

Дерево van Emde Boas

Студент группы

Б9121-09.03.03пикд

Борик Роман Дмитриевич

Руководитель: ст. преподаватель ИМКТ

Кленин Александр Сергеевич



Формальная постановка задачи

- Изучить алгоритм "дерево van Emde Boas".
- Реализовать алгоритм "дерево van Emde Boas".
- Выполнить исследование на производительность.
- Результат работы выложить на **github**.

История

VEB дерево было изобретено командой во главе с голландским ученым-компьютерщиком **Питером ван Эмде Боасом** в 1975 году.



VEB дерево

Хранит целые числа в диапазоне $[0, U]$,

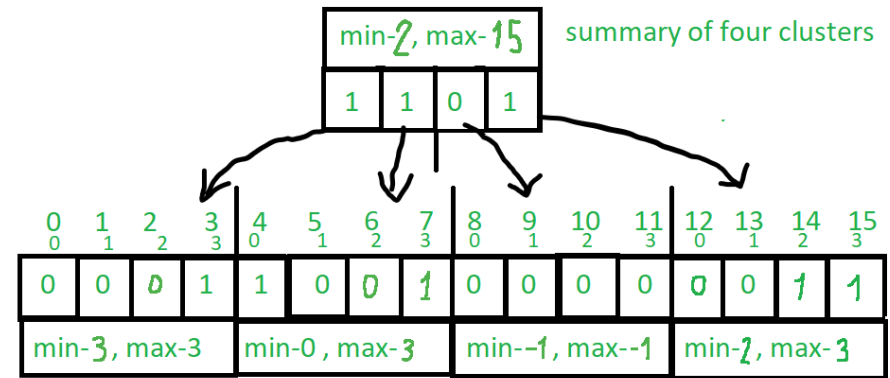
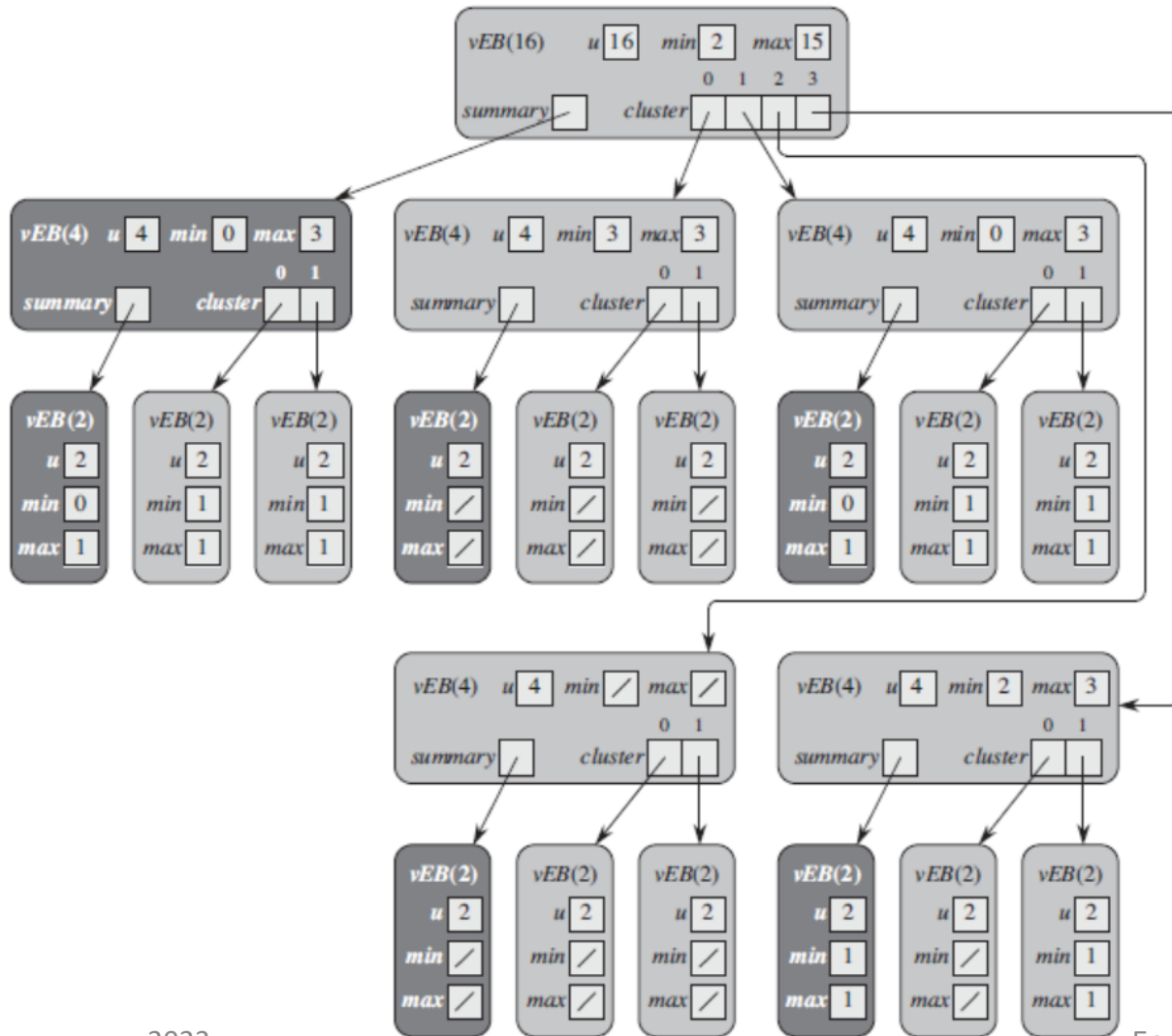
где U – число вида 2^k ,

где k – максимальное количество бит

Структура vEB дерева

- **universeSize** - размер дерева.
- **minimum** - минимальное значение.
- **maximum** - максимальное значение.
- **summary** - вспомогательное дерево.
- **cluster** - массив поддеревьев.

Пример vEB дерева



Summary хранит информацию о заполненности или пустоте детей текущего узла

Вспомогательные функции

- **High(x)** — число, указывающее номер кластера ключа x
- **Low(x)** — число, указывающее позицию x в его кластере
- **GenerateIndex(x, y)** — позиция ключа из его позиции в кластере y и его индекса кластера x .

Операции

- Minimum и maximum
- Find
- Insert
- Remove
- SuccessorVEB
- PredecessorVEB

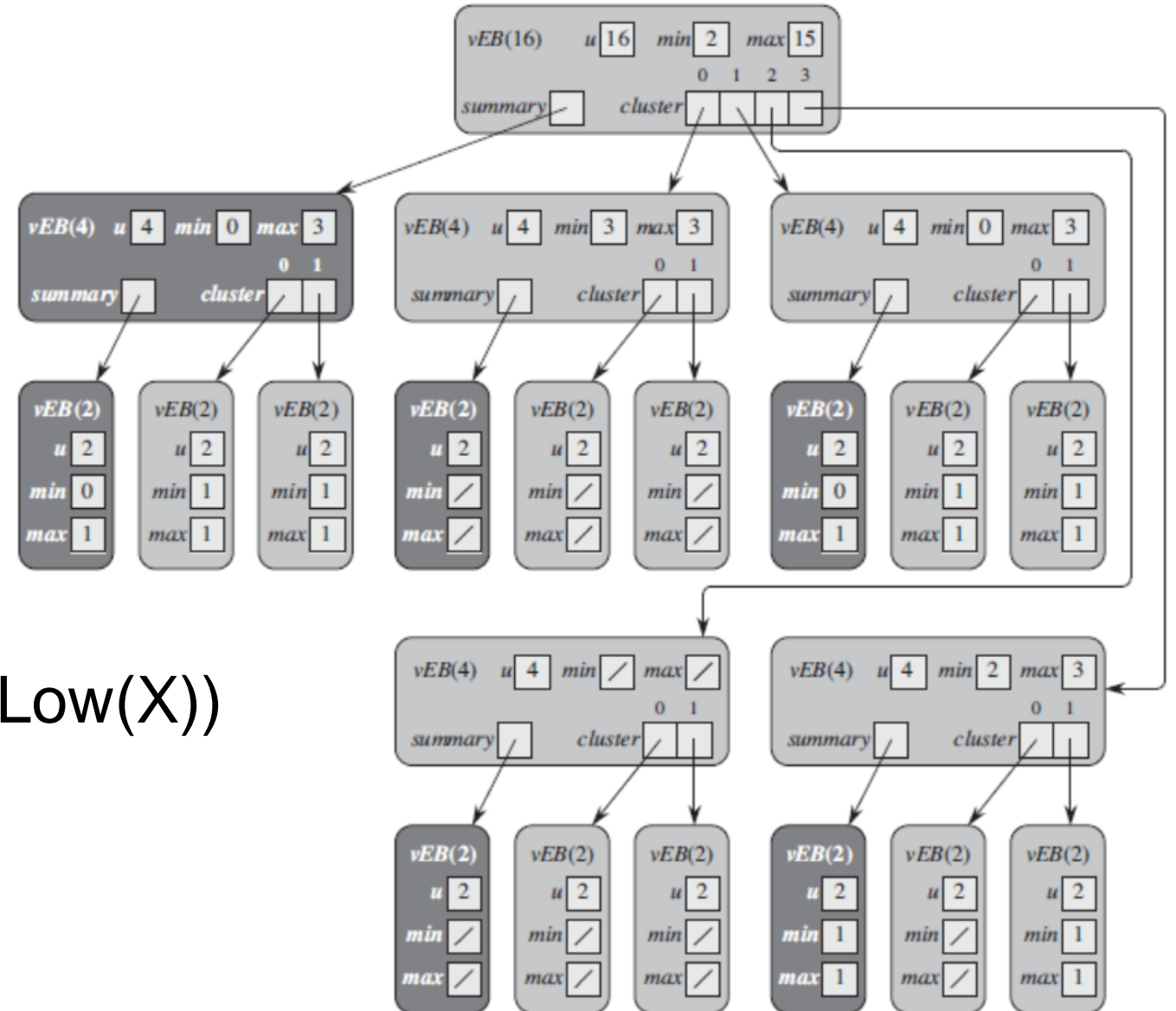
Find (поиск)

На вход подается число X

If $\text{min} == \text{null}$
 return false

If $X == \text{min}$ **or** $X == \text{max}$
 return true

return $\text{cluster}[\text{High}(X)].\text{Find}(\text{Low}(X))$

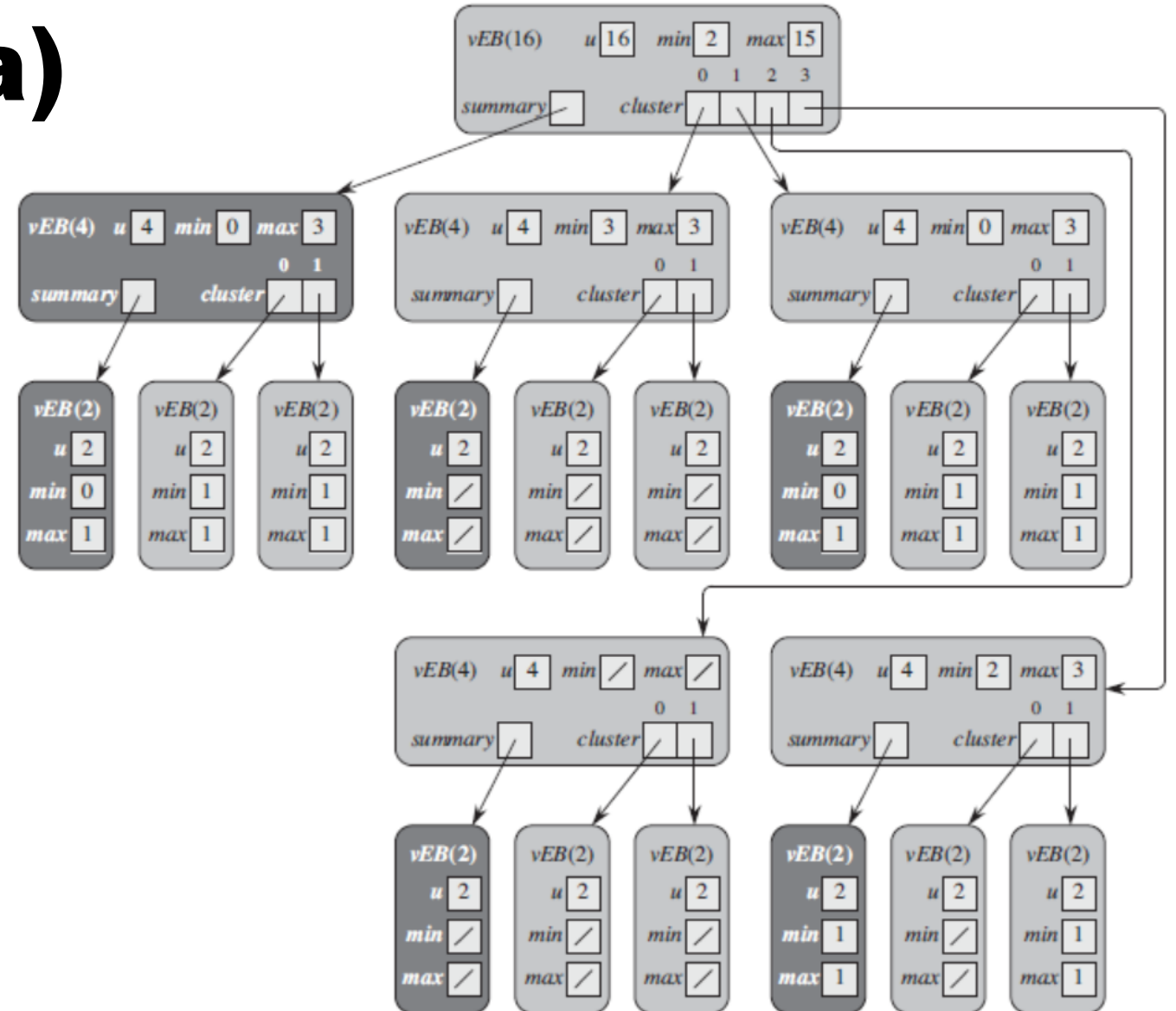


Insert (вставка)

На вход подается число X

```

if min == null
    min =  $X$ , max =  $X$ 
    return null
if  $X > \text{min}$ 
    swap( $X$ , min)
if universeSize > 2
    if cluster[High( $X$ )].min == null
        summary.Insert(High( $X$ ))
        cluster[High( $X$ )].Insert(Low( $X$ ))
if  $X > \text{max}$ 
    max =  $x$ 
    
```



Remove (удаление)

На вход подается число X

```
if min == X and max == X
    min = null
    max = null
    return
```

```
if min == X
    if summary.min == null
        min = max
    return
```

```
    X = GenerateIndex(summary.min, cluster[summary.min].min)
    min = X
```

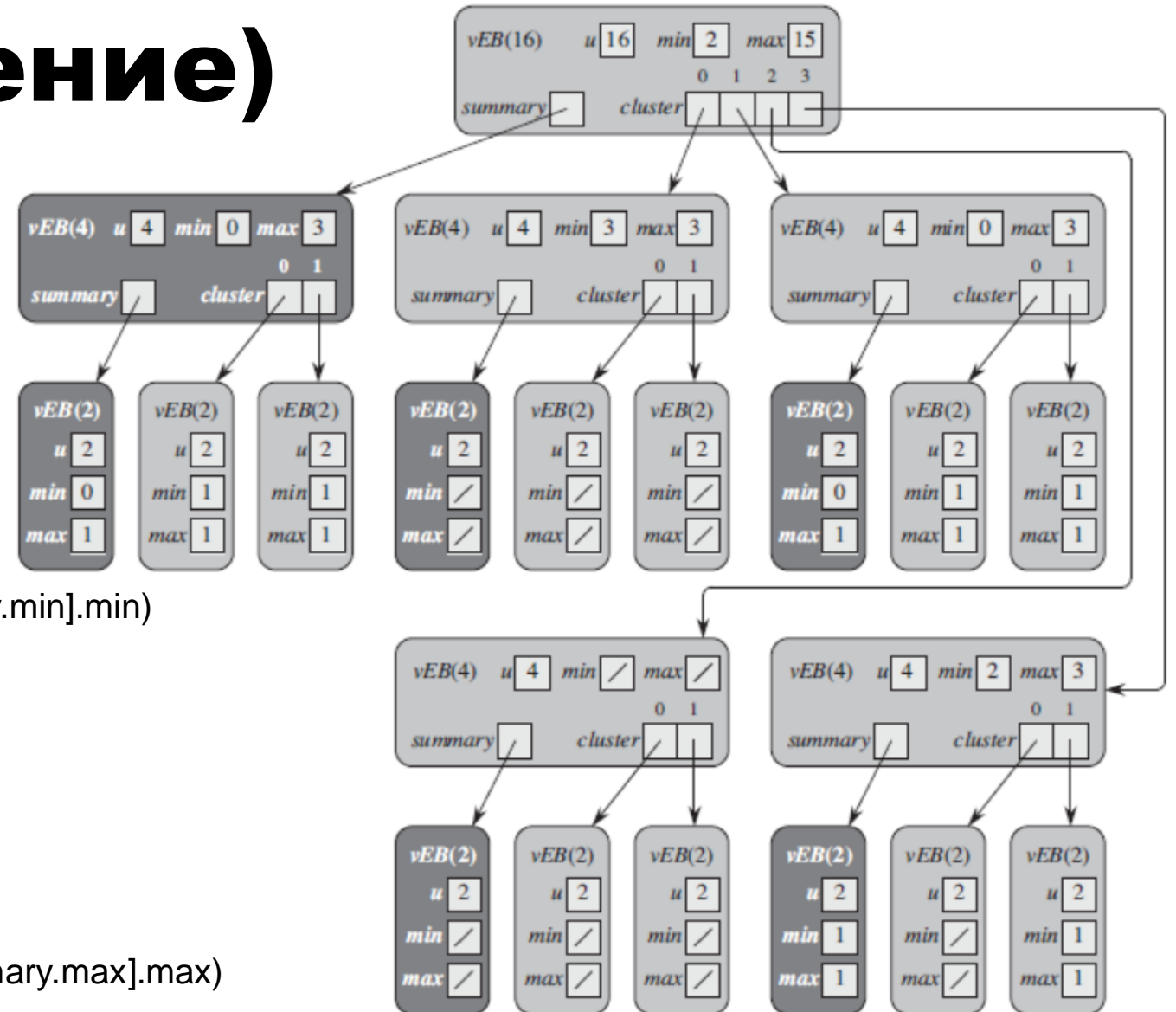
```
if summary.min == null
    return
```

```
cluster[High(X)].Remove(Low(x))
```

```
if cluster[High(X)].min == null
    summary.Remove(High(x))
```

```
if max == X then
    if summary.min == null
        max = min
    return
```

```
    max = GenerateIndex(summary.max, cluster[summary.max].max)
```

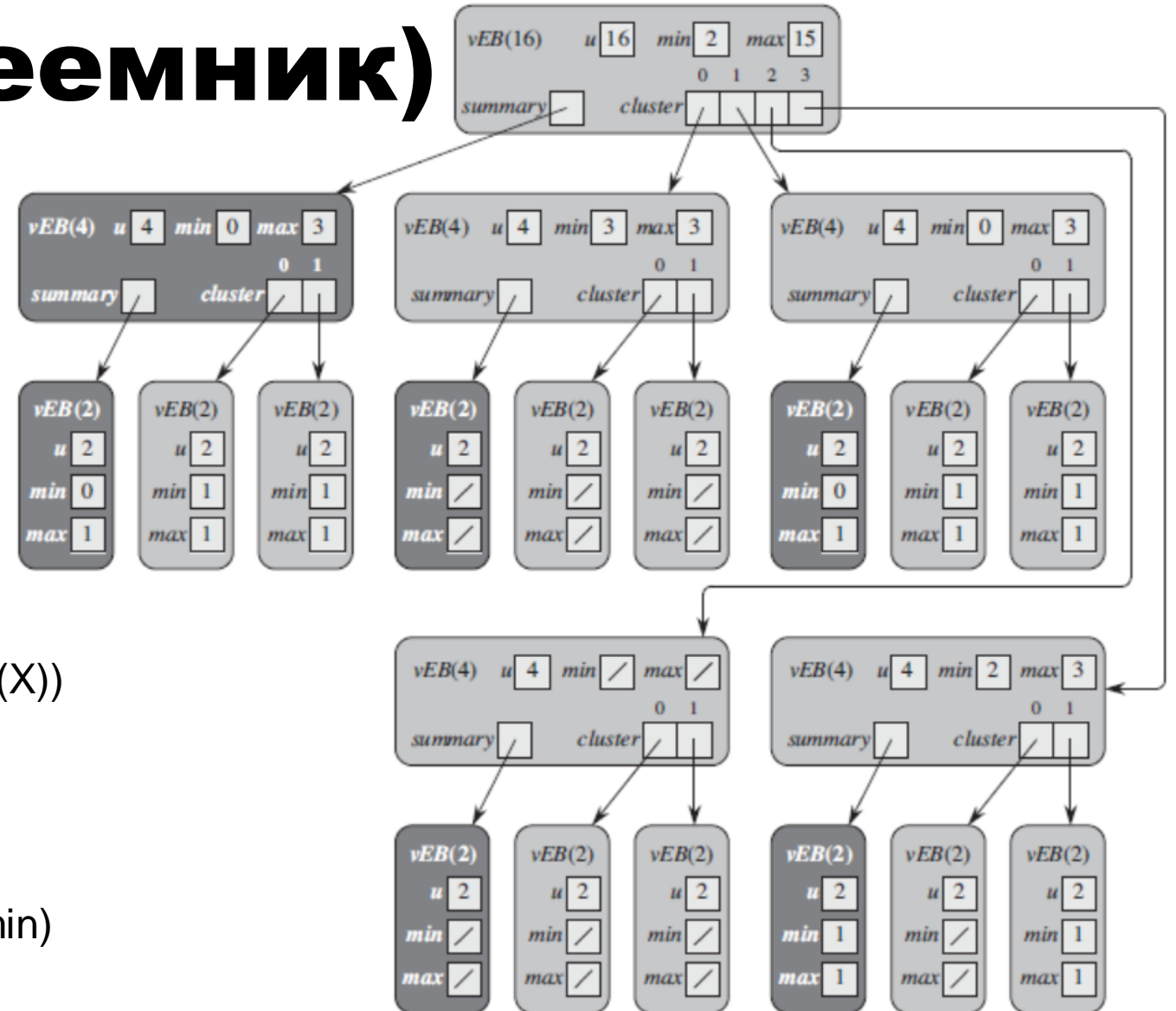


Successor (преемник)

На вход подается число X

```

if universeSize = 2
  if X == 0 and max == 1
    return 1
  else
    return null
if min != null and x < min
  return min
else
  temp = cluster[High(X)].max
  if max != null and Low(X) < temp
    temp = cluster[High(X)].Successor(Low(X))
    GenerateIndex(High(X), temp)
  else
    temp = summary.Successor(High(X))
    if temp == null
      return null
    else
      GenerateIndex(temp, cluster[temp].min)
  
```

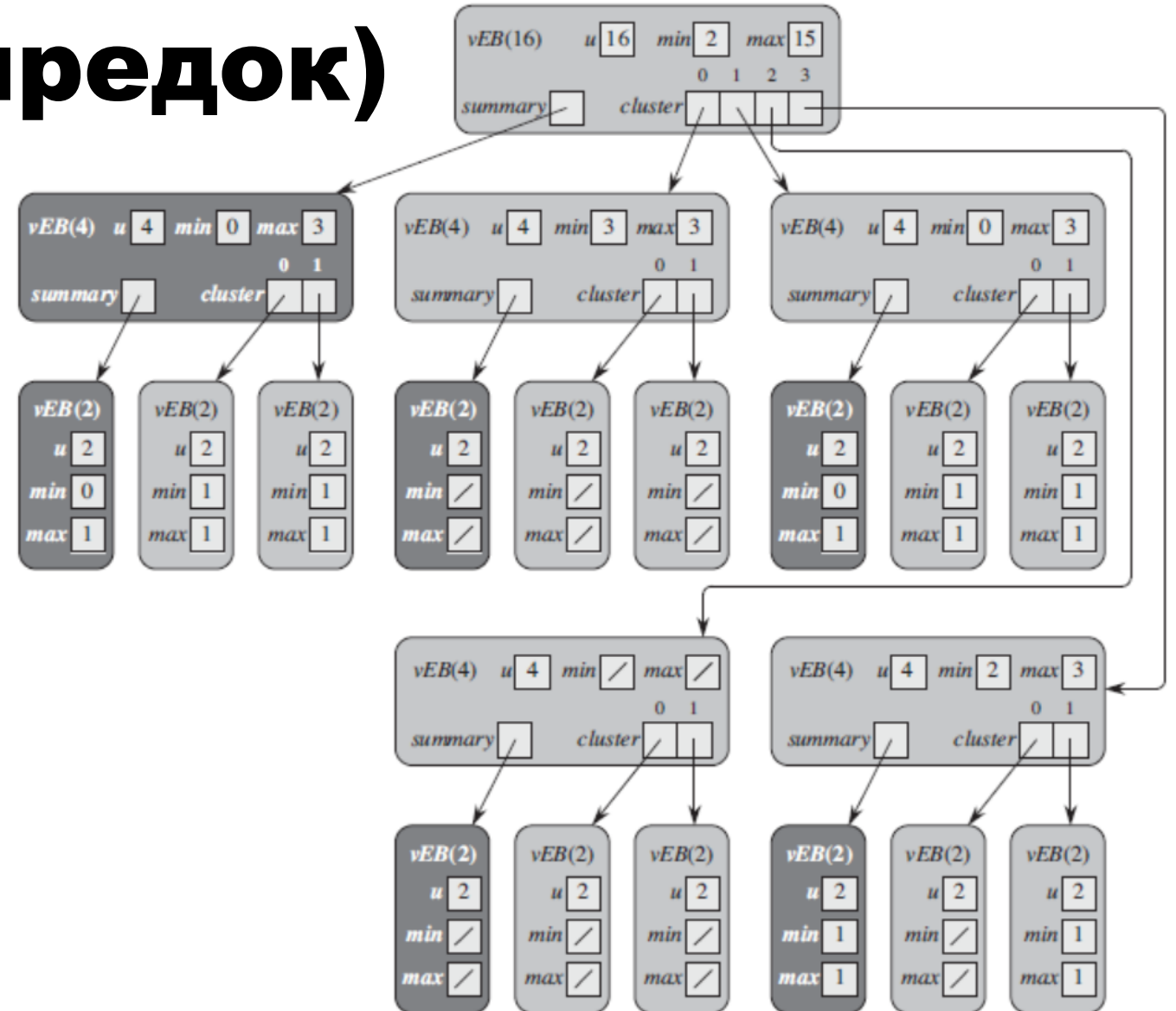


Predecessor (предок)

На вход подается число X

```

if universeSize = 2
  if X == 1 and min == 0
    return 0
  else
    return null
if max != null and X > max
  return max
else
  temp = cluster[High(X)].min
  if max != null and Low(X) > temp
    temp = cluster[High(X)].Predecessor(Low(X))
    GenerateIndex(High(X), temp)
  else
    temp = summary.Predecessor(High(X))
    if temp == null
      if min != null and X > min
        return min
      return null
    else
      GenerateIndex(temp, cluster[temp].max)
  
```



Тестирование производительности

Для сравнения использовались контейнеры

`std::set` и `std::unordered_set`

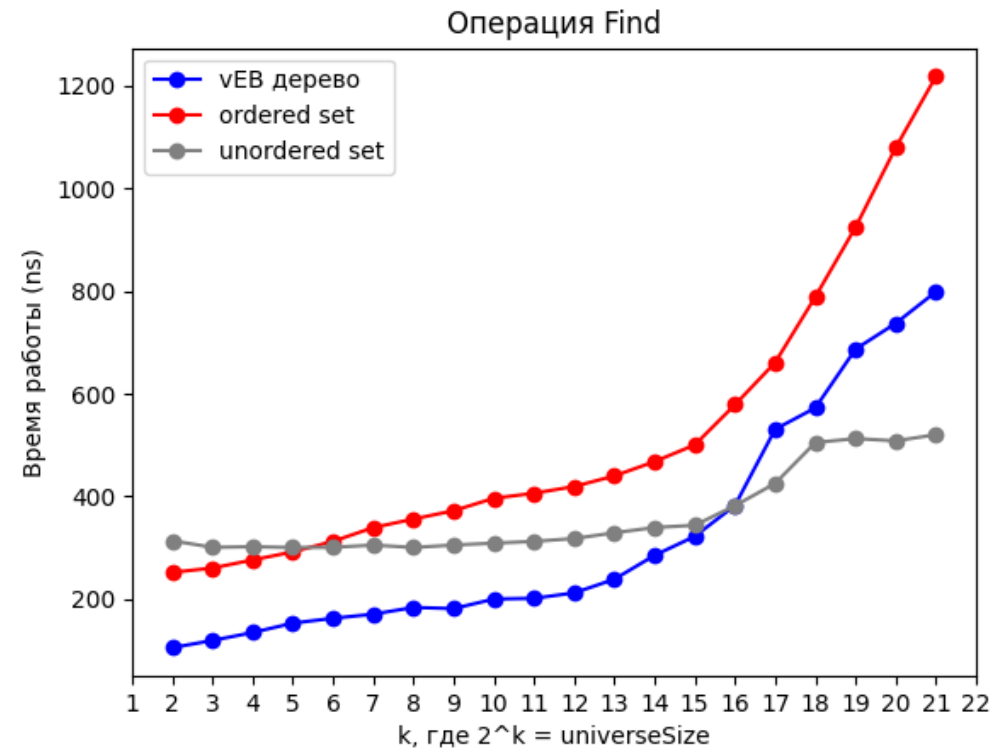
стандартной библиотеки шаблонов **STL**

и аналогичные функции **vEB** дерева.

Find

На вход подаётся набор значений в случайном порядке сгенерированный с помощью алгоритма Фишера-Йетса.

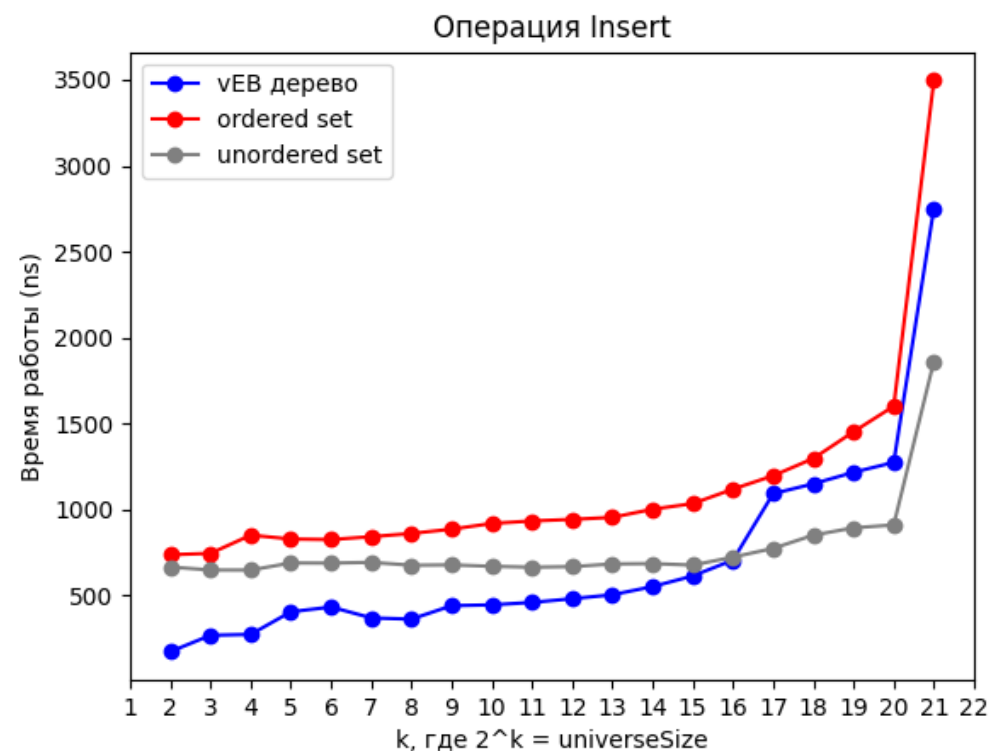
VEB дерево обгоняет `std::set`, но проигрывает в скорости `std::unordered_set` при больших значениях.



Insert

На вход подаётся набор значений в случайном порядке сгенерированный с помощью алгоритма Фишера-Йетса.

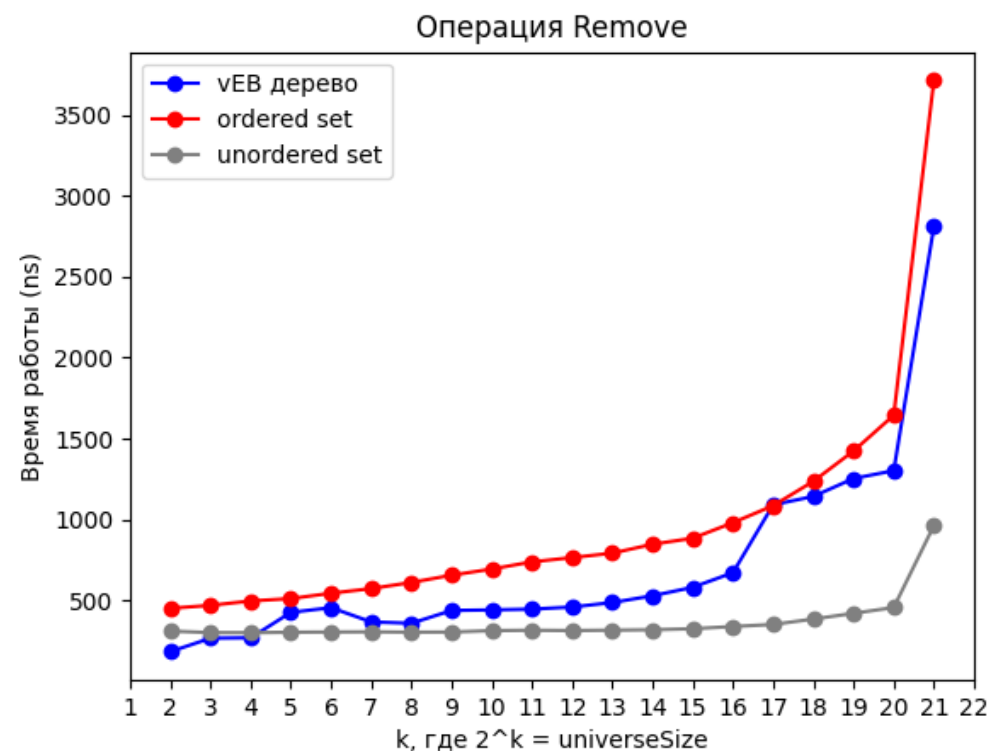
vEB дерево обгоняет **std::set**, но проигрывает в скорости **std::unordered_set** при больших значениях.



Remove

На вход подаётся набор значений в случайном порядке сгенерированный с помощью алгоритма Фишера-Йетса.

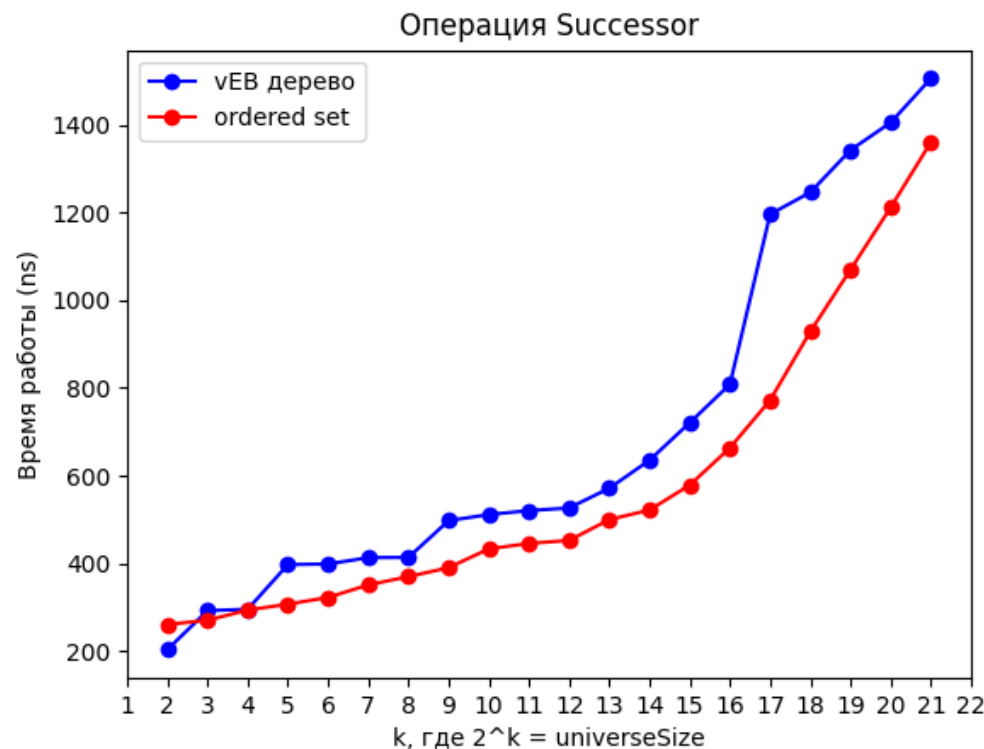
VEB дерево обгоняет `std::set`, но проигрывает в скорости `std::unordered_set`.



Successor

На вход подаётся набор значений в случайном порядке сгенерированный с помощью алгоритма Фишера-Йетса.

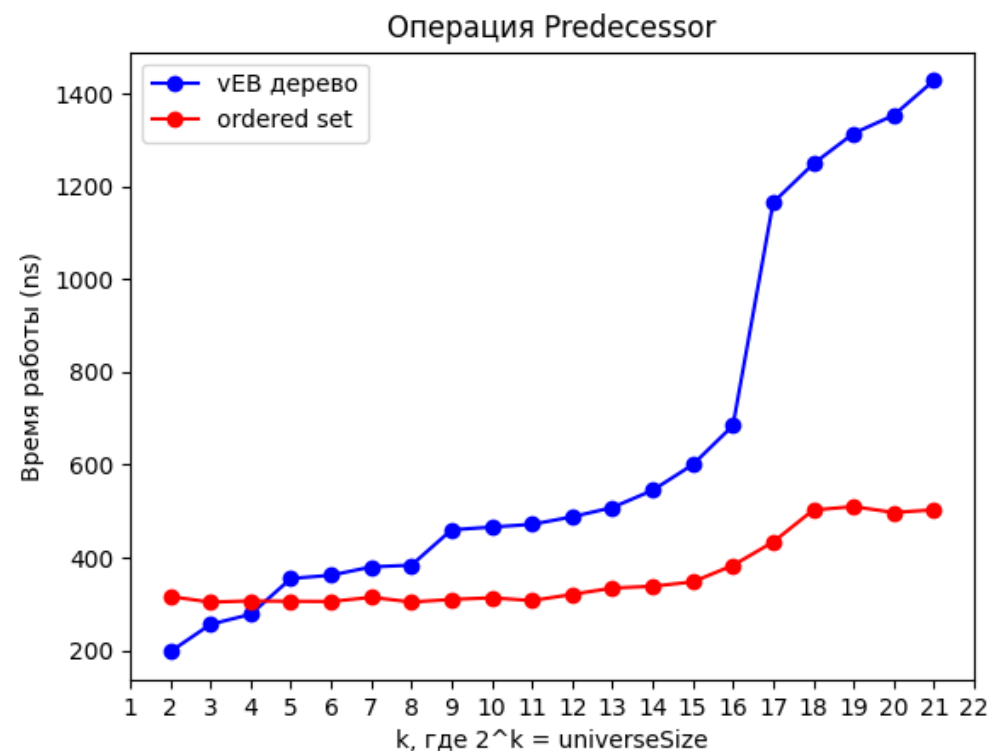
VEB дерево проигрывает в скорости `std::set`.



Predecessor

На вход подаётся набор значений в случайном порядке сгенерированный с помощью алгоритма Фишера-Йетса.

VEB дерево проигрывает в скорости `std::set`.



Итоги тестирования

Алгоритм дерево ван Эмде Боаса показывает отличные результаты.

Он обгоняет контейнер **std::set**, который является деревом поиска во всех операциях, кроме **Successor** и **Predecessor**.

И в некоторых случаях обгоняет контейнер **std::unordered_set**, который является хэш-таблицей.

Репозиторий `github`

Представлено по ссылке:

- Описание алгоритма
- Реализация алгоритма на C++
- Презентация
- Тестирующая система
 - Набор тестов на корректность
- Результаты анализа производительности

