Вариант 3

ВНИМАНИЕ!

Будьте внимательны к красным пометкам. Некоторые задания требуют уникальности, их нельзя копировать полностью. Это важно для вашей безопасности

№ 1

Задание: Связный список описан следующим классом (SimpleLinkedList)

Обратите внимание, что класс элементов списка описан внутри класса списка. Необходимо описать метод для данного списка, который удалит из списка последние к элементов списка. передается в качестве параметра. Если элементов меньше то удалить необходимо все элементы. Не забудьте при этом модифицировать поля head (при необходимости), tail size класса списка.

```
public void removeLast(int k) {
    if (k <= 0 || head == null) {
        return;
    }

    if (k >= size) {
        head = null;
        tail = null;
        size = 0;
        return;
    }

    size -= k;
    ListItem current = head;
    for (int i = 1; i < size; i++) {
        current = current.next;
    }

    tail = current;
    tail.next = null;
}
```

№ 2

Задание: Придумать и описать алгоритм, который обрабатывает двумерный массив и работает за время $O(n^2*m^2)$, где n - кол- во строк, а m - кол-во столбцов в двумерном массиве. Должна быть сформулирована задача и описан принцип ее решения (приветствуется код, хотя бы и схематичный). Решение не обязательно должно быть оптимальным, главное, чтобы оно соответствовало предложенной задаче оценка времени работы была $O(n^2*m^2)$. Объяснить, почему получилось именно $O(n^2* m^2)$. **Постарайтесь придумать именно свою уникальную задачу, если задачи будут массово повторяться, то ответ не будет засчитан**.

Решение:

Комментарий: Решение за n^2 означает, что надо искать подмассивы, дальше будет представлен пример, но его брать не стоит

Уникальная задача:

Нужно придумать задачу, в которой будет 2 раза проходить строки и 2 раза столбцы. Например, в данном двумерном массиве найти все возможные матрицы, размера 2 x 2, и посчитать сумму определителей всех таких матриц.

Формулировка:

Дан двумерный массив размером (n x m). Необходимо найти сумму определителей всех возможных подматриц размера 2x2 в заданном двумерном массиве.

Решение:

```
public static int solve(int[][] arr){
    int sum = 0;
    for(int i = 0; i < arr.length-1; i++){
        for (int j = 0; j < arr[i].length-1; j++){
            for (int i2 = i+1; i2 < arr.length; i2++){
                for (int j2 = j+1; j2 < arr[i2].length; j2++){
                    sum += arr[i][j]*arr[i2][j2]-arr[i2][j]*arr[i][j2];
                }
            }
        }
    }
    return sum;
}
```

Пояснение:

У нас 4 вложенных for, что соответствовало бы сложности $O(n^4)$, но это было бы корректно, если строки равны столбцам, в нашем случае, в for со сложностью O(n) вложен for со сложностью O(m), в него вложен for со сложностью O(n), а в него for со сложностью O(m). И того мы получаем сложность $O(n^2*m^2)$

№ 3

Задание: Расписать по шагам работу алгоритма пирамидальной сортировки (Heap Sort) на следующем наборе данных:

19	1	0	16	13	4	19	12	6	18

Код пирамидальной сортировки приводить не следует, это не будет считаться правильным ответом.

Решение:

Общее описание работы сортировки:

Сортировка состоит из двух основных шагов:

Шаг 1: Построение кучи

Берём массив и превращаем его в кучу. Для этого:

1. Проходим по элементам и "просеиваем" их вверх или вниз, чтобы выполнялось условие кучи.
2. В итоге самый большой элемент оказывается в корне (первый элемент массива).

Алгоритм просеивания вниз:

1. Начинаем с последнего родителя (индекс n//2 — 1).
2. Для текущего элемента проверяем:
    1. Левый ребёнок = 2*i + 1
    2. Правый ребёнок = 2*i + 2
3. Если один из детей больше родителя – меняем их местами.
4. Повторяем для нового положения элемента, пока он не станет больше своих детей или не упрётся в лист.

Шаг 2: Извлечение элементов

Теперь по очереди:

1. Меняем местами первый (максимальный) и последний элементы.
2. "Отрезаем" последний элемент (он теперь на своём месте).
3. Просеиваем новый корень вниз, чтобы восстановить кучу. (Т.е. выполняем шаг 1)
4. Повторяем, пока куча не закончится.

№ 4

Задание:

Класс двоичного дерева описан следующим образом (SimpleTree)

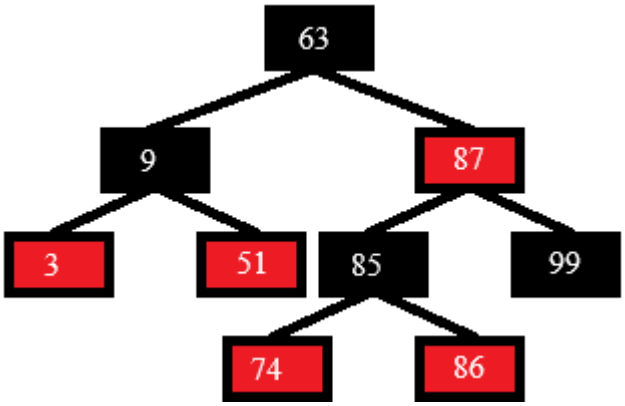
Обратите внимание, что класс для элемента дерева внутри класса дерева. Необходимо для данного дерева описать метод, который определяет и возвращает высоту дерева. При необходимости можно описать вспомогательные методы.

```
public int getHeight() {
    return calculateHeight(root);
}

private int calculateHeight(TreeItem node) {
    if (node == null) {
        return 0;
    }
    int leftHeight = calculateHeight(node.left);
    int rightHeight = calculateHeight(node.right);

    return Math.max(leftHeight, rightHeight) + 1;
}
```

№ 5

Задание:

У вас есть представленное ниже красно-черное дерево.



В это дерево вставляется значение 80. Распишите по шагам, как будет происходить эта операция, включая балансировку дерева после вставки - подразумевается несколько рисунков после вставки и каждого поворота. Укажите, какие именно повороты для каких элементов будут выполняться. Естественно, итоговый вид дерева также должен быть изображен.

Решение:

Основные свойства красно-черных деревьев

Каждый узел либо красный, либо черный

Корень всегда черный

Все листья (NIL) считаются черными

У красного узла оба потомка черные (нет двух красных узлов подряд)

Для каждого узла все пути от него до листьев содержат одинаковое количество черных узлов

Балансировка дерева:

Балансировка происходит после вставки или удаления узла и включает:

Перекрашивание узлов

Повороты поддеревьев (левый и правый)

Алгоритм балансировки (когда родитель красный):

Для каждого случая рассматриваем:

P - родитель нового узла

G - дед (родитель родителя)

U - "дядя" (брат родителя)

Случай 1: Дядя (U) красный

Перекрасить родителя (P) в черный

Перекрасить дядю (U) в черный

Перекрасить деда (G) в красный

Рассмотреть деда (G) как новый нарушающий узел

Продолжить проверку с G

Случай 2: Дядя (U) черный или отсутствует, и новый узел - "внутренний" потомок

(Т.е. новый узел - правый потомок для левого родителя или левый потомок для правого родителя)

Выполнить поворот вокруг родителя (P) в противоположную сторону от нового узла

Если новый узел справа → левый поворот P

Если новый узел слева → правый поворот P

Теперь рассматриваем бывший родитель (P) как новый узел

Перейти к случаю 3

Случай 3: Дядя (U) черный или отсутствует, и новый узел - "внешний" потомок

(Т.е. новый узел - левый потомок для левого родителя или правый потомок для правого родителя)

Перекрасить родителя (P) в черный

Перекрасить деда (G) в красный

Выполнить поворот вокруг деда (G) в противоположную сторону от родителя

Если P левый потомок → правый поворот G

Если P правый потомок → левый поворот G

Правила поворотов

Левый поворот (X):

Сделать правого потомка X (Y) новым корнем поддерева

Переместить левое поддерево Y в качестве правого поддерева X

Сделать X левым потомком Y

Правый поворот (Y):

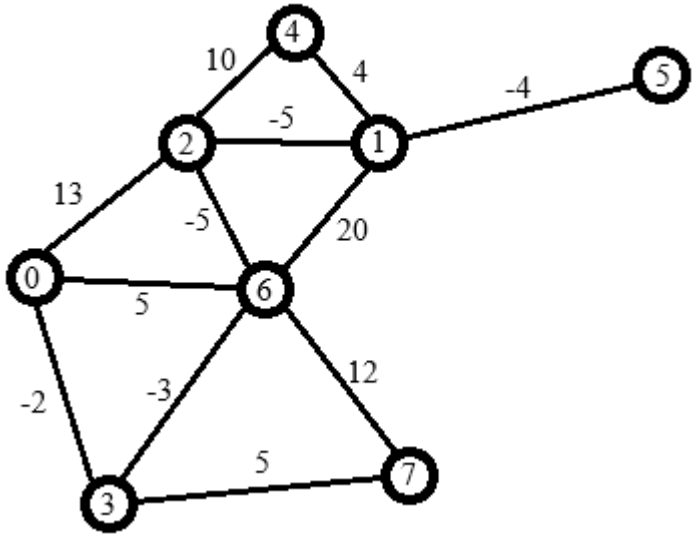
Сделать левого потомка Y (X) новым корнем поддерева

Переместить правое поддерево X в качестве левого поддерева Y

Сделать Y правым потомком X

№ 6

Задание: Для графа ниже расписать, как будет работать алгоритм Беллмана-Форма. Стартовой вершиной является вершина 0. Обратите внимание на отрицательные веса некоторых ребер.



Решение:

Алгоритм Беллмана-Форда — это способ найти кратчайший путь от одной точки до всех остальных в графе, даже если в нём есть отрицательные веса рёбер

Как он работает:

- Стартовой вершине присваивается расстояние 0, остальным — ∞ (бесконечность).
- Алгоритм проходит по всем рёбрам графа и пытается улучшить (уменьшить) расстояние до вершин.
- Повторяет это N-1 раз (где N — число вершин), чтобы гарантировать, что найден кратчайший путь.
- Если после N-1 проходов можно ещё улучшить расстояние — значит, в графе есть отрицательный цикл, и кратчайшего пути не существует (можно "зациклиться" и уменьшать путь бесконечно).

т. е. мы проходим до каждой вершины, просто разными путями и выбираем меньший. Ответ должен выглядеть так:

dist[0] = 0

dist[1] = -15

dist[2] = -10

dist[3] = -2

dist[4] = 0

dist[5] = -19

dist[6] = -5

dist[7] = 3

Отсюда, можно понять, что путь от 0 до 5 имеет длину -19 (0 → 3 → 6 → 2 → 1 → 5), это самая лучшая длина.