## Домашнее задание

**Преамбула.** Поскольку мы изучили хэш-таблицы, операции в которых стоят $O(1)$ за счёт использования вероятности, хотя в худшем случае это не так, в этом задании мы считаем, что когда требуется найти алгоритм со сложностью $O(f(n))$, операции с хэш-таблицами при их использовании стоят $O(1)$.

**1.** В языке C++ структура данных `std::set` реализована с помощью красно-чёрных деревьев. В ней хранится множество ключей, при этом считается, что ключи упорядочены. Помимо стандартных операций (вставки, поиска, удаления), которые стоят $O(\log n)$, где $n$ — число элементов в структуре данных, эта структура данных позволяет вывести все элементы в отсортированном порядке за $O(n)$. Опишите, как модифицировать красно-чёрное дерево так, чтобы сделать возможным такой вывод и не изменить стоимость стандартных операций. Считайте, что балансировка дерева после добавления и удаления элемента стоит $O(\log n)$.

Семейство хэш-функций $\mathcal{H}$ называется *универсальным*, если для любых фиксированных различных $x$ и $y$ вероятность события $f(x) = f(y)$ равна $1/m$, где $f$ выбирается случайно и равновероятно из $\mathcal{H}$ и отображает $n$-элементное множество (пространтство ключей) в $m$-элементное множество.

**2.** Семейство хэш-функций $H = \{f, g, h\}$, отображающих множество ключей $\{0, 1, 2, 3\}$ в множество хэшей $\{0, 1, 2\}$, задано таблицей. Является ли оно универсальным?

| ключи | значения $f$ | значения $g$ | значения $h$ |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 2 | 2 |
| 2 | 1 | 2 | 2 |
| 3 | 1 | 1 | 0 |

**3.** В структуре данных «динамический массив» (в C++ `std::vector`) удаление $k$-го элемента стоит $O(n - k)$, где $n$ — количество элементов в массиве. Это происходит за счёт сдвигов: необходимо, чтобы первый элемент динамического массива находился в первой ячейке выделенной под массив памяти, а также нужно, чтобы между соседними элементами не было пустых ячеек памяти. Поэтому после очищения $k$-ой ячейки в результате удаления требуется последовательно сдвинуть $n - k$ последних элементов на единицу влево.

Так, удаление последнего элемента стоит $O(1)$, поэтому с помощью динамических масивов стандартно реализует стеки, а удаление первого элемента стоит $O(n)$, поэтому использовать динамические массивы для наивной реализации очереди (при извлечении из очереди первый элемент извлекают из массива, а потом удаляют, а новые элементы добавляются в конец массива) — не лучшее решение.

Однако, с помощью динамического массива можно эффективно реализовать очередь поплатившись двукратным увеличением памяти.Как и в случае наивной реализации, элементы будут извлекаться из начала массива $A$, а добавляться будут в его конец. Заведём указатель (целочисленная перменная $b$), который сначала указывает на первый элемент массива ($b = 0$). Когда требуется извлечь элемент из очереди, сам элемент физически удаляться не будет, а будет увеличиваться лишь указатель: при извлечении из очереди возвращается элемент $A[b]$, после чего $b$ увеличивается на единицу. В каждый момент, когда $b \geqslant \lfloor \frac{n}{2} \rfloor$ ($n$ — число элементов в массиве $A$), происходит удаление $b$ элементов из массива, после чего $b$ присваивается значение 0.

Уточните описанный выше алгоритм так, чтобы в результате $n$ любых запросов к структуре данных «очередь» выполнялось $O(n)$ операций и при этом выделяемая под динамический массив память не превосходила удвоенную память, требуемую для хранения очереди.

**Указание**. Для доказательства оценки используйте амортизационный анализ.

**4.** Опишите реализацию структуры данных «очередь с приоритетами» (из классного листка) со следующими стоимостями операций

- `insert` — $O(1)$,
- `extract_max` — $O(n)$,
- `set_priority` — $O(1)$.

При этом известно, что все ключи — уникальные числа в диапазоне от 1 до $n$.

**5.** 1. Постройте оптимальный алгоритм, который находит минимальный элемент в куче на максимум.

2. Докажите, что ваш алгоритм оптимальный: если ваш алгоритм работает за время $O(f(n))$, то любой алгоритм работает за время $\Omega(f(n))$. Считайте, что алгоритм не знает элементы заранее, куча хранится в памяти как массив $a$ и алгоритм может за один запрос $i$ узнать элемент $a[i]$.

**6.** Дан массив из $n+1$ элемента, который содержит элементы от 1 до $n$ и известно, что каждое число от 1 до $n$ встречается хотя бы один раз.

1. Докажите, что какое-то число от 1 до $n$ встречается два раза.

2. Предложите алгоритм, который находит дубликат за $O(n)$.

3. Предложите алгоритм, который находит дубликат за $O(n)$ и $O(1)$ дополнительной памяти.

**7.** Постройте алгоритм, который получив на вход числовой массив выводит количество его подмассивов (непрерывных подпоследовтаельностей), в которых все элементы различны.

**8.** Структура данных «таблица» представляет собой числовую матрицу размера $m \times n$ (двумерный массив), элементы которой в каждой строке и каждом столбце отсортированы по возрастанию. В случае, если в таблице меньше $mn$ элементов, в незаполненных клетках написано $\infty$.

1. Постройте алгоритм, который проверяет находит элемент $x$ в частично заполненную таблицу за $O(m+n)$ или проверяет, что такого элемента в таблице нет.

2. Постройте алгоритм, вставляющий новый элемент в частично заполненную таблицу за $O(m+n)$.

3*. Постройте алгоритм, удаляющий элемент из частично заполненной таблицы за $O(m+n)$.