Тема 4. Методы поиска

4.5. Внешний поиск

Все обсуждавшиеся в предыдущих разделах методы поиска предполагают, что таблица полностью умещается в оперативной памяти. Однако для большинства реальных задач обработки данных таблицы могут быть настолько большими, что они не умещаются в оперативной памяти и доступ к ним возможен только по частям. Такие большие таблицы хранятся во внешней памяти в виде файлов и для них необходимы специальные методы *внешнего поиска*. Очевидно, что для последовательных файлов может быть использован только последовательный поиск, модифицированный так, чтобы учитывать специфику организации данных в виде последовательного файла.

Для внешней памяти с возможностью прямого доступа используют более эффективные методы хранения данных, которые позволяют существенно ускорить поиск и другие операции. Типичным представителем такой памяти является диск. Как известно, диск разбивается на дорожки, а каждая дорожка делится на сектора. Файловая система отводит место для записи файлов кластерами. Каждый кластер содержит определенное количество секторов. Поскольку время доступа (поиск сектора) может быть относительно большим, обычно записывают или считывают сектор целиком. Часто обработка данных из прочитанного сектора занимает меньше времени, чем поиск нужного сектора. Таким образом, следует минимизировать число обращений к диску.

Одной из наиболее эффективных структур данных для представления больших таблиц во внешней памяти является сбалансированное m-арное дерево, которое является обобщением деревьев бинарного поиска.

Сбалансированное сильно ветвящееся дерево (или *Б-дерево*) порядка m есть корневое расширенное m-арное дерево, характеризующееся следующими свойствами:

1) корень либо является листом (пустое Б-дерево), либо имеет от 2 до m сыновей;

2) каждая внутренняя вершина, кроме корня, имеет от $\lceil m/2 \rceil$ до m сыновей. В общем случае нижняя граница числа сыновей определяется как $\lceil \alpha\, m \rceil$, $0 < \alpha < 1$. Чаще используется случай $\alpha = 1/2$, но возможны и другие значения α, которые задаются едиными для всего дерева;

3) все листья расположены на одном уровне, т. е. все пути от корня до любого листа имеют одинаковую длину, равную высоте дерева;

4) каждая внутренняя вершина с t сыновьями содержит упорядоченный набор из $t - 1$ имен и t указателей на сыновей, т. е. ее можно представить в виде $(p_1, x_1, p_2, x_2, p_3, \ldots, p_{t-1}, x_{t-1}, p_t)$, где p_i – указатель на i-го сына $(1 \le i \le t)$, x_i – имя $(1 \le i \le t - 1)$, причем $x_1 < x_2 < \ldots < x_{t-1}$. Имена x_i являются границами, разделяющими имена в сыновьях. Поэтому все имена в вершине, на которую указывает p_i, больше x_{i-1} и меньше x_i.

На рис. 4.19 представлен пример сбалансированного максимально заполненного Б-дерева порядка 5, хранящего 24 имени (внешние вершины не показаны).

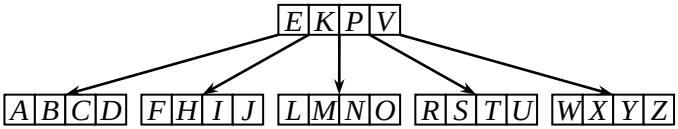

Рис. 4.19. Максимально заполненное Б-дерево порядка 5

Оценим сверху высоту h Б-дерева, хранящего n имен, т. е. на уровне h расположено $n + 1$ листьев (внешних вершин). Наибольшую высоту имеют Б-деревья с минимально заполненными вершинами, т. е. корень имеет двух сыновей, а все другие внутренние вершины – по $k = \lceil m/2 \rceil$ сыновей. Таким образом, на уровнях 1, 2, 3, …, $h – 1$ имеется 2, $2k$, $2k^2$, …, $2k^{h-2}$ внутренних вершин соответственно. Поскольку в корне хранится одно имя, а во всех остальных внутренних вершинах по $k – 1$ имен, получается неравенство

$$n \geq 1 + (k-1)\sum_{i=1}^{h-1} 2k^{i-1} = 1 + 2(k-1)\left(\frac{k^{h-1}-1}{k-1}\right) = 2k^{h-1} - 1,$$

из которого следует верхняя оценка высоты Б-дерева

$$h \leq 1 + \log_k \frac{n+1}{2} = 1 + \log_{\lceil m/2 \rceil} \frac{n+1}{2}.$$

Минимальную высоту имеют Б-деревья с максимально заполненными внутренними вершинами, т. е. все внутренние вершины имеют по m сыновей, причем в каждой вершине хранится по $m – 1$ имен. Поэтому нижняя оценка высоты Б-дерева будет $h \geq \log_m(n + 1)$.

Важным достоинством Б-деревьев является относительная простота выполнения поиска, включения и исключения.

Поиск. Поиск имени z в Б-дереве осуществляется путем сравнения z с именами $x_1, x_2, \ldots, x_{t-1}$, хранящимися в корне (поскольку эти имена упорядочены, можно применить бинарный поиск). Если z совпадает с каким-либо из этих имен, поиск завершается успешно. В противном случае, если $x_{i-1} < z < x_i$, процесс поиска продолжается рекурсивно в вершине, на которую указывает указатель p_i, расположенный между именами x_{i-1} и x_i. Безуспешный поиск завершается в листе (внешней вершине).

Включение. Включение имени *z* в Б-дерево выполняется в том случае, если поиск *z* завершается безуспешно. Безуспешный поиск завершается в листе, в котором могло бы находиться *z*. В отличие от деревьев бинарного поиска Б-деревьям запрещено расти в листьях – их вынуждают расти в корне. Поэтому новое имя продвигается вверх во внутреннюю вершину. Если эта вершина не полностью заполнена, то имя *z* вставляется в соответствующую позицию и процесс включения завершается. Если же в вершине нет места для записи *z* (вершина максимально заполнена), то она расщепляется на две вершины, а среднее имя продвигается вверх в вершину-отца. Процесс продвижения вверх продолжается до тех пор, пока не встретится не полностью заполненная вершина. В худшем случае процесс завершится расщеплением корня, что приведет к образованию нового корня и увеличению высоты Б-дерева на единицу.

В качестве примера рассмотрим включение имени *G* в Б-дерево на рис. 4.19. Безуспешный поиск определяет лист, в котором могло бы находиться имя *G* (рис. 4.20, *а*). Имя *G* продвигается в вершину с именами *F*, *H*, *I*, *J*. Поскольку она полностью заполнена, пять имен *F*, *G*, *H*, *I*, *J* разбиваются на две группы и среднее имя *H* продвигается в вершину-отца, чтобы оно служило разделителем между двумя группами (рис. 4.20, *б*). Вершина-отец также полностью заполнена, поэтому она расщепляется, среднее имя *K* продвигается вверх и становится новым корнем Б-дерева (рис. 4.20, *в*).

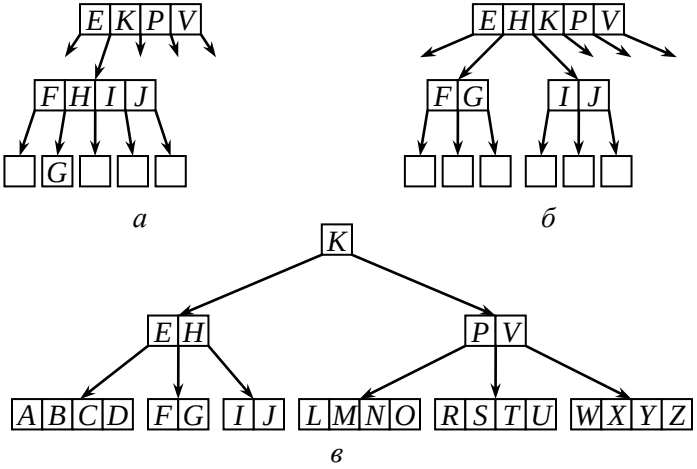

Рис. 4.20. Включение имени *G* в Б-дерево:
а – определение листа, где могло бы находиться имя *G*;
б – разбиение пяти имен *F*, *G*, *H*, *I*, *J* на две группы;
в – разбиение пяти имен *E*, *H*, *K*, *P*, *V* на две группы

Исключение. Если исключаемое имя z служит разделителем между группами имен, его прямо удалить из вершины нельзя. Его необходимо заменить именем y, либо непосредственно предшествующим, либо непосредственно следующим в естественном порядке. Очевидно, что эти имена хранятся во внутренних вершинах, находящихся на самом нижнем уровне Б-дерева. Если вершина, в которой хранится имя y, не является минимально заполненной, то процесс завершается удалением y из этой вершины. В противном случае исключение y приводит к тому, что вершина становится недостаточно заполненной, ей не хватает одного имени. Поэтому в эту вершину перемещается разделитель из вершины-отца, а на его место – имя из одного из смежных с недозаполненной вершиной братьев. Если брат имеет достаточный запас имен, никаких проблем не возникает и процесс завершается. Если же брат является минимально заполненным, т. е. содержит ровно $\lceil m/2 \rceil - 1$ имен, то этот минимально заполненный брат, недозаполненная вершина с $\lceil m/2 \rceil - 2$ именами и их разделитель в вершине-отце объединяются в одну вершину с $m - 1$ или $m - 2$ именами. Если в результате этих действий вершина-отец стала недозаполненной, процесс повторяется на следующем более высоком уровне. В худшем случае процесс исключения завершается в корне, тогда высота Б-дерева может уменьшиться на один уровень.

В качестве примера рассмотрим исключение имени *E* из Б-дерева на рис. 4.20, *в*. Поскольку *E* является разделителем между множествами {*A, B, C, D*} и {*F, G*}, можно на его место переместить непосредственно предшествующее имя *D*, которое будет разделителем между {*A, B, C*} и {*F, G*}, и завершить процесс. Рассмотрим более сложную ситуацию – переместим на место *E* непосредственно следующее имя *F*, которое служит разделителем между {*A, B, C, D*} и {*G*} (рис. 4.21, *а*). В вершине с единственным именем *G* не хватает одного имени, чтобы быть минимально заполненной. Левый брат этой вершины имеет достаточный запас имен, поэтому можно добавить разделитель *F* в вершину с именем *G* и переместить вверх *D* в качестве нового разделителя (рис. 4.21, *б*). Если нет смежного брата с достаточным запасом имен, то должен существовать смежный брат с минимальным заполнением. Для иллюстрации рассмотрим правого брата вершины с именем *G*, который является минимально заполненным (рис. 4.21, *а*). Выполняется объединение имен *G, H, I, J* в одну вершину (рис. 4.21, *в*). Теперь вершина с именем *F* стала недозаполненной. Объединение этой вершины с минимально заполненным правым братом с именами *P* и *V* и разделителем *K* приводит к Б-дереву, показанному на рис. 4.21, *г*.

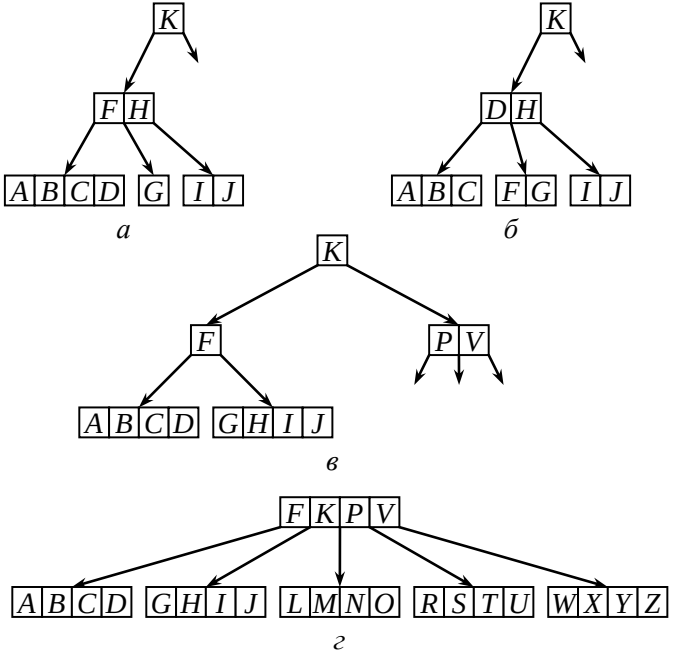

Рис. 4.21. Исключение имени *E* из Б-дерева:
а – замена исключаемого имени *E* на непосредственно следующее имя *F*;
б – перемещение *D* вверх вместо *F*, а *F* добавляется в вершину с *G*;
в – объединение имен *G*, *H*, *I*, *J* в одну вершину;
г – объединение имен *F*, *K*, *P*, *V* в одну вершину

При реализации Б-деревьев следует учитывать ряд факторов. Прежде всего, чтобы применить бинарный поиск имени в вершине, следует хранить в каждой внутренней вершине количество содержащихся в ней имен (поскольку эта величина не фиксирована) и поддерживать эту информацию при выполнении операций включения и исключения. Кроме того, чтобы легче выполнять расщепление вершины точно пополам (при включении нового имени), лучше использовать нечетное значение m.

Как известно, Б-деревья порядка m являются эффективной структурой данных для хранения больших таблиц во внешней памяти с возможностью прямого доступа. Время работы с файлами в основном определяется количеством операций обращения к внешней памяти (чтение, запись). Поэтому стремятся к чтению или записи возможно большей информации за одно обращение к внешней памяти. Таким образом, степень ветвления (величина m) Б-дерева обычно определяется размером сектора или некоторого блока (кластера), состоящего из определенного числа секторов. Типичная степень ветвления Б-деревьев (величина m) находится между 50 и 2000.

Очевидно, что число обращений к внешней памяти пропорционально высоте Б-дерева, т. е. $O(\log_m n)$. Увеличение степени ветвления резко сокращает высоту дерева и число обращений к внешней памяти.

Обычно файл представляет собой последовательность записей, каждая из которых может представлять собой достаточно сложную структуру, состоящую из одного и того же множества полей. Ключ, идентифицирующий запись, часто составляет небольшую часть записи. Если такие записи хранить целиком в вершине Б-дерева, то это приведет к слишком малой степени ветвления. Поэтому в вершинах дерева лучше хранить только ключи, а сами записи – в листьях (при этом в листьях не хранятся указатели на сыновей, поскольку их нет). В этом случае поиск требуемой записи предполагает прохождение пути от корня до листа, в котором содержится запись.

Частным случаем *m*-арных Б-деревьев, когда $m = 3$, являются так называемые *2-3-деревья* [2]. В 2-3-деревьях каждая внутренняя вершина может иметь двух или трех сыновей (соответственно может хранить одно или два имени). Этот вид сбалансированных деревьев является альтернативой АВЛ-деревьям и красно-черным деревьям для представления динамических таблиц.