TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG VÀ CÁC HỆ THỐNG THÔNG TIN

**BÁO CÁO XÂY DỰNG CHƯƠNG TRÌNH**

**HỌC PHẦN TRÍ TUỆ NHÂN TẠO**

**1. Họ và tên: Nguyễn Thị Ngũ**

**2. Mã sinh viên: 191200758**

**3. Lớp: CNTT4\_K60**

**4. Nội dung báo cáo**

**4.1. Kẻ cướp và triệu phú**

**- Đề bài:** Có 2 bờ A và B. Bờ A có 3 phú và 3 cướp. Áp dụng thuật toán BFS hoặc DFS để chuyển 3 phú và 3 cướp từ bờ A sang bờ B. Điều kiện: thuyền chở tối đa 2 người và số cướp không lớn hơn số phú tại mỗi bờ.

**- Mô tả thuật toán:**

+Tạo 1 đối tượng phu\_cuop gồm 3 thuộc tính là phu(lưu số phú), cuop(lưu số cướp), bo(thể hiện thuyền đang ở bờ nào).

+ Tập các trường hợp có thể xảy ra:

- Thuyền chở 1 phú ⬄ (1, 0, true) với bờ A và (1, 0, false) với bờ B

- Thuyền chở 1 cướp ⬄ (0, 1, true) với bờ A và (0, 1, false) với bờ B

- Thuyền chở 1 phú 1 cướp ⬄ (1, 1, true) với bờ A và (1, 1, false) với bờ B

- Thuyền chở 2 phú ⬄ (2, 0, true) với bờ A và (2, 0, false) với bờ B

- Thuyền chở 2 cướp ⬄ (0, 2, true) với bờ A và (0, 2, false) với bờ B

+ Không gian trạng thái: TTĐ(3, 3, 0); TTKT(0, 0, 0)

+ Bắt đầu thuật toán DFS:

- Tạo đối tượng bắt đầu và thêm vào 1 ngăn xếp Stack. Một Map để lưu đường đi.

- Khởi tạo vòng lặp cho đến khi stack rỗng hoặc đạt được trạng thái kết thúc. Mỗi vòng lặp ta thực hiện:

+ Kiểm tra xem thuyền đang ở bờ nào. Nếu bờ A sang bờ B ta thực hiện cộng thêm vào đối tượng các trường hợp có thể xảy. Ngược lại nếu từ bờ B sang bờ A ta thực hiện trừ.

+ Kiểm tra kết quả nhận được có hợp lý không? Nếu hợp lý thì ta thêm vào Stack và lưu điểm này vào Map với key là kết quả vừa nhận được và value là đối tượng đang được xét.

* Như vậy để tìm được đường đi ta sẽ từ TTKT truy ngược lại được TTĐ.

**- Kết quả cài đặt** (mô tả chương trình hoặc mã chương trình)

**\* Class Phu\_Cuop: để tạo đối tượng**

import java.util.Objects;  
public class PhuCuop {  
 private int phu;  
 private int cuop;  
 private boolean bo;  
  
 public PhuCuop() {  
  
 }  
 public PhuCuop(int phu, int cuop, boolean bo){  
 this.cuop = cuop;  
 this.phu = phu;  
 this.bo = bo;  
 }  
  
 public int getPhu() {  
 return phu;  
 }  
  
 public void setPhu(int phu) {  
 this.phu = phu;  
 }  
  
 public int getCuop() {  
 return cuop;  
 }  
  
 public void setCuop(int cuop) {  
 this.cuop = cuop;  
 }  
  
 public boolean isBo() {  
 return bo;  
 }  
  
 public void setBo(boolean bo) {  
 this.bo = bo;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + phu +  
 ", " + cuop +  
 ", " + (bo ? "Bờ A" : "Bờ B") + ")";  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 PhuCuop phuCuop = (PhuCuop) o;  
 return phu == phuCuop.phu && cuop == phuCuop.cuop && bo == phuCuop.bo;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(phu, cuop, bo);  
 }  
}

**\* Class BFS\_DFS: để triển khai thuật toán và test.**

import javafx.util.Pair;

import java.util.\*;

public class BFS\_DFS {

static Map<PhuCuop, PhuCuop> map = new HashMap<>();

static HashSet<PhuCuop> isTham = new HashSet<>();

static List<Pair<Integer, Integer>> c = new ArrayList<>();

public static int check(PhuCuop s){

if(isTham.contains(s)) return -1;

if(s.getCuop() == 0 && s.getPhu() == 0) return 0;

if((s.getPhu() <= 3) && (s.getCuop() <= 3) && (s.getPhu() >= 0) && (s.getCuop() >= 0)){

if(s.getPhu() == s.getCuop() || s.getPhu() == 0 || s.getPhu() == 3) return 1;

}

return -1;

}

public static boolean BFS(Queue<PhuCuop> queue) {

if(queue.isEmpty()) return false;

PhuCuop pc = queue.poll();

for (Pair<Integer, Integer> i : c) {

PhuCuop newPC;

if(pc.isBo()){

newPC = new PhuCuop(pc.getPhu() - i.getKey(), pc.getCuop() - i.getValue(), false);

}

else{

newPC = new PhuCuop(pc.getPhu() + i.getKey(), pc.getCuop() + i.getValue(), true);

}

int check = check(newPC);

if (check == 1) {

isTham.add(newPC);

queue.add(newPC);

map.put(newPC, pc);

}

else if(check== 0){

map.put(newPC, pc);

return true;

}

}

return BFS(queue);

}

public static boolean DFS(Stack<PhuCuop> stack){

if(stack.isEmpty()) return false;

PhuCuop pc = stack.peek();

for (Pair<Integer, Integer> i : c) {

PhuCuop newPC;

if(pc.isBo()){

newPC = new PhuCuop(pc.getPhu() - i.getKey(), pc.getCuop() - i.getValue(), false);

}

else{

newPC = new PhuCuop(pc.getPhu() + i.getKey(), pc.getCuop() + i.getValue(), true);

}

int check = check(newPC);

if (check == 1) {

isTham.add(newPC);

stack.add(newPC);

map.put(newPC, pc);

}

else if(check== 0){

map.put(newPC, pc);

return true;

}

}

return DFS(stack);

}

public static void runBFS(){

PhuCuop p = new PhuCuop(3, 3, true);

isTham.add(p);

Queue<PhuCuop> queue = new ArrayDeque<>();

queue.add(p);

boolean kq = BFS(queue);

if(kq){

Stack<PhuCuop> stack = new Stack<>();

PhuCuop s = new PhuCuop(0,0,false);

do{

stack.add(s);

s = map.get(s);

}while(s != p);

System.out.println("Cần " + stack.size() + " bước chuyển. Cách chuyển: ");

System.out.println(p);

while (!stack.isEmpty()){

System.out.println( stack.pop());

}

}

else {

System.out.println("Không chuyển được!");

}

}

public static void runDFS(){

PhuCuop p = new PhuCuop(3, 3, true);

isTham.add(p);

Stack<PhuCuop> stack = new Stack<>();

stack.push(p);

boolean kq = DFS(stack);

if(kq){

Stack<PhuCuop> st = new Stack<>();

PhuCuop s = new PhuCuop(0,0,false);

do{

st.add(s);

s = map.get(s);

}while(s != p);

System.out.println("Cần " + st.size() + " bước chuyển. Cách chuyển: ");

System.out.println(p);

while (!st.isEmpty()){

System.out.println( st.pop());

}

}

else {

System.out.println("Không chuyển được!");

}

}

public static void main(String[] args){

c.add(new Pair<>(1, 0));

c.add(new Pair<>(0, 1));

c.add(new Pair<>(1, 1));

c.add(new Pair<>(2, 0));

c.add(new Pair<>(0, 2));

// BFS

//runBFS();

//DFS

runDFS();

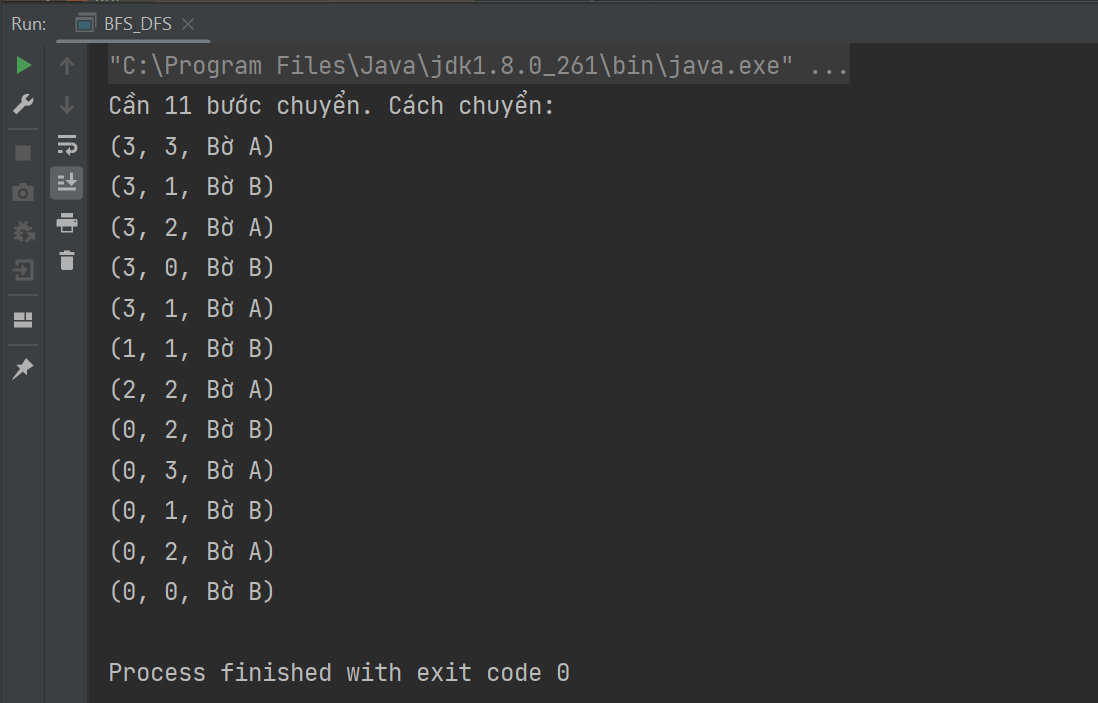
}

}

- Kết quả thử nghiệm

+ Input: (3, 3, true)

+ Output:



**4.2. Bài toán 8 số**

**- Đề bài:** Cho một ma trận 3x3, các phần tử có giá trị từ 0-8. Sắp Dịch chuyển các phần tử theo chiều trái phải lên xuống để đạt được kết quả mong muốn. Điều kiện: muốn dịch chuyển sang ô mới thì giá trị của ô mới phải bằng 0.

**- Mô tả thuật toán:** Sử dụng thuật toán tìm kiếm DFS hoặc BFS.

+ Không gian các trường hợp có thể xảy ra là di chuyển 1 giá trị sang trái, phải, lên, xuống.

+ Tạo emun Facing để lưu các hướng di chuyển.

+ Tạo 1 đối tượng tên là Node gồm:

- 1 mảng 2 chiều.

- 2 biến x,y dùng để lưu vị trí của biến có trị = 0.

- 1 biến facing có kiểu là Facing để lưu hướng di chuyển.

+ Duyệt trên không gian các trường hợp có thể xảy ra. Kiểm tra xem có thể di chuyển theo các hướng đấy không? Nếu có thì cập nhật lại mảng và giá trị của x và y, ngược lại thì không làm gì.

+ Tạo 1 Map để đánh dấu các đối tượng đã thực hiện. Một Stack or Queue để lưu trữ các đối tượng mới.

+ Lặp lại bước duyệt trên cho đến khi tìm được TTKT hoặc Stack(Queue) rỗng.

**- Kết quả cài đặt** (mô tả chương trình hoặc mã chương trình)

**\* Enum Facing:**

public enum Facing {

LEFT , RIGHT , UP , DOWN

}

**\* Class Node: Tạo đối tượng chứa mảng và các phương thưc liên quan.**

import java.util.Arrays;  
import java.util.Objects;  
public class Node {  
 int arr[][] = new int[3][3];  
 int x;  
 int y;  
 static Facing *facing*;  
  
 public Node() {  
 arr = new int[][]{{2, 8, 3}, {1,0,4}, {7,6,5}};  
 }  
 public Node(Node node){  
 for(int i=0; i<3; i++){  
 for (int j = 0; j<3; j++){  
 this.arr[i][j] = node.arr[i][j];  
 }  
 }  
 this.x = node.x;  
 this.y = node.y;  
 }  
  
 public Node(int[][] arr) {  
 for(int i=0; i<3; i++){  
 for (int j = 0; j<3; j++){  
 this.arr[i][j] = arr[i][j];  
 }  
 }  
 for(int i=0; i<3; i++){  
 for (int j = 0; j<3; j++){  
 if(this.arr[i][j] == 0){  
 x = i;  
 y = j;  
 }  
 }  
 }  
 }  
  
 public boolean canMoveLeft(){  
 return y > 0;  
 }  
 public boolean canMoveRight(){  
 return y < 2;  
 }  
 public boolean canMoveUp(){  
 return x > 0;  
 }  
 public boolean canMoveDown(){  
 return x < 2;  
 }  
  
 public boolean move(Facing f) {  
 boolean check = false;  
 int i = 0;  
 int j = 0;  
 if (f == *facing*.*LEFT* && canMoveLeft()) {  
 check = true;  
 j = -1;  
 } else if (f == *facing*.*RIGHT* && canMoveRight()) {  
 check = true;  
 j = 1;  
 } else if (f == *facing*.*UP* && canMoveUp()) {  
 check = true;  
 i = -1;  
 } else if (f == *facing*.*DOWN* && canMoveDown()) {  
 check = true;  
 i = 1;  
 }  
 if (check) {  
 arr[x][y] = arr[x + i][y + j];  
 arr[x + i][y + j] = 0;  
 x = x + i;  
 y = y + j;  
 return true;  
 }  
 return false;  
 }  
 public boolean checkFinish(){  
 for(int i = 0; i<2; i++){  
 if(arr[0][i] != i+1 || arr[2][i] != 7-i) return false;;  
 }  
 return arr[1][0] == 8 && arr[1][2] == 4;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Node node = (Node) o;  
 return x == node.x && y == node.y && Arrays.*deepEquals*(arr, node.arr);  
 }  
  
 @Override  
 public int hashCode() {  
 int result = Objects.*hash*(x, y);  
 result = 31 \* result + Arrays.*deepHashCode*(arr);  
 return result;  
 }  
  
 @Override  
 public String toString() {  
 String s = "";  
 for (int i = 0; i < 3; i++) {  
  
 for (int j = 0; j < 3; j++) {  
 s += arr[i][j] + " ";  
 }  
 s += "\n";  
 }  
 return s;  
 }  
}

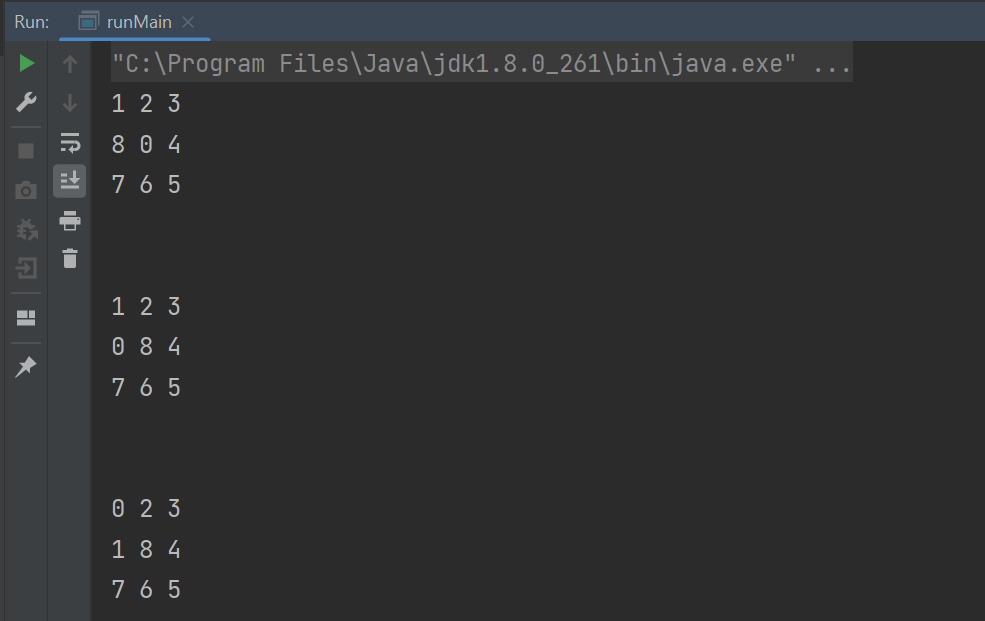
**\* Class runMain: test chương trình**

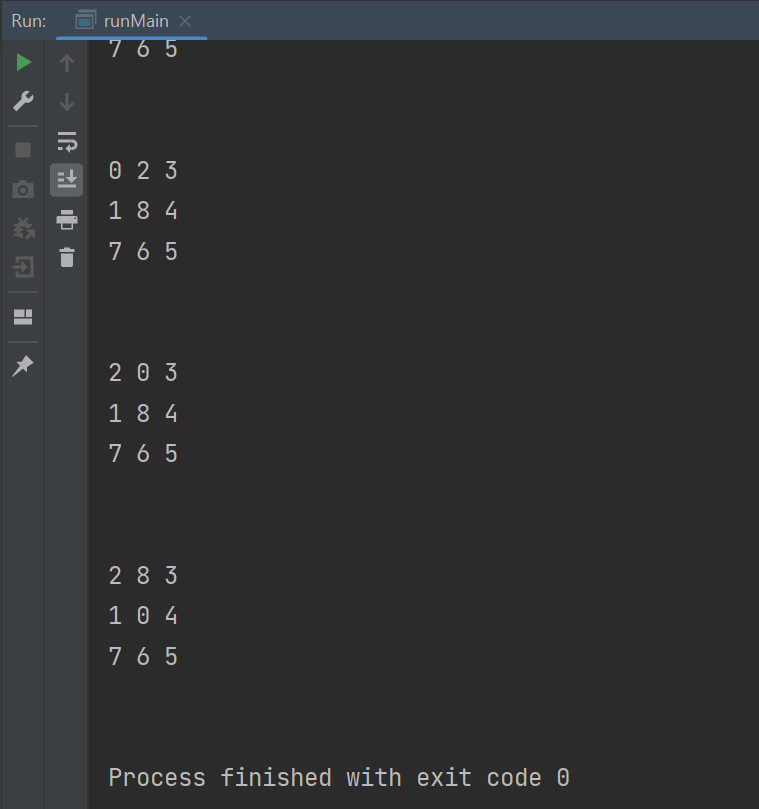
import java.util.\*;  
public class runMain {  
 static Map<Node, Node> *map* = new HashMap<>();  
 static HashSet<Node> *isTham* = new HashSet<>();  
  
 public static void main(String[] args) {  
 Node node = new Node(new int[][]{{2, 8, 3}, {1,0,4}, {7,6,5}});  
 *isTham*.add(node);  
 // BFD  
 *runBFS*(node);  
 // DFS  
// runDFS(node);  
 Node present = new Node(new int[][]{{1, 2, 3}, {8,0,4}, {7,6,5}});  
 while(!present.equals( node)){  
 System.*out*.println(present + "\n");  
 present = *map*.get(present);  
 }  
 System.*out*.println(node);  
 }  
  
 private static void runBFS(Node node){  
 Queue<Node> queue = new ArrayDeque<>();  
 queue.add(node);  
 *BFS*(queue);  
 }  
  
 private static void runDFS(Node node){  
 Stack<Node> stack = new Stack<>();  
 stack.add(node);  
 *DFS*(stack);  
 }  
  
 private static boolean BFS(Queue<Node> queue) {  
 if(queue.isEmpty()) return false;  
 Node root = new Node(queue.poll());  
 for(Facing f : Facing.*values*()){  
 Node node = new Node(root);  
 if(node.move(f)){  
 if(!*isTham*.contains(node)) {  
 queue.add(node);  
 *map*.put(node, root);  
 *isTham*.add(node);  
 }  
 if(node.checkFinish()) return true;  
 }  
 }  
 return *BFS*(queue);  
 }  
  
 private static boolean DFS(Stack<Node> stack){  
 if(stack.isEmpty()) return false;  
 Node root = new Node(stack.pop());  
 for(Facing f : Facing.*values*()){  
 Node node = new Node(root);  
 if(node.move(f)){  
 if(!*isTham*.contains(node)) {  
 stack.add(node);  
 *map*.put(node, root);  
 *isTham*.add(node);  
 }  
 if(node.checkFinish()) return true;  
 }  
 }  
 return *DFS*(stack);  
 }  
}

**- Kết quả thử nghiệm:**

+ Input: mảng 2 chiều {{1, 2, 3}, {8,0,4}, {7,6,5}}

+ Output:





**4.3. Giải thuật chứng minh**

- **Đề bài**: Cho tập các mệnh đề đã được chuẩn hóa về dạng chuẩn hội. Chứng minh 1 mệnh đề.

- **Mô tả thuật toán:** Sử dụng thuật toán tìm kiếm DFS hoặc BFS.

+ Xác định không gian các trường hợp có thể xảy ra là các mệnh đề đã cho.

+ Sử dụng phương pháp chứng minh ngược. Tức là cần chứng minh P là hệ quả của mệnh đề thì ta sẽ giả sử !P mới là hệ quả và đi chứng minh tính đúng sai của việc giả sử này.

+ Ta sử dụng Stack(Queue) để lưu trữ các mệnh đề mới được tạo ra. Sử dụng vòng lặp thực hiện các phép hội các mệnh đề trong không gian các trường hợp với mệnh đề đang xét. Nếu 2 mệnh đề hội với nhau 2 mệnh đề triệt tiêu nhau hoặc Stack(Queue) rỗng thì dùng lại.

+ TH1: Nếu ra 1 mệnh đề rộng thì khẳng định điều đang giả thiết là sai và mệnh đề được chứng minh. Truy vấn các bước thực hiện ta dùng Map để lưu.

+ TH2: Nếu sau khi thực hiện hết vòng lặp mà chưa có mệnh đề rỗng ta kết luận mệnh đề ban đầu không được chứng minh.

- **Kết quả cài đặt** (mô tả chương trình hoặc mã chương trình)

**\* Class BieuThuc: Tạo ra một biểu thức**

public class BieuThuc {  
 private String bt;  
 private boolean isPhu;  
  
 public BieuThuc() {  
 }  
  
 public BieuThuc(String bt, boolean isPhu) {  
 this.bt = bt;  
 this.isPhu = isPhu;  
 }  
  
 public String getBt() {  
 return bt;  
 }  
  
 public void setBt(String bt) {  
 this.bt = bt;  
 }  
  
 public boolean isPhu() {  
 return isPhu;  
 }  
  
 public void setPhu(boolean phu) {  
 isPhu = phu;  
 }  
  
 @Override  
 public String toString() {  
 if(isPhu){  
 return "!"+ bt;  
 }  
 return bt;  
 }  
 public BieuThuc btNguoc(){  
 return new BieuThuc(bt, !isPhu);  
 }  
}

**\* Class MenhDe: Tạo 1 mệnh đề từ các biểu thức**

import javafx.util.Pair;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

public class MenhDe {

private List<BieuThuc> menhDe = new ArrayList<>();

public MenhDe() {

}

public MenhDe(MenhDe menhDes){

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

menhDe.add(menhDes.getMenhDe().get(i));

}

}

public MenhDe(BieuThuc bt) {

menhDe.add(bt);

}

public MenhDe(List<BieuThuc> menhDe) {

this.menhDe = menhDe;

}

public MenhDe(String s) {

String[] list = s.split(" v ");

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

String c = list[i].substring(0, 1);

if(c.equals("!")) {

menhDe.add(new BieuThuc(list[i].substring(1, list[i].length()), true ));

} else{

menhDe.add(new BieuThuc(list[i], false));

}

}

}

public List<BieuThuc> getMenhDe() {

return menhDe;

}

public void setMenhDe(List<BieuThuc> menhDe) {

this.menhDe = menhDe;

}

@Override

public String toString() {

String s = "";

for(int i = 0; i < menhDe.size()-1; i++){

s += menhDe.get(i).toString() + " v ";

}

if(menhDe.size() > 0){

s += menhDe.get(menhDe.size()-1);

}

return s;

}

public void deleteBT(int index){

menhDe.remove(index);

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

MenhDe menhDe1 = (MenhDe) o;

return Objects.equals(menhDe, menhDe1.menhDe);

}

@Override

public int hashCode() {

return Objects.hash(menhDe);

}

}

**\* Class runMain: test chương trình**

import javafx.util.Pair;  
  
import java.util.\*;  
  
public class runMain {  
 static Stack<MenhDe> *THOA* = new Stack<>();  
 static List<MenhDe> *listV*;  
 static List<MenhDe> *listRes*;  
 static Map<MenhDe, Pair<MenhDe, MenhDe>> *map* = new HashMap<>();  
 static boolean *flag* = false;  
 public static List<MenhDe> createMD(){  
 MenhDe md1 = new MenhDe("!E v F");  
// MenhDe md1 = new MenhDe("P");  
 MenhDe md2 = new MenhDe("!A v !B v C");  
 MenhDe md3 = new MenhDe("!B v D");  
 MenhDe md4 = new MenhDe("!C v !D v E");  
 MenhDe md5 = new MenhDe("A");  
 MenhDe md6 = new MenhDe("B");  
 List<MenhDe> list = new ArrayList<>();  
 list.add(md1);  
 list.add(md2);  
 list.add(md3);  
 list.add(md4);  
 list.add(md5);  
 list.add(md6);  
 return list;  
 }  
 public static int findIndexMD(MenhDe u, BieuThuc v){  
 for (int i = 0; i < u.getMenhDe().size(); i++) {  
 if(u.getMenhDe().get(i).getBt().equals(v.getBt()) && u.getMenhDe().get(i).isPhu()!= v.isPhu() )  
 return i;  
 }  
 return -1;  
 }  
 public static MenhDe delete(MenhDe u, MenhDe v){  
 *flag* = false;  
 for (int i = 0; i< v.getMenhDe().size(); i++){  
 BieuThuc b = v.getMenhDe().get(i);  
 int index = *findIndexMD*(u,b);  
 if( index > -1){  
 *flag* = true;  
 u.deleteBT(index);  
 }  
 else {  
 u.getMenhDe().add(b);  
 }  
  
 }  
 return u;  
 }  
 public static boolean res(MenhDe u, MenhDe v){  
 Pair<MenhDe, MenhDe> p = new Pair<>(new MenhDe(u), v);  
 MenhDe kq = *delete*(u,v);  
 if(kq.getMenhDe().size() == 0){  
 *map*.put(new MenhDe(), p);  
 *listV*.add(v);  
 return true;  
 }  
 if(*flag* && *map*.get(kq) == null) {  
 *map*.put(kq, p);  
 *THOA*.add(kq);  
 *listV*.add(v);  
 *listRes*.add(kq);  
 }  
 return false;  
 }  
  
 public static void main(String[] args) {  
 List<MenhDe> listMD = *createMD*();  
  
 BieuThuc MDCM = new BieuThuc("F", false);  
  
  
 *THOA*.add(new MenhDe(new BieuThuc(MDCM.getBt(), !MDCM.isPhu())));  
  
 System.*out*.printf("%15s%30s%40s%50s\n", "u |", "v |", "res(u,v) |", "THOA |");  
 System.*out*.printf("%15s%30s%40s%50s\n", "|", "|", "|", new BieuThuc(MDCM.getBt(), !MDCM.isPhu())+" |");  
  
 while (!*THOA*.isEmpty()){  
  
 MenhDe u = *THOA*.pop();  
 *listV* = new ArrayList<>();  
 *listRes* = new ArrayList<>();  
  
 System.*out*.printf("%15s", u+"|");  
 for(MenhDe md : listMD){  
 if(*res*(new MenhDe(u), md)){  
// System.out.printf("%30s%40s%50s\n", "|", "|", "|");  
 System.*out*.printf("%30s%40s%50s\n",*printV*()+"|",*printRes*()+ "|", *printTHOA*()+"|");  
 System.*out*.println("Mệnh đề "+ MDCM +" là hệ quả của logic");  
 *show*(u);  
 return;  
 }  
 }  
 System.*out*.printf("%30s%40s%50s\n",*printV*()+"|",*printRes*()+ "|", *printTHOA*()+"|");  
 }  
 System.*out*.println("Mệnh đề "+MDCM+" không là hệ quả của logic");  
 }  
  
 private static void show(MenhDe u) {  
 System.*out*.print("("+ *map*.get(new MenhDe()).getKey() +", " + *map*.get(new MenhDe()).getValue() + ")");  
 u = *map*.get(new MenhDe()).getKey();  
 while (*map*.get(u) != null){  
 System.*out*.print(" <- ");  
 System.*out*.print("("+ *map*.get(u).getKey() +", " + *map*.get(u).getValue() + ")");  
 u = *map*.get(u).getKey();  
 }  
 System.*out*.println();  
 }  
  
 private static String printV() {  
 String s = "";  
 for (MenhDe md : *listV*){  
 s += md + ", ";  
 }  
 return s;  
 }  
 private static String printRes() {  
 String s = "";  
 for (MenhDe md : *listRes*){  
 s += md + ", ";  
 }  
 return s;  
 }  
 private static String printTHOA(){  
 Stack<MenhDe> stack = new Stack<>();  
 int i = 0;  
 String res = "";  
 while (i < *THOA*.size()){  
 i++;  
 MenhDe bt = *THOA*.peek();  
 stack.add(*THOA*.pop());  
 res += bt + ", ";  
 }  
 *THOA* = stack;  
 return res;  
 }  
}

**- Kết quả thử nghiệm:**

+ Input:

(1) !E v F

(2) !A v !B v C

(3) !B v D

(4) !C v !D v E

(5) A

(6) B

Chứng minh F.

+ Output:

