**Đong nước**

#include <stdio.h>

#include <stdlib.h>

#define tankcapacity\_X 9 //suc chua binh X

#define tankcapacity\_Y 4 // suc chua binh Y

#define empty 0

#define goal 6 //muc tieu luong nuoc can dong

#define Maxlength 100 //sd cai dat stack

//Ten cac hanh dong

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"};

//Khai bao cau truc trang thai

typedef struct{

int x; //luong nuoc trong binh x

int y; //luong nuoc trong binh y

}State;

//Khoi tao trang thai binh x=0, y=0

void makeNullState(State \*state){

state->x=0;

state->y=0;

}

//In trang thai

void print\_State(State state){

printf("\n X:%d --- Y:%d",state.x, state.y);

}

//Kiem tra trang thai muc tieu

int goalcheck(State state){

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

}

//Lam day nuoc binh X

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;

}

//Lam day nuoc binh Y

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;

}

//Lam rong nuoc binh X

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

if(cur\_state.x > 0){

result->x=empty;

result->y=cur\_state.y;

return 1;

}

return 0;

}

//Lam rong nuoc binh Y

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

if(cur\_state.y>0){

result->y=empty;

result->x=cur\_state.x;

return 1;

}

return 0;

}

//Max

int max(int a, int b){

int max=0;

if(a>b) max=a;

else max=b;

return max;

}

//Min

int min(int a, int b){

int min=0;

if(a<b) min=a;

else min=b;

return min;

}

//Chuyen nuoc tu binh X sang binh Y

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

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

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

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

return 1;

}

return 0;

}

//Chuyen nuoc tu binh Y sang binh X

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

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

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

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

return 1;

}

return 0;

}

//Goi cac phep toan tren trang thai

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 operator");

return 0;

}

}

//main

int main(){

State cur\_state = {5,4}, result;

printf("Trang thai bat dau");

print\_State(cur\_state);

int opt;

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

int thuchien = call\_operator(cur\_state,&result,opt);

if(thuchien==1){ //thuc hien hanh dong thanh cong

printf("\n Hanh dong %s thanh cong", action[opt]);

print\_State(result);

}

else

printf("\n Hanh dong %s KHONG thanh cong", action[opt]);

}

return 0;

}

**Đong nước DFS**

#include <stdio.h>

#include <stdlib.h>

#define tankcapacity\_X 9 //suc chua binh X

#define tankcapacity\_Y 4 // suc chua binh Y

#define empty 0

#define goal 6 //muc tieu luong nuoc can dong

#define Maxlength 100 //sd cai dat stack

//Ten cac hanh dong

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"};

//Khai bao cau truc trang thai

typedef struct{

int x; //luong nuoc trong binh x

int y; //luong nuoc trong binh y

}State;

//Khoi tao trang thai binh x=0, y=0

void makeNullState(State \*state){

state->x=0;

state->y=0;

}

//In trang thai

void print\_State(State state){

printf("\n X:%d --- Y:%d",state.x, state.y);

}

//Kiem tra trang thai muc tieu

int goalcheck(State state){

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

}

//Lam day nuoc binh X

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;

}

//Lam day nuoc binh Y

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;

}

//Lam rong nuoc binh X

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

if(cur\_state.x > 0){

result->x=empty;

result->y=cur\_state.y;

return 1;

}

return 0;

}

//Lam rong nuoc binh Y

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

if(cur\_state.y>0){

result->y=empty;

result->x=cur\_state.x;

return 1;

}

return 0;

}

//Max

int max(int a, int b){

int max=0;

if(a>b) max=a;

else max=b;

return max;

}

//Min

int min(int a, int b){

int min=0;

if(a<b) min=a;

else min=b;

return min;

}

//Chuyen nuoc tu binh X sang binh Y

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

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

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

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

return 1;

}

return 0;

}

//Chuyen nuoc tu binh Y sang binh X

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

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

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

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

return 1;

}

return 0;

}

//Goi cac phep toan tren trang thai

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 operator");

return 0;

}

}

//Khai bao cau truc nut (dinh) de dung cay tim kiem

typedef struct Node{

State state; //trang thai cua nut

struct Node\* Parent; //nut cha

int no\_function; //thu tu phep toan

}Node;

//Khai bao cau truc Stack de luu trang thai duyet

typedef struct{

Node\* Elements[Maxlength];

int Top\_idx;

}Stack;

//Khoi tao ngan xep rong

void makeNull\_Stack(Stack \*stack){

stack->Top\_idx=Maxlength;

}

//Kiem tra ngan xep co rong hay khong

int empty\_Stack(Stack stack){

return stack.Top\_idx==Maxlength;

}

//Kiem tra ngan xep co day hay khong

int full\_Stack(Stack stack){

return stack.Top\_idx==0;

}

//Tra ve phan tu tren dinh ngan xep

Node\* top(Stack stack){

if(!empty\_Stack(stack))

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

return NULL;

}

//Xoa phan tu tai dinh ngan xep

void pop(Stack \*stack){

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

stack->Top\_idx+=1;

else printf("Error!Stack if empty");

}

//CompareStates

int compareStates(State s1, State s2){

if(s1.x==s2.x &&s1.y==s2.y)

return 1;

return 0;

}

//Tim kiem trang thai trong Stack Open/Close

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

while(!empty\_Stack(openStack)){

if(compareStates(top(openStack)->state,state))

return 1;

pop(&openStack);

}

return 0;

}

//Dua 1 phan tu len ngan xep

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

if(full\_Stack(\*stack))

printf("Error!Stack is full");

else{

stack->Top\_idx-=1;

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

}

}

//Thuat toan duyet theo chieu sau

Node\* DFS\_Algorithm(State state){

//Khai bao 2 ngan xep Open va Close

Stack Open\_DFS;

Stack Close\_DFS;

makeNull\_Stack(&Open\_DFS);

makeNull\_Stack(&Close\_DFS);

//Tao nut trang thai cha

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)){

//Lay 1 dinh trong ngan xep

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

pop(&Open\_DFS);

push(node,&Close\_DFS);

//Kiem tra dinh lay ra co phai la trang thai muc tieu khong

if(goalcheck(node->state))

return node;

int opt;

//Goi cac phep toan tren trang thai

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

State newstate;

makeNullState(&newstate);

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

//Neu trang thai moi da ton tai thi bo qua

if(find\_State(newstate,Close\_DFS)||find\_State(newstate,Open\_DFS))

continue;

//Neu trang thai moi chua ton tai thi them vao ngan xep

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

newNode->state=newstate;

newNode->Parent=node;

newNode->no\_function=opt;

push(newNode,&Open\_DFS);

}

}

}

return NULL;

}

//In ket qua

void print\_WaysToGetGoal(Node\* node){

Stack stackPrint;

makeNull\_Stack(&stackPrint);

//Duyet nguoc ve nut cha

while(node->Parent!=NULL){

push(node,&stackPrint);

node=node->Parent;

}

push(node,&stackPrint);

//In ra thu tu hanh dong

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++;

}

}

//main

int main(){

State cur\_state = {0,0};

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

print\_WaysToGetGoal(p);

return 0;

}

**Đong nước BFS**

#include <stdio.h>

#include <stdlib.h>

#define Maxlength 100

#define goal 6

#define tankcapacity\_X 9

#define tankcapacity\_Y 4

#define empty 0

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

typedef struct{

int x;

int y;

}State;

void makenullState(State \*S){

S->x =0;

S->y =0;

}

void print\_State(State S){

printf("\nX : %d --- Y : %d", S.x, S.y);

}

int checkgoal(State S){

return S.x == goal || S.y == goal;

}

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;

}

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

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

result->x = cur\_state.y;

result->y = tankcapacity\_Y;

return 1;

}

return 0;

}

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

if(cur\_state.x>0){

result->x =0;

result->y = cur\_state.y;

return 1;

}

return 0;

}

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

if(cur\_state.y>0){

result->x =cur\_state.x;

result->y = 0;

return 1;

}

return 0;

}

int max(int a, int b){

return a>b?a:b;

}

int min(int a, int b){

return a<b?a:b;

}

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

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

result->x = max(cur\_state.x - (tankcapacity\_Y-cur\_state.y) ,0);

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

return 1;

}

return 0;

}

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

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

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

result->y = max(0, cur\_state.y - (tankcapacity\_X- cur\_state.x));

return 1;

}

return 0;

}

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

switch(opt){

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

break;

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

break;

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

break;

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

break;

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

break;

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

default:printf("Error call operators.");

return 0;

}

}

//khai bao Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_funtion;

}Node;

//Khai bao hang doi

typedef struct{

Node\* Element[Maxlength];

int front, rear;

}Queue;

//Khoi tao hang doi rong

void makenullQueue(Queue \*Q){

Q->front = -1;

Q->rear = -1;

}

//Kiem tra hang doi co day khong

int isFullQueue(Queue Q){

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

}

//Kiem tra hang doi co rong khong

int isEmptyQueue(Queue Q){

return Q.front == -1;

}

//Them phan tu

void push(Queue \*Q, Node\* node){

if(!isFullQueue(\*Q)){

if(isEmptyQueue(\*Q)){

Q->front=0;

}

Q->rear++;

Q->Element[Q->rear] = node;

}else{

printf("Queue is full.");

}

}

//Xoa phan tu

void pop(Queue \*Q){

if(!isEmptyQueue(\*Q)){

Q->front++;

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

makenullQueue(Q);

}else{

printf("Queue is empty.");

}

}

//Lay ra 1 phan tu

Node\* front(Queue Q){

return Q.Element[Q.front];

}

int compareStates(State s1, State s2){

if(s1.x==s2.x &&s1.y==s2.y)

return 1;

return 0;

}

//Tim kiem trang thai trong Stack Open/Close

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

while(!isEmptyQueue(openQueue)){

if(compareStates(front(openQueue)->state,state ))

return 1;

else

pop(&openQueue);

}

return 0;

}

//BFS\_Algorithm

Node\* BFS\_Algorithm(State state){

Queue open\_BFS;

Queue close\_BFS;

makenullQueue(&open\_BFS);

makenullQueue(&close\_BFS);

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

root->no\_funtion = 0;

root->Parent = NULL;

root->state = state;

push(&open\_BFS, root);

while(!isEmptyQueue(open\_BFS)){

Node\* node = front(open\_BFS);

pop(&open\_BFS);

push(&close\_BFS, node);

if(checkgoal(node->state))

return node;

int opt ;

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

State newState;

makenullState(&newState);

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

if(find\_State(newState, open\_BFS)|| find\_State(newState, close\_BFS))

continue;

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

newNode->state = newState;

newNode->Parent = node;

newNode->no\_funtion = opt;

push(&open\_BFS, newNode);

}

}

}

}

//In ket qua

void print\_WaytoGetGoal(Node\* node){

Queue Q;

makenullQueue(&Q);

while(node->Parent!=NULL){

push(&Q, node);

node = node->Parent;

}

push(&Q, node);

int i;

for(i = Q.rear; i>= Q.front; i--){

printf("\nAction %d : %s", Q.rear-i, action[Q.Element[i]->no\_funtion]);

print\_State(Q.Element[i]->state);

}

}

//main

int main(){

State cur\_state = {0,0};

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

print\_WaytoGetGoal(p);

return 0;

}

**Trượt ô số Best first search**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <algorithm>

using namespace std;

#define ROWS 3

#define COLS 3

#define EMPTY 0

#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"

};

//Khai bao cau truc trang thai Puzzel

typedef struct {

int eightPuzzel[ROWS][COLS];

int emptyRow;

int emptyCol;

}State;

//In trang thai

void printState(State state){

printf("\n");

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

for(int j =0; j<COLS ; j++){

printf("|%d ", state.eightPuzzel[i][j]);

}

printf("|\n");

}

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

}

//compareStates

int compareStates(State s1, State s2){

if(s1.emptyCol != s2.emptyCol || s1.emptyRow!=s2.emptyRow)

return 0;

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

for(int j =0; j<COLS; j++){

if(s1.eightPuzzel[i][j]!= s2.eightPuzzel[i][j])

return 0;

}

}

return 1;

}

//Kiem tra muc tieu

int goalCheck(State state, State goal){

return compareStates(state, goal);

}

//Di chuyen len

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyRowCurrent>0){

result->emptyRow = emptyRowCurrent-1;

result->emptyCol = emptyColCurrent;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]= state.eightPuzzel[emptyRowCurrent-1][emptyColCurrent];

result->eightPuzzel[emptyRowCurrent-1][emptyColCurrent] = EMPTY;

return 1;

}

return 0;

}

//Di chuyen xuong

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyRowCurrent<2){

result->emptyRow = emptyRowCurrent+1;

result->emptyCol = emptyColCurrent;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent+1][emptyColCurrent];

result->eightPuzzel[emptyRowCurrent+1][emptyColCurrent]= EMPTY;

return 1;

}

return 0;

}

//Di chuyen sang trai

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyColCurrent>0){

result->emptyRow = emptyRowCurrent;

result->emptyCol = emptyColCurrent-1;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent][emptyColCurrent-1];

result->eightPuzzel[emptyRowCurrent][emptyColCurrent-1]= EMPTY;

return 1;

}

return 0;

}

//Di chuyen sang phai

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyColCurrent<2){

result->emptyRow = emptyRowCurrent;

result->emptyCol = emptyColCurrent+1;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent][emptyColCurrent+1];

result->eightPuzzel[emptyRowCurrent][emptyColCurrent+1]= EMPTY;

return 1;

}

return 0;

}

//Goi cac phep toan

int call\_Operators(State state, State \*result , int opt){

switch (opt){

case 1:return UP(state,result);

case 2:return DOWN(state, result);

case 3:return LEFT(state, result);

case 4:return RIGHT(state, result);

default: printf("\nCannot call operators");

return 0;

}

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_funtion;

int heuristic;

}Node;

//kiem tra trang thai State do co trong close/open khong

Node\* find\_State(State state, vector<Node\*> v, vector<Node\*>::iterator \*position){

vector<Node\*>::iterator it = v.begin();

if(v.size()==0)

return NULL;

while(it!=v.end()){

if(compareStates((\*it)->state, state)){

\*position = it;

return \*it;

}

it= v.erase(it);

}

return NULL;

}

//Heuristic

int heuristic(State state, State goal){

int count =0;

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

for(int j =0; j<COLS ; j++){

if(state.eightPuzzel[i][j]!=goal.eightPuzzel[i][j])

count++;

}

}

return count;

}

//CompareHeuristic

bool compareHeuristic(Node\* a, Node\* b){

return a->heuristic >b->heuristic;

}

//In ket qua

void print\_WaystoGetGoal(Node\* node){

vector<Node\*> listPrint;

listPrint.clear();

while(node->Parent!=NULL){

listPrint.push\_back(node);

node = node->Parent;

}

listPrint.push\_back(node);

int no\_action = 0;

for(int i = listPrint.size()-1; i>=0; i--){

printf("\nAction : %d : %s", no\_action, action[listPrint.at(i)->no\_funtion]);

printState(listPrint.at(i)->state);

no\_action++;

}

}

//Best first search

Node\* best\_first\_search(State state, State goal){

vector<Node\*> open\_BFS;

vector<Node\*> close\_BFS;

open\_BFS.clear();

close\_BFS.clear();

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

root->state = state;

root->Parent = NULL;

root->no\_funtion = 0;

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

open\_BFS.push\_back(root);

while(!open\_BFS.empty()){

Node\* node = open\_BFS.back();

open\_BFS.pop\_back();

close\_BFS.push\_back(node);

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

return node;

int opt;

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

State newState;

newState = node->state;

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

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

newNode->Parent = node;

newNode->state = newState;

newNode->no\_funtion=opt;

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

//kiem tra trang thai moi xin ra co thuoc Open/close

vector<Node\*>::iterator pos\_open,pos\_close;

Node\* nodeFoundOpen = find\_State(newState,open\_BFS,&pos\_open);

Node\* nodeFoundClose = find\_State(newState,close\_BFS,&pos\_close);

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

open\_BFS.push\_back(newNode);

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

open\_BFS.erase(pos\_open);

open\_BFS.push\_back(newNode);

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

close\_BFS.erase(pos\_close);

open\_BFS.push\_back(newNode);

}

}

}

sort(open\_BFS.begin(), open\_BFS.end(), compareHeuristic);

}

}

//Main

int main(){

State state;

state.emptyRow=1;

state.emptyCol =1;

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

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

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

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

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

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

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

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

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

State goal;

goal.emptyRow=0;

goal.emptyCol =0;

goal.eightPuzzel[0][0] =0;

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

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

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

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

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

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

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

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

Node\* p = best\_first\_search(state, goal);

print\_WaystoGetGoal(p);

return 0;

}

**Trượt ô số A Star search**

#include <stdio.h>

#include <stdlib.h>

#define ROWS 3

#define COLS 3

#define EMPTY 0

#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"

};

//Khai bao cau truc trang thai Puzzel

typedef struct {

int eightPuzzel[ROWS][COLS];

int emptyRow;

int emptyCol;

}State;

//In trang thai

void printState(State state){

printf("\n");

int i,j;

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

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

printf("|%d ", state.eightPuzzel[i][j]);

}

printf("|\n");

}

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

}

//CompareStates

int compareStates(State s1, State s2){

if(s1.emptyCol != s2.emptyCol || s1.emptyRow!=s2.emptyRow)

return 0;

int i, j;

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

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

if(s1.eightPuzzel[i][j]!= s2.eightPuzzel[i][j])

return 0;

}

}

return 1;

}

//Kiem tra muc tieu

int goalCheck(State state, State goal){

return compareStates(state, goal);

}

//Di chuyen len

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyRowCurrent>0){

result->emptyRow = emptyRowCurrent-1;

result->emptyCol = emptyColCurrent;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]= state.eightPuzzel[emptyRowCurrent-1][emptyColCurrent];

result->eightPuzzel[emptyRowCurrent-1][emptyColCurrent] = EMPTY;

return 1;

}

return 0;

}

//Di chuyen xuong

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyRowCurrent<2){

result->emptyRow = emptyRowCurrent+1;

result->emptyCol = emptyColCurrent;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent+1][emptyColCurrent];

result->eightPuzzel[emptyRowCurrent+1][emptyColCurrent]= EMPTY;

return 1;

}

return 0;

}

//Di chuyen sang trai

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyColCurrent>0){

result->emptyRow = emptyRowCurrent;

result->emptyCol = emptyColCurrent-1;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent][emptyColCurrent-1];

result->eightPuzzel[emptyRowCurrent][emptyColCurrent-1]= EMPTY;

return 1;

}

return 0;

}

//Di chuyen sang phai

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

\*result = state;

int emptyRowCurrent = state.emptyRow;

int emptyColCurrent = state.emptyCol;

if(emptyColCurrent<2){

result->emptyRow = emptyRowCurrent;

result->emptyCol = emptyColCurrent+1;

result->eightPuzzel[emptyRowCurrent][emptyColCurrent]=state.eightPuzzel[emptyRowCurrent][emptyColCurrent+1];

result->eightPuzzel[emptyRowCurrent][emptyColCurrent+1]= EMPTY;

return 1;

}

return 0;

}

//Goi cac phep toan

int call\_Operators(State state, State \*result , int opt){

switch (opt){

case 1:return UP(state,result);

case 2:return DOWN(state, result);

case 3:return LEFT(state, result);

case 4:return RIGHT(state, result);

default:printf("\nCannot call operators");

return 0;

}

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_funtion;

int h;

int g;

int f;

}Node;

//Khai bao cau truc List

typedef struct{

Node \*Elements[Maxlength];

int size;

}List;

//Khoi tao danh sach rong

void makenullList(List \*L){

L->size =0;

}

//Kiem tra danh sach co rong khong

int emptyList(List L){

return L.size==0;

}

//Kiem tra danh sach co day khong

int fullList(List L){

return L.size == Maxlength;

}

//truy van gia tri cua phan tu vitri p

Node\* element\_at(int p, List L){

return L.Elements[p-1];

}

//Them phan tu vao danh sach

void push\_List(Node\* x, int positon, List \*L){

if(!fullList(\*L)){

int q;

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

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

}

L->Elements[positon-1]=x;

L->size++;

}else

printf("List is full\n");

}

//xoa phan tu tai vi tri position ra khoi danh sach

void delete\_List(int position, List \*L){

if(emptyList(\*L))

printf("List is empty.\n");

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

printf("Position is not possible to delete.\n");

else{

int i;

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

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

}

L->size--;

}

}

//kiem tra trang thai State do co trong close/open khong

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

int i;

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

if(compareStates(element\_at(i,L)->state, state)){

\*position = i;

return element\_at(i, L);

}

}

return NULL;

}

//sap xep danh sach

void sort\_List(List \*L){

int i, j;

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

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

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

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

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

L->Elements[j]= node;

}

}

}

}

//Heuristic

int heuristic(State state, State goal){

int count =0;

int i, j;

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

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

if(state.eightPuzzel[i][j]!=goal.eightPuzzel[i][j])

count++;

}

}

return count;

}

//In ket qua

void print\_WaystoGetGoal(Node\* node){

List listPrint;

makenullList(&listPrint);

while(node->Parent!=NULL){

push\_List(node, listPrint.size+1, &listPrint );

node = node->Parent;

}

push\_List(node, listPrint.size+1, &listPrint );

int no\_action = 0;

int i;

for( i = listPrint.size; i>0; i--){

printf("\nAction : %d : %s", no\_action, action[element\_at(i, listPrint)->no\_funtion]);

printState(element\_at(i, listPrint)->state);

no\_action++;

}

}

//A Star search

Node\* A\_Star\_search(State state, State goal){

List open\_BFS, close\_BFS;

makenullList(&open\_BFS);

makenullList(&close\_BFS);

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

root->state = state;

root->Parent = NULL;

root->no\_funtion = 0;

root->g =0;

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

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

push\_List(root, open\_BFS.size+1, &open\_BFS);

while(!emptyList(open\_BFS)){

Node\* node = element\_at(1, open\_BFS);

delete\_List(1, &open\_BFS);

push\_List(node, close\_BFS.size+1, &close\_BFS);

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

return node;

int opt;

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

State newState;

newState = node->state;

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

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

newNode->Parent = node;

newNode->state = newState;

newNode->no\_funtion=opt;

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

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

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

//kiem tra trang thai moi xin ra co thuoc Open/close

int pos\_open,pos\_close;

Node\* nodeFoundOpen = find\_State(newState,open\_BFS,&pos\_open);

Node\* nodeFoundClose = find\_State(newState,close\_BFS,&pos\_close);

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

push\_List(newNode, open\_BFS.size+1, &open\_BFS);

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

delete\_List(pos\_open, &open\_BFS);

push\_List(newNode,open\_BFS.size+1 , &open\_BFS);

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

delete\_List(pos\_close, &close\_BFS);

push\_List(newNode, open\_BFS.size+1, &open\_BFS);

}

}

}

sort\_List(&open\_BFS);

}

}

//Main

int main(){

State state;

state.emptyRow=1;

state.emptyCol=1;

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

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

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

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

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

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

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

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

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

State goal;

goal.emptyRow=1;

goal.emptyCol=0;

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

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

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

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

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

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

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

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

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

Node\* p = A\_Star\_search(state, goal);

print\_WaystoGetGoal(p);

return 0;

}

**Tu si DFS**

#include<iostream>

#include<queue>

#include<cstdlib>

#include<stack>

using namespace std;

#define Maxlength 1000

#define tankcapacity\_Monk 3

#define tankcapacity\_Devil 3

const char\* action[]={"First State","Move 1 Monk","Move 1 Devil","Move 2 Monk","Move 2 Devil","Move 1 Monk and 1 Devil"};

typedef struct{

int Monk;

int Devil;

int pos\_boat; //0:thuyen ben A, 1:thuyen ben B

}State;

//Khoi tao trang thai ban dau

void makeNullState(State \*state){

state->Monk=3;

state->Devil=3;

state->pos\_boat=0;

}

//Kiem tra muc tieu

int checkgoal(State state){

return(state.Monk==0 && state.Devil==0 && state.pos\_boat==1);

}

//Kiem tra dead

int checkdead(State state){

if((state.Monk>0 && state.Monk<state.Devil)||(tankcapacity\_Monk-state.Monk>0 && tankcapacity\_Monk-state.Monk<tankcapacity\_Devil-state.Devil))

return 1;

return 0;

}

//In trang thai

void printState(State state){

printf("\n Monk: %d --- Devil: %d --- Ben bo: ",state.Monk,state.Devil);

if(state.pos\_boat==0){

printf("A");

}

else{

printf("B");

}

}

//Di chuyen 1 Monk

int Move1Monk(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Monk>0){

result->Monk=cur\_state.Monk-1;

result->Devil=cur\_state.Devil;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Monk<3){

result->Monk=cur\_state.Monk+1;

result->Devil=cur\_state.Devil;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 1 Devil

int Move1Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Devil>0){

result->Devil=cur\_state.Devil-1;

result->Monk=cur\_state.Monk;

result->pos\_boat=1;

return 1;

}

}

else { //ben bo B

if(cur\_state.Devil<3){

result->Devil=cur\_state.Devil+1;

result->Monk=cur\_state.Monk;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 2 Monk

int Move2Monk(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Monk>=2){

result->Monk=cur\_state.Monk-2;

result->Devil=cur\_state.Devil;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Monk<=1){

result->Monk=cur\_state.Monk+2;

result->Devil=cur\_state.Devil;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 2 Devil

int Move2Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Devil>=2){

result->Devil=cur\_state.Devil-2;

result->Monk=cur\_state.Monk;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Devil<=1){

result->Devil=cur\_state.Devil+2;

result->Monk=cur\_state.Monk;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 1 Monk 1 Devil

int Move1Monk1Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo B

if(cur\_state.Monk>0 && cur\_state.Devil>0){

result->Monk=cur\_state.Monk-1;

result->Devil=cur\_state.Devil-1;

result->pos\_boat=1;

return 1;

}

}

else{

if(cur\_state.Monk<3 && cur\_state.Devil<3){

result->Monk=cur\_state.Monk+1;

result->Devil=cur\_state.Devil+1;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Goi cac phep toan tren trang thai

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

switch(option){

case 1:return Move1Monk(cur\_state, result);

case 2:return Move1Devil(cur\_state, result);

case 3:return Move2Monk(cur\_state, result);

case 4:return Move2Devil(cur\_state, result);

case 5:return Move1Monk1Devil(cur\_state, result);

default:printf("Error calls operator\n");

return 0;

}

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_function;

}Node;

//compareStates

int compareStates(State s1, State s2){

if(s1.Monk==s2.Monk && s1.Devil==s2.Devil && s1.pos\_boat==s2.pos\_boat)

return 1;

return 0;

}

//Tim kiem trang thai trong Stack Open/Close

int find\_State(State state, stack<Node\*> Q){

while(!Q.empty()){

if(compareStates(Q.top()->state,state))

return 1;

Q.pop();

}

return 0;

}

//DFS\_Algorithm

Node\* DFS\_Algorithm(State state){

stack<Node\*> open\_DFS;

stack<Node\*> close\_DFS;

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

root->state=state;

root->Parent=NULL;

root->no\_function=0;

open\_DFS.push(root);

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

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

open\_DFS.pop();

close\_DFS.push(node);

if(checkgoal(node->state))

return node;

int opt;

for(opt=1; opt<=5; opt++){

State newState;

makeNullState(&newState);

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

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

continue;

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

newNode->state=newState;

newNode->Parent=node;

newNode->no\_function=opt;

open\_DFS.push(newNode);

}

}

}

}

//In ket qua

void print\_WaysToGetGoal(Node\* node){

stack<Node\*> S;

while(node->Parent!=NULL){

S.push(node);

node=node->Parent;

}

S.push(node);

int no\_action=0;

while(!S.empty()){

cout << "\n Action "<< no\_action << ":"<< action[S.top()->no\_function];

printState(S.top()->state);

S.pop();

no\_action++;

}

}

//Main

int main(){

State S;

makeNullState(&S);

Node\* d=DFS\_Algorithm(S);

print\_WaysToGetGoal(d);

return 0;

}

**Tu si BFS**

#include<iostream>

#include<queue>

#include<cstdlib>

#include<stack>

using namespace std;

#define Maxlength 1000;

#define tankcapacity\_Monk 3

#define tankcapacity\_Devil 3

const char\* action[]={"First State","Move 1 Monk","Move 1 Devil","Move 2 Monk","Move 2 Devil","Move 1 Monk and 1 Devil"};

typedef struct{

int Monk;

int Devil;

int pos\_boat; //0:thuyen ben A, 1:thuyen ben B

}State;

//Khoi tao trang thai ban dau

void makeNullState(State \*state){

state->Monk=3;

state->Devil=3;

state->pos\_boat=0;

}

//Kiem tra muc tieu

int checkgoal(State state){

return(state.Monk==0 && state.Devil==0 && state.pos\_boat==1);

}

//Kiem tra dead

int checkdead(State state){

if((state.Monk>0 && state.Monk<state.Devil)||(tankcapacity\_Monk-state.Monk>0 && tankcapacity\_Monk-state.Monk<tankcapacity\_Devil-state.Devil))

return 1;

return 0;

}

//In trang thai

void printState(State state){

printf("\n Monk: %d --- Devil: %d --- Ben bo: ",state.Monk,state.Devil);

if(state.pos\_boat==0){

printf("A");

}

else{

printf("B");

}

}

//Di chuyen 1 Monk

int Move1Monk(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Monk>0){

result->Monk=cur\_state.Monk-1;

result->Devil=cur\_state.Devil;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Monk<3){

result->Monk=cur\_state.Monk+1;

result->Devil=cur\_state.Devil;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 1 Devil

int Move1Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Devil>0){

result->Devil=cur\_state.Devil-1;

result->Monk=cur\_state.Monk;

result->pos\_boat=1;

return 1;

}

}

else { //ben bo B

if(cur\_state.Devil<3){

result->Devil=cur\_state.Devil+1;

result->Monk=cur\_state.Monk;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 2 Monk

int Move2Monk(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Monk>=2){

result->Monk=cur\_state.Monk-2;

result->Devil=cur\_state.Devil;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Monk<=1){

result->Monk=cur\_state.Monk+2;

result->Devil=cur\_state.Devil;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 2 Devil

int Move2Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo A

if(cur\_state.Devil>=2){

result->Devil=cur\_state.Devil-2;

result->Monk=cur\_state.Monk;

result->pos\_boat=1;

return 1;

}

}

else{ //ben bo B

if(cur\_state.Devil<=1){

result->Devil=cur\_state.Devil+2;

result->Monk=cur\_state.Monk;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Di chuyen 1 Monk 1 Devil

int Move1Monk1Devil(State cur\_state, State \*result){

if(cur\_state.pos\_boat==0){ //ben bo B

if(cur\_state.Monk>0 && cur\_state.Devil>0){

result->Monk=cur\_state.Monk-1;

result->Devil=cur\_state.Devil-1;

result->pos\_boat=1;

return 1;

}

}

else{

if(cur\_state.Monk<3 && cur\_state.Devil<3){

result->Monk=cur\_state.Monk+1;

result->Devil=cur\_state.Devil+1;

result->pos\_boat=0;

return 1;

}

}

return 0;

}

//Goi cac phep toan tren trang thai

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

switch(option){

case 1:return Move1Monk(cur\_state, result);

case 2:return Move1Devil(cur\_state, result);

case 3:return Move2Monk(cur\_state, result);

case 4:return Move2Devil(cur\_state, result);

case 5:return Move1Monk1Devil(cur\_state, result);

default:printf("Error calls operator\n");

return 0;

}

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_function;

}Node;

//compareStates

int compareStates(State s1, State s2){

if(s1.Monk==s2.Monk && s1.Devil==s2.Devil && s1.pos\_boat==s2.pos\_boat)

return 1;

return 0;

}

//Tim kiem trang thai trong Queue open/close

int find\_State(State state, queue<Node\*> Q){

while(!Q.empty()){

if(compareStates(Q.front()->state,state))

return 1;

Q.pop();

}

return 0;

}

//BFS\_Algorithm

Node\* BFS\_Algorithm(State state){

queue<Node\*> open\_BFS;

queue<Node\*> close\_BFS;

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

root->state=state;

root->Parent=NULL;

root->no\_function=0;

open\_BFS.push(root);

while(!open\_BFS.empty()){

Node\* node=open\_BFS.front();

open\_BFS.pop();

close\_BFS.push(node);

if(checkgoal(node->state))

return node;

int opt;

for(opt=1; opt<=5; opt++){

State newState;

makeNullState(&newState);

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

if(find\_State(newState,open\_BFS)||find\_State(newState,close\_BFS)||checkdead(newState))

continue;

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

newNode->state=newState;

newNode->Parent=node;

newNode->no\_function=opt;

open\_BFS.push(newNode);

}

}

}

}

//In ket qua

void print\_WaysToGetGoal(Node\* node){

queue<Node\*> Q;

while(node->Parent!=NULL){

Q.push(node);

node=node->Parent;

}

Q.push(node);

stack<Node\*> S;

while(!Q.empty()){

S.push(Q.front());

Q.pop();

}

int no\_action=0;

while(!S.empty()){

cout << "\n Action" << no\_action << ":" << action[S.top()->no\_function];

printState(S.top()->state);

S.pop();

no\_action++;

}

}

//Main

int main(){

State S;

makeNullState(&S);

Node\* d=BFS\_Algorithm(S);

print\_WaysToGetGoal(d);

return 0;

}

**Úp ly DFS**

#include<iostream>

#include<stack>

#include<queue>

#include<cstdlib>

using namespace std;

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

#define NumGlasses 6

#define Up 1

#define Down -1

//Khai bao cau truc trang thai

typedef struct{

int Glasses[NumGlasses];

}State;

//Khoi tao khong gian rong

void makeNullState(State \*state){

for(int i=0; i<NumGlasses; i++){

if(state->Glasses[i]%2==0)

state->Glasses[i]=Up;

else

state->Glasses[i]=Down;

}

}

//In trang thai

void printState(State S){

cout<<"\n";

for(int i=0; i<NumGlasses; i++){

if(S.Glasses[i]==Up)

cout<< "U ";

else

cout<< "D ";

}

}

//Kiem tra muc tieu

int goalcheck(State S){

for(int i=0; i<NumGlasses; i++){

if(S.Glasses[i]==Down)

return 0;

}

return 1;

}

//Up ly

void flipGlasses(State cur\_state, State \*result, int k){

for(int i=0; i<NumGlasses; i++){

result->Glasses[i]=cur\_state.Glasses[i];

}

result->Glasses[k-1]=-cur\_state.Glasses[k-1];

result->Glasses[k]=-cur\_state.Glasses[k];

result->Glasses[k+1]=-cur\_state.Glasses[k+1];

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_function;

}Node;

//compareStates

int compareStates(State s1, State s2){

int i;

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

if(s1.Glasses[i]!=s2.Glasses[i])

return 0;

}

return 1;

}

//Tim kiem trang thai trong Stack open/close

int findState(State state, stack<Node\*> open){

while(!open.empty()){

if(compareStates(open.top()->state,state))

return 1;

open.pop();

}

return 0;

}

//DFS\_Algorithm

Node\* DFS\_Algorithm(State state){

stack<Node\*> open\_DFS;

stack<Node\*> close\_DFS;

//Tao nut trang thai cha

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

root->state=state;

root->Parent=NULL;

root->no\_function=0;

open\_DFS.push(root);

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

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

open\_DFS.pop();

close\_DFS.push(node);

if(goalcheck(node->state))

return node;

int opt;

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

State newState;

makeNullState(&newState);

flipGlasses(node->state,&newState,opt);

if(findState(newState,open\_DFS)||findState(newState,close\_DFS))

continue;

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

newNode->state=newState;

newNode->Parent=node;

newNode->no\_function=opt;

open\_DFS.push(newNode);

}

}

return NULL;

}

//In ket qua

void print\_WaysToGetGoal(Node\* node){

stack<Node\*> S;

while(node->Parent!=NULL){

S.push(node);

node=node->Parent;

}

S.push(node);

int no\_action=0;

while(!S.empty()){

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

printState(S.top()->state);

S.pop();

no\_action++;

}

}

//Main

int main(){

State cur\_state={Up,Down,Up,Down,Up,Down};

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

print\_WaysToGetGoal(p);

return 0;

}

**Úp ly BFS**

#include<iostream>

#include<stack>

#include<queue>

#include<cstdlib>

using namespace std;

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

#define NumGlasses 6

#define Up 1

#define Down -1

//Khai bao cau truc trang thai

typedef struct{

int Glasses[NumGlasses];

}State;

//Khoi tao khong gian rong

void makeNullState(State \*state){

for(int i=0; i<NumGlasses; i++){

if(state->Glasses[i]%2==0)

state->Glasses[i]=Up;

else

state->Glasses[i]=Down;

}

}

//In trang thai

void printState(State S){

cout<<"\n";

for(int i=0; i<NumGlasses; i++){

if(S.Glasses[i]==Up)

cout<< "U ";

else

cout<< "D ";

}

}

//Kiem tra muc tieu

int goalcheck(State S){

for(int i=0; i<NumGlasses; i++){

if(S.Glasses[i]==Down)

return 0;

}

return 1;

}

//Up ly

void flipGlasses(State cur\_state, State \*result, int k){

for(int i=0; i<NumGlasses; i++){

result->Glasses[i]=cur\_state.Glasses[i];

}

result->Glasses[k-1]=-cur\_state.Glasses[k-1];

result->Glasses[k]=-cur\_state.Glasses[k];

result->Glasses[k+1]=-cur\_state.Glasses[k+1];

}

//Khai bao cau truc Node

typedef struct Node{

State state;

struct Node\* Parent;

int no\_function;

}Node;

//compareStates

int compareStates(State s1, State s2){

int i;

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

if(s1.Glasses[i]!=s2.Glasses[i])

return 0;

}

return 1;

}

//Tim kiem trang thai trong Queue open/close

int findState(State state, queue<Node\*> open){

while(!open.empty()){

if(compareStates(open.front()->state,state))

return 1;

open.pop();

}

return 0;

}

//BFS\_Algorithm

Node\* BFS\_Algorithm(State state){

queue<Node\*> open\_BFS;

queue<Node\*> close\_BFS;

//Tao nut trang thai cha

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

root->state=state;

root->Parent=NULL;

root->no\_function=0;

open\_BFS.push(root);

while(!open\_BFS.empty()){

Node\* node=open\_BFS.front();

open\_BFS.pop();

close\_BFS.push(node);

if(goalcheck(node->state))

return node;

int opt;

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

State newState;

makeNullState(&newState);

flipGlasses(node->state,&newState,opt);

if(findState(newState,open\_BFS)||findState(newState,close\_BFS))

continue;

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

newNode->state=newState;

newNode->Parent=node;

newNode->no\_function=opt;

open\_BFS.push(newNode);

}

}

return NULL;

}

//In ket qua

void print\_WaysToGetGoal(Node\* node){

stack<Node\*> S;

while(node->Parent!=NULL){

S.push(node);

node=node->Parent;

}

S.push(node);

int no\_action=0;

while(!S.empty()){

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

printState(S.top()->state);

S.pop();

no\_action++;

}

}

//Main

int main(){

State cur\_state={Up,Down,Up,Down,Up,Down};

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

print\_WaysToGetGoal(p);

return 0;

}

**Sudoku**

#include <stdio.h>

#include <stdlib.h>

#define Maxlength 500

#define Max\_Value 10

#define EMPTY 0

#define AREA\_SQUARE\_SIZE 3

#define INF 9999999

//Khai bao cau truc Coord

typedef struct {

int x, y;

}Coord;

//Khai bao cau truc ListCoord

typedef struct{

Coord data[Maxlength];

int size;

}ListCoord;

//initListCoord

void initListCoord(ListCoord \*L){

L->size=0;

}

//appendListCoord

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

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

}

//cau truc rang buoc

#define NB\_ROWS 9

#define NB\_COLS 9

//Khai bao cau truc Constrains

typedef struct {

int data[NB\_ROWS\*NB\_COLS][NB\_ROWS\*NB\_COLS];

int n;

}Constrains;

//Khoi tao rang buoc rong

void initConstrains(Constrains \*constrains){

for(int i =0; i<NB\_ROWS\*NB\_COLS; i++){

for(int j =0; j<NB\_ROWS\*NB\_COLS; j++){

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

}

}

constrains->n= NB\_ROWS\*NB\_COLS;

}

//indexOf

int indexOf(Coord coord){

return (NB\_ROWS\*coord.x+coord.y);

}

//positionOfVertex

Coord positionOfVertex(int vertex){

Coord coord;

coord.x = vertex/NB\_ROWS;

coord.y = vertex% NB\_COLS;

return coord;

}

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

int u = indexOf(source);

int v = indexOf(target);

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

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

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

return 1;

}

return 0;

}

//getContrains

ListCoord getConstrains(Constrains constrains, Coord coord){

int v = indexOf(coord);

ListCoord result;

initListCoord(&result);

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

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

appendListCoord(&result, positionOfVertex(i));

}

}

return result;

}

//Khai bao cau truc Sodoku

typedef struct {

int cells[NB\_ROWS][NB\_COLS];

Constrains constrains;

}Sudoku;

//khoi tao bang Sudoku

void initSudoku(Sudoku \*sudoku){

for(int i = 0; i<NB\_ROWS; i++){

for(int j =0; j<NB\_COLS; j++){

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

}

}

initConstrains(&sudoku->constrains);

}

//initSudokuWithValues

void initSudokuWithValues(Sudoku\* sudoku, int inputs[NB\_ROWS][NB\_COLS]){

for(int i =0; i<NB\_ROWS;i++){

for(int j =0; j<NB\_COLS;j++){

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

}

}

initConstrains(&sudoku->constrains);

}

//In Sudoku

void printSudoku(Sudoku sudoku){

printf("Sudoku: \n");

for(int i =0; i<NB\_ROWS;i++){

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

for(int j =0; j<NB\_COLS;j++){

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

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

}

printf("|\n");

}

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

}

//isFilledSudoku

int isFilledSudoku(Sudoku sudoku){

for(int i =0; i<NB\_ROWS;i++){

for(int j =0; j<NB\_COLS;j++){

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

return 0;

}

}

return 1;

}

//spreadContrainsFrom

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

int row = position.x, column = position.y;

//tao rang buoc theo cot

for(int i =0; i<NB\_ROWS;i++){

if(i!=row){

Coord pos = {i, column};

if(addConstrain(constrains, position, pos))

appendListCoord(changeds, pos);

}

}

//tao rang buoc theo hang

for(int i =0; i<NB\_COLS;i++){

if(i!=column){

Coord pos = {row, i};

if(addConstrain(constrains, position, pos))

appendListCoord(changeds, pos);

}

}

//truyen rang buoc theo khoi

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

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

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

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

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

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

if(addConstrain(constrains, position, pos))

appendListCoord(changeds, pos);

}

}

}

}

//Khai bao list

typedef struct{

int Elements[Maxlength];

int size;

}List;

//Khoi tao danh sach rong

void makenullList(List \*L){

L->size =0;

}

//Kiem tra danh sach co rong khong

int emptyList(List L){

return L.size==0;

}

//Kiem tra danh sach co day khong

int fullList(List L){

return L.size == Maxlength;

}

//truy van gia tri cua phan tu vitri p

int element\_at(int p, List L){

return L.Elements[p-1];

}

//Them phan tu vao danh sach

void push\_List(int x, int positon, List \*L){

if(!fullList(\*L)){

int q;

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

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

}

L->Elements[positon-1]=x;

L->size++;

}else

printf("List is full\n");

}

//xoa phan tu tai vi tri position ra khoi danh sach

void delete\_List(int position, List \*L){

if(emptyList(\*L))

printf("List is empty.\n");

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

printf("Position is not possible to delete.\n");

else{

int i;

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

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

}

L->size--;

}

}

//appendList

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

push\_List(x,L->size+1,L);

}

//getAvailableValues

List getAvailableValues(Coord position, Sudoku sudoku){

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

int available[Max\_Value];

for(int i =1; i<Max\_Value;i++) available[i]=1;

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

Coord pos = posList.data[i];

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

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

}

}

List result;

makenullList(&result);

for(int i =1; i<Max\_Value;i++){

if(available[i]) appendList(&result, i);

}

return result;

}

//Xac dinh o nao uu tien dien truoc

//Cach 2

Coord getNextMinDomainCell(Sudoku sudoku){

int minLength = INF;

Coord result;;

for(int i=0; i<NB\_ROWS;i++){

for(int j =0; j<NB\_COLS;j++){

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

Coord pos = {i,j};

int availablesLength = getAvailableValues(pos, sudoku).size;

if(availablesLength<minLength){

minLength = availablesLength;

result = pos;

}

}

}

}

return result;

}

int exploredCounter =0;

int sudokuBackTracking(Sudoku \*sudoku){

if(isFilledSudoku(\*sudoku)) return 1;

Coord position = getNextMinDomainCell(\*sudoku);

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

if(availables.size ==0)

return 0;

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

int value = availables.Elements[j];

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

exploredCounter++;

if(sudokuBackTracking(sudoku))

return 1;

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

}

return 0;

}

//solveSudoku

Sudoku solveSudoku(Sudoku sudoku){

initConstrains(&sudoku.constrains);

for(int i =0; i<NB\_ROWS;i++){

for(int j =0; j<NB\_COLS;j++){

ListCoord history;

initListCoord(&history);

Coord pos = {i, j};

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

}

}

exploredCounter =0;

if(sudokuBackTracking(&sudoku)) printf("Solved\n");

else printf("Can not solve\n");

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

return sudoku;

}

int inputs1[9][9]={

{5,3,0,0,7,0,0,0,0},

{6,0,0,1,9,5,0,0,0},

{0,9,8,0,0,0,0,6,0},

{8,0,0,0,6,0,0,0,3},

{4,0,0,8,0,3,0,0,1},

{7,0,0,0,2,0,0,0,6},

{0,6,0,0,0,0,2,8,0},

{0,0,0,4,1,9,0,0,5},

{0,0,0,0,8,0,0,7,9},

};

//Main

int main(){

Sudoku sudoku;

initSudokuWithValues(&sudoku, inputs1);

printSudoku(sudoku);

Sudoku result = solveSudoku(sudoku);

printSudoku(result);

return 0;

}

**KenKen**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_LENGTH 100

#define NB\_ROWS 4

#define NB\_COLUMNS 4

#define MAX\_VALUE 5

#define EMPTY 0

#define INF 999999

int NB\_BLOCK = 8;

//Khai bao cau truc List

typedef struct List {

int data[MAX\_LENGTH];

int size;

} List;

//Khoi tao danh sach rong

void makeNULLList(List \*list) {

list->size = 0;

}

//Kiem tra danh sach co rong khong

int emptyList(List list) {

return list.size == EMPTY;

}

//Kiem tra danh sach co day khong

int fullList(List list) {

return list.size == MAX\_LENGTH;

}

//ElementAt

int elementAt(List list, int p) {

return list.data[p - 1];

}

//Them vao danh sach

void pushList(List \*list, int n) {

list->data[list->size] = n;

list->size++;

}

//Khai bao cau truc Coord

typedef struct Coord {

int x, y;

}Coord;

//Khai bao cau truc ListCoord

typedef struct ListCoord {

Coord data[MAX\_LENGTH];

int size;

}ListCoord;

//initListCoord

void initListCoord(ListCoord \*list) {

list->size = 0;

}

//appendListCoord

void appendListCoord(ListCoord \*list, Coord coord) {

list->data[list->size++] = coord;

}

//Khai bao cau truc Constrains

typedef struct Constrains {

int data[NB\_ROWS \* NB\_COLUMNS][NB\_ROWS \* NB\_COLUMNS];

int n;

}Constrains;

//initConstrains

void initConstrains(Constrains \*constrains) {

int i, j;

for (i = 0; i < NB\_ROWS \* NB\_COLUMNS; i++)

for (j = 0; j < NB\_ROWS \* NB\_COLUMNS; j++)

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

constrains->n = NB\_ROWS \* NB\_COLUMNS;

}

//indexOf

int indexOf(Coord coord) {

return NB\_ROWS \* coord.x + coord.y;

}

//positionOfVertex

Coord positionOfVertex(int vertex) {

Coord coord;

coord.x = vertex / NB\_ROWS;

coord.y = vertex % NB\_COLUMNS;

return coord;

}

//addConstrains

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

int u = indexOf(source);

int v = indexOf(target);

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

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

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

return 1;

}

return 0;

}

//getConstrains

ListCoord getConstrains(Constrains constrains, Coord coord) {

int i;

int v = indexOf(coord);

ListCoord result;

initListCoord(&result);

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

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

appendListCoord(&result, positionOfVertex(i));

}

}

return result;

}

//Khai bao cau truc Block

typedef struct Block {

int value;

int currentValue;

char operators;

ListCoord list;

int filledCells;

}Block;

//initBLock

void initBlock(Block \*block, int sum, char c, ListCoord list) {

block->value = sum;

block->currentValue = 0;

block->filledCells = 0;

block->operators= c;

int i;

initListCoord(&block->list);

for(i = 0; i < list.size; i++)

appendListCoord(&block->list, list.data[i]);

}

//whichBLockAreYouIn

int whichBlockAreYouIn(Coord position, Block block[]) {

int i, j;

for (i = 0; i < NB\_BLOCK; i++) {

for (j = 0; j < block[i].list.size; j++) {

if (position.x == block[i].list.data[j].x && position.y == block[i].list.data[j].y)

return i;

}

}

}

//blockIsFilled

int blockIsFilled(Block block) {

return block.filledCells == block.list.size;

}

//Khai bao cau truc KenKen

typedef struct KenKen {

int cells[NB\_ROWS][NB\_COLUMNS];

Constrains constrains;

Block block[10];

}KenKen;

//initKenKen

void initKenKen(KenKen \*kenken) {

int i, j;

for (i = 0; i < NB\_ROWS; i++)

for (j = 0; j < NB\_COLUMNS; j++)

kenken->cells[i][j] = EMPTY;

initConstrains(&kenken->constrains);

}

//In KenKen

void printKenKen(KenKen kenken) {

int i, j;

printf("KenKen:\n");

for (i = 0; i < NB\_ROWS; i++) {

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

for (j = 0; j < NB\_COLUMNS; j++) {

printf("|%3d ", kenken.cells[i][j]);

}

printf("|\n");

}

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

printf("KenKen Operator:\n");

for (i = 0; i < NB\_ROWS; i++) {

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

for (j = 0; j < NB\_COLUMNS; j++) {

int k = whichBlockAreYouIn((Coord){i, j}, kenken.block);

if (kenken.block[k].operators== '.')

printf("|%3d ", kenken.block[k].value);

else

printf("|%3d%c ", kenken.block[k].value, kenken.block[k].operators);

}

printf("|\n");

}

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

}

//isFilledKenKen

int isFilledKenKen(KenKen kenken) {

int i, j;

for (i = 0; i < NB\_ROWS; i++)

for (j = 0; j < NB\_COLUMNS; j++)

if (kenken.cells[i][j] == EMPTY)

return 0;

return 1;

}

//spreadConstrainsForm

void spreadConstrainsForm(Coord position, Constrains \*constrains){

int row = position.x, column = position.y;

int i, j;

for (i = 0; i < NB\_ROWS; i++) {

if (i != row) {

Coord pos = {i, column};

addConstrain(constrains, position, pos);

}

}

for (i = 0; i < NB\_COLUMNS; i++) {

if (i != column) {

Coord pos = {row, i};

addConstrain(constrains, position, pos);

}

}

}

//resetConstrains

void resetConstrains(Constrains \*constrains, int cells[NB\_ROWS][NB\_COLUMNS]) {

int i, j;

for (i = 0; i < NB\_ROWS; i++)

for (j = 0; j < NB\_COLUMNS; j++)

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

for(i = 0; i < NB\_ROWS; i++)

for (j = 0; j < NB\_COLUMNS; j++)

if (cells[i][j] != 0)

spreadConstrainsForm((Coord){i, j}, constrains);

}

//getAvailableValues

List getAvailableValues(Coord position, KenKen kenken) {

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

int availables[MAX\_VALUE];

int i;

for (i = 1; i < MAX\_VALUE; i++)

availables[i] = 1;

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

Coord pos = posList.data[i];

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

availables[kenken.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;

}

//getNextEmptyCell

Coord getNextEmptyCell(KenKen kenken) {

int i, j;

for (i = 0; i < NB\_ROWS; i++) {

for (j = 0; j < NB\_COLUMNS; j++) {

Coord pos = {i, j};

if (kenken.cells[i][j] == EMPTY)

return pos;

}

}

}

int resultFromOperatorAndTwoValues(int a, int b, char c) {

if(c == '+')

return a + b;

if(c == '-')

if(a > b)

return a - b;

else

return b - a;

if(c == '/')

if(a > b)

return a / b;

else

return b / a;

if(c == '\*' || c == 'x')

return a \* b;

return b;

}

//getnextMinDomainCell

Coord getNextMinDomainCell(KenKen kenken){

int minLength = INF;

int i, j;

Coord result;

for (i = 0; i < NB\_ROWS; i++){

for (j = 0; j < NB\_COLUMNS; j++){

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

Coord pos = {i, j};

int availablesLength = getAvailableValues(pos, kenken).size;

if (availablesLength < minLength){

minLength = availablesLength;

result = pos;

}

}

}

}

return result;

}

int exploredCounter = 0;

int kenkenBackTracking(KenKen \*kenken) {

if (isFilledKenKen(\*kenken))

return 1;

Coord position = getNextEmptyCell(\*kenken);

// Coord position = getNextMinDomainCell(\*kenken);

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

if (availables.size == 0) {

return 0;

}

int j;

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

int value = availables.data[j];

int blockNB = whichBlockAreYouIn(position, kenken->block);

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

int currentValue = kenken->block[blockNB].currentValue;

if (kenken->block[blockNB].filledCells == 0) {

kenken->block[blockNB].filledCells++;

kenken->block[blockNB].currentValue += value;

spreadConstrainsForm(position, &kenken->constrains);

}else{

kenken->block[blockNB].filledCells++;

kenken->block[blockNB].currentValue = resultFromOperatorAndTwoValues(kenken->block[blockNB].currentValue, value, kenken->block[blockNB].operators);

spreadConstrainsForm(position, &kenken->constrains);

}

if(blockIsFilled(kenken->block[blockNB]) && (kenken->block[blockNB].currentValue != kenken->block[blockNB].value)) {

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

kenken->block[blockNB].filledCells--;

kenken->block[blockNB].currentValue = currentValue;

resetConstrains(&kenken->constrains, kenken->cells);

continue;

// return 0;

}

exploredCounter++;

if(kenkenBackTracking(kenken))

return 1;

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

kenken->block[blockNB].filledCells--;

kenken->block[blockNB].currentValue = currentValue;

resetConstrains(&kenken->constrains, kenken->cells);

}

return 0;

}

int solveKenKen(KenKen \*kenken) {

initConstrains(&kenken->constrains);

exploredCounter = 0;

return kenkenBackTracking(kenken);

}

//readKenKen

void readKenKen(KenKen \*kenken) {

int i;

printf("Number of Cage: ");

scanf("%d", &NB\_BLOCK);

for(i = 0; i < NB\_BLOCK; i++) {

ListCoord cage;

initListCoord(&cage);

int k, n, value;

// printf("Enter value of cage: ");

scanf("%d", &value);

char c;

// printf("Enter operator: ");

scanf(" %c", &c);

getchar();

// printf("Number of cells of cage%d: ", i + 1);

scanf("%d", &n);

for (k = 0; k < n; k++) {

// printf("Enter position(x,y): ");

int u, v;

scanf("%d%d", &u, &v);

appendListCoord(&cage, (Coord){u, v});

}

initBlock(&kenken->block[i], value, c, cage);

}

}

//Main

int main() {

KenKen kenken;

initKenKen(&kenken);

int i;

freopen("test4.txt", "r", stdin);

readKenKen(&kenken);

if (solveKenKen(&kenken)) {

printf("Solved\n");

printKenKen(kenken);

}else

printf("Can not solve\n");

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

}