Отчет

(Каганович Дмитрий)

ЗАДАНИЕ 1 1

ЗАДАНИЕ 2 2

ЗАДАНИЕ 4 3

ЗАДАНИЕ 5 3

Задание 1

За $\mathrm{O}(\mathrm{Scan}(\mathrm{N}))$ можно построить структуру размером $\mathrm{O}\left(\frac{\mathrm{N}}{B}\right)$, представляющую собой последовательность $\{S_i\}$:

$$S_1 = \sum_{j=1}^{\min(B,N)} a_j \text{, а } S_{\overline{i=2,\left\lceil\frac{N}{B}\right\rceil}} = S_{i-1} + \sum_{j=(i-1)\times B+1}^{\min(i\times B,N)} a_j$$

которая для каждого блока исходных данных хранит значение соответствующей частичной суммы элементов последовательности $\{a_i\}$.

В момент поступления запроса $\{i,j\}$ высчитываются его левая (l) и правая (r) блочные границы:

$$l = \left\lceil\frac{i}{B}\right\rceil, r = \left\lceil\frac{j}{B}\right\rceil$$

После этого для ответа на запрос необходимо вычислить:

$$Ans = S_r - S_l + \sum_{k=i}^{\min(l\times B,N)} a_k - \sum_{k=j+1}^{\min(r\times B,N)} a_k$$

для чего потребуется не более 4 чтений с диска.

Задание 2

В дополнение к исходным данным построим структуру, состоящую из двух последовательностей $\{S_i^1\}$ и $\{S_i^2\}$, а также $SparseTable(ST)$ для второй последовательности.

Первая из последовательностей имеет вид:

$$S_i^1 = min_{j=\overline{(i-1)\times B+1, \min(i\times B, N)}} a_j, i = \overline{1, \left\lceil \frac{N}{B} \right\rceil}$$

и хранит для каждого блока исходных данных значение минимального элемента соответствующей подпоследовательности последовательнсти $\{a_i\}$. IO сложность операции $O(Scan(N))$.

Вторая последовательность:

$$S_i^2 = min_{j=\overline{(i-1)\times B+1, \min(i\times B, \left\lceil \frac{N}{B} \right\rceil)}} S_j^1, i = \overline{1, \left\lceil \frac{N}{B^2} \right\rceil}$$

хранит минимальный элемент для каждого блока данных последовательности $\{S_i^1\}$. Время построения $O\left(Scan\left(\frac{N}{B}\right)\right)$.

Дополнительно за $O\left(log_2\left(\left\lceil \frac{N}{B^2} \right\rceil\right) \times \text{Scan}\left(\frac{N}{B^2}\right)\right)$ построим для $\{S_i^2\}$ ST по степеням двойки. Каждый уровень (всего $log_2\left(\left\lceil \frac{N}{B^2} \right\rceil\right)$ штук) полученной ST можно хранить в отдельном файле.

Суммарное IO время построения структуры – $O\left(log_2\left(\left\lceil \frac{N}{B^2} \right\rceil\right) \times \text{Scan}(N)\right)$. Размер структуры удовлетворяет требуемой асимптотике: действительно, $O\left(\left\lceil \frac{N}{B} \right\rceil\right) + O\left(log_2\left(\left\lceil \frac{N}{B^2} \right\rceil\right) \times \left\lceil \frac{N}{B^2} \right\rceil\right) = [log_2 N < B] = O\left(\frac{N}{B}\right) + O\left(C \times \frac{N}{B}\right) = O\left(\frac{N}{B}\right)$.

При обработке запроса $\{i, j\}$ сперва вычисляются блочные границы запроса для блоков размерами B и B^2:

$$l_B = \left\lceil \frac{i}{B} \right\rceil, r_B = \left\lceil \frac{j}{B} \right\rceil$$

$$l_{BB} = \left\lceil \frac{i}{B^2} \right\rceil, r_{BB} = \left\lceil \frac{j}{B^2} \right\rceil$$

После этого ответ представляется в виде:

$$Ans = \min\{min_{k=\overline{i, \min(j, l_B\times B)}} a_k, min_{k=\overline{l_B+1, \min(r_B-1, l_{BB}\times B)}} S_k^1, min_{k=\overline{l_{BB}+1, r_{BB}-1}} S_k^2,$$

$$min_{k=\overline{\max((r_{BB}-1)\times B+1, l_B+1), r_B-1}} S_k^1, min_{k=\overline{\max((r_B-1)\times B+1, i), j}} a_k\}$$

Компонента $min_{k=\overline{l_{BB}+1, r_{BB}-1}} S_k^2$ в выражении выше определяется из ST не более, чем за 2 операции чтения с диска: если длина отрезка $[l_{BB}+1, r_{BB}-1]$ не является степенью двойки, то отрезок логически разбивается на два; затем, по отступам, соответствующим началам отрезков, вычитываются данные из уровней ST, 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 $2V$ (это возможно, т.к. $|\mathrm{V}| < \mathrm{M}$). 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 $|\mathrm{V}|$. 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 $|\mathrm{V}|$ ребер, то и число процессорных операций будет: $\mathrm{O}(|\mathrm{V}| \times \log|\mathrm{V}|) \times \frac{|E|}{|V|} = \mathrm{O}(|\mathrm{E}| \times \log|\mathrm{V}|)$

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 T не минимальное. Тогда, из всего множества минимальных остовных деревьев выберем дерево T_1, отличающееся от T в минимальном числе ребер. Пусть e – ребро, которое было добавлено алгоритмом Крускала в T , но отсутствует в T_1. Очевидно, что $T_1 \cup e$ образует цикл, причем в этом цикле найдется ребро f, которого нет в T (иначе в T был бы цикл). Рассмотрим новое остовное дерево $T_2 = T_1 \cup e \setminus f$. Вес нового дерева T_2 не может быть больше веса T_1, так как T_1 есть минимальное остовное дерево. Поэтому вес $e \geq f$. С другой стороны, алгоритм Крускала добавил e в T. Следовательно, вес e не превосходит весов других ребер в цикле, и значит, вес $e \leq f$. Отсюда получаем, что вес $e = f$ и T_2 также является минимальным остовным деревом. Но T_2 отличается от T в меньшем числе ребер, чем T_1, т.е получили противоречие с выбором T_1.

Задание 5

Предполагаем, уникальную нумерацию всех вершин G.

Первоначально определим множество корней деревьев в лесу $G = (V, E)$. Для этого сортируем ребра по 1-й компоненте, затем по 2-й компоненте, затем делаем $Join$ двух этих множеств и выбираем вершины у которых нет родителя. В результате получим множество корней $\{Roots\}$. Добавим к этому множеству фиктивную вершину v^+: $\{Roots\} = \{Roots\} \cup v^+$.

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 $ListRanking$ на итоговом множестве ребер – получим Эйлеров обход графа (пусть стартовая вершина совпадает с v^+).

После этого начинаем сканирование полученного обхода. Пусть $root$ 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 $root$ 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 $ListRanking$ имеет IO сложность $\mathrm{O}(Sort|V|)$, то и весь алгоритм асимптотически отработает за $\mathrm{O}(Sort|V|)$.