Лабораторная работа №10
ИССЛЕДОВАНИЕ И ОЦЕНКА АЛГОРИТМОВ ПОИСКА НА ДЕРЕВЬЯХ

Краткая теория

Цель работы. Разработка программ, реализующих алгоритмы формирования и обхода двоичных и B+ деревьев, а также поиска элементов в них, и оценка их временной и пространственной сложности.

Двоичные и B+ деревья. Основные понятия и определения

Массивы относятся к линейным структурам данных. Они не позволяют реализовать эффективные алгоритмы для решения некоторых задач, например, поиска. Для таких задач более предпочтительным является древовидное представление данных.

Древовидная структура содержит множество узлов (nodes), происходящих от единственного начального, который называют корнем (root). На Рис. 5.1 корнем является узел *A*. Принято узел считать родителем (parent), указывающим на 0, 1 или более других узлов, называемых сыновьями (children). Например, узел B является родителем сыновей *E* и *F*. Дерево может представлять несколько поколений. Сыновья узла и сыновья их сыновей называются потомками (descendants), а родители и прародители – предками (ancestors) этого узла. Например, узлы *E, F, I, J* – потомки узла *B*. Каждый некорневой узел имеет только одного родителя, и каждый родитель имеет 0 или более сыновей. Узел, не имеющий детей (*E, G, H, I, J*), называется листом (leaf).

Каждый узел дерева является корнем поддерева (subtree), которое включает в себя этот узел и всех его потомков. Так, *F* есть корень

поддерева, содержащего узлы *F*, *I* и *J*. Узел *G* является корнем

поддерева без потомков. Таким образом, узел *A* есть корень поддерева, которое само оказывается деревом.

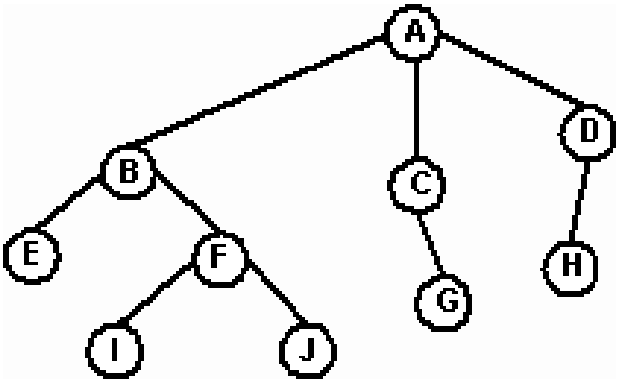

Рис. 5.1. Графическое изображение дерева

Каждый узел дерева является корнем поддерева (subtree), которое включает в себя этот узел и всех его потомков. Так, *F* есть корень поддерева, содержащего узлы *F, I* и *J*. Узел *G* является корнем поддерева без потомков. Таким образом, узел *A* есть корень поддерева, которое само оказывается деревом.

Переход от родительского узла к дочернему и другим потомкам осуществляется вдоль пути (path). Например, на рисунке 5.2 путь от корня *A* к узлу *F* проходит от *A* к *B* и от *B* к *F*. Тот факт, что каждый некорневой узел имеет единственного родителя, гарантирует, что существует единственный путь из любого узла к его потомкам. Длина пути от корня к этому узлу есть уровень узла. Уровень корня равен 0. Каждый сын корня является узлом 1-го уровня, следующее поколение – узлами 2-го уровня и т.д. Например, на рисунке 5.2 узел *F* является узлом 2-го уровня с длиной пути 2.

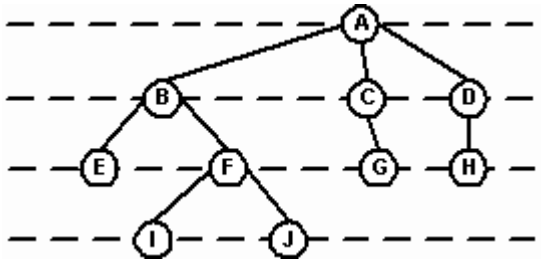

Рис. 5.2. Уровень узла и длина пути

Глубина (depth) дерева есть его максимальный уровень. Понятие глубины также может быть описано в терминах пути. Глубина дерева есть длина самого длинного пути от корня до узла. На рис. 5.2 глубина дерева равна 3.

В программировании широкое распространение получили так называемые ***двоичные*** (***бинарные*** - binary trees) деревья, которые имеют унифицированную структуру, обеспечивающую разнообразные алгоритмы эффективного доступа к элементам. Примеры таких деревьев представлены на рисунке 5.3.

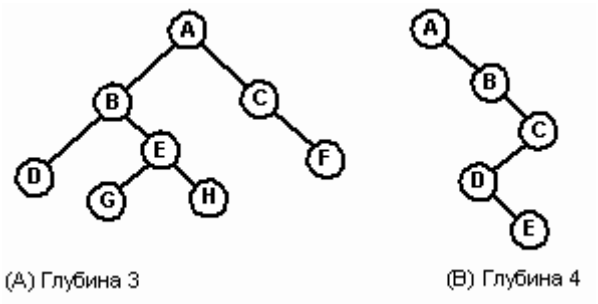

Рис. 5.3. Бинарные деревья

У каждого узла бинарного дерева может быть 0, 1 или 2 сына. Причем, узел слева называют левым сыном (left child), а справа – правым (right child). Эти наименования связаны с графическим представлением дерева. Двоичное дерево является рекурсивной структурой. Каждый узел – это корень своего собственного поддерева. У него есть сыновья, которые сами являются корнями деревьев, которые называются *левым* и *правым поддеревьями* соответственно.

Таким образом, бинарное дерево - это конечное множество элементов, которое либо пусто, либо содержит один элемент, называемый корнем. Остальные элементы делятся на два непересекающихся подмножества, каждое из которых, в свою очередь, является бинарным деревом. Эти подмножества называются правым и левым поддеревьями исходного дерева.

Рекурсивное определение двоичного дерева может быть сформулировано следующим образом.

Двоичное дерево - это такое множество узлов *B*, что

a) *B* является деревом, если множество узлов пусто (пустое дерево – тоже дерево);

b) *B* разбивается на три непересекающихся подмножества:

- $\{R\}$ корневой узел;
- $\{L_1, L_2, ..., L_m\}$ левое поддерево *R*;
- $\{R_1, R_2, ..., R_m\}$ правое поддерево *R*.

И определение, и процедуры обработки деревьев, как правило, рекурсивны.

Узлы дерева могут быть пронумерованы следующим образом.

Номер корня всегда равен 1, левый потомок получает номер 2, правый - номер 3. Левый потомок узла 2 должен получить номер 4, а правый - 5, левый потомок узла 3 получит номер 6, правый - 7 и т.д. Несуществующие узлы не нумеруются, что обычно не нарушает приведенного порядка. При такой системе нумерации в дереве каждый узел получает уникальный номер.

Полное бинарное дерево уровня n - это дерево, в котором каждый узел уровня n является листом. Причем, каждый узел уровня меньше n имеет непустые правое и левое поддеревья.

Почти полное бинарное дерево представляет собой бинарное дерево, для которого существует неотрицательное целое k такое, что:

1) каждый лист в дереве имеет уровень k или k+1;

2) если узел дерева имеет правого потомка уровня k+1, тогда все его левые потомки, являющиеся листами, также имеют уровень k+1.

В программировании структура бинарного дерева образуется элементами, структура которых приведена на рисунке 5.4. Узел содержит поле данных и два поля с указателями: левым (left) и правым (right), поскольку они указывают на соответствующие поддеревья. Значение NULL является признаком пустого дерева.

Корневой узел определяет входную точку дерева, а поля указателей – узлы следующего уровня. Листовой узел содержит NULL в полях правого и левого указателей. В java узел

представляется объектом класса с соответствующей структурой, например.

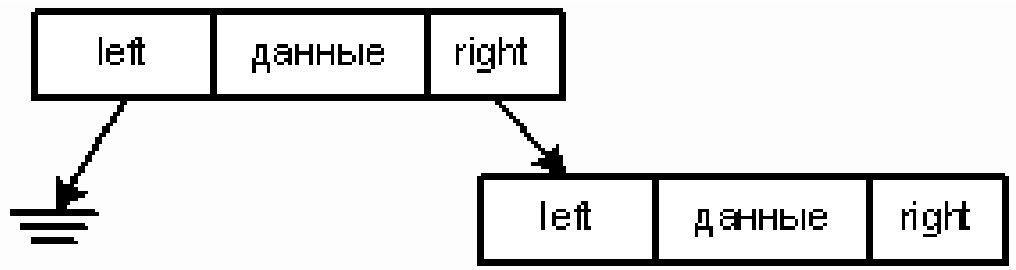

Рис. 5.4. Представление узлов дерева в программировании

```
class Tree{
        public Tree left; // левый указатель
        public Tree right;  // правый указатель
        public int key; // информационное поле целого типа
            }
```

Другой пример – дерево, узел которого хранит его обозначение в виде латинской буквы с номером и числовое поле. Корень обозначается буквой *A*, узлы 2-го уровня – *B*, 3-го – *C* и т.д. Нумерация узлов *k*-ого уровня выполняется от 1 до 2^k.

```
Class MyTree{
// Уровень узла
  private char Level;
// Номер узла
  private int Number;
// Поле узла
  private int key; }
```

При выводе из программы на экран удобно представлять дерево в виде строк текста с отступами. При этом начинают с корня, затем – его левые потомки и только потом – правые. Например, граф на рисунке

5.1 может быть представлен следующим образом.

A

B

E

F

I

J

C

G

D

H

B и B+ деревья

B-дерево является разновидностью дерева поиска. Такая структура была предложена Р. Бэйером и Е. МакКрейтом в 1970 году. Характерной для B-дерева является его малая глубина (высота), обычно равная 2 - 3. Основными свойствами таких деревьев являются: сбалансированность, ветвистость, отсортированность и логарифмическое время выполнения всех стандартных операций (поиск, вставка, удаление). Сбалансированность означает, что все листья находятся на одинаковом расстоянии от корня. В отличие от бинарных, B-деревья допускают большое число потомков для любого узла. Это свойство называется *ветвистостью*. Благодаря ему, B-деревья очень удобны для хранения крупных последовательных блоков данных, поэтому такая структура часто находит применение в базах данных и файловых системах.

Важным параметром B-дерева является его степень (порядок) *m*– максимальное число потомков для любого узла. Ключи располагаются между ссылками на потомков и, таким образом, ключей всегда на 1 меньше. В организации B-дерева можно выделить несколько правил:

1. Каждый узел содержит строго меньше *m* (порядок дерева) потомков.

2. Каждый узел содержит не менее $m/2$ потомков.

3. Корень может содержать меньше $m/2$ потомков.

4. У корневого узла есть хотя бы 2 потомка, если он не является листом.

5. Все листья находятся на одном уровне и содержат только данные (ключи).

Пример B-дерева приведен на рисунке 5.5.

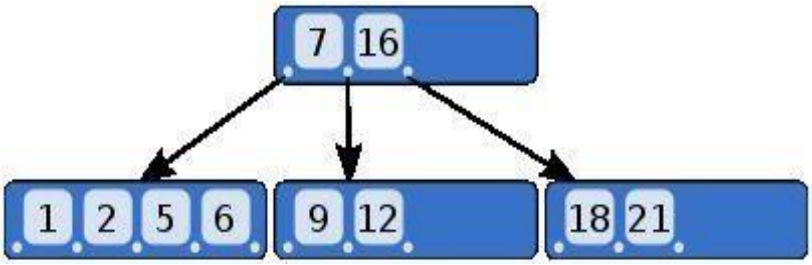

Рис. 5.5. Пример B+ дерева.
Стрелки идут к потомкам. В ключах записаны числа

Действия с бинарными и B+ деревьями

Узлы дерева обычно используются для хранения информации. Основными операциями над деревьями являются:

1) построение дерева,
2) создание узла,
3) включение узла в дерево,
4) удаление узла из дерева,
5) обход дерева,
6) поиск элементов (узлов).

Построение бинарного дерева

В программировании наибольшее распространение получили *упорядоченные двоичные деревья поиска*. Правило построения такого дерева следующее: элементы, у которых значение некоторого признака (ключа) меньше, чем у корня, всегда включаются слева от

некоторого поддерева, а элементы со значениями, большими, чем у

корня - справа. Этот принцип используется как при формировании двоичного дерева, так и при поиске в нем элементов.

Таким образом, при поиске элемента с некоторым значением ключа происходит спуск по дереву, начиная от корня. Выбор направления следующего шага – направо или налево (по значению искомого ключа) - происходит в каждом очередном узле на пути. При поиске элемента результатом будет либо узел с заданным ключом, либо лист с «нулевой» ссылкой (искомый элемент отсутствует на проделанном пути). Последняя ситуация может возникнуть, например, если целью поиска было включение очередного узла в дерево. При этом справа или слева (в зависимости от значения признака) к листу будет присоединен новый узел.

Рассмотрим пример формирования двоичного дерева. Предположим, что нужно сформировать двоичное дерево, узлы (элементы) которого имеют следующие значения признака: 20, 10, 35, 15, 17, 27, 24, 8, 30. В этом же порядке они и будут поступать для включения в двоичное дерево.

Первым узлом в дереве (корнем) станет узел со значением 20. В общем случае, поиск места подключения очередного элемента всегда начинается с корня. К корню слева подключается элемент 10, а справа - 35. Далее элемент 15 подключается справа к 10, проходя путь: корень 20 - налево - элемент 10 - направо - подключение, так как дальше пути нет. Процесс продолжается до тех пор, пока не будут включены в дерево все элементы. Результат представлен на рис. 5.6.

Отметим, что при создании двоичного дерева обычно считается, что ключи элементов не повторяются.

Узлы, у которых заполнены два адреса связи считаются полными, а с одним адресом – неполными. Элементы с двумя незаполненными адресами называются концевыми (листьями).

В *упорядоченном бинарном дереве* значение ключевого атрибута каждого узла должно быть больше, чем значение ключа у любого

элемента на его левой ветви, и не меньше, чем ключ на его правой ветви.

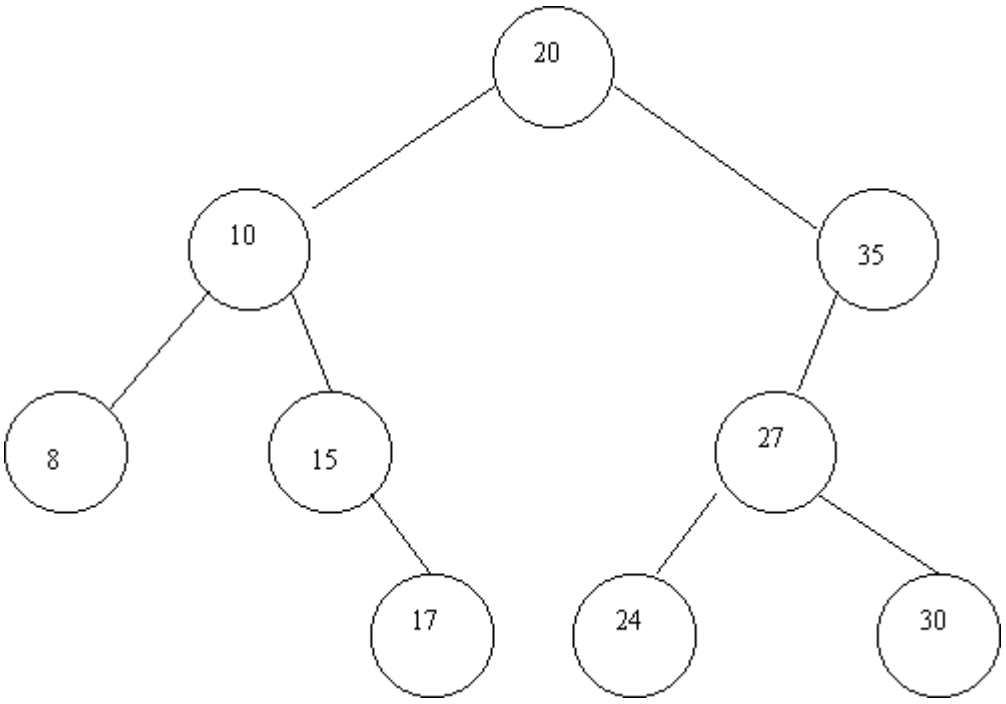

Рис. 5.6. Построение бинарного дерева.
Значения элементов дерева: 20, 10, 35, 15, 17, 27, 24, 8, 30

Алгоритм построения упорядоченного бинарного дерева может быть сформулирован так.

1.Первый элемент с ключом $p[1]$ становится корнем дерева.

2.Значение ключа второго узла $p[2]$ сравнивается с $p[1]$ (корня дерева). Если $p[2] < p[1]$, то второй элемент помещается на левой от корня ветви, в противном случае – на правой. В результате будет получено упорядоченное дерево из первых двух узлов.

3. Далее на каждом шаге создается упорядоченное дерево из первых i элементов. Выбор i-го узла производится следующим образом. Ключ $p[i]$ сравнивается с корневым значением и выполняется переход по левому адресу (если $p[1]>p[i]$), в противном случае (при $p[1] <= p\ [i]$) – по правому адресу. Далее ключ достигнутого узла также сравнивается с $p[i]$, и снова организуется

переход по левому и правому адресу и т.д. При нахождении незаполненного адреса связи ему присваивается ключ $p[i]$.

Пункт 3 повторяется до тех пор, пока не будут включены в дерево все элементы исходного массива.

Из рассмотренного алгоритма следует, что основным методом, используемым при построении двоичного дерева и решении других задач, является ***поиск***. В процессе поиска в упорядоченном бинарном дереве просматривается некоторый путь, начинающийся всегда в его корне. Искомое значение ключа q сравнивается со значением корня $p[1]$. Если $p[1]>q$, просмотр дерева продолжается по левой от корня ветви, иначе (если $p[1] <= q$) – по правой. Для произвольного узла с ключом $p[i]$ могут быть получены следующие результаты сравнения:

a) $p[i] = q$ – элемент, удовлетворяющий условию поиска найден, и поиск заканчивается по правой ветви.

b) $p[i] >q$ – производится переход к элементу, расположенному на левой ветви p(i).

c) $p[i] < q$ – производится переход к элементу, расположенному на правой ветви $p[i]$.

Поиск заканчивается, когда у какого-либо узла отсутствует адрес связи, необходимый для дальнейшего продолжения поиска.

Построение *B*+ дерева

При выполнении этой операции руководствуются основным правилом: изначально все узлы кроме последнего содержат один дополнительный элемент (то есть m+1 потомков в сумме), который будет использован для построения внутренних узлов.

Рассмотрим алгоритм на примере. Пусть количество потомков для каждого листового узла не более 4, то есть степень дерева $m = 5$. На вход подана последовательность чисел от 1 до 24. Разобьѐм их на блоки по приведенному правилу:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Из слоя листовых узлов строим следующий уровень путѐм вынесения туда лишних (в данном случае пятых) элементов на уровень выше. У четырех из пяти блоков такие элементы есть. Это – 5, 10, 15 и 20. Полученный новый уровень разбиваем в соответствии с тем же правилом.

Пусть для внутренних (нелистовых) узлов будет задано ограничение на количество потомков, равное 2 (по свойствам 2 и 3 узлов *B*-дерева). Результат вынесения элементов на верхний уровень приведен ниже.

5|10|15 **20**

1|2|3|4 **6|7|8|9** **11|12|13|14** **16|17|18|19** **21|22|23|24**

Продолжаем процесс, пока не получим корень.

15

5|10 **20**

1|2|3|4 **6|7|8|9** **11|12|13|14** **16|17|18|19** **21|22|23|24**

Осталось добавить ссылки и получится готовое B-дерево.

Включение узла в двоичное дерево

Добавление элемента (ДОБАВИТЬ)
Дано: дерево *T* и пара (*K,V* – ключ и вершина).
Задача: добавить пару (*K, V*) в дерево *T*.

Алгоритм включения узла в дерево

1. Если дерево пусто, заменить его на дерево с одним корневым узлом ((*K,V*), ПУСТО, ПУСТО) и остановиться

   Иначе сравнить K с ключом корневого узла X:

   1.1. Если *K*>=*X*, рекурсивно добавить (*K,V*) в правое поддерево *T*

Иначе (если $K<X$) рекурсивно добавить (K,V) в левое поддерево T.

Включение узла в *B* дерево

Введѐм определение: *дерево потомков узла* – это поддерево, состоящее из самого узла и его потомков.

Вначале определим функцию, которая добавляет элемент с ключом K к дереву потомков узла X. После выполнения функции во всех пройденных узлах, кроме, может быть, самого узла X, будет меньше $2t - 1$, но не меньше $t - 1$ ключей. Величина t определяется из формулы:

$$m = 2(t - 1).$$

Алгоритм добавления элемента к дереву потомков

1. Если X – не лист
   1.1. Определяем интервал, где должен находиться K. Пусть Y – соответствующий сын.
   1.2. Рекурсивно добавляем K к дереву потомков Y.
   1.3. Если узел Y полон, то есть содержит $2t - 1$ ключей, расщепляем его на два. Узел $Y1$ получает первые $t - 1$ ключей Y, и первые t его потомков, а узел $Y2$ – последние $t - 1$ ключей Y и последние t его потомков. Средний из ключей узла Y попадает в узел X, а указатель на Y в узле X заменяется указателями на узлы $Y1$ и $Y2$.
2. Если X – лист, просто добавляем туда ключ K.

Теперь сформулируем алгоритм добавления элемента с ключом K ко всему дереву. Буквой R обозначим корневой узел.

Алгоритм добавления элемента к B дереву

1. Добавим K к дереву потомков R.
2. Если в результате R содержит $2t - 1$ ключей, расщепляем его на два. Узел $R1$ получает первые $t - 1$ из ключей R и первые t его потомков, а $R2$ – последние $t - 1$ из ключей R и последние t его

потомков. Средний из ключей узла R попадает во вновь

созданный узел, который становится корневым. Узлы $R1$ и $R2$ становятся его потомками.

Зачастую добавление элемента приводит к полному или частичному сдвигу всех узлов дерева. Таким образом, эта операция является одной из наиболее трудоемких.

Удаление узла из бинарного дерева

Удаление узла (УДАЛИТЬ)
Дано: дерево T с корнем n и ключом K.
Задача: удалить из дерева T узел с ключом K (если такой есть).

Алгоритм удаления узла из дерева

1. Если дерево T пусто, остановиться

   Иначе сравнить K с ключом X корневого узла n.

   2.1. Если $K>X$, рекурсивно удалить K из правого поддерева T

   2.2. Иначе если $K<X$, рекурсивно удалить K из левого поддерева T.

   2.3. Если $K=X$, то необходимо рассмотреть два случая.

   2.3.1. Если одного из детей нет, то значения второго ребѐнка m ставим вместо соответствующих значений корневого узла, затирая его старые значения, и освобождаем память, занимаемую узлом m.

   2.3.2. Если оба потомка присутствуют, то

   2.3.2.1. найдѐм узел m, являющийся самым левым узлом правого поддерева;

   2.3.2.2. скопируем значения полей (ключ, значение) узла m в соответствующие поля узла n.

   2.3.2.3. у предка узла m заменим ссылку на узел m ссылкой на правого потомка узла m (который может быть равен ПУСТО).

2.3.2.4. освободим память, занимаемую узлом *m* (на него теперь никто не указывает, а его данные были перенесены в узел *n*).

Удаление узла из *B* дерева

При удалении узла (ключа) могут возникнуть две проблемы:

a) Удалѐнный элемент является разделителем потомков внутри какого-либо нелистового узла;

b) После удаления в узле может остаться менее допустимого m/2 числа потомков.

Для того чтобы устранить эти проблемы, требуется перебалансировка дерева. Она может быть выполнена после удаления узла (ключа) или до него. Рассмотрим первый вариант.

Алгоритм удаления узла из B дерева

1. Если корень одновременно является листом (в дереве всего один узел), удаляем ключ из этого узла.
2. В противном случае находим узел, содержащий ключ, запоминая путь к нему. Пусть этот узел – *X*.
3. Если *X* – лист, удаляем оттуда ключ.
   - 3.1. Если в узле *X* осталось не меньше $t - 1$ ключей, останавливаемся. Иначе проверяем количество ключей в следующем, а потом в предыдущем узле.
   - 3.2. Если следующий узел есть, и в нѐм не менее *t* ключей, добавляем в *X* ключ-разделитель между ним и следующим узлом, а на его место ставим первый ключ следующего узла, после чего останавливаемся.
   - 3.3. В противном случае, если есть предыдущий узел, и в нѐм не менее *t* ключей, добавляем в *X* ключ-разделитель между ним и предыдущим узлом, а на его место ставим последний ключ предыдущего узла, после чего останавливаемся.

3.4. Наконец, если и с предыдущим ключом не получилось, объединяем узел X со следующим или предыдущим узлом, и в объединѐнный узел перемещаем ключ, разделяющий два узла. При этом в родительском узле может остаться только $t - 2$ ключа. Тогда, если это не корень, выполняем аналогичную процедуру с ним.

3.5. Если в результате дошли до корня, и в нѐм осталось от 1 до $t - 1$ ключей, остановка, потому что корень может иметь и меньше $t - 1$ ключей.

3.6. Если в корне не осталось ни одного ключа, исключаем корневой узел, а его единственный потомок делаем новым корнем дерева.

4. Если X – не лист, а K – его i-й ключ, удаляем самый правый ключ из поддерева потомков i-го сына X, или, наоборот, самый левый ключ из поддерева потомков $(i + 1)$-го сына X.

Обход дерева

Обход дерева может выполняться одним из следующих способов:

1) «сверху вниз» или *префиксный обход* (функция_прямого_вызова) — обойти всѐ дерево по порядку (вершина, левое поддерево, правое поддерево);

2) «снизу вверх» или *постфиксный обход* (функция_обратного_вызова) — обойти всѐ дерево по порядку (левое поддерево, правое поддерево, вершина);

3) «симметричный обход» или *инфиксный обход* (функция_симметричного_вызова) — обойти всѐ дерево по порядку (левое поддерево, вершина, правое поддерево).

Эти процедуры, судя по названию и описанию, имеют рекурсивный характер. Их алгоритмы можно описать следующим образом.

ИНФИКСНЫЙ_ОБХОД позволяет обойти все узлы дерева в порядке возрастания ключей и применить к каждому узлу заданную

пользователем функцию обратного вызова. Эта функция обычно работает только с парой (*K,V*), хранящейся в узле. Операция ИНФИКСНЫЙ_ОБХОД реализуется рекурсивным образом: сначала она запускает себя для левого поддерева, потом выполняет функцию обратного вызова для корня, а затем вызывает себя для правого поддерева.

Инфиксный (симметричный) обход дерева

Дано: дерево *T* и функция *f*
Задача: применить *f* ко всем узлам дерева *T* в порядке возрастания ключей

Алгоритм обхода дерева

1. Если дерево пусто, остановиться.
   1.1. Иначе
      1.1.1. Рекурсивно обойти правое поддерево *T*.
      1.1.2. Применить функцию *f* к корневому узлу.
      1.1.3. Рекурсивно обойти левое поддерево *T*.

В простейшем случае, функция f может выводить значение пары (*K,V*). При использовании операции ИНФИКСНЫЙ_ОБХОД будут выведены все пары в порядке возрастания ключей. Если же использовать ПРЕФИКСНЫЙ_ОБХОД, то пары будут выведены в порядке, соответствующем описанию дерева.

Поиск элемента в дереве

В общем случае для выполнения этой операции могут использоваться различные условия. В зависимости от способа их задания и организации поиска различают большое количество видов таких операций. Наибольшее распространение получили следующие виды поиска по простому условию:

d) совпадению,

е) близости снизу

f) близости сверху.

Первый вид предполагает нахождение элемента с заданным значением ключа *K*. Если такого ключа нет, то операция закончилась безуспешно.

При *поиске по близости* операция всегда заканчивается успешно. Если он выполняется снизу, то результатом будет элемент, ключ которого является ближайшим меньшим искомого. При поиске по близости сверху результат представляет собой элемент с ключом, ближайшим большим искомого.

Поиск элемента двоичного дерева

Рассмотрим алгоритм поиска по совпадению.

Поиск элемента (ИСКАТЬ)

Дано: дерево *T* и ключ *K*.

Задача: проверить, есть ли узел с ключом *K* в дереве *T*, и если да, то вернуть ссылку на этот узел.

Алгоритм поиска элемента двоичного дерева по ключу

1. Если дерево пусто, сообщить, что узел не найден, и остановиться.
2. Иначе сравнить *K* со значением ключа корневого узла *X*.
   - 2.1. Если *K*=*X*, выдать ссылку на этот узел и остановиться
     - 2.1.1. Иначе, если *K*>*X*, рекурсивно искать ключ *K* в правом поддереве *T*.
     - 2.1.2. Иначе (если *K*<*X*) рекурсивно искать ключ *K* в левом поддереве *T*.

Поиск элемента *B* дерева

Благодаря структуре этого дерева и постоянно поддерживаемому балансу, поиск осуществляется за логарифмическое время от количества элементов. Сам алгоритм описывается следующим образом. Пусть ищется значение *X* в заданном *B*-дереве.

Алгоритм поиска элемента B дерева

1. Начинаем от корня
2. Если *X* –один из ключей текущего узла, то СТОП, ключ найден.
   Иначе находим пару соседних ключей *Y, Z,* таких что *X* лежит в интервале от *Y* до *Z.*
3. Переходим к потомку между *Y* и *Z.*
4. Возврат к шагу 2.
5. Если дошли до листа и *X* нет среди ключей, то СТОП, ключ и элемент не найден.

Порядок выполнения лабораторной работы

Подготовка к работе

Подготовка предполагает выполнение следующих этапов.

1. Знакомство со всеми разделами руководства.

2. Разработка алгоритма и программы построения двоичного дерева поиска.

3. Разработка алгоритма и программы построения B+ дерева.

Последовательность выполнения лабораторной работы

1. Разработать алгоритм и программу построения двоичного дерева поиска:
   a) Получить у преподавателя задание на величину диапазона значений ключей и размер их массива *n*.
   b) Сформировать массив ключей, значения которых задаются с помощью датчика случайных чисел.
   c) По заданию преподавателя упорядочить значения ключей по возрастанию (убыванию).
   d) Приняв за корень элемент с ключом из середины массива (ключ с номером *n* / 2), построить двоичное дерево поиска.

Информационные поля узлов дерева можно оставить пустыми.

e) Вывести значения ключей по уровням дерева.

2. Разработать алгоритм и программу построения B+ дерева:
   a) Получить у преподавателя задание на количество ярусов, величину диапазона значений ключей и размер их массива *n* на каждом ярусе.
   b) Сформировать массив ключей, значения которых задаются с помощью датчика случайных чисел.
   c) По заданию преподавателя упорядочить значения ключей по возрастанию (убыванию).
   d) Приняв за корень элемент с ключом из середины массива (ключ с номером *n* / 2), построить B+ дерево. Информационные поля узлов дерева можно оставить пустыми.
   e) Вывести значения ключей по уровням дерева.
3. Разработать алгоритм и программу поиска заданного преподавателем типа (по совпадению, интервалу или близости) на двоичном и B+ дереве.
4. Оценить сложность разработанных алгоритмов.

Содержание отчета о выполненной работе

Отчет должен содержать следующее.

- Название и цель работы, а также исходные данные.
- Пошаговые алгоритмы построения двоичного и B+ дерева, а также тексты их программ.
- Распечатки результатов, полученных с помощью ЭВМ.
- Пошаговые алгоритмы поиска на двоичном и B+ дереве, а также тексты их программ.
- Выводы о сложности разработанных алгоритмов.

Контрольные вопросы

1. Для каких задач предпочтительными являются древовидные структуры данных?
2. Какие типы деревьев используются в программировании?
3. Что представляет собой двоичное дерево?
4. Чем отличается упорядоченное двоичное дерево от обычного?
5. Что представляет собой узел дерева?
6. Что такое поддерево?
7. Как задаются связи между узлами дерева?
8. Какие характеристики имеют деревья?
9. Чем отличается B+ дерево от двоичного?
10. Какие основные операции выполняются над деревьями?
11. Как строится двоичное дерево?
12. Как может быть выполнен обход узлов дерева?
13. Какие типы операций поиска существуют?
14. Чем отличается поиск по совпадению от поиска по близости?
15. Как выполняется поиск на двоичном дереве?
16. Как выполняется поиск на B+ дереве?

Индивидуальные задания

Вариант 1

1. Построить двоичное дерево, содержащее $n = 15$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 100.
2. Построить B+ дерево, содержащее $n = 15$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 100.
3. Обеспечить обход деревьев «сверху вниз».
4. Выполнить поиск значения ключа по совпадению.

Вариант 2

1. Построить двоичное дерево, содержащее $n = 16$ узлов. Значения

ключей в узлах задавать с помощью датчика случайных чисел с диапазоном *D* от 0 до 150.

2. Построить B+ дерево, содержащее $n = 16$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном *D* от 0 до 150.

3. Обеспечить обход деревьев «снизу вверх».

4. Выполнить поиск значения ключа по близости снизу.

Вариант 3

1. Построить двоичное дерево, содержащее $n = 12$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном *D* от 0 до 80.

2. Построить B+ дерево, содержащее $n = 12$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном *D* от 0 до 80.

3. Обеспечить симметричный обход деревьев.
4. Выполнить поиск значения ключа по близости сверху.

Вариант 4

1. Построить двоичное дерево, содержащее $n = 20$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 180.
2. Построить B+ дерево, содержащее $n = 20$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 180.
3. Обеспечить обход деревьев «сверху вниз».
4. Выполнить поиск значения ключа по совпадению.

Вариант 5

1. Построить двоичное дерево, содержащее $n = 20$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 180.
2. Построить B+ дерево, содержащее $n = 20$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 180.
3. Обеспечить обход деревьев «снизу вверх».
4. Выполнить поиск значения ключа по близости снизу.

Вариант 6

1. Построить двоичное дерево, содержащее $n = 20$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 200.
2. Построить B+ дерево, содержащее $n = 20$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 200.
3. Обеспечить симметричный обход деревьев.
4. Выполнить поиск значения ключа по близости сверху.

Вариант 7

1. Построить двоичное дерево, содержащее $n = 18$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
2. Построить B+ дерево, содержащее $n = 18$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
3. Обеспечить обход деревьев «сверху вниз».
4. Выполнить поиск значения ключа по совпадению.

Вариант 8

1. Построить двоичное дерево, содержащее $n = 14$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 120.
2. Построить B+ дерево, содержащее $n = 14$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 120.
3. Обеспечить обход деревьев «снизу вверх».
4. Выполнить поиск значения ключа по близости снизу.

Вариант 9

1. Построить двоичное дерево, содержащее $n = 15$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 140.
2. Построить B+ дерево, содержащее $n = 15$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 140.
3. Обеспечить симметричный обход деревьев.
4. Выполнить поиск значения ключа по близости сверху.

Вариант 10

1. Построить двоичное дерево, содержащее $n = 20$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 200.
2. Построить B+ дерево, содержащее $n = 20$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 200.
3. Обеспечить обход деревьев «сверху вниз».
4. Выполнить поиск значения ключа по совпадению.

Вариант 11

1. Построить двоичное дерево, содержащее $n = 18$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
2. Построить B+ дерево, содержащее $n = 18$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
3. Обеспечить обход деревьев «снизу вверх».
4. Выполнить поиск значения ключа по близости снизу.

Вариант 12

1. Построить двоичное дерево, содержащее $n = 16$ узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 120.
2. Построить B+ дерево, содержащее $n = 16$ узлов и имеющее степень $m = 5$. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 120.
3. Обеспечить симметричный обход деревьев.
4. Выполнить поиск значения ключа по близости сверху.

Вариант 13

1. Построить двоичное дерево, содержащее n = 12 узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 100.
2. Построить B+ дерево, содержащее n = 12 узлов и имеющее степень m = 5. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 100.
3. Обеспечить обход деревьев «сверху вниз».
4. Выполнить поиск значения ключа по совпадению.

Вариант 14

1. Построить двоичное дерево, содержащее n = 18 узлов. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
2. Построить B+ дерево, содержащее n = 18 узлов и имеющее степень m = 5. Значения ключей в узлах задавать с помощью датчика случайных чисел с диапазоном D от 0 до 160.
3. Обеспечить обход деревьев «снизу вверх».
4. Выполнить поиск значения ключа по близости снизу.