Домашнее задание по АиСД №2

Эмиль Гарипов M3138

2019-09-28

Задача №1

(a)

В худшем случае increment пройдется по всему массива длины k и поменяет все значения, кроме $a[k-1]$ на 0, а $a[k-1]$ поменяет на 1. Таким образом в худшем случае increment работает за k.

(b)

Среднее время работы операций a вычисляется по формуле $a = \frac{\sum\limits_{i=0}^{n} t_i}{n}$, где t_i — фактическое время выполнения операции i.

Для начала заметим, что increment меняет нулевой бит числа в каждой операции, первый - в каждой второй, второй - в каждой четвертой и т д. То есть каждый i-й бит меняется каждые 2^i операций. Тогда средняя стоимость операций increment $a = \frac{\sum\limits_{i=0}^{k-1} \frac{n}{2^i}}{n}$. Так как $\sum\limits_{i=0}^{k-1} \frac{n}{2^i} < 2n$, получаем: среднее время работы операций increment $a = \mathcal{O}(1)$

(c)

Докажем, что n операций increment и decrement выполняются за $\mathcal{O}(nk)$. А после докажем, что выполняются за $\Omega(nk)$. Тогда, имея верхнюю и нижнюю границу, получим, что они работают за $\Theta(nk)$.
1)Нижняя граница
В худшем случае операции increment и decrement работают за k. Таким образом при выполнении n операций получаем верхнюю границу $\mathcal{O}(nk)$.
2)Верхняя граница
Так как операции increment и decrement в худшем случае могут пройти весь массив, длина которого k и поменять все его значения, они не могут сделать больше k операций. Тогда при выполнении n операций increment и decrement они сделают не больше nk операций. То есть работают за $\Omega(nk)$

Итого, операции выполняются за $\Theta(nk)$.

(d)

Операция get выполняется за $\mathcal{O}(1)$, а setZero за $\mathcal{O}(k)$ поэтому модифицируем только операции setZero и increment следующим образом: будем хранить две переменные — l, r — начало и конец отрезка, где сейчас стоят нули после последних взаимодействий с массивом. При вызове операции setZero весь наш массив заполнен нулями, то есть $l = 0$, $r = k-1$. Во время работы операции increment меняется только какой-то префикс массива, поэтому при изменении очередного элемента просто будем двигать левую границу

l. Таким образом операция increment так же работает за $\mathcal{O}(1)$, операция setZero с модификацеей работает за $\mathcal{O}(1)$, при этом мы используем $\mathcal{O}(1)$ дополнительной памяти.

Задача №2

Функция потенциала после выполнения i-й операции $\varphi(i) = |2s - c|$, где s — количество элементов в стеке, c — размер выделенного массива. Проанализируем стоимости операций:
1)Операция добавляения нового элемента в стек: $a_{push} = 1 + |2s + 2 - c| - |2s - c| = 1 + 2 = 3 = \mathcal{O}(1)$
2)Операция удаления элемента из стека: $a_{pop} = 1 + |2s - 2 - c| - |2s - c| = 1 - 2 = -1$. Добавим еще какую-нибудь константу, например 1, чтобы получить положительное значаение. Тогда $a_{pop} = \mathcal{O}(1)$
3)Операция копирования стека при заполнении всего выделенного массива: $a_{copy} = s + |2s - 2c| - |2s - c| = s + 0 - c = 0 = \mathcal{O}(1)$
4)Операция сужения стека при заполнении менее $\frac{1}{4}$ выделенного массива: $a_{comp} = s + |2s - \frac{c}{2}| - |2s - c| = s + 0 - |0.5c - c| = s - 0.5c = -s = \mathcal{O}(1)$
Так как все операции выполняются $\mathcal{O}(1)$, амортизированное время работы тоже $\mathcal{O}(1)$.

Задача №3

Создадим еще один массив b длины n. Будем хранить счетчик sz,который будет указывать на свободный элемент.

Операция Set

При операции set(i, x)(присвоить i-му элементу массива a значение x) в i-й элемент массива a запишем значение нашего счетчика, указывающего на свободный элемент в b, и увеличим sz. Так же в свободную ячейку массива b, на которую указывает наш счетчик, сохраним следуюущие значения : индекс, в который произошло присвоение, и само присваиваемое значание. Это позволит нам при запросе get(i) понимать, храниться ли что-то в i-ой ячейке или ей еще не было присвоено значение.

Операция Get

Операция get(i) выглядит следующим образом: Достаем значение из массива a по данному в запросе индексу. Чтобы убедиться, что это уже поставленное нами значение, а не 'мусор', оставшийся впоследствие отсутствия инициализации, сравним i и sz. Если $sz > i$, то i указывает на уже существующее значение массива b. Теперь, чтобы убедиться, что значение в массиве a действительно установлено нами, достаточно лишь проверить, куда указывает значение (назовем его j) ячейки из массива b(ячейка хранит индекс, в который было выполнено присвоение, и само присваиваемое значение), в которое мы перешли из значения i-го элемента массива a. J должно указывать на i-й элемент массива a. И тогда мы просто возвращаем ранее сохраненное нами значение присваиваемое значение. Иначе возращаем 0.

Такая структура позволяет пользоваться неинициализированной памятью.

Задача №5

Решение: Пусть элемент, который встречается в массиве длины n больше $\frac{n}{k}$ раз — x(таких элементов может быть несколько). Тогда если разбить элементы массива на группы по k элементов таким образом, чтобы в каждой группе все элементы были различны, какие-то элементы x не попадут ни в одну. Так как количество x-ов $> \frac{n}{k}$.

Тогда будем разбивать элемнты на группы(но не будем хранить их явно) следующим алгоритмом:

```
#чтение массива a
cnt = [0] * (k - 1)
d = [0] * (k - 1)
for i in range(n):
    bool flag = False
    for j in range(k - 1):
        if (cnt[j] == 0 or d[j] == a[i]):
            d[j] = a[i]
            cnt[j] = cnt[j] + 1
            flag = True
    if (flag):
        break
    for j in range(k - 1):
        cnt[j] = cnt[j] - 1
```

$cnt[i]$ — количество i-го элемента в сформированной нами группе из $k-1$ элементов, $d[i]$ - элементы группы.

В ходе работы алгоритма пытаемся найти группу для i-го элемента из ранее обработанных элементов. Группа сушествует, если у нас уже есть $k-1$ элемент.

После работы алгоритма, элементы, встречающиеся в массиве больше чем $\frac{n}{k}$ раз будут иметь значение $cnt[i]$ больше нуля, так как для них не найдется группы.

Итого: алгоритм работает за $\mathcal{O}(nk)$ и использует $\mathcal{O}(k)$ памяти.