Трудоемкость в худшем случае

Вспомнить про $O(n)$, $\Theta(n)$, $\Omega(n)$ + сделать небольшой тест (?).

Порекомендовать почитать Кормена (глава 3).

Θ :

$$\Theta(g(n)) = \Big\{ f(n) \quad if\, \exists c_1,\, c_2,\, n_0 : 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)\, \forall n \ge n_0$$

Поскольку $\Theta(g(n))$ представляет собой множество функций, то можно записать $f(n) \in \Theta(g(n))$ или $f(n) = \Theta(g(n))$.

Говорят, что функция $g(n)$ является асимптотически точной оценкой функции $f(n)$.

O :

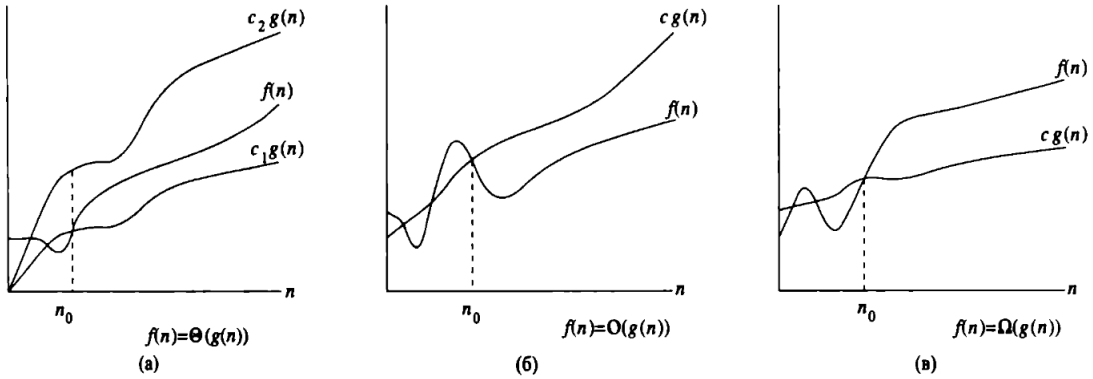
$$O(g(n)) = \Big\{ f(n) \quad if\, \exists c,\, n_0 : 0 \le f(n) \le c \cdot g(n)\, \forall n \ge n_0$$

O-обозначение применяется, когда надо указать верхнюю границу функции с точностью до постоянного множителя.

Ω :

$$\Omega(g(n)) = \Big\{ f(n) \quad if\, \exists c,\, n_0 : 0 \le c \cdot g(n) \le f(n)\, \forall n \ge n_0$$

Ω - это асимптотическая нижняя граница.



Амортизационный анализ

Вопрос: зачем вообще он нужен и что это такое?

Амортизационный анализ (amortized analysis) – метод анализа алгоритмов, позволяющий осуществлять оценку времени выполнения последовательности из n операций над некоторой структурой данных. При выполнении амортизационного анализа гарантируется средняя производительность в наихудшем случае.

Некоторые операции структуры данных могут иметь высокую вычислительную сложность, другие низкую. Например, некоторая операция может подготавливать структуру данных для быстрого выполнения других операций. Такие «тяжелые» операции выполняются редко и могут оказывать незначительное влияние на суммарное время выполнения последовательности из n операций.

Введен в практику Робертом Тарьяном (Robert Tarjan) в 1985 году.

Вспомнить про вектор (задача, которую разбирали на паре) + предложить придумать способ реализовать вектор с оценкой $O(1)$ в худшем случае на добавление элемента в конец (если выделение памяти происходит за константу).

Метод усреднения (метод группового анализа или aggregate analysis)

Метод усреднения - это метод амортизационного анализа, позволяющий оценивать верхнюю границу времени $T(n)$ выполнения последовательности из n операций в худшем случае.

В методе усреднения амортизационная стоимость операций определяется следующим образом: суммарная стоимость всех операций алгоритма делится на их количество.

$$a = \frac{\sum_{i=1}^{n} t_i}{n}$$

где t_i- это время выполнения i-ой операции.

Амортизированная стоимость операции – это оценка сверху среднего времени выполнения операции в худшем случае.

Задачи

1. стек с операцией multipop
2. двоичный счётчик
3. стек с multipop и multipush
4. счетчик с decrement
5. упражнение

Стек с multipop.

```
push(s, x) // добавляем объект x в стек s
pop(s) // "снимает" вершину стека (ошибка для пустого стека)
multipop(s, k) // "снимает" k вершин стека
```

Спросить про стоимость операций push и pop. Написать псевдокод операции.

```
def multipop(s, k):
    while empty(s) == False and k > 0:
        pop(s)
        --k
```

Количество итераций в цикле будет зависеть от реального количества объектов в стеке и $k \Rightarrow$ мы их сделаем $\min(size(s), k)$. На каждой итерации выполняется однократный вызов операции $pop \Rightarrow$ полная стоимость выполнения операции $multipop$ будет равна $\min(size(s), k)$.

Проанализируем последовательность операций $push$, pop, $multipop$, действующие на изначально пустой стек. Пусть $n = size(s)$. Стоимость операции $mulpipop$ в наихудшем случае равна $O(n)$,так как в стеке не более n объектов. Покажем, что произвольная последовательность операций $push$, pop, $multipop$ не превышает $O(n)$.

Каждый помещенный в стек объект может быть извлечен оттуда не более одного раза $\Rightarrow$ число вызовов операции pop (включая их вызовы в процедуре $multipop$) для непустого стека не превышает количество произведенных операций $push$, которое, в свою очередь, не больше n. При любом n для выполнения произвольной последовательности из n операций $push$, pop, $multipop$ требуется суммарное время $O(n) \Rightarrow$ средняя стоимость будет $\frac{O(n)}{n} = O(1)$.

Распишем приведённые рассуждения более формально. Пусть m — количество операций, n — количество элементов, задействованных в этих операциях. Очевидно, $n \leqslant m$.

Тогда:

$$a = \frac{\sum_{i=1}^{m} t_i}{m} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}}{m} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij}}{m}$$

где t_{ij} — стоимость i-ой операции над j-ым элементом. Величина $\sum_{i=1}^{m} t_{ij} \leq 2$, так как над элементом можно совершить только две операции, стоимость которых равна 1 (добавили и удалили).

Тогда: $a \leq \frac{2n}{m} \leq [n \leq m] \leq 2 => a = O(1)$.

В групповом анализе амортизированная стоимость каждой операции принимается равной ее средней стоимости, поэтому в этом случае все три стековые операции будут характеризоваться одинаковой амортизированной стоимостью $O(1)$.

Двоичный счётчик. В качестве счетчика будем использовать битовый массив $A[0 \dots k-1]$, где $A.length = k$. Младший бит хранящегося в счетчике бинарного числа x находится в элементе $A[0]$, старший - в элементе $A[k-1]$. $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$.

Изначально $x = 0 \Rightarrow A[i] = 0$, $i \in [0 \dots k-1]$.

x	7	6	5	4	3	2	1	0	стоимость
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Предложить реализовать процедуру увеличения счетчика на 1.

Чтобы увеличить показания счетчика на 1 используется следующая процедура :

```
def Increment(A):
    i = 0
    while i < A.length and A[i] == 1:
        A[i] = 0
        i += 1
    if i < A.length
        A[i] = 1
```

Заметим, что значение всех бит изменяются не при каждом вызове *Increment*.

Посмотрим, как часто меняется $A[0]$, $A[1]$ и т.д. Увидим, что в общем случае для $i = 0, \dots, k-1$ бит $A[i]$ изменяется $\lfloor \frac{n}{2^i} \rfloor$ раз в последовательности из n операций *Increment* для изначально обнуленного счетчика. Таким образом, общее количество изменений битов при выполнении последовательных операций *Increment* равно:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{k-1} \frac{1}{2^i} < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

Поэтому время выполнения последовательности из n операций *Increment* над изначально обнуленным счетчиком в наихудщем случае равно $O(n)$. Средняя стоимость операций равна $a = \frac{O(n)}{n} = O(1)$.

Стек с multipop и multipush. Остается ли справедливой амортизированная оценка стоимости стековых операций, равная $O(1)$, если включить в множество стековых операций операцию *multipush*, помещающая в стек k элементов?

Счетчик с decrement (можно в anytask?). Покажите, что если бы в примере с $k-$битовым счетчиком была включена операция *decrement*,то стоимость операций была бы равна $\Theta(kn)$.

Упражнение (можно в anytask?). Предположим, что над структурой данных выполняется n операций. Стоимость $i-$ой по порядку операции равна i, если i - это точная степень двойки, и 1 - иначе. Определить с помощью группового анализа, метода предоплаты и метода потенциалов амортизированную стоимость операции.

Метод предоплаты (accounting method)

Разные операции будет оцениваться по-разному, в зависимости от их фактической стоимости.

Величина, которая "начисляется" на операцию называется амортизированной стоимостью (amortized cost). Если амортизированная стоимость операции превышает ее фактическую стоимость, то соответствующая разность присваивается определенным объектам структуры данных как кредит. Кредит можно использовать впоследствии для компенсирующих выплат на операции, амортизированная стоимость которых меньше их фактической стоимости. Т.о., можно полагать, что амортизированная стоимость операции состоит из ее фактической стоимости и кредита, который либо накапливается, либо расходуется.

Пусть c_i- это фактическая стоимость $i-$ой операции, а $\widehat{c_i}-$ это ее же амортизированная стоимость. Тогда должно выполняться следующее неравенство:

$$\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i \tag{1}$$

Общий кредит, хранящийся в структуре данных вычисляется следующим образом: $credit = \sum_{i=1}^{n} \hat{c_i} - \sum_{i=1}^{n} c_i$. Так как у нас есть неравенство **??**, то получаем, что $credit \geq 0$. Если бы полный кредит в каком-то случае мог стать отрицательным, то полная амортизированная стоимость было бы в этот момент ниже соответствующей фактической стоимости.

Задачи

1. стек с операцией multipop
2. двоичный счетчик
3. рекурсивный обход дерева
4. стек с копированием
5. счетчик с reset

Stack (multipop). Фактическая стоимость операция у нас следующая:

```
push      1
pop       1
multipop  min(n, k)
```

Приведем ниже амортизированные стоимости:

```
push      2
pop       0
multipop  0
```

Заметим, что амортизированная стоимость операции *multipop* у нас равна константе, в то время как ее фактическая стоимость - это переменная величина.

В этой схеме все три амортизированные стоимости равны $O(1)$, хотя в общем случае асимптотическое поведение амортизированных стоимостей рассматриваемых операций может быть разным.

Осталось доказать, что любую последовательность операций можно "оплатить" путем начисления амортизированных стоимостей. (Предложить порассуждать самим).

Двоичный счётчик. Начислим на операцию, при которой биту присваивается значение, амортизированную стоимость, равную двум. Когда бит устанавливается, единица расходуется на саму операцию + получаем еще единицу в кредит для последующего использования (для обнуления). В любой момент времени с каждой единицей в счетчике связана единица в кредит $\Rightarrow$ на обнуление бита нет необходимости начислять что-то.

Определим амортизационную стоимость операции *Increment*. Стоимость обнуления битов выплачивается из кредита, который мы получили от единичных битов. В процедуре *Increment* в 1 устанавливается не более одного бита, поэтому амортизационная стоимость операции *Increment* не превышает 2. Кредит не может быть отрицательным $\Rightarrow$ условие **??** выполняется. Амортизационная стоимость n операций *Increment* равна $O(n)$.

Рекурсивный обход дерева.

Стек с копированием. Предположим, что над стеком выполняется последовательность операций. Размер стека при этом никогда не превышает k. После каждых k операций производится резервное копирование стека. Присвоив различным стековым операциям соответствующие стоимости, покажите, что стоимость n стековых операций, включая копирование стека, равна $O(n)$.

Счетчик с reset. Предположим, что нам нужно иметь возможность не только увеличивать показания счетчика, но и сбрасывать его. Считая, что время проверки или модификации одного бита составляют $\Theta(1)$, покажите, как осуществить реализацию счетчика в виде массива битов, чтобы выполнение произвольной последовательности из n операций *Increment* и *Reset* над изначально обнуленным счетчиком потребовало бы время $O(n)$.

Метод потенциалов

Если в методе предоплаты мы использовали кредиты, то в методе потенциалов вводится понятие "потенциальной энергии" или "потенциала", который можно высвободить для оплаты следующих операций. Этот потенциал связан со структурой в целом, а не с ее отдельными объектами.

Мы начинаем с исходной структуры данных D_0, над которой выполняется n операций. Для всех $i = 1, 2, \ldots, n$ обозначим через c_i фактическую стоимость i-ой операции, а через D_i - структуру данных, которая получается в результате применения $i-$ой операции к структуре D_{i-1}. Функция потенциала Φ отображает каждую структуру данных D_i на действительно число $\Phi(D_i)$, которое является потенциалом, связанным с структурой D_i. Амортизированная стоимость $\hat{c_i}$ i-ой операции определяется соотношением:

$$\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) \tag{2}$$

Полная амортизированная стоимость после выполнения n операций будет равна:

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \tag{3}$$

$$\Phi(D_n) \geq \Phi(D_0) \tag{4}$$

Если функцию потенциала определить таким образом, чтобы выполнялось неравенство **??**, то полная амортизированная стоимость $\sum_{i=1}^{n} \hat{c_i}$ является верхней границей полной фактической стоимости $\sum_{i=1}^{n} c_i$.

Вопрос: а почему нам важно, чтобы выполнялось **??**?

Так как на практике не всегда известно, сколько операций будет выполнено, то иногда накладывают дополнительные ограничения: $\Phi(D_i) \geq \Phi(D_0)$. Часто удобно определить $\Phi(D_0) = 0$, а зачем показать, что значение потенциала для всех i неотрицательное.

Задачи

1. стек с операцией multipop
2. двоичный счетчик

Stack (multipop). $\Phi(D_0) = 0$, $\Phi(D_i) = n$

Так как количество объектов в стеке не может быть отрицательным, стеку D_i, полученному после выполнения $i-$ой операции, соответствует неотрицательный потенциал.

Вычислим амортизированные стоимости различных стековых операций.

Для *Push*:

$$\Phi(D_i) - \Phi(D_{i-1}) = (n+1) - n = 1$$

Вспомним формулу **??**. Тогда амортизированная оценка для операции *push* будет следующей:

$$\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

Предположим, что $i-$ая операция над стеком - это $multipop(s, k)$. $k' = \min(k, n)$. $c_i = k'$, $\Phi(D_i) - \Phi(D_{i-1}) = -k'$. $\hat{c_i} = k' - k' = 0$. Аналогично для *pop*.

Получили, что амортизированная стоимость каждой операции равна $O(1)$ + амортизированная стоимость n операций равна $O(n)$ + выполняются все ограничения на значения потенциалов.

Вопрос: чему равна полная стоимость выполнения n стековых операций *push*, *pop*, *multipop*, если предположить, что в начале стек содержит k объектов, а в конце m.

Двоичный счетчик. k - количество бит в счетчике.

Потенциал счетчика после выполнения $i-$ой операции $Increment$ определим как количество b_i,содержащихся в счетчике единиц после этой операции.

Пусть $i-$ая операция $Increment$ обнулит t_i бит. В таком случае фактическая стоимость этой операции не превышает $c_i \leq t_i + 1$.

Если $b_i = 0$, то входе выполнения $i-$ой операции все k бит, так что $b_{i-1} = t_i = k$.

Если $b_i > 0$, то выполняется соотношение $b_i \leq b_{i-1} - t_i + 1$.

$$\Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$$

$$\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) \leq t_i + 1 + 1 - t_i = 2$$

Если вначале показание счетчика равно 0, то $\Phi(D_0) = 0$. $\Phi(D_i) \geq 0$.

Метод потенциалов предоставляет способ анализа счетчика даже для того случая, когда начальное значение счетчика не равно нулю.

Пусть изначально счетчик содержит b_0 единиц, а после выполнения n операций $Increment$ - b_n. $0 \leq b_0, b_n \leq k$.

Тогда уравнение **??** можно переписать следующим образом:

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c_i} - \Phi(D_n) + \Phi(D_0)$$

Для всех $1 \leq i \leq n$ выполняется неравенство $\hat{c_i} \leq 2$ (получили выше). $\Phi(D_0) = b_0$, $\Phi(D_n) = b_n$. Получим:

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} 2 - b_n + b_0 = 2n + b_n - b_0$$

Так как $0 \leq b_0, b_n \leq k$, то при $k = O(n)$ полная фактическая стоимость равна $O(n)$. Другими словами, если выполняется не менее $n = \Omega(k)$ операций $Increment$, то полная фактическая стоимость независимо от начального показания счетчика равна $O(n)$.

Стек, очередь, дек с поиском минимального элемента

Необходимо добавить операцию поиска минимального элемента в классический интерфейс.

Стек

Хотим добавить операции $min(s)$, которая будет возвращать минимальный элемент для стека s за время $O(1)$ + сохранить оценки на *push* и *pop*.

Решение: хранить пары $(current\_value, current\_min)$.

Очередь

Просто очередь на стеках на минимум.

Решение: заведем два стека на минимум s_1 и s_2. Добавлять новые элементы будет всегда в стек s_1, а извлекать элементы - только из стека s_2. При этом, если при попытке извлечения элемента из стека s_2 он оказался пустым, просто перенесём все элементы из стека s_1 в стек s_2 (при этом элементы в стеке s_2 получатся уже в обратном порядке, что нам и нужно для извлечения элементов; стек s_1 же станет пустым). Наконец, нахождение минимума в очереди будет фактически заключаться в нахождении минимума из минимума в стеке s_1 и минимума в стеке s_2.

Тем самым, мы выполняем все операции по-прежнему за $O(1)$ (по той простой причине, что каждый элемент в худшем случае 1 раз добавляется в стек s_1, 1 раз переносится в стек s_2 и 1 раз извлекается из стека s_2).

Дек

Сказать подумать. Спросить на второй практике

```
empty     // проверка на наличие элементов
pushBack  // операция вставки нового элемента в конец
popBack  // операция удаления конечного элемента
pushFront // операция вставки нового элемента в начало
popFront  // операция удаления начального элемента
```

Добавить один много раз

Имеется двоичный счетчик из k бит в начальном состоянии $b_1 b_2 \dots b_k$. Выполним операцию инкремента n раз. Определите наибольшее число изменившихся бит за одну операции инкремента и среднее число изменившихся битов.

Заметим, что если $i+1$ бит равен 1, то каждое нечетное его изменение повлечет изменение i-ого бита, если 0 - то каждое четное.