Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3230

Асташин С.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача I «Машинки»

Пояснение к примененному алгоритму:

Для начала сохраним порядок машинок (`vector<int> order(p)`) и, для каждого типа машинок, сохраним индексы, на которых встречаются машинки этого типа (`vector<list<int>> distances(n)`). Также, создадим `map<int, int> floor` для хранения машинок, которые сейчас находятся на полу (значение – тип, ключ – индекс следующей машинки такого типа). Далее будем итерироваться по order и действовать следующим образом:

1.1. Если на полу нет машинки с индексом `distances[order[i] - 1].front()`, увеличить счётчик на один. Если нет места, то убрать последний элемент в `floor` (то есть уберётся тот тип, следующая машинка которого находится наиболее далеко в `order`);

1.2. Если на полу есть машинка с таким индексом, то убрать её из `floor` (необходимо, поскольку будем обновлять её индекс).

2. Удалить индекс из `distances` (`distances[order[i] - 1].pop_front()`).

3.1. Добавить машинку на пол по новому индексу, если индексы ещё остались;

3.2. Если индексов нет, добавить машинку по индексу 500000 + `order[i]` (значение для каждого типа будет уникальным, поскольку P ≤ 500000). Это нужно, чтобы не потерять возможность определить приоритет машинок такого типа.

4. Ответом будет значение счётчика.

Оценка сложности: O(p) * (O(log(k) + O(log(k)) = O(p*log(k)).

Задача J «Гоблины и очереди»

Пояснение к примененному алгоритму:

Единственная проблема этой задачи – вставка в середину. Поскольку для `std::list` это очень небыстрая операция, пришлось реализовать её самому, сделав два списка – `left` и `right`.

Алгоритм работает так:

1.1. Если операция “+”, то элемент добавляется в конец правой части;

1.2. Если операция “*”, то элемент добавляется в начало правой части;

1.3. Если операция “-“, то удаляется начало левой части.

2. Если правая часть больше левой, переместить её начало в конец левой части.

Поясню второй пункт: такое перемещение необходимо, чтобы длина левой части всегда была больше длины правой на 1 или равна ей. По сути, можно было сделать и наоборот, просто мне так показалось логичнее (естественно, если делать наоборот, то действия для операций следует поменять).

Оценка сложности: O(n).

Задача K «Менеджер памяти-1»

Пояснение к примененному алгоритму:

Заведём три контейнера: map<int, int> by_start, multimap<int, int> by_size, pair<int, int> *history. Первые два предназначены для хранения информации о свободных блоках (в первом ключ – индекс первого элемента блока, значение – размер блока; во втором – наоборот), третий – для ведения истории операций (первое число – индекс первого элемента блока или -1, второе – размер). Изначально имеем один блок с началом 1 и размером n.

Для того, чтобы занять блок размером capacity:

1. Ищем блок размера ≥capacity в by_size (by_size.lower_bound(capacity);
2. Если блок нашёлся, то возвращаем его начало. При этом если блок больше, чем capacity, то добавляем в by_size и by_start новый блок размером size - capacity, идущий за найденным.

Для того, чтобы освободить блок:

1. Находим соседей блока: next = by_start.lower_bound(start), prev = std::prev(next);
2. Проверяем, что это действительно соседи (next->first == start + size, start == prev->second + prev->first, где start и size – параметры блока для освобождения);
3. Объединяем свободные блоки друг с другом (три варианта: объединяем правого соседа с блоком, объединяем левого соседа с блоком или объединяем блок с обоими соседями).

Оценка сложности: Честно говоря, непонятно, как считать. Очень много разветвлений и крайних случаев.

Задача L «Минимум на отрезке»

Пояснение к примененному алгоритму:

Заведём два листа для нашего “окна”: в `list<int> window` будем хранить числа, а в `list<int> indexes` – их порядковые номера. Главная идея алгоритма заключается в том, чтобы для каждого числа перед добавлением его в окно удалить из окна все числа, больше него. Если работать с числами по такой схеме, то первый (крайний левый) элемент в окне) и будет минимумом на отрезке. Старый минимум удаляется тогда, когда окно отдаляется от него на k позиций.

Покажу на примере. Допустим, входные данные такие:

7 3

1 3 2 4 5 3 1

Тогда `window` для разных положений окна (зелёным раскрашены числа, вошедшие в список, красным – удалённые):

1. 1 3 2 4 5 3 1
2. 1 3 2 4 5 3 1
3. 1 3 2 4 5 3 1
4. 1 3 2 4 5 3 1
5. 1 3 2 4 5 3 1

Оценка сложности: O(n) * O(k) = O(n*k).