Алгоритмы и структуры данных
Иерархические структуры данных

Зариковская Наталья Вячеславовна

Лекция 3

Теоретические сведения
ДЕРЕВЬЯ

- **Деревья произвольного (общего) вида.**
- Нелинейные структуры данных отражают более сложные отношения порядка между объектами, чем отношения следования.
- Наиболее важным видом нелинейных структур данных является деревья. Они выражают отношения: предок, потомок, брат, и т.д.
- Формально ***дерево*** (*tree*) определяется как конечное множество T одного или более узлов со следующими свойствами:
- существует один выделенный узел, а именно – ***корень*** *(root)* данного дерева T;
- остальные узлы (за исключением корня) распределены среди $m \geq 0$ непересекающихся множеств $T_1, \ldots, T_m$ называются ***поддеревьями*** *(subtrees)* данного корня.

ДЕРЕВЬЯ

- Как видите, это определение является рекурсивным: дерево определено на основе понятия дерева. Рекурсивный характер деревьев можно наблюдать и в природе, например почки молодых деревьев растут и со временем превращаются в ветви (поддеревья), на которых снова появляются почки, которые также растут и со временем превращаются в ветви (поддеревья), и т.д. Из этого определения следует, что каждый узел дерева является корнем некоторого поддерева данного дерева. Количество поддеревьев узла называется ***степенью*** (*degree*) этого узла. Узел со степенью нуль называется ***концевым узлом*** (*terminal node*) или *листом* (*leaf*). Неконцевой узел называется ***узлом ветвления*** (*branch node*). Уровень (*level)* узла по отношению к дереву T определяется рекурсивно следующим образом. Уровень корня дерева T равен нулю, а уровень любого другого узла на единицу выше, чем уровень корня ближайшего поддерева дерева T, содержащего данный узел.
- Если не считать различными два дерева, которые отличаются только относительным порядком поддеревьев узлов, то дерево называется ***ориентированным*** (*oriented*), поскольку при этом имеет значение только относительная ориентация узлов, а не их порядок.

Теоретические сведения
ДЕРЕВЬЯ

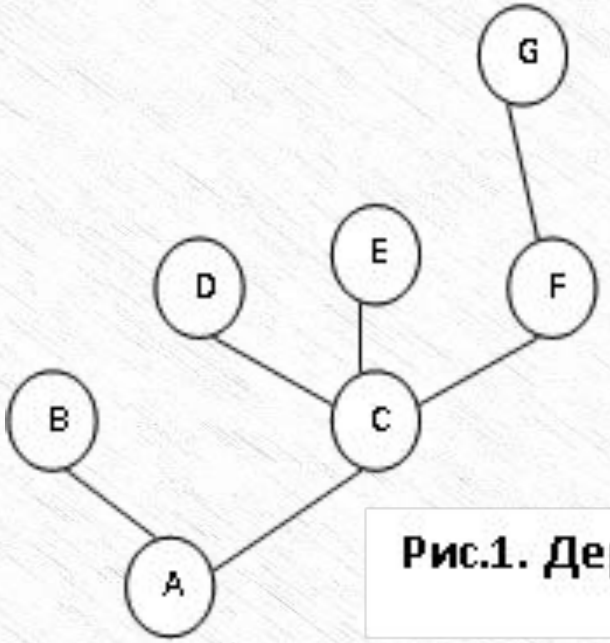


Рис.1. Дерево1.

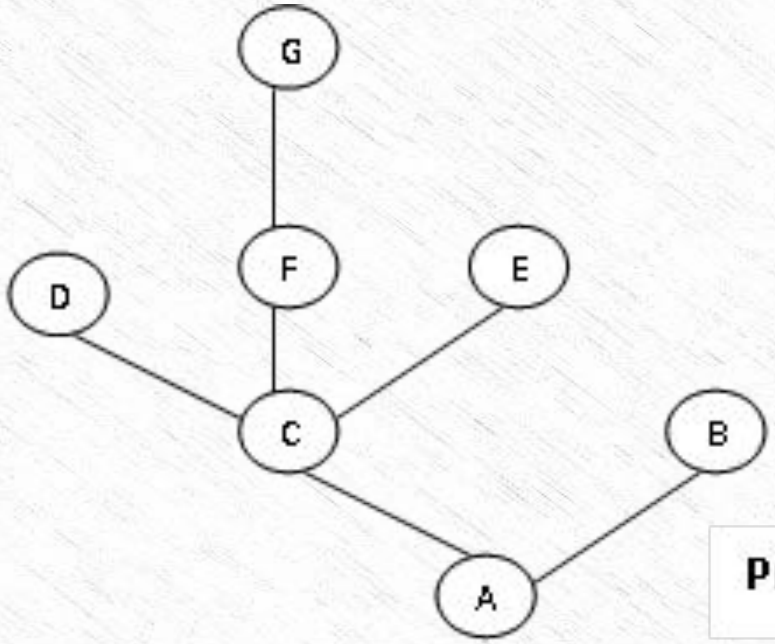


Рис.1. Дерево2.

ДЕРЕВЬЯ

- Сама природа представления данных в компьютере определяет неявный порядок любого дерева, поэтому в большинстве случаев упорядоченные деревья представляют больший интерес. Далее будем неявно предполагать, что все рассматриваемые деревья являются упорядоченными, если явно не указанно обратное. Соответственно деревья на рис. 1 и 2 в общем случае рассматриваются как разные, хотя как ориентированные деревья они совершенно одинаковы.
- Лес (forest)- это множество (обычно упорядоченное), не содержащее ни одного непересекающегося дерева или содержащее несколько непересекающихся деревьев. Тогда еще одна формулировка п. (b) в данном выше определении дерева могла бы выглядеть так: узлы дерева при условии исключении корня образуют лес.

ДЕРЕВЬЯ

- **Основные свойства дерева:**
- корень не имеет предков
- каждый узел за исключением корня, имеет единственного предка
- каждый узел связан с корнем единственным путем, т.е. в дереве нет замкнутых циклов
- если относительный порядок следующих узлов имеет значение, деревья называются упорядоченными

H =

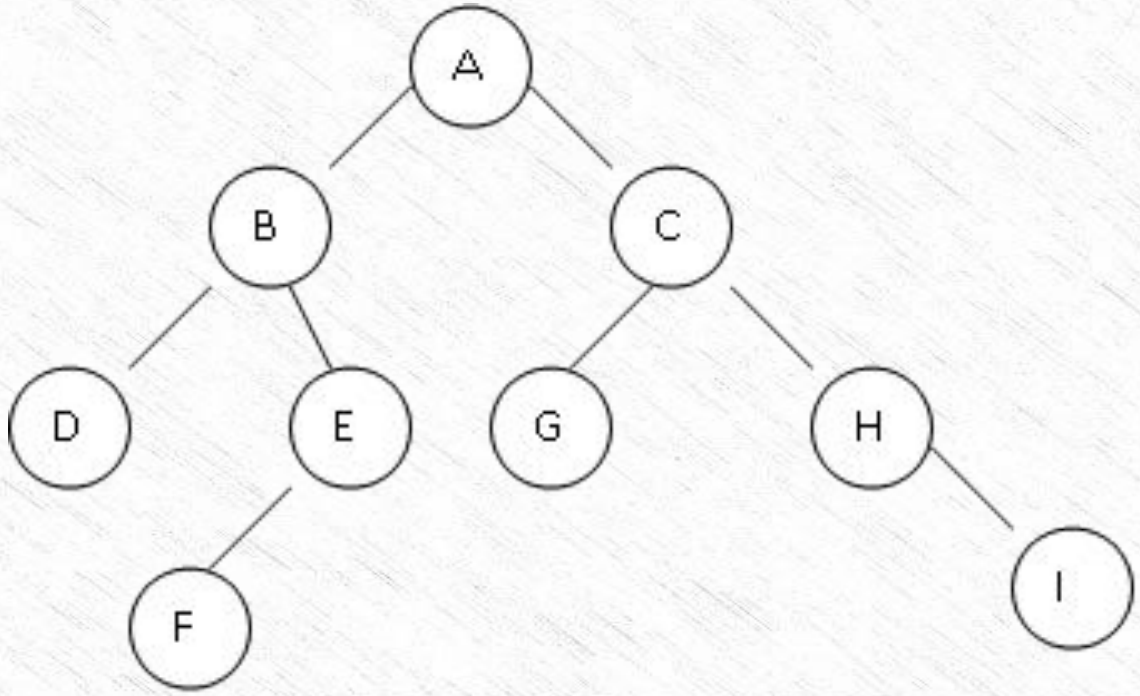
Бинарные (двоичные) деревья.

- Важное значение имеют упорядоченные деревья второй степени – бинарные.
- Бинарным деревом называется конечное множество узлов, которые или пусты или состоят из 2-х непересекающихся подмножеств: левым и правым поддеревьями и корнем.
- Максимальное число узлов в бинарном дереве высотой h составляет : $N_2 = 2^h - 1$
- Максимальное число узлов в бинарном дереве на узле i : $N_i = 2^i - 1$
- Высотой полного бинарного, созданного из n узлов определяется как нижняя граница логарифма из числа узлов +1: H = $\lfloor Log_2 n \rfloor + 1$

Представление бинарных деревьев.

- В памяти с последовательной организацией.
- Если известен максимальный размер дерева, т.е. его высота и соответствующее количество узлов в полном дереве, то структуру дерева можно хранить в виде массива. Корень дерева будет располагаться в элементе с индексом 1. для каждого коренного узла с индексом k, левое поддерево будет располагаться с индексом [2*k], а правое [2*k+1].
-
- A B C D E G H F I
- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Представление бинарных деревьев.



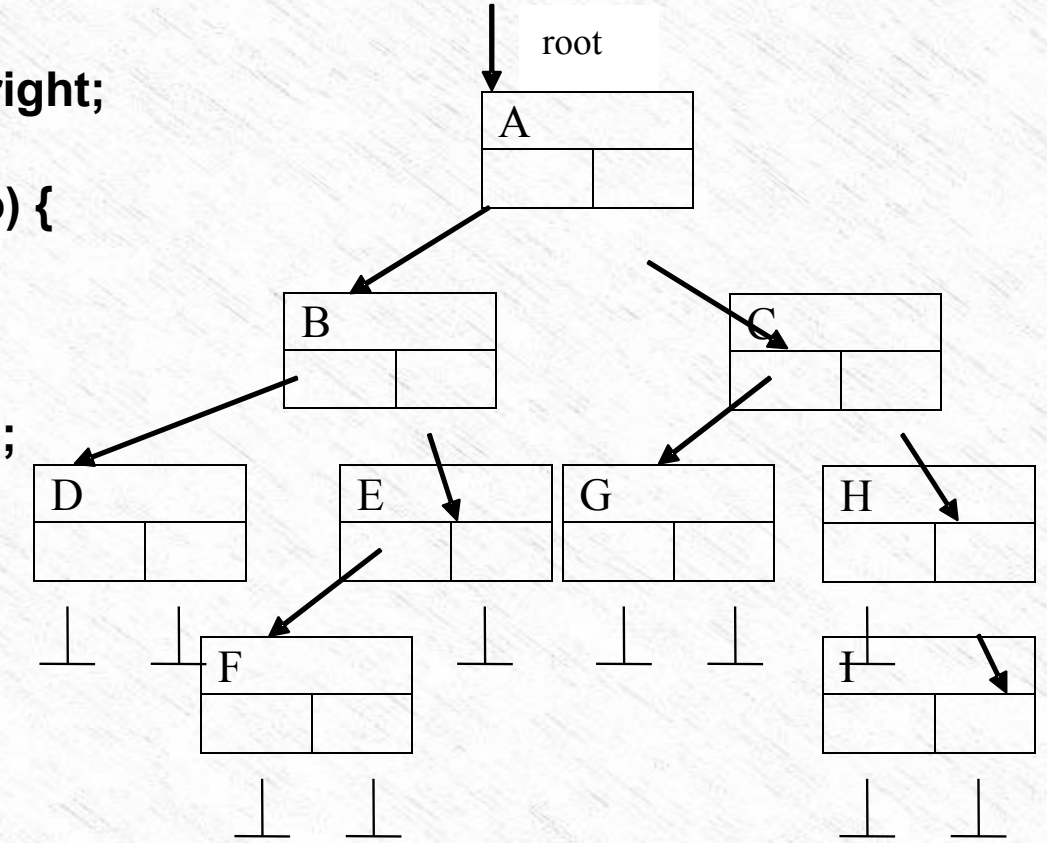
Деревом такой структуры является простота доступа по индексам как от предка к потомку, так и от потомка к предку. Недостатком является то, что если дерево не является полным, в массиве, представляющем его, появляется большое количество пустых элементов. Структура дерева рассчитана на его определенную высоту. Поэтому дерево не может расти. При удалении узлов из дерева и соответственно изменение его структуры, потребуется модификация массива.

Алгоритмы обхода бинарных деревьев

Представление бинарного дерева в динамической памяти со связной организацией. Элемент хранения узла бинарного дерева состоит из одного или нескольких полей и 2-х указателей на левое и правое поддерево каждого корневого узла.

root = NULL – дерево пустое.
Дерево приведенного вида строится начиная с корня с помощью последовательного включения узлов либо в левое, либо в правое поддерево корневого узла.

```
struct TreeNode {
        TreeNode   *left , *right;
        char Info;
        TreeNode(char info) {
                Info =info;
         }
};
TreeNode *root = NULL;
```



Представление бинарных деревьев.

- **Обход** – алгоритм обработки, при котором узел дерева обрабатывается единственный раз одинаковым образом в определенным порядке.
- Обход дерева заключается в поэтапном разбиении структуры дерева на корень, левое и правое поддеревья. И в применении к ним алгоритма обработки узлов до тех пор пока в процессе разбиения не будет получено пустое поддерево. Существуют три принципа упорядочивания узлов, которые соответствуют рекурсивной природе дерева.
- Существуют левосторонние и симметричные к ним правосторонние алгоритмы.

Представление бинарных деревьев

- Левосторонние алгоритмы:
- Низходящий (прямой) обход к – л – п(корень левое правое)
  - Обработать корневой узел
  - В низходящем порядке обойти левое поддерево корневого узла
  - В низходящей порядке обойти правое поддерево коневого узла
- Восходящий обход л – п – к
  - Обойти в восх. порядке левое поддерево
  - Обойти в восх. порядке правое поддерево корневого узла
  - Обработать корневой узел
- Смешанный (обратный) обход л – к – п
  - В смеш. порядке обойти левое поддерево корневого узла
  - Обработать корневой узел
  - В смеш.порядке обойти правое поддерево корневого узла

Представление бинарных деревьев

- Различные алгоритмы обхода бинарного дерева делают различную линейную расстановку множества информационных полей узлов дерева. При различии алгоритмов обхода, относительный порядок узлов не изменяется.
- *Бинарное или двоичное дерево есть конечное множество узлов, которое или пусто, или состоит из корня и двух непересекающихся бинарных деревьев, называемых левым и правым поддеревом данного дерева.*
- *Построить двоичное дерево и распечатать его. Использовать три варианта обхода: префиксный или прямой - корень, левое поддерево, правое; постфиксный или концевой - левое поддерево, правое, корень; и инфиксный или обратный - левое поддерево, корень, правое.*
- *Пусть вершины дерева задаются буквами. Признак конца текста - точка. Пустые поддеревья обозначаются звездочками, то есть задается вид дерева.*
- *Пример вводимого текста - СТУ***ДЕ*Н**Т**. При работе с деревом используются рекурсивные алгоритмы.*

Представление бинарных деревьев

```
#include <iostream>
#include <string>
using namespace std;
 struct TreeNode {
        TreeNode   *left , *right;
        char Info;
        TreeNode(char info) {
                Info =info;
         }
};
TreeNode *root = NULL;

void makeNode(TreeNode **node, char info){
         (*node) = new TreeNode(info);
         (*node)->left = NULL;
         (*node)->right = NULL;
 }


void prefix(TreeNode *node){ //префиксный обход
        if (node == NULL) return;
        cout<<node->Info;
        prefix(node->left);
        prefix(node->right);
}


void infix(TreeNode *node){//инфиксный обход
        if (node == NULL) return;
        infix(node->left);
        cout<<node->Info;
        infix(node->right);
}
```

```
void postfix(TreeNode *node){//постфиксный обход
if (node == NULL) return;
        postfix(node->left);
        postfix(node->right);
        cout<<node->Info;
}
string s;
int index;
void create(TreeNode **node ){ //создание дерева
        index++;
        if (index >= s.length() || s[index]=='.') {
                (*node) = NULL;
                return;
        }
        if (s[index]!='*'){
                 makeNode(node, s[index]);
                 create(&(*node)->left);
                 create(&(*node)->right);
        } else {
                (*node) = NULL;
        }
}

int main() {
        cout<<"Enter string: ";
        cin >>s;
        index = -1;
        create(&root);
        cout<<endl<<"prefix:"<<endl;
        prefix(root);
        cout<<endl<<"infix:"<<endl;
        infix(root);
        cout<<endl<<"postfix"<<endl;
        postfix(root);
        return 0;
}
```

Деревья цифрового поиска

- Существует класс задач, которые оперируют с данными, представляющими собой множество слов некоторого языка. К таким задачам можно отнести задачи проверки орфографии, задачи перевода текста и т. п.
- Одна из основных операций, которая часто используется в подобных задачах – поиск слова в множестве слов. Эффективность организации такого поиска во многом является определяющим для всей системы в целом.
- Пусть имеется множество слов M. Необходимо определить, принадлежит ли слово s множеству M.
- Поставленную выше задачу можно решить с помощью одного из рассмотренных нами ранее алгоритмов поиска. Одним из наиболее эффективных, например является алгоритм бинарного поиска. Однако, во-первых, множество M должно быть упорядоченно, что является существенным ограничением. Во-вторых, время поиска будет зависеть от размера множества M, а не от искомого слова s.

Деревья цифрового поиска

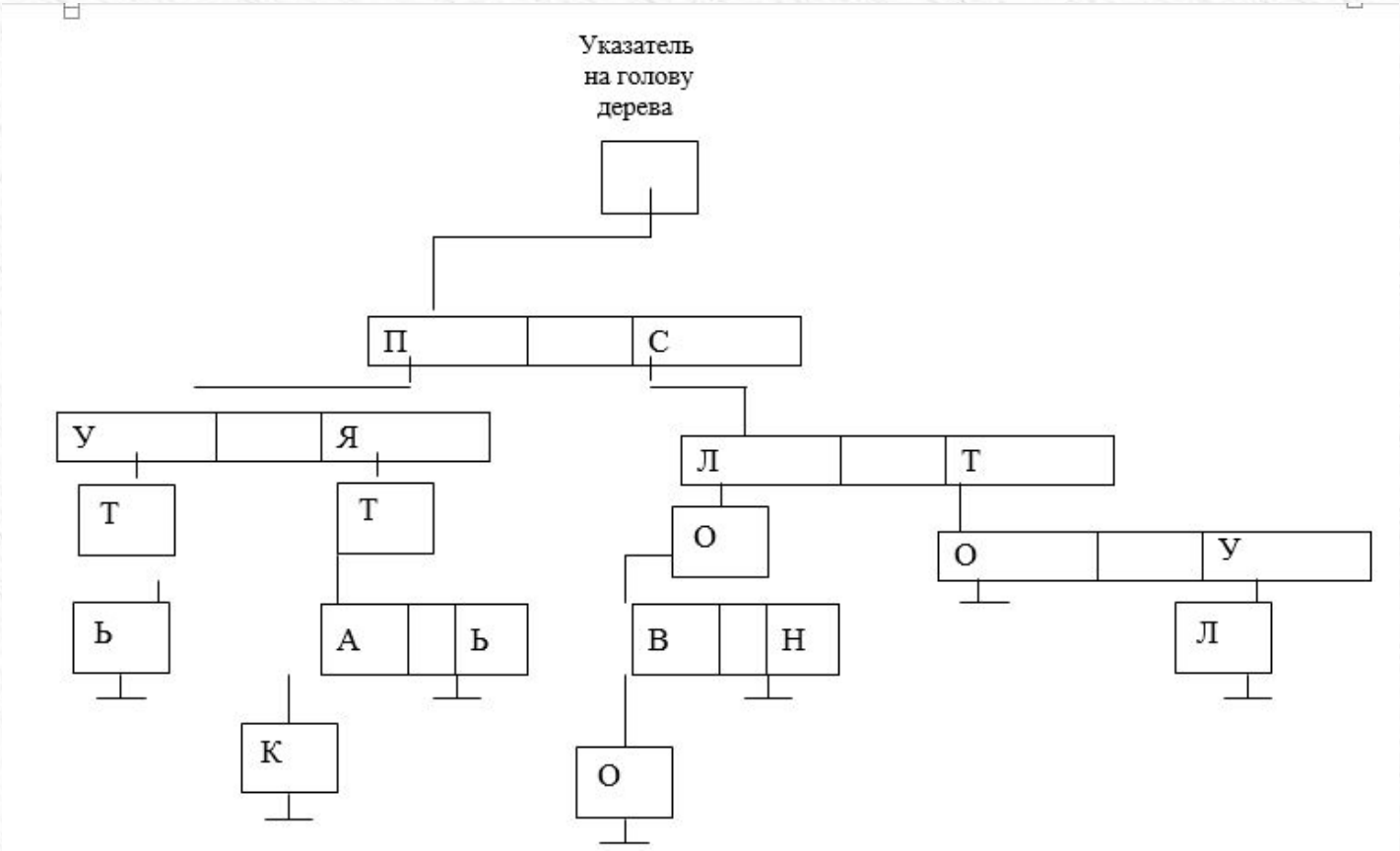
- Для рассмотренного выше класса задач используют еще один алгоритм поиска – алгоритм цифрового поиска.
- Пусть множество слов М неупорядоченно, т.е. для поиска приемлем только последовательный поиск. В этом случае время поиска значительно сократится, если разбить множество М на подмножества МА , МБ …, МЯ, в каждом из которых хранятся слова на соответствующую букву.
- Тогда структура хранения будет выглядеть как двухуровневое дерево.

Деревья цифрового поиска

- На первом уровне этого дерева массив указателей, каждый из которых может быть либо пустым (указатели ъ, ь, й), либо содержать адрес соответствующего подмножества слов, находящегося на втором уровне дерева.
- Рассмотренный выше подход можно развить дальше, организовав трехуровневое дерево, в котором первый уровень – указатели для первой буквы слова, второй – указатели для второй буквы слова, третий – множество слов, начинающихся на АА, АБ, … , ЯЯ.
- Ясно, что при этом количество элементов в подмножествах $M_{АА}$, $M_{АБ}$, … ,$M_{ЯЯ}$ значительно уменьшится, а следовательно, уменьшится и время последовательного поиска.
- Продолжая процесс увеличения уровней дерева, можно прийти к структуре дерева, в котором каждая буква слова будет располагаться на своем уровне. Такое дерево называется **деревом цифрового поиска** или **бором.**
- При такой организации хранения и поиска данных время поиска слова будет зависеть только от количества букв в этом слове.

Создание дерева цифрового поиска.

- Рассмотрим процесс формирования дерева цифрового поиска на примере. Пусть задано множество слов {слово, слон, стул, сто, пять, путь, пятак}.
- Построим дерево цифрового поиска



Поиск в дереве цифрового поиска

- Алгоритм поиска по бору достаточно очевиден. Первая буква искомого слова ищется на первом уровне дерева. Если в соответствующей букве указатель пуст – поиск неудачен. В противном случае вторая буква слова таким же образом ищется в элементе дерева на втором уровне по найденному указателю и т.п.
- Поиск заканчивается либо когда в очередном узле встретится пустой указатель (поиск неудачен), либо когда будет найдена последняя буква искомого слова.
- При всей очевидности алгоритма поиска в нем есть несколько не всегда очевидных моментов. Так, если использовать рассмотренный выше алгоритм формирования бора, то в дереве на рис.2 будет найдено не только слово «пятак», но и слово «пята», а также слова «пят», «пя», «п» и другие подслова введенных слов, хотя мы их не вводили.

Поиск в дереве цифрового поиска

- Для решения этой проблемы необходимо, чтобы каждое заносимое в бор слово имело символ завершения. Если предположить, что мы рассматриваем только слова русского языка, записанные большими буквами, то символом завершения слова может быть буква «а», имеющая следующий за буквой «Я» код.
- В этом случае каждый узел дерева цифрового поиска будет представлять собой массив из 34 указателей (33 буквы русского алфавита, 34 – символ завершения слова). На языке Паскаль такой массив указателей может быть описан следующим образом:
- Mas: array[‘A’..’a’] of <указатель>,
- Что позволит обращаться к каждому указателю по соответствующей букве, не используя ненужные вычисления.
- Очевидно, что при такой организации хранения данных очень расточительно используется память. Требуемую память можно сократить за счет увеличения времени выполнения, представляя каждую вершину бора в виде связанного списка.

Поиск в дереве цифрового поиска

- Другой подход к использованию деревьев цифрового поиска предполагает, что только несколько уровней бора используются для поиска первых букв ключа, а для поиска оставшихся букв используется какая-нибудь другая структура типа линейного списка или бинарного дерева.
- Другими словами, на практике можно использовать цифровой поиск как промежуточный вариант между рассмотренными нами структурами.