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: Hoàng Xuân Tuyền.

2. Mã sinh viên: 191241113

3. Lớp: CNTT6-K60

**4. Nội dung báo cáo**

**4.1. Thuật toán Proof by Resolution**

* Mô tả thuật toán:
  + Thuật toán sử dụng đầu vào là một tập các câu tuyển, và các câu tuyển này là đúng.
  + Thực hiện chứng minh rằng S đúng bằng cách giả sử là !S là đúng sau đó chỉ ra mâu thuẫn khi kết hợp !S với tập các câu tuyển đã có từ trước.
  + Sử dụng 1 list để lưu danh sách các cơ sở tri thức được sử dụng cho bài toán, 1 queue để lưu các mệnh đề mới được sinh ra khi kết hợp mệnh đề hiện tại với một trong các cơ sở tri thức đã có từ trước, 1 list để thực hiện truy vết.
  + Thực hiện kết hợp mệnh đề hiện tại với từng mệnh đề trong cơ sở tri thức và thêm nó vào queue.
  + Một mệnh đề mới sẽ được thêm vào queue khi nó chưa từng xuất hiện trong queue.
  + Thực hiện vòng lặp cho đến khi xuất hiện một câu rỗng hoặc queue rỗng.
* Cài đặt thuật toán bằng Java:

public class Clause implements Cloneable,Comparable{

String content;

boolean type;

public Clause(String content, boolean type) {

this.content = content;

this.type = type;

}

public boolean isSimilar(Clause b){

if(content.equals(b.getContent()) && type==b.type)

return true;

return false;

}

public boolean isContrast(Clause b){

if(content.equals(b.getContent()) && type != b.type)

return true;

return false;

}

@Override

public boolean equals(Object obj) {

Clause d = (Clause) obj;

if(content.equals(d.getContent()) && type==d.type)

return true;

return false;

}

protected Clause clone(){

return new Clause(this.content,this.type);

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public boolean isType() {

return type;

}

public void setType(boolean type) {

this.type = type;

}

@Override

public int compareTo(Object o) {

Clause c = (Clause) o;

if(this.isSimilar(c))

return 0;

return -1;

}

@Override

public String toString() {

String str = "";

if(!type)

str+="!";

return str+=content;

}

}

import java.util.ArrayList;

import java.util.Objects;

public class ConjunctionClause implements Comparable{

ConjunctionClause parent1,parent2;

ArrayList<Clause> list;

@Override

public boolean equals(Object o) {

ConjunctionClause that = (ConjunctionClause) o;

return Objects.*equals*(list, that.list);

}

public ConjunctionClause combine(ConjunctionClause a){

ArrayList<Clause> list = new ArrayList<>();

ArrayList<Clause> thislist = (ArrayList<Clause>)this.list.clone();

ArrayList<Clause> a\_getlist = (ArrayList<Clause>)a.getList().clone();

for (Clause c:this.list){

for(Clause b:a.getList()){

if(c.isContrast(b)){

thislist.set(this.list.indexOf(c),null);

a\_getlist.set(a.getList().indexOf(b),null);

}

if(c.isSimilar(b)){

thislist.set(this.list.indexOf(c),null);

}

}

}

for(Clause c:thislist){

if(c!=null)

list.add(c.clone());

}

for(Clause c:a\_getlist){

if(c!=null)

list.add(c.clone());

}

return new ConjunctionClause(list);

}

public boolean canCombine(ConjunctionClause a){

for(Clause b:a.getList()){

for(Clause c:list){

if(b.isContrast(c))

return true;

}

}

return false;

}

public boolean contains(Clause c){

for(Clause x:this.list)

if(c.equals(x))

return true;

return false;

}

public ConjunctionClause(ArrayList<Clause> list) {

this.list = new ArrayList<>();

for(Clause c:list)

this.list.add(c.clone());

}

public ConjunctionClause setParent1(ConjunctionClause parent1) {

this.parent1 = parent1;

return this;

}

public ConjunctionClause setParent2(ConjunctionClause parent2) {

this.parent2 = parent2;

return this;

}

public ArrayList<Clause> getList() {

return list;

}

@Override

public int compareTo(Object o) {

ArrayList<Clause> list1 = ((ConjunctionClause) o).getList();

if(this.list.isEmpty())

return -1;

if(list1.isEmpty())

return 1;

if(list1.size() == list.size()){

for(Clause c:list)

if(!list.contains(c))

return -1;

return 0;

}

return -1;

}

@Override

public String toString() {

String str = "(";

if(list==null||list.isEmpty())

return "o";

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

str+=list.get(i)+" v ";

str+=list.get(list.size()-1)+")";

return str;

}

}

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Queue;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Queue;

public class Main {

public static void main(String[] args) {

Clause a\_neg=new Clause("A",false),

b\_neg=new Clause("B",false),

b\_pos=new Clause("B",true),

c\_neg=new Clause("C",false),

d\_neg=new Clause("D",false),

e\_neg=new Clause("E",false),

a\_pos=new Clause("A",true),

e\_pos=new Clause("E",true),

d\_pos=new Clause("D",true),

c\_pos=new Clause("C",true),

f\_pos=new Clause("F",true),

f\_neg=new Clause("F",false);

ArrayList<Clause> list = new ArrayList<>();

list.add(e\_neg);

list.add(f\_pos);

ConjunctionClause one = new ConjunctionClause(list);

list = new ArrayList<>();

list.add(a\_neg);

list.add(b\_neg);

list.add(c\_pos);

ConjunctionClause two = new ConjunctionClause(list);

list = new ArrayList<>();

list.add(b\_neg);

list.add(d\_pos);

ConjunctionClause three = new ConjunctionClause(list);

list = new ArrayList<>();

list.add(c\_neg);

list.add(d\_neg);

list.add(e\_pos);

ConjunctionClause four = new ConjunctionClause(list);

list = new ArrayList<>();

list.add(a\_pos);

ConjunctionClause five = new ConjunctionClause(list);

list = new ArrayList<>();

list.add(b\_pos);

ConjunctionClause six = new ConjunctionClause(list);

list = new ArrayList<>();

ArrayList<ConjunctionClause> premise = new ArrayList<>();

premise.add(one);

premise.add(two);

premise.add(three);

premise.add(four);

premise.add(five);

premise.add(six);

list.add(f\_neg);

ConjunctionClause seven = new ConjunctionClause(list);

Queue<ConjunctionClause> queue = new LinkedList<>();

queue.add(seven);

ArrayList<ConjunctionClause> listQueue = new ArrayList<>();

ArrayList<ConjunctionClause> listRes = new ArrayList<>();

boolean flag = false;

while(!queue.isEmpty()){

ConjunctionClause c = queue.poll();

listQueue.add(c);

for(ConjunctionClause cc : premise){

if(cc.canCombine(c))

{

if(cc.combine(c).getList().isEmpty())

{

flag = true;

}

if(!queue.contains(cc.combine(c))&&!listQueue.contains(cc.combine(c)))

{

queue.add(cc.combine(c).setParent1(c).setParent2(cc));

listRes.add(cc.combine(c).setParent1(c).setParent2(cc));

}

if(flag)

break;

}

}

if(flag)

break;

}

if(flag)

System.*out*.println(f\_pos+" duoc chung minh !");

else

System.*out*.println("Khong the chung minh");

//truy vết

ConjunctionClause res = listRes.get(listRes.size()-1);

while(res!=null){

System.*out*.println(res+" <- "+res.parent1+" V "+res.parent2);

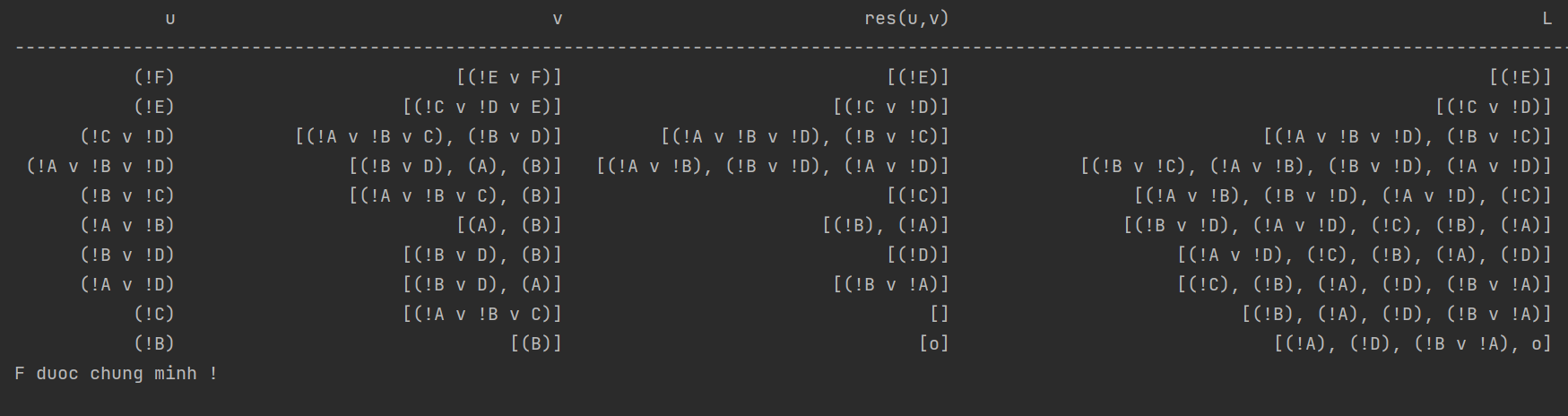
res = res.parent1;

}

}

}

* Kết quả thử nghiệm:



**4.2. Bài toán 8 số:**

* Mô tả thuật toán:
  + Sử dụng chiến lược tìm kiếm mù, tìm kiếm theo chiều rộng.
  + Mỗi một trạng thái được thể hiện là 1 ma trận 2 chiều.
  + Trạng thái đầu tiên là một ma trận cho trước và kết quả là các nước đi để ra trạng thái sau cùng.
  + Trạng thái khởi đầu và trạng thái kết thúc được cho trước.
  + Mỗi một vị trí trên ma trận đều được đánh số từ 0 → 8.
  + Vị trí số 0 sẽ là vị trí thực hiện di chuyển để thay đổi trạng thái.
  + Các toán tử bao gồm: Lên, xuống, trái, phải được thể hiện bằng 2 ma trận một chiều: next\_x = [1,-1,0,0] next\_y = [0,0,1,-1]
  + Đối với từng trạng thái của bàn cờ sẽ sinh ra được một số lượng các trạng thái nhất định, điều này phụ thuộc vào vị trí của phần tử số 0 và trạng thái mới được sinh ra đã được duyệt qua hay chưa.
  + Khởi tạo một Queue để lưu các trạng thái kế tiếp, một mảng lưu các trạng thái đã đi qua. Mỗi một trạng thái sẽ bao gồm ma trận trạng thái và x,y thể hiện nước đi mà nó đã thực hiện để đạt tới trạng thái đó.
  + Tại bước đầu tiên chúng ta thêm trạng thái đầu tiên vào Queue, sau đó thực hiện vòng lặp:
    - Lấy phần tử trên cùng của queue, thực hiện sinh trạng thái mới, nếu trạng thái chưa từng duyệt và chưa có trong queue thì thêm nó vào queue, sau đó thêm trạng thái hiện tại vào danh sách các trạng thái đã duyệt.
* Thực hiện vòng lặp cho đến khi queue rỗng mà vẫn chưa đạt tới trạng thái kết thúc thì kết luận bài toán là không thể giải, nếu gặp trạng thái kết thúc thì tiến hành truy vết.
* Cài đặt chương trình bằng python:

from copy import deepcopy

import traceback

import numpy as np

class status:

matrix = []

x = None

y = None

def \_\_init\_\_(self,matrix,x,y):

self.matrix = np.array(matrix)

self.x = x

self.y = y

class bai2:

init\_status = status([[2,8,3],[1,6,4],[7,0,5]],2,1)

final\_status = np.array([[1,2,3],[8,0,4],[7,6,5]])

visited = np.array([])

next = []

next\_x = [1,-1,0,0]

next\_y = [0,0,1,-1]

def \_\_init\_\_(self):

self.next.append(self.init\_status)

def status\_generator(self,stat):

next\_stat = []

point = np.argwhere(stat.matrix == 0)

x = point[0][0]

y = point[0][1]

for i in range(0,4):

x1 = x+self.next\_x[i]

y1 = y+self.next\_y[i]

if(x1 <= 2 and x1 >= 0 and y1 <=2 and y1 >= 0):

temp = status(stat.matrix,self.next\_x[i],self.next\_y[i])

temp.matrix[x][y],temp.matrix[x1][y1] = temp.matrix[x1][y1],temp.matrix[x][y]

if(not any((temp.matrix == node.matrix).all() for node in self.visited)):

next\_stat.append(temp)

return next\_stat

def exec(self):

while(len(self.next)>0):

status = self.next.pop(0)

if(np.array\_equal(status.matrix,self.final\_status)):

return status

self.visited = np.append(self.visited,status)

temp = self.status\_generator(status)

self.next += temp

def swap(self,matrix,x1,y1):

temp = np.array(matrix)

point = np.argwhere(matrix == 0)

x = point[0][0]

y = point[0][1]

prevx = x-x1

prevy = y-y1

temp[x][y],temp[prevx][prevy] = temp[prevx][prevy],temp[x][y]

return temp

def trace\_back(self,status):

n = len(self.visited) - 1

temp\_status = (status)

traceback = []

while n>=0:

temp = self.swap(temp\_status.matrix,temp\_status.x,temp\_status.y)

if((self.visited[n].matrix == temp).all()):

traceback.append(temp[:])

temp\_status = self.visited[n]

n-=1

return traceback

a = bai2()

final = a.exec()

path = a.trace\_back(final)

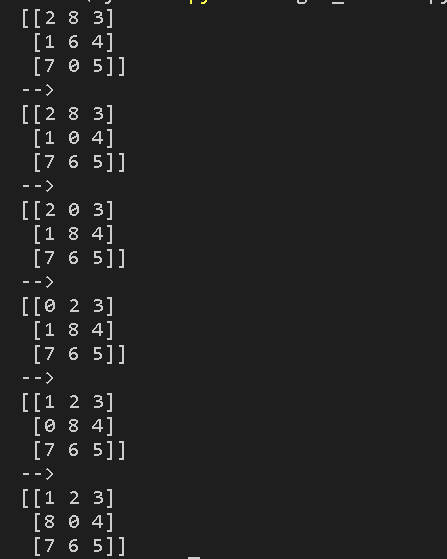
path.reverse()

for i in path:

print(i,"\n-->")

print(final.matrix)

* Kết quả thử nghiệm:



**4.3. Bài toán Triệu phú và kẻ cướp**

* Mô tả thuật toán:
* Dùng một bộ 3 số (a,b,k) để biểu diễn các trạng thái
  + a - số triệu phú, b - số kẻ cướp ở bờ tả
  + k=1 thuyền đang ở bờ tả, ngược lại k=0
* **Không gian trạng thái được xác định như sau:**
  + TTĐ:(3,3,1)
  + TTKT:(0,0,0)
* Tập các toán tử bao gồm các toán tử:
  + Thuyền chở 1 kẻ cướp
  + Thuyền chở 1 triệu phú
  + Thuyền chở 1 triệu phú, 1 kẻ cướp
  + Thuyền chở 2 kẻ cướp
  + Thuyền chở 2 triệu phú
* Tương tứng với mỗi toán tử sẽ được thể hiện bằng 1 bộ 3 số như sau:
  + (1,0,1)
  + (0,1,1)
  + (1,1,1)
  + (2,0,1)
  + (0,2,1)
* Cho dù là chở cướp hay triệu phú thì vẫn luôn có 1 con thuyền di chuyển từ bờ này sang bờ kia, vì vậy, giá trị thứ 3 của tập 3 số luôn là 1:
* Ví dụ: (1,0,1) có nghĩa là sẽ có 1 cướp, 0 triệu phú, và 1 thuyền di chuyển từ bờ bên này sang bờ bên kia.
* Nếu chúng ta di chuyển nhân vật từ bờ bên trái sang bờ bên phải, chúng ta sẽ thực hiện phép trừ cho mỗi toán tử:
  + Ví dụ: Trạng thái hiện tại đang là: (3,3,1) có nghĩa là đang có 3 cướp, 3 triệu phú và 1 thuyền ở bên tả, khi ta di chuyển một cướp sang bên hữu thì ta sẽ thực hiện phép trừ:
    - (3,3,1) - (1,0,1) = (2,3,0) thể hiện rằng ở bờ bên trái lúc này chỉ còn 2 cướp, 3 triệu phú và 0 thuyền.

ngược lại khi ta di chuyển từ bên phải về bên trái ta sẽ thực hiện phép cộng.

* Việc chở người sẽ diễn ra luân phiên từ bên nọ sang bên kia cho đến khi 3 cướp, 3 triệu phú, thuyền đều ở phía bên phải, có nghĩa rằng phía bên trái lúc này là (0,0,0).
* Tại mỗi bước ta chọn trạng thái để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác
* Sử dụng danh sách L để lưu các trạng thái được sinh ra và chờ được phát triển
* Mục tiêu của tìm kiếm là tìm đường đi từ trạng thái đầu đến trạng thái đích nên ta cần lưu vết của đường đi.

- Cài đặt chương trình bằng python:

from collections import deque

from mimetypes import init

import traceback

def sum(a,b,c=1):

return (a[0]+c\*b[0],a[1]+c\*b[1],a[2]+c\*b[2])

class bai1:

visited\_status = []

next\_status = []

init\_status = (3,2,1)

traceback = []

def \_\_init\_\_(self):

self.next\_status.append(self.init\_status)

def status\_generator(self,x):

operator = [(1,0,1),(1,1,1),(2,0,1),(0,1,1),(0,2,1)]

status\_generated = []

if(x[-1]==1):

for i in operator:

temp = sum(x,i,-1)

if(temp[0]>=0 and temp[1]>=0):

if(temp[0]>=temp[1] and temp not in self.visited\_status and temp not in self.next\_status):

status\_generated.append(temp)

if(x[-1]==0):

for i in operator:

temp = sum(x,i)

if(temp[0]<=3 and temp[1]<=3):

if(temp[0]>=temp[1] and temp not in self.visited\_status and temp not in self.next\_status):

status\_generated.append(temp)

# print(status\_generated)

return status\_generated

def exec(self):

while(len(self.next\_status)>0):

status = self.next\_status.pop(0)

if(status[0]==0):

return status

self.visited\_status.append(status)

temp = self.status\_generator(status)

self.next\_status += temp

self.traceback.append(temp)

a = bai1()

final = a.exec()

# print(len(a.visited\_status),len(a.traceback))

for i in range(0,len(a.traceback)):

print(a.visited\_status[i],":",a.traceback[i])

i=len(a.traceback)-1

while(i>=0):

if(final in a.traceback[i]):

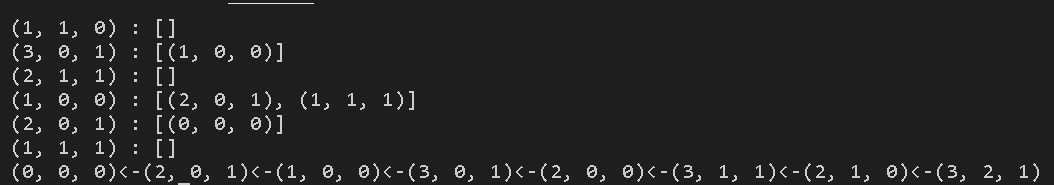
print(final,end="<-")

final = a.visited\_status[i]

i-=1

print(a.init\_status)

- Kết quả thử nghiệm:



* Giải thích kết quả: Từ trên xuốn dưới, phía bên trái là trạng thái hiện tại của chương trình, bên phải là tập các trạng thái có thể sinh ra từ trạng thái hiện tại.
* Dòng cuối cùng là trạng thái kết thúc cùng với các trạng thái đã được thực hiện trước đó để đạt tới trạng thái cuối cùng.