3.2

3.2. Структуры данных. Линейные (списки, очереди, деки, вектора). Очереди с приоритетами. Деревья поиска.

Структуры данных.

В некоторых алгоритмах, предназначенных для обработки множеств, требуется выполнять операции нескольких различных видов. Например, набор операций, используемых во многих алгоритмах, ограничивается возможностью вставлять элементы в множество, удалять их, а также проверять, принадлежит ли множеству тот или иной элемент. Динамическое множество, поддерживающее перечисленные операции, называется словарем (dictionary). В других множествах могут потребоваться более сложные операции. Оптимальный способ реализации динамического множества зависит от того, какие операции должны им поддерживаться.

Стеки и очереди

Стеки и очереди — это динамические множества, элементы из которых удаляются с помощью предварительно определенной операции Delete. Первым из стека (stack) удаляется элемент, который был помещен туда последним: в стеке реализуется стратегия "последним вошел — первым вышел" (last-in, first-out — LIFO). Аналогично, в очереди (queue) всегда удаляется элемент, который содержится в множестве дольше других: в очереди реализуется стратегия "первым вошел — первым вышел" (first-in, first-out — FIFO).
Операция вставки применительно к стекам часто называется Push (запись в стек), а операция удаления — Pop (снятие со стека). Протестировать стек на наличие в нем элементов можно с помощью операции-запроса Stack_Empty. Любая из трех описанных операций со стеком выполняется в течение времени O (1).
Применительно к очередям операция вставки называется Enqueue (поместить в очередь), а операция удаления — Dequeue (вывести из очереди). Подобно стековой операции Pop, операция Dequeue не требует передачи элемента массива в виде аргумента. Благодаря свойству FIFO очередь подобна, например, живой очереди к врачу в поликлинике. У нее имеется голова (head) и хвост (tail). Когда элемент ставится в очередь, он занимает место в ее хвосте, точно так же, как человек занимает очередь последним, чтобы попасть на прием к врачу. Из очереди всегда выводится элемент, который находится в ее головной части аналогично тому, как в кабинет врача всегда заходит больной, который ждал дольше всех.

Очереди с приоритетом

В реальных задачах иногда возникает необходимость в формировании очередей, отличных от FIFO или LIFO. Порядок выборки элементов из таких очередей определяется приоритетами элементов. Приоритет в общем случае может быть представлен числовым значением, которое вычисляется либо на основании значений каких-либо полей элемента, либо на основании внешних факторов. Так, и FIFO, и LIFO-очереди могут трактоваться как приоритетные очереди, в которых приоритет элемента зависит от времени его включения в очередь. При выборке элемента всякий раз выбирается элемент с наибольшим приоритетом.
Очереди с приоритетами могут быть реализованы на линейных списковых структурах - в смежном или связном представлении. Возможны очереди с приоритетным включением - в которых последовательность элементов очереди все время поддерживается упорядоченной, т.е. каждый новый элемент включается на то место в последовательности, которое определяется его приоритетом, а при исключении всегда выбирается элемент из начала. Возможны и очереди с приоритетным исключением - новый элемент включается

всегда в конец очереди, а при исключении в очереди ищется (этот поиск может быть только линейным) элемент с максимальным приоритетом и после выборки удаляется из последовательности. И в том, и в другом варианте требуется поиск, а если очередь размещается в статической памяти - еще и перемещение элементов.
Наиболее удобной формой для организации больших очередей с приоритетами является сортировка элементов по убыванию приоритетов частично упорядоченным деревом (имеется в виду куча, по всей видимости).

Деки

Дек - особый вид очереди. Дек (от англ. deq - double ended queue,т.е очередь с двумя концами) - это такой последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка. Частный случай дека - дек с ограниченным входом и дек с ограниченным выходом. Логическая и физическая структуры дека аналогичны логической и физической структуре кольцевой FIFO-очереди. Однако, применительно к деку целесообразно говорить не о начале и конце, а о левом и правом конце.
Операции над деком:

- включение элемента справа;
- включение элемента слева;
- исключение элемента справа;
- исключение элемента слева;
- определение размера;
- очистка.

Векторы
Вектор (одномерный массив) - структура данных с фиксированным числом элементов одного и того же типа типа. Каждый элемент вектора имеет уникальный в рамках заданного вектора номер. Обращение к элементу вектора выполняется по имени вектора и номеру требуемого элемента.

Связанный список

Связанный список (linked list) — это структура данных, в которой объекты расположены в линейном порядке. Однако, в отличие от массива, в котором этот порядок определяется индексами, порядок в связанном списке определяется указателями на каждый объект. Связанные списки обеспечивают простое и гибкое представление динамических множеств.Списки могут быть разных видов. Список может быть однократно или дважды связанным, отсортированным или неотсортированным, кольцевым или некольцевым. Если список однократно связанный (однонаправленный) (singly linked), то указатель prev в его элементах отсутствует. Если список отсортирован (sorted), то его линейный порядок соответствует линейному порядку его ключей; в этом случае минимальный элемент находится в голове списка, а максимальный — в его хвосте. Если же список не отсортирован, то его элементы могут располагаться в произвольном порядке. Если список кольцевой (circular list), то указатель prev его головного элемента указывает на его хвост, а указатель next хвостового элемента — на головной элемент. Такой список можно рассматривать как замкнутый в виде кольца набор элементов.

Бинарные деревья (деревья поиска)

3.2

Для хранения указателей на родительский, дочерний левый и дочерний правый узлы бинарного дерева, используются поля parent, left и right. Если parent [x] = nil, то x — корень дерева. Если у узла x нет дочерних узлов, то left [x] = right [x] = nil. Атрибут root указывает на корневой узел дерева. Если root = nil, то дерево пустое.

Корневые деревья с произвольным ветвлением (а это, кажется, и не нужно)

Схему представления бинарных деревьев можно обобщить для деревьев любого класса, в которых количество дочерних узлов не превышает некоторой константы k. При этом поля правый и левый заменяются полями child1, child2, ..., childk. Если количество дочерних элементов узла не ограничено, то эта схема не работает, поскольку заранее не известно, место для какого количества полей (или массивов, при использовании представления с помощью нескольких массивов) нужно выделить. Кроме того, если количество дочерних элементов k ограничено большой константой, но на самом деле у многих узлов потомков намного меньше, то значительный объем памяти расходуется напрасно.
К счастью, существует остроумная схема представления деревьев с произвольным количеством дочерних узлов с помощью бинарных деревьев. Преимущество этой схемы в том, что для любого корневого дерева с n дочерними узлами требуется объем памяти O (n). Достаточно хранить левый дочерний и правый сестринский узел (left-child, right-sibling representation). Как и в предыдущем представлении, в каждом узле этого представления содержится указатель parent, а атрибут root указывает на корень дерева T. Однако вместо указателей на дочерние узлы каждый узел x содержит всего два указателя:
1. в поле left _ child [x] хранится указатель на крайний левый дочерний узел узла x
2. в поле right _ sibling [x] хранится указатель на узел, расположенный на одном уровне с узлом x справа от него.
Если узел x не имеет потомков, то left _ child [x] = NIL, а если узел x — крайний правый дочерний элемент какого-то родительского элемента, то right _ sibling [x] — NIL.