tree, F), F > Bound.

% continue( Path, Tree, Bound, NewTree, SubtreeSolved, TreeSolved, Solution)

continue( \_, \_, \_, \_, yes, yes, Sol).

continue( P, t(N,F/G,[T1|Ts]), Bound, Tree1, no, Solved, Sol) :-

insert( T1, Ts, NTs),

bestf( NTs, F1),

expand( P, t(N,F1/G,NTs), Bound, Tree1, Solved, Sol).

continue( P, t(N,F/G,[\_|Ts]), Bound, Tree1, never, Solved, Sol) :-

bestf( Ts, F1),

expand( P, t(N,F1/G,Ts), Bound, Tree1, Solved, Sol).

% --------- 6

% succlist( G0, [ Node1/Cost1, ...], [ l(BestNode,BestF/G), ...]):

succlist( \_, [], []).

succlist( G0, [N/C | NCs], Ts) :-

G is G0 + C,

h( N, H), % Heuristic term h(N)

F is G + H,

succlist( G0, NCs, Ts1),

insert( l(N,F/G), Ts1, Ts).

% --------- 7

insert( T, Ts, [T | Ts]) :-

f( T, F), bestf( Ts, F1),

F =< F1, !.

insert( T, [T1 | Ts], [T1 | Ts1]) :-

insert( T, Ts, Ts1).

% Extract f-value

f( l(\_,F/\_), F). % f-value of a leaf

f( t(\_,F/\_,\_), F). % f-value of a tree

bestf( [T|\_], F) :- % Best f-value of a list of trees

f( T, F).

bestf( [], 9999). % No trees: bad f-value

min( X, Y, X) :-

X =< Y, !.

min( X, Y, Y).

s([Empty | Tiles], [Tile | Tiles1], 1):- % Стоимости всех дуг равны 1

swap(Empty, Tile, Tiles, Tiles1). % Поменять местами пустую фишку

% Empty и фишку Tile в списке Tiles

swap(Empty, Tile, [Tile | Ts], [Empty | Ts] ):-

mandist( Empty, Tile, 1). % Манхэттенское расстояние равно 1

swap( Empty, Tile, [T1 | Ts], [T1 | Ts1] ) :-

swap( Empty, Tile, Ts, Ts1).

mandist( X/Y, X1/Y1, D) :- % D - это манхэттенское расстояние между двумя клетками

dif( X, X1, Dx),

dif( Y, Y1, Dy),

D is Dx + Dy.

dif( A, B, D) :- % D is |A-B|

D is A-B, D >= 0, !

;

D is B-A.

h( [Empty | Tiles], H) :-

goal( [Empty1 | GoalSquares] ),

totdist( Tiles, GoalSquares, D), % Суммарное расстояние от исходных клеток

seq( Tiles, S), % Оценка упорядоченности

H is D + 3\*S.

totdist( [], [], 0).

totdist( [Tile | Tiles], [Square | Squares], D) :-

mandist( Tile, Square, D1),

totdist( Tiles, Squares, D2),

D is D1 + D2.

% seq( TilePositions, Score): оценка упорядоченности

seq( [First | OtherTiles], S) :-

seq( [First | OtherTiles ], First, S).

seq( [Tile1, Tile2 | Tiles], First, S) :-

score( Tile1, Tile2, S1),

seq( [Tile2 | Tiles], First, S2),

S is S1 + S2.

seq( [Last], First, S) :-

score( Last, First, S).

score( 2/2, \_, 1) :- !. % ценка фишки, стоящей в центре, равна 1

score( 1/3, 2/3, 0) :- !. % Оценка фишки, за которой следует допустимый преемник, равна 0

score( 2/3, 3/3, 0) :- !.

score( 3/3, 3/2, 0) :- !.

score( 3/2, 3/1, 0) :- !.

score( 3/1, 2/1, 0) :- !.

score( 2/1, 1/1, 0) :- !.

score( 1/1, 1/2, 0) :- !.

score( 1/2, 1/3, 0) :- !.

score( \_, \_, 2). % Оценка фишкк, за которой следует недопустимый преемник, равна 2

goal( [2/2,1/3,2/3,3/3,3/2,3/1,2/1,1/1,1/2] ). % Goal squares for tiles

**Виконання програми**:

****

**Контрольні питання:**



unction A\*(start, goal, f)

% множество уже пройденных вершин

var closed := the empty set

% множество частных решений

var open := make\_queue(f)

enqueue(open, path(start))

while open is not empty

var p := remove\_first(open)

var x := the last node of p

if x in closed

continue

if x = goal

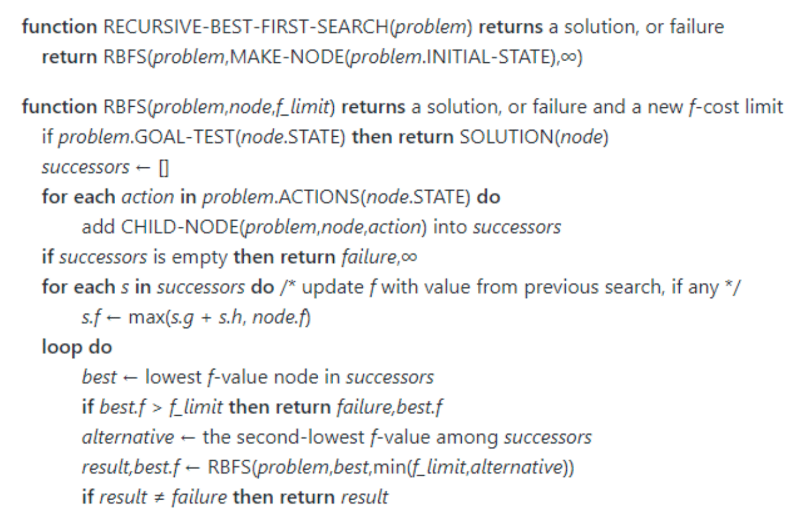
return p

add(closed, x)

% добавляем смежные вершины

foreach y in successors(x)

enqueue(open, add\_to\_path(p, y))



1. Інформативний пошук зазвичай забезпечує більш ефективний пошук у порівнянні з неінформативним пошуком, оскільки використовує додаткові знання про конкретну задачу, що формулюються у вигляді евристичної функції, яка на кожному кроці оцінює альтернативи на основі додаткової інформації, щоб прийняти рішення про подальший напрямок алгоритму.
2. Евристична функція потрібна, щоб спробувати уникнути повного перебору всіх можливих станів.
3. Функція завжди повинна повертати точне значення довжини найкоротшого рішення
4. Алгоритми A\* та RBFS є повними та оптимальними, якщо їх евристичні функції є допустимими, тобто такими, що ніколи не перевищують фактичну мінімальну вартість досягнення мети. Якщо алгоритм пошуку має ідеальну евристичну функцію, тобто таку, що завжди повертає точне значення довжини найкоротшого рішення, то він завжди буде оптимальним.
5. RBFS має структуру, аналогічну структурі рекурсивного пошуку в глибину, але замість нескінченного проходження вниз по поточному шляху даний алгоритм контролює f-значення найкращого альтернативного шляху, доступного з будь-якого предка поточного вузла. Якщо поточний вузол перевищує задану межу, то поточний етап рекурсії скасовується і рекурсія продовжується з альтернативного шляху. Після скасування даного етапу рекурсії в алгоритмі RBFS відбувається заміна f-значення кожного вузла вздовж даного шляху найкращим f-значенням його дочірнього вузла. Завдяки цьому в алгоритмі RBFS запам'ятовується f-значення найкращого листового вузла з забутого піддерева і тому в деякий наступний момент часу може бути прийнято рішення про те, чи варто знову розгортати це піддерево.

**Висновок:** для вирішення даної задачі мною було реалізовано алгоритм RBFS та евристичниа функця на основі манхеттенської відстані. Також, до евристичної функцій додається штраф впорядкованість («оцінка впорядкованості»). Для даної умови, знайдений шлях має глибину 6, відповідний скріншот наведено вище. Також, мною було протестовано ще декілька варіантів початкових станів.