МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по курсу «Алгоритмы: теория и практика. Структуры данных»**

ВЫПОЛНИЛ студент группы 21-ИТ-1 Шиковец.Е.А.

ПРОВЕРИЛ преподаватель Виноградова А.Д.

Полоцк, 2022 г.

**ОПИСАНИЕ ПРОДЕЛАННОЙ РАБОТЫ:**

**МОДУЛЬ 1.2 БАЗОВЫЕ СТРУКТУРЫ ДАННЫХ**

**ЗАДАЧА 1**

**Условие задачи и входные выходные данные**

**Скобки в коде**

Проверить, правильно ли расставлены скобки в данном коде.

**Вход.** Исходный код программы.

**Выход.** Проверить, верно ли расставлены скобки. Если нет, выдать индекс первой ошибки.

**Sample Input 1:**

([](){([])})

**Sample Output 1:**

Success

**Sample Input 2:**

()[]}

**Sample Output 2:**

5

**Sample Input 3:**

{{[()]]

**Sample Output 3:**

7

**Решение:**

import 'dart:convert';

import 'dart:io';

class Stack<T> {

var list = <T>[];

Stack();

void push(T n) => list.add(n);

void pop() => list.removeLast();

T peek() => list.last;

bool isEmpty() => list.isEmpty;

}

void main(List<String> arguments) {

final line = stdin.readLineSync(encoding: utf8) ?? "";

var stack = Stack<String>();

var opened = <int>[];

var closed = <int>[];

for (var i = 0; i < line.length; i++) {

if (line[i] == '(' || line[i] == '{' || line[i] == '[') {

stack.push(line[i]);

opened.add(i);

} else if (line[i] == ')' || line[i] == '}' || line[i] == ']') {

if (!stack.isEmpty()) {

var fromStack = stack.peek();

if (line[i] == ')' && fromStack == '(' ||

line[i] == '}' && fromStack == '{' ||

line[i] == ']' && fromStack == '[') {

stack.pop();

opened.removeLast();

} else {

closed.add(i);

break;

}

} else {

stack.push(line[i]);

closed.add(i);

break;

}

}

}

if (!stack.isEmpty()) {

if (closed.isNotEmpty) {

print(closed.first + 1);

} else if (opened.isNotEmpty) {

print(opened.first + 1);

}

} else {

print('Success');

}

}

**ЗАДАЧА 2**

**Условие задачи и входные выходные данные**

**Высота дерева**

Вычислить высоту данного дерева.

**Вход.** Корневое дерево с вершинами {0, . . . , n−1}, заданное как последовательность parent0 , . . . , parentn−1 , где parenti — родитель i-й вершины.

**Выход.** Высота дерева.

**Sample Input:**

10

9 7 5 5 2 9 9 9 2 -1

**Sample Output:**

4

**Решение:**

def Height(n, tree):

root = -1

height = 1

maxpath = 0

tmp\_save = {} #Словарь содержащий уже рассчитанные значения листьев

for node in range(n):

parent = tree[node]

if parent == root:

tmp\_save[node] = 1

else:

height += 1

while True:

path = tree[parent]

if path != root:

if path in tmp\_save: #Если имеется уже в словаре, берем значение

height = height + tmp\_save[path]

tmp\_save[node] = height #Сохраняем информацию о глубине

if height > maxpath:

maxpath = height

height = 1

break

parent = path

height += 1

else:

tmp\_save[node] = height #Сохраняем информацию о глубине

if height > maxpath:

maxpath = height

height = 1

break

return maxpath

def main():

n = int(input().strip() or 0)

array = input().split()

array = map(float, array)

array = list(map(round, array))

l = len(array)

if l != n:

n = l

print(Height(n, array))

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

main()

**ЗАДАЧА 3**

**Условие задачи и входные выходные данные**

**Обработка сетевых пакетов**

Реализовать обработчик сетевых пакетов.

**Вход.** Размер буфера size и число пакетов n, а также две последовательности arrival1, . . . , arrivaln и duration1, . . . , durationn, обозначающих время поступления и длительность обработки n пакетов.

**Выход.** Для каждого из данных n пакетов необходимо вывести время начала его обработки или −1, если пакет не был обработан (это происходит в случае, когда пакет поступает в момент, когда в буфере компьютера уже находится size пакетов).

**Sample Input 1:**

1 0

**Sample Output 1:**

**Sample Input 2:**

1 1

0 0

**Sample Output 2:**

0

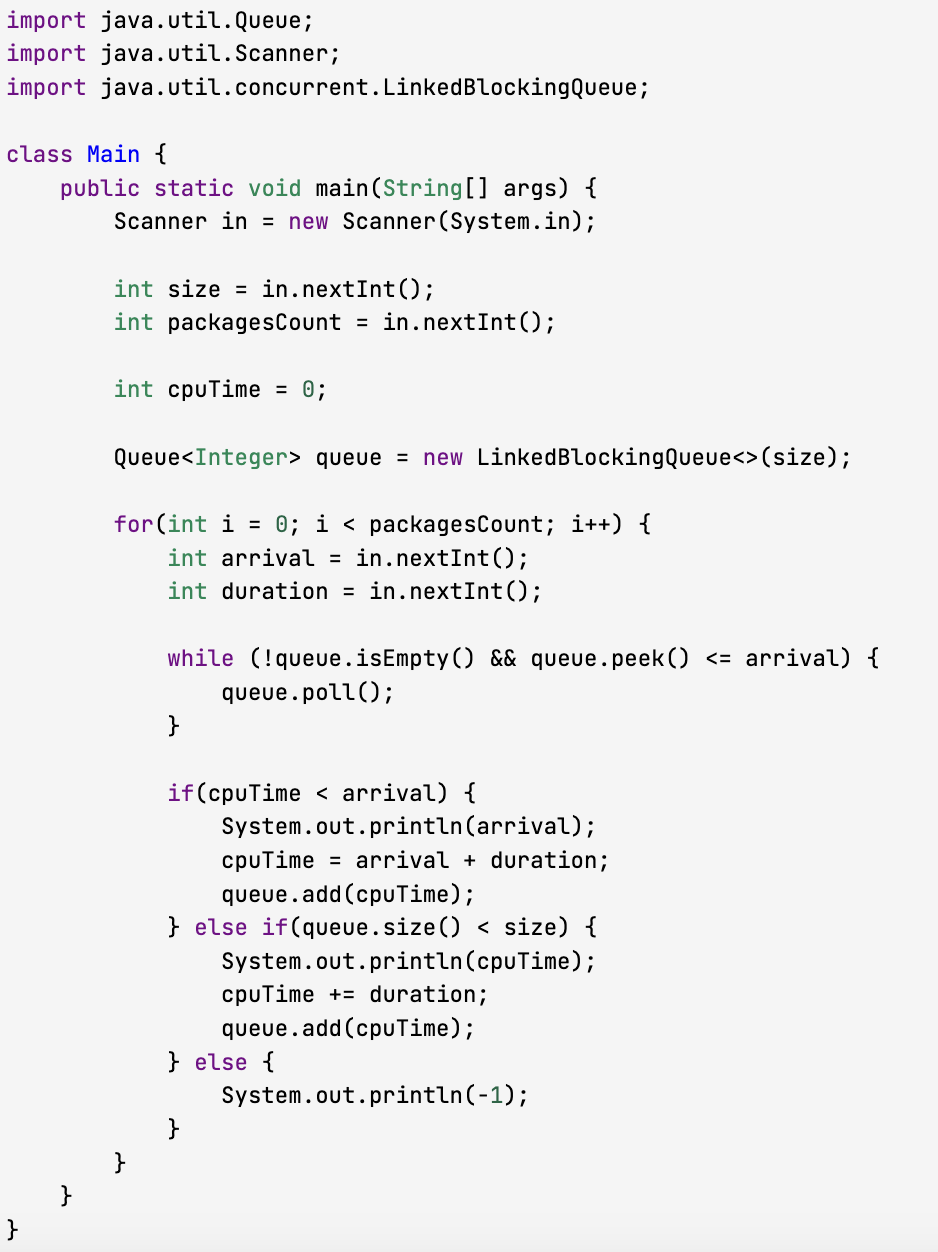
**Sample Input 3:**

1 1

0 1

**Sample Output 3:**

**Решение:**

****

**ЗАДАЧА 4**

**Условие задачи и входные выходные данные**

**Стек с поддержкой максимума**

Реализовать стек с поддержкой операций push, pop и max.

**Вход.** Последовательность запросов push, pop и max.

**Выход.** Для каждого запроса max вывести максимальное число, находящееся на стеке.

**Sample Input 1:**

5

push 2

push 1

max

pop

max

**Sample Output 1:**

2

2

**Sample Input 2:**

5

push 1

push 2

max

pop

max

**Sample Output 2:**

2

1

**Sample Input 3:**

10

push 2

push 3

push 9

push 7

push 2

max

max

max

pop

max

**Sample Output 3:**

9

9

9

9

**Решение:**

class StackMax:

def \_\_init\_\_(self):

self.stack\_ = []

def \_\_bool\_\_(self):

return bool(self.stack\_)

def push(self, item):

if self.stack\_:

new\_max = max(item, self.stack\_[-1][1])

else:

new\_max = item

self.stack\_.append((item, new\_max))

def pop(self):

return self.stack\_.pop()[0]

def get\_max(self):

return self.stack\_[-1][1]

def main():

s = StackMax()

for \_ in range(int(input())):

cmd = input().split()

if 'pop' in cmd[0]:

if s:

s.pop()

else:

print('error')

if 'push' in cmd[0]:

s.push(int(cmd[1]))

if 'max' in cmd[0]:

if s:

print(s.get\_max())

else:

print('None')

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

main()

**ЗАДАЧА 5**

**Условие задачи и входные выходные данные**

**Максимум в скользящем окне**

Найти максимум в каждом окне размера m данного массива чисел A[1 . . . n].

**Вход.** Массив чисел A[1 . . . n] и число 1 ≤ m ≤ n.

**Выход.** Максимум подмассива A[i . . . i + m − 1] для всех 1 ≤ i ≤ n-m+1

**Sample Input 1:**

3

2 1 5

1

**Sample Output 1:**

2 1 5

**Sample Input 2:**

8

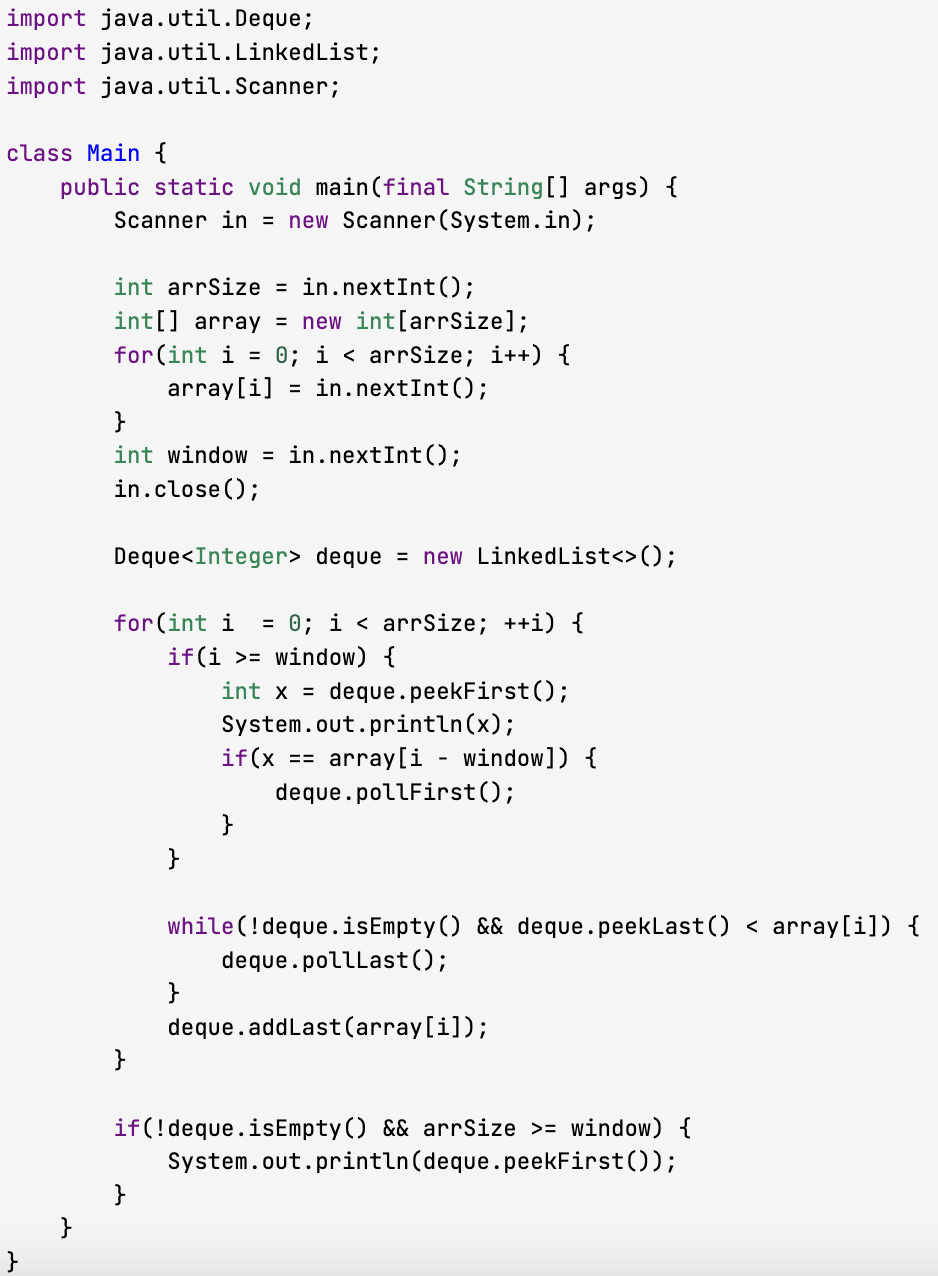
2 7 3 1 5 2 6 2

4

**Sample Output 2:**

7 7 5 6 6

**Решение:**

****

**МОДУЛЬ 2.3 ОЧЕРЕДИ С ПРИОРИТЕТОМ И СИСТЕМЫ НЕПЕРЕСЕКАЮЩИХСЯ МНОЖЕСТВ**

**ЗАДАЧА 1**

**Условие задачи и входные выходные данные**

**Построение кучи**

Переставить элементы заданного массива чисел так, чтобы он удовлетворял свойству мин-кучи.

**Вход.** Массив чисел A[0 . . . n − 1].

**Выход.** Переставить элементы массива так, чтобы выполнялись неравенства A[i] ≤ A[2i + 1] и A[i] ≤ A[2i + 2] для всех i.

**Sample Input 1:**

6

0 1 2 3 4 5

**Sample Output 1:**

0

**Sample Input 2:**

6

7 6 5 4 3 2

**Sample Output 2:**

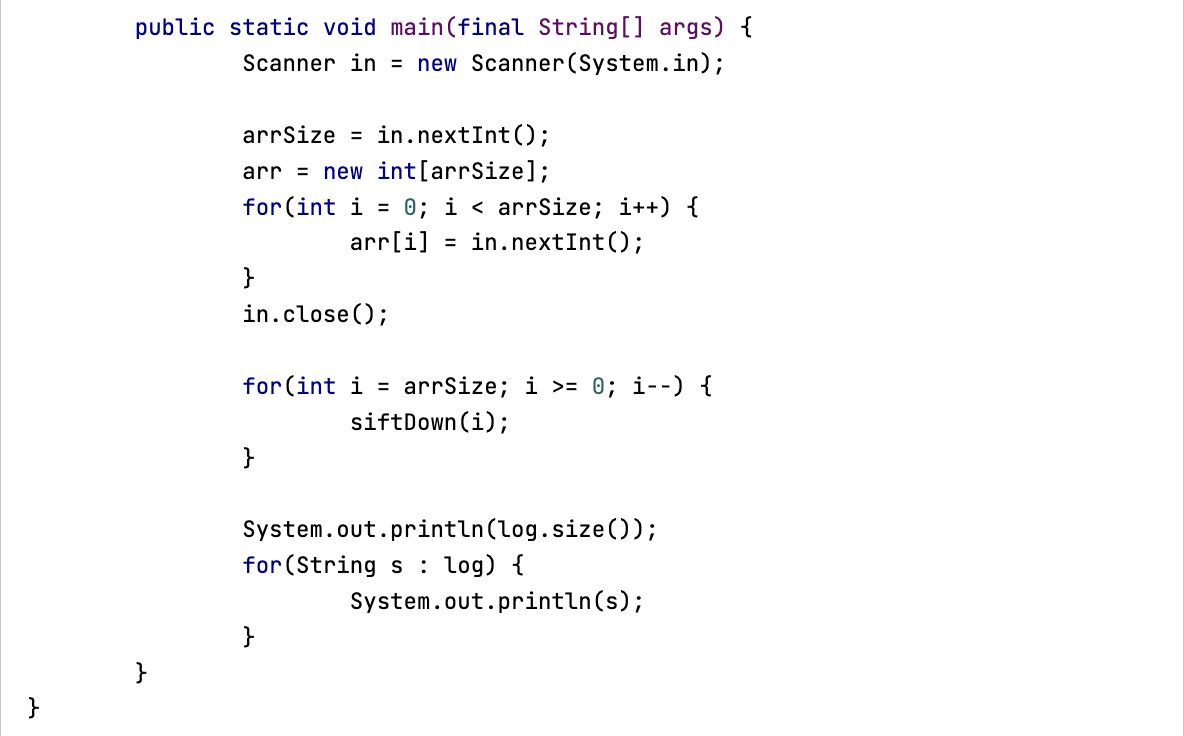
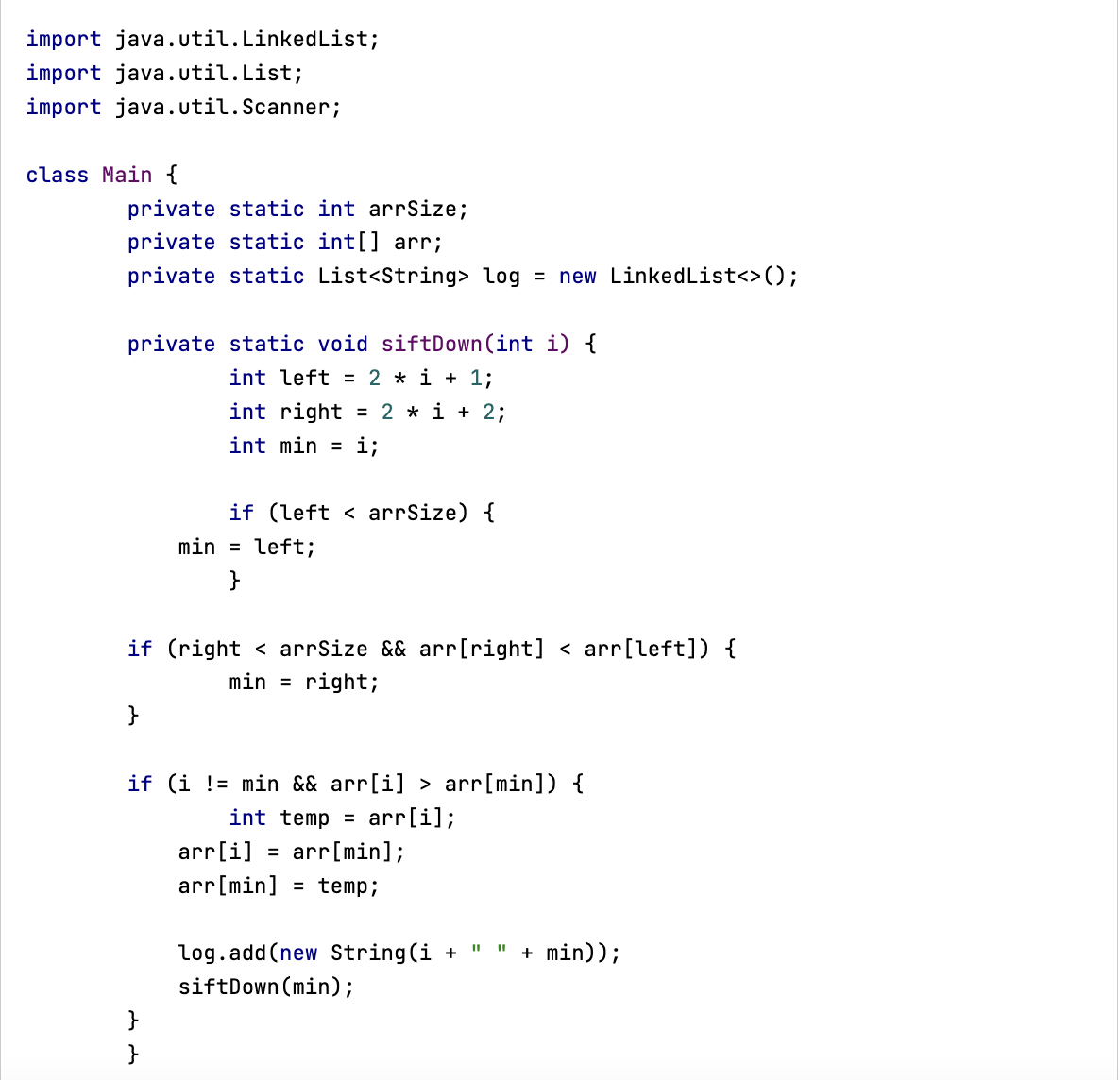
4

2 5

1 4

0 2

2 5

****

**ЗАДАЧА 2**

**Условие задачи и входные выходные данные**

**Параллельная обработка**

По данным n процессорам и m задач определите, для каждой из задач, каким процессором она будет обработана.

**Вход.** Число процессоров n и последовательность чисел t0, . . . , tm−1, где ti — время, необходимое на обработку i-й задачи.

**Выход.** Для каждой задачи определите, какой процессор и в какое время начнёт её обрабатывать, предполагая, что каждая задача поступает на обработку первому освободившемуся процессору.

**Sample Input:**

2 5

1 2 3 4 5

**Sample Output:**

0 0

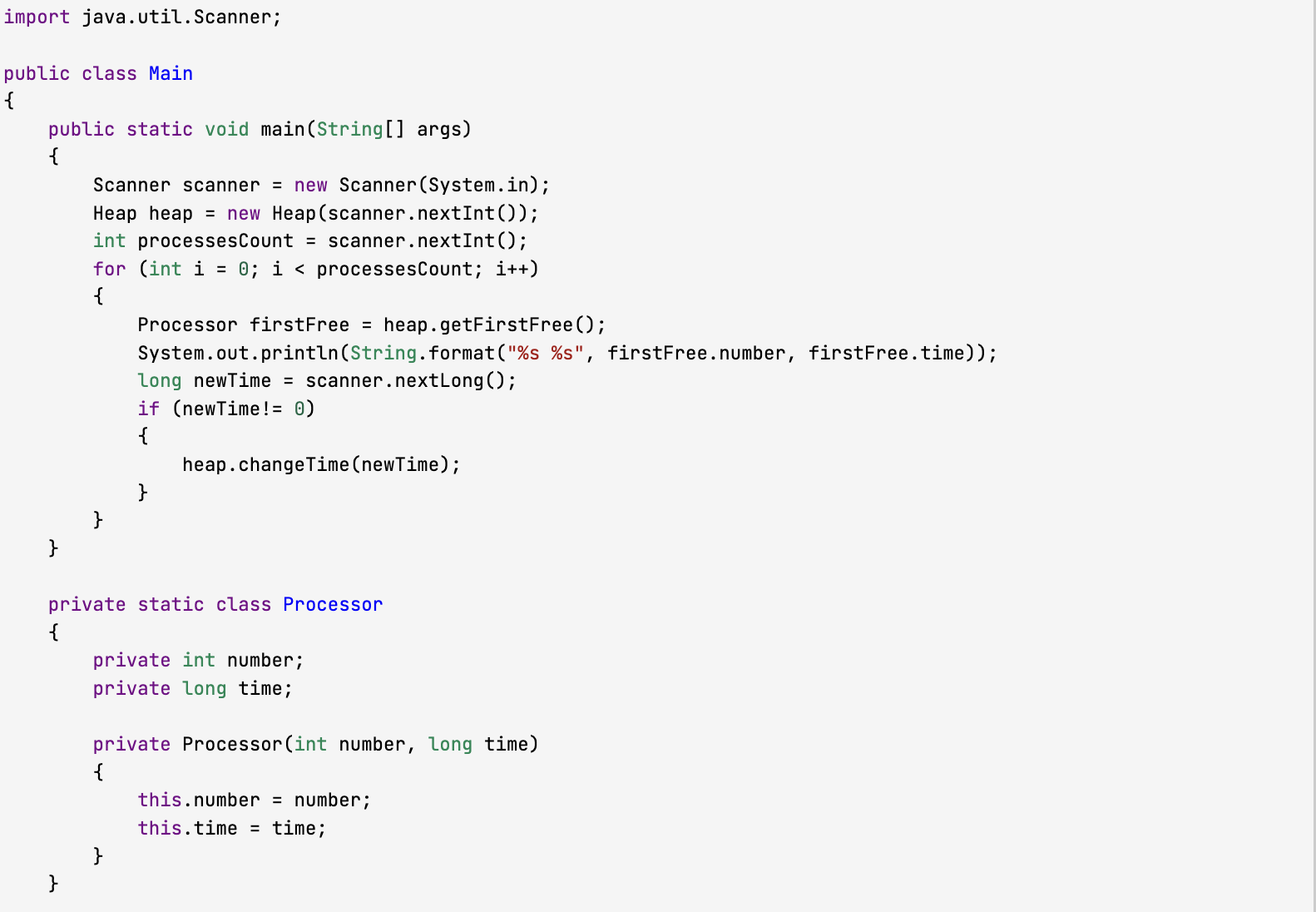
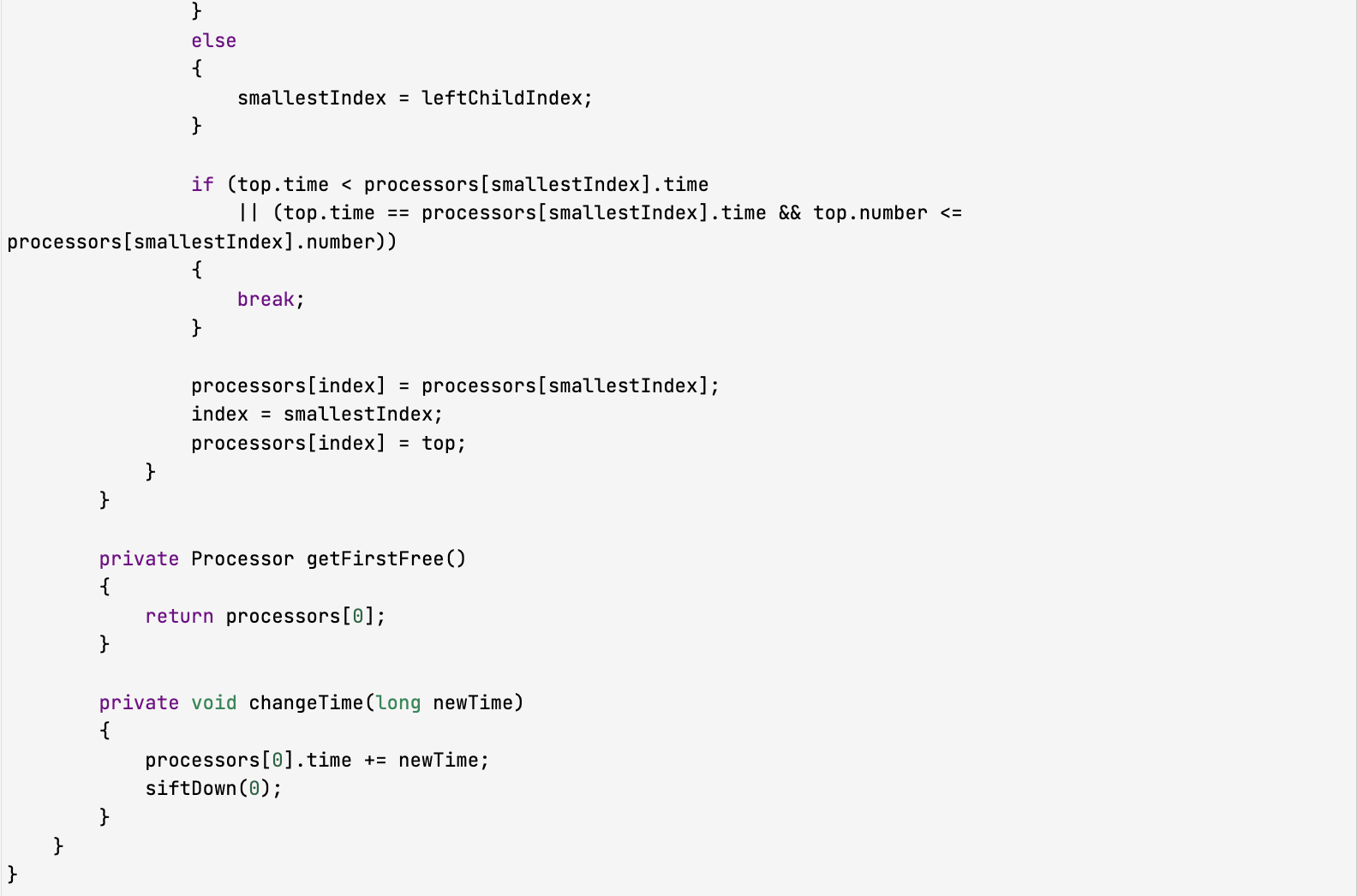
1 0

0 1

1 2

0 4

**Решение:**

**** **** 

**ЗАДАЧА 3**

**Условие задачи и входные выходные данные**

Ваша цель в данной задаче — реализовать симуляцию объединения таблиц в базе данных

**Формат входа.** Первая строка содержит числа n и m — число таблиц и число запросов, соответственно. Вторая строка содержит n целых чисел r1, . . . , rn — размеры таблиц. Каждая из последующих m строк содержит два номера таблиц destinationi и sourcei , которые необходимо объединить.

**Формат выхода.** Для каждого из m запросов выведите максимальный размер таблицы после соответствующего объединения.

**Sample Input:**

5 5

1 1 1 1 1

3 5

2 4

1 4

5 4

5 3

**Sample Output:**

2

2

3

5

5

**Решение:**

# Рекурсивная функция поиска корневого родителя с его записью во всех потомков (с целью уменьшения глубины дерева)

def find(i):

if tables[i][1] is None:

return i

tables[i][1] = find(tables[i][1])

return tables[i][1]

def main():

n, m = map(int, input().split())

global tables

tables = [None] \* n

max\_val = 0

# При инициализации у всех таблиц нет родителей и каждая имеет свой размер

for i, x in enumerate(input().split()):

tables[i] = [int(x), None]

max\_val = max(max\_val, int(x))

# Поиск родителей и подвешивание одного к другому, если родитель не один и тот же

for \_ in range(m):

dest, src = [find(int(x) - 1) for x in input().split()]

if dest != src:

tables[dest][0] += tables[src][0]

tables[src] = [0, dest]

max\_val = max(max\_val, tables[dest][0])

print(max\_val)

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

main()

**ЗАДАЧА 4**

**Условие задачи и входные выходные данные**

**Система равенств и неравенств**

Проверить, можно ли присвоить переменным целые значения, чтобы выполнить заданные равенства вида xi = xj и неравенства вида xp 6= xq.

**Вход.** Число переменных n, а также список равенств вида xi = xj и неравенства вида xp 6= xq.

**Выход.** Проверить, выполнима ли данная система.

**Sample Input 1:**

4 6 0

1 2

1 3

1 4

2 3

2 4

3 4

**Sample Output 1:**

1

**Sample Input 2:**

4 6 1

1 2

1 3

1 4

2 3

2 4

3 4

1 2

**Sample Output 2:**

0

**Sample Input 3:**

4 0 6

1 2

1 3

1 4

2 3

2 4

3 4

**Sample Output 3:**

1

**Решение:**

import sys

def main():

# Читаем все числа из stdin в массив массивов

ls = [[\*map(int, i.split())] for i in sys.stdin]

# Получаем значение e + формируем массив родителей, причем для удобства делаем его на один элемент больше

# нужного (нулевой не будет использоваться). Значения n и d, по сути, неинтересны

(n, e, d), ps = ls[0], [\*range(ls[0][0] + 1)]

# Приравниваем всех родителей массива равенств

for i, j in ls[1:e+1]:

ps[j] = ps[i]

# Проверяем, что все элементы массива неравенств не равны между собой и выводим соответствующий код (0 или 1)

print(int(all(ps[i] != ps[j] for i, j in ls[e+1:])))

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

main()

**МОДУЛЬ 3.2 ХЕШ-ТАБЛИЦЫ**

**ЗАДАЧА 1**

**Условие задачи и входные выходные данные**

**Телефонная книга**

Реализовать структуру данных, эффективно обрабатывающую запросы вида add number name, del number и find number.

**Вход.** Последовательность запросов вида add number name, del number и find number, где number — телефонный номер, содержащий не более семи знаков, а name — короткая строка.

**Выход.** Для каждого запроса find number выведите соответствующее имя или сообщите, что такой записи нет.

**Sample Input 1:**

12

add 911 police

add 76213 Mom

add 17239 Bob

find 76213

find 910

find 911

del 910

del 911

find 911

find 76213

add 76213 daddy

find 76213

**Sample Output 1:**

Mom

not found

police

not found

Mom

daddy

**Sample Input 2:**

8

find 3839442

add 123456 me

add 0 granny

find 0

find 123456

del 0

del 0

find 0

**Sample Output 2:**

not found

granny

me

not found

**Решение:**

****

**ЗАДАЧА 2**

**Условие задачи и входные выходные данные**

**Хеширование цепочками**

**Формат входа.** Первая строка размер хеш-таблицы m. Следующая строка содержит количество запросов n. Каждая из последующих n строк содержит запрос одного из перечисленных выше четырёх типов.

**Формат выхода.** Для каждого из запросов типа find и check выведите результат в отдельной строке.

**Sample Input 1:**

5

12

add world

add HellO

check 4

find World

find world

del world

check 4

del HellO

add luck

add GooD

check 2

del good

**Sample Output 1:**

HellO world

no

yes

HellO

GooD luck

**Sample Input 2:**

4

8

add test

add test

find test

del test

find test

find Test

add Test

find Test

**Sample Output 2:**

yes

no

no

yes

**Решение:**

from collections import deque

def main():

m, p, X = int(input()), 1000000007, [1, 263]

hash\_table = [deque() for \_ in range(m)]

for \_ in range(int(input())):

command, arg = input().split()

if command == "check":

print(' '.join(hash\_table[int(arg)]))

else:

# расчёт степеней Х

for \_ in range(len(arg) - len(X)):

X.append((X[-1] \* X[1]) % p)

chain = hash\_table[sum(ord(c) \* x for c, x in zip(arg, X)) % p % m]

contains = arg in chain

if command == "find":

print("yes" if contains else "no")

elif not contains and command == "add":

chain.appendleft(arg)

elif contains and command == "del":

chain.remove(arg)

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

main()

**ЗАДАЧА 3**

**Условие задачи и входные выходные данные**

**Поиск образца в тексте**

Найти все вхождения строки Pattern в строку Text.

**Вход.** Строки Pattern и Text.

**Выход.** Все индексы i строки Text, начиная с которых строка Pattern входит в Text: Text[i..i + |Pattern| − 1] = Pattern

**Sample Input 1:**

aba

abacaba

**Sample Output 1:**

0 4

**Sample Input 2:**

Test

testTesttesT

**Sample Output 2:**

4

**Sample Input 3:**

aaaaa

baaaaaaa

**Sample Output 3:**

1 2 3

**Решение:**

def main():

pattern, string = input(), input()

P = len(pattern)

needed = sum(ord(c) for c in pattern)

current= sum(ord(c) for c in string[:P])

for i in range(len(string) - P + 1):

if needed == current and pattern == string[i:i + P]:

print(i, sep=' ')

current += ord(string[(i + P) % len(string)]) - ord(string[i])

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

main()

**МОДУЛЬ 4.4 ДЕРЕВЬЯ ПОИСКА**

**ЗАДАЧА 1**

**Условие задачи и входные выходные данные**

**Обход двоичного дерева**

Построить in-order, pre-order и post-order обходы данного двоичного дерева.

**Вход.** Двоичное дерево.

**Выход.** Все его вершины в трёх разных порядках: in-order, pre-order и post-order.

**Sample Input:**

10

0 7 2

10 -1 -1

20 -1 6

30 8 9

40 3 -1

50 -1 -1

60 1 -1

70 5 4

80 -1 -1

90 -1 -1

**Sample Output:**

50 70 80 30 90 40 0 20 10 60

0 70 50 40 30 80 90 20 60 10

50 80 90 30 40 70 10 60 20 0

**Решение:**

# Вместо прописывания трёх однотипных генераторов обхода in-order, pre-order и post-order

# мы прописываем лишь генераторы примитивных операций (печать потомков и ключа) и генератор обхода по генераторам

# в соответствии с заданным порядком

# Массив кортежей вида (значение, левый, правый)

tree = [tuple(map(int, input().split())) for \_ in range(int(input()))]

def print\_order(node, order):

if node != -1:

for f in order:

yield from f(node, order)

def print\_left(node, order): yield from print\_order(tree[node][1], order)

def print\_right(node, order): yield from print\_order(tree[node][2], order)

def print\_key(node, \_): yield tree[node][0]

def main():

in\_order = (print\_left, print\_key, print\_right)

pre\_order = (print\_key, print\_left, print\_right)

post\_order = (print\_left, print\_right, print\_key)

for d in in\_order, pre\_order, post\_order:

print(\*print\_order(0, d))

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

main()

**ЗАДАЧА 2**

**Условие задачи и входные выходные данные**

**Проверка свойства дерева поиска**

Проверить, является ли данное двоичное дерево деревом поиска.

**Вход.** Двоичное дерево.

**Выход.** Проверить, является ли оно корректным деревом поиска: верно ли, что для любой вершины дерева её ключ больше всех ключей в левом поддереве данной вершины и меньше всех ключей в правом поддереве.

**Sample Input:**

3

2 1 2

1 -1 -1

3 -1 -1

**Sample Output:**

CORRECT

**Решение:**

# Нужно проверить, будет ли in-order выход отсортированным. Для этого нужен только генератор и предыдущее значение.

import sys

tree = [tuple(map(int, input().split())) for \_ in range(int(input()))]

sys.setrecursionlimit(100000)

def dfs(r):

if r != -1:

yield from dfs(tree[r][1])

yield tree[r][0]

yield from dfs(tree[r][2])

def main():

if tree:

gen = dfs(0)

prev = next(gen)

for curr in gen:

if curr < prev:

print('INCORRECT')

exit()

prev = curr

print('CORRECT')

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

main()

**ЗАДАЧА 3**

**Условие задачи и входные выходные данные**

**Проверка более общего свойства дерева поиска**

**Sample Input:**

3

2 1 2

1 -1 -1

3 -1 -1

**Sample Output:**

CORRECT

**Решение:**

# Для этого пришлось написать функцию спуска по дереву с передачей максимальных и минимальных значений и проверкой в каждом узле

import sys

tree = [tuple(map(int, input().split())) for \_ in range(int(input()))]

sys.setrecursionlimit(10 \*\* 5)

def check(node=0, minv=-2 \*\* 31, maxv=2 \*\* 31 - 1):

return node == -1 or \

minv <= tree[node][0] <= maxv and \

check(tree[node][1], minv, tree[node][0] - 1) and \

check(tree[node][2], tree[node][0], maxv)

def main():

print('CORRECT' if not tree or check() else 'INCORRECT')

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

main()

**ЗАДАЧА 4**

**Условие задачи и входные выходные данные**

**Множество с запросами суммы на отрезке**

Реализуйте структуру данных для хранения множества целых чисел, поддерживающую запросы добавления, удаления, поиска, а также суммы на отрезке. На вход в данной задаче будет дана последовательность таких запросов. Чтобы гарантировать, что ваша программа обрабатывает каждый запрос по мере поступления (то есть онлайн), каждый запрос будет зависеть от результата выполнения одного из предыдущих запросов. Если бы такой зависимости не было, задачу можно было бы решить оффлайн: сначала прочитать весь вход и сохранить все запросы в каком-нибудь виде, а потом прочитать вход ещё раз, параллельно отвечая на запросы.

**Sample Input:**

15

? 1

+ 1

? 1

+ 2

s 1 2

+ 1000000000

? 1000000000

- 1000000000

? 1000000000

s 999999999 1000000000

- 2

? 2

- 0

+ 9

s 0 9

**Sample Output:**

Not found

Found

3

Found

Not found

1

Not found

10

**Решение:**

**ЗАДАЧА 5**

**Условие задачи и входные выходные данные**

**Rope**

Ваша цель в данной задаче — реализовать структуру данных Rope. Данная структура данных хранит строку и позволяет эффективно вырезать кусок строки и переставить его в другое место.

**Sample Input:**

abcdef

2

0 1 1

4 5 0

**Sample Output:**

efcabd

**Решение:**