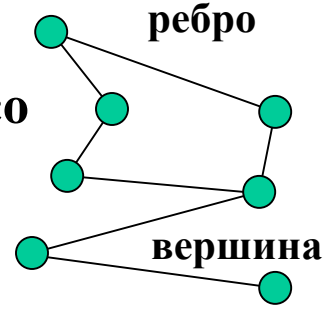
АТД «Словарь»

- Реализация группы операций над данными:
  - хранение
  - добавление
  - удаление
  - поиск (может быть удачным и неудачным)
  - сортировка
  - поиск минимального (максимального, среднего)
  - …

Поиск. Способы хранения данных

Способ	Худший случай			Средний случай		
	Вставка	Поиск	Выбор	Вставка	Поиск +	Поиск –
Массив, индексированный ключами	O(1)	O(1)	O(M)	O(1)	O(1)	O(1)
Упорядоченный массив	O(N)	O(N)	O(1)	O(N/2)	O(N/2)	O(N/2)
Упорядоченный список	O(N)	O(N)	O(N)	O(N/2)	O(N/2)	O(N/2)
Неупорядоченный массив	O(1)	O(N)	O($N\log_2 N$)	O(1)	O(N/2)	O(N)
Неупорядоченный список	O(1)	O(N)	O($N\log_2 N$)	O(1)	O(N/2)	O(N)
Бинарный поиск	O(N)	O($\log_2 N$)	O(1)	O(N/2)	O($\log_2 N$)	O($\log_2 N$)
Бинарное дерево	O(N)	O(N)	O(N)	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)
Рандомизированное дерево	O(N)*	O(N)*	O(N)*	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)
Сбалансированное дерево	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)	O($\log_2 N$)
Хеш-таблица	O(1)	O(N)*	O($N\log_2 N$)	O(1)	O(1)	O(1)

Деревья. Основные понятия



Граф – это конечное множество вершин (узлов) вместе со множеством ребер (дуг), соединяющих пары различных вершин.

Дерево – это связный ациклический граф.

Дерево – это или пустое множество вершин, или [ориентированный] ациклический граф, в котором существует одна вершина, в которую не входит ни одна дуга, в каждую из оставшихся входит ровно одна дуга и существует единственный путь из корня в любую вершину.

Корневое дерево – это множество узлов, среди которых существует один особый узел (корень), а все остальные делятся на n подмножеств T_i, каждое из которых является деревом. Множества T_1, …, T_n – называются поддеревьями корня. Если порядок существенен [и организован], то дерево называется упорядоченным.

Деревья. Основные понятия. Седжвик

Дерево **– это непустая коллекция верши и ребер. удовлетворяющих определенным требованиям.** ***Вершина*** **– это простой объект, с которым может быть связана какая-либо информация;** ***ребро*** **– это связь между двумя вершинами.** ***Путь*** **в дереве – это список конкретных вершин, в котором следующие друг за другом вершины соединяются ребрами дерева.**

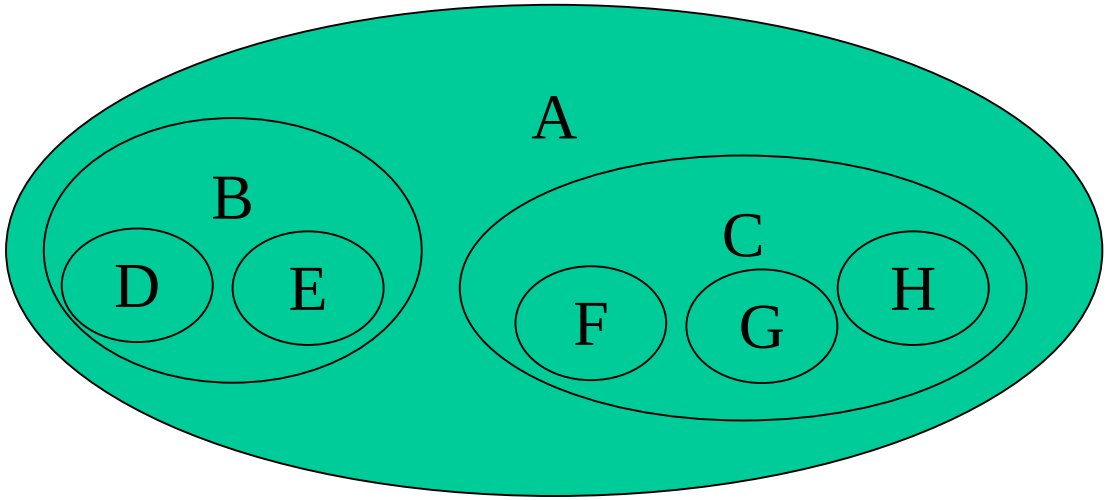
Дерево с корнем **(корневое) – это дерево, в котором один узел назначен корнем. Свободные деревья.**

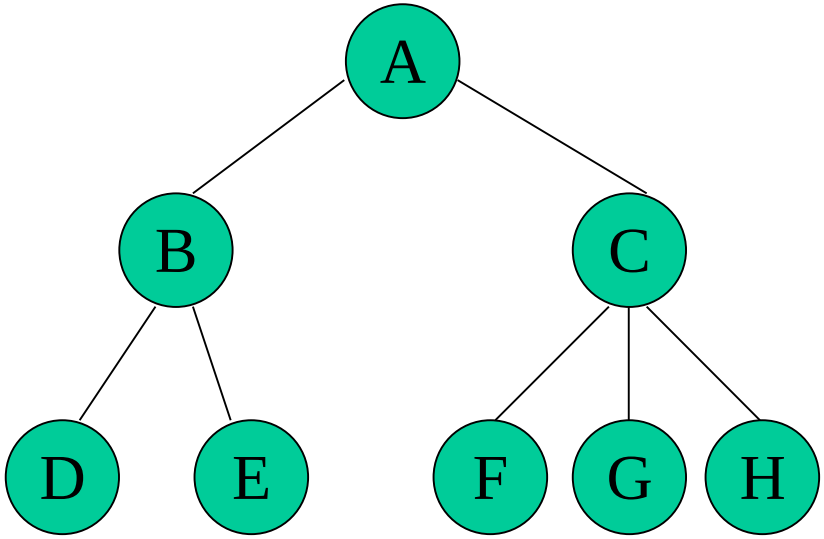
Существует ***единственный путь*** **между корнем или любым другим узлом. Направление не важно, обычно от корня. Каждый узел, кроме корня имеет** ***родительский узел*****. Дочерние узлы, потомки. Листья или терминальные узлы. Нетерминальные узлы – это узлы, имеющие хотя бы одного потомка. Упорядоченное дерево – это корневое дерево, в котором определен порядок следования дочерних узлов каждого узла.**

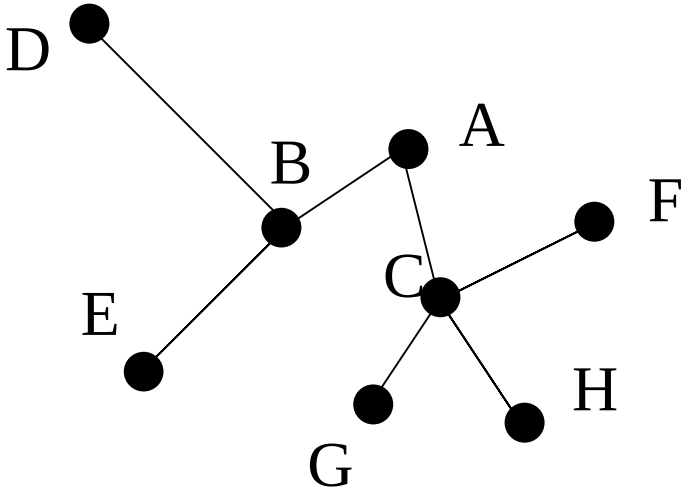
Если каждый узел должен иметь конкретно количество дочерних, то говорят об ***М-арном дереве*****.** ***Внешние*** **узлы не имеют дочерних. Внешние узлы могут выступать как** ***фиктивные,*** **на которые ссылаются узлы, не имеющие заданного количества дочерних узлов.**

Лист **в М-рном дереве – это внутренний узел, все дочерние узлы которого являются внешними.**

Деревья. Графическое представление







Является ли структура «дерево» – АТД?

Бинарные деревья. Основные понятия.

Седжвик

Бинарное дерево **– это упорядоченное корневое дерево, которое либо внешний узел, либо внутренний, связанный с парой бинарных деревьев, называемых левым и правым поддеревьями этого узла.**

Бинарное дерево **с N внутренними узлами имеет:**

$N + 1$ внешних узлов;

$2N$ связей, из которых $N - 1$ связей с внутренними узлами и $N + 1$ связей с внешними узлами;

длина внешнего пути = длина внутреннего пути + $2N$;

$\log_2 N$ <= высота дерева <= $N - 1$;

$N\log_2(N/4)$ <= длина внутреннего пути <= $N(N-1)/2$

….

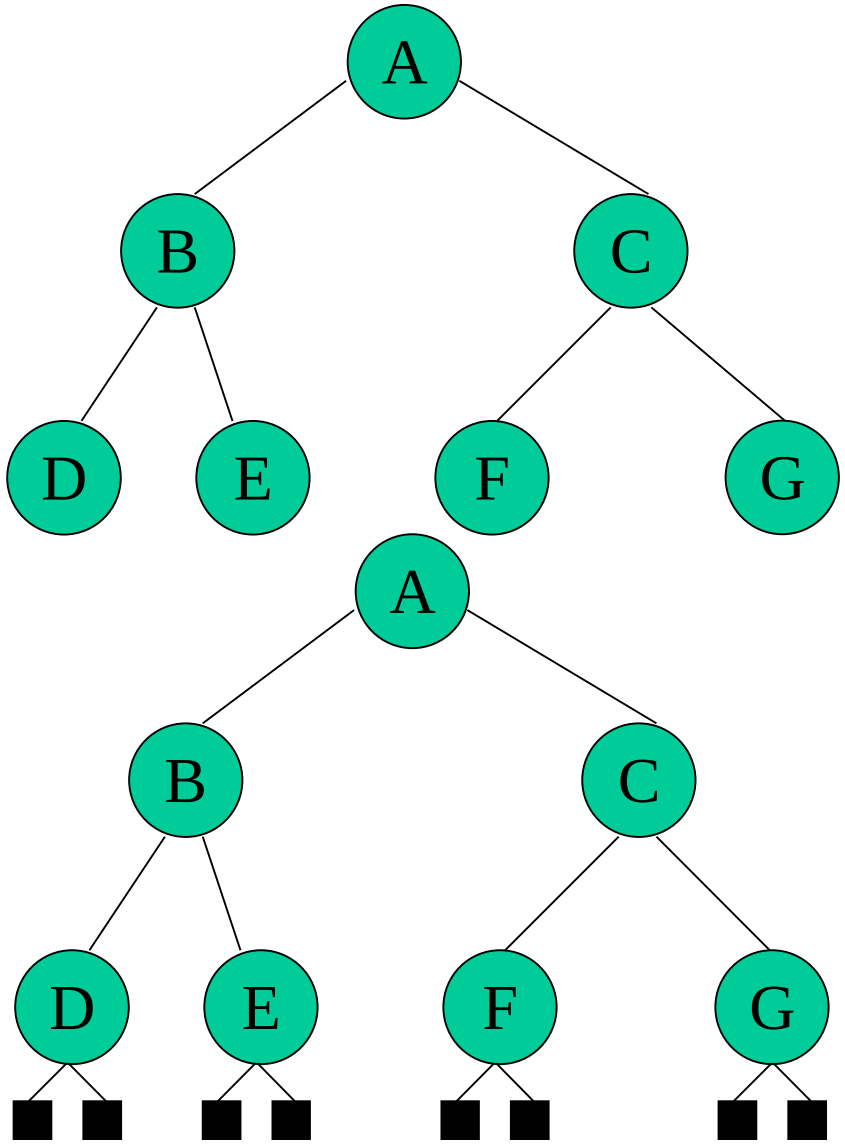
Уровень узла в дереве на единицу выше уровня родительского узла (уровень корня – 0). ***Высота*** **дерева – это максимальный уровень.** ***Длина пути*** **дерева – это сумма уровней всех узлов дерева.** ***Длина внутреннего (внешнего)*** **пути дерева – это сумма уровней всех внутренних (внешних) узлов дерева.**

Представление деревьев

- ссылочные структуры
- массив (см. «пирамида»)
- упакованный массив
- польская запись
- любые способы представления графов (но все эффективны) ! (см. далее)

Ссылочные структуры

```
struct Node {
Item Data;
struct Node * Child0;
struct Node * Child1;
};
/*--------------------*/
struct Node {
Item Data;
struct Node ** Childs;
};
struct Node {
Item Data;
struct Node * Left;
struct Node * Right;
};
```



Упакованный массив

Все располагаются в массиве так, что все узлы поддерева данного узла расположены вслед за этим узлом.

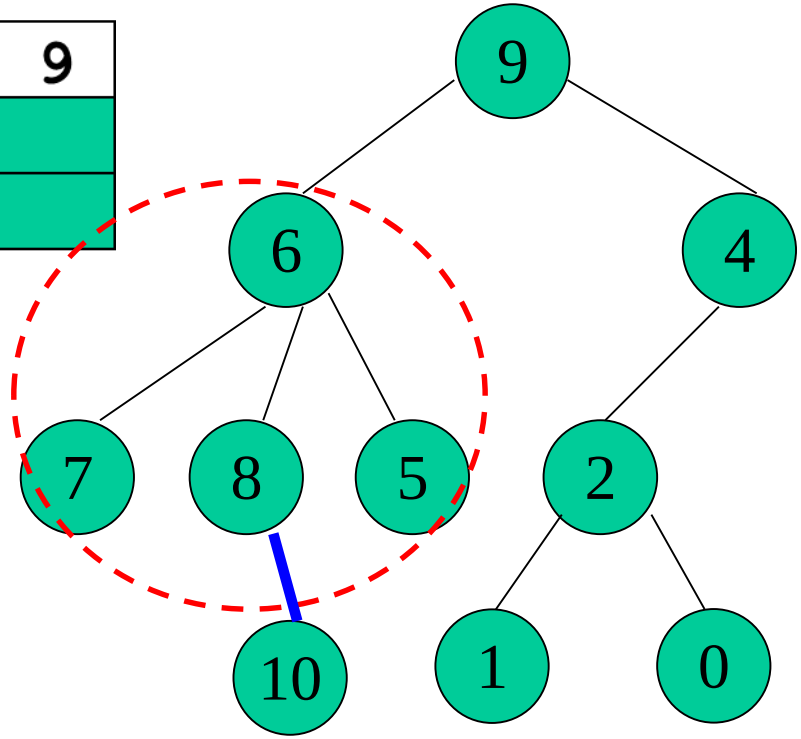
Вместе с каждым узлом хранится информация о последнем узле поддерева данного узла.

```
struct {
Item Data;        //например, целое число
int LastNodeID; //индекс узла, последнего в
                  //данном поддереве
};
```

0	1	2	3	4	5	6	7	8	9
9	6	7	8	5	4	2	1	0	
8	4	2	3	4	8	8	7	8	

Графическое представление упакованного массива

0	1	2	3	4	5	6	7	8	9
9	6	7	8	5	4	2	1	0	
8	4	2	3	4	8	8	7	8	



Польская запись

Все располагаются в массиве так, что все узлы поддерева данного узла расположены вслед за этим узлом.

Вместе с каждым узлом хранится информация о типе узла, т.е. о количестве типе связей: потомков нет, один левый или правый потомок (различают для деревьев поиска), оба потомка.

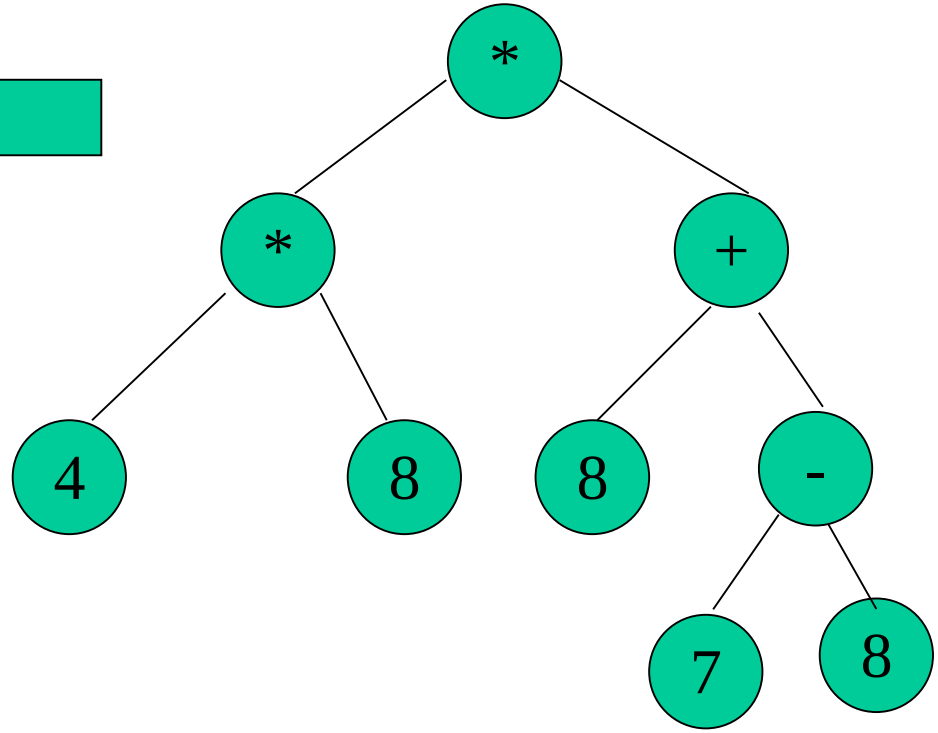
```
struct {
Item Data;
char Childs;
};
```

Если информация о количестве потомков прямо следует из значения узла, то можно «тип» избыточен.

```
* * 4 8 + 8 - 7 8
```

Графическое представление польской записи

```
* * 4 8 + 8 - 7 8
```




```
(* (* 4 8) (+ (8 (- 7 8))))
```

Обход

Обход [элементов] некоторой структуры данных – *систематическое* перечисление её элементов (узлов).

Перечисление может являться основой для решения каких-либо конкретных задач:

- **«сериализация» (например, вывод на экран, в файл, массив и пр. в определенной последовательности),**
- **подсчет количества элементов (возможно, обладающих какими-либо свойствами),**
- **изменение свойств/значений всех или некоторых элементов, обладающих определенными свойствами**
- **и др.**

Обход списка

```
struct Item {
Key Data;
struct Item * Next;
};

void VisitAllItems(struct Item * Head) {
// тело функции
}
```

Обход списка

```
struct Item {
Key Data;
struct Item * Next;
};

void VisitAllItems(struct Item * Head) {
while (Head) {
     DoSomething(Head);
     Head = Head->Next;
     }
}
```

Бинарные деревья. Обход

Бинарное дерево **– это упорядоченное корневое дерево, которое либо *внешний* узел («пустой», «фиктивный»), либо внутренний, связанный с парой бинарных деревьев, называемых левым и правым поддеревьями этого узла.**

Обход дерева **– систематическое перечисление его элементов (узлов).**

Обход *в глубину*
(рекурсия, АТД «Стек»)

- ***Прямой*** **обход (узел, левое поддерево, правое поддерево.**
- ***Симметричный*** **обход (левое поддерево, узел, правое поддерево).**
- ***Обратный*** **обход (левое поддерево, правое поддерево, узел).**

обход *в ширину*
(АТД очередь)

- ***Обход по уровням*** **– сверху вниз, слева направо.**

Обход списка

```
struct Item {
Key Data;
struct Item * Next;
};

void VisitAllItems(struct Item * Head) {
if (Head) {
     DoSomething(Head);
     VisitAllItems(Head->Next);
     }
}
```

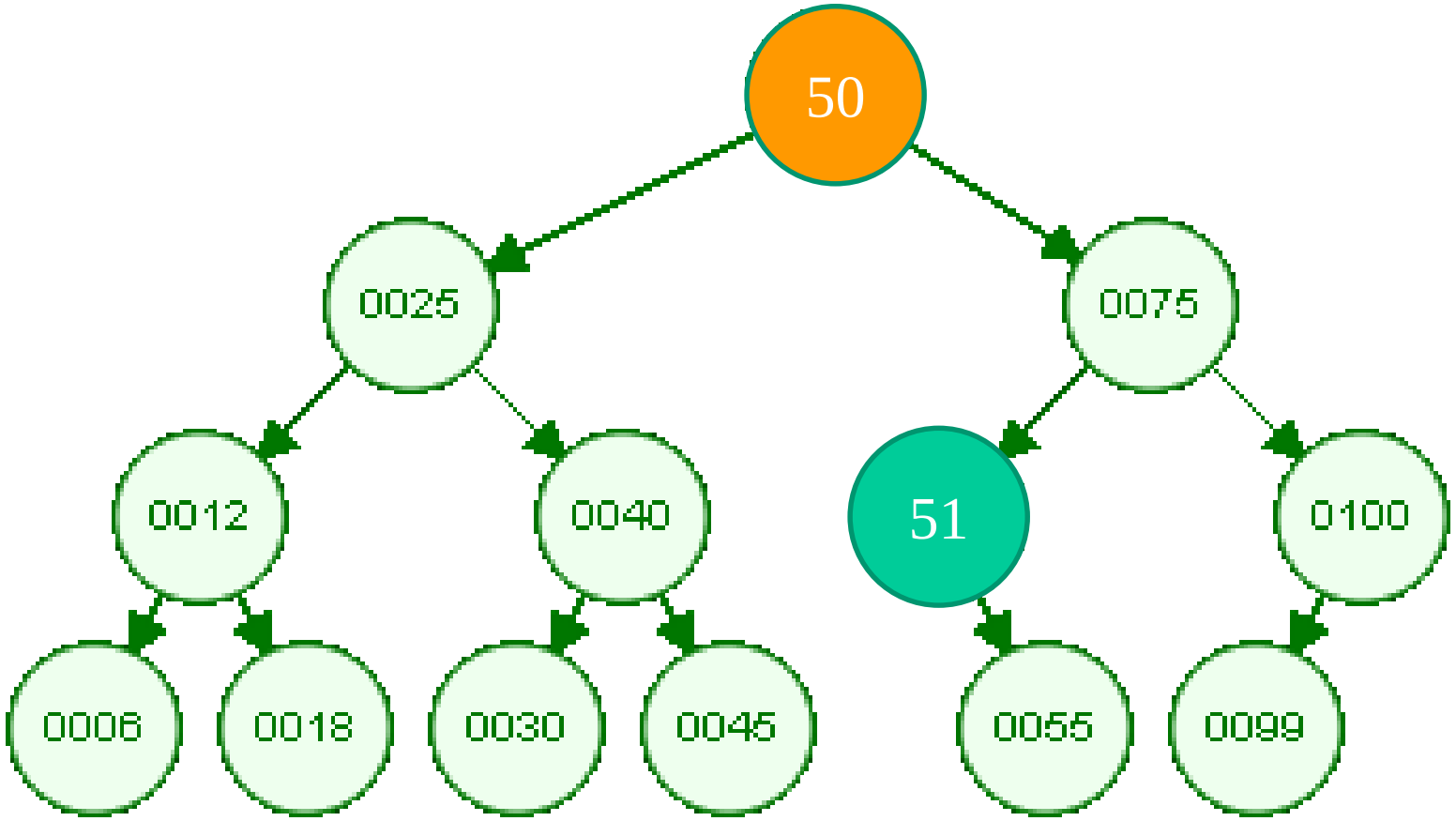
Обход дерева

```
struct Item {
Key Data;
struct Item * Left;
struct Item * Right;
};

void VisitAllItems(struct Item * Root) {
//тело функции
}
```

Обход дерева

```
struct Item {
Key Data;
struct Item * Left;
struct Item * Right;
};

void VisitAllItems(struct Item * Root) {
if (Root) {
     DoSomething(Root);//print("%d ", Root->Data);
     VisitAllItems(Root->Left);
     VisitAllItems(Root->Right);
     }
}
```

Обход дерева



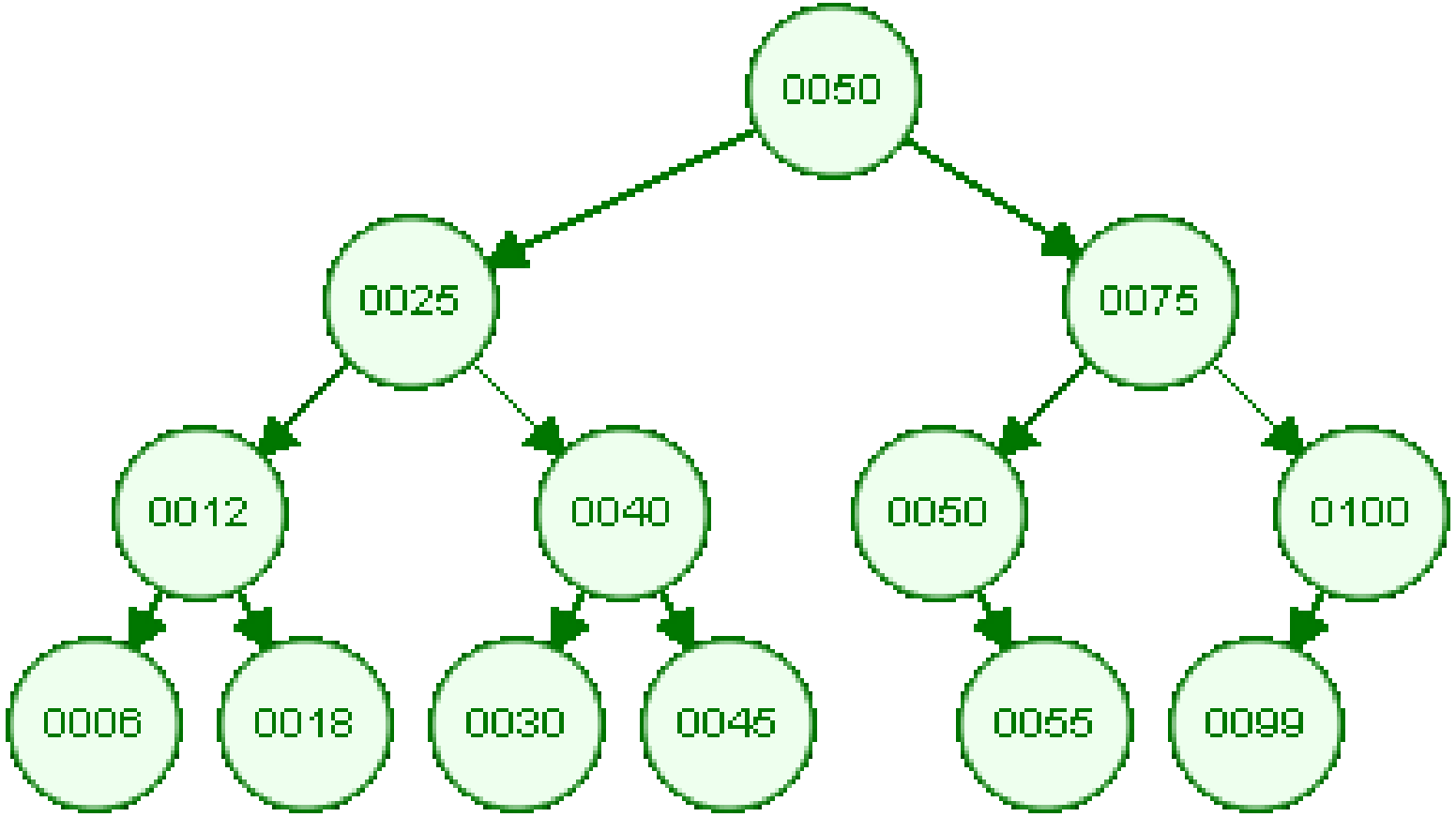
Прямой:50 25 12 6 18 40 30 45 75 51 55 100 99

Симметричный:6 12 18 25 30 40 45 50 51 55 75 99 100

Обратный:6 18 12 30 45 40 25 55 51 99 100 75 50

В ширину:50 25 75 12 40 51 100 6 18 30 45 55 99

Обход дерева



Прямой: 50 25 12 6 18 40 30 45 75 50 55 100 99

Симметричный: 6 12 18 25 30 40 45 50 50 55 75 99 100

Обратный: 6 18 12 30 45 40 25 55 50 99 100 75 50

В ширину:

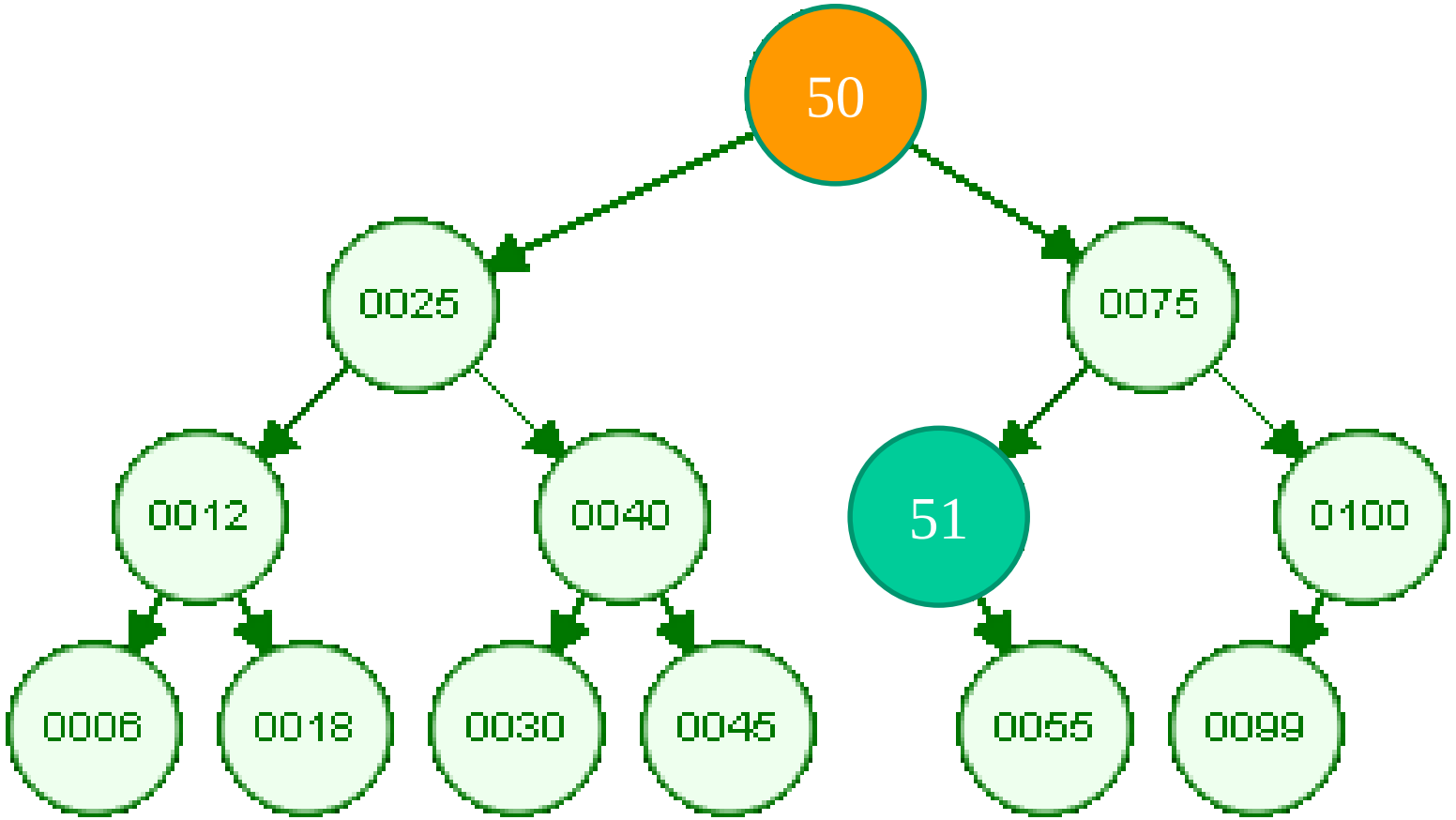
Обход дерева

```
struct Item {
Key Data;
struct Item * Left;
struct Item * Right;
};
Container C;//Count(/**/); Push(/**/); Pop(/**/);

void VisitAllItems(struct Item * Root) {
//тело функции
}
```

Обход дерева

```
Container C; //Count(); Push(/**/); Pop(/**/);

void VisitAllItems(struct Item * Root) {
if (Root) Push(C, Root);

while (Count(C)) {
  Root = Pop(C);
  DoSomething(Root);
  if (Root->Left)  Push(C, Root->Left);
  if (Root->Right) Push(C, Root->Right);
  }
}
```

Обход дерева



Прямой:50 25 12 6 18 40 30 45 75 51 55 100 99

Симметричный:6 12 18 25 30 40 45 50 51 55 75 99 100

Обратный:6 18 12 30 45 40 25 55 51 99 100 75 50

В ширину:50 25 75 12 40 51 100 6 18 30 45 55 99

Бинарное дерево поиска. Определение

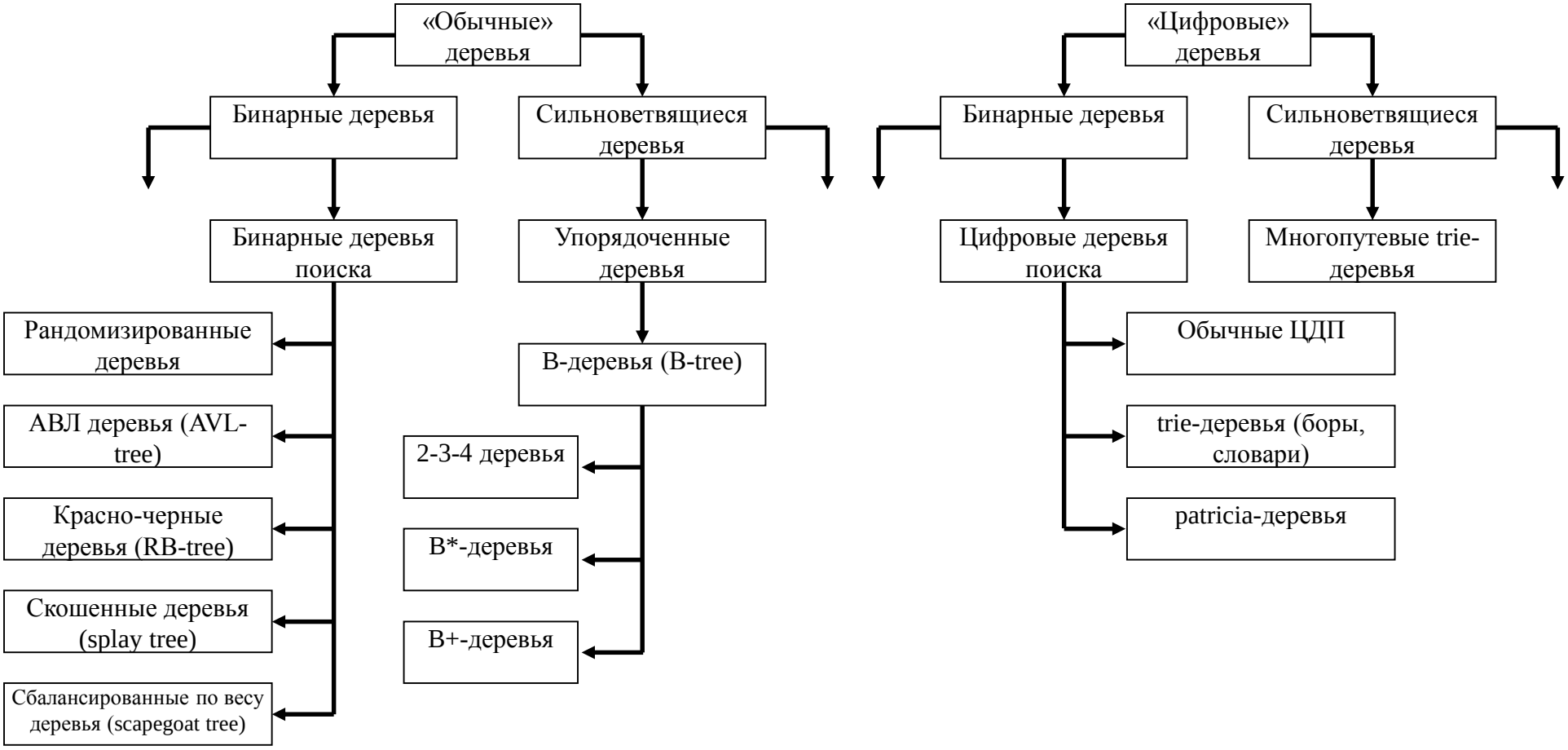
Бинарное дерево поиска – это бинарное дерево, с каждым из внутренних узлов которого связан ключ, причем значение этого ключа больше (или равно) ключей во всех узлах левого поддерева этого и меньше (или равно) ключей во всех узлах правого поддерева. Основные операции:

- **поиск**
- **вставка**
- **удаление:**

1. **Если удаляемый узел – лист, то удаление тривиально.**
2. **Если у удаляемого узла один потомок (одно поддерево), то узел удаляется, а его поддерево «подтягивается».**
3. **Если существуют оба поддерева, то необходимо найти узел с минимальным ключом в правом поддереве, поменять его удаляемым и затем выполнить удаление. Узел с минимальным справа ключом гарантировано будет иметь не более одного поддерева (правого), поэтому дальнейшее удаление будет соответствовать пунктам 1 или 2.**

В худшем случае асимптотическая сложность операций вставки, поиска, удаления может быть равна $O(N)$, вместо средней $O(\log_2 N)$.

Сбалансированные и цифровые деревья поиска



Сбалансированные деревья поиска

Бинарное дерево называется *выровненным*, если все листья находятся на одном уровне.

Бинарное дерево называется *заполненным* (*идеально сбалансированным*), если все узлы, имеющие меньше двух потомком находятся на одном или двух уровнях. Для заполненного дерева

$\log_2(N+1) - 1 <=$ высота дерева $< \log_2(N+1)$;

Бинарное дерево называется *полным*, если все листья находятся на одном уровне и все узлы, кроме листьев имеют ровно двух потомков:

$\log_2(N+1) - 1 =$ высота дерева.

Заполненные **деревья наиболее эффективны при поиске, но вставка/удаление узла могут потребовать перестройки всего дерева, поэтому асимптотическая сложность в худшем случае может быть равна $O(N)$.**

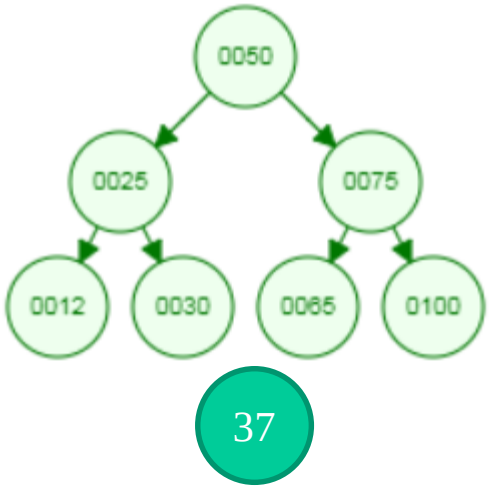
Рандомизированные бинарные деревья поиска

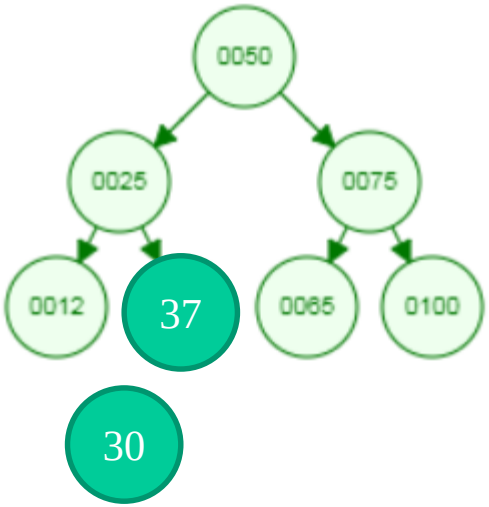
Построение основывается на двух операциях

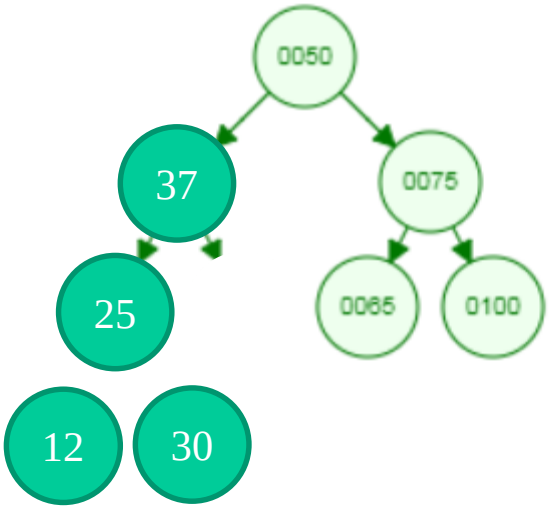
1. **«вращение» («поворот»)**
2. **вставка в корень**

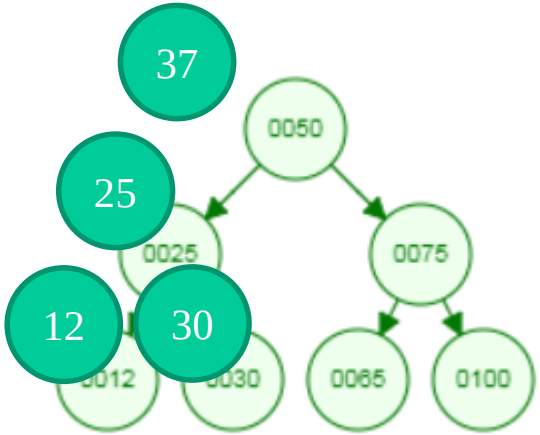
Обобщенный алгоритм:

случайным образом чередуется вставка узлов в листья и в корень – с вероятностью, обратно пропорциональной количеству узлов в текущем поддереве, осуществляется вставка в его корень.


0050
0025
0075
0012
0030
0065
0100
37


0050
0025
0075
0012
37
0065
0100
30


0050
37
0075
25
0065
0100
12
30


37
0050
25
0025
0075
12
30
0012
0030
0065
0100

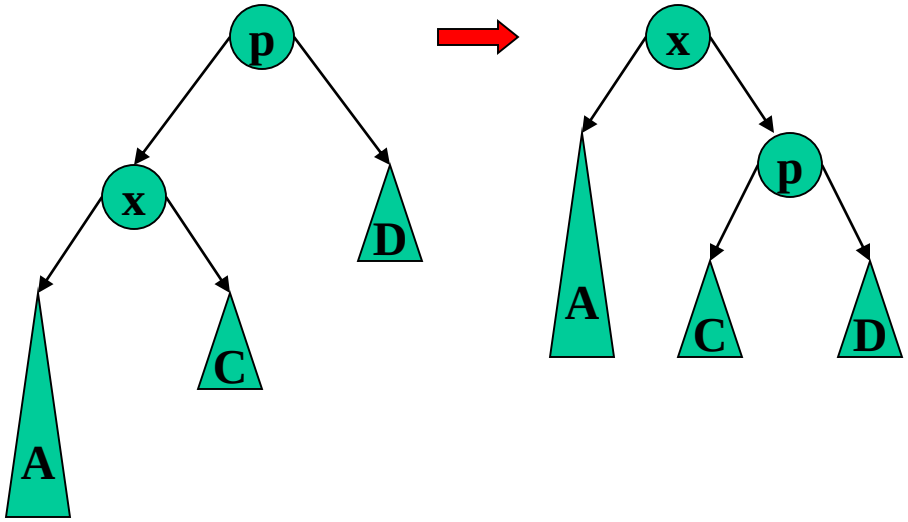
АВЛ-деревья

Разность высот поддеревьев любого узла не превышает 1. Балансировка осуществляется на основе операций одинарного и двойного поворотов.

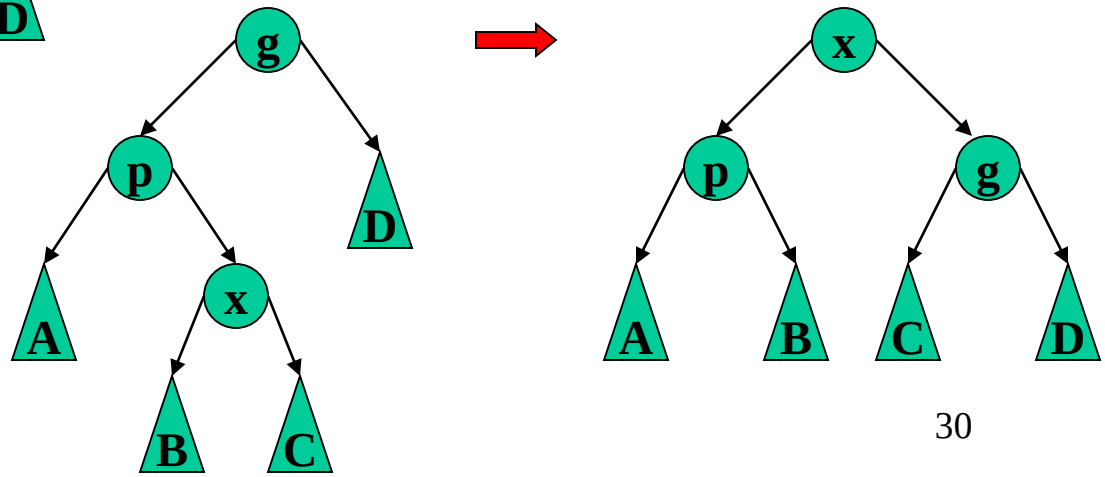
высота дерева <= $2\log_2 N$

(более точная оценка – высота дерева <= $\log_{(sqrt(5)+1)/2} 2 = 1{,}44\log_2 N$)

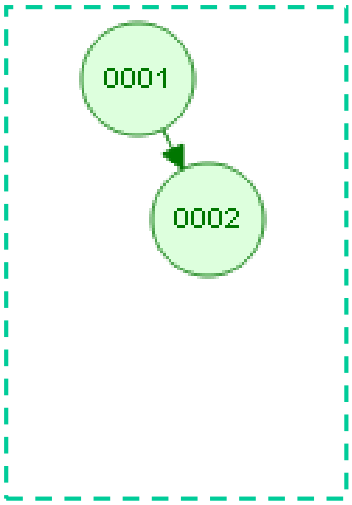
Одинарный поворот

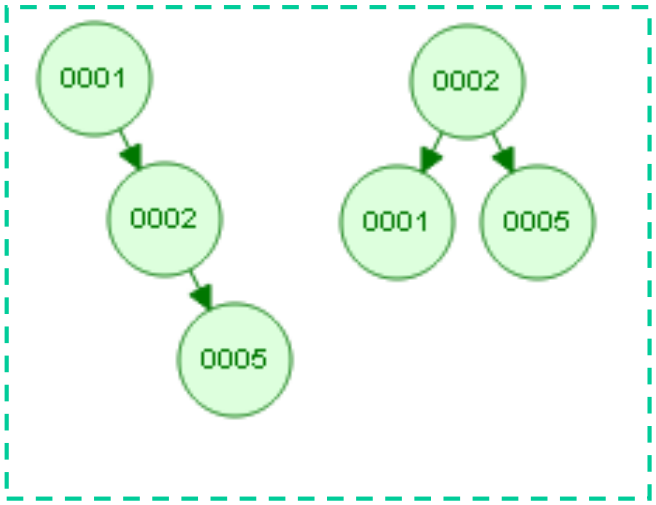


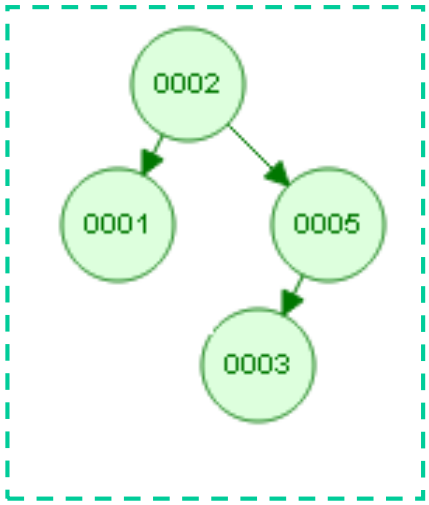
Двойной поворот

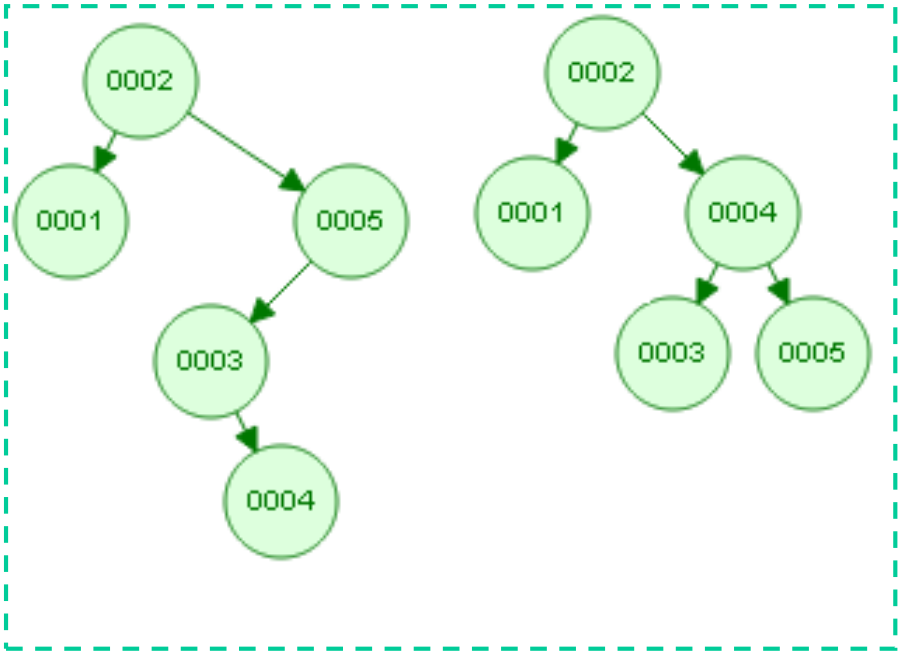


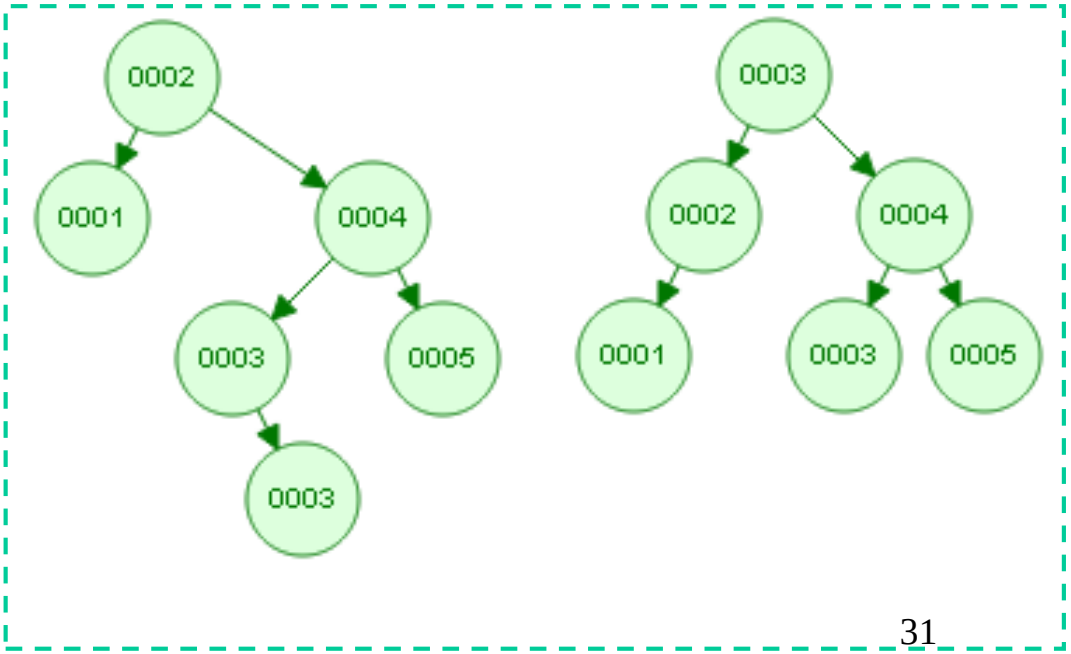
АВЛ-деревья


0001
0002


0001
0002
0002
0001
0005
0005


0002
0001
0005
0003


0002
0001
0005
0003
0004
0002
0001
0004
0003
0005


0002
0001
0004
0003
0005
0003
0003
0002
0004
0001
0003
0005

Красно-черные деревья

- **каждый узел дерева либо красный, либо черный**
- **каждый внешний узел – черный**
- **если узел красный, то его потомки – черные**
- **количество черных узлов, встречающихся в любом пути от корню к листьям – одинаковое (называется «черной» высотой).**

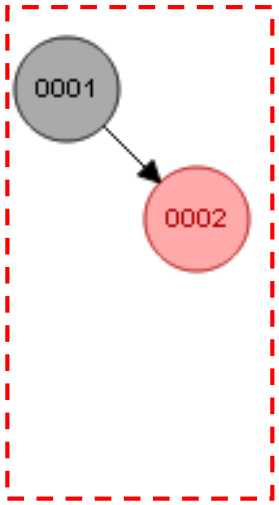
Длина максимального пути не более, чем в 2 раза больше любого другого в дереве.

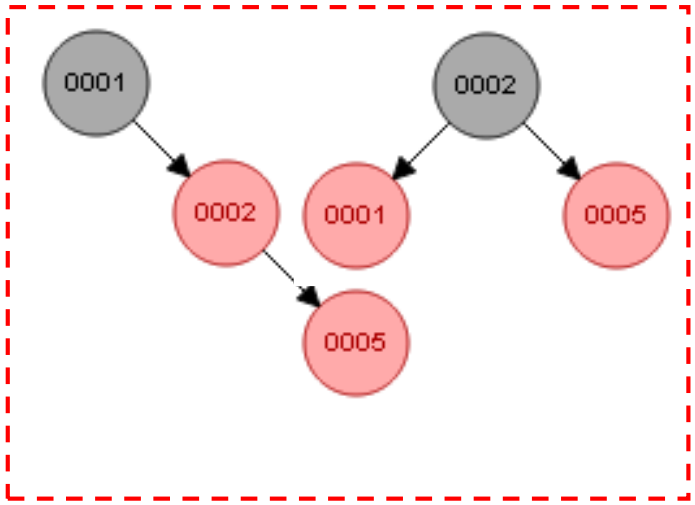
Средняя высота $1{,}002\log_2 N$, в худшем случае – $2(\log_2 N+1)$.

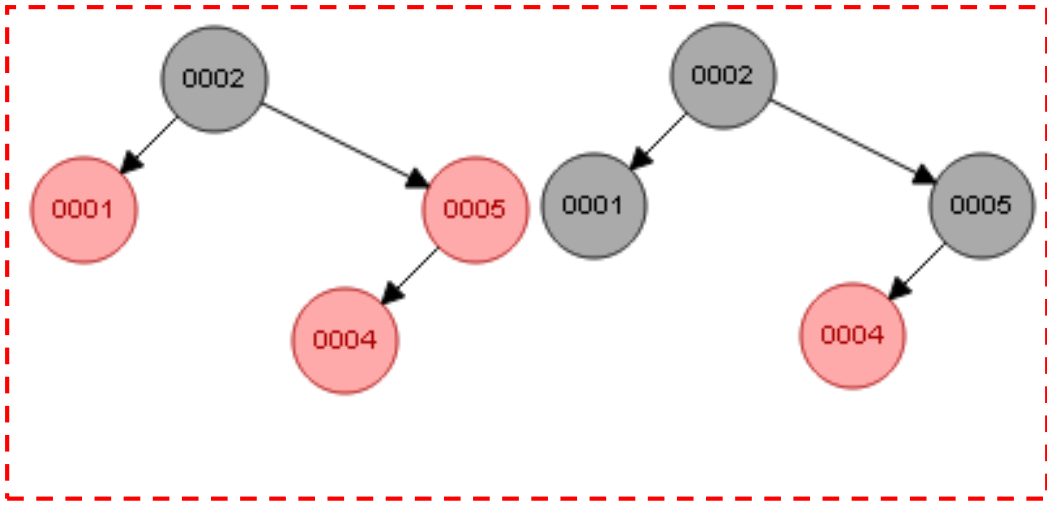
Обеспечение свойств при вставке/удалении достигается путем

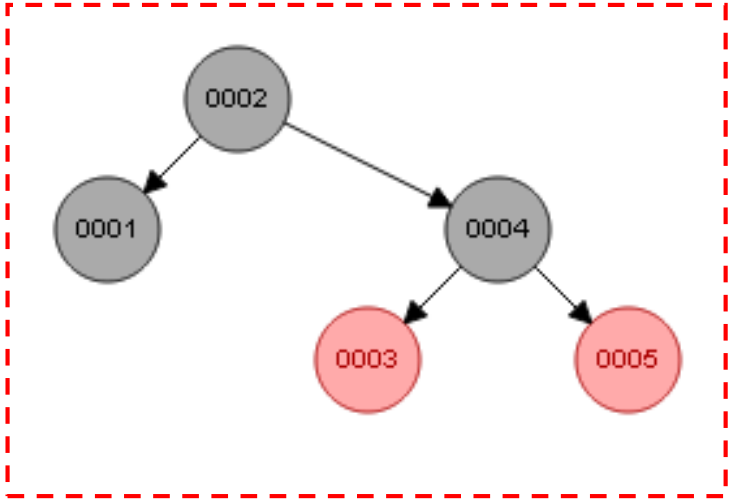
- **изменения цвета узлов**
- **вращений**

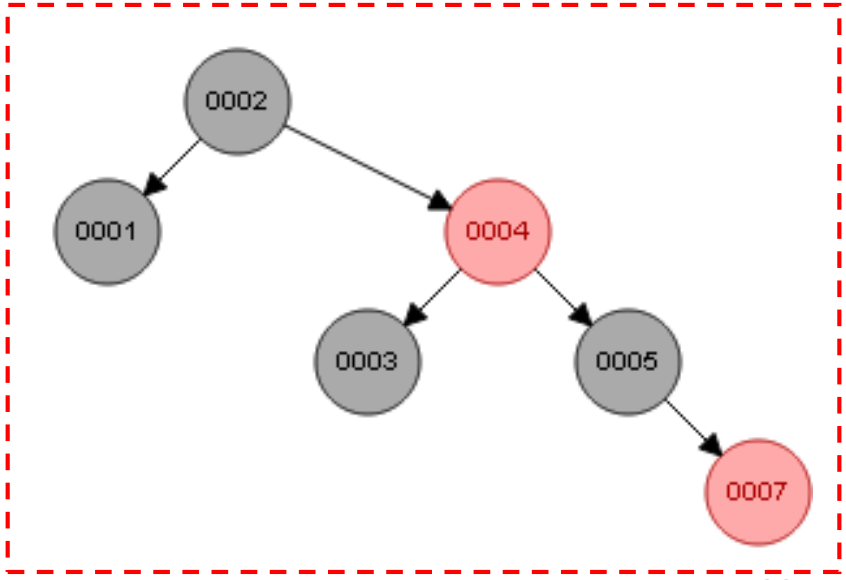
Красно-черные деревья





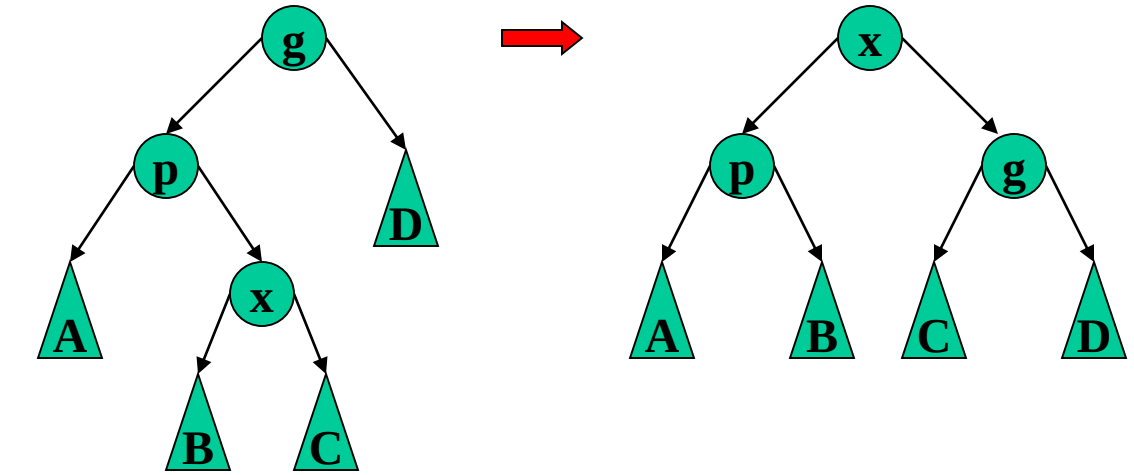


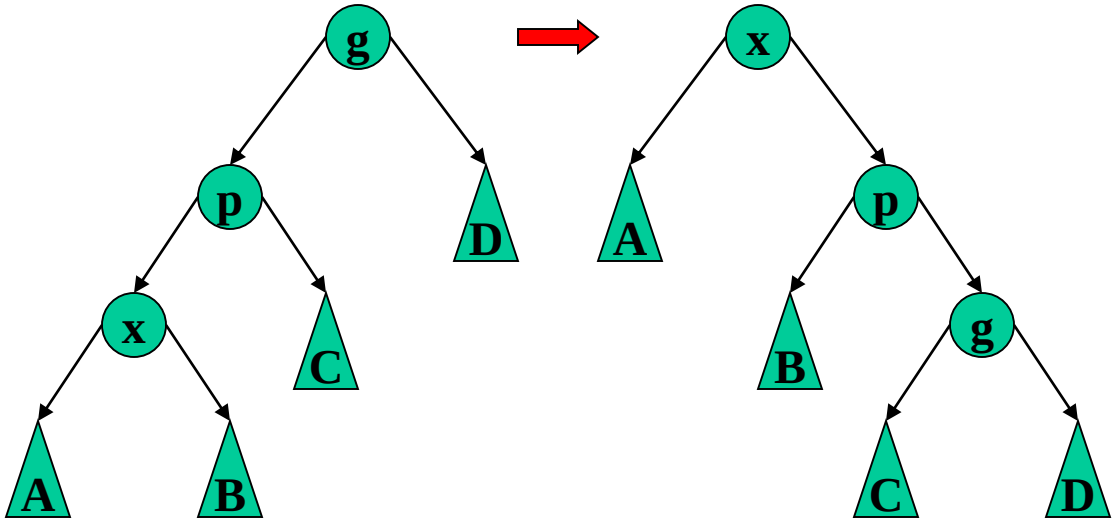


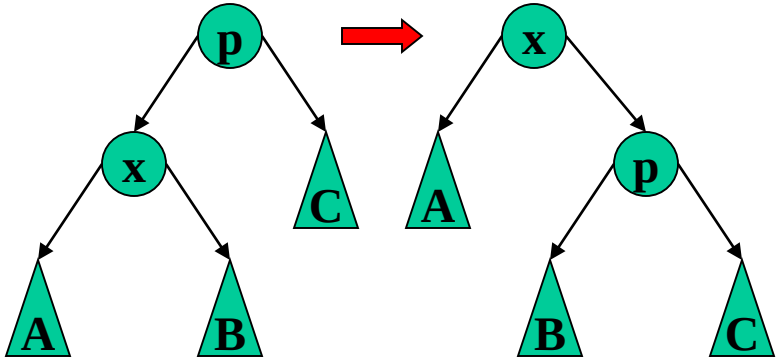


Скошенные деревья

Часто искомые узлы перемещаются ближе к корню за счет одинарных и двойных поворотов. Асимптотическая сложность выполнения *M* операций вставки и/или поиска составляет $O((N+M)\log_2(N+M))$.







B-деревья

Количество ключей от $M/2$ до $M–1$, количество ссылок на 1 больше. Левая ссылка указывает на узлы, содержащие меньшие ключи – остальные по под диапазонам. Растут вверх. Используются для построения файловых систем, так как позволяют отыскивать искомые узлы за меньшое количество обращений к дереву (соответственно диску).

Многопутевое дерево поиска, построенное на основе k-узлов – узлов, содержащих упорядоченный набор k-1 ключей и соответствующих им k указателей на дочерние k-узлы. Если v_0, v_1, …, v_{k-2} – ключи, p_0, p_1, …, p_{k-1} – указатели, то p_i, $i \in [0; k-2]$ указывает на узлы, ключи которых меньше v_i, а p_i, $i \in [1; k-1]$ указывает на узлы, ключи которых больше v_{i-1}. B-дерево порядка M – это дерево, содержащее k-узлы, $k \in [M/2; M]$ (для корня $k \in [2; M]$), длина пути от корня до любого листа в котором одинакова. Для B-дерева порядка M и высоты h количество ключей:

$N \in [\, 2(M/2)^{h-1} ; M^h]$.

Параметр M имеет обычно достаточно большое значение, так как B-деревья и их разновидности широко используются для хранения данных на внешних носителях. Некоторые реализации B-деревьев предполагают наличие указателей на «братьев» – соседние узлы.

B-деревья. Представление узла.

```
struct BNode {
Key Data[M];
struct BNode * Childs[M+1];
int Count;
};
```

B-деревья. Операции

Поиск: бинарный поиск по странице, возможно с переходом к поддереву.

Вставка: 1. Поиск узла для вставки. 2. Если ключей меньше *M*–1, то осуществляется вставка. 3.1. Иначе происходит расщепление заполненного узла на два *M*/2-узла с перемещением центрального элемента на уровень выше. Этот процесс может продолжать рекурсивно, вплоть до корня. 3.2 Расщепление заполненных узлов можно осуществлять при поиске страницы, тем самым устраняя необходимость рекурсивного возврата. 4. B-деревья растут вверх!

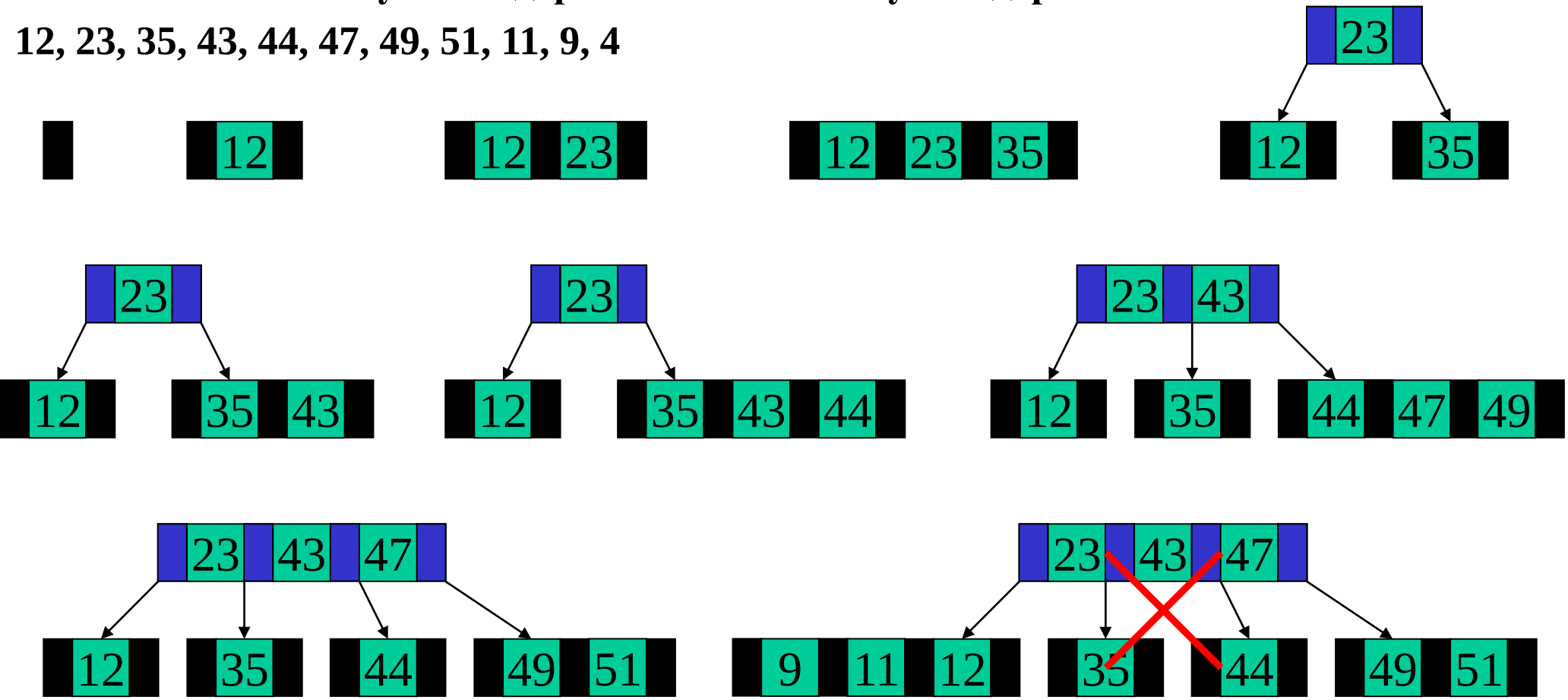
Удаление: 1. Удаление осуществляется только из листьев. Если ключ не в листе, то необходимо найти минимального справа (максимального слева), обменять местами и удалить. 2. Если ключей больше *M*/2, то удаление элементарно. 3. Если сумма ключей в листовых соседних страницах больше *M*–1, то выполняется переливание. 4. Иначе – слияние.

Для пунктов 3,4 – средний из страницы-предка.

2-3-4-деревья

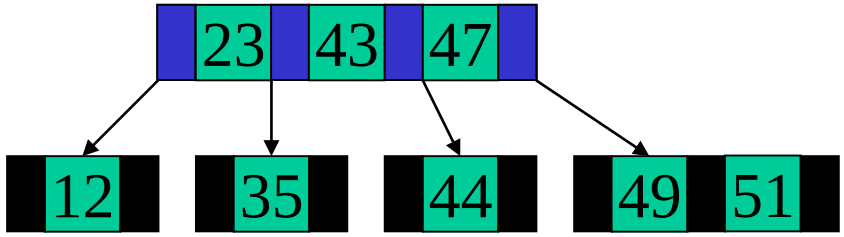
Частный случай B-деревьев – *M* = ? В пустое дерево вставляются ключи:

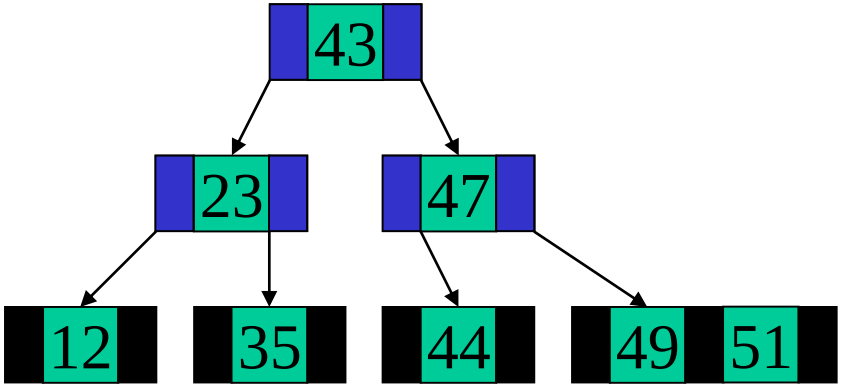
12, 23, 35, 43, 44, 47, 49, 51, 11, 9, 4

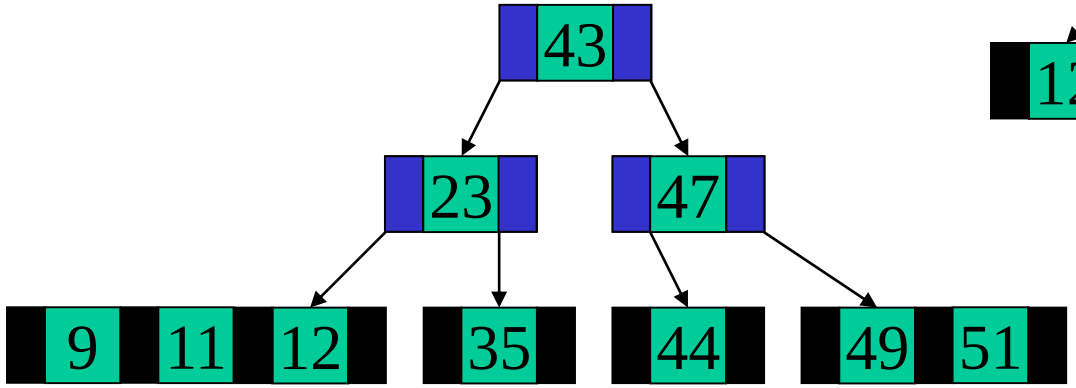


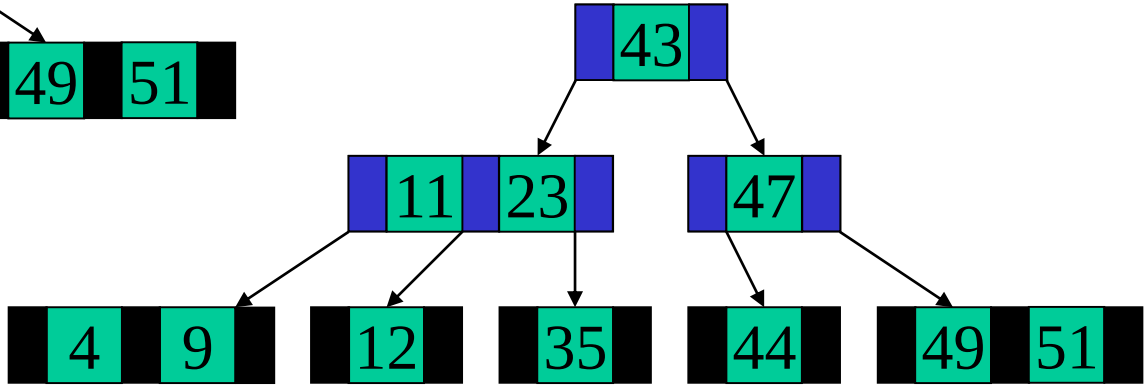
2-3-4-деревья. Вставка

В пустое дерево вставляются ключи: 12, 23, 35, 43, 44, 47, 49, 51, 11, 9, 4



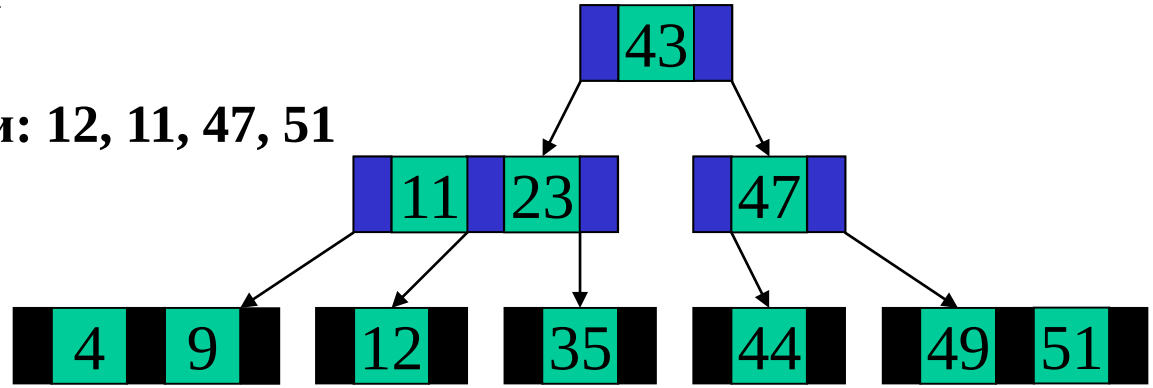


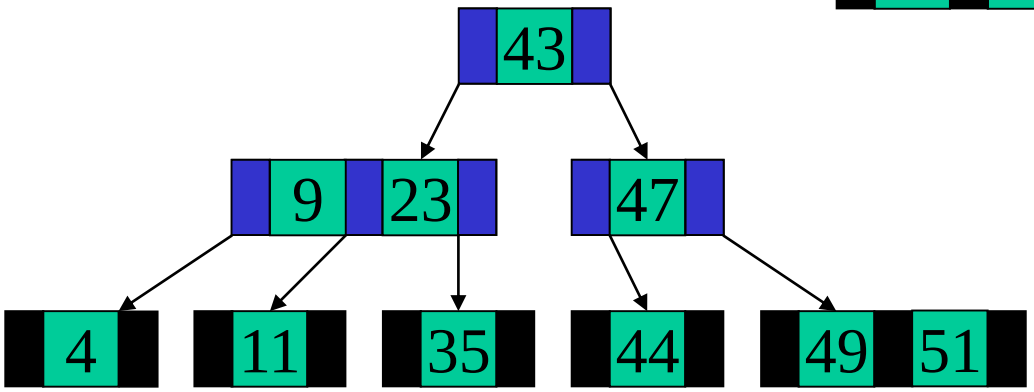


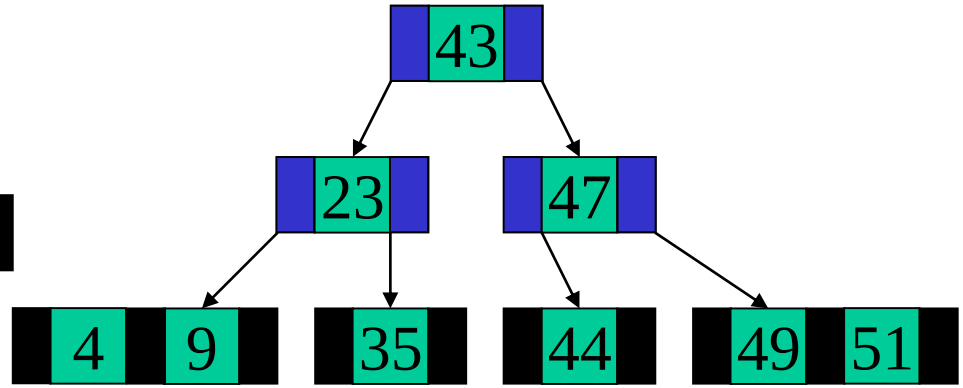


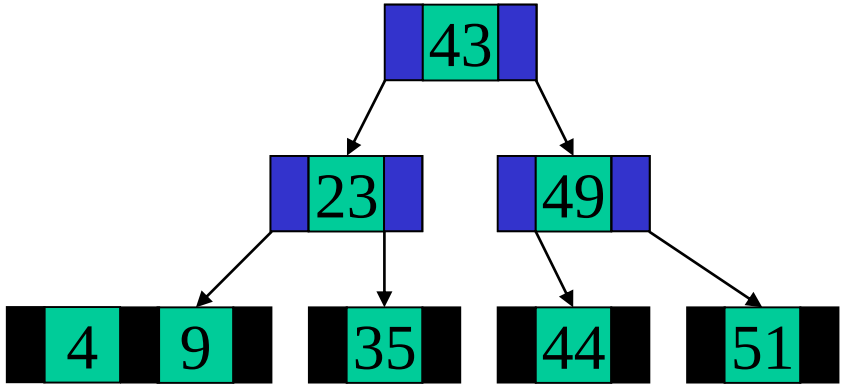
2-3-4-деревья. Удаление

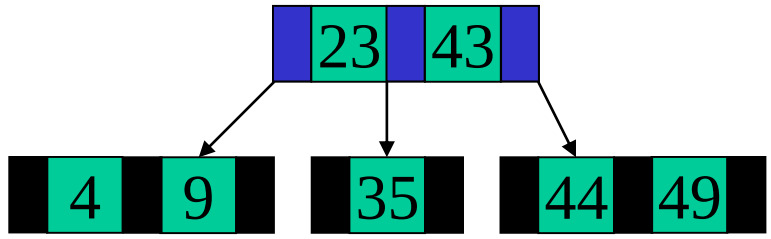
Из дерева удаляются ключи: 12, 11, 47, 51











B*-деревья

Подвид B-деревьев, отличается заполнением узлов от 2*M*/3 до *M*. Обеспечивают более эффективный поиск при усложнении алгоритмов вставки/удаления.

B*-деревья это разновидность B-деревьев, в которых количество связей принадлежит диапазону [2*M*/3; *M*]. В при добавлении ключа, в тот момент когда страница заполнена происходит не расщепление узла, а перераспределение ключей с соседней страницей, вплоть до того момента когда и она не заполнится. После этого происходит расщепление двух страниц на три. Корень такого дерева заполняется до 4*M*/3, после чего расщепляется на две страницы.

B+-деревья

Подвид B-деревьев. Данные хранятся только в листьях (во внешних узлах), во внутренних узлах (кроме листьев) – только копии некоторых ключей из листьев.

B+ – деревья это разновидность B-деревьев, в которых данные хранятся только во внешних ключах, а внутренние узлы содержат только копии некоторых из ключей, хранящихся в листьях. Для B^+-дерева порядка M построенного из N ключей количество зондирований $Z \in [\log_M N; \log_{M/2} N]$ и содержит в среднем $N / (\ln 2 * M) = 1{,}44 N/M$ узлов (страниц).

Цифровые деревья поиска

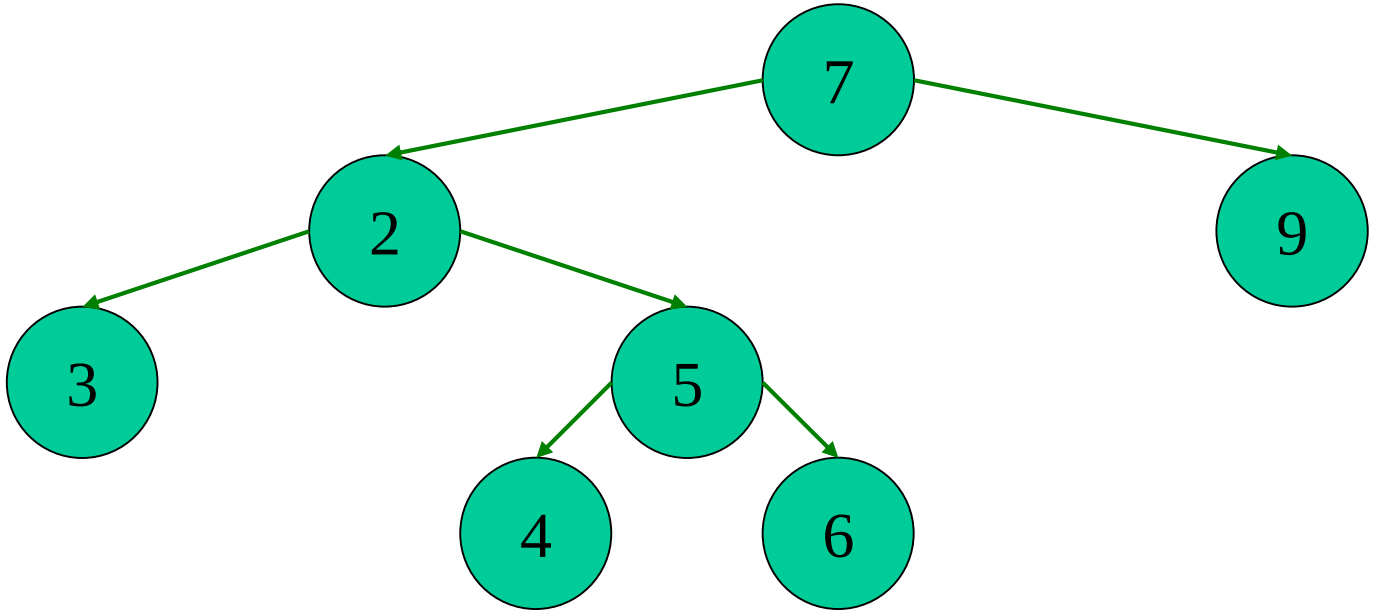
Цифровое дерево поиска (ЦДП) – это бинарное дерево, в котором, в отличие от бинарного дерева поиска, ветвление осуществляется не в соответствии с результатом сравнения полных ключей, а в соответствии с выбранными [двоичными?] разрядами ключа.

Для ЦДП не выполняется условие упорядоченности.

Максимальная длина пути: в среднем – $\log_2 N$, в худшем – $2\log_2 N$.

ЦДП имеет смысл использовать, если $N << 2^w$, а если эти значения сопоставимы, то лучше использовать ключи как индексы.

Цифровые деревья поиска



7	0111
2	0010
3	0011
5	0101
4	0100
6	0110
9	1001

Боры, словари (trie-деревья)

ЦПД, в котором ключи хранятся в листьях, а внутренние узлы содержат ссылки на левое и правое поддеревья, с ключами, начинающимися соответственно с 0 или 1 разряда.

Построение бора описывается следующими правилами:

**** 0 ключей – внешний узел;**

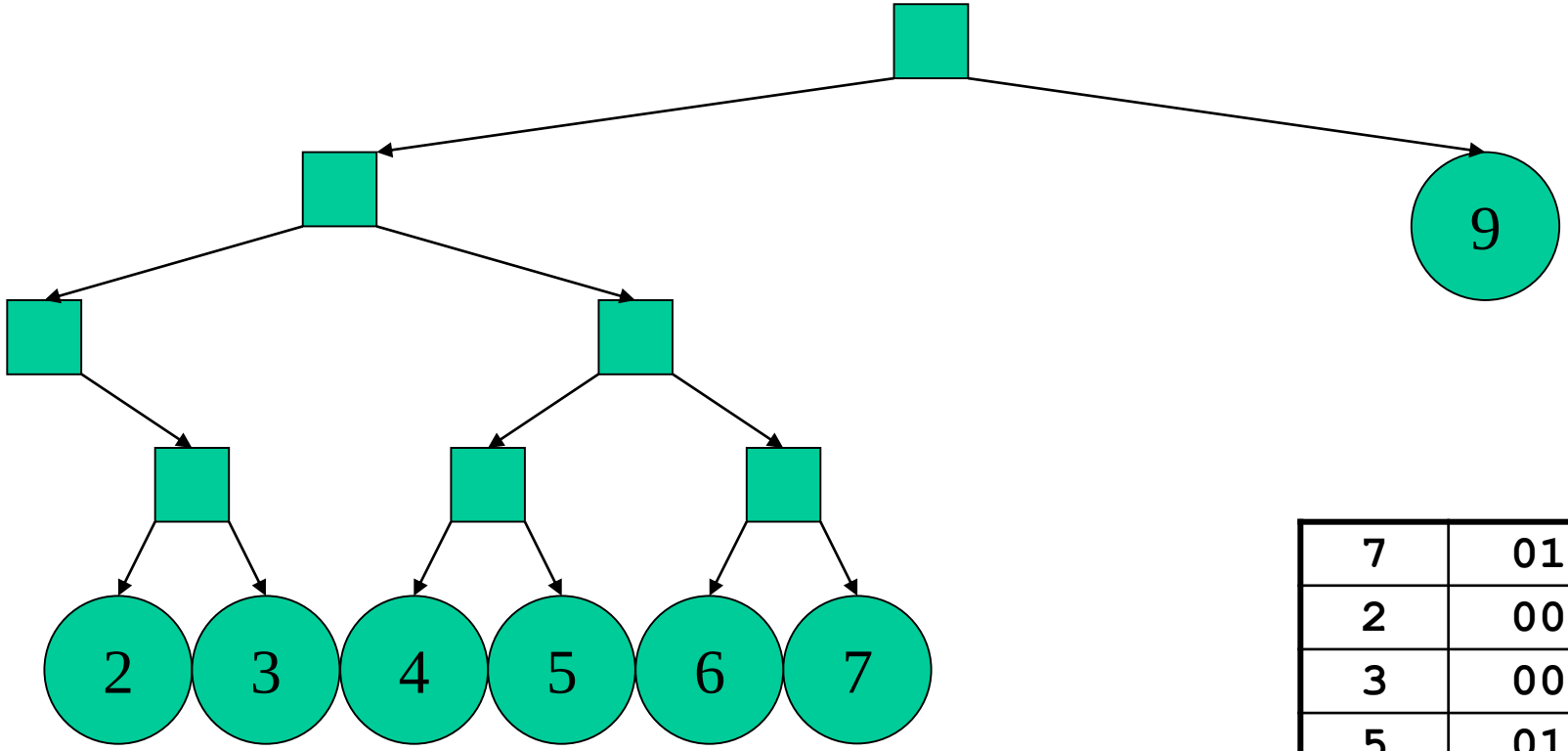
**** 1 ключ – корень;**

**** >> 1 – это ЦПД, корень которого указывает на левое и правое поддеревья, с ключами начинающимися соответственно с нулевого и единичного разрядов в текущей позиции (начиная со старшей).**

Структура бора не зависит от порядка добавления ключей. Бор построенный из N случайных ключей содержит в среднем $N/\ln 2 = 1{,}44N$ узлов.

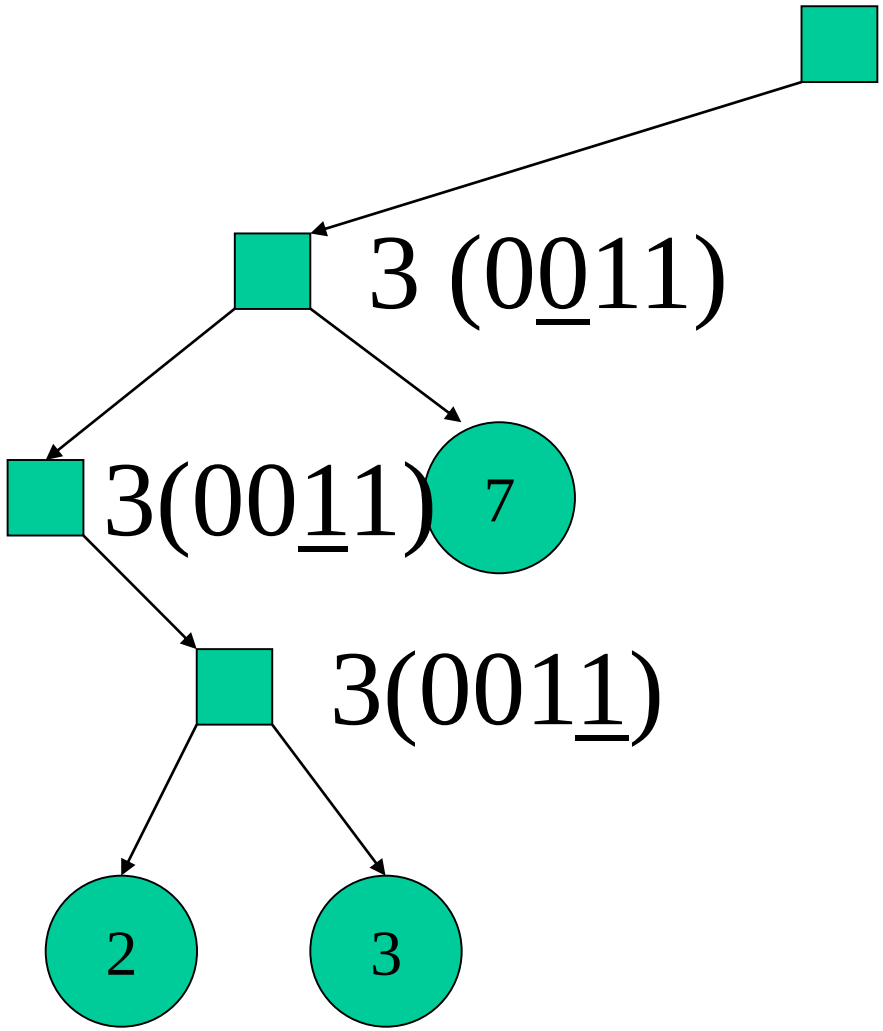
Поиск: в среднем $O(\log_2 N)$, в худшем – $O(w)$, w – количество разрядов. Для ключей переменной длины: префикс любого ключа не должен совпадать с ключом.

Боры, словари (trie-деревья)



7	0111
2	0010
3	0011
5	0101
4	0100
6	0110
9	1001

Боры, словари (trie-деревья)

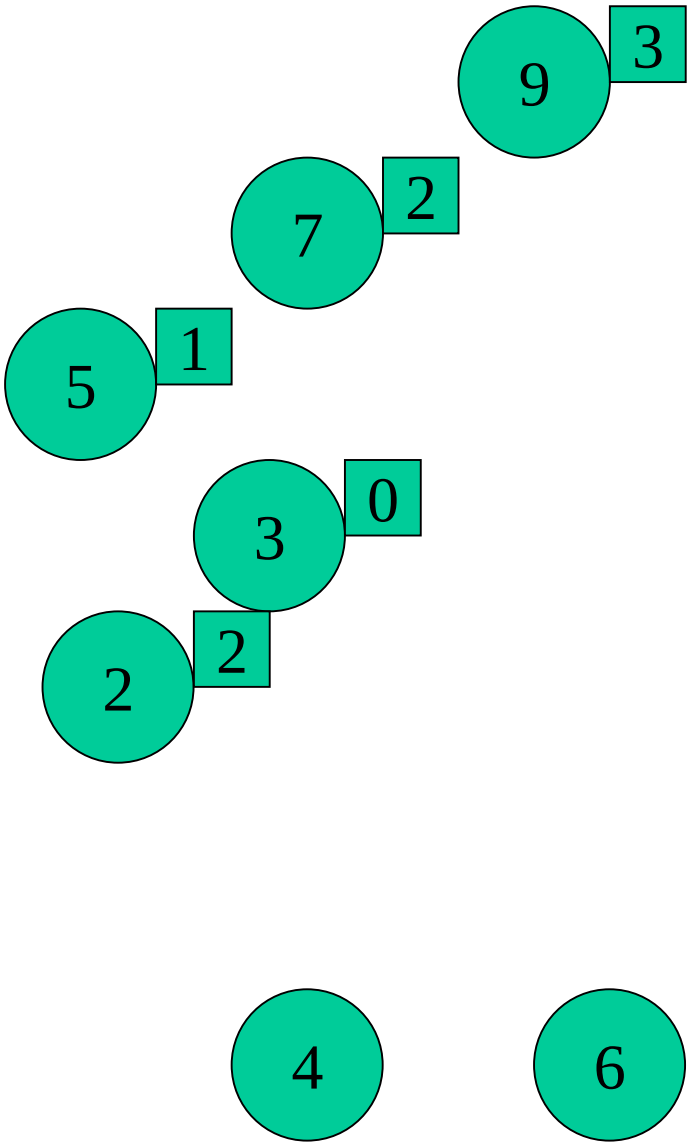


7	0111
2	0010
3	0011
5	0101
4	0100
6	0110
9	1001

Patricia-деревья

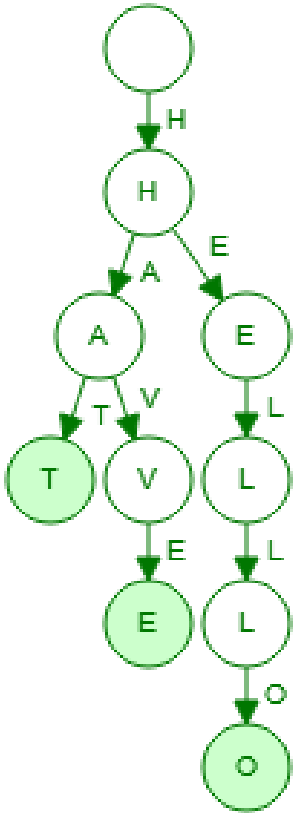
Это бор, в котором хранится индекс (номер) проверяемого разряда, что позволяет (по сравнению с борами) использовать идентичные типы узлов и сократить длину пути от корня до узлов, содержащих ключи.

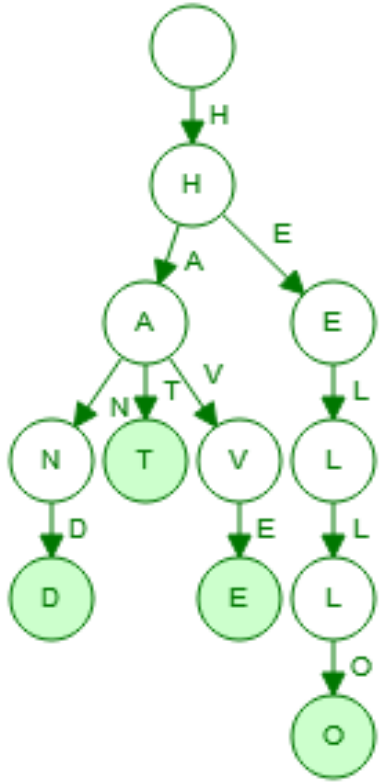
Patricia-деревья

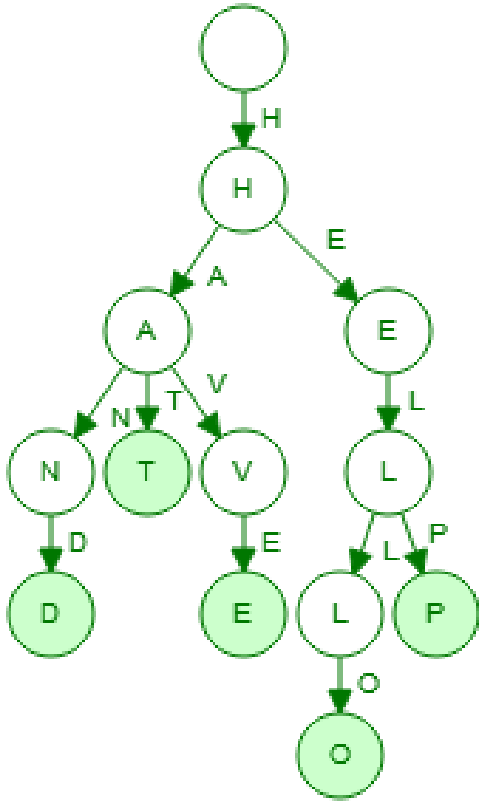


7	0111
2	0010
3	0011
5	0101
4	0100
6	0110
9	1001

Многопутевые trie-деревья







Patricia-деревья