Homework

Panov Ivan, M3139

11-09-2019

Задача 1

a) Максимальное худшее время работы *increment* равняется $O(k)$, так как если все биты равняются 1, то функция проверит их все, что работает за $O(k)$.
b) Для подсчета среднего времени работы воспользуемся методом усреднения. Если мы применили n операций инкримента, то количество изменений значений начиная с младшего бита будут выглядеть так: $n, \frac{n}{2}, \frac{n}{4}, \ldots$. Тогда количество всех операций: $\sum_{i=1}^{\log(n)} \frac{n}{2^i} = O(n)$, следовательно одна операция выполняется за $O(1)$.
c) Функция *decrement* работает за $O(k)$ в худшем случае, так как если все 0 кроме старшего бита, то мы пройдемся по всем битам. Тогда быстрее чем за $\Omega(nk)$ алгоритм не будет работать, так как *increment* и *decrement* работают в худшем случае за $O(k)$, но и дольше чем за $O(k)$ они могут не работать, следавательно алгоритм работает за $\Theta(nk)$ в худшем случае.
d) Заведем переменную *pref*, которая будет хранить количество битов на префиксе которое надо обнулить. Изменим функции:

```
setZero():
    pref = k;

get(i):
    if (i >= k - pref):
        return 0;
    else:
        return a[i];

increment():
int i = 0
    while i < k and a[i] = 1 and i < k - pref:
        a[i] = 0
        i++
    if i < k:
        a[i] = 1;
        if (i >= k - pref):
        pref--
```

Заметим, что *setZero* и *get* работают за $O(1)$, а *increment* не увеличила количество операций, тогда амортизированное время работы равно $O(1)$.

Задача 2

Введем функцию потенциала от двух переменных sz - количество элементов в структуре и cap - размер стека: $\Phi(sz, cap) = |2sz - cap|$. Рассмотрим операции со стеком: $push$, pop, $copy$.
$T(push) = 1 + \Delta\Phi = 1 + (|2sz + 2 - cap| - |2sz - cap|) = O(1)$.
$T(pop) = 1 + \Delta\Phi = 1 + (|2sz - 2 - cap| - |2sz - cap|) = O(1)$.
Если $sz = cap$, то $T(copy) = cap + \Delta\Phi = cap + (|2cap - 2cap| - |2cap - cap|) = O(1)$.
Если $4sz = cap$, то $T(copy) = 2sz + \Delta\Phi = 2sz + (|2sz - 2sz| - |2sz - 4sz|) = O(1)$.

Задача 3

Заведем стек st на массиве из n элементов, в котором будем хранить пары элементов: указатель на элемент массива $a[i]$ и значение val(значение, которое должно находиться по i-му индексу). В массиве a будем хранить для каждого элемента указатель на себя в стеке.
При добавлении элемента будем делать следующие действия: если элемент не инициализирован, то добавим новый элемент в стек, запишем туда новое значение $a[i]$ и указетель на $a[i]$, а в $a[i]$ запишем указатель на последний элемент st. Если элемент инициализирован, то просто пройдем по указателю в $a[i]$ и изменим значение переменной в st.
Теперь нужно научиться различать два случая, когда элемент инициализирован, а когда нет. Если $a[i] < st$ или $st + sz \leq a[i]$ (сравниваем указатели, sz - количество элементов в стеке), то элемент не инициализирован. Если уловие не выполняется, рассмотрим элемент куда показывает указатель(обозначим этот элемент за p), если указатель p указавает обратно на $a[i]$, то он инициализирован, иначе - нет.
Для того чтобы взять элемент проведем следующие действия: проверим инициализирован ли элемент(проверка на инициализированность приведена выше), который запрашивается, если нет, то выведем 0, иначе пройдем по указателю и выведем требуемое значение(val).
Каждая операция работает за $O(1)$ амортизированно(доказательство приведено в задаче 2).

Задача 4

Разделим пямять на блоки нужных нам размеров. Будем из каждого блока хранить указатель на другой блок так, что все блоки будут связаны и существует такой блок, что если начать с этого блока переходить по указателям мы посетим все блоки. Также будем хранить $head$ - указатель на первый пустой блок(изначально на блок из которого можно попасть во все).
Расмотрим операцию выделения памяти. Когда требуется выделить память просто вернем место куда указывает $head$ и после этого присвоим $head$ следующий доступный блок(блок следующий за блоком, на которой указывает $head$).
Рассмотрим операцию удаления. Направим указатель из удаляемого блока на блок, на который указывает $head$, а $head$ пусть теперь указывает на блок который удалили.
И операция выделения, и операция освобождения памяти работает за $O(1)$.

Задача 5

Давайте реализуем такой алгоритм. Будем разбивать массив на множества из k различных элементов, тогда в один момент останутья элементы которые нельзя разбить на такие множества, среди неразбитых элементов гарантированно останется ответ, так как максимальное количество множеств может быть $\frac{n}{k}$, а количество искомого элемента больше чем $\frac{n}{k}$.
Заведем массив пар $a[\ ]$ на $k-1$ элемент. В этом массиве будем хранить $\{val, cnt\}$ (val - какое-то число, cnt - количество этих чисел). Будем заполнять массив по следующему алгоритму: пройдемся по всем элементам изначального массива, на очередной итерации проверим есть ли данный элемент в массиве $a[\ ]$, если есть, то прибавим к счетчику 1, иначе проверим все ли $k-1$ элементов $a[\ ]$ имеются в ненулевом количестве, если есть $k-1$ ненулевых элементов, то объеденим их в одно множество и вычтем из всех счетчиков 1, иначе заменим любой нулевой элемент на $\{b[i], 1\}$ ($b[\ ]$ - изначальный массив).
После прохода по массиву $b[\ ]$ в массиве пар останутся элементы, которые не лежат ни в одном множестве, таких элементов не больше $k-1$ различных, в противном случае их можно было бы объеденить в множество. Тогда можно пройтись по массиву и для каждого элемента из оставшихся посчитать количество раз, которое он врстречается в изначальном массиве. Таким образом можно найти ответ за времени $O(nk)$ и за $O(k)$ дополнительной памяти.

Задача 6

Введем функцию потенциала от n - количество элементов в структуре: $\Phi(n) = \sum_{i=0}^{\log(n)} 2^i(\log(n) - i)$.
Посчитаем время работы *add*:
Пусть k - первая пустая строчка.
$T(add) = 2^k + \Delta\Phi = 2^k + (2^k(\log(n) - k - 2) + \log(n) - 2^k(\log(n) - k - 1)) = \log(n) = O(\log(n))$.
Посчитаем время работы *contains*:
$T(contains) = \log^2(n) + \Delta\Phi = \log^2(n) + 0 = O(\log^2(n))$.