Contents

1 Basic Test Results

```
Starting tests...
Wed Jun  3 23:11:50 IDT 2020
7a6d73e2c9cb6d7f0557c31fcd795e24f2dbcd9c  -


Archive:  /tmp/bodek._DNrs4/intro2cs2/ex8/ofirm57/presubmission/submission
  inflating: src/nonogram.py


Running presubmit code tests...
5 passed tests out of 5 in test set named 'presubmit'.
result_code    presubmit    5    1
Done running presubmit code tests

Finished running the presubmit tests

Additional notes:

The presubmit tests check only for the existence of the correct function names.
Make sure to thoroughly test your code.

```

2 nonogram.py

```python
##################################################################
# FILE : ex8.py
# WRITER :ofir  , ofirm57 , 205660731
# EXERCISE : intro2cs2 ex8 2020
##################################################################
import math

BLACK = 1
WHITE = 0
UNKNOWN = -1


def helper_get_row_variations(row, blocks, tmp_lst, row_lst, index,
                              first_t=True):
    """function that return all the option of row by the block
    :param row: list row
    :param blocks: list of block
    :param tmp_lst: change list
    :param row_lst: the return list
    :param index: the index
    :param first_t: true if u first time entar the function    """

    length = len(row)
    if first_t:
        found_common_cases = common_cases(row, blocks)
        if found_common_cases:
            answer = get_intersection_row([row, found_common_cases])
            if not rows_with_unknown_check([answer]):
                return [found_common_cases]

    if sum(blocks) - sum(tmp_lst) > len(row) - len(tmp_lst):
        return
    if tmp_lst.count(BLACK) > sum(blocks):
        return
    if tmp_lst.count(WHITE) > length - sum(blocks):
        return
    if index > 1 and not row_check(tmp_lst, blocks, index, length):
        return
    if len(row_lst) == count_row_variations(length, blocks):
        return row_lst
    if len(tmp_lst) == length:
        if tmp_lst.count(BLACK) != sum(blocks):
            return
        return row_lst.append(tmp_lst)
    if row[index] == UNKNOWN:
        tmp_lst.append(BLACK)
        helper_get_row_variations(row, blocks, tmp_lst[:], row_lst, index + 1,
                                  False)
        tmp_lst[-1] = WHITE
        helper_get_row_variations(row, blocks, tmp_lst[:], row_lst, index + 1,
                                  False)
    else:
        tmp_lst.append(row[index])
        helper_get_row_variations(row, blocks, tmp_lst[:], row_lst, index + 1,
                                  False)
    return row_lst


def common_cases(row, block):
```

```python
    """recive row and block and return common_cases  """

    row_long = len(row)
    num_of_black = sum(block)
    block_length = len(block)
    lst = []
    if num_of_black + len(block) - 1 == row_long:
        for i, value in enumerate(block):
            lst.extend([BLACK] * value)
            if i != block_length - 1:
                lst.append(WHITE)
        return lst
    if not block: ### zero row
        return lst.extend([WHITE] * row_long)
    if block_length == 1 and num_of_black == row_long:  ## all black
        return lst.extend([BLACK] * row_long)


def row_check(tmp_lst, blocks, index, lenght):  # func 1
    """finction that check the row by the blocks, index and lenght """
    counter = 0
    b_num = 0
    for i, v in enumerate(tmp_lst):
        if i != index - 1: # check if first
            if v == BLACK:
                counter += 1
                if tmp_lst[i + 1] == 0:
                    if blocks[b_num] != counter:
                        return False
                    else:
                        counter = 0
                        b_num += 1
        else:
            if v == BLACK:
                counter += 1
                if blocks[b_num] != counter and not i < lenght - 1:
                    return False
    return True


def get_row_variations(row, blocks): #func 1 real
    """function that return option fo row by the blocks information """
    return helper_get_row_variations(row, blocks, [], [], 0, True)


#######################################################################^1

def make_list_of_index(rows):
    """function that replicates the list of lists"""
    lenght = len(rows[0])
    option_num = len(rows)
    tmp_lst = []
    list_of_index = []
    for i in range(lenght):  # line
        if i != 0:
            list_of_index.append(tmp_lst)
            tmp_lst = []
        for j in range(option_num):  # row external
            tmp_lst.append(rows[j][i])
    list_of_index.append(tmp_lst)
    return list_of_index


def get_intersection_row(rows):  # func 2
    """ recive list of list (option for one row) and return the option for
    the row
    i choose that this func return only what is 100% right"""
    the_rows = make_list_of_index(rows)
```

```python
    result = []
    for i in range(len(the_rows)):
        if all(x == the_rows[i][0] for x in the_rows[i]): ###IF ALL THE SAME
            result.append(the_rows[i][0])
            continue
        if 0 in the_rows[i] and (1 or -1) in the_rows[i]:
            result.append(UNKNOWN)
            continue
        result.append(UNKNOWN)
    return result

##########################################################################^2


def solve_easy_nonogram(constraints):
    """make the firs check and then cotinue to check if it not redey"""
    bord = find_start_bord(constraints)
    return helper_func3(constraints, bord)


def find_start_bord(constraints):
    """
    A function that makes a board according to constraints Length
    """
    line_num = len(constraints[1])
    row_num = len(constraints[0])
    first_bord = []
    for i in range(row_num):
        tmp = []
        tmp.extend([UNKNOWN] * line_num)
        first_bord.append(tmp)
    return first_bord


def helper_func3(constraints, start_bord):
    """
    A function that takes out a resolved or partially resolved board
    With the help of auxiliary functions like first_line_solution and
    find_start_bord
    """
    line_solution = first_line_solution(constraints, start_bord)
    row_solution = first_row_solution(constraints, line_solution)
    if row_solution == line_solution:
        return row_solution
    make_2_check_loops = 2
    while make_2_check_loops:
        tmp_lst = []
        for i in range(len(row_solution)):
            end_com_result = compre_list(row_solution[i], line_solution[i])
            tmp_lst.append(end_com_result)
            if make_2_check_loops == 0:
                return tmp_lst
        row_solution = first_row_solution(constraints, tmp_lst)
        line_solution = first_line_solution(constraints, row_solution)
        if row_solution != line_solution:
            make_2_check_loops = 2
        else:
            make_2_check_loops -= 1
            if make_2_check_loops == 0:
                break
    return row_solution


def first_row_solution(constraints, list_of_rows):
    """A function that sends a line for review"""
    line_number = len(constraints[1])
    check_index = 0
    return check_row_and_line(check_index, constraints, line_number, [],
```

```
                                list_of_rows)


def compre_list(row_1, row_2):
    """ compere two list if value1 is -1 and value2 is 1 or 0 the function
    chage value1 to 1 or 0, the function return the number of -1 appeared in
     the valu1 and value2     """
    if row_1 == row_2:
        return row_2
    for i in range(len(row_1)):
        if row_1[i] != row_2[i]:
            if row_1[i] == -1:
                row_1[i] = row_2[i]
            elif row_2[i] == -1:
                continue
            else:
                return # if problem with the check
    return row_1


def first_line_solution(constraints, start_bord):
    """ its make reverse and if there isnt start bord, its make one  """
    rows_number = len(constraints[0])
    check_index = 1
    row_side_sol = make_list_of_index(start_bord)
    new_lines = check_row_and_line(check_index, constraints, rows_number,
                                   [], row_side_sol)
    line_result = make_list_of_index(new_lines)
    return line_result


def check_row_and_line(index, constraints, line_number,tmp_solution,start_row):
    """
    :return: posibale option for bord by check the row
    """
    tmp_result = []
    for i, value in enumerate(constraints[index]):
        if not (bool(value)):
            tmp_solution.append([WHITE] * line_number)
            continue
        if len(value) == 1 and sum(value) == line_number:
            tmp_solution.append([BLACK] * line_number)
            continue
        else:
            if type(start_row[0]) is list:
                tmp_result = get_row_variations(start_row[i], value)
            if not type(start_row[0]) is list:
                tmp_result = get_row_variations([-1] * line_number, value)
            if len(tmp_result) == 1:
                tmp_solution.append(tmp_result[0])
                continue
            if len(tmp_result) > 1:
                the_best_option = get_intersection_row(tmp_result)
                tmp_solution.append(the_best_option)
            else:
                tmp_solution.append([UNKNOWN] * line_number)
    return tmp_solution

#######################################################################^3

def solve_nonogram_helper4(constraints, option, simple_result, num=0,
                           first_enter=True):
    """
   A function that solves a board with the help of possible rows i[illegible] and if
   there are no contradictions the function takes the board out
    """
    global rows_option
    if first_enter:
```

```
        simple_result = solve_easy_nonogram(constraints) ### solve easy
        if not rows_with_unknown_check(simple_result):
            return [simple_result]

    if not simple_result:
        return
    # list of the index of rows with -1
    list_of_index = rows_with_unknown_check(simple_result)
    if not list_of_index:
        if option:
            if check_if_results_same(option, simple_result):
                return
        return option.append(simple_result)
    index = list_of_index[num]
    if list_of_index:
        rows_option = get_row_variations(simple_result[index],
                                         constraints[0][index])
    for j in range(len(rows_option)):
        if UNKNOWN in rows_option[j] and rows_option[j][1:] \
                == rows_option[j][:-1]:
            continue
        if rows_option[j] == simple_result[index]:
            continue
        simple_result[index] = rows_option[j]
        simple_result = check_row_and_line(0, constraints, len(constraints[1])
                                           , [], simple_result)
        solve_nonogram_helper4(constraints, option, simple_result, 0, False)
        if j == len(rows_option) - 1 and index == list_of_index[-1]:
            return
    return option


def check_if_results_same(option, simple_result):
    """ return TRUE if same
     else return false"""
    for i in option:
        if i == simple_result:
            return True
    return False


def rows_with_unknown_check(bord):
    """check if the bord contain -1
    :param bord: the list of list (the bord)
    :return: [] if the bord is full'
     else return list with the row index who contain -1"""
    rows_with_unknown = []
    if not bord:
        return
    for i, row in enumerate(bord):
        if UNKNOWN in row:
            rows_with_unknown.append(i)
    return rows_with_unknown


def solve_nonogram(constraints):
    return solve_nonogram_helper4(constraints, [], [])

##############################################################^4


def count_row_variations(length, blocks):
    num_block = len(blocks)
    the_amount_of_black = sum(blocks)
    k = length - the_amount_of_black - (num_block - 1)
    if k < 0:
        return 0
    n = length - the_amount_of_black + 1
```

7.1

7.2

```
    return int(binom_n_k(n, k))


def binom_n_k(n, k):
    f = math.factorial
    return f(n) / f(k) / f(n-k)

##########################################################################^5
```

Index of comments

5.1 שם לא משמעותי
-1

6.1 שם לא משמעותי

7.1
חלוקה לפונקציות- היה עדיף לחלק את הפונקציה לתתי פונקציות כך שכל פונקציה קטנה עושה דבר אחד ולא ארוכה ומסורבלת.

7.2 חסר תיעוד
-1