Министерство науки и образования РФ
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Отчёт по заданию № 3
на тему: “Множества + последовательности”
по дисциплине
“Алгоритмы и структуры данных”
Вариант 26

Выполнил студент гр.9308: Яловега Н.В.

Проверил: Колинько П.Г.

Санкт-Петербург, 2020 г.

Оглавление

Введение

Получить практические навыки работы со стандартной библиотекой шаблонов, с деревьями двоичного поиска и с последовательностями.

1. Задание

Реализовать индивидуальное задание темы «Множества + последовательности» в виде программы, используя свой контейнер для заданной структуры данных (хеш-таблицы или одного из вариантов ДДП), и доработать его для поддержки операций с последовательностями.

2. Формализация задания

Мощность множества: 26

Что надо вычислить: A \ (B ∩ C ∩ D) ⊕ E

Базовая СД: АВЛд — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Реализуемые операции над последовательностью:

ERASE — Из последовательности исключается часть, ограниченная порядковыми номерами от p1 до p2.

EXCL — Вторая последовательность исключается из первой, если она является ее частью.

CHANGE — Вторая последовательность заменяет элементы первой, начиная с заданной позиции p.

3. Описание контейнера

Контейнер *set_seq* хранит множество в виде АВЛ-дерева и последовательность в виде вектора итераторов на узлы дерева. Обход дерева даёт упорядоченную последовательность ключей, а обход вектора — произвольную, что позволяет работать со структурой данных и как с множеством, и как с последовательностью.

Для работы с АВЛ-деревом был создан контейнер *tree*. Для доступа к элементам дерева был создан прямой итератор ввода (чтения) *tree_iterator*. Такой итератор перемещается только вперед и поддерживает только чтение. Для него нужно определить операции сравнения (==, !=), разыменования (*), инкремент (++). Для итератора чтения также необходимы функции *begin*() и *end*(), определяющие рабочий интервал значений. Основной операцией является инкремент, и для того, чтобы он выполнялся за константное время, итератор хранит стек с путём от корня до текущего узла.

Для контейнера set_seq также был реализован итератор вставки. Итератор вставки создаётся из контейнера и, возможно, одного из его итераторов, указывающих, где вставка происходит, если это ни в начале, ни в конце контейнера. Итераторы вставки удовлетворяют требованиям итераторов вывода. Для данного итератора определены операторы присвоения (=), разыменовывания (*) и инкремента (++), причем все они фиктивные. Для поддержки итератора вставки нужно определить функцию вставки с сигнатурой *insert*(*where*, *data*), где *where* — итератор места вставки, а *data* — вставляемое значение. Функция должна возвращать итератор на вставленный элемент, чтобы обеспечивать вставку за константное время за счёт исключения необходимости поиска места вставки. Итератор вставки хранит итератор чтения на вставленное значение, поддерживая тем самым возможную последовательность вставок.

Последовательность в общем случае может содержать одинаковые ключи, а множество не может содержать одинаковые ключи. Но операции над последовательностями, в отличие от операций с множествами, могут приводить к появлению дубликатов ключей. Для этого каждый узел дерева хранит поле *duplicates,* которое показывает количество дупликатов ключа.

4. Оценка временной сложности операций

Вставка (insert)

В контейнере set_seq элемент вставляется и в дерево и в последовательность.

Вставка в дерево без указания места начала поиска может быть выполнена только за логарифмическое время O(logn), поскольку корректный поиск места вставки должен начинаться от корня дерева. Вставка за константное время O(1) возможна, только если итератор вставки укажет для начала поиска места вставки на ключ, вставленный последним, однако такая вставка допустима только для упорядоченной последовательности ключей, которые вставляются в пустое дерево. В противном случае вставка будет вызывать хаос в структуре. Поэтому допустимо использовать вставку за константное время только для двуместных операциях с множествами по схеме слияния.

Вставка в последовательность осуществляется за константное время O(1) (при условии, что контейнер не расширяется при добавлении в него элемента, если происходит перераспределение, то само перераспределение является линейным по всему размеру).

Поэтому в среднем временная сложность алгоритма вставки для операций над множествами по схемам слияния O(1), а для произвольной вставки сложность составляет O(logn).

Объединение множеств (operator&=)

Для объединения по схеме слияния происходит не более 2*(N1+N2)-1 сравнений, где N1 и N2 — размеры контейнеров, т.е временная сложность алгоритма O(n).

Разность множеств (operator-=)

Для разности по схеме слияния происходит не более 2*(N1+N2)-1 сравнений, где N1 и N2 — размеры контейнеров, т.е временная сложность алгоритма O(n).

Симметрическая разность множеств (operator^=)

Для симметрической разности по схеме слияния происходит не более 2*(N1+N2)-1 сравнений, где N1 и N2 — размеры контейнеров, т.е временная сложность алгоритма O(n).

Укорачивание (erase)

Временная сложность операции линейная O(n).

Исключение (excl)

Временная сложность операции линейная O(n).

Замена (change).

Временная сложность операции линейная O(n).

5. Примеры работы программы

1) Тестовый пример из программы (мощность множеств меньше 26 для наглядности)

Исходные множества и последовательности:

```
SET (A): 1 2 3
SEQUENCE: (A): < 3 2 1 >

SET (B): 1 3 4 5
SEQUENCE: (B): < 1 3 4 5 >

SET (C): 1 3 8
SEQUENCE: (C): < 1 3 8 >

SET (D): 1 3 8 12 14 23
SEQUENCE: (D): < 14 1 3 8 23 14 12 >

SET (E): 0 1 3 4 5 6
SEQUENCE: (E): < 1 1 0 3 6 5 4 >
```

Рисунок 1: Исходные данные

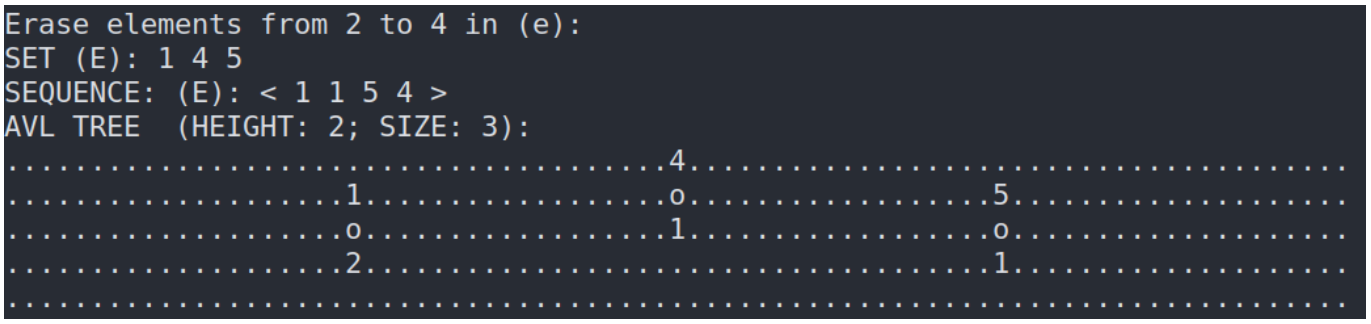
Пример выполнения A \ (B ∩ C ∩ D) ⊕ E:

(Должно получиться 0 1 2 3 4 5 6)

```
SET (F): 0 1 2 3 4 5 6
SEQUENCE: (F): < 0 1 2 3 4 5 6 >
AVL TREE  (HEIGHT: 3; SIZE: 7):
.....................................3.....................................
.................1...................o..................5.................
.........0.......o.......2.........1.........4.......o.......6.........
.........o.......1.......o.................o.......1.......o.........
.........1.............1.................1.............1.........
................................................................................
................................................................................
................................................................................
```

Рисунок 2: Пример операций над множеством

Пример выполения операции над последовательностью укорачивание. Из последовательности e удалим элементы с 2 по 4.



Пример выполения операции исключения. Из последовательности d исключим последовательность c.

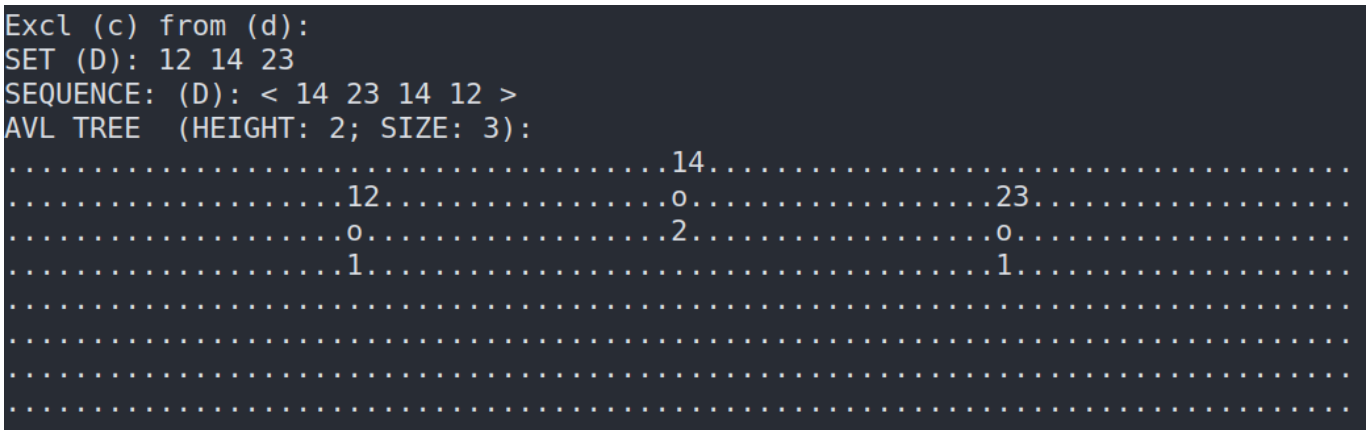


Рисунок 4: Пример исключения

Пример выполения операции замены. Заменим элементы из последовательности d последовательностью с 3 позиции.

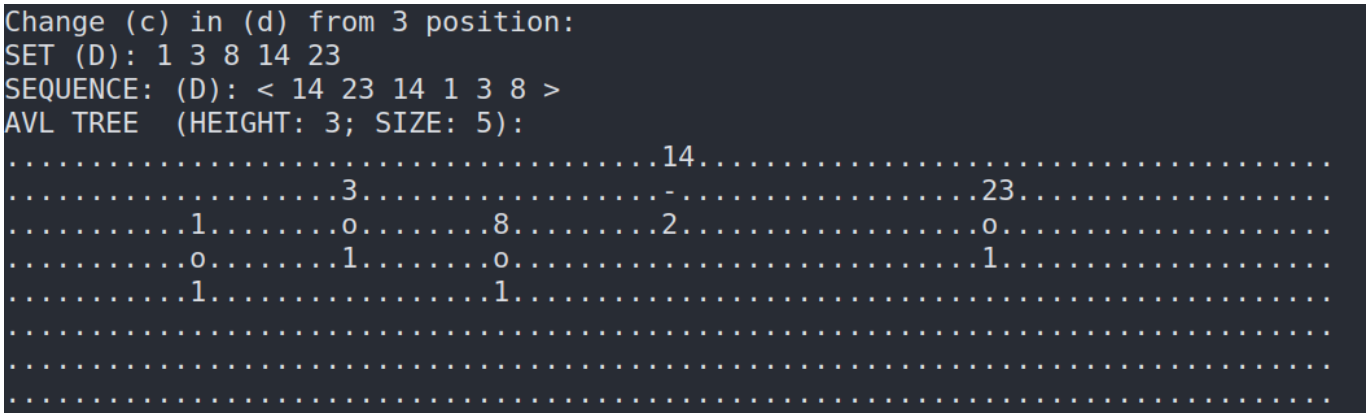


Рисунок 5: Пример замены

2) Пример на случайных данных

Сгенерированные множества и последовательности:

```
SET (A): 3 6 11 12 13 17 23 29 30 31 33 34 37 38 41 46 48 53 54 57 64 65 69
SEQUENCE: (A): < 31 65 53 17 54 23 48 64 30 12 57 38 37 11 46 69 3 13 23 33 3 29 29 6 41 34 >

SET (B): 0 2 8 14 19 21 22 24 25 26 32 33 34 36 42 44 45 59 63 65 70 71
SEQUENCE: (B): < 45 70 34 8 45 65 24 21 32 0 44 2 14 25 42 71 63 2 33 59 21 36 22 44 19 26 >

SET (C): 4 6 8 13 14 16 17 23 29 33 37 38 42 51 52 55 59 60 64 65 67 73 77
SEQUENCE: (C): < 23 77 60 14 33 55 6 17 13 52 4 37 73 64 65 67 16 29 42 8 51 55 38 6 37 59 >

SET (D): 1 14 19 21 22 31 35 38 39 45 46 48 50 54 55 56 59 63 67 70 73 77
SEQUENCE: (D): < 70 59 54 39 35 77 38 45 14 21 50 48 38 14 50 70 1 73 56 67 63 22 46 55 31 19 >

SET (E): 1 4 5 15 17 18 20 27 33 35 39 48 51 53 54 60 61 65 67 69 73 75 76
SEQUENCE: (E): < 33 69 53 20 51 73 1 27 35 65 54 73 60 18 17 61 67 5 75 39 76 76 35 4 15 48 >
```

Рисунок 6: Сгенерированные структуры

Пример выполнения A \ (B ∩ C ∩ D) ⊕ E:

```
SET (F): 1 3 4 5 6 11 12 13 15 18 20 23 27 29 30 31 34 35 37 38 39 41 46 51 57 60 61 64 67 73 75 76
SEQUENCE: (F): < 1 3 4 5 6 11 12 13 15 18 20 23 27 29 30 31 34 35 37 38 39 41 46 51 57 60 61 64 67 73 75 76 >
```

Рисунок 7: Пример операций над множеством

Пример выполения операции над последовательностью укорачивание:

```
Erase elements from 2 to 4 in (a):
SET (A): 3 6 11 12 13 17 23 29 30 31 33 34 37 38 41 46 48 53 54 57 64 65 69
SEQUENCE: (A): < 31 65 53 17 54 23 48 64 30 12 57 38 37 11 46 69 3 13 23 33 3 29 29 6 41 34 >

SET (A): 3 6 11 12 13 23 29 30 31 33 34 37 38 41 46 48 57 64 65 69
SEQUENCE: (A): < 31 65 23 48 64 30 12 57 38 37 11 46 69 3 13 23 33 3 29 29 6 41 34 >
```

Рисунок 8: Пример укорачивание

Пример выполения операции исключения:

```
Excl (b) from (c):
SET (B): 0 2 4 6 8 13 14 16 17 19 21 22 23 24 25 26 29 32 33 34 36 37 38 42 44 45 51 52 55 59 60 63 64 65 67 70 71 73 77
SEQUENCE: (B): < 45 70 34 8 45 65 24 21 32 0 44 2 14 25 42 71 63 2 33 59 21 36 22 44 19 26 23 77 60 14 33 55 6 17 13 52 4 37 73 64 65 67 16 29 42 8
 51 55 38 6 37 59 >

SET (C): 4 6 8 13 14 16 17 23 29 33 37 38 42 51 52 55 59 60 64 65 67 73 77
SEQUENCE: (C): < 23 77 60 14 33 55 6 17 13 52 4 37 73 64 65 67 16 29 42 8 51 55 38 6 37 59 >

SET (B): 0 2 8 14 19 21 22 24 25 26 32 33 34 36 42 44 45 59 63 65 70 71
SEQUENCE: (B): < 45 70 34 8 45 65 24 21 32 0 44 2 14 25 42 71 63 2 33 59 21 36 22 44 19 26 >
```

Рисунок 9: Пример исключение

Пример выполения операции замены:

```
Change (d) in (e) from 3 position:
SET (D): 1 14 19 21 22 31 35 38 39 45 46 48 50 54 55 56 59 63 67 70 73 77
SEQUENCE: (D): < 70 59 54 39 35 77 38 45 14 21 50 48 38 14 50 70 1 73 56 67 63 22 46 55 31 19 >

SET (E): 1 4 5 15 17 18 20 27 33 35 39 48 51 53 54 60 61 65 67 69 73 75 76
SEQUENCE: (E): < 33 69 53 20 51 73 1 27 35 65 54 73 60 18 17 61 67 5 75 39 76 76 35 4 15 48 >

SET (E): 1 14 19 21 22 31 33 35 38 39 45 46 48 50 53 54 55 56 59 63 67 69 70 73 77
SEQUENCE: (E): < 33 69 53 70 59 54 39 35 77 38 45 14 21 50 48 38 14 50 70 1 73 56 67 63 22 46 55 31 19 >
```

Рисунок 10: Пример замены

Вывод

При выполнении данной работы получены практические навыки по реализации собственных контейнеров, работе с множествами и последовательностями.

Список используемых источников

1. Колинько П.Г. Пользовательские контейнеры / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.

Приложение

screen.h — функции для работы с экраном

avl_tree.h — реализация АВЛ дерева

set_seq.h — реализация контейнера множетво+последовательность

main.cpp — демонстрационная программа