

Тема 2. СТРУКТУРЫ ДАННЫХ

Процесс разработки алгоритма для решения некоторой прикладной задачи можно разбить на два этапа. На первом этапе осуществляется формализация исходной задачи, т. е. создается ее математическая модель с привлечением соответствующих математических конструкций, и строится неформальный алгоритм ее решения. На втором этапе определяются используемые абстрактные типы данных, и алгоритм записывается на некотором псевдоязыке (псевдокод, блок-схема алгоритма и др.) в терминах этих абстрактных типов данных.

Абстрактный тип данных (АТД) представляет собой математическую модель данных с множеством операций, определенных для этих данных. Реализация АТД предполагает выбор структуры данных для представления данных и разработку алгоритмов выполнения соответствующих операций. Концептуально АТД соответствует классу в объектно-ориентированном программировании.

При разработке алгоритмов часто приходится иметь дело с таким АТД, как множество. Примерами операций над множествами являются объединение, пересечение, определение принадлежности элемента множеству (поиск), модификация (добавление или исключение элемента, сортировка) и т. д. Трудно сформировать универсальный набор всех операций над множествами, удовлетворяющий всем возможным приложениям. Структуры данных для представления множеств во многом зависят от набора операций, которые необходимо выполнять над ними. Поэтому в этой главе рассматриваются только основные структуры данных для представления конечных множеств и некоторые операции.

Множество представляет собой объединение различных элементов. В некоторых приложениях более естественными являются понятие *мультимножества* (которое отличается от множества тем, что в нем могут содержаться одинаковые элементы) и понятие *последовательности*. Последовательность (список, кортеж, вектор) x_1, x_2, x_3, \dots формально определяется как функция, областью определения которой является множество положительных целых чисел. В такой индексированной последовательности каждый элемент занимает определенную позицию (индекс), т. е. существует однозначное соответствие между индексом элемента и элементом последовательности. В последовательности могут быть одинаковые элементы, но занимающие разные позиции. Рассматриваемые структуры данных для представления множеств очевидным образом можно использовать и для представления мультимножеств и последовательностей.

2.1. Последовательное распределение

Простейшим представлением множества $X = \{x_1, x_2, \dots, x_n\}$ является точный список его элементов, расположенных в последовательных ячейках памяти, т. е. множество представляется с помощью массива и требует для своего хранения непрерывную область памяти. Такое представление будем называть *последовательным распределением*.

Пусть для хранения одного элемента требуется d ячеек памяти, а элемент x_1 хранится, начиная с ячейки с адресом l_1 . Тогда элемент x_2 хранится, начиная с ячейки $l_2 = l_1 + d$, элемент x_3 – начиная с ячейки $l_3 = l_1 + 2d$ и т. д. Таким образом, существует простое соотношение между номером i элемента и адресом l_i ячейки, с которой начинается хранение x_i : $l_i = l_1 + (i - 1)d$. Данное соотношение обеспечивает *прямой доступ* к любому элементу множества, т. е. время доступа к любому элементу множества фиксировано и не зависит от его мощности.

x_1	x_2	x_2	\dots	x_{n-1}	x_n
l_1	l_2	l_3		l_{n-1}	l_n
	$=l_1+d$	$=l_1+2d$		$=l_1+(n-2)d$	$=l_1+(n-1)d$

Элемент x_i множества может иметь сложную структуру. Например, матрицу $[a_{ij}]$ размером $m \times n$ можно представить как множество строк s_1, s_2, \dots, s_m , в котором каждый элемент s_i , в свою очередь, является множеством, состоящим из n элементов. Обозначим через d число ячеек для хранения элемента a_{ij} . Тогда для хранения элемента s_i (строки матрицы) требуется nd ячеек, а сам элемент s_i хранится, начиная с ячейки $l_i = l_{11} + (i - 1)nd$, где l_{11} – адрес первого элемента первой строки. Таким образом, элемент a_{ij} хранится, начиная с ячейки

$$l_{ij} = l_i + (j - 1)d = l_{11} + ((i - 1)n + (j - 1))d.$$

В данном случае получена формула прямого доступа к элементу матрицы для ее построчного представления. Постолбцовое представление матрицы получается, если матрицу рассматривать как множество t_1, t_2, \dots, t_n столбцов, а каждый элемент t_j в свою очередь является множеством из m элементов j -го столбца матрицы.

Следующий пример иллюстрирует возможность обеспечения прямого доступа к элементам матрицы с различной длиной строк. В ряде приложений (например, минимизация конечных автоматов) используются симметричные матрицы $[a_{ij}]$ размером $n \times n$, в которых $a_{ij} = a_{ji}$. Для экономии памяти можно хранить не всю матрицу, а только верхний или нижний треугольник. Построим последовательное распределение для нижнего треугольника. Матрица представляет собой множество строк s_1, s_2, \dots, s_n , в котором каждый элемент s_i , в свою очередь, является множеством, состоящим из i элементов, т. е. для хранения элемента s_i необходимо id ячеек. Тогда элемент s_i хранится, начиная с ячейки

$$l_i = l_{11} + d + 2d + \dots + (i-1)d = l_{11} + di(i-1)/2,$$

а элемент a_{ij} – с ячейки

$$l_{ij} = l_i + (j-1)d = l_{11} + (i(i-1)/2 + j-1)d.$$

Очевидно, что при обращении к элементу матрицы всегда должно выполняться условие $i \geq j$. Поэтому, если $i < j$, необходимо обращаться к элементу a_{ji} . В результате такого представления экономится почти 50 % памяти, но несколько увеличивается время вычислений из-за дополнительной проверки условия $i < j$ при каждом обращении к элементу матрицы.

Множества, которые часто модифицируются, будем называть *динамическими*, в противном случае – *статическими*. Ограничимся рассмотрением двух операций для динамических множеств: включение (добавление) элемента в множество и исключение (удаление) элемента из множества.

При последовательном распределении операция включения вставляет элемент z в заданную позицию i множества X . Сначала производится перемещение элементов x_i, x_{i+1}, \dots, x_n на одну позицию вправо, затем в освободившуюся позицию i помещается элемент z . Операция исключения удаляет элемент из заданной позиции i , перемещая элементы x_{i+1}, \dots, x_n на одну позицию влево. Очевидно, что эти операций требуют времени $O(n)$.

Основным достоинством последовательного распределения является возможность прямого доступа к любому элементу множества, поскольку существует простое соотношение между порядковым номером элемента в множестве и адресом ячейки памяти, в которой он хранится. Кроме того, данное представление легко реализуемо и требует небольших расходов памяти (для статических множеств).

Существенным недостатком последовательного распределения является неэффективность реализации динамических множеств. Этот недостаток является следствием того, что массив по своей сути является статической структурой. Во-первых, размер массива приходится задавать (резервировать объем памяти) исходя из максимально возможного размера множества независимо от реального размера в конкретный момент времени, что может привести к неэффективному использованию памяти. Кроме того, не всегда можно заранее определить верхнюю границу мощности множества. Во-вторых, время выполнения операций включения и исключения зависит от размера множества.

Важной разновидностью последовательного распределения является *характеристический вектор*. Он представляет собой двоичный вектор и может быть полезен в тех случаях, когда необходимо представить подмножество некоторого основного множества. Пусть $S = \{s_1, s_2, \dots, s_n\}$ – основное множество. Тогда подмножество $X \subseteq S$ можно представить в виде характеристического вектора V_X , состоящего из n двоичных разрядов, такого, что i -й разряд в V_X равен единице, если $s_i \in X$, и равен нулю в противном случае. Например, характеристический вектор множества X всех простых чисел между 1 и 15, т. е. основное множество $S = \{1, 2, \dots, 15\}$, имеет вид

$$\begin{array}{l} S: \quad 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \\ V_X: \quad 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \end{array}$$

В качестве основных достоинств характеристических векторов можно отметить их компактность (для представления элемента используется один двоичный разряд), легкость выполнения таких операций, как определение принадлежности элемента данному множеству, включение и исключение (эти операции выполняются за фиксированное время, не зависящее от размера множества). Более того, такие операции, как объединение и пересечение, в ряде случаев можно осуществить с помощью операций дизъюнкции и конъюнкции над двоичными векторами.

Основной недостаток характеристических векторов обусловлен тем, что должно существовать простое соотношение между i и s_i . Если простого соотношения не существует, то это затрудняет (если соотношение сложное, может возрасти время обработки) или делает невозможным использование характеристических векторов. Кроме того, сильно разреженные векторы (которые содержат относительно мало единиц) неэкономичны с точки зрения памяти, поскольку требуемый объем памяти пропорционален мощности множества S , а не множества X . По этой же причине операции, основанные на обработке каждого элемента множества X (включая и операции объединения и пересечения множеств, если нет возможности использовать логические операции над двоичными векторами), требуют времени, пропорционального $|S|$.