**350 Program 4 – MEMO**

**Time/Space Analysis**

|  |  |  |
| --- | --- | --- |
| Methods | Time | Space |
| def nCheck(component) | O(N) | O(N) |
| def canGroup(component, group) | O(N) | O(N) |
| def main() | O(N) | O(N) + O() |

**Test Plan**

|  |  |
| --- | --- |
| Test  Plan |  |
| 1 | Add on another component that it can’t be matched with at the end |
| 2 | 0 0 for the input, so no components |
| 3 | Extra groups that aren’t used - will they be printed if the group is empty or not |
| 4 | Input without enough groups for all the components |

**Source Code**

# Srinivas Simhan

# CIS 350 - Elenbogen: Winter 2018

# Program 4

# Due: 4/18/18

groups = []

constraints = []

ungrouped = []

"""

PRE:

- input a component into the nCheck

POST:

- returns True if the component is able to fit into one of the groups, or else it backtracks

DESCRIPTION:

- nCheck takes an input component and checks if the component is able to be put into a group or not

"""

def nCheck(component):

for i in range(len(groups)):

group = groups[i]

if (canGroup(component, group)):

group.append(component)

if (len(ungrouped) > 0):

nextComponent = ungrouped.pop()

if (nCheck(nextComponent)):

return True

else: # backtrack. undo your actions

popped = group.pop()

else:

return True

ungrouped.append(component)

return False

"""

PRE:

- it takes the component and the group as the input parameters

POST:

- if an ungroupable component is in a group, then it returns it as false

- once all the components from ungroupable are set into groups after checking if it canGroup, then it returns True

DESCRIPTION:

- this method checks if a component has the ability to be grouped

"""

def canGroup(component, group):

constraintsOfComponent = constraints[component - 1]

for ungroupable in constraintsOfComponent:

if ungroupable in group:

return False

return True

"""

PRE:

- global variables set are groups, constraints, and ungrouped

POST:

- it prints out all of the components in their sorted groups (if possible)

DESCRIPTION:

- takes in the number of components and the number of groups available to split the components into

- prints out the groups if able to sort all of them (or prints what it can)

"""

def main():

nk = input().split()

n = int(nk[0]) # number of components

k = int(nk[1]) # number of groups

for i in reversed(range(1,n+1)):

ungrouped.append(i)

for i in range(1, n+1):

# populate noGroup with all the components i cannot be grouped with

noGroupIds = input().split()

noGroupIds.pop() # remove the trailing zero

noGroup = list(map(int, noGroupIds)) # a list of the components (ints) that component i cannot be grouped with

constraints.append(noGroup) # put this list in the constraints

# create the groups (as empty lists)

for i in range(k):

group = []

groups.append(group)

if (nCheck(ungrouped.pop())):

for group in groups:

print(' '.join(map(str, group)))

else:

print("no solution")

if \_\_name\_\_ == "\_\_main\_\_":

main()