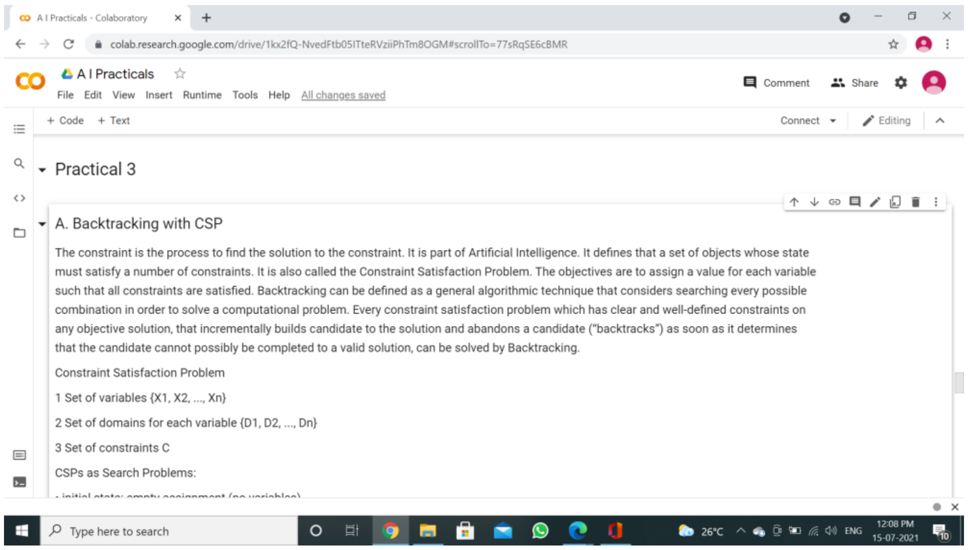
Name: Zeenat Class: TYIT Roll no: 578

Practical 3(A)

Aim: Backtracking with CSP
Colab linkL: https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=Fq1DVQK159NV



Practical 3

A. Backtracking with CSP

The constraint is the process to find the solution to the constraint. It is part of Artificial Intelligence. It defines that a set of objects whose state must satisfy a number of constraints. It is also called the Constraint Satisfaction Problem. The objectives are to assign a value for each variable such that all constraints are satisfied. Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem. Every constraint satisfaction problem which has clear and well-defined constraints on any objective solution, that incrementally builds candidate to the solution and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution, can be solved by Backtracking.

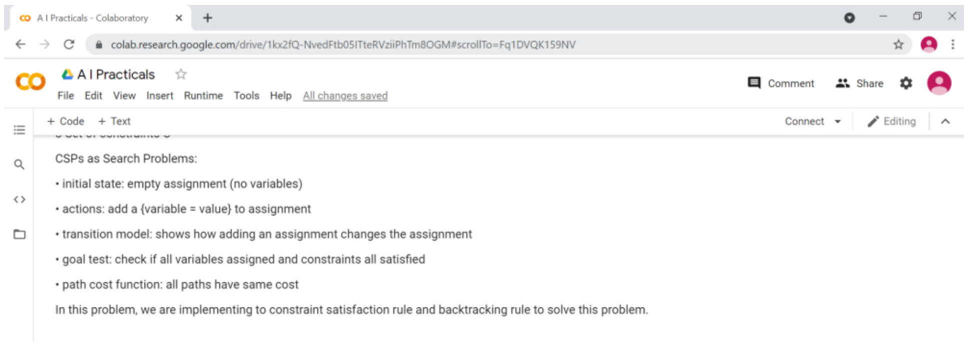
Constraint Satisfaction Problem

1 Set of variables {X1, X2, ..., Xn}

2 Set of domains for each variable {D1, D2, ..., Dn}

3 Set of constraints C

CSPs as Search Problems:



CSPs as Search Problems:

• initial state: empty assignment (no variables)

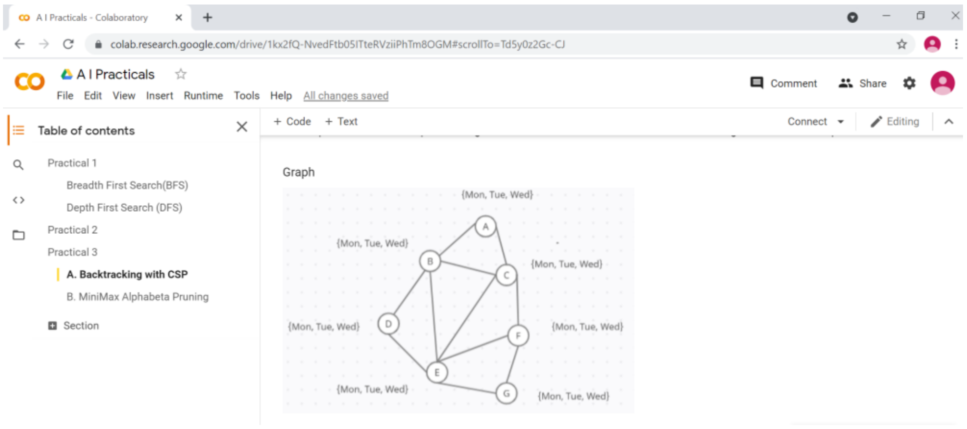
• actions: add a {variable = value} to assignment

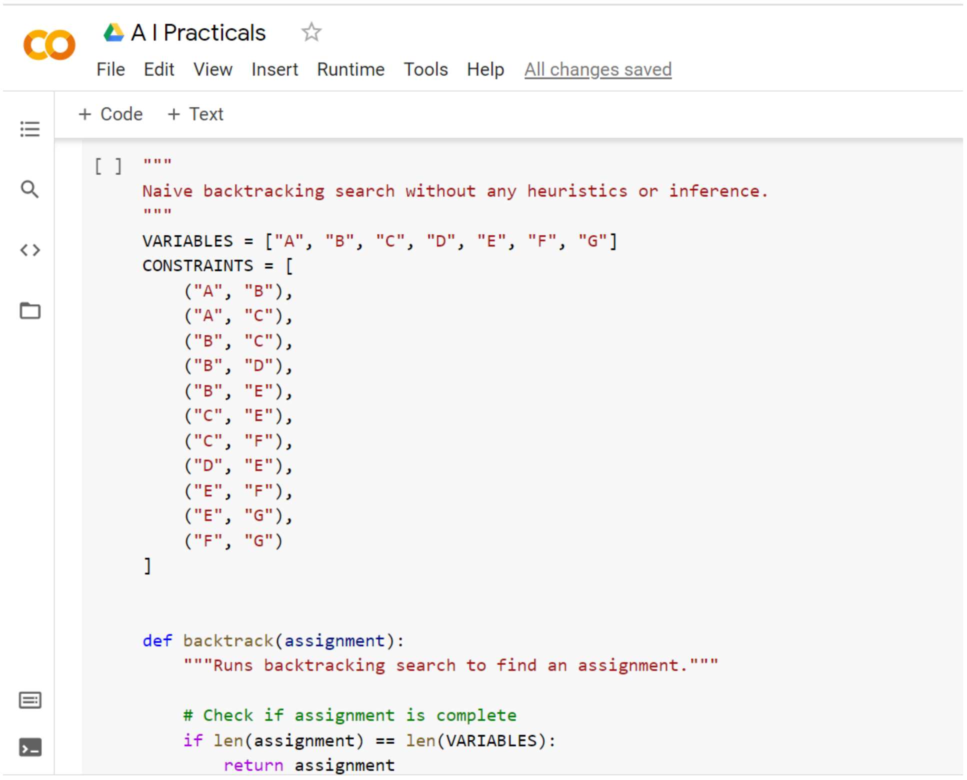
• transition model: shows how adding an assignment changes the assignment

• goal test: check if all variables assigned and constraints all satisfied

• path cost function: all paths have same cost

In this problem, we are implementing to constraint satisfaction rule and backtracking rule to solve this problem.

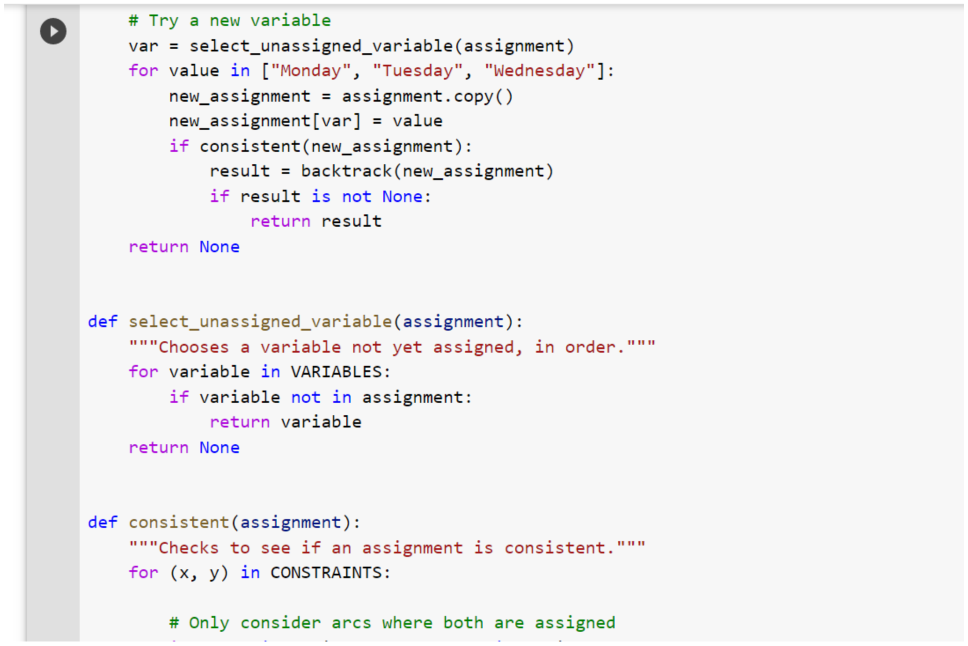

A I Practicals - Colaboratory
colab.research.google.com/drive/1kx2fQ-NvedFtb05lTteRVziiPhTm8OGM#scrollTo=Td5y0z2Gc-CJ
A I Practicals
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
Table of contents
Practical 1
Breadth First Search(BFS)
Depth First Search (DFS)
Practical 2
Practical 3
A. Backtracking with CSP
B. MiniMax Alphabeta Pruning
Section
Graph
{Mon, Tue, Wed}
A
B
C
D
E
F
G



```
"""
Naive backtracking search without any heuristics or inference.
"""
VARIABLES = ["A", "B", "C", "D", "E", "F", "G"]
CONSTRAINTS = [
    ("A", "B"),
    ("A", "C"),
    ("B", "C"),
    ("B", "D"),
    ("B", "E"),
    ("C", "E"),
    ("C", "F"),
    ("D", "E"),
    ("E", "F"),
    ("E", "G"),
    ("F", "G")
]


def backtrack(assignment):
    """Runs backtracking search to find an assignment."""

    # Check if assignment is complete
    if len(assignment) == len(VARIABLES):
        return assignment
```

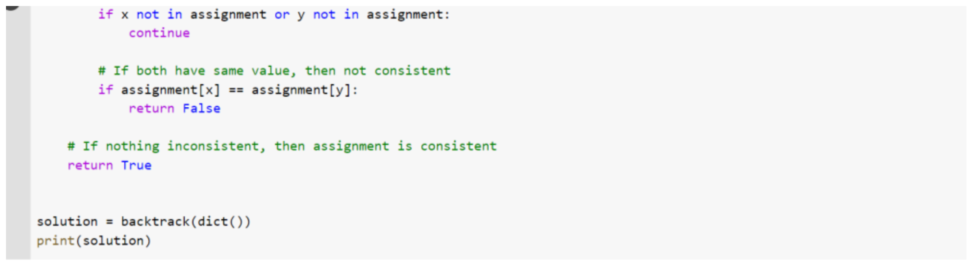


```python
    # Try a new variable
    var = select_unassigned_variable(assignment)
    for value in ["Monday", "Tuesday", "Wednesday"]:
        new_assignment = assignment.copy()
        new_assignment[var] = value
        if consistent(new_assignment):
            result = backtrack(new_assignment)
            if result is not None:
                return result
    return None


def select_unassigned_variable(assignment):
    """Chooses a variable not yet assigned, in order."""
    for variable in VARIABLES:
        if variable not in assignment:
            return variable
    return None


def consistent(assignment):
    """Checks to see if an assignment is consistent."""
    for (x, y) in CONSTRAINTS:

        # Only consider arcs where both are assigned
```



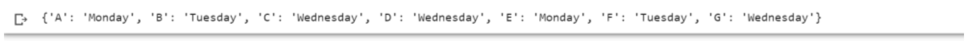
```python
        if x not in assignment or y not in assignment:
            continue

        # If both have same value, then not consistent
        if assignment[x] == assignment[y]:
            return False

    # If nothing inconsistent, then assignment is consistent
    return True


solution = backtrack(dict())
print(solution)
```

Output:



```
{'A': 'Monday', 'B': 'Tuesday', 'C': 'Wednesday', 'D': 'Wednesday', 'E': 'Monday', 'F': 'Tuesday', 'G': 'Wednesday'}
```