**ГЛАВА 3. Структуры данных**

В зависимости от реализации (посредством массивов или указателей) структуры

данных можно четко разбить на два типа — смежные и связные.

***Смежные структуры данных*** реализованы в виде непрерывных блоков памяти.

К ним относятся массивы, матрицы, кучи и хэш-таблицы.

***Связные структуры данных*** реализованы в отдельных блоках памяти, связанных

вместе с помощью указателей. К этому виду структур данных относятся списки,

деревья и списки смежных вершин графов.

Массивы

Массив представляет собой основную структуру данных смежного типа. Записи данных в массивах имеют постоянный размер, что позволяет с легкостью найти любой элемент по его индексу (или адресу).

**Постоянное время доступа** при условии наличия индекса. Так как индекс каждого элемента массива соответствует определенному адресу в памяти, то при наличии соответствующего индекса доступ к произвольному элементу массива осуществляется практически мгновенно.

**Эффективное использование памяти**. Массивы содержат только данные, поэтому память не тратится на указатели и другую форматирующую информацию. Кроме этого, для элементов массива не требуется использовать метку конца записи, т. к. все элементы массива имеют одинаковый размер.

**Локальность в памяти**. Одна из самых распространенных идиом

программирования — обработка элементов структуры данных в цикле. Массивы хорошо подходят для операций такого типа, поскольку обладают отличной локальностью в памяти. В современных компьютерных архитектурах физическая непрерывность последовательных обращений к данным помогает воспользоваться высокоскоростной кэш-памятью.

**Стеки.** Извлечение данных осуществляется в порядке LIFO ("last in, first out", "последним вошел — первым вышел"). Стеки легко реализуются и обладают высокой эффективностью. По этой причине стеки удобно применять в случаях, когда порядок извлечения данных не имеет никакого значений, например, при обработке пакетных заданий. Операции вставки и извлечения данных для стеков называются push (запись в стек) и pop (снятие со стека):

• push(x,s) — вставить элемент х на верх (в конец) стека s;

• pop(s) — извлечь (и удалить) верхний (последний) элемент из стека s.

Порядок LIFO возникает во многих реальных ситуациях. Например, из набитого битком вагона метро пассажиры выходят в порядке LIFO. Продукты из холодильника нередко вынимаются в этом же порядке, с игнорированием сроков годности. По крайней мере, такой порядок применяется в моем холодильнике. В алгоритмах порядок LIFO обычно возникает при выполнении рекурсивных операций.

**Очереди**. Очереди поддерживают порядок извлечения FIFO ("first-in, first-out", "первым вошел — первым вышел"). Использование этого порядка определенно самый справедливый способ управления временем ожидания обслуживания. Компьютер обрабатывает задачи в порядке FIFO, чтобы минимизировать максимальное время ожидания. Обратите внимание, что среднее время ожидания будет одинаковым, независимо от применяемого порядка, будь то FIFO или LIFO. Но так как данные многих вычислительных приложений не теряют актуальность бесконечно долго, вопрос максимального времени ожидания становится чисто академическим.

**Словари**

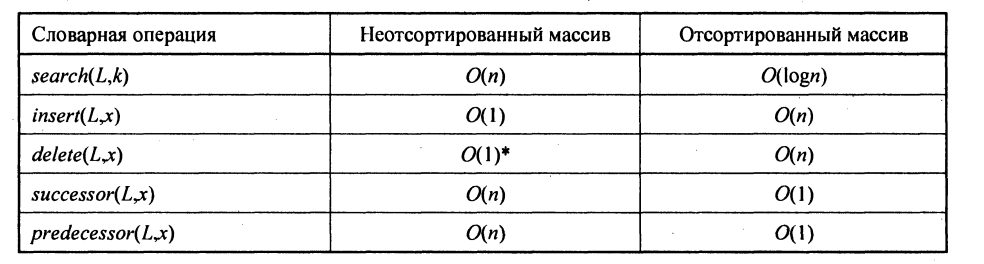
Тип данных словарь позволяет доступ к данным по содержимому. Словари применяются для хранения данных, которые можно быстро найти в случае надобности. Далее приводится список основных операций, поддерживаемых в словарях:

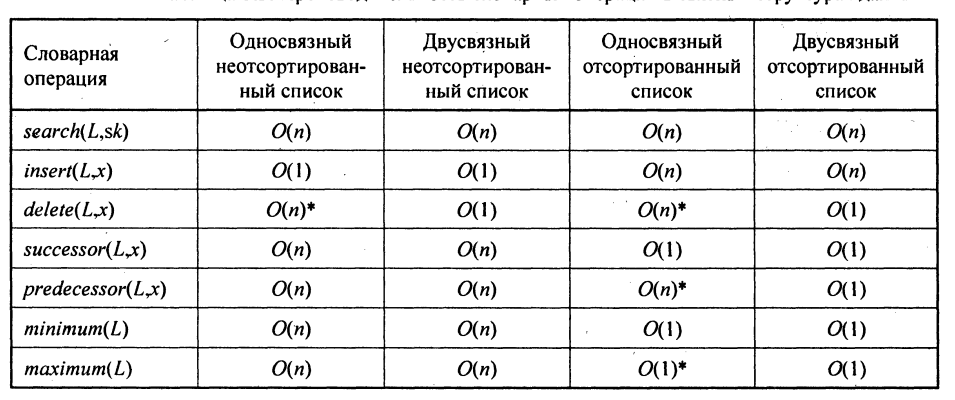
search(D,k) — возвращает указатель на элемент словаря D с ключом £, если такой элемент существует;

insert(D,x) — добавляет элемент, на который указывает х, в словарь Д delete(D,x) — удаляет из словаря D элемент, на который указывает х. Некоторые словари также поддерживают другие полезные операции:

max(D) и min(D)— возвращает указатель на элемент множества Д имеющий наибольший или наименьший ключ. Это позволяет использовать словарь в качестве очереди с приоритетами, которая рассматривается в разделе 3.5;

predecessor(D,k) и successor(D,к) — возвращает элемент из отсортированного массива Д предшествующий элементу с ключом к или стоящий сразу после него соответственно. Это позволяет обрабатывать в цикле все элементы этой структуры данных.





**Двоичные деревья поиска**

