Name:Vikas Mane
Roll No: B1921152
Class: BE-B

Assignment 11

Title: Implement goal stack planning for the following configurations from the blocks world,

System Requirements: Java JDK, Visual Studio/Netbeans

Theory:

- Planning is process of determining various actions that often lead to a solution.
- Planning is useful for non-decomposable problems where subgoals often interact.
- Goal Stack Planning (in short GSP) is the one of the simplest planning algorithm that is designed to handle problems having compound goals. And it utilizes STRIP as a formal language for specifying and manipulating the world with which it is working.
- This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order.

Algorithm:

Push the Goal state in to the Stack
Push the individual Predicates of the Goal State into the Stack Loop till the Stack
is empty Pop an element E from the stack
IF E is a Predicate
IF E is True
then
Do Nothing
ELSE
Push the relevant action into the Stack
Push the individual predicates of the Precondition of the action into the Stack
ELSE IF E is an Action
Apply the action to the current
State. Add the action 'a' to the
plan

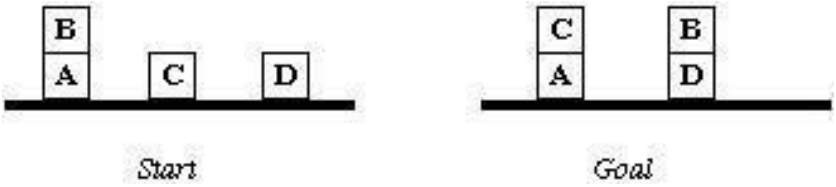
- The Goal Stack Planning Algorithms works will the stack. It starts by pushing the unsatisfied goals into the stack. Then it pushes the individual subgoals into the stack and its pops an element out of the stack. When popping an element out of the stack the element could be either a predicate describing a situation about our world or it could be an action that can be applied to our world under consideration. So based on the kind of element we are popping out from the stack a decision has to be made. If it is a Predicate. Then compares it with the description of the current world, if it is satisfied or is already present in our current situation then there is nothing to do because already its true. On the contrary if the Predicate is not true then we have to select and push relevant action satisfying the predicate to the Stack.
- So after pushing the relevant action into the stack its precondition should also has to be pushed into the stack. In order to apply an operation its precondition has to be satisfied. In other words the present situation of the world should be suitable enough to apply an operation. For that, the preconditions are pushed into the stack once after an action is pushed.

Objective: Student will learn:

I) The Basic Concepts of Goal Stack Planning
II) General Algorithm for Goal Stack Planning
III) Logic of goal stack planning implementation

Input:

Consider the following where wish to proceed from the start to goal state.



No of Blocks : 4

Initial stage : (on b a)^(ontable c)^(ontable a)^(ontable d)^(clear b)^(clear c)^(clear d)^(AE) Final stage : (on c a)^(on b d)^(ontable a)^(ontable d)^(clear c)^(clear b)^(AE)

Output:

Set of actions to be taken:

1. (unstack b d)
2. (stack b d)
3. (pick c)
4. (stack c a)

Result / Conclusion: Hence we have implemented Goal Stack Planning Algorithm

Program:

```
tab = [] result = []
goalList = ["a", "b", "c", "d", "e"]


def parSolution(N): for i in
    range(N):
        if goalList[i] != result[i]: return False
    return True


def Onblock(index, count):

    # break point of recursive call if count ==
    len(goalList)+1:
        return True
    # copy tab of index value to result block =
    tab[index]
    # stack block result.append(block)
    print(result)
    if parSolution(count):
        print("Pushed a result solution ") # remove
        block from tab tab.remove(block)
        Onblock(0, count + 1) else:
        print("result solution not possible, back to the tab") # pop out if no partial
        solution
        result.pop() Onblock(index+1,
        count)


def Ontab(problem):
    # check if everything in stack is on the tab if len(problem) !=
    0:
        tab.append(problem.pop())
```

```
        Ontab(problem)
    # if everything is on the tab the we return true else:
        return True


def goal_stack_planing(problem): # pop problem and put in tab
    Ontab(problem)
    # print index and number of blocks on result stack if Onblock(0, 1):
        print(result)

problem = ["c", "a", "e", "d", "b"] print("Goal Problem")
for k, j in zip(goalList, problem): print(k+"    "+j)
print("\n") goal_stack_planing(problem)
print("\nResult Solution:",result)
```

Output: