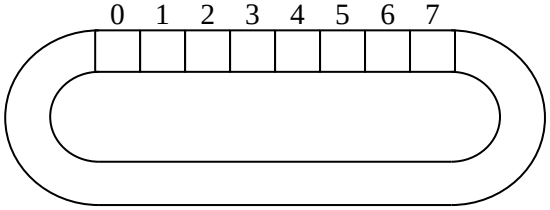
Тема 2. СТРУКТУРЫ ДАННЫХ

2.4. Очереди

Очередь представляет собой динамическую последовательность элементов с двумя точками доступа: начало (голова) и конец (хвост). Новый элемент добавляется всегда в конец очереди, исключается всегда элемент, расположенный в начале очереди. Все остальные элементы очереди недоступны. Таким образом, очередь работает по принципу «первым пришел – первым ушел» и часто называется структурой FIFO (First In – First Out).

Операцию включения элемента в очередь будем записывать в виде $Q \Leftarrow x$ (элемент x поместить в конец очереди Q), а операцию исключения из очереди – в виде $x \Leftarrow Q$ (исключить элемент из начала очереди Q и присвоить его значение переменной x).

Последовательная реализация очереди требует специальных приемов. Это связано с тем, что очередь растет на одном конце и убывает на другом, т. е. двигается в сторону правой границы массива и может ее перейти, хотя в левой части массива может быть достаточно места для размещения элементов очереди. Чтобы этого избежать, необходимо массив свернуть в кольцо. Для этого удобно использовать операцию **mod** (вычисление остатка от целочисленного деления). В этом случае для очереди Q выделяется массив из m компонентов $Q_0, Q_1, Q_2, \ldots, Q_{m-1}$ и в соответствии с операцией **mod** считается, что Q_0 следует за Q_{m-1}. Если использовать переменную f в качестве указателя позиции в массиве, расположенной непосредственно перед началом очереди, а переменную r в качестве указателя ее конца, то очередь будет состоять из элементов $Q_{f+1}, Q_{f+2}, \ldots, Q_r$. Пустая очередь будет соответствовать случаю $r = f$. Реализация операций включения и исключения представлена в алгоритме 2.5.



Переполнение означает ошибку, а пустота очереди – окончание работы алгоритма (или некоторого его фрагмента). Следует обратить внимание на то, что переполнение появляется, когда к очереди из $m-1$ элементов добавляется m-й элемент, поэтому один из m компонентов, отведенных для хранения элементов очереди, остается пустым (емкость очереди равна $m-1$).

$Q \Leftarrow x$	$x \Leftarrow Q$
$r \leftarrow (r+1)$ **mod** m **if** $r = f$ **then** // переполнение **else** $Q_r \leftarrow x$	**if** $r = f$ **then** // очередь пуста **else** $\begin{cases} f \leftarrow (f+1) \textbf{ mod } m \\ x \leftarrow Q_f \end{cases}$

Алгоритм 2.5. Операции включения и исключения для очереди на базе массива

На рис. 2.11 показан пример реализации очереди на базе массива (емкость очереди равна 7) и процесс изменения ее состояния во времени. Пусть очередь находится в некотором состоянии *а*, когда она содержит 4 элемента, начало очереди – элемент 2, конец очереди – элемент 9 (рис. 2.11, *а*). После выполнения операций включения $Q \Leftarrow 3$ и $Q \Leftarrow 8$ очередь переходит в состояние *б*, началом очереди остается элемент 2, концом очереди становится элемент 8 (рис. 2.11, *б*). После исключения элемента 2 из начала очереди она переходит из состояния *б* в состояние *в*, началом очереди становится элемент 5, концом очереди остается элемент 8 (рис. 2.11, *в*). Хотя после исключения элемента 2 он по-прежнему присутствует в массиве, в очереди его уже нет, так как началом очереди является элемент 5.

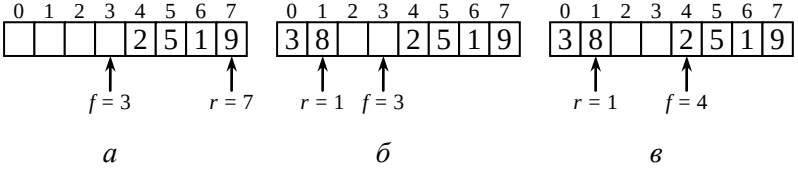


Рис. 2.11. Реализация очереди на базе массива:
а – очередь содержит 4 элемента, начало – элемент 2, конец – элемент 9;
б – очередь после выполнения операций $Q \Leftarrow 3$ и $Q \Leftarrow 8$;
в – очередь после исключения элемента 2 из начала очереди

Недостатком последовательной реализации очереди является фиксированная емкость очереди, что может привести к переполнению. От указанного недостатка свободна реализация очереди с использованием связного распределения.

Очередь можно реализовать на базе односвязного списка, узел которого состоит из поля *info*, содержащего элемент очереди, и поля связи *next* для указания элемента последовательности, идущего после данного. Функции указателя *f* начала очереди выполняет внешний указатель *list* связного списка. Требуется также внешний указатель *r* конца очереди, который ссылается на последний элемент списка. Очередь пуста, если $f = \Lambda$. Операции $Q \Leftarrow x$ соответствует процедура *INSERT* включения элемента в список, при этом поскольку включение производится всегда в конец связного списка, присваивание $l.next \leftarrow p.next$ можно заменить на $l.next \leftarrow \Lambda$. Необходимо учитывать также особый случай, когда элемент включается в пустую очередь, поскольку в этом случае требуется установка указателя *f* на начало очереди. Операции $x \Leftarrow Q$ соответствует процедура *DEL_FIRST* исключения первого элемента списка, модифицированная так, чтобы переменной *x* присваивалось значение исключаемого элемента. При этом следует учитывать особый случай, когда исключается элемент из очереди, состоящей из этого единственного элемента, поскольку в результате исключения очередь становится пустой и, следовательно, значение указателя *r* конца очереди должно быть нулевым.

Реализация очереди упрощается, если использовать для ее представления связный список с заголовком, поскольку в этом случае нет необходимости в выявлении особых случаев. При таком представлении очереди поле *info* заголовка не используется, поле *next* заголовка указывает на первый элемент очереди, указатель f – на заголовок списка, указатель r – на последний элемент очереди. Пустая очередь состоит из одного узла-заголовка, на который ссылаются указатели f и r, и значение его поля *next* равно Λ. В качестве критерия пустоты очереди можно использовать либо условие $f.next = \Lambda$, либо условие $f = r$. Соответствующие операции включения и исключения представлены в алгоритме 2.6.

$Q \Leftarrow x$	$x \Leftarrow Q$
$new(r.next)$ $r \leftarrow r.next$ $r.info \leftarrow x$ $r.next \leftarrow \Lambda$	**if** $f = r$ **then** // очередь пуста **else** $\begin{cases} x \leftarrow f.next.info \\ l \leftarrow f \\ f \leftarrow f.next \\ dispose(l) \end{cases}$

Алгоритм 2.6. Операции включения и исключения для очереди на базе списка

Для многих алгоритмов проверку пустоты очереди целесообразно выделить в отдельную операцию (будем записывать в виде $Q = \varnothing$). В этом случае из операции $x \Leftarrow Q$ исключается проверка пустоты очереди и считается, что она определена только для непустой очереди. Операцию, которая устанавливает очередь Q в начальное состояние (т. е. делает ее пустой), будем записывать в виде $Q \leftarrow \varnothing$. Все операции с очередью выполняются за время $O(1)$.