In [1]:

```
import random
from pprint import pprint

import numpy as np

from utils import read_data
from constants import FILENAME_EXAMPLE_1, FILENAME_EXAMPLE_4
```

In [2]:

```
#input_data = read_data(FILENAME_EXAMPLE_1)
input_data = read_data(FILENAME_EXAMPLE_4)

days    = input_data['length_of_schedule']
ne      = input_data['number_of_employees']
ns      = input_data['number_of_shifts']
demand  = input_data['temporal_requirements_matrix']
sn      = input_data['shift_name']
ss      = input_data['start_shift']
ls      = input_data['length_shift']
min_ls  = input_data['min_length_of_blocks']
max_ls  = input_data['max_length_of_blocks']
min_do  = input_data['min_days_off']
max_do  = input_data['max_days_off']
min_lw  = input_data['min_length_work_blocks']
max_lw  = input_data['max_length_work_blocks']
nf2     = input_data['nr_sequences_of_length_2']
nf3     = input_data['nr_sequences_of_length_3']
f2      = input_data['not_allowed_shift_sequences_2']
f3      = input_data['not_allowed_shift_sequences_3']
```

In [3]:

```
shifts = ns + 1
day, afternoon, night, dayoff = 1, 2, 3, 4
code = sn + ['-']
```

In [4]:

```
def calculate_result(input_data):
    return [[random.choice(code) for d in range(days)] for e in range(ne)]
```

In [5]:

```
def demand_constraint(result):
    for e in range(ne):
        for d in range(days):
            sum_day, sum_afternoon, sum_night = 0, 0, 0
            if result[e][d] == 'D': sum_day += 1
            elif result[e][d] == 'A': sum_afternoon += 1
            elif result[e][d] == 'N': sum_night += 1
        if sum_day >= demand[0][d] and sum_afternoon >= demand[1][d] and sum_nig
            pass
        else:
            return False
    return True
```

In [6]:

```
def update_demand_constraint(result):
    updated_result = result
    for d in range(days):
        sum_day, sum_afternoon, sum_night = 0, 0, 0
        for e in range(ne):
            if updated_result[e][d] == 'D': sum_day += 1
            elif updated_result[e][d] == 'A': sum_afternoon += 1
            elif updated_result[e][d] == 'N': sum_night += 1

        if sum_day >= demand[0][d]:
            pass
        else:
            updated_result[random.randint(0, ne-1)][d] = 'D'
        if sum_afternoon >= demand[1][d]:
            pass
        else:
            updated_result[random.randint(0, ne-1)][d] = 'A'

        if sum_night >= demand[2][d]:
            pass
        else:
            updated_result[random.randint(0, ne-1)][d] = 'N'
    return updated_result
```

In [7]:

```
#day off constraint
def day_off_constraint(result):
    for e in range(ne):
        count_dayoff = 0
        for d in range(days):
            if result[e][d] == '-': count_dayoff += 1
        if min_do <= count_dayoff <= max_do: pass
        else: return False
    return True
```

In [8]:

```
#working days in a row constraint
def length_work_blocks_constraint(result):
    for e in range(ne):
        count_consecutive = 0
        min_flag, max_flag = False, False
        for d in range(days - 1):
            if result[e][d] != '-' and result[e][d+1] != '-':
                count_consecutive += 1
                if count_consecutive >= min_lw - 1:
                    min_flag = True
                    if count_consecutive <= max_lw - 1:
                        max_flag = True
            else:  count_consecutive = 0
        if min_flag and max_flag: pass
        else: return False
    return True
```

In [9]:

```
#forbidden shifts constraint
def forbidden_constraint2(result):
    if f2 == []: return True
    if f2 != []:
        for e in range(ne):
            for d in range(days - 1):
                for f in f2:
                    if result[e][d] == f[0] and result[e][d+1] == f[1]: return F
        return True
```

In [10]:

```
def forbidden_constraint3(result):
    if f3 == []: return True
    if f3 != []:
        for e in range(ne):
            for d in range(days - 2):
                for f in f3:
                    if result[e][d] == f[0] and result[e][d+1] == f[1] \
                                            and result[e][d+2] == f[2]: return F
        return True
```

In [45]:

```
def solve_problem(input_data):
    result = calculate_result(input_data)
    while True:
        if demand_constraint(result) and day_off_constraint(result) \
                                     and length_work_blocks_constraint(result):
                                     #and forbidden_constraint2(result) \
                                     #and forbidden_constraint3(result):

            break
        else:
            result = calculate_result(input_data)
            if not demand_constraint(result):
                result = update_demand_constraint(result)
            # update day_off constraint
    return result
```

In [41]:

```
def test_contraints(solution):
    return {
            'demand_constraint': demand_constraint(solution),
            'day_off_constraint': day_off_constraint(solution),
            'length_work_blocks_constraint': length_work_blocks_constraint(solut
            'forbidden_constraint2': forbidden_constraint2(solution),
            'forbidden_constraint3': forbidden_constraint3(solution)
            }
```

In [46]:

```
solution = solve_problem(input_data)
solution
```

```
[['-', 'A', 'N', 'D', 'A', 'A', '-'],
 ['A', 'D', 'D', 'A', 'N', 'A', '-'],
 ['D', 'D', 'N', 'A', '-', 'N', 'N'],
 ['A', 'N', 'D', 'N', 'N', '-', 'A'],
 ['D', 'N', 'A', 'D', '-', '-', 'A'],
 ['N', 'D', 'D', 'N', 'N', '-', 'N'],
 ['N', '-', 'D', 'A', 'A', 'N', '-'],
 ['-', 'D', 'D', 'N', 'A', 'N', 'D'],
 ['A', 'D', 'D', 'D', '-', '-', 'D'],
 ['-', 'A', 'N', 'D', 'D', 'D', '-'],
 ['N', '-', 'N', 'D', 'D', 'D', '-'],
 ['D', 'D', 'D', '-', 'A', '-', 'N'],
 ['N', 'A', 'D', '-', '-', '-', 'D']]
```

In [47]:

```
pprint(test_contraints(solution))
```

```
{'day_off_constraint': True,
 'demand_constraint': True,
 'forbidden_constraint2': False,
 'forbidden_constraint3': False,
 'length_work_blocks_constraint': True}
```