Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

**Лабораторна робота №1**

з дисципліни «Основи штучного інтелекту»

на тему:

*«***Неінформативний пошук***»*

Варіант -13

Виконав:

студент групи ІС-72

Кривохижа Роман Андрійович

Київ - 2020

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

**Завдання**: Використати алгоритм пошуку в глибину

**Умова**: сталося так, що до берега великого Гангу під'їхали відразу троє магараджею зі своїми пери. Всі вони хотіли переправитися на інший берег, але звичаї не дозволяли жодній пери залишатися в човні або на березі одній з чужим чоловіком, якщо поруч не буде свого. Біля берега стояв невеликий човен, який витримує тільки двох людей. Звичаї не забороняють пери самої керувати човном. Потрібно знайти послідовність поїздок, яка гарантує переміщення всіх магараджі і пери на інший берег.

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

**Лістинг програми**:

* main.py

import sys

from state import State

from computation import expand

def dfs():

state\_number = 0

while True:

state\_number = state\_number + 1

print("Visited states: ", state\_number)

if not layer:

return None

current = layer.pop() # get last item

if sum(current.shore) == 6:

return current

print('Depth:', current.depth)

print('current', current.shore)

expand(current, layer, visited)

visited.append(current)

print()

if \_\_name\_\_ == '\_\_main\_\_':

couples\_number = 3

boat\_capacity = 2

initial = State([], 0)

layer = [] # open list

visited = [] # closed list

initial.shore.extend([0 for \_ in range(couples\_number \* 2)])

print('Initial shore:', initial.shore)

layer.append(initial)

goal = dfs()

if goal is None:

print('\nCan`t find path!')

sys.exit(-1)

print('---------------------------------------------------------')

print("\nSuccess: ", goal.shore)

print("Depth: ", goal.depth)

path = []

for item in goal.path:

path.append(item.shore)

path.append([1 for \_ in range(couples\_number \* 2)])

print("Path: ", path)

* computation.py

from copy import deepcopy

def is\_jealousy(current, couples\_number=3):

"""

Check jealousy for current state.

Args:

current:

couples\_number:

Returns:

"""

for i in range(0, couples\_number):

# husband is not with his wife

if current.shore[i] != current.shore[couples\_number + i]:

for j in range(couples\_number, couples\_number \* 2):

# another man is with the wife

if current.shore[j] == current.shore[i]:

print(current.shore, '-', 'X')

return True

print(current.shore, '-', 'is not jealousy')

return False

def change\_position(bit):

"""

Used to change position of people or the boat.

Args:

bit:

Returns:

"""

return abs(bit - 1)

def people\_near\_boat(state):

"""

Get people on the same side as the boat.

Args:

state:

Returns:

"""

people = deepcopy(state.shore)

for i in range(0, len(state.shore)):

if state.shore[i] == state.boat:

people[i] = True

else:

people[i] = False

return people

def is\_visited(state, visited):

"""

Determines whether a State has already been visited.

Args:

state:

visited:

Returns:

"""

for visited\_state in visited:

if state.shore == visited\_state.shore and state.boat == visited\_state.boat:

return True

return False

def move(cap, state, movement, result, start):

"""

Computes all possible moves from a current State with a certain boat capacity.

Args:

cap:

state:

movement:

result:

start:

Returns:

"""

for i in range(start, len(state.shore)):

if people\_near\_boat(state)[i]:

movement.append(i)

if cap > 1:

# without duplicates (permutations)

move(cap - 1, state, movement, result, i)

if cap == 1:

result.append(deepcopy(movement))

movement.pop()

return result

def expand(state, layer, visited, couples\_number=3, boat\_capacity=2):

"""

Expand and add new states to layer.

Args:

state:

layer:

visited:

couples\_number:

boat\_capacity:

Returns:

"""

result = []

# get all possible moves for the current State and capacity

possible\_moves = move(boat\_capacity, state, [], result, 0)

for pos\_move in possible\_moves:

# create next state

following\_state = deepcopy(state)

# move one or two person

if len(set(pos\_move)) == 1:

following\_state.shore[pos\_move[0]] = change\_position(state.shore[pos\_move[0]])

else:

for person\_idx in pos\_move:

following\_state.shore[person\_idx] = change\_position(state.shore[person\_idx])

# move boat

following\_state.boat = change\_position(state.boat)

if is\_visited(following\_state, visited):

continue

elif is\_jealousy(following\_state, couples\_number):

visited.append(following\_state)

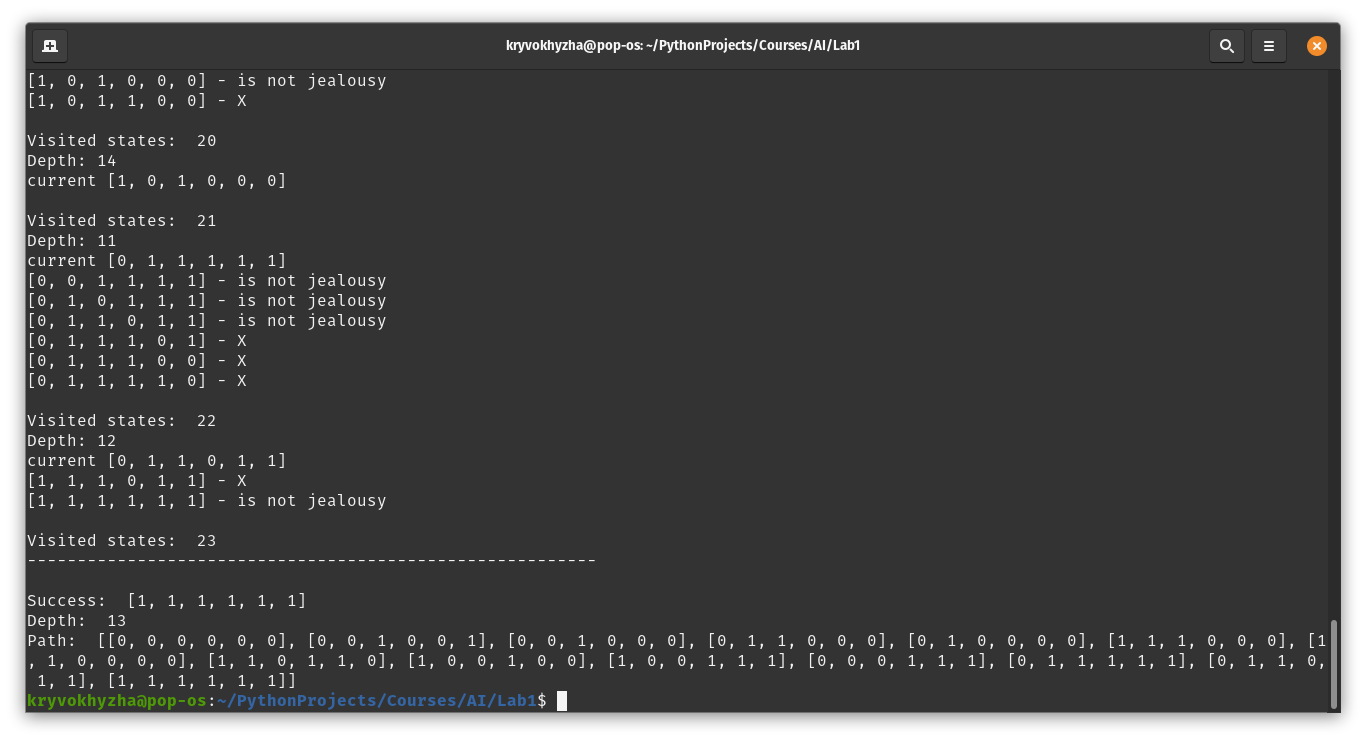
else:

following\_state.depth += + 1

following\_state.path.append(state)

layer.append(following\_state)

**Виконання програми**:

****

**Висновок**:

Даний метод (пошук в глибину) знаходить шлях, тобто розв’язок задачі, який є не обов’язково оптимальним. Також пошук може ніколи не завершитись, якщо граф матимемо нескінченний граф.

Алгоритмічна складність даного методу складає О(m).