Тема 2. СТРУКТУРЫ ДАННЫХ

2.2. Связное распределение

2.2.1. Связный список

При *связном распределении* для хранения множества не требуется выделения непрерывной области памяти (в отличие от последовательного распределения). Поэтому элементы множества $X = \{x_1, x_2, \ldots, x_n\}$ могут располагаться в произвольных областях памяти. Чтобы сохранить информацию о порядке следования элементов, необходимо каждому элементу x_i поставить в соответствие *указатель* (*ссылку*) на следующий элемент x_{i+1}. Значение указателя можно трактовать как адрес области памяти, где находится соответствующий элемент. Такое представление множества называется *связным списком*. Каждый элемент списка, называемый *узлом*, состоит из двух полей: в поле *info* размещается сам элемент множества, а в поле связи *next* – указатель на следующий за ним элемент. Доступ к узлу возможен только в том случае, если на него указывает хотя бы один указатель. Значение поля узла представлено в виде *указатель***.***имя_поля*. Например, запись *p***.***info* определяет значение поля *info* узла, на который ссылается указатель *p*. Для доступа ко всему списку должен существовать внешний указатель *list*, который указывает на первый элемент списка. Поле *next* последнего элемента списка, поскольку для него не существует следующего элемента, должно содержать так называемое *пустое*, или *нулевое*, значение указателя, которое будем обозначать символом Λ (**nil** в Паскале). Список, не содержащий элементов, называется *пустым*, и для него *list* = Λ. Представление множества $X = \{x_1, x_2, \ldots, x_n\}$ в виде связного списка показано на рис. 2.1.

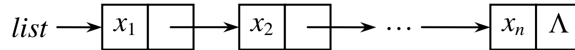

Рис. 2.1. Связный список

Связное распределение ориентировано на реализацию динамических множеств и облегчает выполнение соответствующих операций. Поэтому подобные структуры часто называют *динамическими*. Следует отметить, что в отличие от массива для хранения списка память заранее не резервируется. Поэтому должен существовать механизм, который по мере необходимости выделяет требуемый объем памяти для размещения узла. Предполагая, что такой механизм существует (а он существует во многих языках программирования), для запроса на выделение памяти будем использовать процедуру *new* (p), которая размещает новый пустой узел в некоторой области памяти и присваивает указателю p ссылку на этот узел. Необходим также механизм освобождения области памяти, занимаемой некоторым узлом (*сборка мусора*). Это связано с тем, что если к узлу по каким-либо причинам (например, в результате исключения) отсутствует возможность доступа, то этот узел не может быть использован и становится бесполезным, однако он занимает определенную область памяти. Для освобождения памяти, занимаемой узлом, на который ссылается указатель p, будем использовать процедуру *dispose*(p), после выполнения которой указатель $p = \Lambda$ и любая ссылка по этому значению указателя запрещена.

Включение элемента z в позицию i множества X предполагает создание нового узла, запись значения элемента z в поле *info* этого узла и установку значений двух указателей (рис. 2.2). Новый узел должен следовать в списке за узлом с элементом x_{i-1}, поэтому с помощью указателя p необходимо обеспечить доступ к этому узлу для изменения значения поля *next*, чтобы оно указывало на добавленный узел. Старое значение этого поля, которое указывало на узел с элементом x_i, для сохранения целостности списка должно быть предварительно записано в поле *next* нового узла. Процедура *INSERT*(z, p), реализующая включение элемента z после узла, на который ссылается указатель p, представлена в алгоритме 2.1. Данная процедура не может включить элемент в начало списка или в пустой список, поскольку в этом особом случае необходимо изменить значение указателя *list*. Для реализации такого включения служит процедура *INS_FIRST*. При необходимости эти процедуры можно объединить в одну, включив в нее действия по распознаванию особого случая.

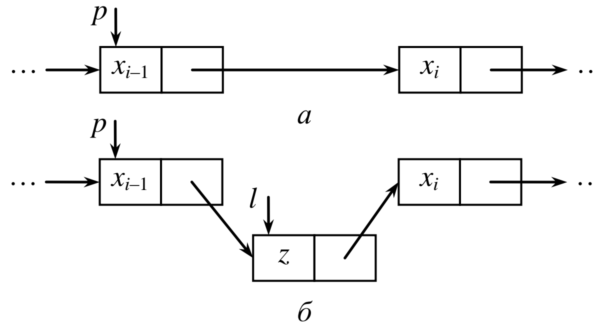

Рис. 2.2. Включение элемента в связный список:
а – до включения; *б* – после включения

procedure $INSERT(z, p)$
  $new(l)$
  $l.info \leftarrow z$
  $l.next \leftarrow p.next$
  $p.next \leftarrow l$
return

procedure $INS_FIRST(z, list)$
  $new(l)$
  $l.info \leftarrow z$
  $l.next \leftarrow list$
  $list \leftarrow l$
return

Алгоритм 2.1. Процедуры включения элемента в связный список

Исключение элемента x_i из множества X предполагает удаление из списка узла, содержащего элемент x_i, установку значения одного указателя и освобождение памяти, занимаемой исключаемым узлом (рис. 2.3). Исключаемый узел следует в списке за узлом с элементом x_{i-1}, поэтому с помощью указателя p необходимо обеспечить доступ к этому узлу для изменения значения поля *next*, чтобы оно указывало на узел с элементом x_{i+1}. Процедура *DELETE* (p), реализующая исключение узла, расположенного после узла, на который ссылается указатель p, приведена в алгоритме 2.2. Очевидно, что, если не стоит задача сборки мусора, для исключения узла достаточно установить $p.next \leftarrow p.next.next$). Процедура *DELETE* не может исключить элемент из начала списка. Для реализации этого особого случая служит процедура *DEL_FIRST*. Эти процедуры также можно объединить в одну, включив в нее действия по распознаванию особого случая.

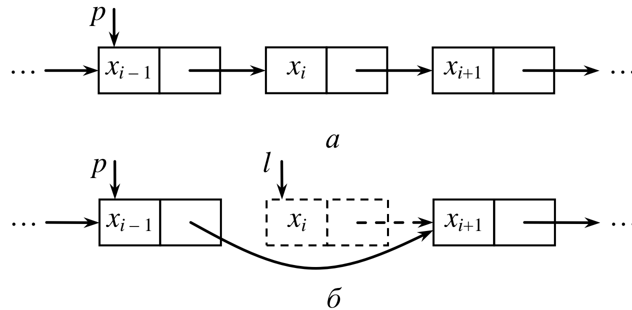

Рис. 2.3. Исключение элемента из связного списка:
а – до исключения; *б* – после исключения

```
procedure DELETE(p)
   l ← p.next
   p.next ← l.next
   dispose(l)
return
```

```
procedure DEL_FIRST(list)
   l ← list
   list ← l.next
   dispose(l)
return
```

Алгоритм 2.2. Процедуры исключения элемента из связного списка

Таким образом, операции включения и исключения выполняются за некоторое фиксированное время, не зависящее от размеров множества. Так же легко за фиксированное время выполняются операции конкатенации (сцепления) и разбиения списков. Для сцепления двух списков достаточно установить значение поля *next* последнего элемента первого списка (при этом должен быть обеспечен доступ к последнему элементу, например, с помощью специального указателя), равным значению указателя на первый элемент второго списка. Разбиение на два списка можно выполнить, если обеспечен доступ к узлу, непосредственно предшествующему месту разбиения.

Основным достоинством связного распределения является их удобство для реализации динамических множеств, поскольку большинство операций, ориентированных на модификации множеств, выполняются за время, не зависящее от их размеров.

Существенным недостатком связного распределения является невозможность прямого доступа к элементу множества (за исключением первого), т. е. возможен только последовательный доступ. Например, если необходимо получить доступ к элементу x_i, следует последовательно, начиная с первого элемента, перемещаться по $i - 1$ узлам списка, пока не будет достигнут узел с элементом x_i. Это требует времени в среднем $O(n)$. Другой недостаток связан с тем, что необходима дополнительная память для хранения указателей.

2.2.2. Реализация связных списков

В таких языках программирования, как Паскаль, предусмотрены специальные средства для реализации объектов, имеющих несколько полей (комбинированные типы), и указателей (ссылочные типы) с соответствующими механизмами выделения и освобождения памяти. Однако это не единственный способ реализации связных списков. Иногда (либо язык не имеет соответствующих средств, либо алгоритм становится более эффективным) для реализации указателей может оказаться полезным использование массивов. В этом случае в качестве указателя используется индекс в массиве. Пустому значению Λ указателя должно соответствовать число, не являющееся индексом никакого элемента массива (например, 0). В приводимых ниже примерах доступ к элементу массива осуществляется путем указания имени массива и в квадратных скобках индекса элемента.

Пример представления с помощью массивов двух связных списков $X = \{a, b, c\}$ и $Y = \{d, e, g\}$ показан на рис. 2.4. В массиве *info* хранятся элементы множества, а в массиве *next* – указатели, т. е. индексы позиций в массивах, где расположены последующие элементы. Поскольку при таком представлении нет встроенных механизмов выделения и освобождения памяти, они должны быть предусмотрены при реализации. Чтобы вести учет использования позиций массивов, все свободные позиции объединены в отдельный связный *список свободных позиций* с указателем *free*, который представляет собой индекс первой свободной позиции в списке. Данный список может обеспечивать обслуживание нескольких связных списков, имеющих одинаковый тип элементов.

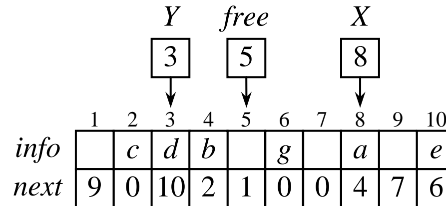

Рис. 2.4. Представление связных списков с помощью массивов

Процедура выделения памяти $new(p)$ для обеспечения включения нового элемента в список сначала должна проверить наличие свободных позиций (если $free = 0$, свободных позиций нет). Затем, если имеются свободные позиции ($free \neq 0$), определить индекс первой из них (установив значение указателя $p = free$) и исключить эту позицию из списка свободных позиций (присвоив указателю *free* значение $next[free]$). Например, пусть требуется включить элемент f в список Y после элемента e, хранящегося в $info[10]$. Тогда процедура $new(p)$ установит указатель $p = 5$ и, установив указатель $free = next[free] = 1$, исключит позицию 5 из списка свободных позиций. Операция включения запишет элемент f в $info[5]$ и установит значения $next[5] = next[10] = 6$ и $next[10] = 5$ (рис. 2.5).

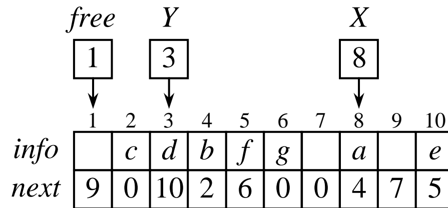

Рис. 2.5. Включение элемента *f* в список *Y*

Процедура освобождения памяти *dispose* (*p*) позицию *p*, которая стала свободной в результате исключения элемента из списка, включает в начало списка свободных позиций, записав значение указателя *free* в позицию *p* массива *next* и установив новое значение указателя *free* = *p*. В качестве примера рассмотрим исключение элемента *e* из списка $Y = \{d, e, f, g\}$ (рис. 2.5). Исключаемый элемент *e* хранится в *info* [10], а предшествующий ему элемент *d* – в *info* [3]. Операция исключения устанавливает значение *next* [3] = *next* [10] = 5 и передает процедуре *dispose* (*p*) значение *p* = 10 для освобождения памяти, которая включает эту позицию в начало списка свободных позиций, установив значения *next* [10] = *free* = 1 и *free* = 10 (рис. 2.6). Хотя исключенный элемент *e* по-прежнему присутствует в *info* [10], в списке *Y* его уже нет, так как позиция 10 находится в списке свободных позиций.

Таким образом, список свободных позиций имеет одну точку доступа: новый элемент всегда добавляется в начало, исключается всегда первый элемент, т. е. список ведет себя как стек.

Y = 3, *X* = 8, *free* = 10

	1	2	3	4	5	6	7	8	9	10
info		*c*	*d*	*b*	*f*	*g*		*a*		*e*
next	9	0	5	2	6	0	0	4	7	1

Рис. 2.6. Исключение элемента *e* из списка *Y*

Вместо нескольких массивов, когда каждому полю узла связного списка соответствует свой массив, можно использовать один массив, элементами которого являются объекты комбинированного типа (записи). Каждая запись представляет собой узел списка и состоит из тех же полей.

2.2.3. Разновидности связных списков

Существуют различные виды связных списков, ориентированных на облегчение выполнения тех или иных операций.

Можно поместить в начало связного списка дополнительный фиктивный элемент (называемый *заголовком списка*), имеющий ту же структуру, что и остальные узлы списка (рис. 2.7). Поле *next* заголовка указывает на узел с первым элементом множества, поле *info* не содержит элемента множества. На практике поле *info* заголовка (если позволяет его структура) можно использовать для хранения некоторой информации о множестве (например, его размер). Пустой список представляется не нулевым значением указателя *list*, а состоит из одного заголовка, поле *next* которого равно Λ, т. е. критерием пустоты списка является условие *list*.*next* = Λ, а указатель *list* никогда не будет иметь нулевое значение.

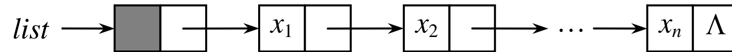

Рис. 2.7. Связный список с заголовком

Такая модификация связного списка упрощает выполнение операций включения и исключения, поскольку отпадает необходимость выявления особых случаев, связанных с включением элемента в начало списка (новый узел добавляется после заголовка) и исключением первого элемента списка (удаляется узел, непосредственно следующий за заголовком). Таким образом, для реализации этих операций достаточно использовать только процедуры *INSERT* и *DELETE*.

Другой разновидностью связного списка является *циклический* список, в котором поле *next* последнего элемента содержит указатель на первый элемент, а не пустое значение (рис. 2.8).

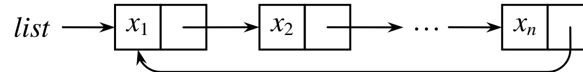

Рис. 2.8. Циклический список

Такой список дает возможность достигнуть любого элемента (хотя и не прямым доступом) из любого другого элемента списка. Операции включения и исключения реализуются так же, как и в нециклических списках. Особыми случаями являются включение в пустой список и исключение элемента из списка, если этот элемент является единственным (после его исключения список становится пустым). Другое удобство заключается в следующем. Если указатель *list* списка установить так, чтобы он указывал на последний элемент, то *list.next* будет указывать на первый элемент, т. е. одним указателем *list* обеспечивается возможность быстрого доступа не только к последнему, но и к первому элементу списка. Как и в обычный список, в циклический список при необходимости можно добавить заголовок.

В тех случаях, когда возникает необходимость в эффективном перемещении по списку как в прямом, так и обратном направлениях, удобно использовать *двусвязные списки*. В таких списках каждому элементу x_i вместо одного указателя ставится в соответствие два: один из них указывает на следующий элемент x_{i+1}, а другой – на предшествующий элемент x_{i-1}, т. е. в структуру узла добавляется поле *prev*, указывающее на предшествующий узел (рис. 2.9). В двусвязных списках имеется возможность прямого доступа к предшествующему и последующему элементам, поэтому удобно выполнять операции включения нового элемента перед заданным элементом x_i и исключения элемента x_i без предварительной установки указателя на предшествующий узел с элементом x_{i-1}. Все эти возможности достигаются за счет дополнительных затрат памяти и усложнения основных операций со списками. При необходимости в двусвязный список можно добавить заголовок. Двусвязный список можно сделать также циклическим, если поле *next* последнего элемента будет указывать на первый элемент списка, а поле *prev* первого элемента – на последний элемент списка.

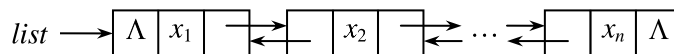

Рис. 2.9. Двусвязный список

Использование той или иной разновидности связных списков позволяет упростить выполнение одних операций со списками, но усложнить другие. Поэтому выбор представления должен осуществляться на основе анализа типов операций, которые будут выполняться над множеством. Реализация операций для различных видов связных списков предлагается в качестве упражнений для самостоятельной разработки.