MUC LUC

Contents

[A. PROLOG 2](#_Toc120606584)

[B. PDDL 5](#_Toc120606585)

[I.ROBOT 5](#_Toc120606586)

[1.domain 5](#_Toc120606587)

[2. problem 6](#_Toc120606588)

[2.1 Problem 1 6](#_Toc120606589)

[2.2 Problem 2 6](#_Toc120606590)

[2.3 problem 3 6](#_Toc120606591)

[2.4 problem 4 7](#_Toc120606592)

[II. AIRCARGO 7](#_Toc120606593)

[1. domain 7](#_Toc120606594)

[2. problem 8](#_Toc120606595)

[2.1 problem 1 8](#_Toc120606596)

[III. Thap Ha Noi 9](#_Toc120606597)

[1. domain 9](#_Toc120606598)

[2. problem 9](#_Toc120606599)

[2.1 problem 1 9](#_Toc120606600)

[2.2 problem 2 9](#_Toc120606601)

[III. 8 puzzle 10](#_Toc120606602)

[1. domain 10](#_Toc120606603)

[2. problem 11](#_Toc120606604)

[2.1 problem1 11](#_Toc120606605)

[2.2 problem2 12](#_Toc120606606)

[2.3 problem3 12](#_Toc120606607)

[C. SOURCE C 13](#_Toc120606608)

[I. DONG NUOC 13](#_Toc120606609)

[1. struct state 13](#_Toc120606610)

[2. struct Node 13](#_Toc120606611)

[3. struct Stack 13](#_Toc120606612)

[4. push Stack 13](#_Toc120606613)

[5. makenull Stack 14](#_Toc120606614)

[6. empty Stack 14](#_Toc120606615)

[7. top Stack 14](#_Toc120606616)

[8. pop Stack 14](#_Toc120606617)

[9. full Stack 14](#_Toc120606618)

[10. compare State 14](#_Toc120606619)

[11. find State 14](#_Toc120606620)

[12. goal check 14](#_Toc120606621)

[13. cac action 15](#_Toc120606622)

[13.1 pourWaterFullX 15](#_Toc120606623)

[13.2 pourWaterFullY 15](#_Toc120606624)

[13.3 pourWaterEmptyX 15](#_Toc120606625)

[13.4 pourWaterEmptyY 15](#_Toc120606626)

[13.5 pourWaterXY 15](#_Toc120606627)

[13.6 pourWaterYX 16](#_Toc120606628)

[14. call\_operator 16](#_Toc120606629)

[15. DFS\_Algorithm 16](#_Toc120606630)

[16. print\_WaysToGetGoal 17](#_Toc120606631)

[17. struct Queue 17](#_Toc120606632)

[18. BFS\_Algorithm 18](#_Toc120606633)

[II. DONG SUA 19](#_Toc120606634)

[1. cac action 19](#_Toc120606635)

[2. call\_operator 21](#_Toc120606636)

[3. compare state 21](#_Toc120606637)

[4. find state Stack 21](#_Toc120606638)

[5. find state Queue 22](#_Toc120606639)

[6. BFS 22](#_Toc120606640)

[7. print goal 23](#_Toc120606641)

[III. Tu Si Va Ke An Thit 23](#_Toc120606642)

[1. BFS 23](#_Toc120606643)

[IV. UP LY 27](#_Toc120606644)

[V. 8 puzzle Hueristic 29](#_Toc120606645)

[VI. 8 puzzle A start 35](#_Toc120606646)

[VII. sudoku thoa man rang buoc 42](#_Toc120606647)

[IX. rang buoc 8 quan hau 49](#_Toc120606648)

# A. PROLOG

%luy thừa

luythua(X, 1, X) :- !.

luythua(X, Y, Z) :- Y1 is Y-1, luythua(X, Y1, Z1), Z is Z1\*X.

%giai thua

giaithua(0, 1) :- !.

giaithua(X, Y) :- X1 is X-1, giaithua(X1, Y1), Y is X\*Y1.

%to hop chap k cua n

tohop(K, N, Y) :- giaithua(N, Q1), giaithua(K, Q2), giaithua(N-K, Q3), Y is Q1 / (Q2\*Q3).

%thap ha noi

move(1,X,Y,\_) :-

write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N,X,Y,Z) :-

N>1,

M is N-1,

move(M,X,Z,Y),

move(1,X,Y,\_),

move(M,Z,Y,X).

%fibonaci

fib(0, 0) :- !.

fib(1, 1) :- !.

fib(X, Y) :- N1 is X-1, N2 is X-2, fib(N1, Y1), fib(N2, Y2), Y is Y1+Y2.

%4.1 and 4.2

noids([], L2, L2).

noids([X | L1], L2, [X | L3]) :- noids(L1, L2, L3).

addptu(L, X, NewL) :- noids(L, [X], NewL).

%4.3 query ptle([1,2,3,4,5,6,7,8,9,13,15,20], L).

is\_odd(I) :- 0 =\= I mod 2.

ptle(L, Newlist) :- include(is\_odd, L, Newlist).

%4.5 tongptle([1,2,3,4,5,6,7,8,9,13,15,20]).

tongptle([], 0):- !.

tongptle([H|D], T):- 0 =\= H mod 2, tongptle(D, T1), T is H+T1, !.

tongptle([\_|D], T):- tongptle(D, T1), T is T1.

% Bai thap ha noi

move(1, A, B, \_) :- print(A), print(' --> '), print(B), nl.

move(N, A, B, C) :- N1 is N-1, move(N1, A, C, B),move(1, A, B, C), move(N1, C, B, A).

% move(3, 'A', 'B', 'C'). ---------------------------------------------------------

% Bia tinh giai thua

giai\_thua(0, 1).

giai\_thua(N, R) :- N > 0, N1 is N-1, giai\_thua(N1, R1), R is R1\*N.

% giai\_thua(5, R).-----------------------------------------------------------------

% Bai fibonaci thu N

fibo(1, 1).

fibo(2, 1).

fibo(N, R) :- N > 2, N1 is N-1, N2 is N-2, fibo(N1, R1), fibo(N2, R2), R is R1 + R2.

% fibo(5, R).----------------------------------------------------------------------

% Kiem tra P co phai la so nguyen to hay k

divide(N, P) :- P mod N =:= 0.

divide(N, P) :- P mod N \= 0, N\*N < P, N1 is N+1, divide(N1, P).

prime(P) :- \+divide(2, P).

% prime(7).-----------------------------------------------------------------------

% dem so phan tu cua mot danh sach

count([], 0).

count([\_|T], N) :- count(T, N1), N is N1 + 1.

% count([1, 3, 5, 2, 6, 3], N).----------------------------------------------------

% viet chuong trinh kiem tra phan tu co trong danh sach hay khong

member(X, [X|\_]).

member(X, [Y|L]) :- X \= Y, member(X, L).

% member(2, [3, 4, 2, 5, 7]).------------------------------------------------------

% Viết chương trình thêm một phần tử X vào cuối danh sách L, kết quả lưu trong R.

push\_back(X, [], [X]).

push\_back(X, [H|T], [H|R1]) :- push\_back(X, T, R1).

% push\_back(8, [5, 6, 7], R).------------------------------------------------------

% kiem tra do thi co lien thong tu 2 dinh

edge(1,2).

edge(1,5).

edge(2,3).

edge(2,5).

edge(3,4).

edge(4,5).

edge(4,6).

edge(7,8).

canh(X, Y) :- edge(X, Y) ; edge(Y, X).

member1(X, [X|\_]).

member1(X, [Y|T]) :- X \= Y, member1(X, T).

co\_duong\_di(Y, Y, \_).

co\_duong\_di(X, Y, P) :- canh(X, Z), \+member1(Z, P),

co\_duong\_di(Z, Y, [Z|P]).

lien\_thong(X, Y) :- canh(X, Y).

lien\_thong(X, Y) :- co\_duong\_di(X, Y, [X]).

% lien\_thong(1, 4).-----------------------------------------------------------------

Loi

% 8L -> 5L

pour([A1, B1, C1], [A2, B2, C2]) :- A1>0, B1<5, A2=0, B2 is A1+B1, C2=C1, B2 < 5.

pour([A1, B1, C1], [A2, B2, C2]) :- A1>0, B1<5, A2 is A1+B1-5, B2=5, C2=C1.

% 8L -> 3L

pour([A1, B1, C1], [A2, B2, C2]) :- A1 > 0, C1 < 3, A2 = 0, C2 is A1 + C1, B2 = B1, C2 < 3.

pour([A1, B1, C1], [A2, B2, C2]) :- A1 > 0, C1 < 3, A2 is A1 + C1 - 3, C2 = 3, B2 = B1.

% 5L -> 8L

pour([A1, B1, C1], [A2, B2, C2]) :- B1 > 0, A1 < 8, B2 = 0, A2 is A1 + B1, C2 = C1, A2 < 8.

pour([A1, B1, C1], [A2, B2, C2]) :- B1 > 0, A1 < 8, B2 =0 , A2 = 8, C2 = C1.

% 5L -> 3L

pour([A1, B1, C1], [A2, B2, C2]) :- B1 > 0, C1 < 3, B2 = 0, C2 is B1 + C1, A2 = A1, C2 < 3.

pour([A1, B1, C1], [A2, B2, C2]) :- B1 > 0, C1 < 3, B2 is B1 + C1 - 3, C2 = 3, A2 = A1.

% 3L -> 8L

pour([A1, B1, C1], [A2, B2, C2]) :- C1 > 0, A1 < 8, C2 = 0, A2 is A1 + C1, B2 = B1, A2 < 8.

pour([A1, B1, C1], [A2, B2, C2]) :- C1 > 0, A1 < 8, C2=0 , A2 = 8, B2 = B1.

% 3L -> 5L

pour([A1, B1, C1], [A2, B2, C2]) :- C1 > 0, B1 < 5, C2 = 0, B2 is B1 + C1, A2 = A1, B2 < 5.

pour([A1, B1, C1], [A2, B2, C2]) :- C1 > 0, B1 < 5, C2 is C1 + B1 - 5, B2 = 5, A2 = A1.

%is B1 + A1 - 8 is C1 + A1 - 8

# B. PDDL

## I.ROBOT

### 1.domain

(define (domain gripper)

(:predicates

(room ?room)

(ball ?ball)

(gripper ?gripper)

(at ?ball ?room)

(at-robby ?room)

(free ?gripper)

(carry ?ball ?gripper)

)

(:action move

:parameters (?rooma ?roomb)

:precondition (and

(room ?rooma)(room ?roomb)(at-robby ?rooma)

)

:effect (and

(at-robby ?roomb)(not (at-robby ?rooma))))

(:action pick

:parameters(?ball ?room ?gripper)

:precondition(and

(ball ?ball)(room ?room)(gripper ?gripper)

(at ?ball ?room)(free ?gripper)(at-robby ?room)

)

:effect(and

(carry ?ball ?gripper)(not(at ?ball ?room))

(not(free ?gripper))

)

)

(:action drop

:parameters(?ball ?room ?gripper)

:precondition(and

(ball ?ball)(room ?room)(gripper ?gripper)

(carry ?ball ?gripper)

(at-robby ?room)

)

:effect(and

(at ?ball ?room)

(free ?gripper)

(not (carry ?ball ?gripper))

)

)

)

### 2. problem

#### 2.1 Problem 1

(define (problem gripperpickball)

(:domain gripper)

(:objects rooma roomb ball1 ball2 left right)

(:init

(room rooma) (room roomb) (ball ball1)(ball ball2)

(gripper left)(gripper right)

(at-robby rooma)

(free left)(free right)

(at ball1 rooma)(at ball2 rooma)

)

(:goal (and

(at ball1 roomb)

))

)

#### 2.2 Problem 2

(define (problem gripperpickball)

(:domain gripper)

(:objects rooma roomb ball1 ball2 ball3 ball4 ball5 ball6 left right)

(:init

(room rooma) (room roomb)

(ball ball1)(ball ball2)

(ball ball3)(ball ball4)

(ball ball5)(ball ball6)

(gripper left)(gripper right)

(at-robby rooma)

(free left)(free right)

(at ball1 rooma)(at ball2 rooma)

(at ball3 rooma)(at ball4 rooma)

(at ball5 rooma)(at ball6 rooma)

)

#### 2.3 problem 3

(define (problem gripperpickball)

(:domain gripper)

(:objects rooma roomb ball1 ball2 ball3 ball4 ball5 ball6 left right)

(:init

(room rooma) (room roomb)

(ball ball1)(ball ball2)

(ball ball3)(ball ball4)

(ball ball5)(ball ball6)

(gripper left)(gripper right)

(at-robby rooma)

(free left)(free right)

(at ball1 rooma)(at ball2 rooma)

(at ball3 rooma)(at ball4 rooma)

(at ball5 rooma)(at ball6 rooma)

)

(:goal (and

(at ball1 roomb)(at ball2 roomb)

(at ball3 roomb)(at ball4 roomb)

(at ball5 roomb)

))

)

(:goal (and

(at ball1 roomb)(at ball2 roomb)

(at ball3 roomb)

)))

#### 2.4 problem 4

(define (problem gripperpickball)

(:domain gripper)

(:objects rooma roomb ball1 ball2 ball3 ball4

ball5 ball6 ball7 ball8 left right)

(:init

(room rooma) (room roomb)

(ball ball1)(ball ball2)

(ball ball3)(ball ball4)

(ball ball5)(ball ball6)

(ball ball7)(ball ball8)

(gripper left)(gripper right)

(at-robby rooma)

(free left)(free right)

(at ball1 rooma)(at ball2 rooma)

(at ball3 rooma)(at ball4 rooma)

(at ball5 rooma)(at ball6 rooma)

(at ball7 rooma)(at ball8 rooma)

)

(:goal (and

(at ball1 roomb)(at ball2 roomb)

(at ball3 roomb)(at ball4 roomb)

(at ball5 roomb)(at ball6 roomb)

(at ball7 roomb)

)))

## II. AIRCARGO

### 1. domain

(define (domain air)

(:predicates

(cargo ?c)

(plane ?p)

(airport ?a)

(at ?obj ?place)

(in ?obj ?place)

)

(:action LOAD

:parameters (?c ?p ?a)

:precondition (and

(cargo ?c)(plane ?p)(airport ?a)

(at ?c ?a)(at ?p ?a)

)

:effect (and

(in ?c ?p)(not (at ?c ?a))

)

)

(:action UNLOAD

:parameters (?c ?p ?a)

:precondition (and

(cargo ?c)(plane ?p)(airport ?a)

(in ?c ?p)(at ?p ?a)

)

:effect(and

(at ?c ?a)(not (in ?c ?p))

)

)

(:action FLY

:parameters (?p ?from ?to)

:precondition (and

(plane ?p)(airport ?from)(airport ?to)

(at ?p ?from)

)

:effect(and

(at ?p ?to)(not (at ?p ?from))

)

)

)

### 2. problem

#### 2.1 problem 1

(define (problem abc)

(:domain air)

(:objects c1 c2 p1 p2 SFO JFK)

(:init

(cargo c1)(cargo c2)(plane p1)(plane p2)

(airport SFO)(airport JFK)

(at c1 SFO)(at c2 JFK)(at p1 SFO)(at p2 JFK)

)

(:goal

(and

(at c1 JFK)(at c2 SFO)

)

)

)

## III. Thap Ha Noi

### 1. domain

(define (domain air)

(:predicates

(disc ?d)

(larger ?a ?b)

(on ?a ?b)

(clear ?a)

)

(:action move

:parameters (?d ?from ?to)

:precondition (and

(disc ?d)(disc ?from)(disc ?to)

(clear ?d)(on ?d ?from)(clear ?to)(larger ?to ?d)

)

:effect(and

(clear ?from)(on ?d ?to)(not (clear ?to))(not (on ?d ?from))

)

)

)

### 2. problem

#### 2.1 problem 1

(define (problem abc)

(:domain air)

(:objects d1 d2 d3 d4 p1 p2 p3)

(:init

(disc d1)(disc d2)(disc d3)(disc d4)

(disc p1)(disc p2)(disc p3)

(larger p1 d1)(larger p1 d2)(larger p1 d3)(larger p1 d4)

(larger p2 d1)(larger p2 d2)(larger p2 d3)(larger p2 d4)

(larger p3 d1)(larger p3 d2)(larger p3 d3)(larger p3 d4)

(larger d4 d3)(larger d4 d2)(larger d4 d1)

(larger d3 d2)(larger d3 d1)

(larger d2 d1)

(on d4 p1)(on d3 d4)(on d2 d3)(on d1 d2)

(clear p2)(clear p3)(clear d1)

)

(:goal (and

(clear p1)(clear p2)

(on d4 p3)(on d3 d4)(on d2 d3)(on d1 d2)(clear d1)

)

)

)

#### 2.2 problem 2

(define (problem abc)

(:domain air)

(:objects d1 d2 d3 p1 p2 p3)

(:init

(disc d1)(disc d2)(disc d3)

(disc p1)(disc p2)(disc p3)

(larger p1 d1)(larger p1 d2)(larger p1 d3)

(larger p2 d1)(larger p2 d2)(larger p2 d3)

(larger p3 d1)(larger p3 d2)(larger p3 d3)

(larger d3 d2)(larger d2 d1)

(on d3 p1)(on d2 d3)(on d1 d2)

(clear p2)(clear p3)(clear d1)

)

(:goal (and

(clear p1)(clear p2)(on d3 p3)(on d2 d3)(on d1 d2)(clear d1)

)

)

)

## III. 8 puzzle

### 1. domain

(define (domain air)

(:predicates

(tile ?d)

(at ?t ?a ?b)

(blank ?a ?b)

(position ?a)

(sub ?a ?b)

(add ?a ?b)

)

(:action MOVE\_UP

:parameters (?bx ?by ?t ?tx)

:precondition (and

(position ?bx)(position ?by)(tile ?t)(position ?tx)

(blank ?bx ?by)(at ?t ?tx ?by)(sub ?bx ?tx)

)

:effect(and

(blank ?tx ?by)(not (blank ?bx ?by))

(at ?t ?bx ?by)(not (at ?t ?tx ?by))

)

)

(:action MOVE\_DOWN

:parameters (?bx ?by ?t ?tx)

:precondition (and

(position ?bx)(position ?by)(tile ?t)(position ?tx)

(blank ?bx ?by)(at ?t ?tx ?by)(add ?bx ?tx)

)

:effect(and

(blank ?tx ?by)(not (blank ?bx ?by))

(at ?t ?bx ?by)(not (at ?t ?tx ?by))

)

)

(:action MOVE\_LEFT

:parameters (?bx ?by ?t ?ty)

:precondition (and

(position ?bx)(position ?by)(tile ?t)(position ?ty)

(blank ?bx ?by)(at ?t ?bx ?ty)(sub ?by ?ty)

)

:effect(and

(blank ?bx ?ty)(not (blank ?bx ?by))

(at ?t ?bx ?by)(not (at ?t ?bx ?ty))

)

)

(:action MOVE\_RIGHT

:parameters (?bx ?by ?t ?ty)

:precondition (and

(position ?bx)(position ?by)(tile ?t)(position ?ty)

(blank ?bx ?by)(at ?t ?bx ?ty)(add ?by ?ty)

)

:effect(and

(blank ?bx ?ty)(not (blank ?bx ?by))

(at ?t ?bx ?by)(not (at ?t ?bx ?ty))

)

)

)

### 2. problem

#### 2.1 problem1

(define (problem abc)

(:domain air)

(:objects t1 t2 t3 t4 t5 t6 t7 t8 x1 x2 x3 y1 y2 y3)

(:init

(tile t1)(tile t2)(tile t3)(tile t4)

(tile t5)(tile t6)(tile t7)(tile t8)

(position x1)(position x2)(position x3)

(position y1)(position y2)(position y3)

(add x1 x2)(add x2 x3)

(sub x3 x2)(sub x2 x1)

(add y1 y2)(add y2 y3)

(sub y3 y2)(sub y2 y1)

(at t8 x1 y1)(blank x1 y2)(at t2 x1 y3)

(at t7 x2 y1)(at t4 x2 y2)(at t5 x2 y3)

(at t6 x3 y1)(at t1 x3 y2)(at t3 x3 y3)

)

(:goal (and

;(blank x1 y1)(at t3 x1 y2)(at t6 x1 y3)

;(at t1 x2 y1)(at t4 x2 y2)(at t7 x2 y3)

;(at t2 x3 y1)(at t5 x3 y2)(at t8 x3 y3)

;(at t8 x1 y1)(at t4 x1 y2)(at t2 x1 y3)

;(at t1 x2 y1)(at t5 x2 y2)(at t3 x2 y3)

;(at t7 x3 y1)(at t6 x3 y2)(blank x3 y3)

(at t1 x1 y1)(at t5 x1 y2)(at t3 x1 y3)

(at t4 x2 y1)(at t8 x2 y2)(at t6 x2 y3)

(at t2 x3 y1)(blank x3 y2)(at t7 x3 y3)

))

)

;802

;745

;613

#### 2.2 problem2

(define (problem abc)

(:domain air)

(:objects t1 t2 t3 t4 t5 t6 t7 t8 x1 x2 x3 y1 y2 y3)

(:init

(tile t1)(tile t2)(tile t3)(tile t4)

(tile t5)(tile t6)(tile t7)(tile t8)

(position x1)(position x2)(position x3)

(position y1)(position y2)(position y3)

(add x1 x2)(add x2 x3)

(sub x3 x2)(sub x2 x1)

(add y1 y2)(add y2 y3)

(sub y3 y2)(sub y2 y1)

(at t8 x1 y1)(blank x1 y2)(at t2 x1 y3)

(at t7 x2 y1)(at t4 x2 y2)(at t5 x2 y3)

(at t6 x3 y1)(at t1 x3 y2)(at t3 x3 y3)

)

(:goal (and

;(blank x1 y1)(at t3 x1 y2)(at t6 x1 y3)

;(at t1 x2 y1)(at t4 x2 y2)(at t7 x2 y3)

;(at t2 x3 y1)(at t5 x3 y2)(at t8 x3 y3)

(at t8 x1 y1)(at t4 x1 y2)(at t2 x1 y3)

(at t1 x2 y1)(at t5 x2 y2)(at t3 x2 y3)

(at t7 x3 y1)(at t6 x3 y2)(blank x3 y3)

;(at t1 x1 y1)(at t5 x1 y2)(at t3 x1 y3)

;(at t4 x2 y1)(at t8 x2 y2)(at t6 x2 y3)

;(at t2 x3 y1)(blank x3 y2)(at t7 x3 y3)

))

)

;802

;745

;613

#### 2.3 problem3

(define (problem abc)

(:domain air)

(:objects t1 t2 t3 t4 t5 t6 t7 t8 x1 x2 x3 y1 y2 y3)

(:init

(tile t1)(tile t2)(tile t3)(tile t4)

(tile t5)(tile t6)(tile t7)(tile t8)

(position x1)(position x2)(position x3)

(position y1)(position y2)(position y3)

(add x1 x2)(add x2 x3)

(sub x3 x2)(sub x2 x1)

(add y1 y2)(add y2 y3)

(sub y3 y2)(sub y2 y1)

(at t8 x1 y1)(blank x1 y2)(at t2 x1 y3)

(at t7 x2 y1)(at t4 x2 y2)(at t5 x2 y3)

(at t6 x3 y1)(at t1 x3 y2)(at t3 x3 y3)

)

(:goal (and

(blank x1 y1)(at t3 x1 y2)(at t6 x1 y3)

(at t1 x2 y1)(at t4 x2 y2)(at t7 x2 y3)

(at t2 x3 y1)(at t5 x3 y2)(at t8 x3 y3)

;(at t8 x1 y1)(at t4 x1 y2)(at t2 x1 y3)

;(at t1 x2 y1)(at t5 x2 y2)(at t3 x2 y3)

;(at t7 x3 y1)(at t6 x3 y2)(blank x3 y3)

;(at t1 x1 y1)(at t5 x1 y2)(at t3 x1 y3)

;(at t4 x2 y1)(at t8 x2 y2)(at t6 x2 y3)

;(at t2 x3 y1)(blank x3 y2)(at t7 x3 y3)

))

)

;802

;745

;613

# C. SOURCE C

## I. DONG NUOC

#define tankcapacity\_X 9

#define tankcapacity\_Y 4

#define empty 0

#define goal 6

#define Maxlength 100

### 1. struct state

typedef struct {

int x;

int y;

}State;

### 2. struct Node

typedef struct {

State state;

struct Node\* Parent;

int no\_function;

}Node;

const char\* action[] = {"First State", "Pour Water Full X", "Pour Water Full Y", "Pour Water Empty X", "Pour Water Empty Y", "Pour Water X to Y", "Pour Water Y to X"};

### 3. struct Stack

typedef struct {

Node\* Element[Maxlength];

int Top\_idx;

}Stack;

### 4. push Stack

void push(Node\* x, Stack \*stack){

if(full\_Stack(\*stack)){

printf("Error! Stack is full");

} else {

stack->Top\_idx--;

stack->Element[stack->Top\_idx] = x;

}

}

### 5. makenull Stack

void makeNull\_Stack(Stack \*stack){

stack->Top\_idx = Maxlength;

}

### 6. empty Stack

int empty\_Stack(Stack stack){

return stack.Top\_idx == Maxlength;

}

### 7. top Stack

Node\* top(Stack stack){

if(!empty\_Stack(stack)){

return stack.Element[stack.Top\_idx];

}

return NULL;

}

### 8. pop Stack

void pop(Stack \*stack){

if(!empty\_Stack(\*stack)){

stack->Top\_idx++;

} else {

printf("Error! Stack is empty");

}

}

### 9. full Stack

int full\_Stack(Stack stack){

return stack.Top\_idx==0;

}

### 10. compare State

int CompareState(State state1, State state2){

if(state1.x == state2.x && state1.y == state2.y){

return 1;

}

return 0;

}

### 11. find State

int find\_State(State state, Stack openStack){

while(!empty\_Stack(openStack)){

if(CompareState(top(openStack)->state, state)){

return 1;

}

pop(&openStack);

}

return 0;

}

### 12. goal check

int goalCheck(State state){

return (state.x == goal || state.y == goal);

}

### 13. cac action

#### 13.1 pourWaterFullX

int pourWaterFullX(State cur\_state, State \*result){

if(cur\_state.x < tankcapacity\_X){

result->x = tankcapacity\_X;

result->y = cur\_state.y;

return 1;

}

return 0;

}

#### 13.2 pourWaterFullY

int pourWaterFullY(State cur\_state, State \*result){

if(cur\_state.y < tankcapacity\_Y){

result->y = tankcapacity\_Y;

result->x = cur\_state.x;

return 1;

}

return 0;

}

#### 13.3 pourWaterEmptyX

int pourWaterEmptyX(State cur\_state, State \*result){

if(cur\_state.x > 0){

result->x = empty;

result->y = cur\_state.y;

return 1;

}

return 0;

}

#### 13.4 pourWaterEmptyY

int pourWaterEmptyY(State cur\_state, State \*result){

if(cur\_state.y > 0){

result->y = empty;

result->x = cur\_state.x;

return 1;

}

return 0;

}

#### 13.5 pourWaterXY

int pourWaterXY(State cur\_state, State \*result){

if(cur\_state.x > 0 && cur\_state.y < tankcapacity\_Y){

result->x = cur\_state.x - (tankcapacity\_Y - cur\_state.y);

if(result->x < 0){

result->x = empty;

}

result->y = (cur\_state.y + cur\_state.x);

if(result->y > tankcapacity\_Y){

result->y = tankcapacity\_Y;

}

return 1;

}

return 0;

}

#### 13.6 pourWaterYX

int pourWaterYX(State cur\_state, State \*result){

if(cur\_state.x < tankcapacity\_X && cur\_state.y > 0){

result->x = cur\_state.x + cur\_state.y;

if(result->x > tankcapacity\_X){

result->x = tankcapacity\_X;

}

result->y = cur\_state.y - (tankcapacity\_X - cur\_state.x);

if(result->y < 0){

result->y = empty;

}

return 1;

}

return 0;

}

### 14. call\_operator

int call\_operator(State cur\_state, State \*result, int option){

switch(option){

case 1: return pourWaterFullX(cur\_state, result);

case 2: return pourWaterFullY(cur\_state, result);

case 3: return pourWaterEmptyX(cur\_state, result);

case 4: return pourWaterEmptyY(cur\_state, result);

case 5: return pourWaterXY(cur\_state, result);

case 6: return pourWaterYX(cur\_state, result);

default :

printf("Error calls operators");

return 0;

}

}

### 15. DFS\_Algorithm

Node\* DFS\_Algorithm(State state){

Stack open\_DFS;

Stack close\_DFS;

makeNull\_Stack(&open\_DFS);

makeNull\_Stack(&close\_DFS);

Node\* root = (Node\*)malloc(sizeof(Node));

root->state = state;

root->Parent = NULL;

root->no\_function = 0;

push(root, &open\_DFS);

while(!empty\_Stack(open\_DFS)){

Node\* node = top(open\_DFS);

pop(&open\_DFS);

push(node, &close\_DFS);

if(goalcheck(node->state)){

return node;

}

int opt;

for(opt = 1; opt <=6; opt++){

State newState;

makeNullState(&newState);

if(call\_operator(node->state, &newState, opt)){

if(find\_State(newState, open\_DFS) || find\_State(newState, close\_DFS)){

continue;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->state = newState;

newNode->Parent = node;

newNode->no\_function = opt;

push(newNode, &open\_DFS);

}

}

}

return NULL;

}

### 16. print\_WaysToGetGoal

void print\_WaysToGetGoal(Node\* node){

Stack stackPrint;

makeNull\_Stack(&stackPrint);

while(node->Parent != NULL){

push(node, &stackPrint);

node = node->Parent;

}

push(node, &stackPrint);

int no\_action = 0;

while(!empty\_Stack(stackPrint)){

printf("\n Action %d: %s", no\_action, action[top(stackPrint)->no\_function]);

print\_state(top(stackPrint)->state);

pop(&stackPrint);

no\_action++;

}

}

### 17. struct Queue

typedef struct{

Node\* Elements[Maxlength];

int front, rear;

}Queue;

int empty\_Queue(Queue Q){

return Q.front == -1;

}

void makeNull\_Queue(Queue \*Q){

Q->rear = -1;

Q->front = -1;

}

int full\_Queue(Queue Q){

return ((Q.rear - Q.front + 1) % Maxlength) == 0;

}

Node\* get\_Front(Queue Q){

if(empty\_Queue(Q)){

printf("Queue is empty");

} else {

return Q.Elements[Q.front];

}

}

void del\_Queue(Queue \*Q){

if(!empty\_Queue(\*Q)){

if(Q->front == Q->rear){

makeNull\_Queue(Q);

} else {

Q->front = (Q->front+1) % Maxlength;

}

} else {

printf("Error, Delete");

}

}

void push\_Queue(Node\* x, Queue \*Q){

if(!full\_Queue(\*Q)){

if(empty\_Queue(\*Q)){

Q->front = 0;

}

Q->rear = (Q->rear+1) % Maxlength;

Q->Elements[Q->rear] = x;

} else {

printf("Error, delete");

}

}

int CompareState(State state1, State state2){

if(state1.x == state2.x && state1.y == state2.y){

return 1;

}

return 0;

}

int find\_State(State state, Queue open){

while(!empty\_Queue(open)){

if(CompareState(get\_Front(open)->state, state)){

return 1;

}

del\_Queue(&open);

}

return 0;

}

### 18. BFS\_Algorithm

Node\* BFS\_Algorithm(State state){

Queue open, close;

makeNull\_Queue(&open);

makeNull\_Queue(&close);

Node\* root = (Node\*)malloc(sizeof(Node));

root->no\_function = 0;

root->Parent = NULL;

root->state = state;

push\_Queue(root, &open);

while(!empty\_Queue(open)){

Node\* node = get\_Front(open);

del\_Queue(&open);

push\_Queue(node, &close);

if(goalcheck(node->state)){

return node;

}

int opt;

for(opt = 1; opt <=6; opt++){

State newstate;

makeNullState(&newstate);

if(call\_operator(node->state, &newstate, opt)){

if(find\_State(newstate, open) || find\_State(newstate, close)){

continue;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->no\_function = opt;

newNode->Parent = node;

newNode->state = newstate;

push\_Queue(newNode, &open);

}

}

}

return NULL;

}

void print\_Goal(Node\* node){

Node\* Elements[Maxlength];

int index = 0;

while(node->Parent != NULL){

Elements[index] = node;

index++;

node = node->Parent;

}

int i;

int no\_action=0;

for(i = index-1; i>=0; i--){

printf("\nAction %d: %s", no\_action, action[Elements[i]->no\_function]);

print\_state(Elements[i]->state);

no\_action++;

}

}

int main(){

State cur\_state, result;

cur\_state.x = 0;

cur\_state.y = 0;

Node\* p = DFS\_Algorithm(cur\_state);

print\_WaysToGetGoal(p);

return 0;

}

## II. DONG SUA

### 1. cac action

int PourXY(State cur, State \*result){

if(cur.x > 0 && cur.y < tank\_Y){

result->x = cur.x - (tank\_Y - cur.y);

if(result->x < 0){

result->x = 0;

}

result->y = cur.x + cur.y;

if(result->y > tank\_Y){

result->y = tank\_Y;

}

result->z = cur.z;

return 1;

}

return 0;

}

int PourXZ(State cur, State \*result){

if(cur.x > 0 && cur.z < tank\_Z){

result->x = cur.x - (tank\_Z - cur.z);

if(result->x < 0){

result->x = 0;

}

result->z = cur.x + cur.z;

if(result->z > tank\_Z){

result->z = tank\_Z;

}

result->y = cur.y;

return 1;

}

return 0;

}

int PourYX(State cur, State \*result){

if(cur.y > 0 && cur.x < tank\_X){

result->y = cur.y - (tank\_X - cur.x);

if(result->y < 0){

result->y = 0;

}

result->x = cur.x + cur.y;

if(result->x > tank\_X){

result->x = tank\_X;

}

result->z = cur.z;

return 1;

}

return 0;

}

int PourYZ(State cur, State \*result){

if(cur.y > 0 && cur.z < tank\_Z){

result->y = cur.y - (tank\_Z - cur.z);

if(result->y < 0){

result->y = 0;

}

result->z = cur.y + cur.z;

if(result->z > tank\_Z){

result->z = tank\_Z;

}

result->x = cur.x;

return 1;

}

return 0;

}

int PourZX(State cur, State \*result){

if(cur.z > 0 && cur.x < tank\_X){

result->z = cur.z - (tank\_X - cur.x);

if(result->z < 0){

result->z = 0;

}

result->x = cur.x + cur.z;

if(result->x > tank\_X){

result->x = tank\_X;

}

result->y = cur.y;

return 1;

}

return 0;

}

int PourZY(State cur, State \*result){

if(cur.z > 0 && cur.y < tank\_Y){

result->z = cur.z - (tank\_Y - cur.y);

if(result->z < 0){

result->z = 0;

}

result->y = cur.z + cur.y;

if(result->y > tank\_Y){

result->y = tank\_Y;

}

result->x = cur.x;

return 1;

}

return 0;

}

### 2. call\_operator

int call\_operator(State cur, State \*result, int opt){

switch (opt){

case 1: return PourXY(cur, result);

case 2: return PourXZ(cur, result);

case 3: return PourYX(cur, result);

case 4: return PourYZ(cur, result);

case 5: return PourZX(cur, result);

case 6: return PourZY(cur, result);

default:

printf("Error, Call Operator");

return 0;

}

}

### 3. compare state

int compareState(State x, State y){

return (x.x == y.x && x.y == y.y && x.z == y.z);

}

### 4. find state Stack

int findState(State state, Stack S){

while(!emptyStack(S)){

Node\* node = top(S);

if(compareState(node->state, state)){

return 1;

}

pop(&S);

}

return 0;

}

### 5. find state Queue

int findState(State state, Queue Q){

while(!emptyQueue(Q)){

Node\* node = getFront(Q);

if(compareState(state, node->state)){

return 1;

}

delQueue(&Q);

}

return 0;

}

### 6. BFS

Node\* BFS(State state){

Queue open;

Queue close;

makeNullQueue(&open);

makeNullQueue(&close);

Node\* root = (Node\*)malloc(sizeof(Node));

root->no\_function = 0;

root->Parent = NULL;

root->state = state;

pushQueue(root, &open);

while(!emptyQueue(open)){

Node\* node = getFront(open);

delQueue(&open);

pushQueue(node, &close);

if(checkgoal(node->state)){

return node;

}

int opt;

for(opt = 1; opt <= 6; opt++){

State newstate;

makeNullState(&newstate);

if(call\_operator(node->state, &newstate, opt)){

if(findState(newstate, open) || findState(newstate, close)){

continue;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->no\_function = opt;

newNode->Parent = node;

newNode->state = newstate;

pushQueue(newNode, &open);

}

}

}

return NULL;

}

### 7. print goal

void printGoal(Node\* p){

Node\* re[Maxlength];

int index = 0;

while(p->Parent != NULL){

re[index] = p;

index++;

p = p->Parent;

}

re[index] = p;

index++;

int i;

int dem = 0;

for(i = index-1; i >= 0; i--){

printf("Action %d: %s\n",dem, action[re[i]->no\_function]);

dem++;

printState(re[i]->state);

}

}

## III. Tu Si Va Ke An Thit

### 1. BFS

#define Maxlength 100

typedef struct{

int si;

int quy;

char vt;

}State;

const char\* action[] = {"First action", "Chuyen 1 tu si A->B", "Chuyen 2 tu si A->B", "Chuyen 1 con quy A->B", "Chuyen 2 con quy A->B","Chuyen 1 tu si va 1 con quy A->B", "Chuyen 1 tu si B->A", "Chuyen 2 tu si B->A", "Chuyen 1 con quy B->A", "Chuyen 2 con quy B->A", "Chuyen 1 tu si va 1 con quy B->A"};

void printState(State state){

printf("Tu si: %d ----- Con quy: %d ----- Vi tri bo: %c\n", state.si, state.quy, state.vt);

}

typedef struct Node{

State state;

struct Node\* Parent;

int no\_function;

}Node;

int goalcheck(State state){

return state.si == 0 && state.quy == 0 && state.vt == 'B';

}

int checkOver(State cur){

if(cur.si == cur.quy){

return 1;

}

if(cur.si == 0 || cur.si == 3){

return 1;

}

return 0;

}

int AB1S(State cur, State \*re){

if(cur.si >= 1){

re->si = cur.si - 1;

re->quy = cur.quy;

re->vt = 'B';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int AB2S(State cur, State \*re){

if(cur.si >= 2){

re->si = cur.si - 2;

re->quy = cur.quy;

re->vt = 'B';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int AB1Q(State cur, State \*re){

if(cur.quy >= 1){

re->si = cur.si;

re->quy = cur.quy - 1;

re->vt = 'B';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int AB2Q(State cur, State \*re){

if(cur.quy >= 2){

re->si = cur.si;

re->quy = cur.quy - 2;

re->vt = 'B';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int AB1S1Q(State cur, State \*re){

if(cur.quy >= 1 && cur.si >= 1){

re->si = cur.si - 1;

re->quy = cur.quy - 1;

re->vt = 'B';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int BA1S(State cur, State \*re){

if((3-cur.si) >= 1){

re->si = cur.si + 1;

re->quy = cur.quy;

re->vt = 'A';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int BA2S(State cur, State \*re){

if((3-cur.si) >= 2){

re->si = cur.si + 2;

re->quy = cur.quy;

re->vt = 'A';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int BA1Q(State cur, State \*re){

if((3-cur.quy) >= 1){

re->si = cur.si;

re->quy = cur.quy + 1;

re->vt = 'A';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int BA2Q(State cur, State \*re){

if((3-cur.quy) >= 2){

re->si = cur.si;

re->quy = cur.quy + 2;

re->vt = 'A';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int BA1S1Q(State cur, State \*re){

if((3-cur.quy) >= 1 && (3-cur.si) >=1){

re->si = cur.si + 1;

re->quy = cur.quy + 1;

re->vt = 'A';

if(checkOver(\*re)){

return 1;

}

}

return 0;

}

int call\_operator(State cur, State \*re, int opt){

if(cur.vt == 'A'){

switch(opt){

case 1: return AB1S(cur, re);

case 2: return AB2S(cur, re);

case 3: return AB1Q(cur, re);

case 4: return AB2Q(cur, re);

case 5: return AB1S1Q(cur, re);

}

} else if(cur.vt == 'B') {

switch(opt){

case 6: return BA1S(cur, re);

case 7: return BA2S(cur, re);

case 8: return BA1Q(cur, re);

case 9: return BA2Q(cur, re);

case 10: return BA1S1Q(cur, re);

}

}

return 0;

}

int findState(State state, Queue Q){

while(!empty\_Queue(Q)){

Node\* node = get\_Front(Q);

if(state.si == node->state.si && state.quy == node->state.quy && state.vt == node->state.vt){

return 1;

}

del\_Queue(&Q);

}

return 0;

}

Node\* BFS(State state){

Queue open;

Queue close;

makeNull\_Queue(&open);

makeNull\_Queue(&close);

Node\* root = (Node\*)malloc(sizeof(Node));

root->no\_function = 0;

root->Parent = NULL;

root->state = state;

push\_Queue(root, &open);

while(!empty\_Queue(open)){

Node\* node = get\_Front(open);

// printState(node->state);

del\_Queue(&open);

push\_Queue(node, &close);

if(goalcheck(node->state)){

return node;

}

int opt;

for(opt = 1; opt <= 10; opt++){

State newstate;

if(call\_operator(node->state, &newstate, opt)){

if(findState(newstate, open) || findState(newstate, close)){

continue;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->no\_function = opt;

newNode->Parent = node;

newNode->state = newstate;

push\_Queue(newNode, &open);

}

}

}

return NULL;

}

## IV. UP LY

const char\* action[] = {"First action", "UP 1 2 3", "UP 2 3 4", "UP 3 4 5", "UP 4 5 6"};

typedef struct{

int vt1, vt2, vt3, vt4, vt5, vt6;

}State;

int checkGoal(State s){

return s.vt1 == 1 && s.vt2 == 1 && s.vt3 == 1 && s.vt4 == 1 && s.vt5 == 1 && s.vt6 == 1;

}

int A123(State cur, State \*re){

re->vt1 = -cur.vt1;

re->vt2 = -cur.vt2;

re->vt3 = -cur.vt3;

re->vt4 = cur.vt4;

re->vt5 = cur.vt5;

re->vt6 = cur.vt6;

return 1;

}

int A234(State cur, State \*re){

re->vt1 = cur.vt1;

re->vt2 = -cur.vt2;

re->vt3 = -cur.vt3;

re->vt4 = -cur.vt4;

re->vt5 = cur.vt5;

re->vt6 = cur.vt6;

return 1;

}

int A345(State cur, State \*re){

re->vt1 = cur.vt1;

re->vt2 = cur.vt2;

re->vt3 = -cur.vt3;

re->vt4 = -cur.vt4;

re->vt5 = -cur.vt5;

re->vt6 = cur.vt6;

return 1;

}

int A456(State cur, State \*re){

re->vt1 = cur.vt1;

re->vt2 = cur.vt2;

re->vt3 = cur.vt3;

re->vt4 = -cur.vt4;

re->vt5 = -cur.vt5;

re->vt6 = -cur.vt6;

return 1;

}

int call\_operator(State cur, State \*re, int opt){

switch(opt){

case 1: return A123(cur, re);

case 2: return A234(cur, re);

case 3: return A345(cur, re);

case 4: return A456(cur, re);

default:

printf("Error, call Operator");

return 0;

}

}

int compareState(State x, State y){

return x.vt1 == y.vt1 && x.vt2 == y.vt2 && x.vt3 == y.vt3 && x.vt4 == y.vt4 && x.vt5 == y.vt5 && x.vt6 == y.vt6;

}

## V. 8 puzzle Hueristic

#include <stdio.h>

#include <stdlib.h>

#define ROWS 3

#define COLS 3

#define EMPTY 0

#define MAX\_OPERATOR 4

#define Maxlength 500

const char\* action[] = {"First State", "Move cell EMPTY to UP",

"Move cell EMPTY to DOWN",

"Move cell EMPTY to LEFT",

"Move cell EMPTY to RIGHT"};

typedef struct {

int puzzel[ROWS][COLS];

int emptyRow;

int emptyCol;

}State;

typedef struct Node {

State state;

struct Node\* Parent;

int no\_func;

int heuristic;

}Node;

typedef struct {

Node\* element[Maxlength];

int size;

}List;

void makeNullList(List \*L){

L->size = 0;

}

int emptyList(List L){

return L.size == 0;

}

int fullList(List L){

return L.size == Maxlength;

}

Node\* elementAt(int p, List L){

return L.element[p-1];

}

void pushList(Node\* node,int position, List \*L){

if(!fullList(\*L)){

int q;

for(q = L->size; q >= position; q--){

L->element[q] = L->element[q-1];

}

L->element[position-1] = node;

L->size++;

} else {

printf("List is full\n");

}

}

void deleteList(int position, List \*L){

if(emptyList(\*L)){

printf("List is empty\n");

} else if(position < 1 || position > L->size){

printf("position iss not possible to delete\n");

} else {

int i;

for(i = position-1; i < L->size; i++){

L->element[i] = L->element[i+1];

}

L->size--;

}

}

int compareState(State state1, State state2){

if(state1.emptyRow != state2.emptyRow || state1.emptyCol != state2.emptyCol){

return 0;

}

int row, col;

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

if(state1.puzzel[row][col] != state2.puzzel[row][col]){

return 0;

}

}

}

return 1;

}

Node\* findState(State state, List L, int \*position){

int i;

for(i = 1; i <= L.size; i++){

if(compareState(elementAt(i, L)->state, state)){

\*position = i;

return elementAt(i, L);

}

}

return NULL;

}

void sortList(List \*L){

int i, j;

for(i = 0; i < L->size-1; i++){

for(j=i+1; j<L->size; j++){

if(L->element[i]->heuristic > L->element[j]->heuristic){

Node\* node = L->element[i];

L->element[i] = L->element[j];

L->element[j] = node;

}

}

}

}

void printState(State state){

int row, col;

printf("\n-------------\n");

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

printf("| %d ", state.puzzel[row][col]);

}

printf("|\n");

}

printf("-------------\n");

}

int goalCheck(State state, State goal){

return compareState(state, goal);

}

int UP(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ERC > 0){

r->emptyRow = ERC - 1;

r->emptyCol = ECC;

r->puzzel[ERC][ECC] = state.puzzel[ERC-1][ECC];

r->puzzel[ERC-1][ECC] = EMPTY;

return 1;

}

return 0;

}

int DOWN(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ERC < ROWS-1){

r->emptyRow = ERC + 1;

r->emptyCol = ECC;

r->puzzel[ERC][ECC] = state.puzzel[ERC+1][ECC];

r->puzzel[ERC+1][ECC] = EMPTY;

return 1;

}

return 0;

}

int LEFT(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ECC > 0){

r->emptyRow = ERC;

r->emptyCol = ECC-1;

r->puzzel[ERC][ECC] = state.puzzel[ERC][ECC-1];

r->puzzel[ERC][ECC-1] = EMPTY;

return 1;

}

return 0;

}

int RIGHT(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ECC < COLS-1){

r->emptyRow = ERC;

r->emptyCol = ECC+1;

r->puzzel[ERC][ECC] = state.puzzel[ERC][ECC+1];

r->puzzel[ERC][ECC+1] = EMPTY;

return 1;

}

return 0;

}

int callOperator(State cur, State \*r, int opt){

switch (opt){

case 1: return UP(cur, r);

case 2: return DOWN(cur, r);

case 3: return LEFT(cur, r);

case 4: return RIGHT(cur, r);

default: printf("Error, call operator");

return 0;

}

}

int heuristic1(State state, State goal){

int row, col, count;

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

if(state.puzzel[row][col] != goal.puzzel[row][col]){

count++;

}

}

}

return count;

}

Node\* BFS(State state, State goal){

List open;

List close;

makeNullList(&open);

makeNullList(&close);

Node\* root = (Node\*)malloc(sizeof(Node));

root->state = state;

root->Parent = NULL;

root->no\_func = 0;

root->heuristic = heuristic1(state, goal);

pushList(root, open.size+1, &open);

while(!emptyList(open)){

Node\* node = elementAt(1, open);

deleteList(1, &open);

pushList(node, close.size+1, &close);

if(goalCheck(node->state, goal)){

return node;

}

int opt;

for(opt = 1; opt <= 4; opt++){

State newState;

if(callOperator(node->state, &newState, opt)){

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->state = newState;

newNode->no\_func = opt;

newNode->Parent = node;

newNode->heuristic = heuristic1(newState, goal);

int pos\_open, pos\_close;

Node\* nodeFoundOpen = findState(newState, open, &pos\_open);

Node\* nodeFoundClose = findState(newState, close, &pos\_close);

if(nodeFoundOpen == NULL && nodeFoundClose == NULL){

pushList(newNode, open.size+1, &open);

} else if(nodeFoundOpen != NULL && nodeFoundOpen->heuristic > newNode->heuristic){

deleteList(pos\_open, &open);

pushList(newNode, open.size+1, &open);

} else if(nodeFoundClose != NULL && nodeFoundClose->heuristic > newNode->heuristic){

deleteList(pos\_close, &close);

pushList(newNode, close.size+1, &close);

}

}

sortList(&open);

}

}

return NULL;

}

void printGoal(Node\* p){

Node\* Array[Maxlength];

int index = 0;

while(p->Parent != NULL){

Array[index] = p;

index++;

p=p->Parent;

}

Array[index] = p;

index++;

int i;

int dem = 0;

for(i = index-1; i>=0; i--){

printf("Action %d: %s\n", dem, action[Array[i]->no\_func]);

printState(Array[i]->state);

dem++;

}

}

int main(){

State state;

state.emptyCol = 1;

state.emptyRow = 1;

state.puzzel[0][0] = 3;

state.puzzel[0][1] = 4;

state.puzzel[0][2] = 5;

state.puzzel[1][0] = 1;

state.puzzel[1][1] = 0;

state.puzzel[1][2] = 2;

state.puzzel[2][0] = 6;

state.puzzel[2][1] = 7;

state.puzzel[2][2] = 8;

State goal;

goal.emptyCol = 0;

goal.emptyRow = 0;

goal.puzzel[0][0] = 0;

goal.puzzel[0][1] = 1;

goal.puzzel[0][2] = 2;

goal.puzzel[1][0] = 3;

goal.puzzel[1][1] = 4;

goal.puzzel[1][2] = 5;

goal.puzzel[2][0] = 6;

goal.puzzel[2][1] = 7;

goal.puzzel[2][2] = 8;

Node\* p = BFS(state, goal);

printGoal(p);

return 0;

}

## VI. 8 puzzle A start

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#define ROWS 3

#define COLS 3

#define EMPTY 0

#define MAX\_OPERATOR 4

#define Maxlength 500

const char\* action[] = {"First State", "Move cell EMPTY to UP",

"Move cell EMPTY to DOWN",

"Move cell EMPTY to LEFT",

"Move cell EMPTY to RIGHT"};

typedef struct {

int puzzel[ROWS][COLS];

int emptyRow;

int emptyCol;

}State;

typedef struct{

State state;

struct Node\* Parent;

int no\_func;

int f;

int h;

int g;

}Node;

typedef struct {

Node\* element[Maxlength];

int size;

}List;

void makeNullList(List \*L){

L->size = 0;

}

int emptyList(List L){

return L.size == 0;

}

int fullList(List L){

return L.size == Maxlength;

}

Node\* elementAt(int p, List L){

return L.element[p-1];

}

void pushList(Node\* node,int position, List \*L){

if(!fullList(\*L)){

int q;

for(q = L->size; q >= position; q--){

L->element[q] = L->element[q-1];

}

L->element[position-1] = node;

L->size++;

} else {

printf("List is full\n");

}

}

void deleteList(int position, List \*L){

if(emptyList(\*L)){

printf("List is empty\n");

} else if(position < 1 || position > L->size){

printf("position iss not possible to delete\n");

} else {

int i;

for(i = position-1; i < L->size; i++){

L->element[i] = L->element[i+1];

}

L->size--;

}

}

int compareState(State state1, State state2){

if(state1.emptyRow != state2.emptyRow || state1.emptyCol != state2.emptyCol){

return 0;

}

int row, col;

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

if(state1.puzzel[row][col] != state2.puzzel[row][col]){

return 0;

}

}

}

return 1;

}

Node\* findState(State state, List L, int \*position){

int i;

for(i = 1; i <= L.size; i++){

if(compareState(elementAt(i, L)->state, state)){

\*position = i;

return elementAt(i, L);

}

}

return NULL;

}

void sortList(List \*L){

int i, j;

for(i = 0; i < L->size-1; i++){

for(j=i+1; j<L->size; j++){

if(L->element[i]->f > L->element[j]->f){

Node\* node = L->element[i];

L->element[i] = L->element[j];

L->element[j] = node;

}

}

}

}

void printState(State state){

int row, col;

printf("\n-------------\n");

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

printf("| %d ", state.puzzel[row][col]);

}

printf("|\n");

}

printf("-------------\n");

}

int goalCheck(State state, State goal){

return compareState(state, goal);

}

int UP(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ERC > 0){

r->emptyRow = ERC - 1;

r->emptyCol = ECC;

r->puzzel[ERC][ECC] = state.puzzel[ERC-1][ECC];

r->puzzel[ERC-1][ECC] = EMPTY;

return 1;

}

return 0;

}

int DOWN(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ERC < ROWS-1){

r->emptyRow = ERC + 1;

r->emptyCol = ECC;

r->puzzel[ERC][ECC] = state.puzzel[ERC+1][ECC];

r->puzzel[ERC+1][ECC] = EMPTY;

return 1;

}

return 0;

}

int LEFT(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ECC > 0){

r->emptyRow = ERC;

r->emptyCol = ECC-1;

r->puzzel[ERC][ECC] = state.puzzel[ERC][ECC-1];

r->puzzel[ERC][ECC-1] = EMPTY;

return 1;

}

return 0;

}

int RIGHT(State state, State \*r){

\*r = state;

int ERC = state.emptyRow;

int ECC = state.emptyCol;

if(ECC < COLS-1){

r->emptyRow = ERC;

r->emptyCol = ECC+1;

r->puzzel[ERC][ECC] = state.puzzel[ERC][ECC+1];

r->puzzel[ERC][ECC+1] = EMPTY;

return 1;

}

return 0;

}

int callOperator(State cur, State \*r, int opt){

switch (opt){

case 1: return UP(cur, r);

case 2: return DOWN(cur, r);

case 3: return LEFT(cur, r);

case 4: return RIGHT(cur, r);

default: printf("Error, call operator");

return 0;

}

}

int heuristic1(State state, State goal){

int row, col, count;

for(row = 0; row < ROWS; row++){

for(col = 0; col < COLS; col++){

if(state.puzzel[row][col] != goal.puzzel[row][col]){

count++;

}

}

}

return count;

}

Node\* BFS(State state, State goal){

List open;

List close;

makeNullList(&open);

makeNullList(&close);

Node\* root = (Node\*)malloc(sizeof(Node));

root->state = state;

root->Parent = NULL;

root->no\_func = 0;

root->g = 0;

root->h = heuristic1(root->state, goal);

root->f = root->g + root->h;

pushList(root, open.size+1, &open);

while(!emptyList(open)){

Node\* node = elementAt(1, open);

deleteList(1, &open);

pushList(node, close.size+1, &close);

if(goalCheck(node->state, goal)){

return node;

}

int opt;

for(opt = 1; opt <= 4; opt++){

State newState;

if(callOperator(node->state, &newState, opt)){

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->state = newState;

newNode->no\_func = opt;

newNode->Parent = node;

newNode->g = node->g + 1;

newNode->h = heuristic1(newState, goal);

newNode->f = newNode->g + newNode->h;

int pos\_open, pos\_close;

Node\* nodeFoundOpen = findState(newState, open, &pos\_open);

Node\* nodeFoundClose = findState(newState, close, &pos\_close);

if(nodeFoundOpen == NULL && nodeFoundClose == NULL){

pushList(newNode, open.size+1, &open);

} else if(nodeFoundOpen != NULL && nodeFoundOpen->g > newNode->g){

deleteList(pos\_open, &open);

pushList(newNode, pos\_open, &open);

} else if(nodeFoundClose != NULL && nodeFoundClose->g > newNode->g){

deleteList(pos\_close, &close);

pushList(newNode, open.size+1, &open);

}

}

sortList(&open);

}

}

return NULL;

}

void printGoal(Node\* p){

Node\* Array[Maxlength];

int index = 0;

while(p->Parent != NULL){

Array[index] = p;

index++;

p=p->Parent;

}

Array[index] = p;

index++;

int i;

int dem = 0;

for(i = index-1; i>=0; i--){

printf("Action %d: %s\n", dem, action[Array[i]->no\_func]);

printState(Array[i]->state);

dem++;

}

}

int main(){

State state;

state.emptyRow = 1;

state.emptyCol = 1;

state.puzzel[0][0] = 1;

state.puzzel[0][1] = 2;

state.puzzel[0][2] = 3;

state.puzzel[1][0] = 8;

state.puzzel[1][1] = 0;

state.puzzel[1][2] = 4;

state.puzzel[2][0] = 7;

state.puzzel[2][1] = 6;

state.puzzel[2][2] = 5;

State goal;

goal.emptyRow = 1;

goal.emptyCol = 0;

goal.puzzel[0][0] = 2;

goal.puzzel[0][1] = 8;

goal.puzzel[0][2] = 1;

goal.puzzel[1][0] = 0;

goal.puzzel[1][1] = 4;

goal.puzzel[1][2] = 3;

goal.puzzel[2][0] = 7;

goal.puzzel[2][1] = 6;

goal.puzzel[2][2] = 5;

Node\* p = BFS(state, goal);

printGoal(p);

return 0;

}

## VII. sudoku thoa man rang buoc

#include <stdio.h>

#include <stdlib.h>

#define maxLength 100

#define ROWS 9

#define COLUMNS 9

#define EMPTY 0

#define AREA\_SQUARE 3

#define INF 999999

#define max\_value 10

int Counter = 0;

typedef struct {

int x;

int y;

}Coord;

typedef struct {

Coord data[maxLength];

int size;

}ListCoord;

typedef struct{

int data[ROWS\*COLUMNS][ROWS\*COLUMNS];

int n;

}Constrains;

typedef struct {

int cells[ROWS][COLUMNS];

Constrains constrains;

}Sudoku;

typedef struct {

int element[maxLength];

int size;

}List;

void makeNullList(List \*L){

L->size = 0;

}

int emptyList(List L){

return L.size == 0;

}

int fullList(List L){

return L.size == maxLength;

}

int elementAt(int p, List L){

return L.element[p-1];

}

void pushList(List \*L, int x){

if(!fullList(\*L)){

L->element[L->size] = x;

L->size++;

} else {

printf("List is full\n");

}

}

void deleteList(int position, List \*L){

if(emptyList(\*L)){

printf("List is empty\n");

} else if(position < 1 || position > L->size){

printf("position iss not possible to delete\n");

} else {

int i;

for(i = position-1; i < L->size; i++){

L->element[i] = L->element[i+1];

}

L->size--;

}

}

void initSudoku(Sudoku \*sudoku){

int i, j;

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

sudoku->cells[i][j] = 0;

}

}

initConstrains(&sudoku->constrains);

}

void initListCoord(ListCoord \*L){

L->size = 0;

}

void appendList(ListCoord \*L, Coord coord){

L->data[L->size] = coord;

L->size++;

}

void initConstrains(Constrains \*Con){

int i,j;

for(i=0; i<81; i++){

for(j=0; j<81; j++){

Con->data[i][j] = 0;

}

}

Con->n = ROWS\*COLUMNS;

}

int indexOf(Coord coord){ // chuyen doi chi so toa do thanh chi so dinh

return (ROWS\*coord.x + coord.y);

}

Coord positionOfVertex(int vertex){

Coord coord;

coord.x = vertex / ROWS;

coord.y = vertex % COLUMNS;

return coord;

}

int addConstrain(Constrains \*con, Coord source, Coord target){

int u = indexOf(source);

int v = indexOf(target);

if(con->data[u][v] == 0){

con->data[u][v] = 1;

con->data[v][u] = 1;

return 1;

}

return 0;

}

ListCoord getConstrains (Constrains con, Coord coord){

int i;

int v = indexOf(coord);

ListCoord result;

initListCoord(&result);

for(i=0; i<con.n; i++){

if(con.data[v][i] == 1){

appendList(&result, positionOfVertex(i));

}

}

return result;

}

void initSudokuValues(Sudoku \*sudoku, int inputs[ROWS][COLUMNS]){

int i, j;

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

sudoku->cells[i][j] = inputs[i][j];

}

}

initConstrains(&sudoku->constrains);

}

void printSudoku(Sudoku sudoku){

int i, j;

printf("Sudoku:\n");

for(i=0; i<ROWS; i++){

if(i % AREA\_SQUARE == 0) printf("-------------------------\n");

for(j=0; j<COLUMNS; j++){

if(j % AREA\_SQUARE == 0) printf("| ");

printf("%d ", sudoku.cells[i][j]);

}

printf("|\n");

}

printf("-------------------------\n");

}

int isFilledSudoku(Sudoku sudoku){

int i, j;

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

if(sudoku.cells[i][j] == 0) return 0;

}

}

return 1;

}

void spreadConstrainsFrom(Coord position, Constrains \*con, ListCoord \*changeds){

int row = position.x;

int column = position.y;

int i, j;

//tao rang buoc theo cot

for(i=0; i<ROWS; i++){

if(i != row){

Coord pos = {i, column};

if(addConstrain(con, position, pos)){

appendList(changeds, pos);

}

}

}

//tao rang buoc theo hang

for(i=0; i<COLUMNS; i++){

if(i != column){

Coord pos = {row, i};

if(addConstrain(con, position, pos)){

appendList(changeds, pos);

}

}

}

// tao rang buoc theo khoi

for(i=0; i<AREA\_SQUARE; i++){

for(j=0; j<AREA\_SQUARE; j++){

int areaX = (row/AREA\_SQUARE)\*AREA\_SQUARE;

int areaY = (column/AREA\_SQUARE)\*AREA\_SQUARE;

if(areaX+i != row || areaY+j != column){

Coord pos = {areaX+i, areaY+j};

if(addConstrain(con, position, pos)){

appendList(changeds, pos);

}

}

}

}

}

List getAvailableValues(Coord position, Sudoku sudoku){

ListCoord posList = getConstrains(sudoku.constrains, position);

int i;

int availables[max\_value];

for(i = 1; i<max\_value; i++){

availables[i]=1;

}

for(i = 0; i<posList.size; i++){

Coord pos = posList.data[i];

if(sudoku.cells[pos.x][pos.y] != EMPTY){

availables[sudoku.cells[pos.x][pos.y]]= 0;

}

}

List result;

makeNullList(&result);

for(i=1; i<max\_value; i++){

if(availables[i]) pushList(&result, i);

}

return result;

}

Coord getNextEmptyCell(Sudoku sudoku){

int i, j;

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

Coord pos = {i, j};

if(sudoku.cells[i][j] == EMPTY) return pos;

}

}

}

Coord getNextMinDomainCell(Sudoku sudoku){

int minLength = INF;

int i, j, a;

Coord result;

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

if(sudoku.cells[i][j] == EMPTY){

Coord pos = {i, j};

List mienGia = getAvailableValues(pos, sudoku);

int availablesLength = mienGia.size;

if(availablesLength < minLength){

minLength = availablesLength;

result = pos;

}

}

}

}

return result;

}

void clearConstrains(Constrains \*con){

int i, j;

for(i=0; i<81; i++){

for(j=0; j<81; j++){

con->data[i][j] = 0;

}

}

}

int sudokuBackTracking(Sudoku \*sudoku) {

int i, j;

clearConstrains(&sudoku->constrains);

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

if(sudoku->cells[i][j] != EMPTY){

ListCoord history;

initListCoord(&history);

Coord pos = {i, j};

spreadConstrainsFrom(pos, &sudoku->constrains, &history);

}

}

}

if(isFilledSudoku(\*sudoku)) {

return 1;

}

Coord position = getNextMinDomainCell(\*sudoku);

List availables = getAvailableValues(position, \*sudoku);

if(availables.size == 0){

return 0;

}

for(j=0; j<availables.size; j++){

int value = availables.element[j];

sudoku->cells[position.x][position.y] = value;

Counter++;

if(sudokuBackTracking(sudoku)){

return 1;

}

sudoku->cells[position.x][position.y] = EMPTY;

}

return 0;

}

Sudoku resolveSudoku(Sudoku sudoku){

int i, j;

clearConstrains(&sudoku.constrains);

for(i=0; i<ROWS; i++){

for(j=0; j<COLUMNS; j++){

if(sudoku.cells[i][j] != EMPTY){

ListCoord history;

initListCoord(&history);

Coord pos = {i, j};

spreadConstrainsFrom(pos, &sudoku.constrains, &history);

}

}

}

Counter = 0;

if(sudokuBackTracking(&sudoku)){

printf("Solved\n");

} else{

printf("Can not Solve\n");

}

printf("Explored State %d states\n", Counter);

return sudoku;

}

int inputs1[9][9] = {

{9,7,0,0,1,3,2,0,0},

{0,0,0,0,0,0,6,0,7},

{0,8,0,0,0,0,0,0,0},

{0,0,0,0,0,0,9,8,0},

{0,9,0,5,0,0,0,7,0},

{0,0,0,0,0,2,0,0,0},

{0,0,0,0,3,5,0,1,0},

{0,3,4,9,0,0,0,0,0},

{0,0,6,0,0,0,8,0,0}};

int main(){

Sudoku sudoku;

initSudoku(&sudoku);

initSudokuValues(&sudoku, inputs1);

printSudoku(sudoku);

Sudoku result = resolveSudoku(sudoku);

printSudoku(result);

return 0;

}

## IX. rang buoc 8 quan hau

#include <stdio.h>

#define size 8

#define maxLength 100

typedef struct{

int banco[size][size];

int constrain[size][size];

int soquan;

}HAU;

typedef struct {

int x;

int y;

}Coord;

typedef struct {

Coord data[maxLength];

int count;

}ListCoord;

void initListCoord(ListCoord \*L){

L->count = 0;

}

void appendList(ListCoord \*L, Coord coord){

L->data[L->count] = coord;

L->count++;

}

void clearBanCo(int banco[size][size]){

int i, j;

for (i=0; i<size; i++){

for(j=0; j<size; j++){

banco[i][j]=0;

}

}

}

void clearConstrain(int constrain[size][size]){

int i, j;

for (i=0; i<size; i++){

for(j=0; j<size; j++){

constrain[i][j]=0;

}

}

}

void taoRangBuoc(HAU \*hau){

int i, j;

for(i=0; i<size; i++){

for(j=0; j<size; j++){

if(hau->banco[i][j] != 0){

int num = hau->banco[i][j];

hau->constrain[i][j] = num;

int c, h;

// cot

for(c=0; c<size; c++){

hau->constrain[c][j] = num;

}

//hang

for(c=0; c<size; c++){

hau->constrain[i][c] = num;

}

//cheo1

c=j;

h=i;

while(1){

hau->constrain[h][c] = num;

h--;

c++;

if(h<0 || c ==size) break;

}

//cheo2

c=j;

h=i;

while(1){

hau->constrain[h][c] = num;

h--;

c--;

if(h<0 || c<0) break;

}

//cheo3

c=j;

h=i;

while(1){

hau->constrain[h][c] = num;

h++;

c--;

if(h == size || c<0) break;

}

//cheo 4

c=j;

h=i;

while(1){

hau->constrain[h][c] = num;

h++;

c++;

if(h ==size || c == size) break;

}

}

}

}

}

void printArray(int a[size][size]){

int i, j;

for(i=0; i<size; i++){

for(j=0; j<size; j++){

if(a[i][j] != 0){

printf("%c ", 254);

} else{

printf("%d ",a[i][j]);

}

}

printf("\n");

}

}

int backTracking(HAU \*hau){

clearConstrain(hau->constrain);

taoRangBuoc(hau);

// printf("rang buoc\n");

// printArray(hau->constrain);

int i, j;

if(hau->soquan == 9){

return 1;

}

int check=0;

for(i=0; i<size; i++){

for(j=0; j<size; j++){

if(hau->constrain[i][j] == 0){

check = 1;

}

}

}

if(check == 0){

return 0;

}

ListCoord list;

initListCoord(&list);

for(i=0; i<size; i++){

for(j=0; j<size; j++){

if(hau->constrain[i][j] == 0){

Coord coord = {i, j};

appendList(&list, coord);

}

}

}

for(i=0; i<list.count; i++){

hau->banco[list.data[i].x][list.data[i].y] = hau->soquan;

// printf("dien %d tai vi tri %d - %d\n",hau->soquan, list.data[i].x, list.data[i].y);

hau->soquan++;

if(backTracking(hau)){

return 1;

}

hau->banco[list.data[i].x][list.data[i].y] = 0;

hau->soquan--;

}

return 0;

}

int main(){

HAU hau;

clearBanCo(hau.banco);

clearConstrain(hau.constrain);

hau.soquan = 1;

taoRangBuoc(&hau);

int re = backTracking(&hau);

if(re == 1){

printf("Resolve\n", re);

}

printArray(hau.banco);

return 0;

}