In [ ]:

```
import random
from pprint import pprint
import math
from copy import deepcopy
import time
import traceback
import collections


import numpy as np
```

In [ ]:

```
emand_constraint(result, input_data):
olution_demand = solution_to_demand(result, input_data['number_of_shifts'])
or s in range(input_data['number_of_shifts']):
    for d in range(input_data['length_of_schedule']):
        if solution_demand[s][d] < input_data['temporal_requirements_matrix'][s][d
            return False
eturn True

emand_day_constraint(result, input_data):
olution_demand = solution_to_demand(result, input_data['number_of_shifts'])
or d in range(input_data['length_of_schedule']):
    if solution_demand[0][d] < input_data['temporal_requirements_matrix'][0][d]:
        return False
eturn True

emand_afternoon_constraint(result, input_data):
olution_demand = solution_to_demand(result, input_data['number_of_shifts'])
or d in range(input_data['length_of_schedule']):
        if solution_demand[1][d] < input_data['temporal_requirements_matrix'][1][d
            return False
eturn True

emand_night_constraint(result, input_data):
olution_demand = solution_to_demand(result, input_data['number_of_shifts'])
or d in range(input_data['length_of_schedule']):
        if solution_demand[2][d] < input_data['temporal_requirements_matrix'][2][d
            return False
eturn True

hift_to_index(shift):
f shift == 'D': return 0
lif shift == 'A': return 1
lif shift == 'N': return 2
lif shift == '-': return 3
eturn False

in_length_of_shift_constraint(result, input_data):
or e in range(input_data['number_of_employees']):
    counter = collections.Counter(result[e])
    for s in range(input_data['number_of_shifts']):
        if index_to_shift(s) in counter:
```

```
            if input_data['min_length_of_blocks'][s] <= counter[index_to_shift(s)]
                continue
            else:
                return False
        else:
            return False
eturn True

ax_length_of_shift_constraint(result, input_data):
or e in range(input_data['number_of_employees']):
    counter = collections.Counter(result[e])
    for s in range(input_data['number_of_shifts']):
        if index_to_shift(s) in counter:
            if input_data['max_length_of_blocks'][s] >= counter[index_to_shift(s)]
                continue
            else:
                return False
eturn True


ay_off_constraint(result, input_data):
or e in range(input_data['number_of_employees']):
    count_day_off = 0
    for d in range(input_data['length_of_schedule']):
        if result[e][d] == '-': count_day_off += 1
    if count_day_off >= input_data['min_days_off'] and count_day_off <= input_data
        continue
    else:
        return False
eturn True

in_day_off_constraint(result, input_data):
ount_day_off = 0
or e in range(input_data['number_of_employees']):
    count_day_off = 0
    for d in range(input_data['length_of_schedule']):
        if result[e][d] == '-': count_day_off += 1
    if count_day_off >= input_data['min_days_off']:
        continue
    else:
        return False
eturn True
```

```
ax_day_off_constraint(result, input_data):
ount_day_off = 0
or e in range(input_data['number_of_employees']):
    count_day_off = 0
    for d in range(input_data['length_of_schedule']):
        if result[e][d] == '-': count_day_off += 1
    if count_day_off <= input_data['max_days_off']:
        continue
    else:
        return False
eturn True

ength_work_blocks_constraint(result, input_data):
or e in range(input_data['number_of_employees']):
    count_consecutive = 0
    min_flag, max_flag = False, False
    for d in range(input_data['length_of_schedule'] - 1):
        if result[e][d] != '-' and result[e][d+1] != '-':
            count_consecutive += 1
            if count_consecutive >= input_data['min_length_work_blocks'] - 1:
                min_flag = True
                if count_consecutive <= input_data['max_length_work_blocks'] - 1:
                    max_flag = True
        else:  count_consecutive = 0
    if min_flag and max_flag: pass
    else: return False
eturn True

in_length_work_blocks_constraint(result, input_data):
or e in range(input_data['number_of_employees']):
    count_consecutive = 0
    min_flag = False
    for d in range(input_data['length_of_schedule'] - 1):
        if result[e][d] != '-' and result[e][d+1] != '-':
            count_consecutive += 1
            if count_consecutive >= input_data['min_length_work_blocks'] - 1:
                min_flag = True
        else: count_consecutive = 0
    if min_flag: continue
    else: return False
eturn True

ax_length_work_blocks_constraint(result, input_data):
```

```
or e in range(input_data['number_of_employees']):
    count_consecutive = 0
    max_flag = False
    for d in range(input_data['length_of_schedule'] - 1):
        if result[e][d] != '-' and result[e][d+1] != '-':
            count_consecutive += 1
            if count_consecutive >= input_data['min_length_work_blocks'] - 1:
                if count_consecutive <= input_data['max_length_work_blocks'] - 1:
                    max_flag = True
        else: count_consecutive = 0
    if max_flag: continue
    else: return False
eturn True

orbidden_constraint2(result, input_data):
f input_data['not_allowed_shift_sequences_2'] == []: return True
f input_data['not_allowed_shift_sequences_2'] != []:
    for e in range(input_data['number_of_employees']):
        for d in range(input_data['length_of_schedule'] - 1):
            for f in input_data['not_allowed_shift_sequences_2']:
                if result[e][d] == f[0] and result[e][d+1] == f[1]: return False
    return True

orbidden_constraint3(result, input_data):
f input_data['not_allowed_shift_sequences_3'] == []: return True
f input_data['not_allowed_shift_sequences_3'] != []:
    for e in range(input_data['number_of_employees']):
        for d in range(input_data['length_of_schedule'] - 2):
            for f in input_data['not_allowed_shift_sequences_3']:
                if result[e][d] == f[0] and result[e][d+1] == f[1]
    return True
```

In [ ]:

```
def calculate_optimization_sum(b, l):
    sum_ = 0
    l_ = 5
    for i in range(b):
        sum_ += (l[i] - l_)**2
    return sum_

def optimize_sum(result, input_data):
    updated_result = deepcopy(result)
    length_working_block = 1
    b = 0
    l = []

    for e in range(input_data['number_of_employees']):
        length_working_block = 1
        for d in range(input_data['length_of_schedule'] - 1):
            if updated_result[e][d] != '-' and updated_result[e][d+1] != '-':
                length_working_block += 1
            else:
                if d+1 < input_data['length_of_schedule'] - 1:
                    l.append(length_working_block)
                    length_working_block = 1
                    b += 1
        b += 1
        l.append(length_working_block)
    return b, l

def eval_solution(solution, input_data):
    score = 0
    c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12 = (
                        demand_day_constraint,
                        demand_afternoon_constraint,
                        demand_night_constraint,
                        min_day_off_constraint,
                        max_day_off_constraint,
                        min_length_work_blocks_constraint,
                        max_length_work_blocks_constraint,
                        forbidden_constraint2,
                        forbidden_constraint3,
                        min_length_of_shift_constraint,
                        max_length_of_shift_constraint,
```

```
                        optimize_sum
                        )

    if c1(solution, input_data): score += 15
    if c2(solution, input_data): score += 10
    if c3(solution, input_data): score += 10
    if c4(solution, input_data): score += 15
    if c5(solution, input_data): score += 10
    if c6(solution, input_data): score += 10
    if c7(solution, input_data): score += 10
    if c8(solution, input_data): score += 10
    if c9(solution, input_data): score += 10
    if c10(solution, input_data): score += 10
    if c11(solution, input_data): score += 10

    score -= calculate_optimization_sum(*c12(solution, input_data))

    return score

def eval_solution_5(solution, input_data):
    score = 0
    c1, c2, c3, c4, c5 = (demand_constraint,
                        day_off_constraint,
                        length_work_blocks_constraint,
                        forbidden_constraint2,
                        forbidden_constraint3)

    if c1(solution, input_data): score += 30
    if c2(solution, input_data): score += 20
    if c3(solution, input_data): score += 20
    if c4(solution, input_data): score += 15
    if c5(solution, input_data): score += 15

    return score

def eval_solution_2(solution, input_data):
    score = 0
    c1, c2, c3, c4, c5 = (demand_constraint,
                        day_off_constraint,
                        length_work_blocks_constraint,
                        forbidden_constraint2,
                        forbidden_constraint3)
```

```
    if c1(solution, input_data): score += 20
    if c2(solution, input_data): score += 20
    if c3(solution, input_data): score += 20
    if c4(solution, input_data): score += 20
    if c5(solution, input_data): score += 20

    return score

def eval_solution_3(solution, input_data):
    score = 0
    c1, c2, c3, c4, c5 = (demand_constraint,
                          day_off_constraint,
                          length_work_blocks_constraint,
                          forbidden_constraint2,
                          forbidden_constraint3)
    if c1(solution, input_data): score += 50
    if c2(solution, input_data): score += 15
    if c3(solution, input_data): score += 15
    if c4(solution, input_data): score += 10
    if c5(solution, input_data): score += 10

    return score

def eval_solution_4(solution, input_data):
    score = 0
    c1, c2, c3, c4, c5 = (demand_constraint,
                          day_off_constraint,
                          length_work_blocks_constraint,
                          forbidden_constraint2,
                          forbidden_constraint3)
    if c1(solution, input_data): score += 60
    if c2(solution, input_data): score += 10
    if c3(solution, input_data): score += 10
    if c4(solution, input_data): score += 10
    if c5(solution, input_data): score += 10

    return score

def report_solution(solution, input_data):
    return {
        'demand_constraint': demand_constraint(solution, input_data),
        'demand_day_constraint': demand_day_constraint(solution, input_data),
        'demand_afternoon_constraint': demand_afternoon_constraint(solution, inp
```

```
        'demand_night_constraint': demand_night_constraint(solution, input_data)
        'day_off_constraint': day_off_constraint(solution, input_data),
        'min_day_off_constraint': min_day_off_constraint(solution, input_data),
        'max_day_off_constraint': max_day_off_constraint(solution, input_data),
        'length_work_blocks_constraint': length_work_blocks_constraint(solution,
        'min_length_work_blocks_constraint': min_length_work_blocks_constraint(s
        'max_length_work_blocks_constraint': max_length_work_blocks_constraint(s
        'forbidden_constraint2': forbidden_constraint2(solution, input_data),
        'forbidden_constraint3': forbidden_constraint3(solution, input_data),
        'min_length_of_shift_constraint': min_length_of_shift_constraint(solutic
        'max_length_of_shift_constraint': max_length_of_shift_constraint(solutic
        'optimization_score': calculate_optimization_sum(optimize_sum(solution,
    }
```

In [ ]:

```
def transpose_matrix(matrix):
    transposed_matrix = [[matrix[j][i] for j in range(len(matrix))] for i in ran
    return transposed_matrix

def solution_to_demand(solution, numshifts):
    demand = [[0 for i in range(len(solution[0]))] for i in range(numshifts)]
    for d in range(len(solution[0])):
        for e in range(len(solution)):
            if solution[e][d] == 'D': demand[0][d] += 1
            elif solution[e][d] == 'A': demand[1][d] += 1
            elif solution[e][d] == 'N': demand[2][d] += 1
    return demand

def count_sums_per_day_solution(solution_day):
    sum_day, sum_afternoon, sum_night = 0, 0, 0
    for e in range(len(solution_day)):
        if solution_day[e] == 'D': sum_day += 1
        elif solution_day[e] == 'A': sum_afternoon += 1
        elif solution_day[e] == 'N': sum_night += 1
    return [sum_day, sum_afternoon, sum_night]

def index_to_shift(index):
    if index == 0: return 'D'
    elif index == 1: return 'A'
    elif index == 2: return 'N'
    elif index == 3: return '-'
    return False

def adapt_solution_day(solution_day, demand_day):
    solution = deepcopy(solution_day)
    solution_shifts_sums = count_sums_per_day_solution(solution_day)
    available_shifts, demand_shifts = [], []


    for i in range(len(demand_day)):
        if demand_day[i] > solution_shifts_sums[i]:
            for j in range(demand_day[i] - solution_shifts_sums[i]):
                demand_shifts.append(index_to_shift(i))
        elif demand_day[i] < solution_shifts_sums[i]:
            available_shifts.append(index_to_shift(i))
    for shift in solution:
```

```
        if shift == '-':
            available_shifts.append('-')


    for shift in demand_shifts:
        for i in range(len(solution)):
            if solution[i] in available_shifts:
                available_shifts.remove(solution[i])
                solution[i] = shift
                break

    return solution

def move_all_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not demand_constraint(updated_result, input_data):
        solution = []
        demand = transpose_matrix(input_data['temporal_requirements_matrix'])
        for d in range(len(input_data['temporal_requirements_matrix'][0])):
            solution.append(adapt_solution_day(transpose_matrix(result)[:][d], c
        updated_result = transpose_matrix(solution)
    return updated_result

def move_one_day_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not demand_constraint(updated_result, input_data):
        solution = transpose_matrix(updated_result)
        demand = transpose_matrix(input_data['temporal_requirements_matrix'])
        max_days = len(input_data['temporal_requirements_matrix'][0]) - 1
        random_day = random.randint(0, max_days)
        solution[:][random_day] = adapt_solution_day(transpose_matrix(result)[:]
        solution = transpose_matrix(solution)
        updated_result = solution
    return updated_result

def move_day_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not demand_constraint(updated_result, input_data):
        for d in range(input_data['length_of_schedule']):
            sum_day = 0
            for e in range(input_data['number_of_employees']):
                if updated_result[e][d] == 'D': sum_day += 1
            if sum_day >= input_data['temporal_requirements_matrix'][0][d]: pass
```

```
            else:
                updated_result[random.randint(0, input_data['number_of_employees
                break
    return updated_result

def move_afternoon_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not demand_constraint(updated_result, input_data):
        for d in range(input_data['length_of_schedule']):
            sum_afternoon = 0
            for e in range(input_data['number_of_employees']):
                if updated_result[e][d] == 'A': sum_afternoon += 1
            if sum_afternoon >= input_data['temporal_requirements_matrix'][0][d]
            else:
                updated_result[random.randint(0, input_data['number_of_employees
                break
    return updated_result

def move_night_demand_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not demand_constraint(updated_result, input_data):
        if input_data['number_of_shifts'] < 3 : return updated_result
        for d in range(input_data['length_of_schedule']):
            sum_night = 0
            for e in range(input_data['number_of_employees']):
                if updated_result[e][d] == 'N': sum_night += 1
            if sum_night >= input_data['temporal_requirements_matrix'][0][d]: pa
            else:
                updated_result[random.randint(0, input_data['number_of_employees
                break
    return updated_result

def move_all_day_off_constraint(result, input_data):
    count_day_off, min_days_off, max_days_off = 0,input_data['min_days_off'],inp
    updated_result = deepcopy(result)
    if not day_off_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        for e in range(input_data['number_of_employees']):
            count_day_off = 0
            for d in range(input_data['length_of_schedule']):
                if updated_result[e][d] == '-': count_day_off += 1
            if count_day_off >= input_data['min_days_off']: pass
            else:
```

```
                while count_day_off <= min_days_off:
                    updated_result[e][random.randint(0, input_data['length_of_sc
                    count_day_off += 1
            if count_day_off <= input_data['max_days_off']: pass
            else:
                while count_day_off >= max_days_off:
                    updated_result[e][random.randint(0, input_data['length_of_sc
                    count_day_off -= 1
    return updated_result

def move_min_day_off_constraint(result, input_data):
    count_day_off, min_days_off, max_days_off = 0,input_data['min_days_off'],inp
    updated_result = deepcopy(result)
    if not day_off_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        for e in range(input_data['number_of_employees']):
            count_day_off = 0
            for d in range(input_data['length_of_schedule']):
                if updated_result[e][d] == '-': count_day_off += 1
            if count_day_off >= input_data['min_days_off']: pass
            else:
                while count_day_off <= min_days_off:
                    updated_result[e][random.randint(0, input_data['length_of_sc
                    count_day_off += 1
    return updated_result

def move_max_day_off_constraint(result, input_data):
    count_day_off, min_days_off, max_days_off = 0,input_data['min_days_off'],inp
    updated_result = deepcopy(result)
    if not day_off_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        for e in range(input_data['number_of_employees']):
            count_day_off = 0
            for d in range(input_data['length_of_schedule']):
                if updated_result[e][d] == '-': count_day_off += 1
            if count_day_off <= input_data['max_days_off']: pass
            else:
                while count_day_off >= max_days_off:
                    updated_result[e][random.randint(0, input_data['length_of_sc
                    count_day_off -= 1
    return updated_result

def move_all_length_work_blocks_constraint(result, input_data):
```

```
    updated_result = deepcopy(result)
    if not length_work_blocks_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        for e in range(input_data['number_of_employees']):
            count_consecutive = 0
            min_flag, max_flag = False, False
            for d in range(input_data['length_of_schedule'] - 1):
                if updated_result[e][d] != '-' and updated_result[e][d+1] != '-'
                    count_consecutive += 1
                    if count_consecutive >= input_data['min_length_work_blocks']
                        min_flag = True
                        if count_consecutive <= input_data['max_length_work_bloc
                            max_flag = True
                else:  count_consecutive = 0

            if min_flag: pass
            else:
                count_concecutive = 0
                for d in range(input_data['length_of_schedule']):
                    if updated_result[e][d] == '-' and count_concecutive <= inpu
                        updated_result[e][d] = random.choice(list(set(code) - se
                    count_concecutive += 1
            if max_flag: pass
            else: updated_result[e][random.randint(0, input_data['length_of_sche

    return updated_result

def move_min_length_work_blocks_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not length_work_blocks_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']

        for e in range(input_data['number_of_employees']):
            count_consecutive = 0
            min_flag, max_flag = False, False
            for d in range(input_data['length_of_schedule'] - 1):
                if updated_result[e][d] != '-' and updated_result[e][d+1] != '-'
                    count_consecutive += 1
                    if count_consecutive >= input_data['min_length_work_blocks']
                        min_flag = True
                        if count_consecutive <= input_data['max_length_work_bloc
                            max_flag = True
                else:  count_consecutive = 0
```

```
            if min_flag: pass
            else:
                count_concecutive = 0
                for d in range(input_data['length_of_schedule']):
                    if updated_result[e][d] == '-' and count_concecutive <= inpu
                        updated_result[e][d] = random.choice(list(set(code) - se
                        return updated_result
                    count_concecutive += 1
    return updated_result

def move_max_length_work_blocks_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not length_work_blocks_constraint(updated_result, input_data):
        code = input_data['shift_name'] + ['-']

        for e in range(input_data['number_of_employees']):
            count_consecutive = 0
            min_flag, max_flag = False, False
            for d in range(input_data['length_of_schedule'] - 1):
                if updated_result[e][d] != '-' and updated_result[e][d+1] != '-'
                    count_consecutive += 1
                    if count_consecutive >= input_data['min_length_work_blocks']
                        min_flag = True
                        if count_consecutive <= input_data['max_length_work_bloc
                            max_flag = True
                else:  count_consecutive = 0

            if min_flag: pass
            else:
                count_concecutive = 0
                for d in range(input_data['length_of_schedule']):
                    if updated_result[e][d] == '-' and count_concecutive <= inpu
                        pass
                    count_concecutive += 1
            if max_flag: pass
            else:
                updated_result[e][random.randint(0, input_data['length_of_schedu
                return updated_result

    return updated_result

def move_all_0_forbidden_constraint2(result, input_data):
```

```
    updated_result = deepcopy(result)
    if not forbidden_constraint2(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_2'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_2'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 1):
                    for f in input_data['not_allowed_shift_sequences_2']:
                        if updated_result[e][d] == f[0] and updated_result[e][d+
                            for c in code:
                                if [f[0], c] not in input_data['not_allowed_shi
                                    updated_result[e][d+1] = c
                                    break
    return updated_result

def move_all_1_forbidden_constraint2(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint2(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_2'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_2'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 1):
                    for f in input_data['not_allowed_shift_sequences_2']:
                        if updated_result[e][d] == f[0] and updated_result[e][d+
                            for c in code:
                                if [c, f[1]] not in input_data['not_allowed_shi
                                    updated_result[e][d] = c
                                    break
    return updated_result

def move_0_forbidden_constraint2(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint2(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        move_flage = False
        if input_data['not_allowed_shift_sequences_2'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_2'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 1):
                    for f in input_data['not_allowed_shift_sequences_2']:
                        if updated_result[e][d] == f[0] and updated_result[e][d+
                            for c in code:
```

```
                                if [f[0], c] not in input_data['not_allowed_shi
                                    updated_result[e][d+1] = c
                                    return updated_result
    return updated_result

def move_1_forbidden_constraint2(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint2(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_2'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_2'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 1):
                    for f in input_data['not_allowed_shift_sequences_2']:
                        if updated_result[e][d] == f[0] and updated_result[e][d+
                            for c in code:
                                if [c, f[1]] not in input_data['not_allowed_shi
                                    updated_result[e][d] = c
                                    return updated_result
    return updated_result

def move_all_0_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [c, f[1], f[2]] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    break
    return updated_result

def move_0_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
```

```
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [c, f[1], f[2]] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    return updated_result

    return updated_result

def move_all_1_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [f[0], c, f[2]] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    break

    return updated_result

def move_all_2_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [f[0], f[1], c] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    break
```

```
    return updated_result

def move_1_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [f[0], c, f[2]] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    return updated_result

    return updated_result

def move_2_forbidden_constraint3(result, input_data):
    updated_result = deepcopy(result)
    if not forbidden_constraint3(updated_result, input_data):
        code = input_data['shift_name'] + ['-']
        if input_data['not_allowed_shift_sequences_3'] == []: return updated_res
        if input_data['not_allowed_shift_sequences_3'] != []:
            for e in range(input_data['number_of_employees']):
                for d in range(input_data['length_of_schedule'] - 2):
                    for f in input_data['not_allowed_shift_sequences_3']:
                        if result[e][d] == f[0] and result[e][d+1] == f[1]
                            for c in code:
                                if [f[0], f[1], c] not in input_data['not_allowe
                                    updated_result[e][d] = c
                                    return updated_result

    return updated_result

def replace(list_, element_1, element_2):
    result = deepcopy(list_)
    for i in range(len(result)):
        if result[i] == element_1:
            result[i] = element_2
            break
    return result
```

```
def update_shift_employee_min(employee, shift, times):
    result = deepcopy(employee)
    for i in range(times):
        counter = collections.Counter(result)
        max_shifts = counter.most_common(1)[0][0]
        if '-' in result:
            max_shifts = '-'
        else:
            max_shifts = counter.most_common(1)[0][0]
        result = replace(result, max_shifts, shift)
    return result

def move_min_length_of_shift_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not min_length_of_shift_constraint(result, input_data):
        for e in range(input_data['number_of_employees']):
            counter = collections.Counter(result[e])
            for s in range(input_data['number_of_shifts']):
                if index_to_shift(s) in counter:
                    if input_data['min_length_of_blocks'][s] > counter[index_to_
                        updated_result[e] = update_shift_employee_min(updated_re
                                                                      index_to_s
                                                                     input_data[
                else:
                    updated_result[e] = update_shift_employee_min(updated_result
                                                                  index_to_shi
                                                                  input_data[
    return updated_result

def update_shift_employee_max(employee, shift, times):
    result = deepcopy(employee)
    for i in range(times):
        result = replace(result, shift, '-')
    return result

def move_max_length_of_shift_constraint(result, input_data):
    updated_result = deepcopy(result)
    if not max_length_of_shift_constraint(result, input_data):
        for e in range(input_data['number_of_employees']):
            counter = collections.Counter(result[e])
            for s in range(input_data['number_of_shifts']):
                if index_to_shift(s) in counter:
```

```
                if input_data['max_length_of_blocks'][s] < counter[index_to_
                    updated_result[e] = update_shift_employee_max(updated_re
                                                                  index_to_s
                                                                  counter[inc
    return updated_result
```

In [ ]:

In [ ]:

```
def generate_random_solution(input_data):
    code = input_data['shift_name'] + ['-']
    return [[random.choice(code) for d in range(input_data['length_of_schedule']
```

In [ ]:

```
def exp_probability(solution, eval_function, neighbor, T, input_data):
    return math.exp((eval_function(neighbor, input_data) - eval_function(solutio
```

In [ ]:

```
def simulated_annealing(input_data, eval_function, T_max, r, termination_conditi
    tc = 0
    hc = 0
    T = T_max

    solution = generate_random_solution(input_data)
    best_solution = deepcopy(solution)

    while hc < halting_condition:
        while tc < termination_condition:
            if eval_function(solution, input_data) == 100:
                print(f'params: T_max={T_max}, r={r}, termination_condition={ter
                return solution, best_solution

            moves = [
                move_day_demand_constraint,
                move_one_day_demand_constraint,
                move_afternoon_demand_constraint,
                move_night_demand_constraint,
                move_min_day_off_constraint,
                move_max_day_off_constraint,
                move_min_length_work_blocks_constraint,
                move_max_length_work_blocks_constraint,
                move_0_forbidden_constraint2,
                move_1_forbidden_constraint2,
                move_0_forbidden_constraint3,
                move_1_forbidden_constraint3,
                move_2_forbidden_constraint3,
                move_all_demand_constraint,
                move_all_day_off_constraint,
                move_all_length_work_blocks_constraint,
                move_all_0_forbidden_constraint2,
                move_all_1_forbidden_constraint2,
                move_all_0_forbidden_constraint3,
                move_all_1_forbidden_constraint3,
                move_all_2_forbidden_constraint3,
                move_min_length_of_shift_constraint,
                move_max_length_of_shift_constraint,
            ]

            move_choice = random.choice(moves)
            move = move_choice(solution, input_data)
```

```
            if eval_function(solution, input_data) < eval_function(move, input_d
            elif random.uniform(0, 1) < exp_probability(solution, eval_function,

            if eval_function(solution, input_data) > eval_function(best_solution
                best_solution = deepcopy(solution)
            tc += 1

        T *= r
        hc += 1

    return "Not satified in the given time", solution, best_solution
```

In [ ]:

```
def test_SA(T_max, r, termination_condition, halting_condition, show_non_optimal
    best_solutions = []
    for example in get_examples():
        if show_non_optimal:
            print(example)
        input_data = read_data(example)
        solution = generate_random_solution(input_data)
        solution = simulated_annealing(input_data, T_max, r, termination_conditi
        if "Not satified in the given time" not in solution:
            if not show_non_optimal:
                print(example)
            solution, best_solution = solution
            best_solutions.append((example, best_solution, eval_function(best_so
            print(f'Best solution: {eval_function(best_solution, input_data)}')
            print(f'Last solution: {eval_function(solution, input_data)}')
            print('passed')
            print()
        else:
            if show_non_optimal:
                message, solution, best_solution = solution
                best_solutions.append((example, best_solution, eval_function(bes
                print(f'{message}. Best solution found has a score of: {eval_fun
                print(f'Last solution: {eval_function(solution, input_data)}')
                print('passed')
                print()
    return best_solutions


def test_SA_per_example(example, eval_function, T_max, r, termination_condition,
    if show_non_optimal:
        print(example)
    input_data = read_data(example)
    solution = simulated_annealing(input_data, eval_function, T_max, r, terminat
    best_solution = deepcopy(solution)
    if "Not satified in the given time" not in solution:
        if not show_non_optimal:
            print(example)
        solution, best_solution = solution
        print(f'Best solution: {eval_function(best_solution, input_data)}')
        print(f'Last solution: {eval_function(solution, input_data)}')
        print('passed')
```

```
        print()
    else:
        message, solution, best_solution = solution
        if show_non_optimal:
            print(f'{message}. Best solution found has a score of: {eval_functic
            print(f'Last solution: {eval_function(solution, input_data)}')
            print('passed')
            print()
    return best_solution
```

In [ ]:

```
def read_data(filename):
    input_data = {}
    #Length of the schedule
    length_of_schedule = 0

    #Number of Employees
    number_of_employees = 0

    ##Number of Shifts
    number_of_shifts = 0

    # Temporal Requirements Matrix Shifts per Days
    temporal_requirements_matrix = []

    #ShiftName, Start, Length, MinlengthOfBlocks, MaxLengthOfBlocks
    shift_name, start_shift, length_shift, min_length_of_blocks, max_length_of_b

    # Minimum and maximum length of days-off blocks
    min_days_off = 0
    max_days_off = 0

    # Minimum and maximum length of work blocks
    min_length_work_blocks = 0
    max_length_work_blocks = 0

    # Number of not allowed shift sequences: NrSequencesOfLength2, NrSequencesOf
    nr_sequences_of_length_2 = 0
    nr_sequences_of_length_3 = 0

    # Not allowed shift sequences
    not_allowed_shif_sequences = [
        ['N', 'D'], ['N', 'A'], ['A', 'D']
    ]
    with open(filename, 'r') as f:
        lines = iter(f.readlines())
        for line in lines:
            if "#Length of the schedule" in line:
                length_of_schedule = int(next(lines))
            if "#Number of Employees" in line:
                number_of_employees = int(next(lines))
            if "##Number of Shifts" in line:
```

```
            number_of_shifts = int(next(lines))
        if "# Temporal Requirements Matrix" in line:
            ns = number_of_shifts
            temporal_requirements_matrix = []
            for i in range(number_of_shifts):
                temporal_requirements_matrix.append(list(map(int, next(lines
        if "#ShiftName" in line:
            ns = number_of_shifts
            shift_name, start_shift, length_shift, min_length_of_blocks, max
            for i in range(number_of_shifts):
                shift_name[i], start_shift[i], length_shift[i], min_length_o
                start_shift[i], length_shift[i] = list(map(int, start_shift[
            min_length_of_blocks, max_length_of_blocks = [int(x) for x in mi
        if "# Minimum and maximum length of days-off blocks" in line:
            min_days_off, max_days_off = list(map(int, next(lines).split()))
        if "# Minimum and maximum length of work blocks" in line:
            min_length_work_blocks, max_length_work_blocks = list(map(int, n
        if "# Number of not allowed shift sequences: NrSequencesOfLength2, N
            nr_sequences_of_length_2, nr_sequences_of_length_3 = list(map(in
        if "# Not allowed shift sequences" in line:
            not_allowed_shift_sequences_2, not_allowed_shift_sequences_3 = [
            for i in range(nr_sequences_of_length_2):
                not_allowed_shift_sequences_2.append(next(lines).split('\n'))
            for i in range(nr_sequences_of_length_3):
                not_allowed_shift_sequences_3.append(next(lines).split('\n'))

input_data = {
    'length_of_schedule': length_of_schedule,
    'number_of_employees': number_of_employees,
    'number_of_shifts': number_of_shifts,
    'temporal_requirements_matrix': temporal_requirements_matrix,
    'shift_name': shift_name,
    'start_shift': start_shift,
    'length_shift': length_shift,
    'min_length_of_blocks': min_length_of_blocks,
    'max_length_of_blocks': max_length_of_blocks,
    'min_days_off': min_days_off,
    'max_days_off': max_days_off,
    'min_length_work_blocks': min_length_work_blocks,
    'max_length_work_blocks': max_length_work_blocks,
    'nr_sequences_of_length_2': nr_sequences_of_length_2,
    'nr_sequences_of_length_3': nr_sequences_of_length_3,
    'not_allowed_shift_sequences_2': not_allowed_shift_sequences_2,
```

```
        'not_allowed_shift_sequences_3': not_allowed_shift_sequences_3
    }

    return input_data

def get_examples():
    RWS_INSTANCES = 'rws_instances/'
    EXAMPLE = 'Example'
    filenames = []

    for i in range(1,21):
        filenames.append(RWS_INSTANCES + EXAMPLE + str(i) + '.txt')
    return filenames
```