Algorithms With Java

Week 2 Day 3: Префиксные деревья

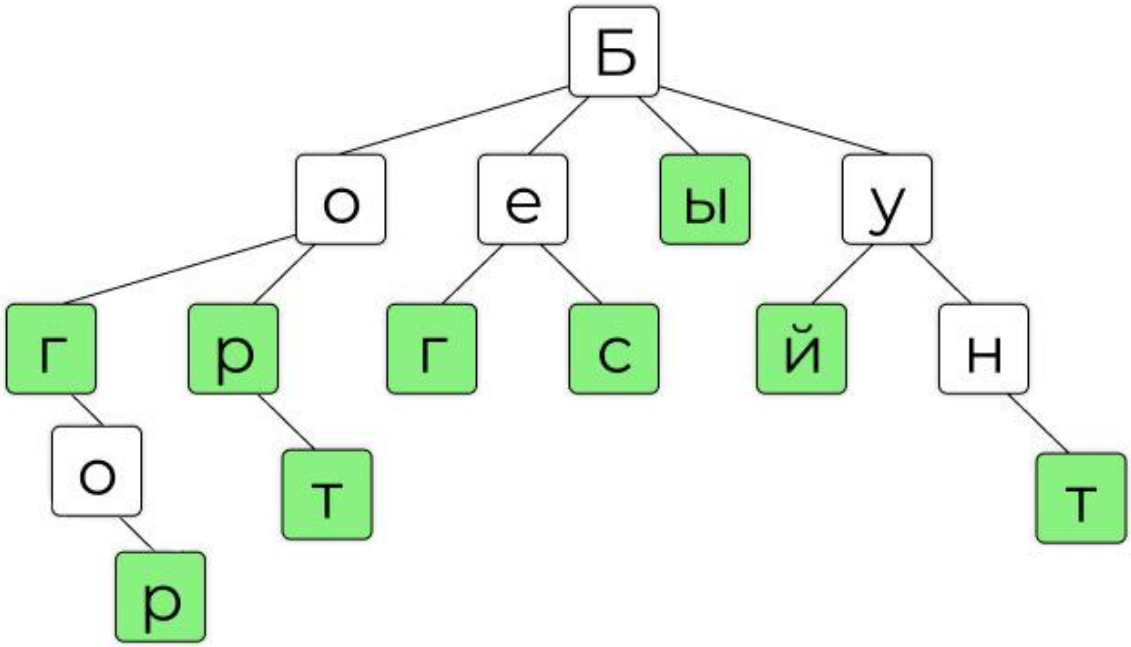
Академия Ковалевского

# Содержание

# 1. Теория

## Префиксные деревья

Префиксное дерево (trie, бор, или нагруженное дерево) — это структура данных, n-арное дерево, в узлах которого хранятся не ключи, а символы. А ключ — это путь от корня дерева до этого узла. Пример:




В этом дереве лежат ключи “Бог”, “Богор”, “Бор“, “Борт”, “Бег“, “Бес“, “Бы“, “Буй“ и “Бунт“. У каждого узла есть дополнительная характеристика, которая указывает, является ли этот узел конечной точкой или промежуточным значением. На дереве выше это зеленые и белые узлы: так компьютер знает, что “Бунт” — это ключ, а “Бун” — нет. Символов в узле может быть и несколько, например не буквы слов, а слоги. Но принцип тот же: ключ — не сам узел, а путь от корня до него.

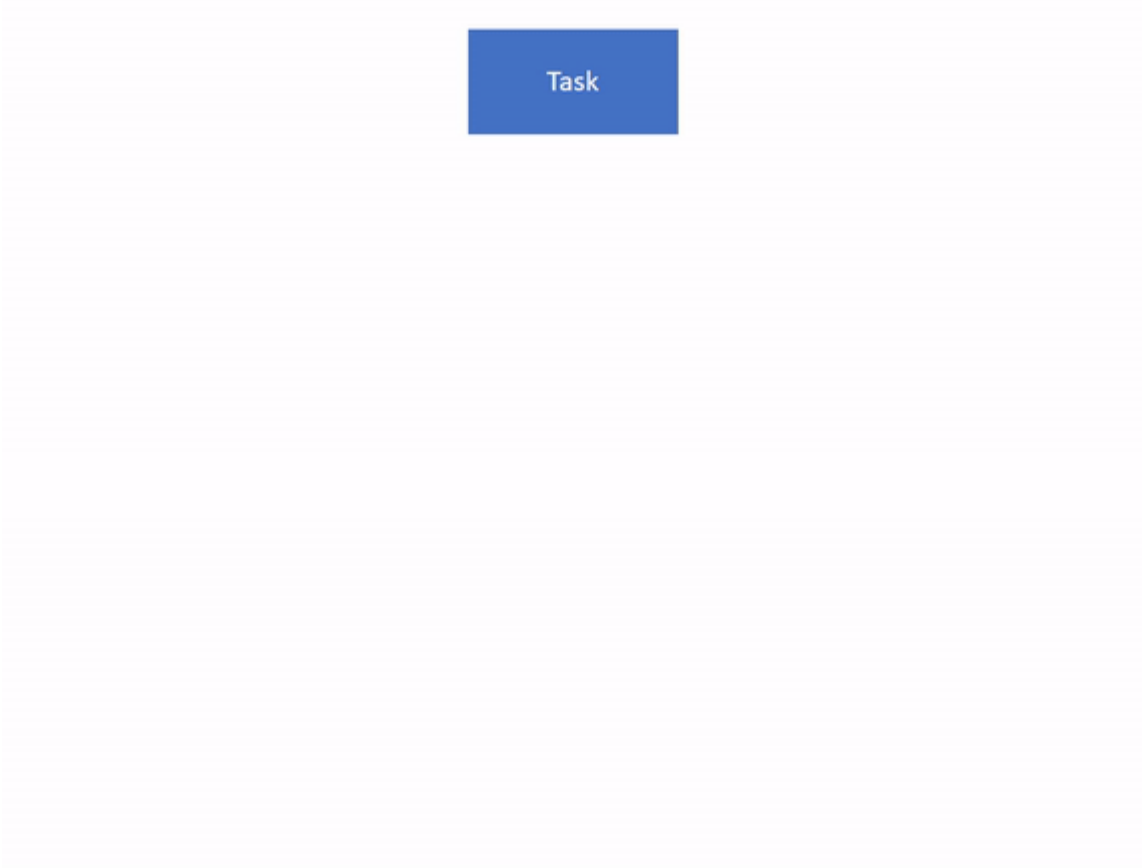
Префиксное дерево лежит в основе быстрого поиска строк, начинающихся на префикс — символ или несколько символов, которые вводит пользователь.

# Фреймворк ForkJoin

Фреймворк *ForkJoin* был разработан для ускорения выполнения задач, которые можно разделить на другие более мелкие подзадачи, выполняя их параллельно с использованием доступных ядер ЦП. Подзадачи должны быть независимыми друг от друга, а операции не должны иметь состояния, что делает этот фреймворк не лучшим решением для всех проблем.

Применяя принцип «разделяй и властвуй», фреймворк рекурсивно делит задачу на более мелкие подзадачи, пока не будет достигнут заданный порог. Это часть *fork*. Затем подзадачи обрабатываются независимо, и если они возвращают результат, все результаты рекурсивно объединяются в один результат. Это часть *join*.

После стольких мучений с рекурсией есть небольшая надежда на то, что теперь будет немного понятнее, почему рекурсивно решенные задачи так легко преобразовать в многопоточные решения.



Для параллельного выполнения задач фреймворк использует пул потоков с количеством потоков, равным количеству процессоров, доступных виртуальной машине Java (JVM) по умолчанию.

В практической работе мы будем использовать *ForkJoin* фреймворк для многопоточного обхода графа. На практике при малом числе состояний графа эффективней делать это в одном потоке, однако мы все равно будем использовать многопоточность не ради скорости, а ради “тренировки на кошках”.

# 2. Практическая работа

## TrieNode

Нода префиксного дерева:

```
package academy.kovalevskyi.algorithms.week2.day3;

import java.util.HashMap;
import java.util.Map;

public class TrieNode {
  public String value = "";
  public Map<Character, TrieNode> children = new HashMap<>();
  public boolean finalCharacter = false;
}
```

## Trie

Основное дерево:

```
package academy.kovalevskyi.algorithms.week2.day3;

import java.util.List;
import java.util.concurrent.ForkJoinPool;

public class Trie {

  private static final ForkJoinPool POOL = ForkJoinPool.commonPool();

  public void add(String word) {
    // TODO
  }

  public boolean containsExact(String word) {
    // TODO
  }

  public int count() {
```

```
    // TODO
  }

  public List<String> startsWith(String prefix) {
    // TODO
  }
 }
```

**Важно**: методы count и startsWith должны просто создавать начальную задачу для ForkJoinPool, запускать ее и получать результат.

## WordCountAction

Если у текущей ноды нет потомков или нода терминальная — таска должна вернуть *1*, а также запустить новую задачу для каждого потомка в пуле. (ВАЖНО: НЕ ВЫЗЫВАЙТЕ НАПРЯМУЮ метод compute, это не то, как работает многопоточность!)

```
package academy.kovalevskyi.algorithms.week2.day3;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class WordCountAction extends RecursiveTask<Integer> {

  private final TrieNode node;

  public WordCountAction(TrieNode node) {
    this.node = node;
  }

  @Override
  protected Integer compute() {
    // TODO
  }
}
```

# WordSearchAction

Эта задача должна вернуть лист со значением текущей ноды, если текущая нода терминальная и начинается с заданного префикса.

Задачи для всех потомков ноды нужно сабмитить ( = добавить и запустить) в пул.

**ВАЖНО**:

- Начальная нода у задачи должна быть нода, которая содержит префикс, иначе нам бы пришлось еще проверять у каждой ноды соответствия префиксу. Дело в том, что поиск префикса на префиксном дереве не имеет никакого смысла делать многопоточным, поэтому самая первая задача (task) должна быть запущена только после нахождения префикса
- НЕ ВЫЗЫВАЙТЕ НАПРЯМУЮ метод compute, это не то, как работает многопоточность!

```
package academy.kovalevskyi.algorithms.week2.day3;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class WordSearchAction extends RecursiveTask<List<String>> {

  private final TrieNode node;

  public WordSearchAction(TrieNode node) {
    this.node = node;
  }

  @Override
  protected List<String> compute() {
    // TODO
  }
}
```