# Отчет

(Каганович Дмитрий)

[Задание 1 1](#_Toc469163470)

[Задание 2 2](#_Toc469163471)

[Задание 4 3](#_Toc469163472)

[Задание 5 3](#_Toc469163473)

## Задание 1

За можно построить структуру размером , представляющую собой последовательность :

которая для каждого блока исходных данных хранит значение соответствующей частичной суммы элементов последовательности .

В момент поступления запроса высчитываются его левая () и правая () блочные границы:

После этого для ответа на запрос необходимо вычислить:

для чего потребуется не более 4 чтений с диска.

## Задание 2

В дополнение к исходным данным построим структуру, состоящую из двух последовательностей .

Первая из последовательностей имеет вид:

и хранит для каждого блока исходных данных значение минимального элемента соответствующей подпоследовательности последовательнсти . IO сложность операции .

Вторая последовательность:

хранит минимальный элемент для каждого блока данных последовательности . Время построения .

Дополнительно за построим для по степеням двойки. Каждый уровень (всего ) полученной можно хранить в отдельном файле.

Суммарное IO время построения структуры – . Размер структуры удовлетворяет требуемой асимптотике: действительно, .

При обработке запроса сперва вычисляются блочные границы запроса для блоков размерами и :

После этого ответ представляется в виде:

Компонента в выражении выше определяется из не более, чем за 2 операции чтения с диска: если длина отрезка не является степенью двойки, то отрезок логически разбивается на два; затем, по отступам, соответствующим началам отрезков, вычитываются данные из уровней , соответствующих длинам отрезков. Поэтому, в общем потребуется не более 6 чтений с диска для ответа на запрос.

## Задание 4

Воспользуемся модификацией алгоритма Крускала. Представим исходный файл как поток входных данных и будем обрабатывать ребра пачками размером до (это возможно, т.к. ). Один шаг алгоритма заключается в запуске на уже сформированном остовном лесу (первоначально – пустое множество) и на вновь прибывших ребрах алгоритма Крускала. На выходе будет получен новый остовный лес размером не больше . После этого из потока дочитывается следующая пачка ребер и производится следующий шаг алгоритма. Утверждается, что в результате мы построим минимальное остовное дерево исходного графа. Поскольку алгоритм Крускала запускается не чаще, чем каждые ребер, то и число процессорных операций будет:

Докажем корректность работы предложенного алгоритма. Очевидно, что алгоритм строит остовное дерево (нет циклов, связность всех вершин). Докажем, что оно минимальное. От противного. Пусть построенное алгоритмом остовное дерево не минимальное Тогда, из всего множества минимальных остовных деревьев выберем дерево , отличающееся от в минимальном числе ребер. Пусть – ребро, которое было добавлено алгоритмом Крускала в , но отсутствует в . Очевидно, что образует цикл, причем в этом цикле найдется ребро , которого нет в (иначе в был бы цикл). Рассмотрим новое остовное дерево . Вес нового дерева не может быть больше веса , так как есть минимальное остовное дерево. Поэтому вес . С другой стороны, алгоритм Крускала добавил в . Следовательно, вес не превосходит весов других ребер в цикле, и значит, вес . Отсюда получаем, что вес и также является минимальным остовным деревом. Но отличается от в меньшем числе ребер, чем , т.е получили противоречие с выбором .

## Задание 5

Предполагаем, уникальную нумерацию всех вершин .

Первоначально определим множество корней деревьев в лесу . Для этого сортируем ребра по 1-й компоненте, затем по 2-й компоненте, затем делаем двух этих множеств и выбираем вершины у которых нет родителя. В результате получим множество корней . Добавим к этому множеству фиктивную вершину .

Для каждого ребра исходного орграфа добавим противоположное. Далее, получившееся множество ребер дополним ребрами, идущими от фиктивной вершины к корням и от корней к фиктивной вершине. Запустим на итоговом множестве ребер – получим Эйлеров обход графа (пусть стартовая вершина совпадает с ).

После этого начинаем сканирование полученного обхода. Пусть есть метка корня. Если в какой-то момент вершина, предшествующая рассматриваемой, совпадает с фиктивной, то рассматриваемая вершина является корнем, и дальше будут следовать вершины поддерева с этим корнем. В этом случае метка устанавливается равной номеру корня и распространяется всем вершинам поддерева. На выходе из поддерева мы снова попадем в фиктивную вершину и на следующем шаге поменяем значение метки на номер корня следующего поддерева и т.д.. В результате получим список вершин с соответствующими им номерами корней.

Из полученного списка удалим записи о фиктивной вершине. После этого избавимся от дубликатов. Для этого сортируем список по значению (по номеру вершины). Затем сканируем его и оставляем лишь записи с последним вхождением номера вершины.

Поскольку имеет IO сложность , то и весь алгоритм асимптотически отработает за .