УДК 004.9

М.В. Виноградова, Э.Г. Игушев

ПРИНЦИПЫ ОРГАНИЗАЦИИ СТРУКТУРЫ ДАННЫХ С ПРОИЗВОЛЬНЫМ ДОСТУПОМ И БЫСТРОЙ ОПЕРАЦИЕЙ ВСТАВКИ ИЛИ УДАЛЕНИЯ

Сформулирована задача разработки структуры данных с произвольным доступом и с меньшим временем удаления, чем у обычного массива, а также предложены принципы ее решения.

E-mail: vinogradova.m@gmail.com

Ключевые слова: *структура данных, произвольный доступ, вставка, удаление.*

Введение. Существуют две базовые структуры организации данных: массив и список. Массив — структура с произвольным доступом к данным, но с трудоемкой линейной операцией вставки или удаления. Список — структура с последовательным доступом к данным, имеющая быструю константную операцию вставки или удаления.

В настоящее время различные задачи в информационных технологиях предъявляют все большие требования к структурам данных. Например, при поиске по сети Интернет после формирования списка документов, его необходимо пропустить через фильтр для удаления некоторых записей. Во время обхода сети Интернет поисковым роботом вследствие неправильного конфигурирования системным администратором прав доступа какого-либо ресурса робот может записать в базу личные данные пользователей этого ресурса или другие данные, которые не должны быть в поисковой выдаче. Однако поисковая база обновляется в лучшем случае раз в сутки, а документы должны быть удалены из выдачи немедленно после выявления подобного факта. Для этого и разработан фильтр, удаляющий некоторые документы из выдачи на последнем этапе фильтрации. Подобное удаление происходит редко, однако оно не должно влиять на скорость, так как формирование осуществляется в режиме реального времени [2]. Кроме того, полученная структура данных с множеством документов требует произвольного доступа к ее элементам. Поэтому была сформулирована задача разработки структуры данных с произвольным доступом и с меньшим временем удаления, чем у обычного массива.

Аналоги и прототипы. Структура ***FastArray*** (рис. 1) представляет собой бинарное дерево, в одних узлах которого записаны элементы, а в других — количество листьев в левом поддереве [4].

Доступ к элементу по индексу (рис. 2, *а*) осуществляется путем прохождения от корня к листу и сравнения требуемого индекса со значением в узле:

— если найден требуемый элемент, то алгоритм закончен;

— если требуемый индекс меньше, то алгоритм продолжается рекурсивно в левом поддереве;

— если индекс уменьшается на значение в узле, то алгоритм продолжается рекурсивно в правом поддереве.

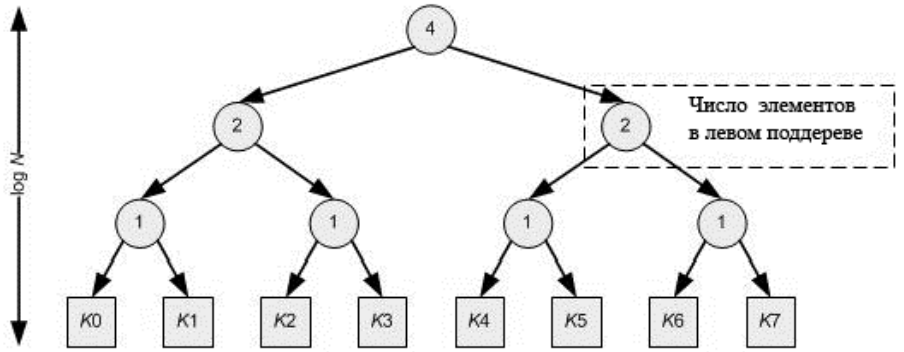


Рис. 1. Структура FastArray

Сложность операции доступа к элементу по индексу составляет $O(\log_2 N)$.

Удаление элемента по индексу (рис. 2, *б*) происходит путем прохождения от корня к листу и сравнения требуемого индекса со значением в узле:

— если найден элемент, то он удаляется и все узлы на пути от корня с одним потомком также удаляются;

— если требуемый индекс меньше, то значение в узле декрементируется и алгоритм продолжается рекурсивно в левом поддереве;

— если индекс уменьшается на значение в узле, то алгоритм продолжается рекурсивно в правом поддереве.

Сложность операции удаления элемента по индексу — $O(\log_2 N)$.

Вставка элемента по индексу (рис. 2, *в*) также осуществляется путем прохождения от корня к листу и сравнения требуемого индекса со значением в узле:

— если найден элемент, на место которого необходимо вставить новый элемент, то на его месте образуется поддерево с узлом и единичным значением, новым элементом в качестве левого потомка и текущим элементом в качестве правого потомка;

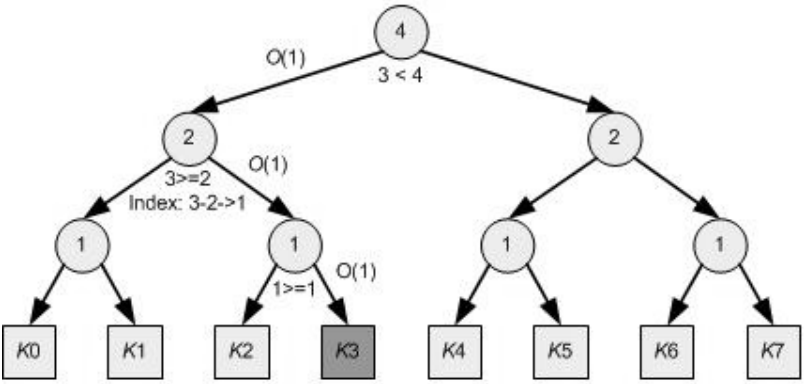
— если индекс меньше, то значение в узле инкрементируется и алгоритм продолжается рекурсивно в левом поддереве;

— если индекс уменьшается на значение в узле, то алгоритм продолжается рекурсивно в правом поддереве.

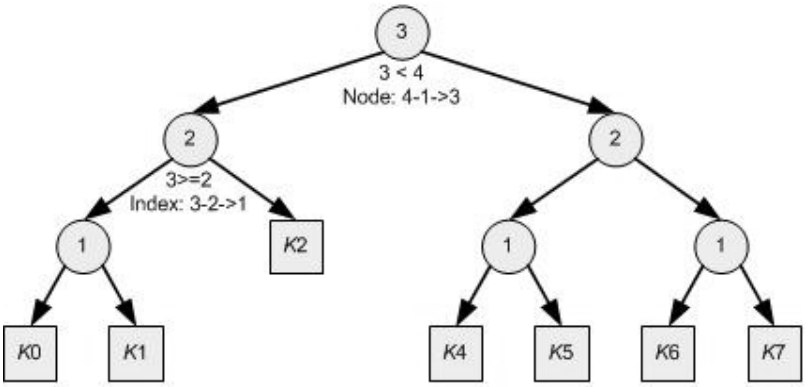
Обратим внимание, что индексы всех элементов, расположенных «справа» от нового элемента, сдвигаются «автоматически».

Сложность операции вставки по индексу — $O(\log_2 N)$.

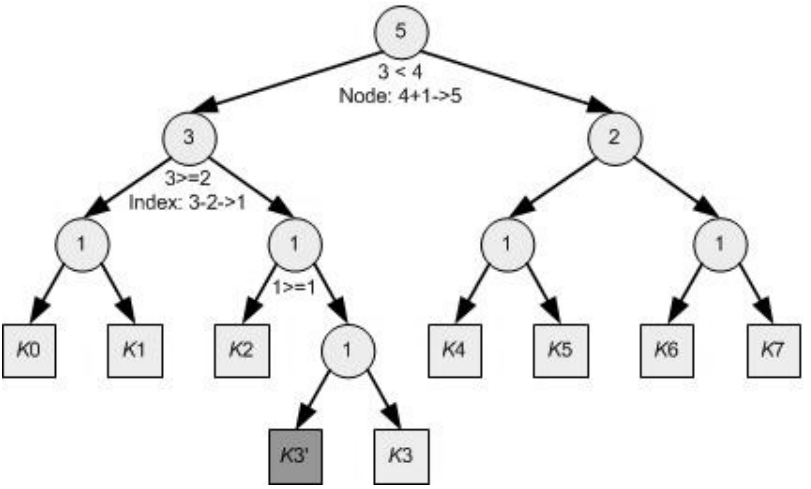
Данная структура обладает отличными свойствами.



а



б



в

Рис. 2. Примеры доступа к элементу (*а*), удаления элемента (*б*) и вставки элемента (*в*) с индексом 3

Поскольку в рассматриваемой выше задаче приоритетом является произвольный доступ, то применение структуры FastArray не достаточно эффективно.

Структура ***IgushArray*** — структура с произвольным доступом, которая как массив имеет быструю константную операцию доступа, но сложность операции вставки или удаления составляет всего лишь $O(N^{1/2})$. Структура может рассматриваться как «быстрый массив» или «массив с быстрой операцией вставки». Сложность выполнения операции определяется как зависимость ее времени выполнения от количества элементов массива. Время выполнения операций для различных структур данных приведено в табл. 1 [1].

Таблица 1

Время выполнения операций для различных структур данных

Операция	Массив	IgushArray	FastArray	Список
Доступ	$O(1)$	$O(1)$	$O(\log N)$	$O(N)$
Вставка	$O(N)$	$\boldsymbol{O(N^{1/2})}$	$O(\log N)$	$O(1)$
Удаление	$O(N)$	$\boldsymbol{O(N^{1/2})}$	$O(\log N)$	$O(1)$
Вставка:	$O(1)$	$O(1)$	$O(\log N)$	$O(1)$
в конец				
в начало	$O(N)$	$\boldsymbol{O(N^{1/2})}$	$O(\log N)$	$O(1)$

Примечание. Выделенные показатели лучше аналогичных показателей стандартного массива.

IgushArray — массив указателей размером приблизительно $N^{1/2}$. Каждый элемент массива указывает на очередь с двумя концами размером около $N^{1/2}$. Все очереди имеют одинаковый размер за исключением последней (может иметь размер меньше $N^{1/2}$).

Логическое представление

$$I = \left\{E_0, E_1, ..., E_{N-1}\right\},$$

где I — структура IgushArray; E_k —элемент логической структуры; N — количество элементов в структуре.

Физическое представление

$$I = V = \left\{D_0, D_1, ..., D_{S_V-1}\right\},$$

где V — массив указателей на очереди; D_i – очередь с двумя концами, $D_i = \{E_{i,0}, E_{i,1}, ..., E_{i,S_{D_i}-1}\}$, $E_{i,j}$ — элемент физической структуры; $E_{i,j} = E_{iS_{D_i}+1}$ — связь логического и физического представления; S_V — размер массива указателей; S_{D_i} — размер очереди.

Параметры физического представления следующие:

$S_{D_0} = S_{D_1} = S_{D_{S_V-2}}$ — все очереди имеют одинаковый размер;

$S_{D_{S_V-1}} \le S_D$ — последняя очередь может иметь меньший размер;

$$S_D = \lceil N^{1/2} \rceil;$$

$$S_V = \lceil N / S_D \rceil;$$

$$S_D \cong S_V \cong N^{1/2}.$$

Размер массива может изменяться по мере работы со структурой; размер очередей не меняется никогда. После работы со структурой можно выполнить полное перестроение для подгонки размера массива и очередей до размера приблизительно $N^{1/2}$. Рекомендуется знать приблизительный размер будущей структуры во время ее создания, чтобы изначально рассчитать оптимальные размеры очереди. Иначе размер очередей будет 1 и время вставки или удаления выродится в линейное.

Очередь с двумя концами может быть реализована с использованием массива (набора массивов), либо списка. В первом случае обеспечивается константный доступ, но линейные вставка или удаление, во втором — линейный доступ, но константные вставка или удаление. Очень важно, чтобы в структуре IgushArray очередь была реализована с помощью именно первого способа. Поскольку размер очередей известен во время создания структуры и никогда не изменяется, то очередь можно создать из **одного массива**, что улучшит и упростит реализацию. Таким образом, массив и очереди, применяемые в структуре IgushArray, являются структурами с произвольным доступом (рис. 3).

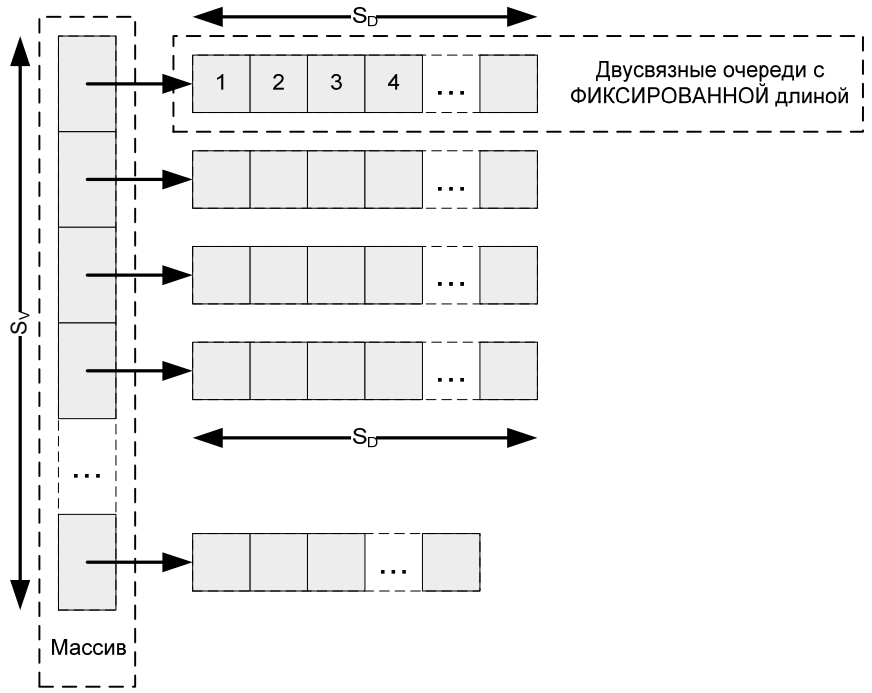


Рис. 3. Струтура IgushArray

Для доступа к произвольному элементу по его индексу k в структуре IgushArray необходимо:

1) рассчитать индекс в массиве с помощью простой арифметической операции $i = \lfloor k / deq\_size \rfloor$, сложность $O(1)$;

2) получить указатель на очередь D_i в массиве V, сложность $O(1)$;

3) вычислить индекс в очереди при помощи простой арифметической операции $j = k \% deq\_size$, O(1);

4) получить элемент $E_{i,j}$ в очереди D_i, сложность $O(1)$.

Псевдокод операции доступа: $i = \lfloor k / deq\_size \rfloor$; $D_i = V[i]$; $j = k \% deq\_size$; $E_{i,j} = D_i[j]$; $E_k = E_{i,j}$.

Произвольный доступ к элементу по индексу (рис. 4) состоит из двух простых арифметических операций и двух операций доступа к памяти, а значит операция в целом имеет сложность $O(1) + O(1) + O(1) + O(1) = O(1)$.

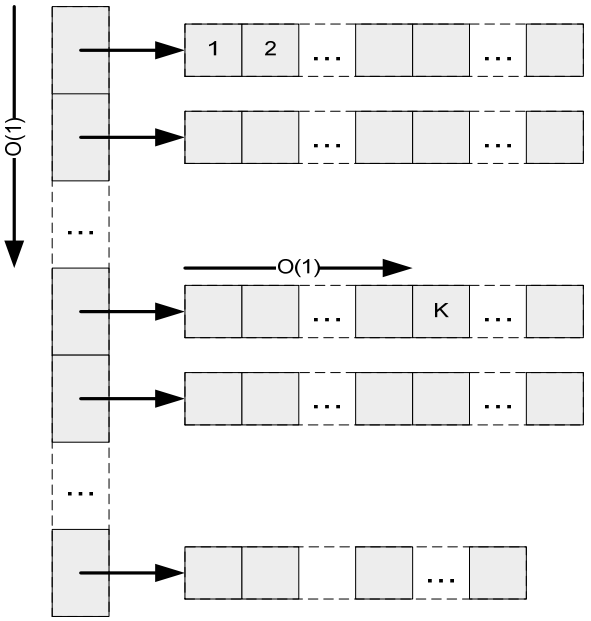


Рис. 4. Операция доступа в IgushArray

Для вставки элемента в позицию k в структуре IgushArray, необходимо:

1) получить доступ к элементу $E_{i,j}$, на место которого требуется вставка, сложность $O(1)$;

2) вставить новый элемент в очередь D_i в полученную позицию j, сложность $O(S_D)$;

3) «сдвинуть» элементы в этой и следующих очередях вниз, для чего следует:

3.1) вынуть элемент с конца текущей очереди D_i, сложность $O(1)$;

3.2) вставить элемент в начало следующей очереди D_{i+1}, сложность $O(1)$.

Каждый сдвиг имеет сложность $O(1)$. Необходимо «сдвинуть» вниз порядка S_V элементов. Таким образом, вся сложность «сдвига» составляет $2S_V O(1) = O(S_V)$.

Примечание. Операция может потребовать вставку нового указателя в конец массива указателей и создание новой очереди (если последняя очередь до операции полная, после операции переполнена). Операция может потребовать $O(S_V)$ времени и не испортить общую производительность.

Псевдокод операции вставки:

```
//доступ к позиции
i = ⌊k / deq_size⌋
D_i = V[i]
j = k%deq_size

//вставка в очереди
insert(D_i, j, val)

//сдвиг элементов
пока(i < S_V − 1)
    temp = pop_back(D_i)
    push_front(D_{i+1}, temp)
    ++i
end

//если последняя очередь переполнена
если(size(D_{S_V−1}) > S_D)
temp = pop_back(D_{S_V−1})
push_front(D_new, temp)
push_back(V, D_new)
end
```

Здесь $insert(D, pos, val)$ — вставка элемента *val* в очередь *D* в позицию *pos*; $pop\_back(D)$ — вынимание элемента с конца очереди *D*; $push\_front(D, val)$ – вставка элемента *val* в начало очереди *D*; $size(D)$ — текущий размер очереди *D*; $push\_back(V, D)$ — вставка в конец массива *V* указателя на очередь *D*.

Операция вставки одного элемента (рис. 5) состоит из доступа, вставки в очередь и «сдвига» вниз, операция в целом имеет сложность $O(1) + O(S_D) + O(S_V) \cong O(1) + O(N^{1/2}) + O(N^{1/2}) = O(N^{1/2})$.

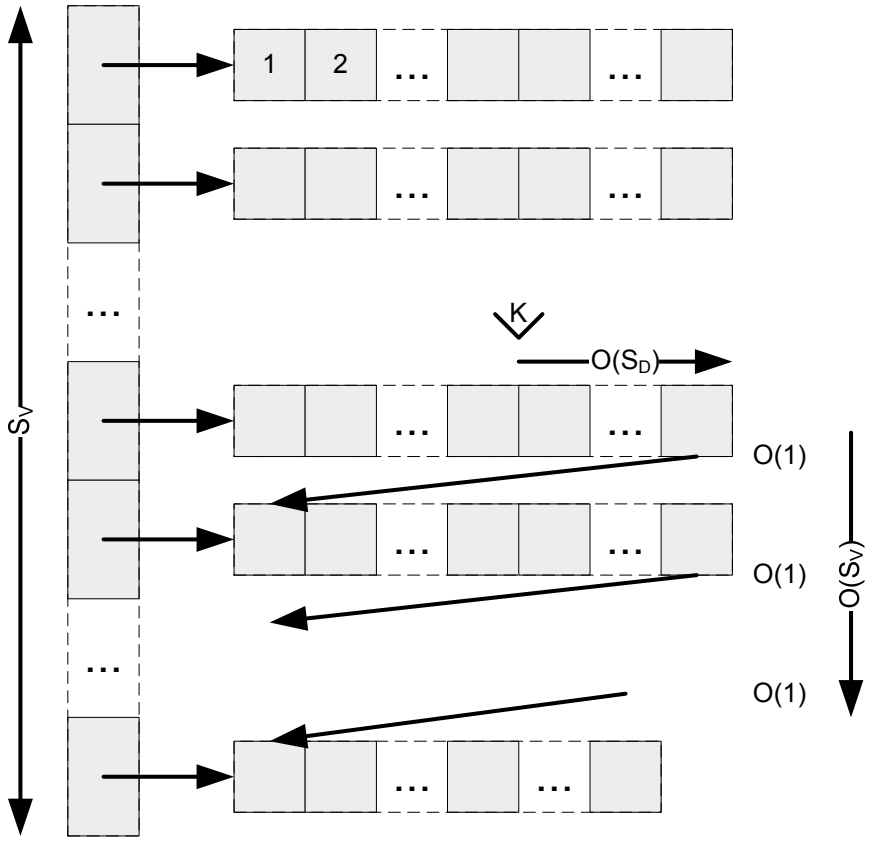


Рис. 5. Операция вставки в IgushArray

Для удаления одного элемента из позиции *k* в IgushArray необходимо:

1) получить доступ к удаляемому элементу $E_{i,j}$, сложность $O(1)$;

2) удалить элемент в очереди D_i в позиции *j*, сложность $O(S_D)$;

3) «сдвинуть» элементы в этой и следующих очередях вверх, для чего требуется:

3.1) вынуть элемент с начала следующей очереди D_{i+1}, сложность $O(1)$;

3.2) вставить в конец текущей очереди D_i, сложность $O(1)$.

Каждый сдвиг имеет сложность $O(1)$. Необходимо «сдвинуть» вверх порядка S_V элементов, тогда сложность всего «сдвига» составляет $2S_V O(1) = O(S_V)$.

Примечание. Операция может потребовать удаление последней очереди (в случае, если последняя очередь до операции имеет один элемент, после операции — пустая).

Псевдокод операции удаления:

//доступ к позиции

$i = \lfloor k / deq\_size \rfloor$

$D_i = V[i]$

$j = k\%deq\_size$
//удаление в очереди
$erase(D_i, j)$

//сдвиг элементов
пока$(i < S_V - 1)$
    $temp = pop\_front(D_{i+1})$
    $push\_back(D_i, temp)$
    $++i$
end

//если последняя пуста
если$(empty(D_{S_V-1}))$
$D_{last} = pop\_back(V)$
$destroy(D_{last})$
end

Здесь $erase(D, pos)$ — удаление элемента в очереди D в позиции pos; $pop\_front(D)$ — вынимание элемента с начала очереди D; $push\_back(D, val)$ — вставка элемента val в конец очереди D; $empty(D)$ — узнать, пуста ли очередь D; $pop\_back(V)$ — вынуть указатель на очередь с конца массива V; $destroy(D)$ — уничтожить очередь D.

Операция удаления одного элемента (рис. 6) состоит из доступа, удаления из очереди и «сдвига» вверх, сложность: $O(1) + O(S_D) + O(S_V) \cong O(1) + O(N^{1/2}) + O(N^{1/2}) = O(N^{1/2})$.

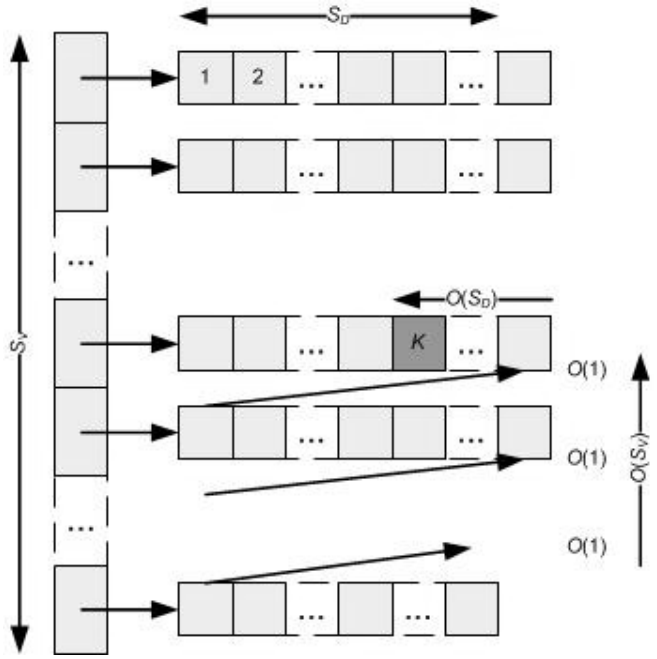


Рис. 6. Операция удаления в IgushArray

Отметим, что в массиве, используемом для реализации очереди с двумя концами, можно применить структуру IgushArray и т. д. Тогда сложность операции доступа будет $O(k)$, где k — степень; сложность операции вставки или удаления составит $O(N^{1/k})$. В простом случае, описанном в статье, $k = 2$.

Тесты производительности. Структура данных была реализована. Тесты были проведены с помощью компьютера со следующей конфигурацией: Inter Core 2 Duo 1,83 МГц, 2 Гбайт, Ubuntu, GCC версии 4.5.2. Каждая операция повторялась 1 000 раз. Сравнивалась стандартная реализация массива из математической библиотеки std::vector [3].

В табл. 2 приведены значения времени суммирования элементов структур при произвольном доступе к элементу по индексу.

Таблица 2

Значения времени суммирования элементов структур при произвольном доступе к элементу по индексу, мс

Количество элементов	IgushArray	std::vector	Результат
1 000	80	10	Ниже 8, 0
10 000	840	110	7,6
100 000	8 220	1 130	7,3
1 000 000	89 340	15 720	5,7
10 000 000	903 520	130 200	6,9

Из результатов ясно, что время суммирования возрастает линейно вместе с количеством элементов. Следовательно, обе структуры имеют константное время доступа к элементу.

В табл. 3 даны значения времени суммирования элементов структуры при доступе по итератору.

Таблица 3

Значения времени суммирования элементов структур при доступе по итератору, мс

Количество элементов	IgushArray	std::vector	Результат
1 000	170	40	Ниже 4,2
10 000	1 700	340	5,0
100 000	16 910	3 420	4,9
1 000 000	167 600	34 310	4,9
10 000 000	1 686 220	343 090	4,9

Согласно результатам, время суммирования увеличивается линейно вместе с количеством элементов.

Худшее время доступа в структуре IgushArray обусловлено тем, что она является более сложной структурой, чем массив.

В табл. 4 приведены значения времени выполнения вставки одного элемента в середину структуры.

Таблица 4

Значения времени выполнения вставки одного элемента в середину структуры, мс

Количество элементов	IgushArray	std::vector	Результат
1 000	10	10	Одинаково
10 000	10	30	Более 3
100 000	60	360	6
1 000 000	80	3 580	45
10 000 000	550	36 940	67

Из результатов видно, что время операции вставки std::vector возрастает линейно вместе с количеством элементов, а время операции вставки IgushArray — как $N^{1/2}$.

В табл. 5 даны значения времени удаления одного элемента из середины структуры.

Таблица 5

Значения времени удаления элемента из середины структуры, мс

Количество элементов	IgushArray	std::vector	Результат
1 000	10	10	Одинаково
10 000	40	10	Ниже 4
100 000	60	300	Более 5
1 000 000	200	3 270	16
10 000 000	590	31 310	53

Из результатов ясно, что время удаления std::vector увеличивается линейно вместе с количеством элементов, а время операции удаления IgushArray — как $N^{1/2}$.

Заключение. В статье описана структура с произвольным доступом, которая как массив имеет быструю константную операцию доступа, сложность операции вставки или удаления составляет всего $O(N^{1/2})$. Структура может рассматриваться как «быстрый массив» или «массив с быстрой операцией вставки». Структура была реализована и протестирована.

СПИСОК ЛИТЕРАТУРЫ

1. Cormen T.H., Leiserson C.E., Rivest R.L., Stein Cl. Introductin to Algorithms. Third Edition. Cambridge: The MIT Press, 2009.
2. Макконнелл Дж. Основы совеременных алгоритмов. М.: Техносфера, 2006.
3. Stroustrup B. The C++ Programming Language. Special Edition. Boston: Addison-Wesley, 2000.
4. http://caladan.nanosoft.ca/fastarray.php. FastArray.

Статья поступила в редакцию 4.07.2012