# Способы представления словарей

# Способы представления

- Линейный упорядоченный список
- Деревья
- Хэш-таблицы

# Словарь, базирующийся на линейном списке

- Потребуются функции
  - Вставки элемента
  - Поиска элемента
  - Просмотр всех элементов списка

Недостатки данной реализации:

- В случае большого числа слов данное представление приводит к длительному поиску

# Представление словарей в виде деревьев

- Двоичного поиска
  - В этом случае потребуются функции вставки элемента, поиска элемента, обхода дерева
- Реализация словарей в виде бинарного дерева даже для текста, содержащего 40-50 слов оказывается более выгодной, чем реализация в виде линейного списка

# Представление словарей в виде Бора

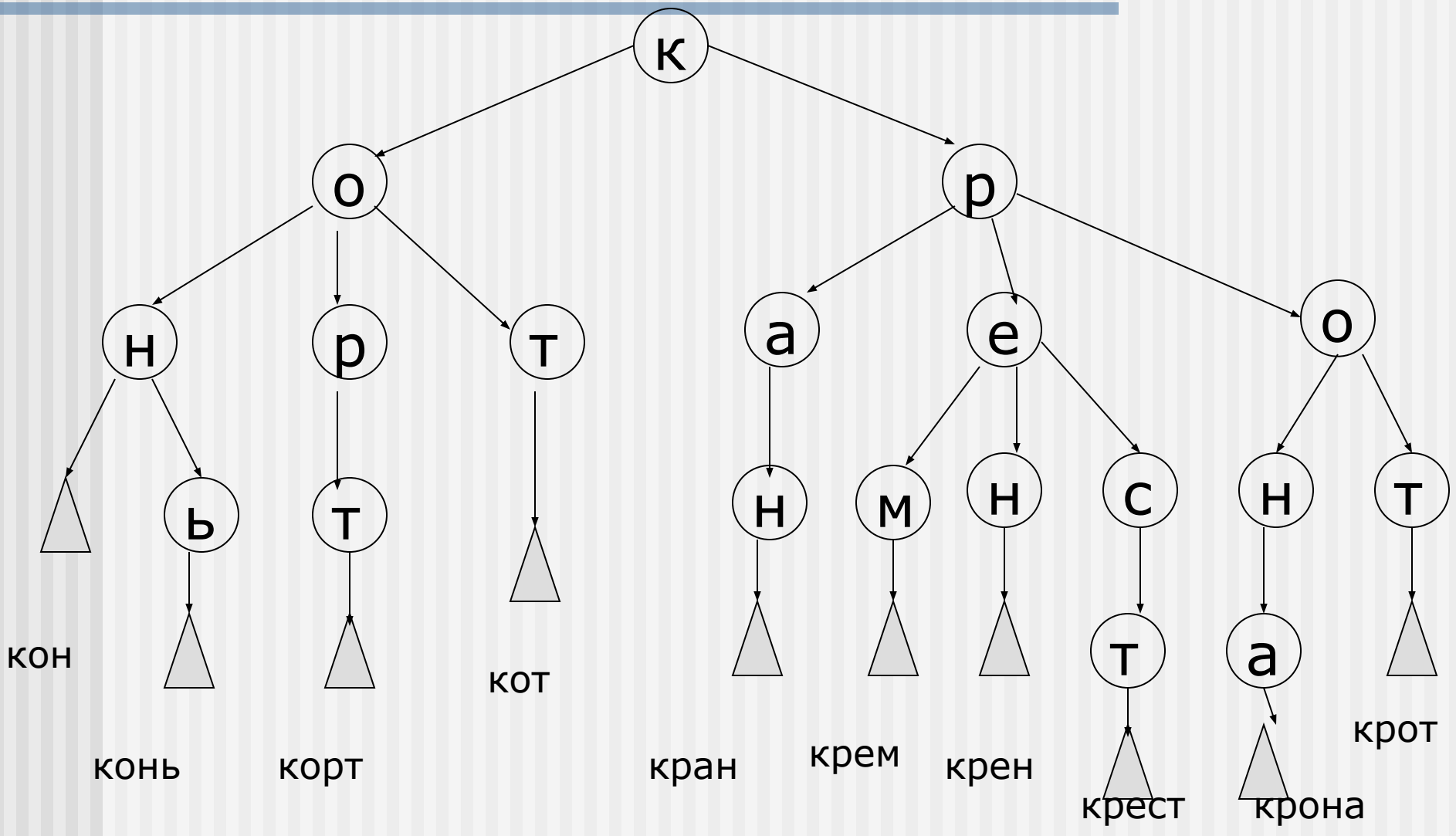
- В боре слова хранятся не целиком, а побуквенно
- Данная реализация особенно удобна, если многие слова имеют одинаковые префиксы и отличаются только своими окончаниями

# Представление словарей в виде Бора

- Допустим в словаре необходимо хранить следующие слова:

  *кон, конь, корт, кот, кран, крем, крест, крона, крот.*

# Организация словаря в виде бора



# Реализация словаря в виде бора

- В реализации бор может быть представлен в виде списка деревьев, узлы которого состоят из букв
- Каждый узел дерева будет содержать:
  - Info – поле, содержащее очередную букву либо указатель на связанный со словом объект
  - Son – указатель на список поддеревьев следующего уровня
  - Brother – указатель на следующий узел в списке узлов одного уровня

# Реализация словаря с помощью хэш-таблицы

- Словарь представляет собой массив слов
- Для эффективной организации вставки и удаления элементов в массив используются *функции расстановки или хэширования*

# Реализация словаря с помощью хэш-таблицы

- *Функция расстановки* получая в качестве параметра слова, выдает в качестве результата некоторое целое число – индекс в словаре, под которым следует хранить это слово.
- В этом случае вместо поиска вычисляется значение функции расстановки

# Реализация словаря с помощью хэш-таблицы

- *Способы* задания функций расстановки:

  Необходимо, чтобы данная функция легко вычислялась и зависела от всех символов слова.

# Реализация словаря с помощью хэш-таблицы

Например:

- Можно взять сумму кодов всех букв.
- Чтобы функция расстановки зависела от позиции символа в слове к коду каждой буквы можно добавить номер ее позиции
- При создании функции расстановки необходимо учитывать, чтобы ее значения были равномерно распределены между на множестве обрабатываемых слов

# Реализация словаря с помощью хэш-таблицы

- Часто используется следующая формула:

  (P*X+Q)%N

  где P и Q – некоторые простые числа, по порядку близкие к N

  X – вычисленная сумма кодов

  N – размер массива

- Например, при N=1000

  F=(557*x+811)%1000

# Реализация словаря с помощью хэш-таблицы

- При использовании функций расстановки может оказаться, что два слова имеют один и тот же индекс. В этом случае говорят, что происходит ***конфликт хэш-индексов***

# Разрешение конфликтов хэш-индексов

- Существует 2 подхода к разрешению конфликтов:
  - *Открытая адресация: поиск внутри таблицы другой свободной ячейки*
  - *Перестройка структуры таблицы*

# *Открытая адресация*

- Если при попытке записи элемента в ячейку выяснилось, что она занята, *зондируется* другая пустая, или открытая ячейка, в которую можно записать новый элемент

- Последовательность проверяемых ячеек называется *зондируемой последовательностью.*

# *Методы зондирования*

- Линейное зондирование:
  Допустим в таблицу хэширования необходимо занести значения:
  7597, 4567, 0628, 3658

- Возьмем функцию
  $H(x)=x \bmod 101$
  Для каждого из значений получаем
  $H(x)=22$

# *Допустим после удаления элемента 0628, необходимо найти - 3658*

- Получаем таблицу

| | | |
|---|---|---|
| … | | |
| … | | |
| 22 | 7597 | H=22 |
| 23 | 4567 | H=H+1 |
| 24 | 0628 | H=H+2 |
| 25 | 3658 | H=H+3 |
| … | | |
| … | | |

- После удаления

| | | |
|---|---|---|
| … | | |
| … | | |
| 22 | 7597 | H=22 |
| 23 | 4567 | H=H+1 |
| 24 | | |
| 25 | 3658 | H=H+3 |
| … | | |
| … | | |

# *Методы зондирования*

- Квадратичное зондирование : проводится зондирование свободных ячеек $h(x)$, $h(x)+1^2$, $h(x)+2^2$, …
- Проблемы:

  происходит вторичная кластеризация:

  если два элемента хэшируются в одну и ту же ячейку, проверяется одна и та же последовательность

# *Пример*

- Получаем таблицу

| | | |
|---|---|---|
| … | | |
| 22 | 7597 | H=22 |
| 23 | 4567 | H=H+1 |
| | | |
| | | |
| 26 | 0628 | H=H+4 |
| … | | |
| 31 | 3658 | H=H+9 |

# *Методы зондирования*

- Двойное хэширование : последовательность проверяемых ячеек зависит от значения Хэширование начинается с ячейки $h_1(x)$. Размер шага задается функцией $h_2(x)$. При этом $h_2(x) \neq 0$, $h_1(x) \neq h_2(x)$.

# Двойное хэширование
## *Пример*

- Допустим таблица хэширования содержит 11 элементов.
- Определим функции хэширования:
  $h_1(x)=x \bmod 11$
  $h_2(x)=7 - (x \bmod 7)$
- Допустим x=58. $h_1=3$, $h_2=5$
- Последовательность зондируемых ячеек: 3, 8, 2, 7, 1, 6, 0, 5, 10, 4, 9
- Для x=14, последовательность выглядит иначе: 3, 10, 6, 2, 9, 5, 1, 8, 4 ($h_1=3$, $h_2=7$)

# *Перестройка таблицы хэширования*

- Способ 1:
  каждая ячейка таблицы сама является массивом, содержащий элементы, которые хэшируются в эту ячейку
- Способ 2:
  Таблица хэширования представляется в виде массива связных списков

# Чем отличается хорошая функция хэширования

- Функция хэширования должна легко и быстро вычисляться
- Функция хэширования должна равномерно распределять данные по таблице

# Достоинство хэширования

- При хэшировании такие операции как вставка, удаление, поиск элемента выполняются наиболее эффективно (даже по сравнению со сбалансированными деревьями)

# Недостатки хэширования

- Если в разрабатываемом приложении нужно выполнять упорядоченные операции, хэширование не дает эффективного решения.

# Одновременное применение нескольких структур данных

- В приложениях бывают необходимы структуры данных, предназначенные для решения разных задач.

# Одновременное применение нескольких структур данных

- Например, в приложении рассматривается очередь записей о клиентах банка.
- При этом следует предусмотреть возможность вывода фамилий клиентов в алфавитном порядке

# Одновременное применение нескольких структур данных

Проблема:

- Если реализовать структуру в виде очереди, записи не будут идти в алфавитном порядке
- Если записи хранятся в виде упорядоченного списка, нарушается принцип FIFO

# Одновременное применение нескольких структур данных

- Удобно предусмотреть две независимые структуры данных: очередь и упорядоченный список
- Легко реализуются операции, при которых данные извлекаются: GetFront и Traverse(обход списка)

# Одновременное применение нескольких структур данных

- Операции вставки и удаления элемента выполнить труднее:
- Вставка
  - Вставляем новое значение в конец очереди
  - Вставляем копию значения в упорядоченный список

# Одновременное применение нескольких структур данных

- Удаление
  - Удаляем значение из начала очереди очереди
  - Удаляем значение из упорядоченного списка

# Одновременное применение нескольких структур данных

- Улучшение:
- В структуре данных линейный список продолжает хранить записи о клиентах
- Очередь содержит только указатели на соответствующие значения в списке

# Одновременное применение нескольких структур данных

- В этом случае операция удаления элемента будет реализована очень эффективно, поскольку в удаляемом элементе очереди содержится указатель, который необходимо удалить из списка
- Список в этом случае должен быть двусвязным

# Одновременное применение нескольких структур данных

- Еще более эффективной получится схема, которая объединяет очередь с бинарным деревом: в этом случае операция вставки элемента будет происходить много быстрее
- При этом очередь должна по-прежнему содержать указатели на соответствующие узлы дерева.