Priyanka Srinivas | RA1911026010014

Artificial Intelligence Lab

Exercise 3: Course Prerequisite Checking - A Constraint Satisfaction Problem

AIM: To implement a constraint satisfaction problem (CSP) from the real world

INTRODUCTION:

What is a constraint satisfaction problem?

Constraint satisfaction is a problem-solving strategy in which the values of a problem satisfy specific restrictions or criteria. A strategy like this leads to a better grasp of the problem's structure and complexity.

Constraint satisfaction depends on three components, namely:

- **X:** It is a set of variables.
- **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
- **C:** It is a set of constraints which are followed by the set of variables.

PROBLEM STATEMENT:

There are a total of numCourses courses you have to take, labelled from 0 to numCourses - 1. You are given an array of prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

ALGORITHM:

1. Create a function that accepts the number of courses to be taken along with the prerequisites.
2. Make a recursive function that accepts the graph, the current index, the number of vertices, and the output graph as inputs.
3. Check if the configuration is safe for each provided course (i.e. check if the course is not a prerequisite to some other course), then recursively run the function with the next index and number of vertices.
4. Break the loop and return true if any recursive function returns true.
5. Return false if no recursive function returns true.

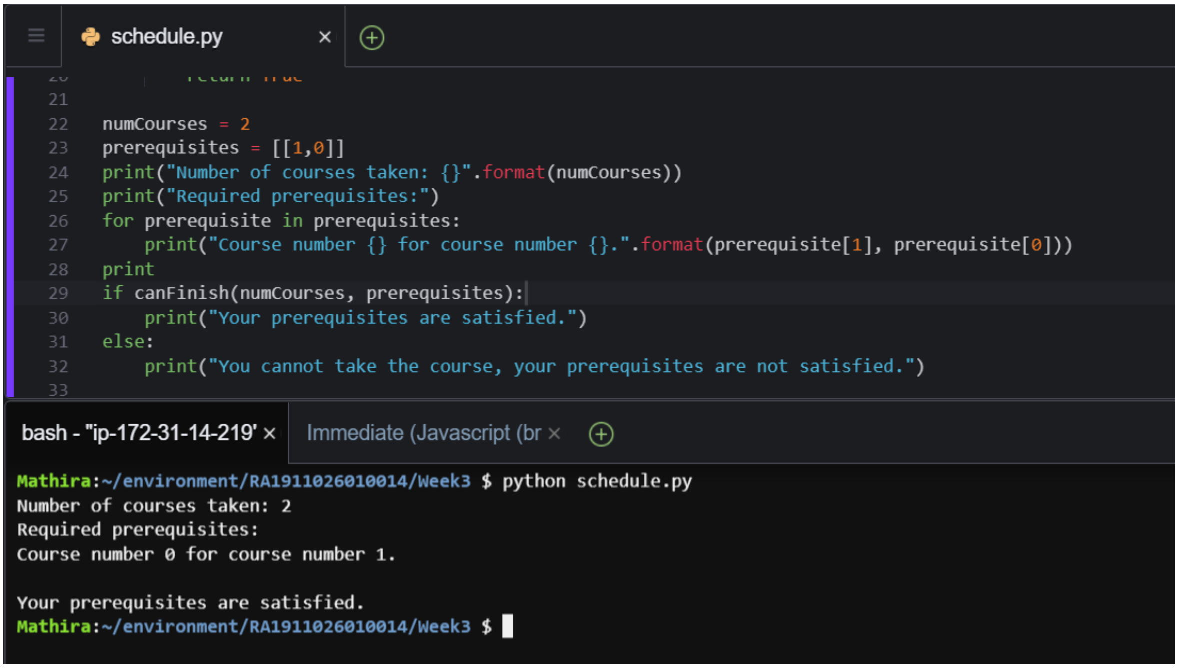
PROGRAM:

```
from collections import defaultdict
def canFinish(numCourses, prerequisites):
        def cycle(v, G, R, B):
            if v in R:
                return True
            R.add(v)
            if v in G:
                for _v in G[v]:
                    if _v not in B and cycle(_v, G, R, B):
                        return True
            R.remove(v); B.add(v)
            return False

        G, R, B = defaultdict(list), set(), set()
        for p in prerequisites:
            G[p[0]].append(p[1])
        for v in G:
            if v not in B and cycle(v, G, R, B):
                return False
        return True

numCourses = 2
prerequisites = [[1,0]]
print(canFinish(numCourses, prerequisites))
```

OUTPUT:



OBSERVATION:

From the above algorithm, we can see that the constraint of having taken a course prior to pursuing another course has been satisfied.

RESULT:

The constraint satisfaction problem was implemented successfully.