Алгоритмы и Структуры Данных ДЗ-2

Гарипов Роман М3138

29.09.2019

Задача №1

(a)

В худшем случаем while пройдет по всем $i = 0 \dots k-1$, то есть k операций. Получаем истинное время работы в худшем случае $\Theta(k)$.

(b)

Заметим, что первый бит меняет свое значение в каждую операцию, второй бит в $\frac{n}{2}$ операциях, третий бит в $\frac{n}{4}$ операциях, i-ый бит в $n \cdot 2^{-i}$ операциях. Получаем, что для фикисированного n $increment$ отработает суммарно за :

$$\sum_{i=0}^{k} \frac{n}{2^i} = n + \dots + \frac{n}{2^k} = \frac{n \cdot 2^k + n \cdot 2^{k-1} + \dots + n}{2^k} = \frac{n \cdot (2^k + \dots + 2^0)}{2^k} = \frac{n \cdot (2^{k+1} - 1)}{2^k}$$

Учитывая то, что $2^k \approx n, 2^{k+1} \approx 2n$, получаем :

$\frac{n \cdot (2n-1)}{n} = 2n - 1 = \mathcal{O}(n)$. Тогда амортизированная стоимость одной операции $\mathcal{O}(1)$.

(c)

Операция decrement в худшем случае работает за $\Theta(k)$ ровно по тем же самым соображениям, что и increment.
Тогда если проделать n операций, стоимость каждой их которых в худшем случае $\Theta(k)$, получим $\Theta(nk)$.

(d)

Заведём одну переменную pos(изначально равную inf), в которой будем хранить конец префикса массива $a[\,]$ в котором стоит актуальная информация. Будем двигать ее после каждого обращения в первый неактуальный элемент массива, а в сам неактуальные элемент запишем 0, чтобы там была актуальная информация. Когда необходимо выполнить $setZero$, просто скажем что актуальный префикс теперь пустой, то есть вся информация в массиве неактуальна, так как там должны стоять нули, но явно их проставлять мы не будем, а только *лениво* в $get(i)$. Тогда, чтобы операция $get(i)$ работала правильно, реализуем $get(i)$, $setZero$ и $increment()$ следующим образом :

```
pos = MAX_INT
get(i):
    if i > pos:
        a[i] = 0
        pos++
    return a[i]
```

```
setZero():
    a[0] = 0
    pos = 0

increment():
    i = 0
    while i < k and get(i) = 1:
        a[i] = 0
        i++
    if i < k:
        a[i] = 1
        pos = max(pos, i)
```

В операции *decrement* аналогично заменим обращение к элементу массива на $get(i)$, а так же в последнем if'е можно уменьшить pos.
Поскольку $get(i)$ работает за $\mathcal{O}(1)$, ассимпотика для *increment* и *decrement* не изменилась, а $setZero$ работает за $\mathcal{O}(1)$. Получаем, что при использовании $\mathcal{O}(1)$ памяти, имеем амортизированное время работы для всех операций равное $\mathcal{O}(1)$.

Замечание

Важно, что $get(i)$ всегда будет вызываться либо от тех элементов, в которые мы уже лениво поставили ноль, либо от первого с неактуальной информацией. Поэтому мы можем двигать *pos* каждый раз, когда смотрим в неактуальный элемент.

Задача №2

Выберем в качестве потенциала функцию $\Phi_i(S, C) = |2S - C|$, где S - колво элементов в стэке, C - колво ячеек выделенных для стека.

Push

$a_{push} = 2 + |2(S + 1) - C| - |2S - C| = 2 + 2 = 4$

Pop

$a_{pop} = 2 + |2(S - 1) - C| - |2S - C| = 1 + |2S - 2 - C| - |2S - C| = 2 - 2 = 0$

Copy

За Copy я обозначу операцию увеличения размера массива и копирования всего содержимого. Это происходит только если $S = C$.
$a_{copy} = 1 + S + |2S - 2C| - |2S - C| = 1 + S - C = 1$

Shrink

За Shrink я обозначу операцию сужения массива, она происходит только при $S = \frac{C}{4}$(Возьмем здесь реальное время работы $2S$ вместо настоящего S, чтобы не получалось отрицательное значение)
$a_{shrink} = 2S + |2S - C/2| - |2S - C| = 2S + 0 - \frac{C}{2} = 0$

Задача №3

Заведём стек s на расширяющемся массиве, в котором будем хранить id и val - индекс и значение элемента, все это будет в том порядке, в котором нам приходят запросы $set(i, x)$, и массив pos в который будем записывать индексы элементов в стеке. То есть если пришел запрос $get(5, 1)$ закинем в стек элемент с $id = 5$ и $val = 1$ и запишем $pos[5] = s.size() - 1$.
Для того чтобы проверить, есть ли элемент в стэке, посмотрим на $s[pos[i]].id$. Если это значение равно i, то этот элемент уже был добавлен в стек.
Для того чтобы выполнить $get(i)$ надо проверить, был ли i добавлен в стек, если да, то возвращаем $s[pos[i]].val$. Иначе этот элемент ещё не был проинициализирован, возвращаем ноль.
Для того чтобы сделать $set(i, x)$, надо просто сделать $s[pos[i]].val =$ x, если i уже есть в стеке, и добавить его в стек в противном случае.

Задача №5

Заведём два массива, $b[i]$ и $c[i]$. Будем хранить в $b[i]$ какой-то элемент из входного массива $a[i]$, а в $c[i]$ текущее количество этих элементов, которые мы успели набрать.
Воспользуемся следующим алгоритмом:

```
b = []
c = []
for j = 1 .. k - 1:
    c[j] = 0
for i = 1 .. n:
    f = 0
    for j = 1 .. k - 1:
        if b[j] = a[i] || c[j] = 0:
            c[j]++
            b[j] = a[j]
            f = 1
    if f = 1:
        break
    for j = 1 .. k - 1:
        c[j]--
```

```
//На этом моменте программа сформирует несколько групп размером k.
//Теперь надо найти среди оставшихся элементов тот,
//который встречается нужное кол-во раз.
    ans = []
    for j = 1 .. k - 1:
        if c[j] = 0:
            continue
        x = 0
        for i = 1 .. n:
            if a[i] = b[j]:
                x++
        if x > n / k:
            ans.push(b[j])
    print(ans)
```

Наш алгоритм будет собирать элементы в группы по k различных элементов, всего их может быть $\frac{n}{k}$. Несколько элементов из тех, что встречаются большее чем $\frac{n}{k}$ кол-во раз останутся. Поэтому этот алгоритм всегда найдет ответ.