Лекция

Динамические структуры данных: стек, очередь, дек

Абстрактные структуры данных предназначены для удобного хранения и доступа к информации. Они предоставляют удобный интерфейс для типичных операций с хранимыми объектами, скрывая детали реализации от пользователя. Конечно, это весьма удобно и позволяет добиться большей модульности программы. Абстрактные структуры данных иногда делят на две части: *интерфейс*, набор операций над объектами, который называют АТД (*абстрактный тип данных*) и *реализацию*.

Простыми абстрактные структурами данных, являются, например, числа в языках программирования. Языки программирования высокого уровня (*Pascal*, *C++* и др.) предоставляют удобный интерфейс для чисел: операции **+**, *****, **=** и т.п., но при этом скрывают саму реализацию этих операций, машинные команды.

Статические структуры данных

Статические структуры относятся к разряду непримитивных структур, которые, фактически, представляют собой структурированное множество примитивных, базовых, структур. Например, *вектор* может быть представлен упорядоченным множеством чисел. Поскольку по определению *статические структуры* отличаются отсутствием изменчивости, память для них выделяется один раз и ее объем остается неизменным до уничтожения структуры. Слово ‘статический’ относится скорее к реализации структуры, нежели к АТД.

Простейшая статическая структура данных — массив, где обращение к элементу происходит через его номер.

Слово ‘массив’ употребляется в различных контекстах:

- как АТД, т.е. множество с операциями:
  - Получить элемент с номером N
  - Записать элемент с номером N
- как физическая структура, реализованная в виде непрерывной области памяти.

В случае реализации массива через такую структуру (языки *Pascal*, *C++* и др.) номер соответствует смещению от начала области. В некоторых языках, например, PHP, массив (здесь АТД!) реализован по-другому.

Плюсов у массива (здесь о реализации) всего два, но зато больших:

- доступ за константное время к любому элементу
- память тратится только на данные[1]

Минус — один, но тоже большой: статичность, неизменность структуры.

Одномерный массив иногда называют *вектором*.

Динамические структуры данных

Динамические структуры по определению характеризуются отсутствием физической смежности элементов структуры в памяти непостоянством и непредсказуемостью размера (числа элементов) структуры в процессе ее обработки[2].

Поскольку элементы динамической структуры располагаются по непредсказуемым адресам памяти, адрес элемента такой структуры не может быть вычислен из адреса начального или предыдущего элемента.

Для установления связи между элементами динамической структуры используются *указатели*, через которые устанавливаются явные связи между элементами. Такое представление данных в памяти называется связным. Элемент динамической структуры состоит из двух полей:

- *информационного поля* или поля данных, в котором содержатся те данные, ради которых и создается структура; в общем случае информационное поле само является интегрированной структурой — вектором, массивом, другой динамической структурой и т.п.;
- *поле связок*, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры.

Когда связное представление данных используется для решения прикладной задачи, для конечного пользователя ‘видимым’ делается только содержимое информационного поля, а поле связок используется только программистом-разработчиком.

Достоинства связного представления данных — в возможности обеспечения значительной изменчивости структур:

- размер структуры ограничивается только доступным объемом машинной памяти;

[1] Здесь и далее будут опущены константные затраты памяти операционной системой, возникающие при реализации структуры. В частности, в большинстве ОС, при динамическом выделении памяти под массив, в начале соответствующей области памяти ставится специальная метка.
[2] Но такая структура позволяет оперировать и с элементами переменной длины. Но это еще дополнительно усложняет алгоритмы работы с ними и поэтому такие виды элементов применяются редко.

- при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей;
- большая гибкость структуры.

Вместе с тем связное представление не лишено и *недостатков*, основные из которых:

- на поля связок расходуется дополнительная память;
- доступ к элементам связной структуры может быть менее эффективным по времени.

Последний недостаток является наиболее серьезным и именно им ограничивается применимость связного представления данных. Если в смежном представлении данных (массивах) для вычисления адреса любого элемента во всех случаях достаточно было номера элемента и информации, содержащейся в дескрипторе структуры, то для связного представления адрес элемента не может быть вычислен из исходных данных. *Дескриптор* связной структуры *содержит один* или *несколько указателей*, позволяющих войти в структуру, далее поиск требуемого элемента выполняется следованием по цепочке указателей от элемента к элементу. Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид вектора или массива - с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другой исходной информации доступа (таблицы, списки, деревья и т.д.).

Списки: односвязные и двусвязные

Списком называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения. Список, отражающий отношения соседства между элементами, называется линейным. *Длина* списка равна числу элементов, содержащихся в списке, список нулевой длины называется *пустым списком*. Линейные связные списки являются простейшими динамическими структурами данных.

Графически связи в списках удобно изображать с помощью стрелок. Если компонента не связана ни с какой другой, то в поле указателя записывают значение, не указывающее ни на какой элемент. Такая ссылка обозначается специальным именем — NULL (nil).

На рис. 1 приведена структура *односвязного списка*. На нем поле INF — информационное поле, данные, NEXT — указатель на следующий элемент списка. Каждый список должен иметь особый элемент, называемый указателем начала списка или головой списка, который обычно по формату отличен от остальных элементов. В поле указателя последнего элемента списка находится специальный признак NULL (nil), свидетельствующий о конце списка.

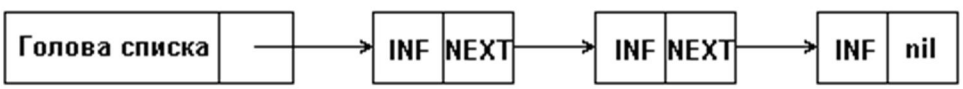

Рис. 1. Представление односвязного списка в памяти

Двусвязный список характеризуется наличием пары указателей в каждом элементе: на предыдущий элемент и на следующий:

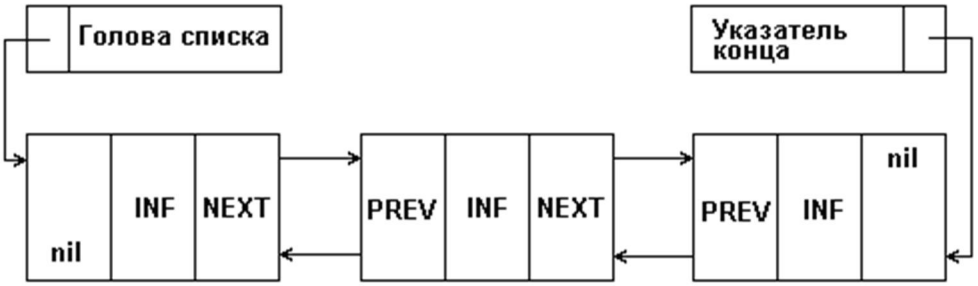

Рис. 2. Представление двусвязного списка в памяти

Очевидный плюс тут в том, что от данного элемента структуры мы можем пойти в обе стороны. Таким образом упрощаются многие операции. Однако на указатели тратится дополнительная память.

Разновидностью рассмотренных видов линейных списков является *кольцевой список*, который может быть организован на основе как односвязного, так и двухсвязного списков. При этом в односвязном списке указатель последнего элемента должен указывать на первый элемент; в двухсвязном списке в первом и последнем элементах соответствующие указатели переопределяются, как показано на рис. 3.

При работе с такими списками несколько упрощаются некоторые процедуры. Однако, при просмотре такого списка следует принять некоторых мер предосторожности, чтобы не попасть в бесконечный цикл.

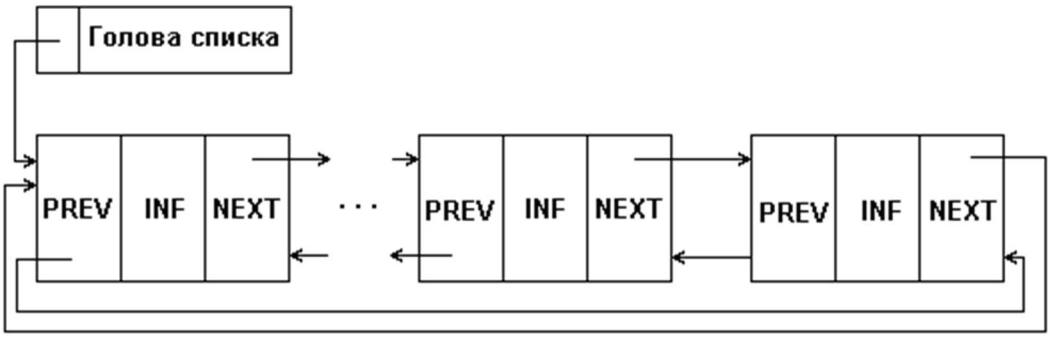

Рис. 3: Структура кольцевого двухсвязного списка

Описываемые ниже АТД могут быть организованы на основе элементов

1. **массива**: выделяется место под N элементов разом, а затем описываются операции над данным типом данных в терминах операций над элементами массива.
2. **списка**: память выделяется и освобождается по мере необходимости.

Первый вариант быстрее, но лишь второй истинно динамический. Соответственно, в различных приложениях может быть предпочтителен первый (размер структуры известен и небольшой) или второй (размер заранее неизвестен). Здесь будет рассматриваться преимущественно динамические решения.

Уменьшить несколько затраты на ‘переупорядочивание’ элементов массива с большими структурами можно используя дополнительный массив — массив индексов. Т.е. выполнять операции над ‘легкими’ элементами массива с индексами, не трогая ‘тяжелые’ элементы массива структур.

Стек

Стек (англ. *stack*) — такой последовательный список с переменной длиной, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого вершиной стека. Применяются и другие названия стека - магазин и очередь, функционирующая по принципу LIFO (Last–In–First–Out — ‘последним пришел — первым исключается’). Примеры стека: винтовочный патронный магазин, тупиковый железнодорожный разъезд для сортировки вагонов, стопка книг.

Основные операции над стеком — включение нового элемента (английское название *push* — заталкивать) и исключение элемента из стека (англ. *pop* — выскакивать).

Полезными могут быть также вспомогательные операции:

- определение текущего числа элементов в стеке;
- очистка стека;
- неразрушающее чтение элемента из вершины стека, которое может быть реализовано, как отдельная операция или как комбинация основных операций:

```
x = pop(stack); push(stack,x);
```

Предупреждение. Некоторые авторы рассматривают также операции включения/исключения элементов для середины стека, однако структура, для которой возможны такие операции, не соответствует стеку по определению.

Стек должен поддерживать следующие операции:

push	Добавить (положить) в конец стека новый элемент
pop	Извлечь из стека последний элемент
back	Узнать значение последнего элемента (не удаляя его)
size	Узнать количество элементов в стеке
clear	Очистить стек (удалить из него все элементы)

Для наглядности рассмотрим небольшой пример, демонстрирующий принцип включения элементов в стек и исключения элементов из стека. На рис. 4 изображены состояния стека:

- а) пустого;
- б-г) после последовательного включения в него элементов с именами 'A', 'B', 'C';
- д, е) после последовательного удаления из стека элементов 'C' и 'B';
- ж) после включения в стек элемента 'D'.

Рис. 4. Включение и исключение элементов из стека

Как видно из рис. 4, стек можно представить, например, в виде стопки книг (элементов), лежащей на столе. Присвоим каждой книге свое название, например, A, B, C, D.... Тогда в момент времени, когда на столе книг нет, про стек аналогично можно сказать, что он пуст, т.е. не содержит ни одного элемента. Если же мы начнем последовательно класть книги одну на другую, то получим стопку книг (допустим, из n книг), или получим стек, в котором содержится n элементов, причем вершиной его будет являться элемент n+1. Удаление элементов из стека осуществляется аналогичным образом т. е. удаляется последовательно по одному элементу, начиная с вершины, или по одной книге из стопки.

Очередь (список FIFO)

Очередью (англ. *queue*) FIFO (First–In–First–Out — ‘первым пришел — первым исключается’) называется такой последовательный список с переменной длиной, в котором включение элементов выполняется только с одной стороны списка (эту сторону часто называют *концом* или хвостом очереди), а исключение — с другой стороны (называемой *началом* или головой очереди). Те самые очереди к прилавкам и к кассам, которые мы так не любим, являются типичным бытовым примером очереди FIFO.

Основные операции над очередью — те же, что и над стеком — включение, исключение, определение размера, очистка, неразрушающее чтение.

Дек

Дек — особый вид очереди. Дек (от англ. deque — 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.

Система команд дека:

push_front Добавить (положить) в начало дека новый элемент
push_back Добавить (положить) в конец дека новый элемент
pop_front Извлечь из дека первый элемент
pop_back Извлечь из дека последний элемент
front Узнать значение первого элемента (не удаляя его)
back Узнать значение последнего элемента (не удаляя его)
size Узнать количество элементов в деке
clear Очистить дек (удалить из него все элементы)

Физическая структура дека в статической памяти идентична структуре кольцевой очереди. Динамическая реализация является очевидным объединением стека и очереди.

Задачи, требующие структуры дека, встречаются в вычислительной технике и программировании гораздо реже, чем задачи, реализуемые на структуре стека или очереди. Как правило, вся организация дека выполняется программистом без каких-либо специальных средств системной поддержки.

Примером дека может быть, например, некий терминал, в который вводятся команды, каждая из которых выполняется какое-то время. Если ввести следующую команду, не дождавшись, пока закончится выполнение предыдущей, то она встанет в очередь и начнет выполняться, как только освободится терминал. Это FIFO очередь. Если же дополнительно ввести операцию отмены последней введенной команды, то получается дек.

Послесловие

Стеки, деки, очереди можно рассматривать как динамические АТД — списки, но с простыми наборами действий. Стеки, деки, очереди — списки с небольшим набором операций над конечными элементами. Над списками можно разрабатывать разнообразные алгоритмы по оперированию с любыми элементами, как конечными, так и внутренними.

Приведённый материал может вызвать лёгкое недоумение и сомнения в целесообразности всех этих ограничений на обычные массивы.

На самом деле, организация работы над данными по принципу *стек*, *очередь* и *дек* являются лишь простейшими примерами структур данных, не предоставляющими выгоды в скорости выполнения операций. Серьёзным структурам данных нужно посвящать отдельные лекции. Конкретная форма таких структур определяется условиями задачи. Стек, очередь и дек призваны на простых примерах проиллюстрировать, что главной характеристикой структуры данных является набор поддерживаемых операций и скорость их выполнения.

Использованы ресурсы

http://algolist.manual.ru/ds/basic/

http://algolist.manual.ru/ds/basic/simple_list.php

http://informatics.mccme.ru/mod/book/view.php?id=580