МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Курсовая работа по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил: студент группы БФИ2001

Фаттахов Т.Ф.

Руководитель: Симонов С.Е

Москва, 2022

Курсовая работа по дисциплине Структуры и алгоритмы обработки данных

Выполнил студент группы БФИ2001

Фаттахов Тагир Фанисович

Задача 1

«Заяц» и контролер играют в следующую игру.

Электричка представляет собой n вагонов, которые нумеруются натуральными числами от 1 до n в порядке от головы до хвоста. «Заяц» и контролер изначально находятся в некоторых двух различных вагонах. Электричка каждую минуту может быть в одном из двух состояний — в движении или на остановке. Каждую минуту игроки делают ходы.

Ход контролера заключается в следующем. Контролер имеет направление движения — к голове электрички или к ее хвосту. За свой ход контролер перемещается в соседний вагон в соответствии со своим направлением движения. Если по окончании своего хода контролер заходит в 1-ый или n-ый вагон, то он меняет свое направление движения на противоположное. Другими словами — контролер циклически ходит от головы электрички до хвоста и обратно в течение всего времени игры, на каждом ходу сдвигаясь на один вагон. Заметим, что у контролера есть всегда есть ровно один допустимый ход.

Ход «зайца» зависит от того, в каком состоянии находится электричка. Если электричка движется, то «заяц» может перейти в один из соседних вагонов или же никуда не перемещаться. Если электричка на остановке, то «заяц» покидает электричку (то есть теперь не находится ни в одном из вагонов электрички) и затем, если остановка не конечная, заходит в электричку снова в любой из n вагонов (не обязательно в тот, из которого он только что вышел, и не обязательно в соседний для этого вагона). Если электричка стоит на остановке несколько минут, «заяц» покидает электричку и заходит в нее каждую минуту.

Определим очередность ходов игроков. Если в данную минуту электричка находится в движении — то сначала делает ход «заяц», а затем — контролер. Если в данную минуту электричка на остановке, то сначала «заяц» выходит из электрички, затем контролер делает ход, а затем «заяц» заходит в электричку.

Если в какой-то момент времени «заяц» и контролер оказываются в одном вагоне, то выигрывает контролер: он заставляет «зайца» выплатить штраф. Если через некоторое время «заяц» доезжает до конечной остановки — выигрывает «заяц»: своим ходом он просто выходит из электрички и более в нее не возвращается.

В любой момент времени игроки знают положения друг друга. Игроки играют оптимально. В частности, если выигрывает контролер, то «заяц» играет так, чтобы проиграть как можно позже. Поскольку для контролера возможные ходы определены однозначно, то считается, что он играет оптимально всегда. Определите победителя.

Входные данные В первой строке находятся три целых числа n, m и k — количество вагонов в электричке, начальное положение «зайца» и контролера соответственно ($2 <= n <= 50$, $1 <= m, k <= n$, $m \neq k$).

Во второй строке задано направление движения контролера. «to head» означает, что контролер движется к голове электрички, «to tail» — что контролер движется к ее хвосту. Гарантируется, что в направлении движения контролера имеется хотя бы один вагон. Вагон 1 — это голова, вагон n — хвост.

Третья строка имеет длину от 1 до 200 и состоит из символов «0» и «1». i-ый символ содержит информацию о состоянии электрички в i-ую минуту времени. «0» означает, что в данную минуту электричка движется, «1» — что электричка в данную минуту стоит на остановке. Последний символ третьей строки всегда «1» — это конечная остановка.

Выходные данные Если выигрывает «заяц» — выведите «Stowaway» без кавычек. Иначе выведите «Controller» опять же без кавычек, затем через пробел — номер минуты, на которой «заяц» будет пойман.

Примеры
входные данные
5 3 2
to head
0001001
выходные данные
Stowaway
входные данные
3 2 1
to tail
0001

выходные данные
Controller 2

Если электричка едет и заяц находится между контролёром и началом поезда, то он двигается к началу поезда, иначе если заяц между контролёром и головой позде, то он двигается к голове Если электричка стоит, то заяц заходит в вагон, расположенный противоположно направлению движения контролёра

In [6]:
```
import sys
from typing import Any
```

In [5]:
```
def validate_int(n_str: str, min: int, max: int) -> int:
    try:
        n = int(n_str)
    except ValueError:
        raise Exception(f"{n_str} is not a number")

    if n < min or n > max:
        raise Exception(
                f"Inputed {n=} not in range [{min}, {max}]")

    return n
```

In [13]:
```
def request_list(size: int, min: int, max: int) -> list[int]:
    a_str = input().split()

    if len(a_str) > size:
        raise Exception(f"Inputed list is longer then {size}")

    return [validate_int(i, min, max) for i in a_str]
```

In [14]:
```
def request_int(min: int, max: int) -> int:
    n_str = input()
    return validate_int(n_str, min, max)
```

In [4]:
```
def solution_task1(
    n: int,
    hare: int,
    controller: int,
    direction_str: str,
    states: str
) -> tuple[str, int] | str:

    direction: int = 1 if direction_str == "to tail" else -1

    for i, c in enumerate(states):
        if c == '0':
            if hare < controller and hare > 1:
                hare -= 1
            elif hare > controller and hare < n:
                hare += 1

            controller += direction

            if controller < 1 or controller > n:
                controller -= 2 * direction
                direction = -direction

            if hare == controller:
                return "Controller", i + 1
        else:
            hare += direction
            controller += direction

            if controller < 1 or controller > n:
                controller -= 2 * direction
                direction =- direction

            if 1 <= controller - direction <= n:
                hare = controller - direction
            else:
                hare = controller + direction

    return "Stowaway"
```

In [7]:
```
def request_data1() -> Any:
    s: list[str] = input().split()

    if len(s) != 3:
        raise Exception(f"Must input 3 numbers, got {len(s)}")
```

```
    n = validate_int(s[0], 2, 50)
    m = validate_int(s[1], 1, sys.maxsize)
    k = validate_int(s[2], 1, n)

    if n == k:
        raise Exception(f"Hare and controller must be different")

    direction: str = input()

    if direction not in ("to head", "to tail"):
        raise Exception(f"Input 'to head' or 'to tail'")

    states: str = input()

    if len(states) < 1 or len(states) > 200:
        raise Exception(
                f"States must be in range [1, 200], got {len(states)}")

    states_set: set[str] = set(states)
    if len(states_set) > 2 or "0" not in states_set or "1" not in states_set:
        raise Exception(f"States must contain only 0 or 1")

    return n, m, k, direction, states
```

In [29]:
```
def test_solution_task1() -> None:
    tests = {
        (5, 3, 2, "to head", "0001001"): "Stowaway",
        (3, 2, 1, "to tail", "0001"): ("Controller", 2),
    }

    for before, mustbe in tests.items():
        after = solution_task1(*before)
        msg = f"{before=}, {after=}, {mustbe=}"
        assert after == mustbe, msg
```

Задача 2

Умный Бобер из ABBYY придумал новый вид шифрования сообщений и хочет проверить его работу. Делать это вручную долго и трудоемко, поэтому он решил обратиться к участникам ABBYY Cup.

Сообщение представляет собой n целых чисел a1, a2, ..., an. Для шифрования используется ключ, который представляет собой m целых чисел b1, b2, ..., bm (m <= n). Все числа из сообщения и из ключа лежат в интервале от 0 до c - 1, включительно, и все последующие вычисления проводятся по модулю c.

Шифрование проводится в n - m + 1 этапов. На первом этапе к каждому из чисел a1, a2, ..., am прибавляются соответствующие числа b1, b2, ..., bm. На втором этапе к числам a2, a3, ..., am + 1 (измененным на предыдущем этапе) прибавляются числа b1, b2, ..., bm. И так далее: на этапе номер i к числам ai, ai + 1, ..., ai + m - 1 прибавляются числа b1, b2, ..., bm. Результатом шифрования является последовательность a1, a2, ..., an после n - m + 1 этапов шифрования.

Помогите Бобру: напишите программу, которая будет осуществлять шифрование сообщений описанным способом.

Входные данные Первая строка входных данных содержит три целых числа n, m и c, разделенных единичными пробелами.

Вторая строка входных данных содержит n целых чисел ai (0 <= ai < c), разделенных единичными пробелами, — исходное сообщение.

Третья строка входных данных содержит m целых чисел bi (0 <= bi < c), разделенных единичными пробелами, — ключ шифрования.

Ограничения на входные данные для получения 30 баллов:

1 <= m <= n <= 10^3 1 <= c <= 10^3 Ограничения на входные данные для получения 100 баллов:

1 <= m <= n <= 10^5 1 <= c <= 10^3 Выходные данные Выведите n целых чисел, разделенных пробелами, — результат шифрования сообщения.

Примеры
входные данные
4 3 2
1 1 1 1
1 1 1
выходные данные
0 1 1 0

In [30]:
```
def solution_task2(a: list[int], b: list[int], c: int) -> list[int]:
    n: int = len(a)
```

```
    m: int = len(b)
    range_: int = n - m + 1
    sum: int = 0

    for i in range(n):
        if i < m:
            sum = (sum + b[i]) % c
        if i >= range_:
            sum = (c + sum - b[i-range_]) % c

        a[i] = (a[i] + sum) % c

    return a
```

In [15]:
```
def request_data2() -> Any:
    s: list[str] = input().split()

    if len(s) != 3:
        raise Exception(f"Must input 3 numbers, got {len(s)}")

    n = validate_int(s[0], 1, 1000)
    m = validate_int(s[1], 1, n)
    c = validate_int(s[2], 1, 1000)

    a = request_list(n, 0, c)
    b = request_list(m, 0, c)

    return a, b, c
```

In [32]:
```
def test_solution_task2() -> None:
    tests = {
        ((4, 3, 2), (1, 1, 1, 1), (1, 1, 1)): [0, 1, 1, 0],
        ((5, 3, 3), (1, 2, 3, 4, 5), (1, 2, 3)): [2, 2, 0, 0, 2],
    }

    for before, mustbe in tests.items():
        c: int = before[0][2]
        a: list[int] = list(before[1])
        b: list[int] = list(before[2])
        res = solution_task2(a, b, c)
        msg: str = f"before: [{c=}, {a=}, {b=}], after: {res}, mustbe: {mustbe}"
        assert res == mustbe, msg
```

Задача 3

Чемпионат Формула-1 состоит из серии гонок, называемых Гран-при. После каждой гонки первым 10 гонщикам начисляются призовые очки в соответствие с занятым местом: 25, 18, 15, 12, 10, 8, 6, 4, 2, 1. По завершении чемпионата гонщик с наибольшим количеством очков объявляется чемпионом. Если таких несколько, чемпионом объявляется тот из них, у кого больше побед (т. е. первых мест). Если таких все еще несколько, выбирается тот из них, у кого больше вторых мест, и так далее, пока есть места, по которым можно сравнивать.

В прошлом году была предложена, но отклонена другая система подсчета очков. По ней чемпион — тот, у кого больше побед. Если таких несколько, то чемпион — тот из них, у кого больше призовых очков. Если таких все еще несколько, то дальнейшее сравнение происходит так же, как и в исходной системе подсчета очков, т. е. сравнивается количество вторых, третьих, четвертых мест и так далее.

Вам даны результаты всех Гран-при сезона. Ваша задача — определить чемпионов по обеим системам подсчета очков. Гарантируется, что в обеих системах чемпион определяется однозначно.

Входные данные В первой строке записано целое число t (1 <= t <= 20) — количество гонок (Гран-при). Далее следуют описания всех гонок. Описание каждой гонки начинается с целого числа n (1 <= n <= 50) на отдельной строчке — количество гонщиков, участвовавших в данной гонке. В следующих n строках заданы результаты гонки, каждая строка содержит имя гонщика. Имена гонщиков даны в порядке от первого до последнего места. Имена состоят из строчных и заглавных латинских букв и имеют длину не более 50 символов. При сравнении имен большие и маленькие буквы следует различать.

Выходные данные Выведите ровно две строки. В первой должно быть записано имя чемпиона по исходной системе подсчета очков, а во второй — имя чемпиона по предложенной системе. Примеры
входные данные
3
3
Hamilton
Vettel
Webber
2
Webber

Vettel
2
Hamilton
Vettel
выходные данные
Vettel
Hamilton

In [33]:
```python
def solution_task3(a: list[list[str]]) -> tuple[str, str]:
    points: tuple[int, ...] = (25, 18, 15, 12, 10, 8, 6, 4, 2, 1)
    results: dict[str, tuple[list[int], str]] = {}

    for racer_names in a:
        for i, racer_name in enumerate(racer_names):
            if racer_name not in results:
                results[racer_name] = ([0] * 50, [racer_name])

            results[racer_name][0][i+1] += 1

            if i < 10:
                results[racer_name][0][0] += points[i]

    results_values = results.values()

    first: str = sorted(results_values)[-1][1][0]

    d: list[tuple[list[int], str]] = []
    for z in results_values:
        d.append(([z[0][1]] + z[0], z[1]))

    second: str = sorted(d)[-1][1][0]

    return first, second
```

In [34]:
```python
def request_data3() -> Any:
    t = request_int(1, 20)
    a: list[list[str]] = []

    for _ in range(t):
        s = input()
        if s.isdigit():
            a.append([input() for _ in range(int(s))])
        else:
            raise Exception(f"Must be a number")

    return (a, )
```

In [35]:
```python
def test_solution_task3() -> None:
    tests = {
        ("Vettel", "Hamilton"): [
            ["Hamilton", "Vettel", "Webber"],
            ["Webber", "Vettel"],
            ["Hamilton", "Vettel"]
        ]
    }

    for mustbe, before in tests.items():
        after = solution_task3(before)
        assert after == mustbe
```

Задача 4

Залы всех кинотеатров Берлядии представляют собой прямоугольники в K рядов по K кресел в каждом, причем K — нечетное число. Ряды и места нумеруются с 1 до K. Из соображений безопасности, человек, пришедший в кассу за билетами, не может сам выбрать места. Раньше это делал кассир, а теперь этим будет заниматься специальная программа рассадки. Было выяснено, что подавляющее большинство жителей Берляндии в кинотеатре предпочитают смотреть фильм, поэтому хотят сидеть как можно ближе к центру зала. Кроме того, компания из M человек, желающая посмотреть фильм, хочет непременно занять M последовательных мест в одном ряду. Сформулируем алгоритм, в соответствии с которым программа выдает людям билеты. При поступлении запроса на M мест программа должна определить номер ряда x и отрезок [yl, yr] номеров кресел на этом ряду, причем yr - yl + 1 = M. Из всех таких возможных вариантов программа должна выдать тот, для которого значение функции суммарной удаленности от центра минимально. Пусть — номер ряда и номер места у самого "центрального" кресла. Тогда значением функции удаленности от центра зала будет . Если вариантов с минимальным значением этой функции несколько, то программа должна выдать тот, что ближе к экрану (т.е. номер ряда x меньше). Если же вновь имеется неоднозначность, следует выдать вариант с минимальным yl. Если вы еще не догадались, то ваша задача — промоделировать работу программы.

Входные данные Первая строка содержит два целых числа N и K (1 <= N <= 1000, 1 <= K <= 99) — число запросов и размер зала соответственно. Во второй строке содержится N целых чисел Mi из диапазона [1, K], разделенных пробелами, — запросы к программе.

Выходные данные Выведите N строк. В i-й строке выведите «-1» (без кавычек), если невозможно найти Mi свободных подряд идущих кресел в одном ряду, или выведите тройку x, yl, yr в противном случае. Числа разделяйте пробелами. Примеры
входные данные
2 1
1 1
выходные данные
1 1 1
-1

In [36]:
```python
def solution_task4(k: int, requests: list[int]) -> list[str]:
    INF: int = sys.maxsize

    xc = yc = (k + 1) // 2
    row = [abs(j - yc) for j in range(k + 1)]
    dp = [
        [k + 1 - j for j in range(k + 1)]
        for _ in range(k + 1)
    ]

    result: list[str] = []

    for l in requests:
        xt, yt, c = -1, -1, INF

        for i in range(1, k + 1):
            for j in range(1, k - l + 2):

                if dp[i][j] >= l:
                    ct = row[j + l - 1] - row[j - 1] + abs(i - xc) * l

                    if ct < c or (ct == c and i < xt) \
                              or (ct == c and i == xt and j < yt):

                        xt, yt, c = i, j, ct

        if c < INF:
            for j in range(yt + l - 1, 0, -1):
                dp[xt][j] = min(dp[xt][j], max(0, yt - j))

            result.append(" ".join([str(z) for z in [xt, yt, yt + l - 1]]))
        else:
            result.append("-1")

    return result
```

In [39]:
```python
def request_data4() -> Any:
    s = input().split()
    if len(s) != 2:
        raise Exception(f"Must input 2 nubmers")

    n = validate_int(s[0], 1, 1000)
    k = validate_int(s[1], 1, 99)

    a = request_list(n, 1, k)

    return k, a
```

In [40]:
```python
def test_solution_task4() -> None:
    tests = {
        (1, (1, 1)): ["1 1 1", "-1"]
    }

    for before, mustbe in tests.items():
        after = solution_task4(before[0], list(before[1]))
        assert after == mustbe
```

Задача 5

Вася — учитель физкультуры в школе. В отличие от других учителей физкультуры, Вася не любит когда ученики выстраиваются в шеренгу по росту. Вместо этого, он требует, чтобы дети выстраивались в порядке a1, a2, ..., an, где ai — рост i-го ученика в шеренге, а n — количество учеников в шеренге. Детям сложно запомнить этот странный порядок, и сегодня они выстроились в порядке b1, b2, ..., bn, что очень расстроило Васю. Теперь Вася хочет переставить детей так, чтобы получился порядок a1, a2, ..., an. За одно действие Вася может поменять местами двух человек, стоящих подряд в шеренге. Помогите Васе — составьте

последовательность обменов, приводящую к нужной Васе расстановке. Количество действий минимизировать не требуется.

Входные данные В первой строке записано целое число n (1 <= n <= 300) — количество учеников. Во второй строке через пробел записано n целых чисел ai (1 <= ai <= 109) — какой рост должен иметь ученик на месте i. В третьей строке через пробел записано n целых чисел bi (1 <= bi <= 109) — какой рост имеет ученик на месте i в начальной расстановке. Возможно, что некоторые ученики имеют одинаковый рост. Гарантируется, что расставить детей в требуемом порядке возможно, т. е. a и b совпадают как мультимножества.

Выходные данные В первой строке выведите целое число k (0 <= k <= 106) — количество действий. Минимизировать k не требуется, но оно не должно превосходить 106. Далее выведите k строк по два целых числа через пробел. Строка pi, pi + 1 (1 <= pi <= n - 1) означает, что Вася должен поменять местами учеников на местах pi и pi + 1. Примеры
входные данные
4
1 2 3 2
3 2 1 2
выходные данные
4
2 3
1 2
3 4
2 3

In [41]:
```python
def solution_task5(
    mustbe: list[int],
    current: list[int]
) -> tuple[int, list[tuple[int, int]]]:
    moves: list[tuple[int, int]] = []

    for i in range(len(mustbe)-1,-1,-1):
        for j in range(i+1):
            if j != i and current[j] == mustbe[i]:
                moves.append((j+1, j+2))
                current[j], current[j+1] = current[j+1], current[j]

    return (len(moves), moves)
```

In [42]:
```python
def request_data5() -> Any:
    n = request_int(1, 300)
    a = request_list(n, 1, 109)
    b = request_list(n, 1, 109)

    return a, b
```

In [43]:
```python
def test_solution_task5() -> None:
    tests = (
        (
            [1, 2, 3, 2],
            [3, 2, 1, 2],
            (4, [(2, 3), (3, 4), (1, 2), (2, 3)])
        ),
    )

    for *before, mustbe in tests:
        after = solution_task5(*before)
        msg = f"{before=}, {after=}, {mustbe=}"
        assert mustbe == after, msg
```

Задача 6

По торжественному случаю открытия Зимней Компьютерной Школы организаторы решили закупить n литров колы. Однако в магазине возникло неожиданное затруднение: оказалось, что кола продается в бутылках по 0.5, 1 и 2 литра. При этом имеется ровно a бутылок по 0.5 литра, b литровых бутылок и c — двухлитровых. У организаторов достаточно денег, чтобы купить любое количество колы. Ожесточенные споры вызвало то, сколько каких бутылок покупать, ведь этот вопрос имеет принципиальное значение с точки зрения распределения колы между участниками (и организаторами тоже).

Итак, пока организаторы спорят, перебирая различные варианты закупки колы, Зимняя Компьютерная Школа не начнется. Ваша задача — посчитать количество всех возможных способов закупить ровно n литров колы и убедить организаторов, что это количество слишком велико и что если они будут продолжать свой спор, то Зимнюю Компьютерную Школу придется проводить летом.

Все бутылки колы одного объема считаются неразличимыми. Т.е. два варианта закупки отличаются друг от друга только в случае, когда они отличаются количеством бутылок хотя бы одного вида.

Входные данные В первой строке заданы четыре целых числа — n, a, b, c (1 <= n <= 10000, 0 <= a, b, c <= 5000).

Выходные данные Выведите единственное число — ответ на задачу. Если купить ровно n литров колы невозможно, выведите 0.
Пример входные данные
10 5 5 5
выходные данные
9

In [44]:
```python
def solution_task6(n: int, a: int, b: int, c: int) -> int:
    count: int = 0
    for ci in range(c + 1):
        liters_reminder: int = n - 2 * ci
        if liters_reminder < 0:
            break
        high = min(b, liters_reminder)
        low = max(0, liters_reminder - a // 2)
        if high >= low:
            count += high - low + 1

    return count
```

In [45]:
```python
def request_data6() -> Any:
    s = input().split()
    if len(s) != 4:
        raise Exception("Must be 4 numbers inputed")

    n = validate_int(s[0], 1, 1000)
    a = validate_int(s[1], 0, 5000)
    b = validate_int(s[2], 0, 5000)
    c = validate_int(s[3], 0, 5000)

    return n, a, b, c
```

In [46]:
```python
def test_solution_task6() -> None:
    tests = {
        (10, 5, 5, 5): 9
    }

    for before, mustbe in tests.items():
        after = solution_task6(*before)
        assert after == mustbe
```

Задача 7

Перед берляндскими учёными встала важнейшая задача — восстановить по имеющимся частям коротких фрагментов ДНК геном динозавра! У берляндского динозавра геном совсем не похож на привычный нам: в нём могут встречаться 26 различных типов нуклеотидов, причём нуклеотид каждого типа может встречаться не более одного раза. Если присвоить всем нуклеотидам различные буквы английского алфавита, то геном берляндского динозавра будет представлять собой непустую строку, состоящую из строчных букв английского алфавита, такую что каждая буква встречается в ней не более одного раза.

У учёных есть n фрагментов генома, которые представляют собой подстроки (непустые последовательности подряд идущих нуклеотидов) искомого генома.

Перед вами стоит задача: помочь учёным восстановить геном динозавра. Гарантируется, что входные данные непротиворечивы и хотя бы одна подходящая строка всегда существует. Узнав, что вы сильный программист, учёные дополнительно попросили вас из подходящих строк выбрать ту, длина которой минимальна. Если и таких строк несколько, то их устроит любая.

Входные данные В первой строке входных данных следует целое положительное число n (1 <= n <= 100) — количество фрагментов генома.

В каждой из следующих строк следует по одному описанию фрагмента. Каждый фрагмент — это непустая строка, состоящая из различных строчных букв английского алфавита. Не гарантируется, что заданные фрагменты различны. Фрагменты могли перекрываться произвольным образом, и один фрагмент мог быть подстрокой другого.

Гарантируется, что найдется такая строка из различных букв, которая содержит все заданные фрагменты в качестве подстрок.

Выходные данные В единственной строке выходных данных выведите геном минимальной длины, который содержит все заданные части. Все нуклеотиды в геноме должны быть различными. Если подходящих строк несколько, выведите строку минимальной длины. Если подходящих строк минимальной длины также несколько, то разрешается вывести любую из них.
Примеры
входные данные
3
bcd
ab

cdef
выходные данные
abcdef

In [47]:
```python
def solution_task7(p: list[str]) -> str:
    d = {}

    for t in p:
        for a, b in zip(t, t[1:]):
            d[a] = b

    s: str = ""

    for q in set("".join(p)) - set(d.values()):
        s += q
        while q in d:
            q = d[q]
            s += q

    return s
```

In [48]:
```python
def request_data7() -> Any:
    n = request_int(1, 100)
    return ([input() for _ in range(n)], )
```

In [49]:
```python
def test_solution_task7() -> None:
    tests = {
        ("bcd", "ab", "cdef"): "abcdef"
    }

    for before, mustbe in tests.items():
        after = solution_task7(list(before))
        assert after == mustbe
```

Задача 8

IPv6-адрес — это некоторое 128-битное число. Для удобства это число записывают блоками по 16 бит в шестнадцатеричной системе счисления, разделяя блоки двоеточиями — всего 8 блоков, в каждом из которых по 4 шестнадцатеричные цифры. Вот пример корректной записи адреса IPv6: «0124:5678:90ab:cdef:0124:5678:90ab:cdef». Будем называть такой формат записи IPv6-адреса полным.

Помимо полного формата записи IPv6-адреса, существует сокращенный формат записи. Запись IPv6-адреса может быть сокращена путем удаления одного или нескольких ведущих нулей в начале каждого блока. Однако каждый из блоков в итоге должен содержать хотя бы одну цифру. Например, ведущие нули можно сократить следующим образом: «a56f:00d3:0000:0124:0001:f19a:1000:0000» → «a56f:d3:0:0124:01:f19a:1000:00». Существуют и другие способы сократить нули в этом IPv6-адресе.

Некоторые IPv6-адреса содержат длинные последовательности нулей. Сплошные последовательности из нулевых 16-битных блоков могут быть сокращены до «::». Последовательность должна состоять из одного или нескольких последовательных блоков, все 16 бит в которых равны 0.

Примеры сокращения последовательностей нулевых блоков можно видеть ниже:

«a56f:00d3:0000:0124:0001:0000:0000:0000» → «a56f:00d3:0000:0124:0001::»; «a56f:0000:0000:0124:0001:0000:1234:0ff0» → «a56f::0124:0001:0000:1234:0ff0»; «a56f:0000:0000:0000:0001:0000:1234:0ff0» → «a56f:0000::0000:0001:0000:1234:0ff0»; «a56f:00d3:0000:0124:0001:0000:0000:0000» → «a56f:00d3:0000:0124:0001::0000»; «0000:0000:0000:0000:0000:0000:0000:0000» → «::». Сокращение нулевых блоков в адресе разрешается использовать не более одного раза. Это означает, что в сокращенной записи последовательность символов «::» может встретиться не более одного раза. В противном случае иногда будет невозможно определить число нулевых блоков, представленных каждым двойным двоеточием.

Формат записи IPv6-адреса после удаления ведущих нулей и сокращения нулевых блоков называется сокращенным.

Вам даны несколько сокращенных записей IPv6-адресов. Восстановите их полную запись.

Входные данные В первой строке записано единственное целое число n — количество записей для восстановления (1 <= n <= 100).

В каждой из последующих n строк записано по одной строке — сокращенные IPv6-адреса. Каждая строка состоит только из символов строки «0123456789abcdef:».

Гарантируется, что каждый сокращенный адрес получен путем, описанным в условии, из некоторого полного IPv6-адреса.

Выходные данные Для каждого сокращенного IPv6-адреса из входных данных выведите его полную запись на отдельной строке. Полные записи для сокращенных IPv6-адресов выводите в том порядке, в котором сокращенные записи идут во входных

данных.

Примеры
входные данные
6
a56f:d3:0:0124:01:f19a:1000:00
a56f:00d3:0000:0124:0001::
a56f::0124:0001:0000:1234:0ff0
a56f:0000::0000:0001:0000:1234:0ff0
::
0ea::4d:f4:6:0
выходные данные
a56f:00d3:0000:0124:0001:f19a:1000:0000
a56f:00d3:0000:0124:0001:0000:0000:0000
a56f:0000:0000:0124:0001:0000:1234:0ff0
a56f:0000:0000:0000:0001:0000:1234:0ff0
0000:0000:0000:0000:0000:0000:0000:0000
00ea:0000:0000:0000:004d:00f4:0006:0000

In [50]:
```python
def solution_task8(a: list[str]) -> list[str]:
    res = []

    for ipv6 in a:
        ipv6_splited: list[str] = ipv6.split(":")
        if "" in ipv6_splited:
            i: int = ipv6_splited.index("")
            ipv6_splited[i:i+1] = [""] * (9 - len(ipv6_splited))


        res.append(":".join(b.rjust(4, "0") for b in ipv6_splited))

    return res
```

In [64]:
```python
def request_data8() -> Any:
    n = request_int(1, 100)
    letters = set("0123456789abcdefa:")
    a: list[str] = []

    for _ in range(n):
        s = input()

        for letter in set(s):
            if letter not in letters:
                raise Exception(f"IPv6 must containd only {letters}")

        a.append(s)


    return (a, )
```

In [52]:
```python
def test_solution_task8() -> None:
    tests = (
        ([
            "a56f:d3:0:0124:01:f19a:1000:00",
            "a56f:00d3:0000:0124:0001::",
            "a56f::0124:0001:0000:1234:0ff0",
            "a56f:0000::0000:0001:0000:1234:0ff0",
            "::",
            "0ea::4d:f4:6:0",
        ], [
            "a56f:00d3:0000:0124:0001:f19a:1000:0000",
            "a56f:00d3:0000:0124:0001:0000:0000:0000",
            "a56f:0000:0000:0124:0001:0000:1234:0ff0",
            "a56f:0000:0000:0000:0001:0000:1234:0ff0",
            "0000:0000:0000:0000:0000:0000:0000:0000",
            "00ea:0000:0000:0000:004d:00f4:0006:0000",
        ]),
    )

    for before_list, mustbe_list in tests:
        for i, before in enumerate(before_list):
            after = solution_task8([before])
            mustbe = [mustbe_list[i]]
            msg = f"{before=}, {after=}, {mustbe=}"
            assert after == mustbe, msg
```

Задача 9

. .

У Васи есть последовательность кубиков, на каждом из которых написано ровно одно целое число. Вася любит порядок, поэтому выставил все свои кубики в ряд. Таким образом, последовательность чисел, записанных на кубиках в порядке слева направо равна a1, a2, ..., an.

Пока Вася отсутствовал, его маленький брат Степан поиграл с кубиками и поменял местами некоторые из них, после чего последовательность чисел на кубиках стала выглядеть как b1, b2, ..., bn.

Степан утверждает, что трогал только кубики, стоящие на позициях с l по r, не убирал и не добавлял другие кубики (иными словами, он переставил кубики на позициях с l по r произвольным образом).

Перед вами стоит задача определить, возможна ли то, что Степан говорит правду, либо он гарантированно обманывает своего брата.

Входные данные В первой строке следует три целых числа n, l, r (1 <= n <= 105, 1 <= l <= r <= n) — количество кубиков, которые есть у Васи, и позиции, о которых сказал Степан.

Во второй сроке следует последовательность a1, a2, ..., an (1 <= ai <= n) — последовательность номеров на кубиках в том порядке, в котором их расставил Вася.

В третьей сроке следует последовательность b1, b2, ..., bn (1 <= bi <= n) — последовательность номеров на кубиках после того, как Степан поиграл с ними.

Гарантируется, что Степан не убирал и не добавлял никакие кубики, а только переставлял имеющиеся.

Выходные данные Выведите «LIE» (без кавычек), если Степан гарантированно обманывает брата. В противном случае, выведите «TRUTH» (без кавычек). Примеры
входные данные
5 2 4
3 4 2 3 1
3 2 3 4 1
выходные данные
TRUTH
входные данные
3 1 2
1 2 3
3 1 2
выходные данные
LIE

In [53]:
```python
def solution_task9(a: list[int], b: list[int], l: int, r: int) -> str:
    l -= 1
    a_part: list[int] = a[l:r]

    for bi in b[l:r]:
        if bi not in a_part:
            return "LIE"

    return "TRUTH"
```

In [55]:
```python
def request_data9() -> Any:
    s = input().split()
    if len(s) != 3:
        raise Exception("Must input only 3 numbers")

    n = validate_int(s[0], 1, 105)
    l = validate_int(s[1], 1, n)
    r = validate_int(s[2], 1, n)

    a = request_list(n, 1, n)
    b = request_list(n, 1, n)

    return a, b, l, r
```

In [56]:
```python
def test_solution_task9() -> None:
    tests = {
        (2, 4, (3, 4, 2, 3, 1), (3, 2, 3, 4, 1)): "TRUTH",
        (1, 2, (1, 2, 3), (3, 1, 2)): "LIE",
    }

    for (l, r, a, b), mustbe in tests.items():
        res = solution_task9(list(a), list(b), l, r)
        msg = f"{l=}, {r=}, {a=}, {b=}, {res=}, {mustbe=}"
        assert res == mustbe, msg
```

Задача 10

Недавно Поликарп изучил операцию «побитового И» (она же — операция «AND») целых неотрицательных чисел. Теперь он хочет продемонстрировать учителю информатики в школе свое виртуозное владение изученной операцией.

Для этого Поликарп пришел в школу пораньше и записал на доске последовательность целых неотрицательных чисел a1, a2, ..., an. А также квадратную матрицу b размера n × n. Элемент матрицы b, стоящий в i-той строке в j-том столбце (обозначим его bij), равен:

«побитовому И» чисел ai и aj (то есть bij = ai & aj), если i ≠ j; -1, если i = j. Выписав матрицу b, Поликарп очень обрадовался и стер с доски последовательность a. Вот незадача — без этой последовательности учитель не сможет проверить правильно ли Поликарп произвел вычисления. Поликарпу срочно надо восстановить стертую последовательность чисел, иначе он не докажет, что умеет правильно считать.

Помогите Поликарпу, по матрице b восстановите последовательность чисел a1, a2, ..., an, которую он стер с доски. Поликарп не любит большие числа, поэтому каждое число в восстановленной последовательности не должно превышать 109.

Входные данные В первой строке записано единственное целое число n (1 <= n <= 100) — размер квадратной матрицы b. В следующих n строках записана матрица b. В i-той из этих строк записаны n целых чисел, разделенных пробелами: j-тое число обозначает элемент матрицы bij. Гарантируется, что для всех i (1 <= i <= n) выполняется bii = -1. Гарантируется, что для всех i, j (1 <= i, j <= n; i ≠ j) выполняется 0 <= bij <= 109, bij = bji.

Выходные данные Выведите n целых неотрицательных чисел a1, a2, ..., an (0 <= ai <= 109) — последовательность, которую стер Поликарп. Выведенные числа разделяйте пробельными символами.

Гарантируется, что существует последовательность a, удовлетворяющая условиям задачи. Если существует несколько таких последовательностей, разрешается вывести любую.
Примеры
входные данные
1
-1
выходные данные
0
входные данные
3
-1 18 0
18 -1 0
0 0 -1
выходные данные
18 18 0

In [60]:
```python
def solution_task10(b: list[list[int]]) -> list[int]:
    a: list[int] = [0] * len(b)

    for i, row in enumerate(b):
        for j, col in enumerate(row):
            if i != j:
                a[i] |= col
                a[j] |= col

    return a
```

In [61]:
```python
def request_data10() -> Any:
    n = request_int(1, 100)
    a = [request_list(n, -1, 109) for _ in range(n)]

    for i in range(n):
        if a[i][i] != -1:
            raise Exception("Diagonal must contain only -1")

    return (a, )
```

In [62]:
```python
def test_solution_task10() -> None:
    tests = (
        ([[-1]], [0]),
        ([[-1, 18, 0], [18, -1, 0], [0, 0, -1]], [18, 18, 0])
    )

    for before, mustbe in tests:
        after = solution_task10(before)
        msg = f"{before=}, {after=}, {mustbe=}"
        assert after == mustbe, msg
```

In [69]:
```python
def run_tests() -> None:
```

```
    test_solution_task1()
    test_solution_task2()
    test_solution_task3()
    test_solution_task4()
    test_solution_task5()
    test_solution_task6()
    test_solution_task7()
    test_solution_task8()
    test_solution_task9()
    test_solution_task10()
```

In [70]:
```
run_tests()
```

In [71]:
```
solution_task1(*request_data1())
```

Out[71]:
```
'Stowaway'
```

In [72]:
```
solution_task2(*request_data2())
```

Out[72]:
```
[0, 1, 1, 0]
```

In [73]:
```
solution_task3(*request_data3())
```

Out[73]:
```
('Vettel', 'Hamilton')
```

In [74]:
```
solution_task4(*request_data4())
```

Out[74]:
```
['1 1 1', '-1']
```

In [75]:
```
solution_task5(*request_data5())
```

Out[75]:
```
(4, [(2, 3), (3, 4), (1, 2), (2, 3)])
```

In [76]:
```
solution_task6(*request_data6())
```

Out[76]:
```
9
```

In [77]:
```
solution_task7(*request_data7())
```

Out[77]:
```
'abcdef'
```

In [78]:
```
solution_task8(*request_data8())
```

Out[78]:
```
['a56f:00d3:0000:0124:0001:f19a:1000:0000',
 'a56f:00d3:0000:0124:0001:0000:0000:0000',
 'a56f:0000:0000:0124:0001:0000:1234:0ff0',
 'a56f:0000:0000:0000:0001:0000:1234:0ff0',
 '0000:0000:0000:0000:0000:0000:0000:0000',
 '00ea:0000:0000:0000:004d:00f4:0006:0000']
```

In [79]:
```
solution_task9(*request_data9())
```

Out[79]:
```
'TRUTH'
```

In [80]:
```
solution_task10(*request_data10())
```

Out[80]:
```
[18, 18, 0]
```

In [ ]:
```

```