СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Методическое пособие для заочного обучения

Часть 2
Древовидные структуры данных

Новосибирск 2006

ЦЕЛЬ КУРСА

В результате изучения курса студент должен

знать о методах построения деревьев для хранения и поиска информации, а также о трудоемкости рассматриваемых алгоритмов;

уметь разрабатывать программы для решения задач построения дерева и поиска информации;

иметь навыки применения методов построения деревьев для организации хранения информации.

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

1. Двоичные деревья.

Определение двоичного дерева. Высота дерева. Вычисление основных характеристик двоичного дерева. Обходы дерева.

2. Деревья поиска.

Определение двоичного дерева поиска. Алгоритм поиска вершины с заданным ключом в дереве поиска. Случайное дерево поиска (СДП). Идеально сбалансированное дерево поиска. (ИСДП) Алгоритмы построения ИСДП и СДП.

3. АВЛ-дерево.

Определение АВЛ-дерева. Повороты. Алгоритм включения вершины в АВЛ-дерево. Алгоритм удаления вершины из АВЛ-дерева. Трудоемкость алгоритмов для АВЛ-дерева.

4. Б-дерево

Определение Б-дерева порядка m. Алгоритмы включения и исключения вершины Б-дерева порядка m. Алгоритм поиска в Б-дереве. Трудоемкость построения Б-дерева порядка m и поиска в нем. Определение двоичного Б-дерева (ДБД). Построение ДБД. Трудоемкость построения ДБД.

5. Дерево оптимального поиска.

Определение дерева оптимального поиска (ДОП). Средневзвешенная высота ДОП. Точный алгоритм построения ДОП. Трудоемкость алгоритма. Приближенные алгоритмы построения ДОП.

1. ДВОИЧНЫЕ ДЕРЕВЬЯ

1.1 Основные определения и понятия

Определим двоичное дерево следующим образом :

1. Отдельная вершина V является двоичным деревом.
2. Двоичное дерево – это вершина V, соединенная с (возможно пустыми) левым (T_L) и правым (T_R) двоичными деревьями.

Пример двоичного дерева приведен на следующем рисунке. Вершины дерева обозначаются кружочками, связи между вершинами стрелками.

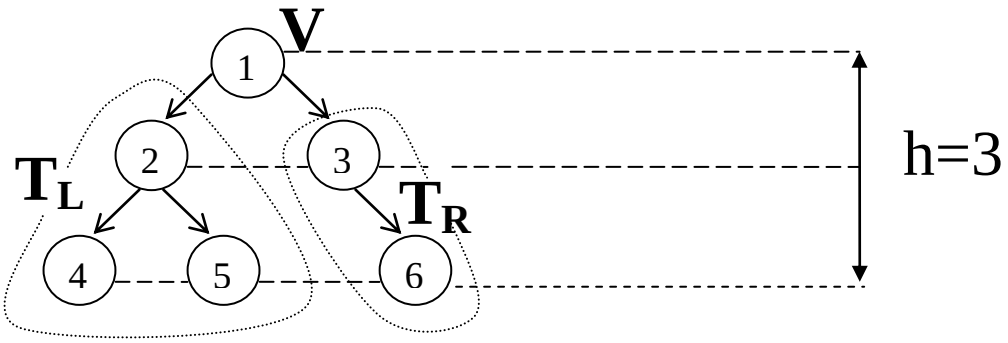


Рисунок 11 Пример двоичного дерева

Каждая вершина дерева может содержать какую-либо информацию. Выделенная вершина дерева называется *корнем.* Концевые вершины дерева, не имеющие поддеревьев, называются *листьями***.** Дуги ориентированы по направлению от корня к листьям. Путь от корня к листу называется *ветвью***.** Под *длиной ветви* будем понимать число входящих в неё вершин. *Высота дерева h* определяется как число вершин в самой длинной ветви дерева. *Размер дерева* – число входящих в него вершин. Средней высотой дерева называется усредненная по количеству вершин в дереве сумма длин путей от корня к каждой вершине.

Приведем некоторые свойства двоичных деревьев.

Свойство 1. *Максимальное число вершин в двоичном дереве высоты h равно*

$$n_{max}(h) = 2^h - 1$$

Свойство 2. *Минимально возможная высота двоичного дерева с n вершинами равна* $h_{min}(n) = \lceil \log(n+1) \rceil$

1.2 Различные обходы двоичных деревьев

При разработке алгоритмов для работы с деревьями будем считать, что каждая вершина содержит некоторые данные и указатели на вершины слева и справа. Поэтому вершина дерева это переменная специального типа. В качестве заголовка для дерева используем переменную Root, указывающую на корень.

```
TYPE    pVertex = ^tVertex;
        tVertex =record
                Data: integer;
                Left: pVertex;
```

```
                Right: pVertex;
                end;
VAR     Root: pVertex;
```

При решении многих задач, связанных с деревьями, часто возникает необходимость перебрать все вершины дерева или совершить *обход дерева*, чтобы выполнить некоторые действия с каждой вершиной дерева.

Существуют три основных порядка обхода дерева:

1. Сверху вниз (↓): корень, левое поддерево, правое поддерево.
2. Слева направо (→): левое поддерево, корень, правое поддерево.
3. Снизу вверх (↑): левое поддерево, правое поддерево, корень.

Пример. Совершить обход слева направо для двоичного дерева, изображенного на рисунке 28.

Результат обхода: 4 2 5 1 3 6

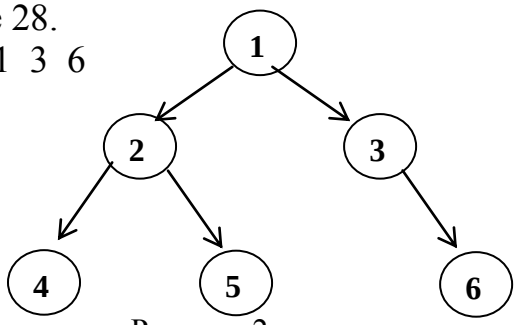


Рисунок 2

Обходы легко программируются с помощью рекурсивных процедур.

Алгоритм на псевдокоде

Обход слева направо(p: pVertex)

```
IF (p ≠ NIL)
        Обход слева направо (p→Left)
        Печать (p→Data)
        Обход слева направо (p→Right)
FI
```

1.3 Вычисление основных характеристик дерева

Приведем алгоритмы для вычисления различных характеристик двоичных деревьев.

1) Определение размера дерева

Алгоритм на псевдокоде

Размер(p: pVertex)

```
IF (p = NIL) Размер := 0
ELSE Размер := 1 + Размер (p→Left) + Размер (p→Right)
FI
```

2) Определение высоты дерева

Алгоритм на псевдокоде

Высота(p: pVertex)

IF (p = NIL) Высота := 0
ELSE Высота := 1+ max(Высота (p→Left), Высота(p→Right))
FI

3) Определение средней высоты дерева

Для определения средней высоты дерева понадобится функция вычисления суммы длин путей от корня до каждой вершины на L-том уровне.

Алгоритм на псевдокоде

СДП (p: pVertex; L: уровень вершины)

IF (p = NIL) СДП:= 0
ELSE СДП:= L + СДП(p → Left, L+1) + СДП(p → Right, L+1)
FI

Тогда средняя высота вычисляется следующим образом

$$h_{ср} := \text{СДП(Root, 1)}/\ \text{Размер(Root)}$$

4) Определение контрольной суммы для дерева

Алгоритм на псевдокоде

Сумма (p: pVertex)

IF (p = NIL) Сумма := 0
ELSE Сумма:= p→Data + Сумма(p→Left) + Сумма(p→Right)
FI

1.4 Контрольные вопросы

1. Дайте определение двоичного дерева.
2. Что такое высота дерева? Средняя высота?
3. Что такое обход дерева?
4. Назовите основные обходы дерева.

2. ДЕРЕВЬЯ ПОИСКА

2.1 Поиск в дереве

Двоичные деревья часто употребляются для представления множества данных, среди которых идет поиск элементов по уникальному ключу. Будем считать, что

1) часть данных, хранящихся в каждой вершине дерева, является ключом для поиска.
2) Для всех ключей определены операции сравнения <, >, =.
3) В дереве нет элементов с одинаковыми ключами.

Двоичное дерево называется *деревом поиска*, если ключ в каждой его вершине больше ключа любой вершины в левом поддереве и меньше ключа

любой вершины в правом поддереве. Пример такого дерева приведен на рисунке.

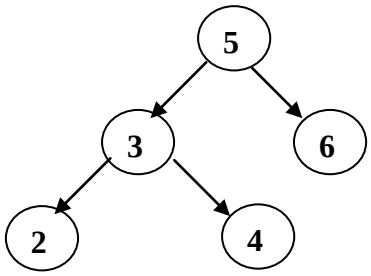


Рисунок 3 Дерево поиска

Чтобы определить является ли двоичное дерево деревом поиска приведем описание на псевдокоде следующей функции. Функция возвращает значение ИСТИНА в случае, если дерево является деревом поиска, и значение ЛОЖЬ в противном случае.

Алгоритм на псевдокоде

Дерево поиска(p: pVertex)

```
Дерево поиска = ИСТИНА
IF (p ≠ NIL и ((p→Left ≠ NIL и (p→Data ≤ p→Left→Data или
                                не Дерево поиска (p→Left)))
    или (p→Right ≠ NIL и (p→Data ≥ p →Right→Data или
                                не Дерево поиска (p→Right)))))
  Дерево поиска := ЛОЖЬ
FI
```

В основном деревья поиска используются для организации быстрого и удобного поиска элемента с заданным ключом во множестве данных, которое динамически изменяется. Приведенная ниже процедура поиска элемента в дереве поиска возвращает указатель на вершину с заданным ключом, в противном случае возвращаемое значение равно пустому указателю.

Алгоритм на псевдокоде

Поиск вершины с ключом X

```
p: = Root
DO (p ≠ NIL)
      IF (p→Data < x) p: = p→Right
      ELSEIF (p→Data > x) p: = p→Left
      ELSE OD   { p→Data = x }
OD
IF (p ≠ NIL) <вершина найдена>
ELSE <вершина не найдена>
```

Нетрудно видеть, что максимальное количество сравнений при поиске равно $C_{max} = 2h$, где h высота дерева.

2.2 Идеально сбалансированное дерево поиска

Двоичное дерево называется *идеально сбалансированным* (ИСД), если для каждой его вершины размеры левого и правого поддеревьев отличаются

не более чем на 1.

На рисунке приведены примеры деревьев, одно из которых является идеально сбалансированным, а другое – нет.

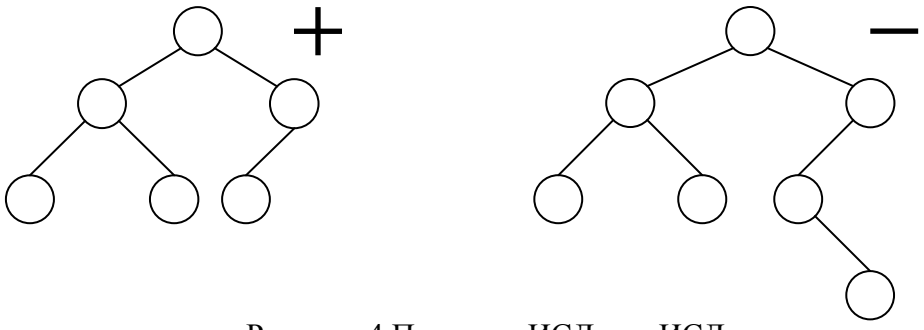


Рисунок 4 Примеры ИСД и неИСД

Отметим некоторые свойства идеально сбалансированного дерева.

Свойство 1. *Высота ИСД с n вершинами минимальна и равна*

$$h_{ИСД(n)} = h_{min}(n) = \lceil \log(n+1) \rceil.$$

Свойство 2. *Если дерево идеально сбалансировано, то все его поддеревья также идеально сбалансированы.*

Задача построения идеально сбалансированного дерева поиска из элементов массива $A = (a_1, a_2, \ldots, a_n)$ решается в два этапа:

1. Сортировка массива A.
2. В качестве корня дерева возьмем средний элемент отсортированного массива, из меньших элементов массива строим левое поддерево, из больших – правое поддерево. Далее процесс построения продолжается до тех пор, пока левое или правое поддерево не станет пустым.

Пример. Построить ИСДП из элементов массива A. Пусть n = 16, а элементы массива это числа в 16-ричной системе счисления.

A: B 9 2 4 1 7 E F A D C 3 5 8 6

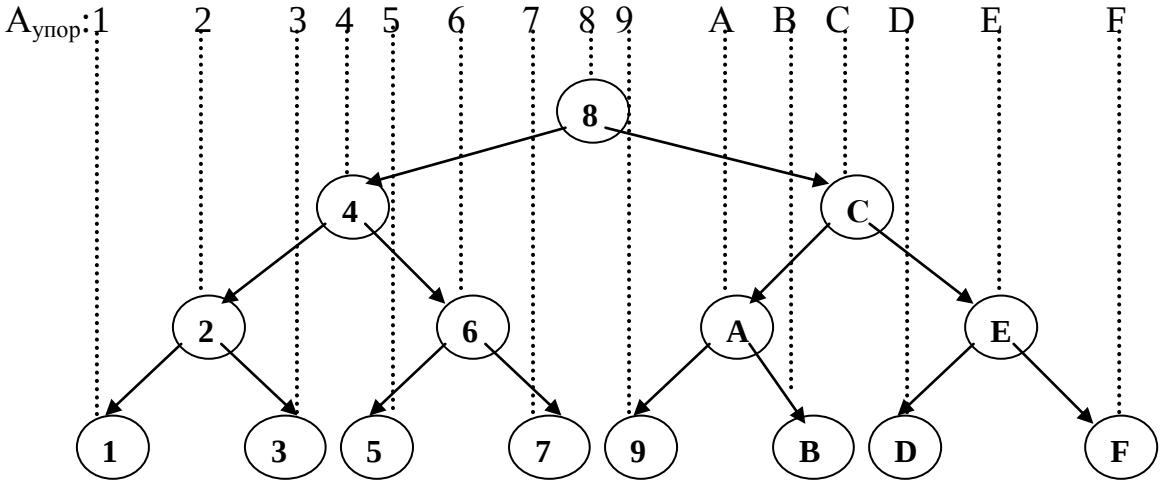


Рисунок 5 Построение ИСДП

Приведем на псевдокоде алгоритм построения ИСДП. Функция ИСДП (L, R) возвращает указатель на построенное дерево, где L, R – левая и правая границы той части массива, из элементов которой строится дерево.

Алгоритм на псевдокоде

ИСДП (L, R)

```
IF (L > R) ИСДП:=NIL
ELSE m : = [(L+R) / 2]
        <Выделяем память для p>
        p → Data : = A[m]
        p → Left : = ИСДП ( L, m-1)
        p → Right : = ИСДП ( m+1, R)
        ИСДП:= p
FI
```

Для идеально сбалансированного дерева $C_{max} = 2\lceil \log(n+1) \rceil$. Если считать, что поиск любой вершины происходит одинаково часто, то ИСДП обеспечивает минимальное среднее время поиска. К существенным недостаткам ИСДП можно отнести то, что при добавлении нового элемента к множеству данных необходимо строить заново ИСДП.

2.3 Контрольные вопросы

1. Дайте определение двоичного дерева поиска.
2. Что такое идеально сбалансированное дерево поиска?
3. Какова высота ИСДП?
4. Каким образом строится ИДСП?

3. СЛУЧАЙНОЕ ДЕРЕВО ПОИСКА

3.1 Определение случайного дерева поиска

При решении многих типов задач объем данных заранее неизвестен, но необходима такая структура данных, для которой достаточно быстро выполняются операции поиска, добавления и удаления вершин. Одно из решений этой проблемы построение *случайного дерева поиска* (СДП). При построении СДП данные поступают последовательно в произвольном порядке и добавление нового элемента происходит в уже имеющееся дерево.
Пример случайного дерева поиска для последовательности D:

B 9 2 4 1 7 E F A D C 3 5 8 6

приведен на рисунке 6.

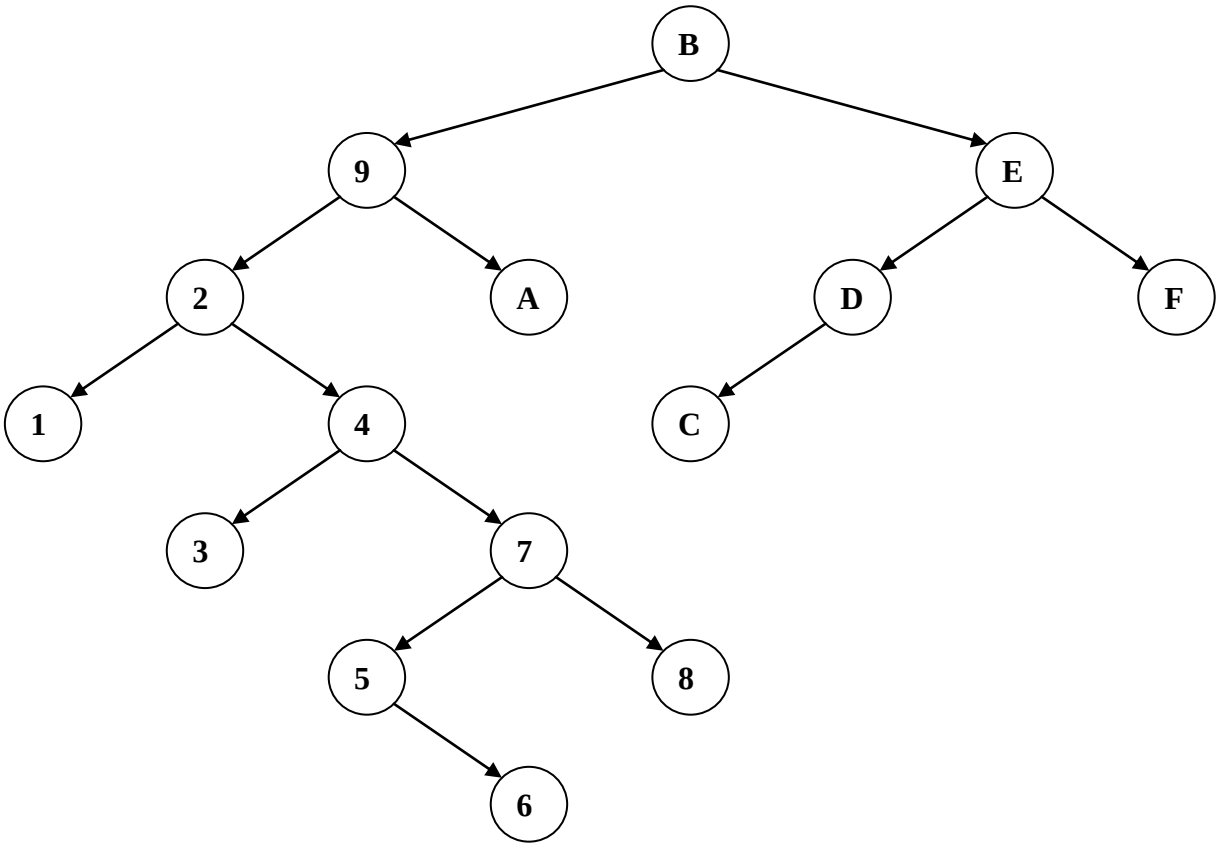


Рисунок 6 Случайное дерево поиска

Это дерево не является ИСДП. В худшем случае СДП может выродиться в список.

Пример. 1) D: 1 2 3 4 5 2) D: 5 1 2 4 3

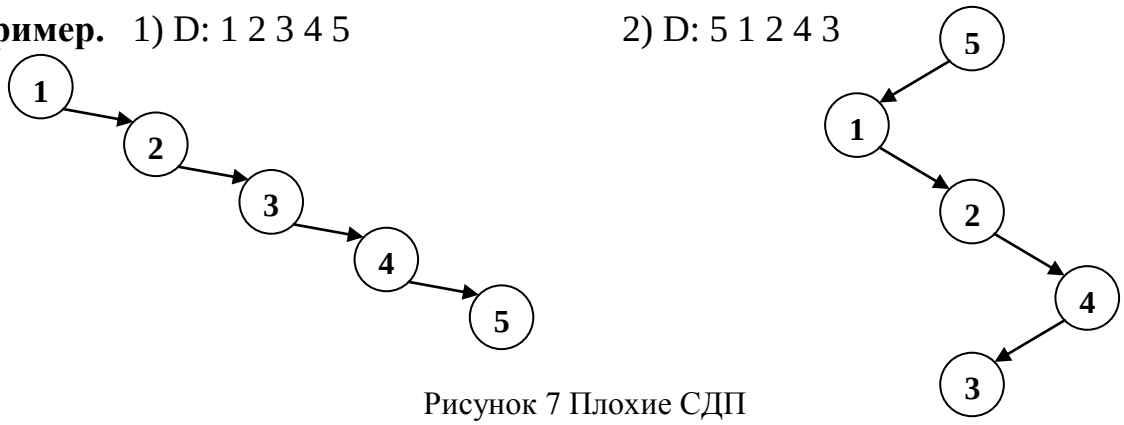


Рисунок 7 Плохие СДП

Таким образом, средняя высота случайного дерева поиска может изменяться в пределах $\log n < h_{ср\ сд} < n$ при больших n. В [1] показано, что

$\lim_{n \to \infty} (h_{ср\ сд} / h_{ср\ исдп}) = 2\ln 2 = 1{,}386...$ и $h_{ср\ сд} = O(\log n)$ при $n \to \infty$.

3.2 Добавление вершины в дерево

Алгоритм добавления вершины в СДП заключается в следующем. Если дерево пустое, то создается корневая вершина, в которую записываются данные. В противном случае вершина добавляется к левому или правому поддереву в зависимости от результата сравнения с данными в текущей вершине.

При создании новой вершины нужно будет изменить значение указателя на нее, поэтому нам понадобиться указатель на указатель (двойная косвенность). На языках высокого уровня двойная косвенность обычно может быть реализована с помощью ссылки на указатель.

```
type pVertex = ^tVertex;
var p: ^pVertex;
```

Алгоритм на псевдокоде

*Добавление в СДП (D, *Root)*

Описание переменных: Root – указатель на корень дерева.
D – данные, которые необходимо добавить.
P – указатель на указатель на вершину дерева.

```
p := @Root
DO (*p ≠ NIL)
        IF (D < (*p)→Data) p := @((*p)→Left)
        ELSEIF (D > (*p)→Data) p := @((*p)→Right)
        ELSE OD {данные с ключом D уже есть в дереве}
OD
IF (*p = NIL)
        new(*p), (*p)→Data := D,
        (*p)→Right := NIL, (*p)→Left := NIL
FI
```

Пример. Построение дерева для последовательности B 9

1). D = B

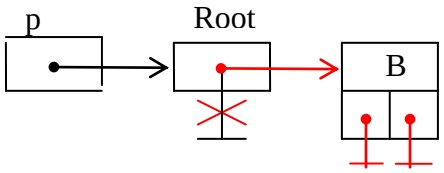


Рисунок 8 Добавление вершины B

2). D = 9

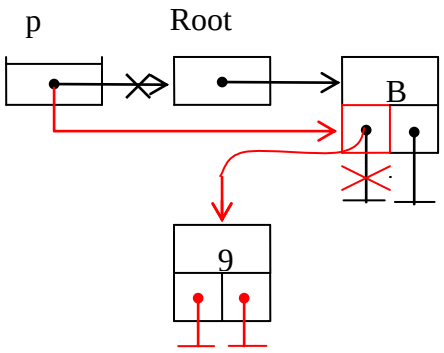


Рисунок 9 Добавление вершины 9

Нетрудно видеть, что трудоемкость добавления вершины в случайное дерево поиска сравнима по порядку величины с трудоемкостью поиска в двоичном дереве, т.е. $C_{ср} = O(\log n)$, $n \to \infty$.

3.3 Удаление вершины из дерева

Алгоритм удаления вершины с ключом равным X из случайного дерева поиска состоит в следующем. Сначала нужно найти вершину с ключом равными X. Если найденная вершина имеет не более одного поддерева, то её просто удаляем. На рисунке 10 показаны различные варианты расположения вершин. Удаляемая вершина выделена черным цветом.

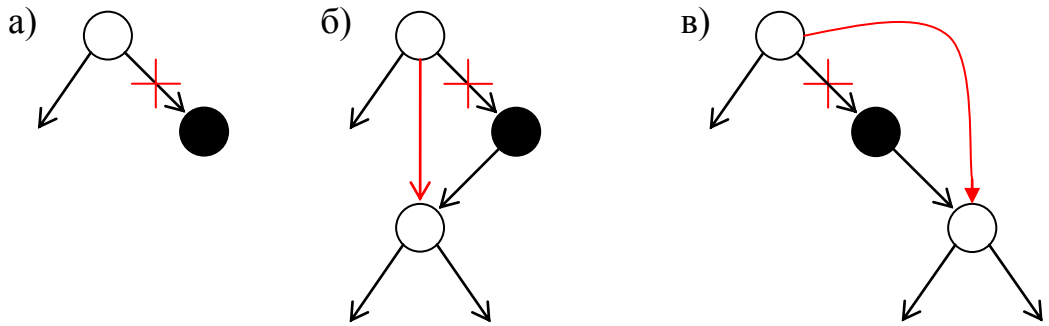


Рисунок 10 Варианты удаления вершин

Если найденная вершина имеет два поддерева (рис. 11), то тогда порядок действий следующий. На место удаляемой вершины ставится наибольшая вершина из левого поддерева (наименьшая из правого поддерева), т.е. самая правая вершина левого поддерева (самая левая из правого поддерева), которая не имеет правого поддерева (левого поддерева) (рис. 12).

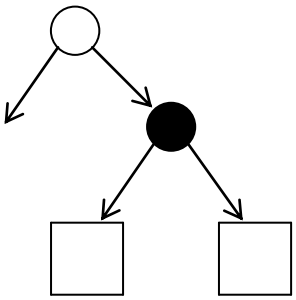


Рисунок 11 Удаляемая вершина с двумя поддеревьями

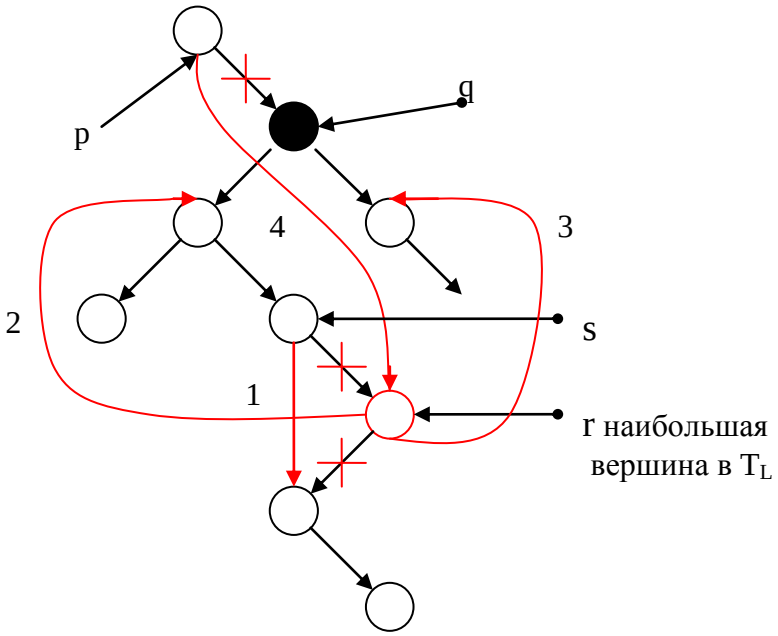


Рисунок 12 Порядок изменения указателей при удалении вершины

Алгоритм на псевдокоде

*Удаление из СДП (X, *Root)*

```
Обозначения
    p := @Root
    DO (*p ≠ NIL)
        IF ((*p)→Data < X) p := @((*p)→Right)
        ELSEIF ((*p)→Data > X) p := @((*p)→Left)
        ELSE  OD
        FI
    OD
    IF (*p ≠ NIL)
        q := *p
        IF (q→Left = NIL) *p := q→Right
        ELSEIF (q→Right = NIL) *p :=q→Left
        ELSE r := q→Left, s := q
        DO (r→Right ≠ NIL)
            s := r, r := r→Right
        OD
        s→Right := r→Left          1)
        r→Left := q→Left           2)
        r→Right := q→Right         3)
        *p := r                        4)
        FI
    dispose(q)
    FI
```

Трудоемкость удаления вершины складывается из конечного количества операций сравнения и присваивания на каждом шаге поиска в дереве. Таким образом, трудоемкость удаления такая же, как и в случае добавления в случайное дерево поиска.

3.4 Контрольные вопросы

1. Что такое случайное дерево поиска?
2. Какова средняя высота СДП?
3. Какова трудоемкость добавления и удаления вершины в СДП?
4. Каким образом вершина удаляется из СДП?

4. СБАЛАНСИРОВАННЫЕ ПО ВЫСОТЕ ДЕРЕВЬЯ (АВЛ-ДЕРЕВЬЯ)

4.1 Определение и свойства АВЛ-дерева

Как было показано выше, ИСДП обеспечивает минимальное среднее время поиска. Однако перестройка дерева после случайного включения вершины – довольно сложная операция. СДП дает среднее время поиска на 40 % больше, но процедура построения достаточно проста. Возможное промежуточное решение – введение менее строгого определения

сбалансированности. Одно из таких определений было предложено Г. М. Адельсон – Вельским и Е. М. Ландисом (1962).

Дерево поиска называется *сбалансированным по высоте*, или *АВЛ – деревом*, если для каждой его вершины высоты левого и правого поддеревьев отличаются не более чем на 1.

На рисунке 39 приведены примеры деревьев, одно из которых является АВЛ-деревом, а другое – нет. В выделенной вершине нарушается баланс высот левого и правого поддеревьев.

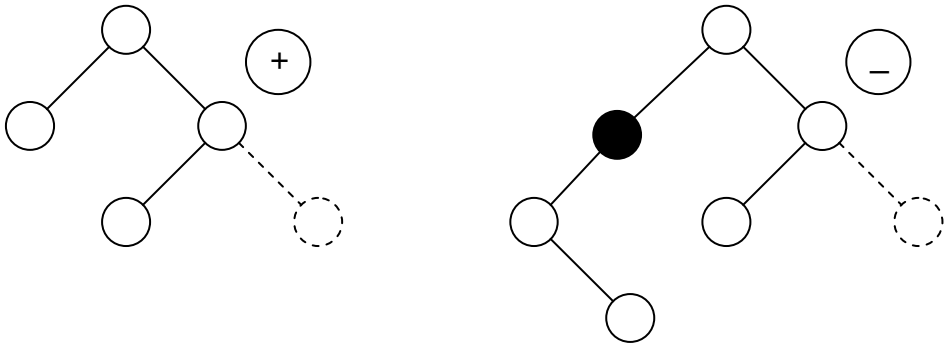


Рисунок 13 Пример АВЛ-дерева и не АВЛ-дерева

Заметим, что ИСДП является также и АВЛ – деревом. Обратное утверждение не верно.

Адельсон – Вельский и Ландис доказали теорему, гарантирующую, что АВЛ-дерево никогда не будет в среднем по высоте превышать ИСДП более, чем на 45% независимо от количества вершин:

$$\log(n+1) \le h_{АВЛ}(n) < 1{,}44 \log(n+2) - 0{,}328 \text{ при } n \to \infty.$$

Таким образом, лучший случай сбалансированного по высоте дерева – ИСДП, худший случай – плохое АВЛ – дерево. *Плохое АВЛ – дерево* это АВЛ-дерево, которое имеет наименьшее число вершин при фиксированной высоте. Рассмотрим процесс построения плохого АВЛ-дерева. Возьмём фиксированную высоту h и построим АВЛ – дерево с минимальным количеством вершин. Обозначим такое дерево через T_h. Ясно, что T_0 – пустое дерево, T_1 – дерево с одной вершиной. Для построения T_h при $h > 1$ будем брать корень и два поддерева с минимальным количеством вершин.

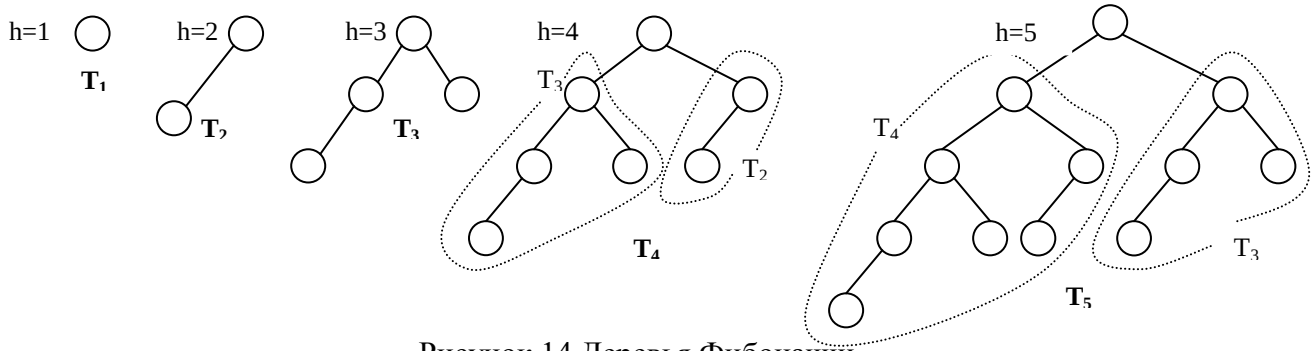


Рисунок 14 Деревья Фибоначчи

Одно поддерево должно быть высотой h–1, а другое высотой h–2. Поскольку принцип их построения очень напоминает построение чисел Фибоначчи, то такие деревья называют *деревьями Фибоначчи*: $T_h = < T_{h-1}, x, T_{h-2} >$. Число вершин в T_h определяется следующим образом:

$$n_0 = 0,\ n_1 = 1,\ n_h = n_{h-1} + 1 + n_{h-2}$$

4.2 Повороты при балансировке

Рассмотрим, что может произойти при включении новой вершины в сбалансированное по высоте дерево. Пусть r – корень АВЛ-дерева, у которого имеется левое поддерево (T_L) и правое поддерево (T_R). Если добавление новой вершины в левое поддерево приведет к увеличению его высоты на 1, то возможны три случая:

1) если $h_L = h_R$, то T_L и T_R станут разной высоты, но баланс не будет нарушен;
2) если $h_L < h_R$, то T_L и T_R станут равной высоты, т. е. баланс даже улучшится;
3) если $h_L > h_R$, то баланс нарушиться и дерево необходимо перестраивать.

Введём в каждую вершину дополнительный параметр Balance (показатель баланса), принимающий следующие значения:

-1, если левое поддерево на единицу выше правого;
0, если высоты обоих поддеревьев одинаковы;
1, если правое поддерево на единицу выше левого.

Если в какой-либо вершине баланс высот нарушается, то необходимо так перестроить имеющееся дерево, чтобы восстановить баланс в каждой вершине. Для восстановления баланса будем использовать процедуры поворотов АВЛ-дерева.

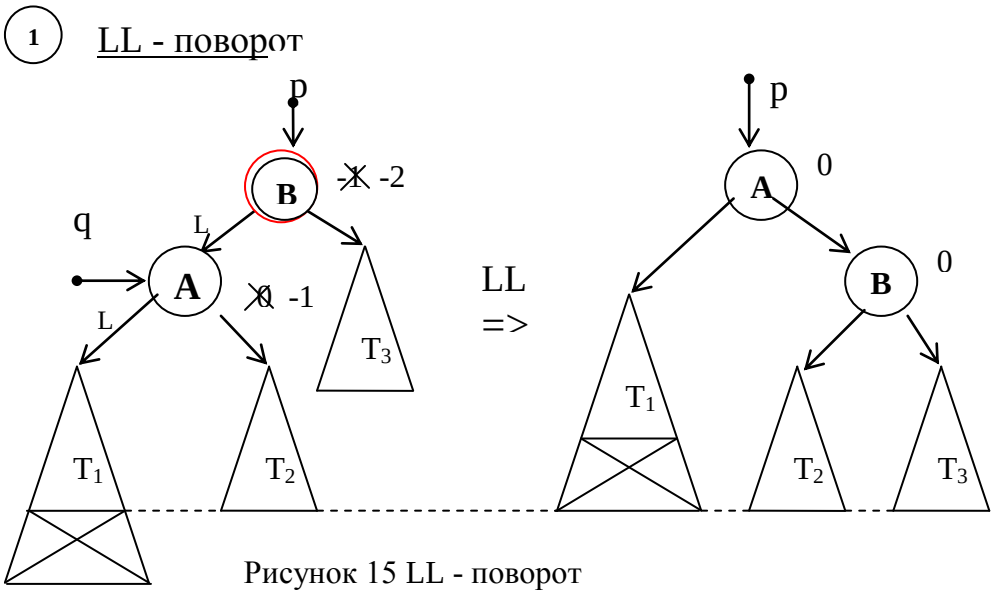


Рисунок 15 LL - поворот

Алгоритм на псевдокоде

LL - поворот

q := p→Left
q→Balance := 0
p→Balance := 0
p→Left := q→Right
q→Right := p
p := q

2 LR – поворот

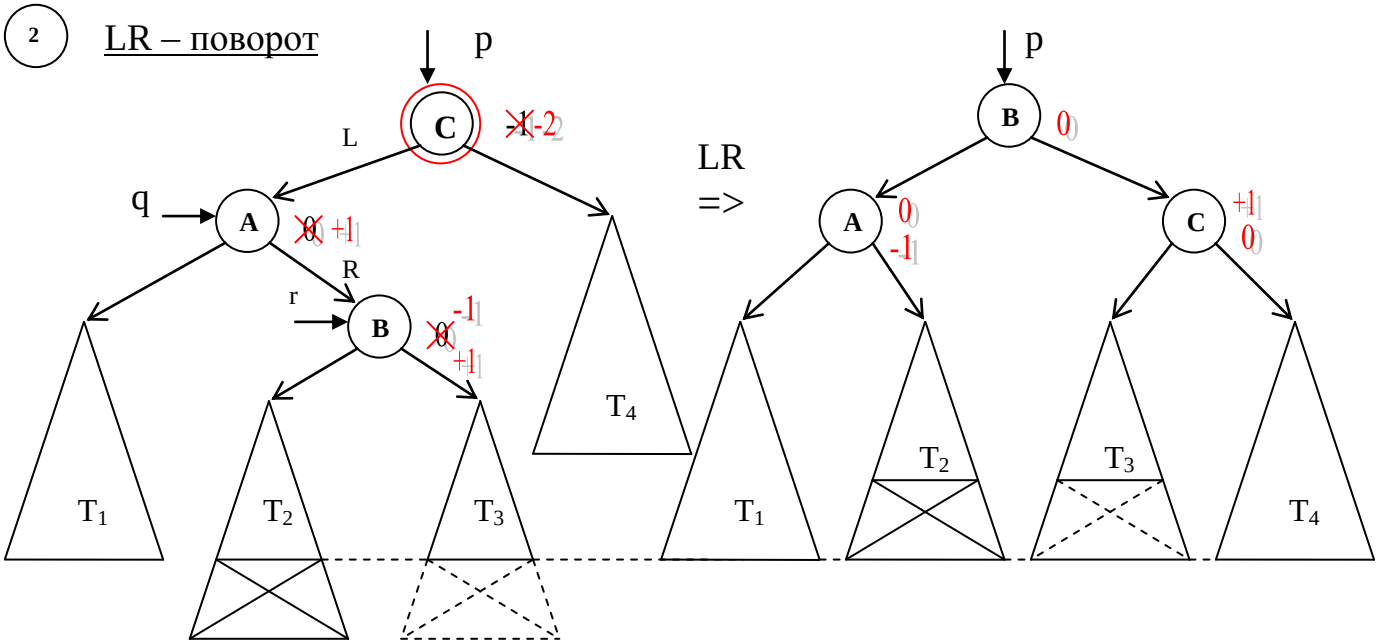


Рисунок 16 LR – поворот

Алгоритм на псевдокоде

LR - поворот

q := p→Left, r := q→Right
IF (r→Balance<0) p→Balance := +1 ELSE p→Balance := 0 FI
IF (r→Balance>0) q→Balance := –1 ELSE q→Balance := 0 FI
r→Balance := 0
p→Left := r→Right, q→Right := r→Left
r→Left := q, r→Right := p, p := r

3 RR – поворот

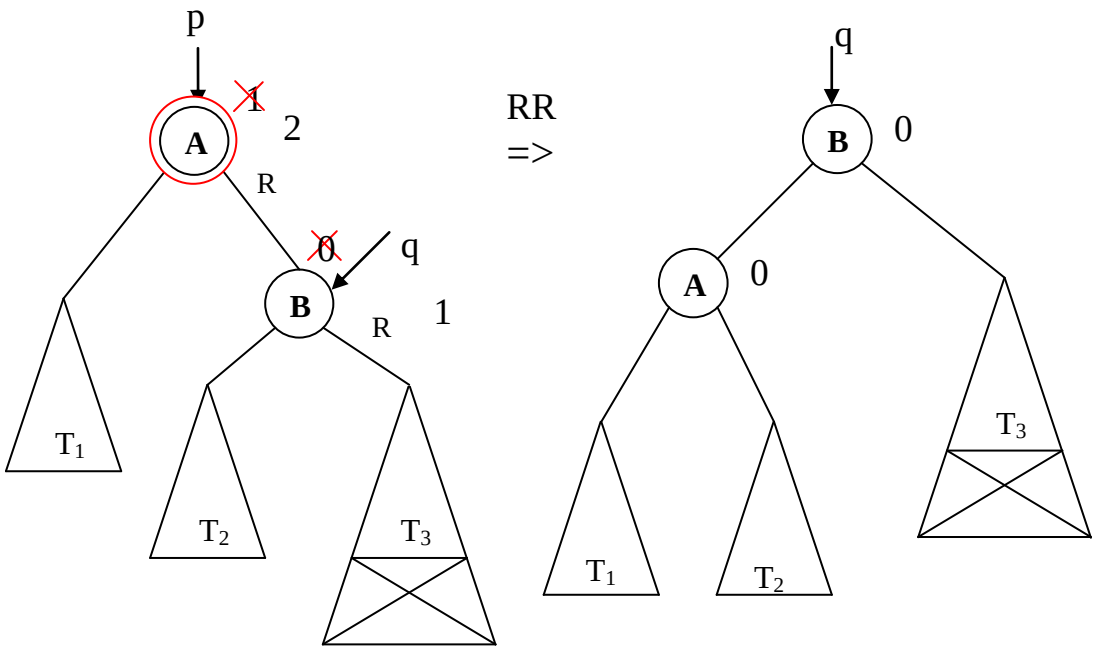


Рисунок 17 RR – поворот

Алгоритм на псевдокоде

RR - поворот

```
q := p→Right
q→Balance := 0
p→Balance := 0
p→ Right:= q→ Left
q→ Left := p
p := q
```

4 RL – поворот

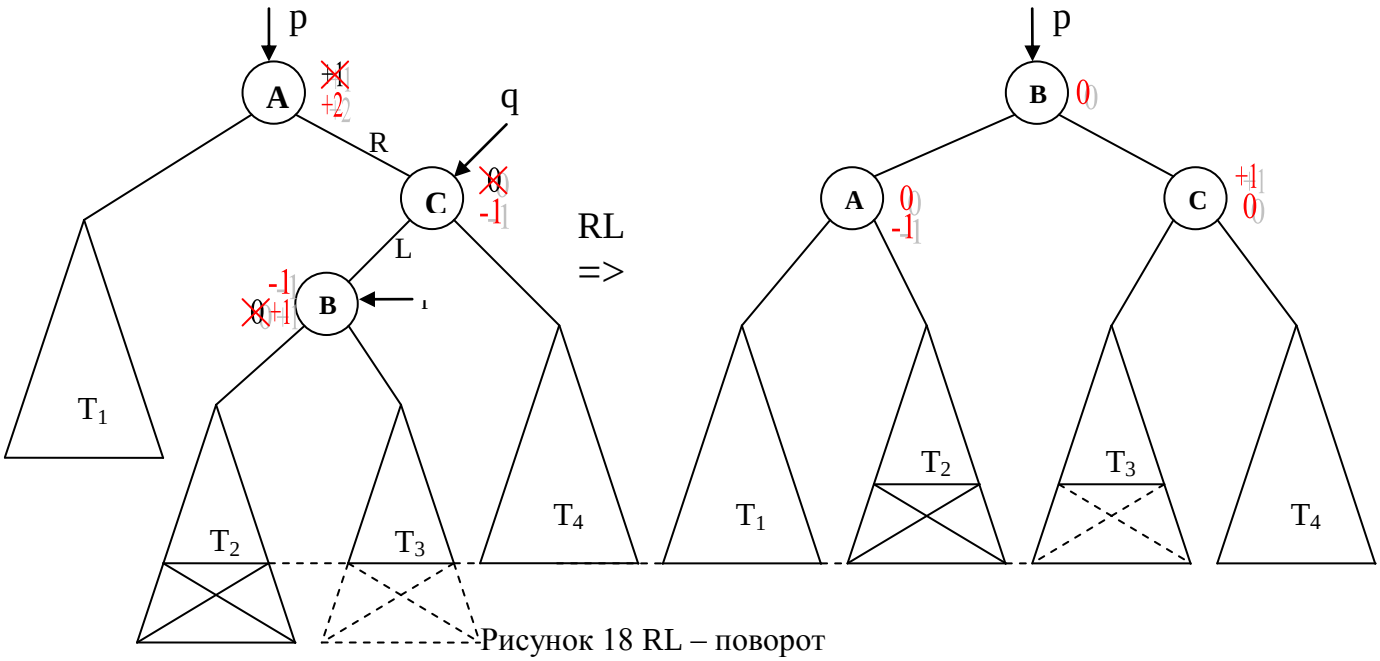


Рисунок 18 RL – поворот

Алгоритм на псевдокоде

RL - поворот

```
q := p→ Right, r := q→ Left
IF (r→Balance>0) p→Balance := -1 ELSE p→Balance := 0 FI
IF (r→Balance<0) q→Balance := 1 ELSE q→Balance := 0 FI
r→Balance := 0
p→ Right:= r→ Left, q→ Left:= r→ Right
r→ Left := p, r→Right := q, p := r
```

4.3 Добавление вершины в дерево

Добавление новой вершины в АВЛ-дерево происходит следующим образом. Вначале добавим новую вершину в дерево так же как в случайное дерево поиска (проход по пути поиска до нужного места). Затем, двигаясь назад по пути поиска от новой вершины к корню дерева, будем искать вершину, в которой нарушился баланс (т. е. высоты левого и правого поддеревьев стали отличаться более чем на 1). Если такая вершина найдена, то изменим структуру дерева для восстановления баланса с помощью процедур поворотов.

Алгоритм на псевдокоде

Добавление в АВЛ – дерево (D: данные; Var p: pVertex);

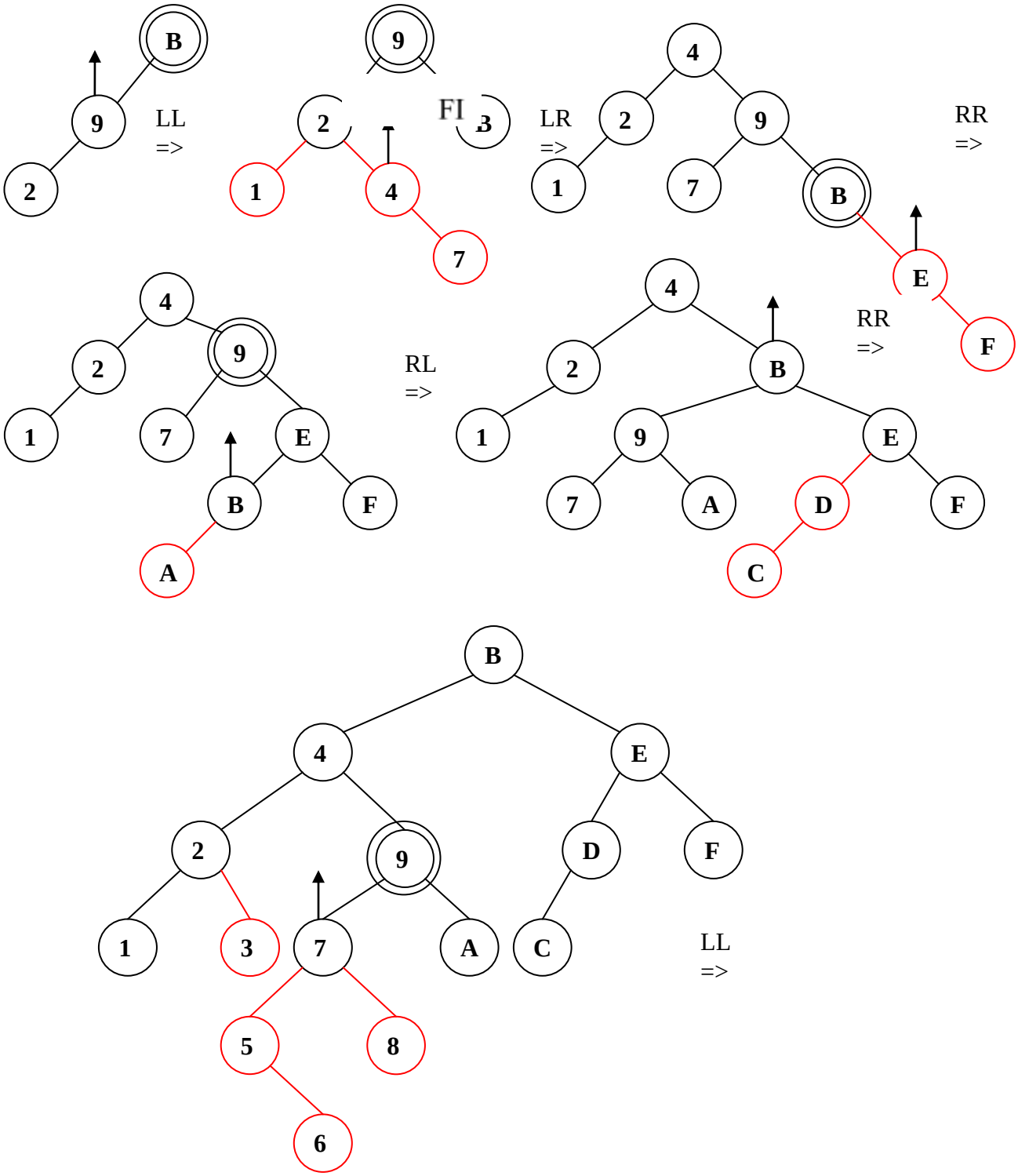
Обозначим

Рост – логическая переменная, которая показывает выросло дерево или нет.

```
IF (p = NIL)
        new(p), p→Data := D, p→Left := NIL, p→Right := NIL
        p→Balance := 0, Рост := ИСТИНА
ELSE
        IF (p→Data > D)
                Добавление в АВЛ – дерево (D, p→Left)
                IF (Рост = ИСТИНА)     {выросла левая ветвь}
                        IF (p→Balance > 0) p→Balance := 0, Рост := ЛОЖЬ
                        ELSE IF (p→Balance = 0) p→Balance := -1
                                ELSE
                                        IF (p→Left→Balance < 0)  <LL – поворот>
                                        ELSE   <LR – поворот>  FI
                                        Рост := ЛОЖЬ
                                        FI
                        FI
        ELSE IF (p→Data < D)
                <аналогичные действия для правого поддерева
                ELSE {p→Data = D, такая вершина уже есть}
                FI
        FI
```

FI

Пример: Построение АВЛ-дерева с вершинами B 9 2 4 1 7 E F A D C 3 5 8 6



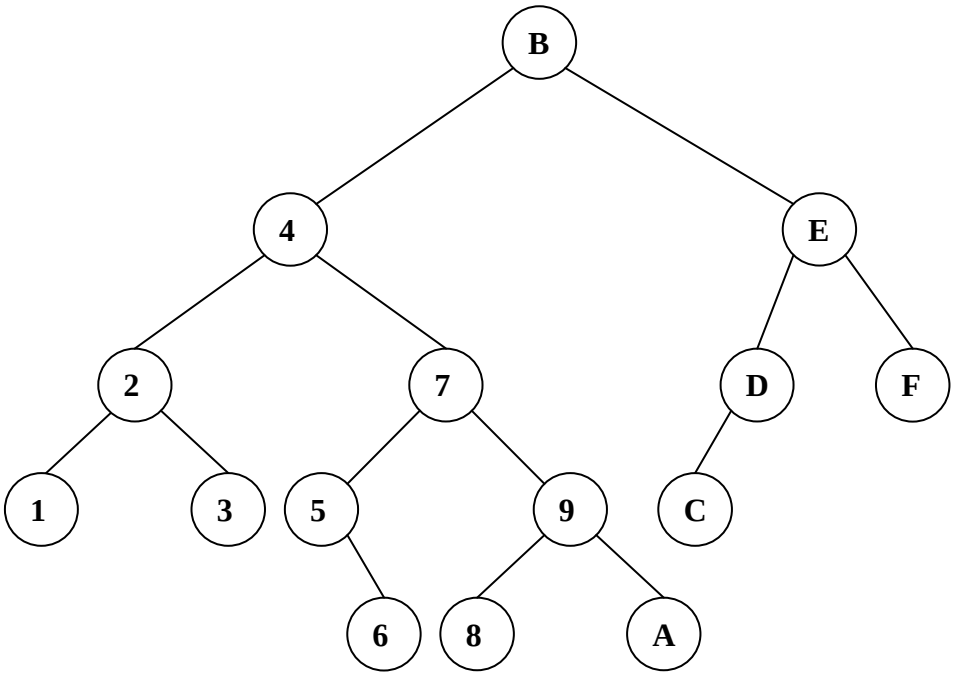


Рисунок 19 Построение АВЛ-дерева

4.4 Удаление вершины из дерева

Очевидно, удаление вершины – процесс намного более сложный, чем добавление. Хотя алгоритм операции балансировки остаётся тем же самым, что и при включении вершины. Балансировка по-прежнему выполняется с помощью одного из четырёх уже рассмотренных поворотов вершин.

Удаление из АВЛ-дерева происходит следующим образом. Удалим вершину так же, как это делалось для СДП. Затем двигаясь назад от удалённой вершины к корню дерева, будем восстанавливать баланс в каждой вершине (с помощью поворотов). При этом нарушение баланса возможно в нескольких вершинах в отличие от операции включения вершины в дерево.

Как и в случае добавления вершин, введём логическую переменную *Уменьшение*, показывающую уменьшилась ли высота поддерева. Балансировка идёт, только если *Уменьшение* = ИСТИНА. Это значение присваивается переменной *Уменьшение*, если обнаружена и удалена вершина или высота поддерева уменьшилась в процессе балансировки.

Введём две симметричные процедуры балансировки, т. к. они будут использоваться несколько раз в алгоритме удаления:

BL – используется при уменьшении высоты левого поддерева,

BR – используется при уменьшении высоты правого поддерева.

Рисунки 46 и 47 иллюстрируют три случая, возникающие при удалении вершины из левого (для BL) или правого (для BR) поддерева, в зависимости от исходного состояния баланса в вершине по адресу p.

Алгоритм на псевдокоде

BL (p: pVertex, Уменьшение: boolean)

```
IF (p→Bal = -1) p→Bal := 0
ELSEIF (p→Bal =  0) p→Bal := 1, Уменьшение := ЛОЖЬ
ELSEIF (p→Bal =  1)
                IF (p→Right→Bal ≥ 0)  <RR1-поворот>
                ELSE  <RL - поворот> FI
FI
```

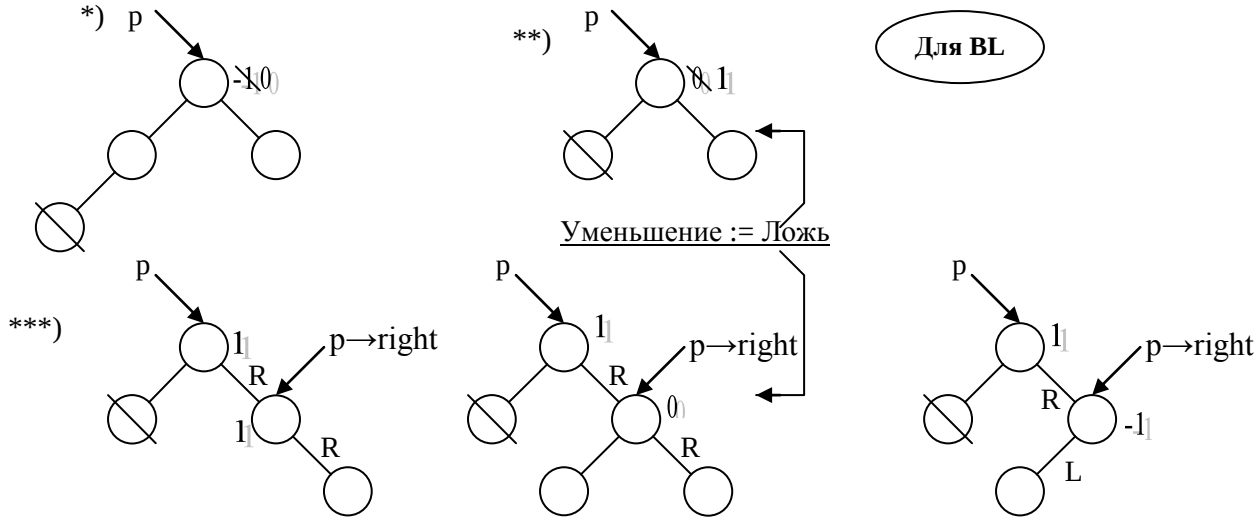


Рисунок 20 Три случая при удалении вершины из левого (для BL) поддерева

Алгоритм на псевдокоде

BR (p: pVertex, Уменьшение: boolean)

```
IF (p→Bal =  1) p→Bal := 0
ELSEIF (p→Bal =  0) p→Bal := -1, Уменьш := ЛОЖЬ
ELSEIF (p→Bal = -1)
        IF (p→Left→Bal ≤ 0)  <LL1 - поворот>
        ELSE  <LR - поворот> FI
FI
```

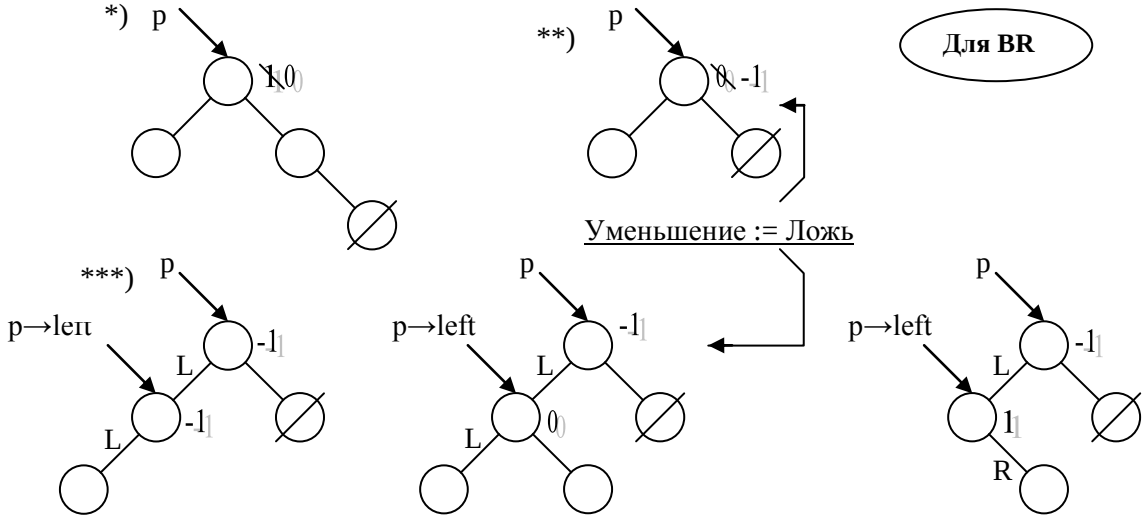


Рисунок 21 Три случая при удалении вершины правого (для BR) поддерева

При добавлении вершины не может быть случая, когда p→left→Bal = 0, поэтому LL – поворот необходимо изменить, чтобы учесть эту ситуацию.

Алгоритм на псевдокоде

LL1 – поворот

```
q := p→Left
IF (q→Bal = 0) p→Bal := -1, q→Bal := 1, Уменьш := false
ELSE p→Bal := 0, q→Bal := 0
p→Left := q→Right
q→Right := p
p := q
```

Аналогично изменяется RR – поворот, LR и RL – повороты не изменяются.

Алгоритм на псевдокоде

RR1 – поворот

```
q := p→Right
IF (q→Bal = 0) p→Bal := 1, q→Bal := -1, Уменьшение := ЛОЖЬ
ELSE p→Bal := 0, q→Bal := 0
FI
p→ Right:= q→ Left
q→ Left := p
p := q
```

Алгоритм на псевдокоде

Удаление из АВЛ-дерева (x: Данные, p: pVertex, Уменьшение: boolean)

```
IF (p = NIL) {ключа в дереве нет}
ELSE IF (p→Data > x) Удаление (x, p→Left, Уменьшение)
        IF Уменьшение BL (p, Уменьш) FI
ELSE IF (p→Data < x) Удаление (x, p→Right, Уменьшение)
        IF Уменьшение BR (p, Уменьшение) FI
ELSE IF {удаление вершины по адресу p}
    q := p
    IF (q→Right = NIL) p := q→Left, Уменьшение := ИСТИНА
    ELSE IF (q→Left = NIL) p := q→Right, Уменьшение := ИСТИНА
    ELSE del (q→Left, Уменьшение)
        IF Уменьшение  BL (p, Уменьшение) FI
    FI
    dispose(q)
FI
```

Используемая при удалении процедура del удаляет вершину, имеющую 2 поддерева, т. е. заменяет её на самую правую вершину из левого поддерева.

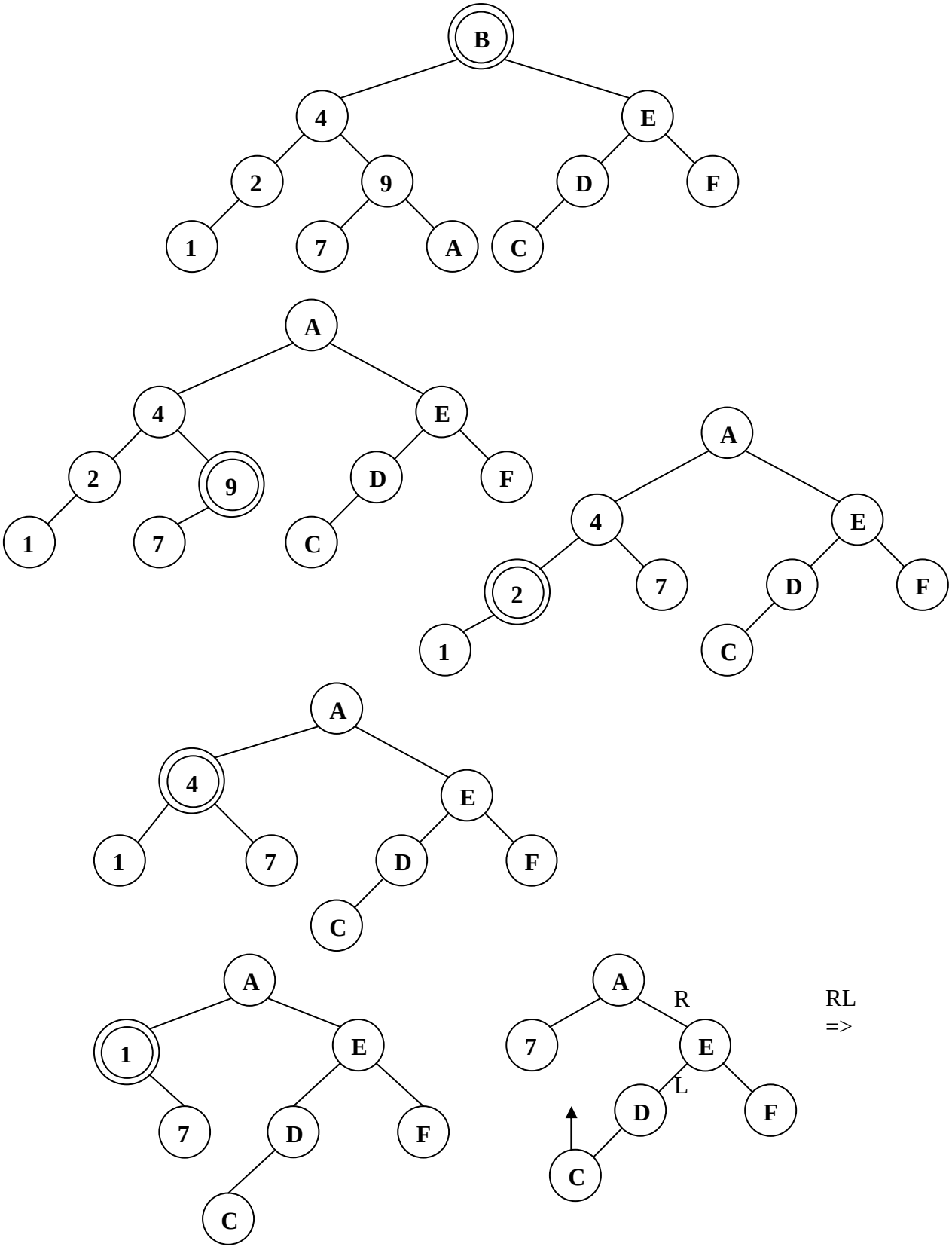
Алгоритм на псевдокоде

del (r: pVertex, Уменьшение: boolean)

```
IF (r→right ≠ NIL)
```

```
        del (r→right, Уменьшение)
        IF Уменьшение BR (r, Уменьшение) FI
    ELSE q→Data := r→Data
        q := r
        r := r→Left
        Уменьшение := ИСТИНА
    FI
```

Пример: Удаление из АВЛ-дерева вершин B 9 2 4 1 7 E F



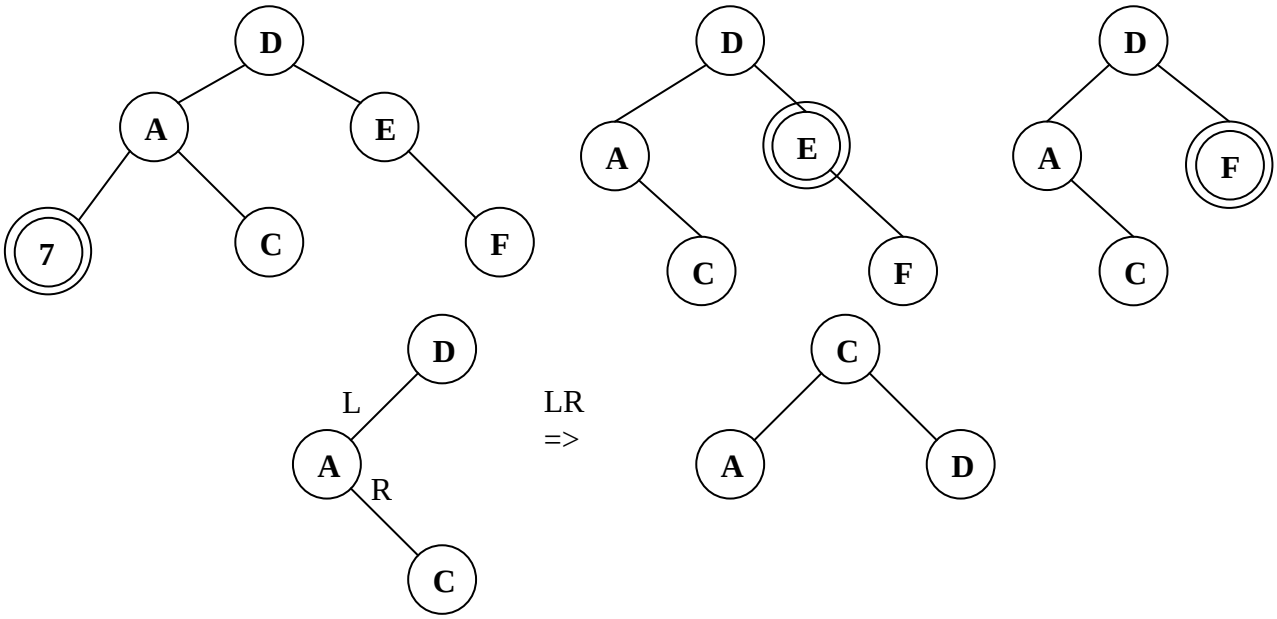


Рисунок 22 Удаление из АВЛ-дерева

Поиск элемента с заданным ключом, включения нового элемента, удаления элемента – каждое из этих действий в АВЛ-дереве можно произвести в худшем случае за O(log *n*) операций.

Отличие между процедурами включения и удаления заключается в следующем. Включение может привести самое большое к одному повороту, исключение может потребовать поворот во всех вершинах вдоль пути поиска. Наихудшим случаем с точки зрения количества балансировок является удаление самой правой вершины у плохого АВЛ-дерева (дерева Фибоначчи). По экспериментальным оценкам на каждые два включения встречается один поворот, а при исключении поворот происходит в одном случае из пяти.

4.5 Контрольные вопросы

1. Дайте определение АВЛ-дерева.
2. Является ли АВЛ-дерево ИСДП?
3. Является ли АВЛ-дерево деревом поиска?
4. Какова высота АВЛ-дерева?
5. Для чего нужны повороты АВЛ-дерева?
6. Какова трудоемкость включения и исключения вершины АВЛ-дерева?
7. Что такое показатель баланса вершины?

5. Б-ДЕРЕВЬЯ

5.1 Определение Б-дерева порядка m

Деревья, имеющие вершины со многими потомками, будем называть *сильноветвящимися*. Такие деревья могут быть эффективно использованы для решения следующей задачи: формирование и поддержание крупномасштабных деревьев поиска, для которых необходимы включения и

удаление элементов, но для которых либо не хватает оперативной памяти, либо она слишком дорога для долговременного использования.

В этом случае вершины дерева могут храниться во внешней памяти (например, на диске), тогда ссылки представляют собой адреса на диске, а не в оперативной памяти. Если использовать двоичное дерево для $n = 10^6$ элементов, то потребуется $\log(10^6) = 20$ шагов поиска. Так как каждый шаг включает в себя обращение к диску, то необходимо минимизировать число обращений к диску. Сильноветвящиеся деревья – идеальное решение этой проблемы, т.к. при обращении к диску можно считывать не один элемент, а целую группу, причём размер сектора диска определяет размер минимальной порции (обычно кратен 512 байт). Таким образом, за одно обращение считывается поддерево, которое будем называть *страницей.* Например, пусть для дерева из 10^6 элементов размер страницы равен 100 вершин. Поиск будет требовать в среднем $\log_{100}(10^6) = 3$, а не 20 обращений к диску. Однако если дерево растёт случайным образом, то в худшем случае может потребоваться даже 10^4 обращений. Поэтому Р. Байером и Е. Маккрейтом был сформулирован критерий управления ростом дерева: каждая страница (кроме одной) должна содержать (при заданном постоянном m) от m до $2m$ элементов.

Таким образом, для дерева с n элементами и максимальном размером в $2m$ вершин в худшем случае потребуется $\log_m n$ обращений к страницам (диску). При этом коэффициент использования памяти не меньше, чем 50%, так как страницы всегда заполнены минимум наполовину. Также эта схема позволяет достаточно просто осуществлять поиск, включения и удаления элементов.

Введем следующее определение. *Б – дерево порядка m* – это дерево со следующими свойствами:

1. В каждой странице хранится k элементов данных $d_1 < d_2 < ... < d_k$ и $k+1$ указатель $p_0, p_1, ...p_k$. Каждый указатель p_i либо равен NIL, либо указывает на вершину, все элементы которой больше d_i, но меньше d_{i+1}.

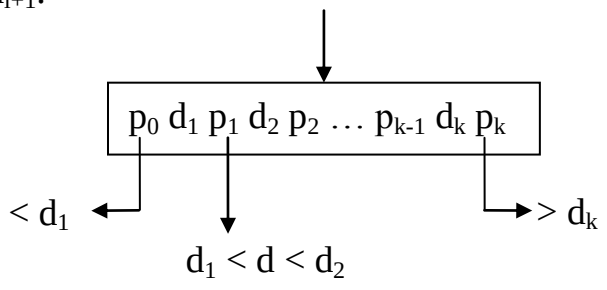


Рисунок 23 Страница Б-дерева

2. Для каждой вершины, кроме корня $m \le k \le 2m$, для корня $1 \le k \le 2m$.
3. Все листья дерева расположены на одном уровне.

Пример Б-дерева при $m = 2$.

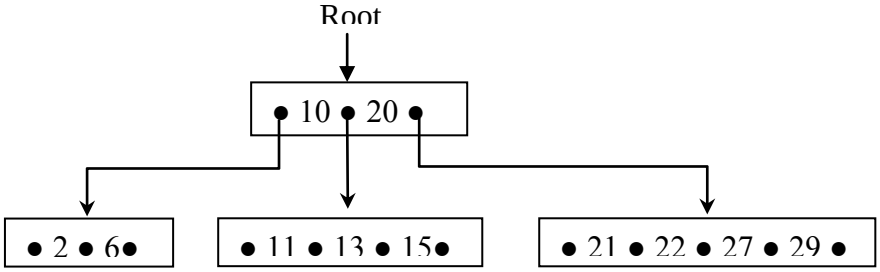


Рисунок 24 Пример Б-дерева

Очевидно, количество обращений к диску равно высоте Б-дерева. Легко видеть, что минимальное количество вершин в Б-дереве при заданной высоте h определяется равенством $n_{\min} = 2(m+1)^{h-1} - 1$. Отсюда высота Б-дерева порядка m с n элементами данных $h = \frac{\log(n+1)-1}{\log(m+1)} + 1$. Например, при m =255 высота h меньше, чем log n/8, т.е. по сравнению с обычными двоичными деревьями требуется в 8 раз меньше обращений к диску.

5.2 Поиск в Б-дереве

Если спроецировать Б – дерево на один уровень, то элементы расположатся в возрастающем порядке слева направо. Такое размещение определяет способ поиска элемента с заданным ключом. Страница считывается в оперативную память, а затем производится поиск среди элементов $d_1, d_2, \ldots, d_k$ (упорядоченных!). Если k мало, то можно использовать простой последовательный поиск, если k достаточно большое, то – двоичный поиск.

Будем считать, что элементы на странице представлены в виде массива. Тогда структура данных может быть следующим образом описана (на Паскале):

```
type pPage = ^Page; {указатель на страницу}
        Item = record  {тип элемента}
                Data: integer;
                p: pPage;
                end;
        Page = record  {тип страницы}
                k: 0 . . 2*m;
                p0: pPage;
                e: array [1 . . 2*m] of Item;
                end;
```

где m – порядок Б – дерева,
k – количество элементов на странице,
p0 – указатель на страницу с элементами < d_1,
e – массив элементов на странице.

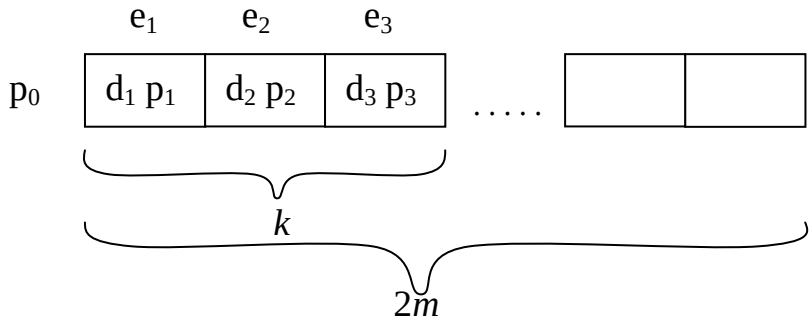


Рисунок 25 Структура страницы Б-дерева

Алгоритм на псевдокоде

Поиск в Б – дереве (D: Данные, a: pPage)

Обозначим L, R левая и правая границы поиска на странице

```
IF (a = NIL) (Ключа D нет в дереве)
ELSE L:= 1, R := k + 1
    DO (L < R)
        i := [(L + R)/2]
        IF (a→e_i.data ≤ D) L := i + 1
        ELSE R := i
        FI
    OD
    R := R – 1
    IF (R > 0 и a→e_R.data = D) (ключ D есть в дереве)
    ELSE (на этой странице ключа D нет)
        IF (R = 0) Поиск в Б-дереве (D, a→p_0)
        ELSE Поиск в Б-дереве (D, a→e_R.p)
        FI
    FI
FI
```

5.3 Построение Б-дерева

Построение Б-дерева или включение нового элемента данных D в Б-дерево происходит следующим образом:

- Выполним поиск элемента D в дереве.
- Если элемента D нет в дереве, то мы имеем страницу a и позицию R, в которой ожидали найти элемент D.
- Вставим элемент в позицию R+1, при этом количество элементов на странице *k* увеличилось на 1.
- Если $k <= 2m$, то процесс включения закончен.
- Если $k > 2m$ (переполнение страницы), то создаём новую страницу b, переносим в неё *m* правых элементов из страницы a, а средний элемент переносим на один уровень вверх на родительскую страницу.
- Включение элемента в родительскую страницу производится по такому же алгоритму.

- Если родительской страницы нет, то она создаётся и в неё включается один элемент.

Эта схема сохраняет все характерные свойства Б-дерева. Получившиеся две новые страницы содержат ровно по *m* элементов. Включение элемента в родительскую страницу может опять перевести к переполнению, то есть разделение страниц может распространиться до самого корня. В этом случае может увеличиться высота дерева. Таким образом, Б-деревья растут обратно: от листа к корню.

Пример построения Б-дерева при *m* = 2. Предварительно удалим повторяющиеся буквы.

КУРАПОВАЕЛЕНАВИКТОРОВНА

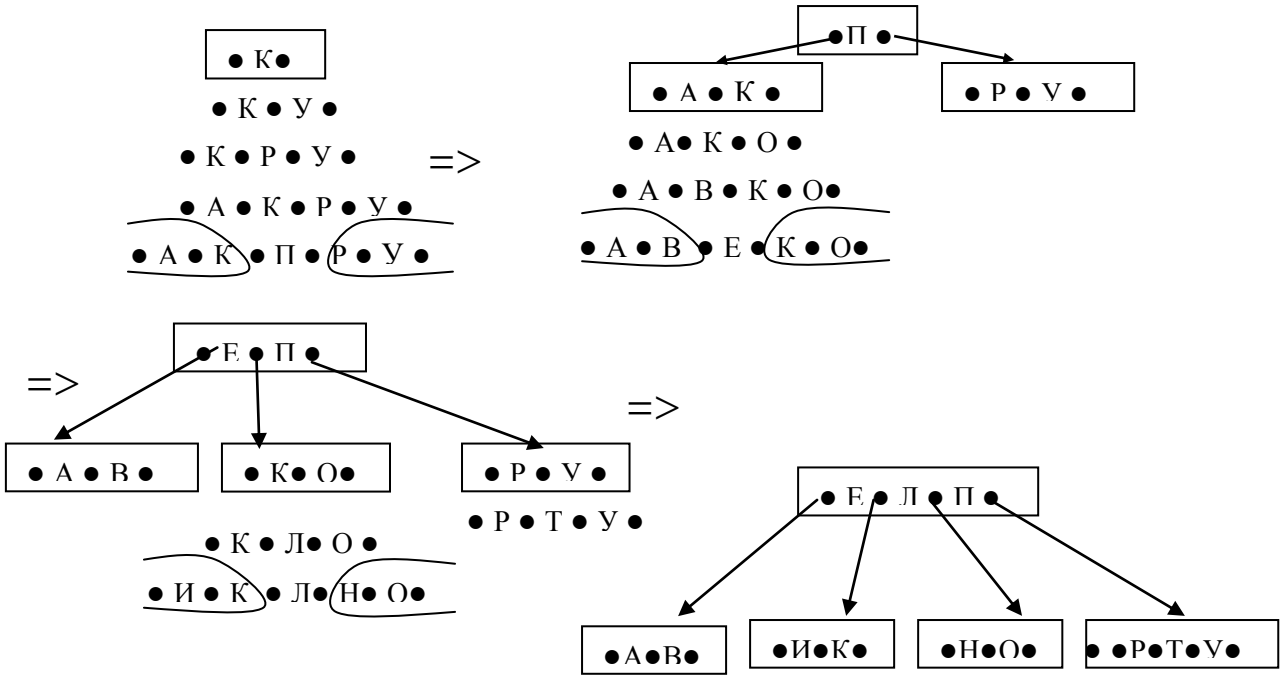


Рисунок 26 Построение Б-дерева

Алгоритм на псевдокоде

Построение Б – дерева (D: Данные, a: pPage, Rost: Boolean, V: Item)

Обозначим D – добавляемые данные

a – адрес страницы

Rost – логическая переменная; (принимает значение ИСТИНА, если дерево выросло).

V – рабочая переменная типа Item для элемента, передаваемого вверх по дереву.

u – рабочая переменная типа Item, фактический параметр при вызове процедуры построения

```
IF (a = NIL) {D нет в дереве, включение}
    V.Data := D
    V.p := NIL
    Rost := ИСТИНА
ELSE  Поиск в Б-дереве (D,a)
    IF (R > 0 и a→ e_R.data = D ) {элемент D есть в дереве}
```

```
ELSE (на этой странице ключа D нет)
    IF(R = 0) Построение Б-дерева(D, a→p0, Rost, u)
    ELSE Построение Б-дерева (D, a→eR.p, Rost, u)
    FI
    IF (Rost = true) {включение u вправо от eR}
        IF (a→k < 2m) Rost := false {есть место на странице}
            a→k := a→k+1
            DO (i = a→k, a→k-1, ..., R+2) a→ei = a→ei-1
            OD
            a→e R+1 := u
        ELSE {переполнение страницы}
            New(b) {создание новой страницы}
            IF (R<= m)
                IF (R=m) V := u
                ELSE V :=a→em
                    DO ( i = m, m-1, ..., R+2) a→ei := a→ei-1
                    OD
                    a→eR+1:= u
                FI
                DO (i:=1, 2, ... m ) b→ei := a→ei+m
                OD
            ELSE {включение в правую страницу}
                R:= R-m
                V:=em+1
                DO(i=1, 2, ... R-1) b→ei := a→ei+m+1
                OD
                b→eR:= u
                DO ( i:=R+1, R+2, ..., m) b→ei := a→ei+m
                OD
            FI
            a→k := m
            b→k := m
            b→p0:= V.p
            V.p := b
        FI
    FI
FI
```

При создании Б-дерева необходимо предусмотреть разделение корневой страницы и создания нового корня из одного элемента. Это будет выполняться, если после обращения к процедуре построения Б-дерева с параметром Rost равным ИСТИНА.

Алгоритм на псевдокоде

Разделение корневой страницы

```
Root := NIL
DO <ввести D>
        Построение Б-дерева (D, Root, Rost, u)
        IF (Rost = true) (включение новой корневой страницы)
                q :=Root
                new(Root)
                root→k := 1
                root→p0 := q
                root→e1:= u
        FI
OD
```

5.4 Определение двоичного Б-дерева

На первый взгляд кажется, что наименьший интерес представляют Б-деревья первого порядка. Но иногда стоит обращать внимание и на такие исключительные варианты. Очевидно, что Б-дерево первого порядка не имеет смысла использовать для представления больших множеств данных, требующих вторичной памяти, т.к. приблизительно 50% всех страниц будут содержать только один элемент.

Двоичное Б-дерево состоит из вершин (страниц) с одним или двумя элементами. Следовательно, каждая страница содержит две или три ссылки на поддеревья. На рисунке 53 показаны примеры страниц Б – дерева при *m* = 1.

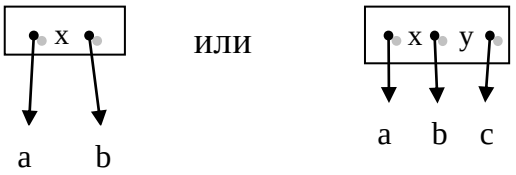


Рисунок 27 Виды вершин ДБД

Поэтому вновь рассмотрим задачу построения деревьев поиска в оперативной памяти компьютера. В этом случае неэффективным с точки зрения экономии памяти будет представление элементов внутри страницы в виде массива. Выход из положения – динамическое размещение на основе списочной структуры, когда внутри страницы существует список из одного или двух элементов.

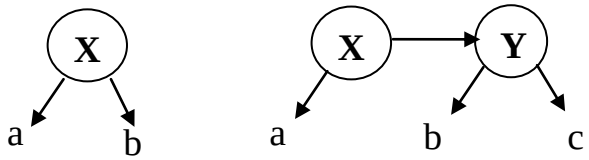


Рисунок 28 Вершины двоичного Б-дерева

Таким образом, страницы Б-дерева теряют свою целостность и элементы списков начинают играть роль вершин в двоичном дереве. Однако остается

необходимость делать различия между ссылками на потомков (вертикальными) и ссылками на одном уровне (горизонтальными), а также следить, чтобы все листья были на одном уровне.

Очевидно, двоичные Б-деревья представляют собой альтернативу АВЛ-деревьям. При этом поиск в двоичном Б-дереве происходит как в обычном двоичном дереве.

Высота двоичного Б-дерева $h = \frac{\log(n+1)-1}{\log(1+1)} + 1 = \log(n+1)$. Если рассматривать двоичное Б-дерево как обычное двоичное дерево, то его высота может увеличиться вдвое, т.е. $h = 2\log(n+1)$. Для сравнения, в АВЛ-дереве даже в самом плохом случае $h<1.44 \log n$. Поэтому сложность поиска в двоичном Б-дереве и в АВЛ-дереве одинакова по порядку величины.

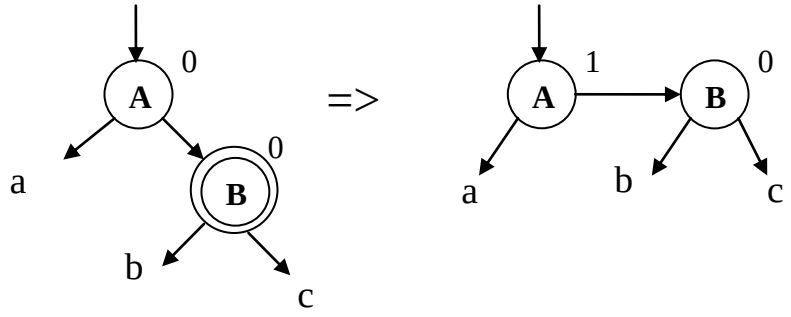
5.5 Добавление вершины в дерево

Построение двоичного Б-дерева происходит путем добавления новой вершины в уже существующее дерево. Введем логическую переменную VR, показывающую вертикальный рост дерева (в случае, если страница переполнилась и средний элемент передается на вышележащий уровень) и логическую переменную HR, определяющую горизонтальный рост дерева (если новый элемент размещается на этой же условной странице). Также определим показатель баланса BAL для каждой вершины, который принимает значение 0, если у данной вершины есть только вертикальные ссылки (вершина одна на странице), и значение 1, если у данной вершины есть правая горизонтальная ссылка.

При добавлении элементов в двоичное Б-дерево различают 4 ситуации, возникающих при росте левых или правых поддеревьев (см. рис. 55). Самый простой случай (1) — рост правого поддерева вершины A, когда A — единственный элемент на странице. Тогда вертикальная ссылка просто превращается в горизонтальную (HR=1, баланс вершины A равен 1). Если на странице уже два элемента (2), то при добавлении новой вершины C средняя вершина B передается на вышестоящий уровень (VR=1, баланс вершины B равен 0).

В случае роста левого поддерева, если вершина B одна на странице (3), то вершина A добавляется на эту страницу. Однако левая ссылка не может быть горизонтальной, поэтому требуется переопределение ссылок (HR=1, баланс вершины A равен 1). Если на странице два элемента, то, как и в случае 2, средняя вершина B поднимается на вышестоящий уровень (VR=1, баланс вершины B равен 0).

1)



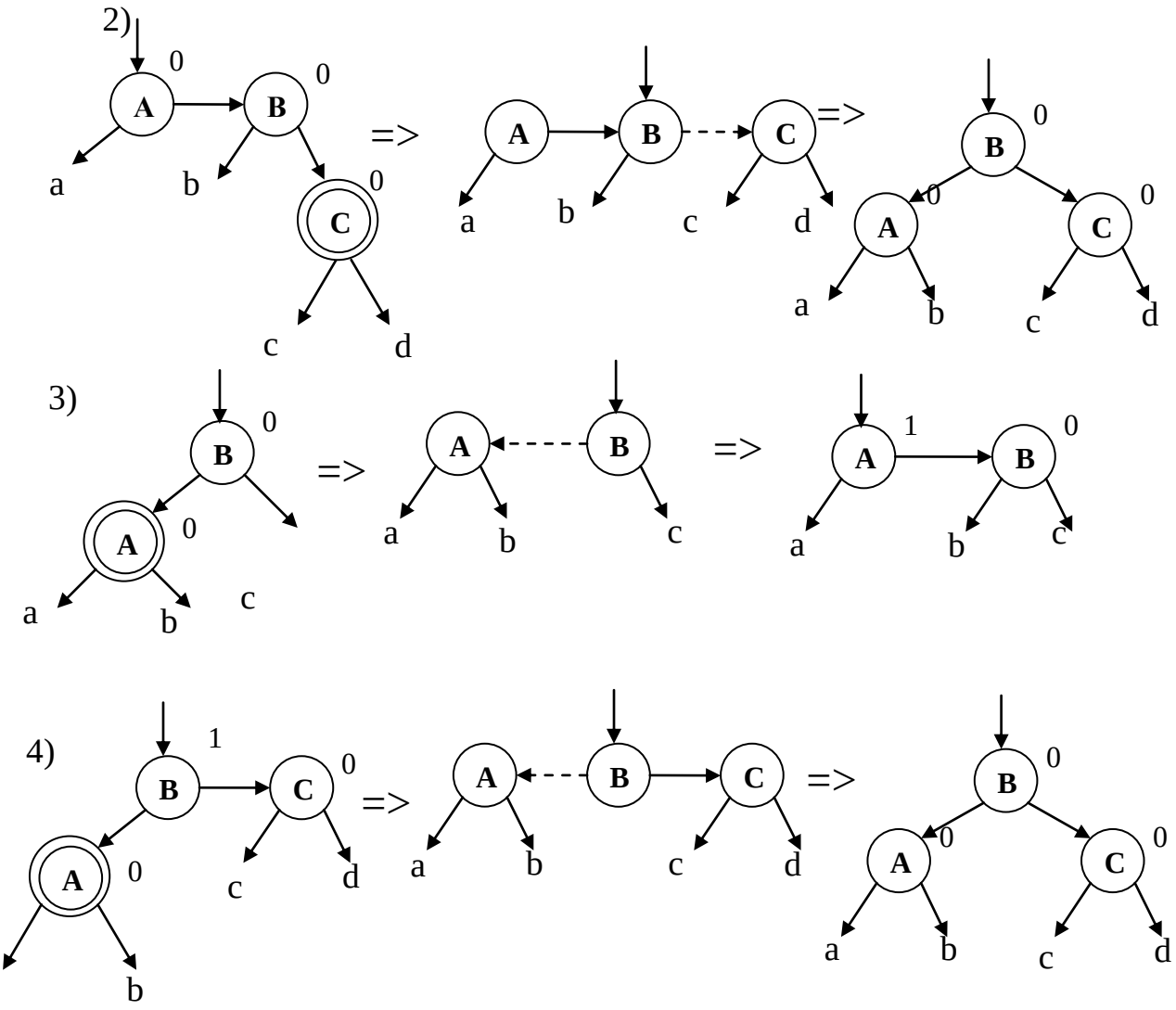


Рисунок 29 Четыре ситуации, возникающих при росте левых или правых поддеревьев

Алгоритм на псевдокоде

Добавление в ДБ-дерево(D: Данные, p: pVertex)

Обозначим

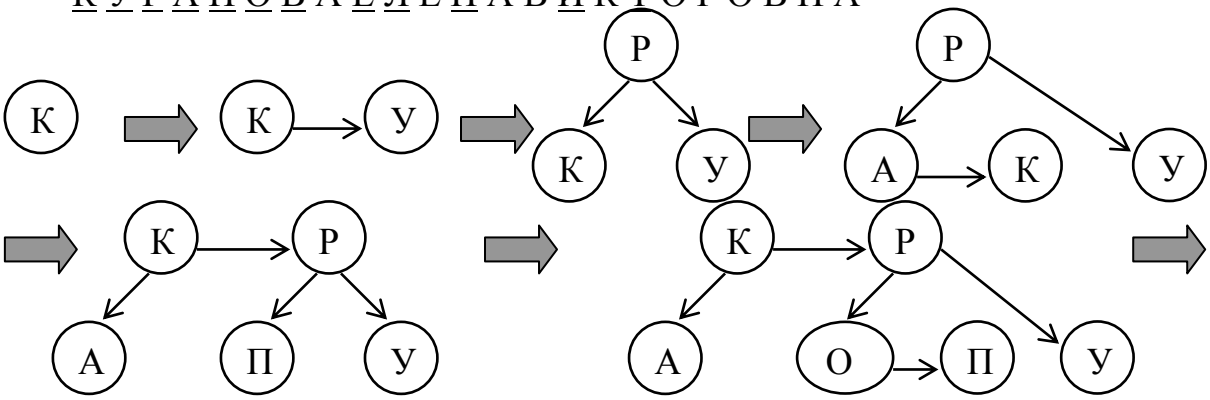
VR, HR — логические переменные, определяющие вертикальный или горизонтальный рост дерева (в начале VR, HR равны ИСТИНА)

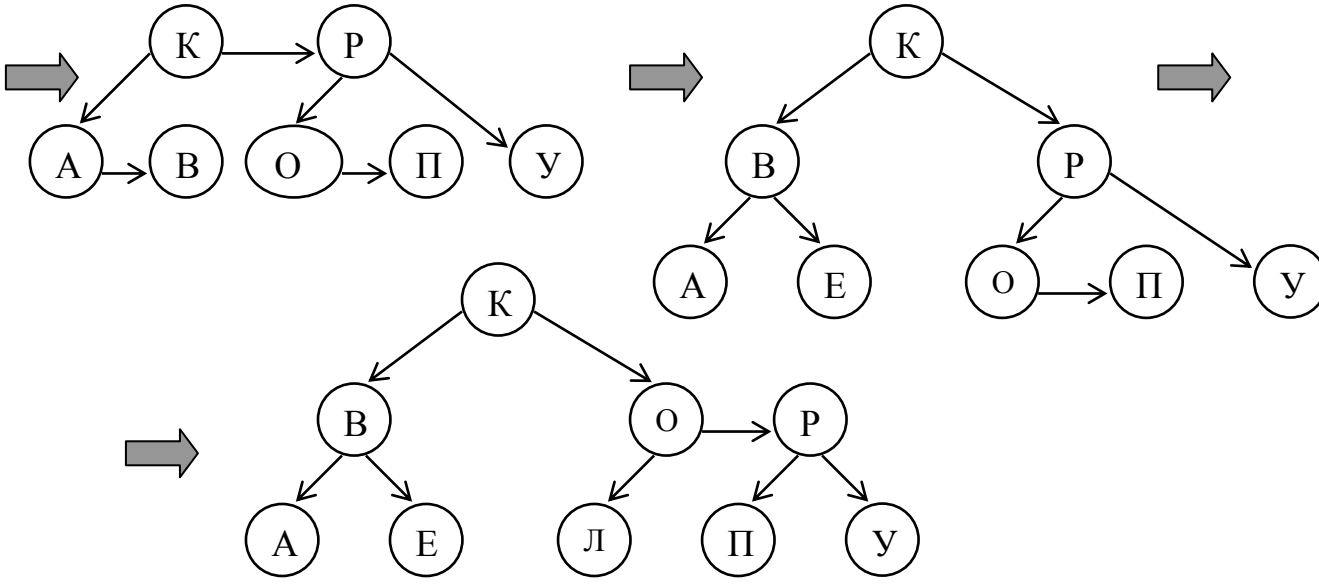
```
            IF (p=NIL)
        new(p); p→Date =D, p→Left= NIL; p→Right = NIL;
        p→Balance=0; VR= ИСТИНА;
    ELSE
        IF (p→Date> D) Добавление в ДБ-дерево(D, p→Left)
            IF (VR = ИСТИНА)
                IF (p→Balance = 0) q:= p→Left; p→Left:= q→Right;
                    q→Right := p; p:=q; q→Balance :=1
                    VR := ЛОЖЬ; HR =ИСТИНА;
                ELSE p→Balance:= 0; HR:=ИСТИНА;
                FI
```

```
            ELSE HR := ЛОЖЬ;
            FI
        ELSE IF (p→Date< D)
                Добавление в ДБ-дерево (D, p→Right),
                IF (VR = ИСТИНА) p→Balance:= 1; VR: = ЛОЖЬ;
                                        HR := ИСТИНА;
                ELSE IF (HR = ИСТИНА)
                        IF(p→Balance > 0) q := p→Right;
                            p→Right := q→Left;
                            p→Balance := 0; q→Balance := 0;
                            q→Left := p; p :=q
                            VR:= ИСТИНА; HR:= ЛОЖЬ;
                        ELSE HR:= ЛОЖЬ;
                        FI
                    FI
                FI
            FI
        FI
```

Пример построения двоичного Б-дерева приведен на следующем рисунке.

К У Р А П О В А Е Л Е Н А В И К Т О Р О В Н А





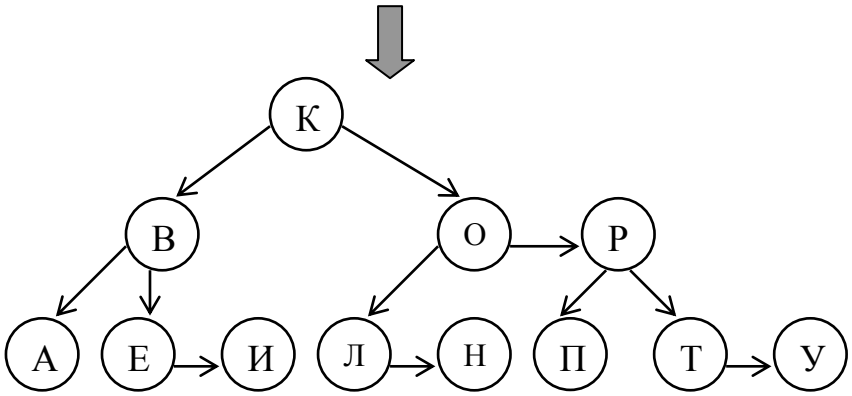


Рисунок 30 Построение двоичного Б-дерева

При построении двоичного Б-дерева реже приходится переставлять вершины, поэтому АВЛ-деревья предпочтительней в тех случаях, когда поиск ключей происходит значительно чаще, чем добавление новых элементов. Кроме того, существует зависимость от особенностей реализации, поэтому вопрос о применение того или иного тапа деревьев следует решать индивидуально для каждого конкретного случая.

5.6 Контрольные вопросы

1. Дайте определение Б-дерева порядка *m*.
2. Назовите трудоемкость поиска вершины в Б-дереве порядка *m*.
3. Какова высота Б-дерева порядка *m*.
4. Каким образом добавляется новый элемент в Б-дерева порядка *m*.
5. Что такое двоичное Б-дерево?
6. Какова высота двоичного Б-дерева?
7. Какова трудоемкость добавления вершины в двоичное Б-дерево?

6. ДЕРЕВЬЯ ОПТИМАЛЬНОГО ПОИСКА (ДОП)

6.1 Определение дерева оптимального поиска

До сих пор предполагалось, что частота обращения ко всем вершинам дерева поиска одинакова. Однако встречаются ситуации, когда известна информация о вероятностях обращения к отдельным ключам. Обычно для таких ситуаций характерно постоянство ключей, т.е. в дерево не включаются новые вершины и не исключаются старые и структура дерева остается неизменной. Эту ситуацию иллюстрирует сканер транслятора, который определяет, является ли каждое слово программы (идентификатор) служебным. Статистические измерения на сотнях транслируемых программ могут в этом случае дать точную информацию об относительных частотах появления в тексте отдельных ключей.

Припишем каждой вершине дерева V_i вес w_i, пропорциональный частоте поиска этой вершины (например, если из каждых 100 операций поиска 15 операций приходятся на вершину V_1, то w_1=15). Сумма весов всех вершин дает вес дерева W. Каждая вершина V_i расположена на высоте h_i, корень расположен на высоте 1. Высота вершины равна количеству операций сравнения, необходимых для поиска этой вершины. Определим

средневзвешенную высоту дерева с n вершинами следующим образом: $h_{ср}=(w_1h_1+w_2h_2+...+w_nh_n)/W$. Дерево поиска, имеющее минимальную средневзвешенную высоту, называется *деревом оптимального поиска* (ДОП).

Пример. Рассмотрим множество из трех ключей V_1=1, V_2=2, V_3=3 со следующими весами: w_1=60, w_2=30, w_3=10, W=100. Эти три ключа можно расставить в дереве поиска пятью различными способами.

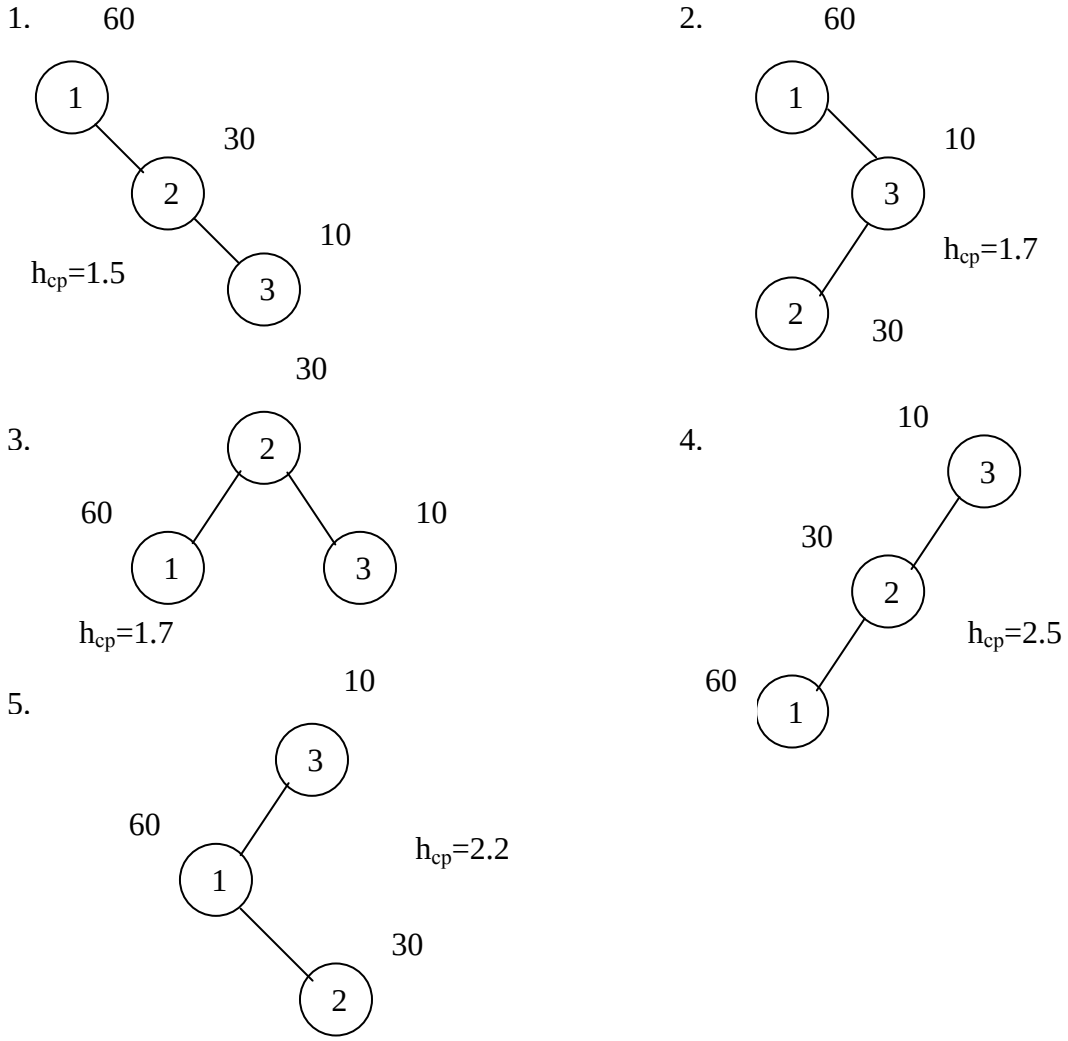


Рисунок 31 Различные деревья поиска с вершинами V_1=1, V_2=2, V_3=3

Легко видеть, что минимальной средневзвешенной высотой обладает дерево 1 на рисунке 57, которое представляет собой список или вырожденное дерево. Дерево 3 не является деревом оптимального поиска, хотя представляет собой идеально сбалансированное дерево. Очевидно, для минимизации средней длины пути поиска нужно стремится располагать наиболее часто используемые вершины ближе к корню дерева.

Задача построения ДОП может ставится в двух вариантах:

- Известны вершины и их веса.
- Вес вершины определяется в процессе работы. Например, после каждого поиска вершины ее вес увеличивается на 1. В этом случае необходимо перестраивать структуру дерева при изменении весов.

Далее будем рассматривать задачу построения ДОП с фиксированным набором ключей и их весов.

6.2 Точный алгоритм построения ДОП

Поскольку число возможных конфигураций из n вершин растет экспоненциально с ростом n, то решение задачи построения ДОП при больших n методом перебора нерационально. Однако деревья оптимального поиска обладают свойствами, которые позволяют получить алгоритм построения ДОП, начиная с отдельных вершин с последовательным включением новых вершин в дерево. Далее будем считать, что множество вершин, входящих в дерево, упорядочено. Поскольку вес дерева остается неизменным, то вместо средневзвешенной высоты будем рассматривать взвешенную высоту дерева: $P=h_1w_1+h_2w_2+\ldots+h_nw_n$

Свойство 1. *Для дерева поиска с весом W справедливо соотношение $P=P_L+W+P_R$, где P_L, P_R – взвешенные высоты левого и правого поддеревьев корня.*

Доказательство. Пусть вершина V_i с весом w_i является корневой для некоторого $i=1, \ldots n$. Поскольку левое и правое поддеревья являются деревьями поиска, то в левое поддерево входят вершины $V_1, V_2, \ldots, V_{i-1}$, а в правое – $V_{i+1}, \ldots, V_n$. Взвешенные высоты этих поддеревьев вычисляются следующим образом.

$$P_L = (h_1-1)w_1+(h_2-1)w_2+\ldots+(h_{i-1}-1)w_{i-1}$$
$$P_R = (h_{i+1}-1)w_{i+1}+ \ldots + (h_n-1)w_n$$

Рассмотрим выражение взвешенной высоты для всего дерева, замечая, что $h_i=1$

$$P=h_1w_1+h_2w_2+\ldots+h_nw_n= (h_1-1)w_1 + w_1+(h_2-1)w_2+ w_2\ldots+(h_{i-1}-1)w_{i-1} + w_{i-1} + + w_i + (h_{i+1}-1)w_{i+1}+ w_{i+1} \ldots + (h_n-1)w_n + w_n = P_L+W+P_R$$

Свойство 2. *Все поддеревья дерева оптимального поиска также являются деревьями оптимального поиска для соответствующих подмножеств вершин.*

Доказательство. Предположим, что одно из поддеревьев, например, правое, не является ДОП, т.е. существует дерево поиска с тем же множеством вершин, но с меньшей взвешенной высотой. Тогда по свойству 1 взвешенная высота всего дерева также не является минимальной. Данное противоречие доказывает свойство 2.

На основе приведенных свойств можно разработать точный алгоритм построения ДОП. Обозначим T_{ij} – оптимальное поддерево, состоящее из вершин $V_{i+1}, \ldots, V_j$. Введем матрицу $AR=\|Ar_{ij}\|$, $0\le i,j\le n$ элементы которой содержат номер корневой вершины поддерева T_{ij}, $0\le i<j\le n$. Взвешенную высоту поддерева T_{ij} обозначим Ap_{ij}, а вес поддерева T_{ij} обозначим Aw_{ij}, $0\le i<j\le n$. Очевидно, что $P=Ap_{o,n}$, $W=Aw_{o,n}$, T_{ii} – пустые деревья (без вершин), $Aw_{ii}=0$, $Ap_{ii}=0$, $i=1, \ldots n$.

Используя свойство 2, величины Aw_{ij}, Ap_{ij} можно вычислить рекуррентно по следующим соотношениям (для всех возможных поддеревьев):

$$Aw_{ij}=Aw_{i,j-1}+Aw_j, \qquad 0\le i < j \le n \qquad (1)$$

$$Ap_{ij}=Aw_{ij}+\min_{i<k\le j}(Ap_{i,k-1}+Ap_{k,j}),\ 0\le i < j \le n \qquad (2)$$

Во время вычислений будем запоминать индекс k^*, при котором достигается минимум во втором соотношении. Значение k^* является индексом корневой вершины поддерева T_{ij} во всем множестве вершин. Занесем в матрицу AR k^* –индекс корня T_{ij}, т.е. $Ar_{ij} = k^*$, $0\le i<j\le n$.

Идея построения дерева состоит в следующем. В матрице AR берем значение $Ar_{o,n}$ (номер корневой вершины всего дерева в упорядоченном массиве вершин), пусть оно равно k. Добавляем вершину V_k в дерево, используя обычный алгоритм добавления вершин в дерево поиска. Затем из матрицы AR берем значения $Ar_{o,k-1}$ и добавляем вершину с этим номером в левое поддерево. Далее берем $Ar_{k,n}$ и добавляем вершину с этим номером в правое поддерево и т.д.

Пример. Построить дерево оптимального поиска с вершинами V_1=1, V_2=2, V_3=3 и весами w_1=60, w_2=30, w_3=10. Сначала вычислим Aw_{ij}, Ap_{ij}, Ar_{ij}, $0\le i<j\le n$. Легко видеть, что

T_{00}, T_{11}, T_{22}, T_{33} – пустые поддеревья

T_{01}, T_{12}, T_{23} – поддеревья из одной вершины (1), (2), (3)

T_{02}, T_{13} – поддеревья из двух вершин (1,2) и (2,3)

T_{03} – поддерево из трех вершин (1,2,3)

По формулам (1) и (2) вычислим элементы матрицы весов AW и элементы матрицы взвешенных высот AP, значения матрицы AR запишем в верхних уголках ячеек матрицы AP.

Aw_{ij}	0	1	2	3
0	0	60	90	100
1		0	30	40
2			0	10
3				0

Ap_{ij}	0	1	2	3
0	0	60^1	120^1	150^1
1		0	30^2	50^2
2			0	10^3
3				0

$$Ap_{01}=Aw_{01}+\min_{0<k\le 1}(Ap_{00}+Ap_{11}) =60 \quad (k^*=1)$$

$$Ap_{12}=Aw_{12}+\min_{1<k\le 2}(Ap_{11}+Ap_{22}) =30 \quad (k^*=2)$$

$$Ap_{23}=Aw_{23}+\min_{2<k\le 3}(Ap_{22}+Ap_{33}) =10 \quad (k^*=3)$$

$$Ap_{02}=Aw_{02}+\min_{0<k\le 2}(Ap_{00}+Ap_{12},\ Ap_{01}+Ap_{22})=90+30=120 \quad (k^*=1).$$

$$Ap_{13}=Aw_{13}+\min_{1<k\le 3}(Ap_{11}+Ap_{23},\ Ap_{12}+Ap_{33})=40+10=50 \quad (k^*=2).$$

$$Ap_{03}=Aw_{03}+\min_{0<k\le 3}(Ap_{00}+Ap_{13},\ Ap_{01}+Ap_{23},\ Ap_{02}+Ap_{33})=100+50=150 \quad (k^*=3).$$

Корневой вершиной будет вершина V_1, поскольку $Ar_{0,3}=1$. Левое поддерево пустое, корень правого поддерева – вершина V_2 ($r_{1,3}=2$) и т.д. Полученное дерево показано на рисунке.

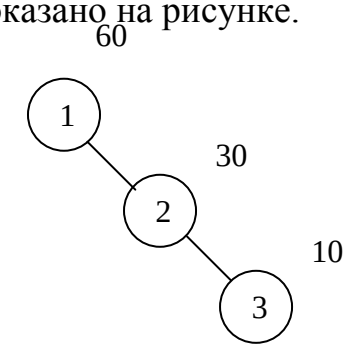


Рисунок 32 ДОП для w_1=60, w_2=30, w_3=10

Поскольку существует около $n^2/2$ значений Ap_{ij} , а вычисление (2) требует выбора одного из $0<i-j\le n$ значений, то весь процесс будет занимать $O(n^3)$ операций при $n\to\infty$. Д. Кнут отмечает, что можно избавиться от одного множителя n и тем самым сохранить практическую ценность алгоритма. Поиск Ar_{ij} можно ограничить, то есть сократить число вычислений до $j-i$ (если найден корень оптимального поддерева T_{ij}, то ни добавление справа новой вершины, ни отбрасывание левой вершины не могут сдвинуть вправо этот оптимальный корень). Это свойство выражается соотношением $Ar_{i,j-1} \le Ar_{ij} \le Ar_{i+1,j}$, что и ограничивает поиск Ar_{ij} диапазоном $Ar_{i,j-1}...Ar_{i+1,j}$. Это уменьшает трудоемкость алгоритма до $O(n^2)$ операций при $n\to\infty$.

Алгоритм на псевдокоде

Вычисление матрицы весов AW (формула 1).

```
DO (i = 0,...,n)
    DO (j = i+1,...,n)
        Aw_ij := Aw_i,j-1 + w_j
    OD
OD
```

Алгоритм на псевдокоде

Вычисление матриц AP и AR (формула 2).

```
DO (i = 0,...,n - 1)
    DO (j = i+1,...,n)
    Ap_ij+1 := Aw_ij+1
    Ar_ij+1 := i+1
```

OD
OD
DO (h = 2,...,n)
DO (i = 0,...,n - h)
j := i + h
m := $Ar_{i,j-1}$
min : = $Ap_{i,m-1} + Ap_{m,j}$
DO (k = m+1,...,$AR_{i+1,j}$)
x : = $Ap_{i,k-1} + Ap_{k,j}$
IF (x < min) m:=k , min:= x FI
OD
$Ap_{i,j}$:= min + $Aw_{i,j}$
$Ar_{i,j}$:=m
OD
OD

Алгоритм на псевдокоде

Создание дерева (L, R – границы массива вершин V, Root – указатель на корень дерева)

IF (L < R)
k:= $Ar_{L,R}$
Добавить (Root, V_k)
Создание дерева (L, k-1)
Создание дерева (k, R)
FI

Для построения дерева необходимо вызвать процедуру создания дерева с параметрами L=0, R=n, Root=NIL.

6.3 Приближенные алгоритмы построения ДОП

Рассмотренный выше точный алгоритм имеет квадратичную трудоемкость. Однако при больших объемах деревьев такие алгоритмы становятся неэффективными. Известны быстрые алгоритмы, строящие почти оптимальные деревья. Рассмотрим два из них.

Первый алгоритм (A1) предлагает в качестве корня использовать вершину с наибольшим весом. Затем среди оставшихся вершин снова выбирается вершина с наибольшим весом и помещается в левое или правое поддерево в зависимости от ее значения, и т.д.

Алгоритм на псевдокоде

Алгоритм A1(Root – указатель на корень дерева)

Обозначим

V.use – логическая переменная в структуре вершины дерева, которая показывает, что данная вершина была использована при построении дерева;

V.w – вес вершины.

```
Root : = NIL
DO (i = 1,...,n)
        V[i].use = ЛОЖЬ
OD
DO (i = 1,...,n)
        max:=0, Index:=0
        DO (j = 1,...,n)
                IF (V[j].w > max и V[j]. use=ЛОЖЬ)
                        max:=V[j].w
                        Index:=j
                FI
        OD
        V [index].use :=ИСТИНА
        Добавление в СДП (Root, V[index])
OD
```

Рассмотрим пример построения дерева почти оптимального поиска для символов строки К У Р А П О В А Е Л Е Н А В И К Т О Р О В Н А. Всего символов в строке 23, т.е. W=23. Различные символы определяют различные вершины дерева. Частоты вхождения символов (веса) приведены в таблице.

Таблица 1 Частоты вхождения символов в строку

К	У	Р	А	П	О	В	Е	Л	Н	И	Т
2	1	2	4	1	3	3	2	1	2	1	1

Посчитаем средневзвешенную высоту построенного дерева

$$P=4\cdot1+3\cdot2+3\cdot3+2\cdot3+2\cdot4+1\cdot4+1\cdot4+2\cdot5+ +2\cdot5+1\cdot5+1\cdot6+1\cdot6=78$$

$$h_{ср}=P/W=78/23=3,39$$

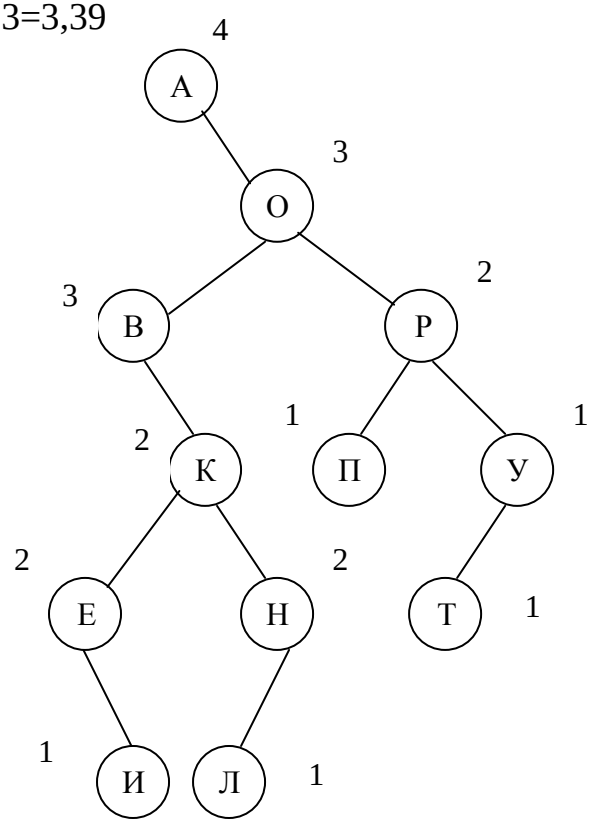


Рисунок 33 Дерево, построенное приближенным алгоритмом А1

Второй алгоритм (А2) использует предварительно упорядоченный набор вершин. В качестве корня выбирается такая вершина, что разность весов левого и правого поддеревьев была минимальна. Для этого путем последовательного суммирования весов определим вершину V_k, для которой справедливы неравенства:

$$\sum_{i=1}^{k-1} w_i < \frac{W}{2} \text{ и } \sum_{i=1}^{k} w_i \geq \frac{W}{2}.$$

Тогда в качестве "центра тяжести" может быть выбрана вершина V_k, V_{k-1} или V_{k+1}, т. е. вершина, для которой разность весов левого и правого поддерева минимальна. Далее действия повторяются для каждого поддерева

Алгоритм на псевдокоде

Алгоритм А2(Root – указатель на корень дерева,
L, R – левая и правая границы рабочей части массива)

```
wes=0, summa=0
IF (L<=R)
    DO (i=L, L+1, …,R) wes=wes+ V[i].w OD
    DO (i=L, L+1,…,R)
        IF (summa<wes/2 and summa + V[i].w>=wes/2) OD
        summa=summa+ V[i].w
    OD
    Добавление в СДП (Root, V[i])
    A2(Root, L, i-1)
    A2(Root, i+1, R)
FI
```

Пример. Рассмотрим процесс построения приближенного ДОП алгоритмом А2 для строки символов из предыдущего примера. Предварительно упорядочим символы по алфавиту.

Таблица 2 Упорядоченный набор вершин

А	В	Е	И	К	Л	Н	О	П	Р	Т	У
4	3	2	1	2	1	2	3	1	2	1	1

В качестве корня дерева выбираем вершину V_5=К, поскольку

$$\sum_{i=1}^{5-1} w_i = 4+3+2+1 < \frac{23}{2},\ \sum_{i=1}^{5} w_i = 4+3+2+1+2 \geq \frac{23}{2}.$$

Все вершины левее вершины V_5 образуют левое поддерево, вершины правее V_5 – правое поддерево. Далее алгоритм применяется для каждого из поддеревьев в отдельности. Посчитаем средневзвешенную высоту построенного дерева.

$$P=2\cdot1+3\cdot2+3\cdot2+4\cdot3+2\cdot3+2\cdot3+2\cdot3+1\cdot4+1\cdot4+1\cdot4+1\cdot4+1\cdot5=65$$

$$h_{ср}=65/23=2.82$$

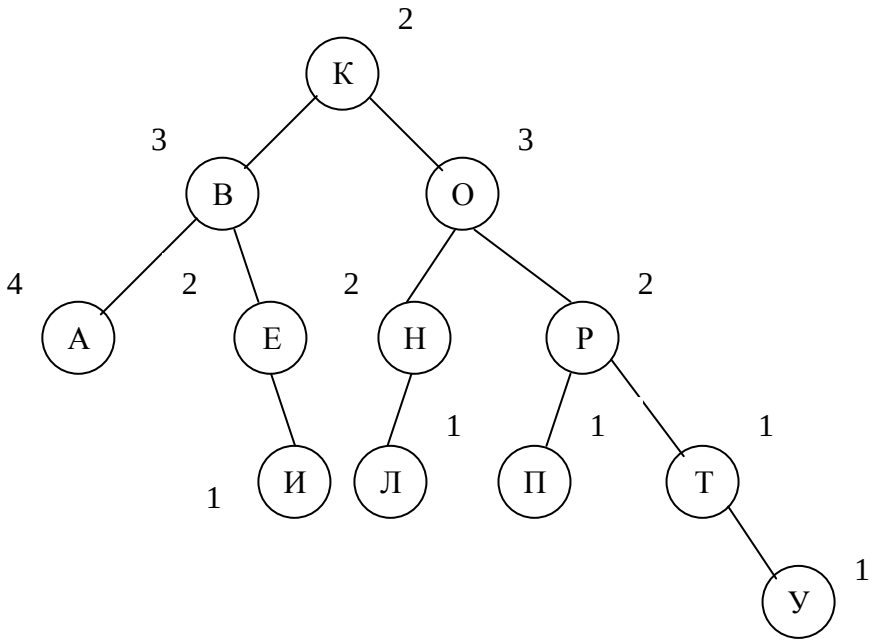


Рисунок 34 Дерево, построенное приближенным алгоритмом А2

Приведем некоторые свойства рассмотренных приближенных алгоритмов:

1) Сложность алгоритмов как функция от n (количество элементов) зависит следующим образом: время $T = O(n \log n)$, память $M = O(n)$ при $n \to \infty$. (Время определяется трудоемкостью сортировки элементов, а память – размером массива для хранения элементов)
2) Дерево, построенное приближенным алгоритмом А1, равносильно случайному (с точки зрения средней высоты) при $n \to \infty$, т.е. алгоритм А1– плохой.
3) Дерево, построенное приближенным алгоритмом А2, асимптотически приближается к оптимальному (с точки зрения средней высоты) при $n \to \infty$, т.е. алгоритм А2 является хорошим.
4) ИСДП нельзя считать даже приближением к дереву оптимального поиска.

6.4 Контрольные вопросы

1. Сформулируйте задачу построения ДОП.
2. Что такое ДОП?
3. Что такое средневзвешенная высота ДОП?
4. Сформулируйте основные свойства ДОП.
5. Назовите приближенные алгоритмы построения ДОП.
6. Назовите основные свойства приближенных алгоритмов ДОП.

ПРАВИЛА ВЫПОЛНЕНИЯ ЗАДАНИЙ НА ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Программы выполняются на языках высокого уровня (Паскаль, Си). По согласованию с преподавателем допускается работа в средах Delphi, Builder C++, Visual C++. Для защиты задания студенту необходимо представить

- Исходные тексты программ;

- Исполняемые файлы;
- Результаты работы программы

Защита задания включает в себя следующие разделы

- Формулировку задания;
- Описание основных методов и алгоритмов, используемых при выполнении задания;
- Анализ результаты работы программы.

Тестирование программ должно проводиться для различных случаев: упорядоченный массив (прямой и обратный порядок), случайный массив.

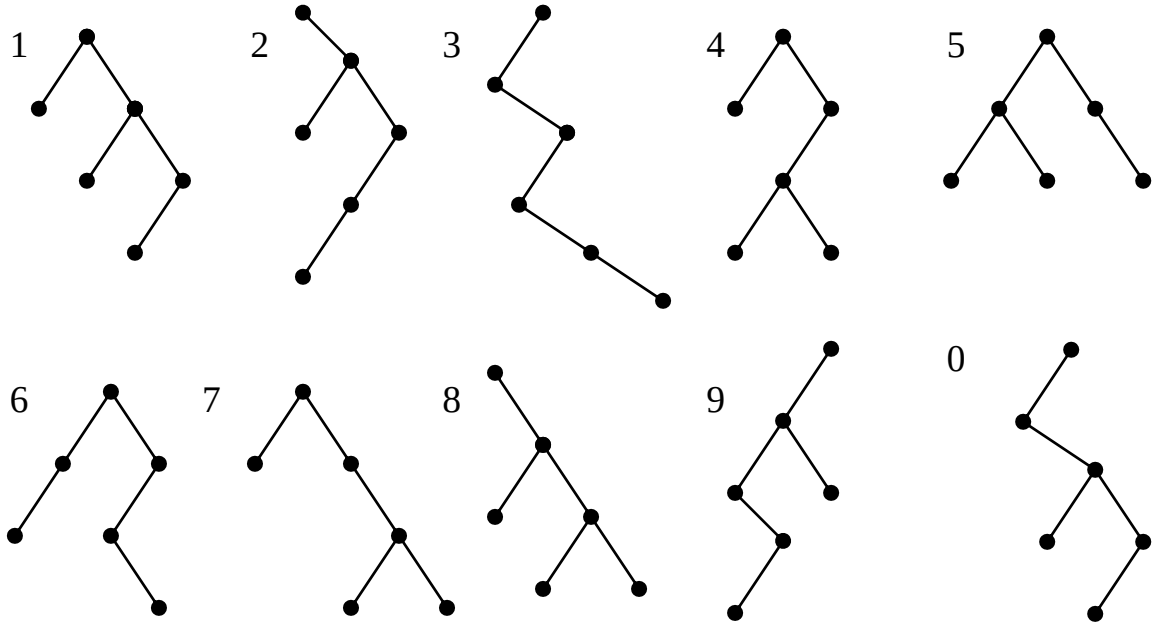
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1

Тема: Построение двоичного дерева. Вычисление характеристик дерева.

Цель работы: Освоить понятие двоичного дерева.

Порядок выполнения работы:

1. Разместить в памяти компьютера данное двоичное дерево (см. ниже, номер задания соответствует последней цифре шифра), данные в вершинах заполнить случайными числами.
2. Написать процедуры для вычисления размера дерева, высоты дерева, средней высоты дерева, контрольной суммы для дерева и проверить их работу на конкретном примере.
3. Запрограммировать обход двоичного дерева слева направо и вывести на экран получившуюся последовательность данных.



ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2

Тема: Построение случайного дерева поиска и идеально сбалансированного дерева поиска

Цель работы: Освоить методы построения случайного дерева поиска и идеально сбалансированного дерева поиска.

Порядок выполнения работы:

1. Разработать процедуры построения СДП и ИСДП.
2. Вычислить среднюю высоту построенных деревьев для n=10, 50, 100, 200, 400 (n –количество вершин в дереве). Заполнить таблицу следующего вида и проанализировать полученные результаты

N	Высота СДП	Высота ИСДП
10		
50		
100		
200		
400		

3. Написать процедуру, определяющую является ли двоичное дерево деревом поиска. Проверить ее работу на построенных СДП и ИСДП.
4. Запрограммировать процедуру поиска в дереве поиска элемента с заданным ключом и проверить ее работу на построенных СДП и ИСДП.
5. Определить количество операций, необходимых для поиска. Сравнить эту величину с высотой дерева.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 3

Тема: Построение АВЛ-дерева.

Цель работы: Освоить построение АВЛ-дерева.

Порядок выполнения работы:

1. Разработать процедуру построения АВЛ-дерева.
2. Вычислить среднюю высоту АВЛ-дерева для n=10, 50, 100, 200, 400 (n –количество вершин в дереве) и заполнить таблицу следующего вида. Проанализировать полученные результаты, сравнить их с теоретическими оценками и результатами из лабораторной работы 1.

N	Высота АВЛ-дерева	Теоретическая оценка

10		
50		
100		
200		
400		

3. Экспериментально определить среднее количество поворотов на одну включаемую вершину в АВЛ-дерево.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 4

Тема: Построение двоичного Б-дерева.

Цель работы: Освоить построение двоичного Б-дерева.

Порядок выполнения работы:

1. Разработать процедуру построения двоичного Б-дерева.
2. Вычислить среднюю высоту двоичного Б-дерева для n=10, 50, 100, 200, 400 (n –количество вершин в дереве) и заполнить таблицу следующего вида. Проанализировать полученные результаты, сравнить их с теоретическими оценками и результатами из лабораторной работы 3.

N	Высота ДБД	Теоретическая оценка
10		
50		
100		
200		
400		

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 5

Тема: Построение дерева почти оптимального поиска

Цель работы: Освоить методы построения ДОП приближенными методами.

Порядок выполнения работы:

1. Разработать процедуры построения ДОП приближенными методами A_1 и A_2.
2. Вычислить средневзвешенную высоту построенных ДОП для n=10, 50, 100, 200, 400 (n –количество вершин в дереве) и заполнить

таблицу следующего вида. Проанализировать полученные результаты, сравнить их между собой.

N	Средневз. высота Алгоритм A_1	Средневз. высота Алгоритм A_2
10		
50		
100		
200		
400		

КУРСОВОЙ ПРОЕКТ

ПРАВИЛА ВЫПОЛНЕНИЯ И ОФОРМЛЕНИЯ КУРСОВОГО ПРОЕКТА

При выполнении курсового проекта необходимо строго придерживаться указанных ниже правил. Работы, выполненные без соблюдения этих правил, не засчитываются и возвращаются студенту для переработки.

1. Курсовой проект выполняется на языке высокого уровня (Паскаль, Си). В работу должны быть включены все задачи, указанные в задании, строго по своему варианту. Курсовые проекты, содержащие не все задачи или задачи не своего варианта, не засчитываются.
2. В ходе работы должна быть создана программа, выполняющая поставленную задачу, и оформлен отчет, включающий в себя следующие разделы:
   - титульный лист;
   - постановка задачи для конкретного варианта;
   - краткое изложение основных идей и характеристик применяемых алгоритмов (сортировка, поиск) и структур данных;
   - распечатка текста программы;
   - распечатка результатов.
3. Для эащиты курсового проекта студенту необходимо предоставить
   - Исходные тексты программ;
   - Исполняемые файлы;
   - Результаты работы программы
4. Защита курсового проекта включает в себя следующие разделы
   - Формулировку задания;
   - Описание основных методов и алгоритмов, используемых при выполнении задания;

- Анализ результатов работы программы.

ЗАДАНИЕ ДЛЯ КУРСОВОГО ПРОЕКТА

Общая постановка задачи

1. Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде массива или списка (в зависимости от варианта), вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра.
2. Упорядочить данные в соответствии с заданным условием упорядочения, используя указанный метод сортировки. Упорядоченные данные вывести на экран.
3. Предусмотреть возможность быстрого поиска по ключу в упорядоченной базе, в результате которого из записей с одинаковым ключом формируется очередь, содержимое очереди выводится на экран.
4. Из записей очереди построить дерево поиска по ключу, отличному от ключа сортировки, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.
5. При выполнении задания главное внимание следует уделить эффективности применяемых алгоритмов, исключению всех лишних операций.
6. Операции, выражающие логически завершенные действия, рекомендуется оформлять в виде подпрограмм, грамотно выбирая между процедурами и функциями. Имена переменных и подпрограмм, параметры подпрограмм, используемые языковые конструкции должны способствовать удобочитаемости программы.
7. Для сравнения символьных строк не рекомендуется пользоваться встроенными языковыми средствами и библиотечными функциями.

ВАРИАНТЫ БАЗ ДАННЫХ (БД)

Общие замечания

1. Все текстовые поля следует рассматривать как символьные массивы (array of char), а не строки (string). Это сделано для совместимости между языками Паскаль и Си, а также из-за того, что в базах данных не принято хранить лишнюю информацию, такую как длина строки. Если длина поля превышает размер хранимой в нем информации, то оно дополняется пробелами справа. Каждое текстовое поле имеет свой формат, который определяет смысл записанных в него данных. При описании формата в угловых скобках < и > указываются отдельные его элементы (сами угловые скобки в состав текста не входят); пробелы обозначаются с помощью символа подчеркивания. Если

поле включает только один текстовый элемент, то формат не указывается.

2. Целочисленные поля представляются 16-разрядными положительными числами (типа word в Паскале).
3. При описании структуры записей в программах необходимо точно соблюдать порядок и размер полей.

ПРИМЕЧАНИЕ. Предварительный просмотр содержимого баз данных возможен с помощью программы VIEWBASE.EXE

Описание баз данных

B = 1 (файл base1.dat)

Библиографическая база данных "Жизнь замечательных людей"

Структура записи:

Автор: текстовое поле 12 символов
формат <Фамилия>_<буква>_<буква>

Заглавие: текстовое поле 32 символа
формат <Имя>_<Отчество>_<Фамилия>

Издательство: текстовое поле 16 символов

Год издания: целое число

Кол-во страниц: целое число

Пример записи из БД:

Кловский_В_Б
Лев_Николаевич_Толстой_____
Молодая_гвардия_
1963
864

Варианты условий упорядочения и ключи поиска (К):

C = 1 - по фамилиям замечательных людей, К = три первые буквы фамилии;

C = 2 - по году издания и автору, К = год издания;

C = 3 - по издательству и автору, К = три первые буквы издательства.

B = 2 (файл base2.dat)

База данных "Предприятие"

Структура записи:

ФИО сотрудника: текстовое поле 32 символа
формат <Фамилия>_<Имя>_<Отчество>

Номер отдела: целое число

Должность: текстовое поле 22 символа

Дата рождения: текстовое поле 8 символов
формат дд-мм-гг

Пример записи из БД:
  Петров_Иван_Иванович_____
  130
  начальник_отдела_____
  15-03-46

Варианты условий упорядочения и ключи поиска (К):
С = 1 - по номеру отдела и ФИО, К = номер отдела;
С = 2 - по дням рождения и ФИО, К = день рождения;
С = 3 - по дате рождения, К = год рождения.

В = 3 (файл base3.dat)
База данных "Обманутые вкладчики"
Структура записи:
  ФИО вкладчика: текстовое поле 32 символа
    формат <Фамилия>_<Имя>_<Отчество>
  Сумма вклада: целое число
  Дата вклада: текстовое поле 8 символов
    формат дд-мм-гг
  ФИО адвоката: текстовое поле 22 символа
    формат <Фамилия>_<буква>_<буква>
Пример записи из БД:
  Петров_Иван_Федорович_____
  130
  15-03-46
  Иванова_И_В_____

Варианты условий упорядочения и ключи поиска (К):
С = 1 - по ФИО и сумме вклада, К = первые три буквы фамилии;
С = 2 - по сумме вклада и дате, К = сумма вклада;
С = 3 - по ФИО адвоката и сумме вклада, К = первые три
  буквы фамилии адвоката.

В = 4 (файл base4.dat)
База данных "Населенный пункт"
Структура записи:
  ФИО гражданина: текстовое поле 32 символа
    формат <Фамилия>_<Имя>_<Отчество>
  Название улицы: текстовое поле 20 символов
  Номер дома: целое число
  Номер квартиры: целое число
  Дата поселения: текстовое поле 8 символов
    формат дд-мм-гг

Пример записи из БД:

Петров_Иван_Федорович_____
Ленина_____
10
67
29-02-65

Варианты условий упорядочения и ключи поиска (К):

С = 1 - по ФИО и названию улицы, К = первые три буквы фамилии;

С = 2 - по названию улицы, номеру дома и ФИО, К = первые три буквы названия улицы;

С = 3 - по дате поселения и названию улицы, К = год поселения.

Варианты методов сортировки

S = 1 Метод пирамидальной сортировки
Файл базы данных загружается в динамическую память с формированием индексного массива как массива указателей.

S = 2 Метод Хоара
Файл базы данных загружается в динамическую память с формированием индексного массива как массива указателей.

S = 3 Метод прямого слияния
Файл базы данных загружается в динамическую память, сортировка проводится с использованием очередей, для проведения поиска строится индексный массив.

S = 4 Цифровая сортировка
В качестве ключа для упорядочения нужно взять всего по нескольку (обычно не менее трех) байт из соответствующих полей. Файл базы данных загружается в динамическую память в виде очереди. Затем очередь сортируется цифровым методом. На основе упорядоченной очереди строится индексный массив.

Типы деревьев поиска

D = 1 АВЛ-дерево
D = 2 Двоичное Б-дерево
D = 3 Дерево оптимального поиска (приближенный алгоритм)

ПРАВИЛА ВЫБОРА ВАРИАНТА

Вариант задания задается с помощью чисел B, C, S, D, где

B - номер базы данных;

C - вариант условия упорядочения для этой базы данных;

S - метод сортировки;

D - тип дерева поиска.

Ключ поиска указывается вместе с условием упорядочения и, как правило, представляет собой упрощенный вариант ключа сортировки. Числа B, C, S, D определяются с помощью таблицы соответствия вариантов, приведенной ниже. Каждый студент разрабатывает программу для одного варианта. Допускаются различные творческие дополнения, ведущие в сторону развития. Выполнение работы по чужому варианту не допускается.

Таблица соответствия вариантов

Номер шифра	B	C	S	D
1	1	1	1	1
2	1	1	2	2
3	1	1	3	3
4	1	1	4	1
5	1	2	1	2
6	1	2	2	3
7	2	2	3	1
8	2	2	4	2
9	2	3	1	3
10	2	3	2	1
11	2	3	3	2
12	2	3	4	3
13	3	1	1	1
14	3	1	1	1
15	3	1	3	3
16	3	1	4	1
17	3	2	1	2
18	3	2	2	3
19	4	2	3	1
20	4	2	4	2
21	4	3	1	3
22	4	3	2	1
23	4	3	3	2
24	4	3	4	3
25	4	1	1	1

ПРИЛОЖЕНИЕ А

Псевдокод для записи алгоритмов

Для записи алгоритма будем использовать специальный язык – псевдокод. Алгоритм на псевдокоде записывается на естественном языке с использованием двух конструкций: ветвления и повтора. В круглых скобках будем писать комментарии. В треугольных скобках будем описывать действия, алгоритм выполнения которых не требует детализации, например, <обнулить массив>.

:= Операция присваивания значений.

⟷ Операция обмена значениями.

Конструкции ветвления.

1. IF (условие) Если выполняется условие,
<действие> то выполнить действие
FI FI указывает на конец этих действий.

2. IF (условие)
<действия 1>
ELSE <действия 2> Действия 2 выполняются,
FI если неверно условие.

3. IF (условие1)
<действия1>
ELSEIF (условие2) Действия 2 выполняются,
<действия2> если неверно условие1 и верно условие 2
…FI

Конструкции повтора.

1. Цикл с предусловием.
DO (условие) Действия повторяются
<действия> пока условие истинно.
OD OD указывает на конец цикла.

2. Цикл с постусловием.
DO <действия>
OD (условие выполнения)

3. Цикл с параметром.
DO (i=1, 2, ... n) Действия выполняются для значений
<действия> параметра из списка
OD

4. Бесконечный цикл.
DO
<действия>
OD

5. Принудительный выход из цикла.
DO
...IF (условие) OD Если условие истинно, то выйти из цикла.
OD