Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №2

по дисциплине **«Функциональное программирование»**

Вариант: **pre-dict**

Преподаватель:
Доморацкий Эридан Алексеевич

Выполнил: Беля Алексей
Группа: P3317

Санкт-Петербург, 2025

1. Задание

Лабораторная работа №2

Цель: освоиться с построением пользовательских типов данных, полиморфизмом, рекурсивными алгоритмами и средствами тестирования (unit testing, property-based testing), а также разделением интерфейса и особенностей реализации.

В рамках лабораторной работы вам предлагается реализовать одну из предложенных классических структур данных (список, дерево, бинарное дерево, hashmap, граф...).

Требования:

1. Функции:
   - добавление и удаление элементов;
   - фильтрация;
   - отображение (map);
   - свертки (левая и правая);
   - структура должна быть моноидом.
2. Структуры данных должны быть неизменяемыми.
3. Библиотека должна быть протестирована в рамках unit testing.
4. Библиотека должна быть протестирована в рамках property-based тестирования (как минимум 3 свойства, включая свойства моноида).
5. Структура должна быть полиморфной.
6. Требуется использовать идиоматичный для технологии стиль программирования. Примечание: некоторые языки позволяют получить большую часть API через реализацию небольшого интерфейса. Так как лабораторная работа про ФП, а не про экосистему языка -- необходимо реализовать их вручную и по возможности -- обеспечить совместимость.
7. Обратите внимание:
   - API должно быть реализовано для заданного интерфейса и оно не должно "протекать". На уровне тестов -- в первую очередь нужно протестировать именно API (dict, set, bag).
   - Должна быть эффективная реализация функции сравнения (не наивное приведение к спискам, их сортировка с последующим сравнением), реализованная на уровне API, а не внутреннего представления.

Содержание отчёта:

- титульный лист;
- требования к разработанному ПО;
- ключевые элементы реализации с минимальными комментариями;
- тесты, отчет инструмента тестирования, метрики;
- выводы (отзыв об использованных приёмах программирования).

Варианты (колонка -- интерфейс, строка -- структура данных):

	Dict	Bag(multiset)	Set
AVL Tree	`avl-dict`	`avl-bag`	`avl-set`
Prefix Tree	`pre-dict`	`pre-bag`	`pre-set`
OpenAddress Hashmap	`oa-dict`	`oa-bag`	`oa-set`
Separate Chaining Hashmap	`sc-dict`	`sc-bag`	`sc-set`
RedBlack Tree	`rb-dict`	`rb-bag`	`rb-set`
Binary Tree	`bt-dict`	`bt-bag`	`br-set`

2. Ключевые элементы реализации

1. insert : string -> 'a -> 'a Prefix_tree_dict.t -> 'a Prefix_tree_dict.t

```
let insert key value t =
  let rec aux chars node =
    match chars with
    | [] -> { node with value = Some value }
    | c :: rest ->
      let child =
        match CharMap.find_opt c node.children with
        | Some child -> aux rest child
        | None -> aux rest empty
      in
      { node with children = CharMap.add c child node.children }
  in
  aux (List.of_seq (String.to_seq key)) t
```

Ключ проходит посимвольно через дерево, при необходимости создаются новые
узлы, в конце записывается значение. Возвращается новое дерево, исходное не изменяется.

2. remove : string -> 'a Prefix_tree_dict.t -> 'a Prefix_tree_dict.t

```
let remove key t =
  let rec aux chars node =
    match chars with
    | [] ->
      let new_node = { node with value = None } in
      if CharMap.is_empty new_node.children then None else Some
new_node
    | c :: rest ->
      match CharMap.find_opt c node.children with
      | None -> Some node
      | Some child ->
        match aux rest child with
        | None ->
          let new_children = CharMap.remove c node.children in
          if Option.is_none node.value && CharMap.is_empty
new_children then None
          else Some { node with children = new_children }
        | Some new_child ->
          Some { node with children = CharMap.add c new_child
node.children }
  in
  match aux (List.of_seq (String.to_seq key)) t with
  | None -> empty
  | Some t' -> t'
```

Рекурсивно ищет ключ, обнуляет значение и поднимаясь вверх, удаляет пустые ветви. Если вся ветка стала пустой, возвращается empty.

3. fold_left : ('b -> string -> 'a -> 'b) -> 'b -> 'a Prefix_tree_dict.t -> 'b

```
let fold_left f init t =
  let rec aux prefix acc node =
    let acc =
      match node.value with
      | Some v -> f acc prefix v
      | None -> acc
    in
    CharMap.fold (fun c child acc ->
      aux (prefix ^ String.make 1 c) acc child
    ) node.children acc
  in
  aux "" init t
```

Обходит все пары (ключ, значение) в лексикографическом порядке ключей, накапливая результат функцией f. prefix хранит текущий ключ как строку.

4. merge : 'a Prefix_tree_dict.t -> 'a Prefix_tree_dict.t -> 'a Prefix_tree_dict.t

```
let merge t1 t2 =
  let rec aux n1 n2 =
    let new_value =
      match n2.value with
      | Some _ -> n2.value
      | None -> n1.value
    in
    let new_children =
      CharMap.merge (fun _ c1 c2 ->
        match c1, c2 with
        | None, None -> None
        | Some c, None | None, Some c -> Some c
        | Some c1, Some c2 -> Some (aux c1 c2)
      ) n1.children n2.children
    in
    { value = new_value; children = new_children }
  in
  aux t1 t2
```

Рекурсивно объединяет два дерева: значения во второй структуре перекрывают значения в первой, дочерние карты объединяются через CharMap.merge. Эта функция вместе с empty задаёт структуре закон моноида.

3. Результаты работы тестов

Полученный отчёт:

Все тесты прошли: 39 тестов (24 unit + 15 property-based)

Время выполнения ~0.11s

Итог:

```
Ran: 39 tests in: 0.11 seconds.
OK
```

Метрики:

Покрытие API: все публичные функции библиотеки участвуют либо в unit-тестах, либо в PBT (часто и там и там).

Моноидные свойства проверяются property-based тестами на случайных словарях.

Свойство equal тестируется как:

детерминированные кейсы (разные значения/разные размеры)

рандомизированная независимость от порядка вставки (с учётом дедупликации ключей).

4. Листинг программы

https://github.com/aeshabb/lab2_fp

5. Вывод

В лабораторной работе реализован неизменяемый полиморфный словарь на основе префиксного дерева (trie), где вся логика построена на чистых рекурсивных функциях и алгебраической структуре моноида: это позволило естественно описать операции добавления, удаления, свёрток и объединения словарей, а также формализовать и проверить их свойства через property-based тесты; в результате получилась компактная, но выразительная реализация, хорошо демонстрирующая преимущества функционального подхода — простоту рассуждений о корректности, возможность композиции операций и независимость интерфейса от внутреннего представления структуры.