1. 8 Puzzle problem

from collections import deque

def bfs(start, goal='123456780'):

moves = [(-1,0), (1,0), (0,-1), (0,1)]

visited, queue = set(), deque([(start, [])])

while queue:

state, path = queue.popleft()

if state in visited: continue

visited.add(state)

if state == goal: return path + [state]

i = state.index('0')

x, y = divmod(i, 3)

for dx, dy in moves:

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

ni = nx \* 3 + ny

lst = list(state)

lst[i], lst[ni] = lst[ni], lst[i]

queue.append((''.join(lst), path + [state]))

return None

start\_state = '123405678'

for step in bfs(start\_state): print(step)

2. prolog product

% Base case: product from 1 to 1 is 1

product\_to\_n(1, 1).

% Recursive case: product from 1 to N is N \* product from 1 to N-1

product\_to\_n(N, Product) :-

N > 1,

N1 is N - 1,

product\_to\_n(N1, Product1),

Product is N \* Product1.

:- initialization(main).

main :-

product\_to\_n(10, Product),

write('Product from 1 to 10 is: '), write(Product), nl.

3. DB Name,DOB

% Knowledge Base: Facts about people (Name and Date of Birth)

% Format: person(Name, DOB).

person('Alice', '1998-03-15').

person('Bob', '2000-07-22').

person('Charlie', '1995-11-30').

person('Diana', '2001-01-10').

person('Eve', '1999-09-09').

% Sample query: get all people and their DOBs

list\_all\_people :-

person(Name, DOB),

write('Name: '), write(Name), write(', DOB: '), write(DOB), nl,

fail. % forces backtracking to get all entries

list\_all\_people. % succeed at the end

:- initialization(main).

main :-

write('People in the database:'), nl,

list\_all\_people,

halt.

4.Planet DB

% Facts: planet(Name, Size, DistanceFromSun).

planet(mercury, small, 57).

planet(venus, medium, 108).

planet(earth, medium, 150).

planet(mars, small, 228).

planet(jupiter, large, 778).

planet(saturn, large, 1430).

% Query to find planets by size:

small\_planet(X) :- planet(X, small, \_).

large\_planet(X) :- planet(X, large, \_).

5.Teacher Student

% Facts: student(Name, SubjectCode).

student(john, cs101).

student(mary, cs102).

student(alice, cs101).

student(rahul, cs103).

% Facts: teacher(Name, SubjectCode).

teacher(mr\_smith, cs101).

teacher(mr\_jones, cs102).

teacher(mrs\_clark, cs103).

% Rule to find which teacher teaches which student

teaches(Teacher, Student) :-

student(Student, Code),

teacher(Teacher, Code).

:- initialization(main).

% "main" function to display the results using write

main :-

teaches(Teacher, Student),

write(Teacher), write(' teaches '), write(Student), nl,

fail. % force backtracking to get all results

main. % stop when no more results

6. bird can and cannot fly:

% Facts

bird(sparrow).

bird(pigeon).

bird(penguin).

% Rule: penguin is a bird but can't fly

cannot\_fly(penguin).

can\_fly(X) :-

bird(X),

\+ cannot\_fly(X).

% Main function

main :-

write('Can Fly:'), nl,

can\_fly(Bird),

write('- '), write(Bird), nl,

fail.

:- initialization(main).

main :-

write('Cannot Fly:'), nl,

cannot\_fly(Bird),

write('- '), write(Bird), nl,

fail.

main.

7. Family Tree Qn:

parent(john, mary). % john is parent of mary

parent(john, david).

parent(mary, alice).

parent(david, bob).

% Rules

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

:- initialization(main).

main :-

write('Grandparent Relationships:'), nl,

grandparent(X, Y),

write(X), write(' is grandparent of '), write(Y), nl,

fail.

main :-

write('Sibling Relationships:'), nl,

sibling(X, Y),

write(X), write(' and '), write(Y), write(' are siblings'), nl,

fail.

main.

8. Count Vowels:

count\_vowels([], 0).

count\_vowels([H|T], Count) :-

count\_vowels(T, RestCount),

(member(H, [a,e,i,o,u]) -> Count is RestCount + 1 ; Count = RestCount).

% Convert string to char list and count vowels

main :-

String = [h,e,l,l,o,o],

count\_vowels(String, Count),

write('Vowel Count: '), write(Count), nl.

:- initialization(main).

9. Pattern Matching :

% Match first element of a list

match\_first([H|\_], H).

% Match second element

match\_second([, X|], X).

% Match specific pattern [a, b, c]

match\_abc([a, b, c]).

main :-

match\_first([1,2,3], X),

write('First element: '), write(X), nl,

match\_second([1,2,3], Y),

write('Second element: '), write(Y), nl,

match\_abc([a,b,c]),

write('Pattern [a,b,c] matched.'), nl.

:- initialization(main).

10. Fruit and its Colour using Backtracking:

fruit(apple, red).

fruit(tomato, red).

fruit(orange, safron).

fruit(grapes, green).

fruit(kiwi, green).

main:-

fruit(Fruit, Color),

write(Fruit), write(' is '), write(Color), nl,

fail.

main. % <- this line is IMPORTANT to stop backtracking

:- initialization(main).

11. tower of Hanoi:

% hanoi(N, Source, Target, Auxiliary)

hanoi(1, A, B, \_) :-

write('Move disk from '), write(A), write(' to '), write(B), nl.

hanoi(N, A, B, C) :-

N > 1,

M is N - 1,

hanoi(M, A, C, B),

hanoi(1, A, B, \_),

hanoi(M, C, B, A).

:- initialization(main).

main :-

NumDisks = 3,

hanoi(NumDisks, left, right, center).

12. deitng system based o disease:

% disease(Disease, RecommendedFood).

disease(diabetes, oats).

disease(diabetes, brown\_rice).

disease(hypertension, banana).

disease(hypertension, spinach).

disease(obesity, salad).

disease(obesity, fruits).

recommend(Disease) :-

disease(Disease, Food),

write('Recommended food for '), write(Disease), write(': '), write(Food), nl,

fail.

recommend(\_).

:- initialization(main).

main :- recommend(diabetes).

13. Medical Diagnosis:

% symptom(Disease, Symptom)

symptom(flu, fever).

symptom(flu, cough).

symptom(cold, sneezing).

symptom(cold, cough).

symptom(malaria, fever).

symptom(malaria, chills).

has\_symptom(john, fever).

diagnose(Patient, Disease) :-

symptom(Disease, Sym),

has\_symptom(Patient, Sym),

write(Patient), write(' may have '), write(Disease), nl.

diagnose(Patient, Disease).

:- initialization(main).

main :-

diagnose(Patient, Disease).

14. Backchaining:

% Facts

rain.

wet\_grass :- rain.

slippery\_road :- wet\_grass.

% Query to check if slippery\_road is true

:- initialization(main).

main :-

( slippery\_road

-> write('It is slippery on the road.'), nl

; write('Road is not slippery.'), nl

),

halt.

15. Water jug Problem :

from collections import deque

def water\_jug(b, a, target):

visited = set()

queue = deque([(0, 0)])

while queue:

x, y = queue.popleft()

if x == target or y == target:

return True

if (x, y) in visited:

continue

visited.add((x, y))

queue.extend([

(b, y), # fill jug b

(x, a), # fill jug a

(0, y), # empty jug b

(x, 0), # empty jug a

(min(b, x + y), max(0, x + y - b)), # pour a -> b breadth first search

(max(0, x + y - a), min(a, x + y)) # pour b -> a

])

return False

print(water\_jug(4, 3, 2)) # Example: jug sizes 4 and 3, target 2

16. breadth first search :

def vacuum\_cleaner(state):

# state: tuple (location, room1\_dirty, room2\_dirty)

location, r1, r2 = state

if not r1 and not r2:

print("All clean!")

return

if location == 0:

if r1:

print("Clean room 1")

vacuum\_cleaner((0, False, r2))

else:

print("Move to room 2")

vacuum\_cleaner((1, r1, r2))

else:

if r2:

print("Clean room 2")

vacuum\_cleaner((1, r1, False))

else:

print("Move to room 1")

vacuum\_cleaner((0, r1, r2))

vacuum\_cleaner((0, True, True))