Дерево van Emde Boas

Студент группы

Б9121-09.03.03пикд

Борик Роман Дмитриевич

Руководитель: доцент ИМКТ

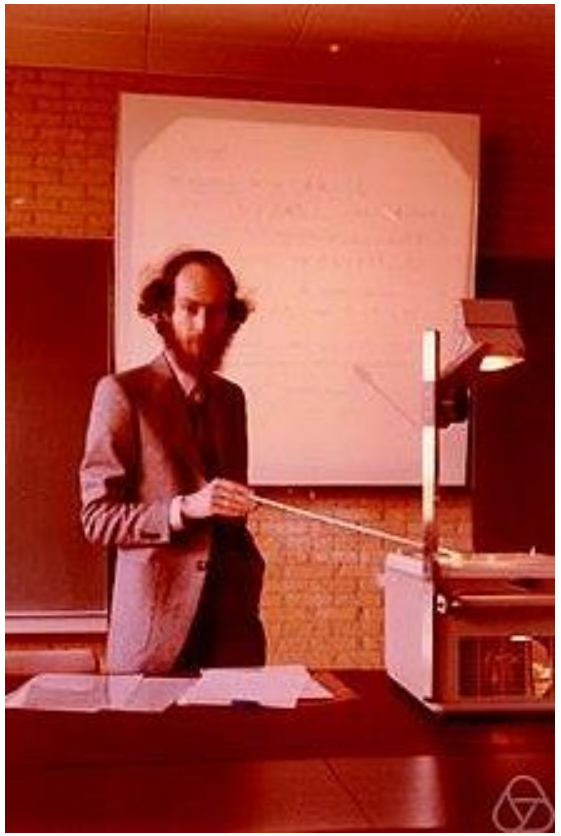
Кленин Александр Сергеевич

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 $[\mathbf{0}, \mathbf{U}]$,

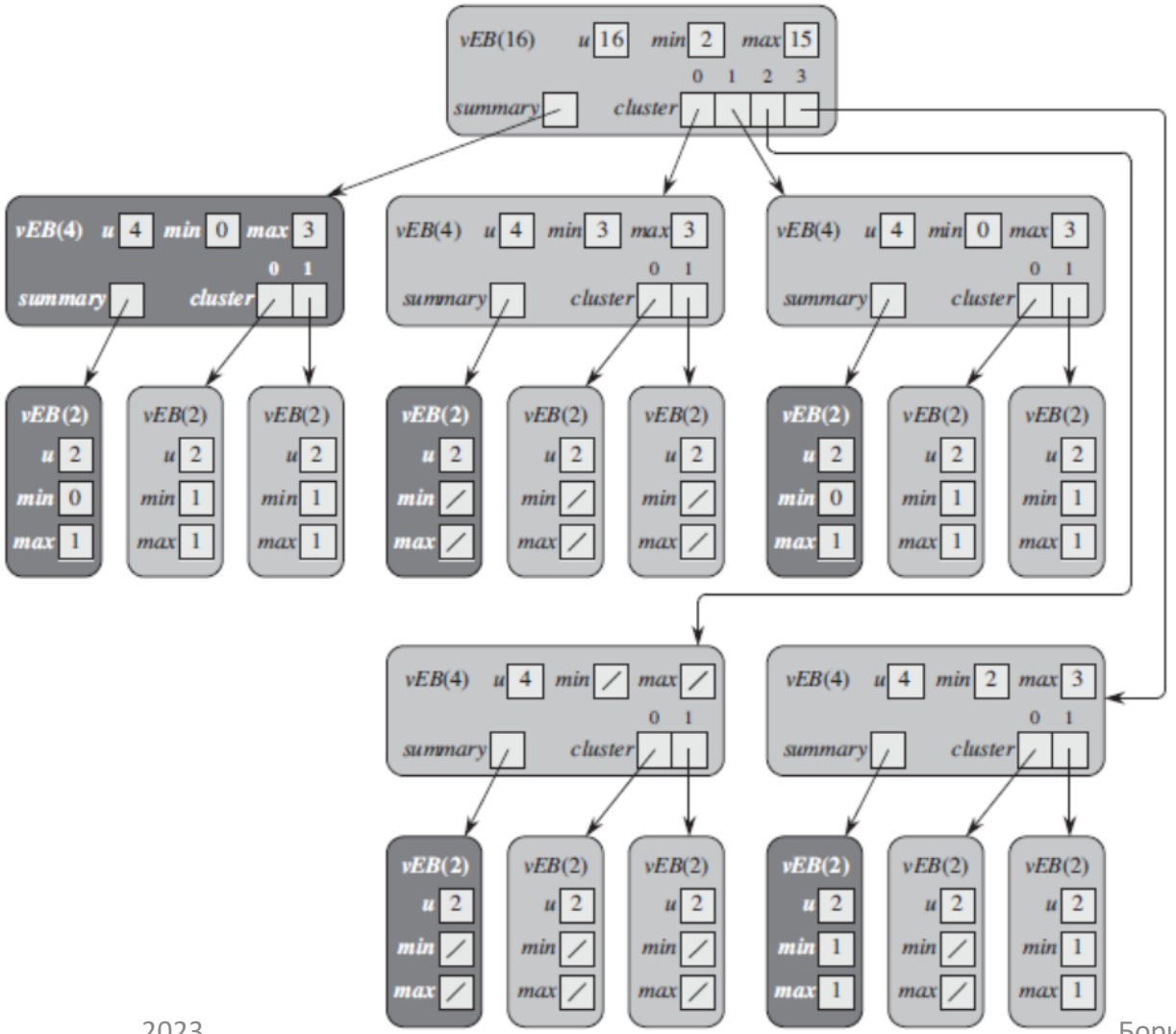
где $\mathbf{U}$ – число вида $\mathbf{2^k}$,

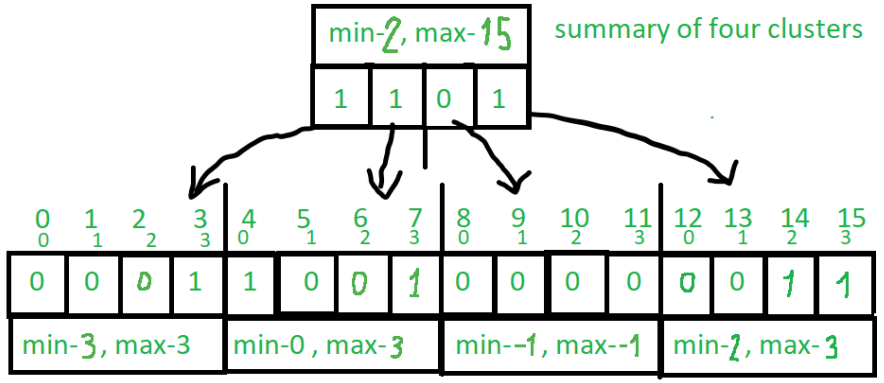
где $\mathbf{k}$ – максимальное количество бит

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

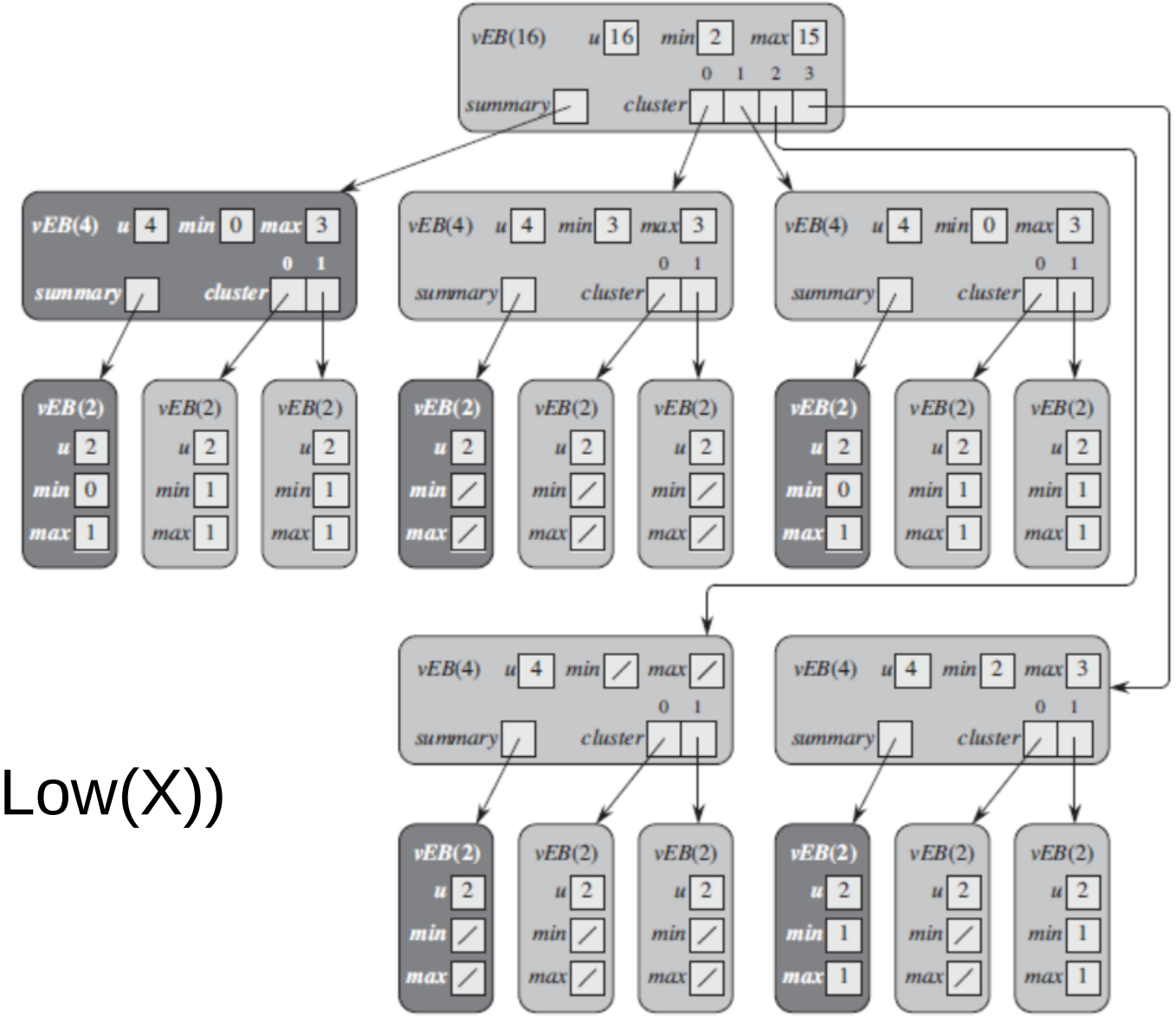
Операции

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

Find (поиск)

На вход подается число X

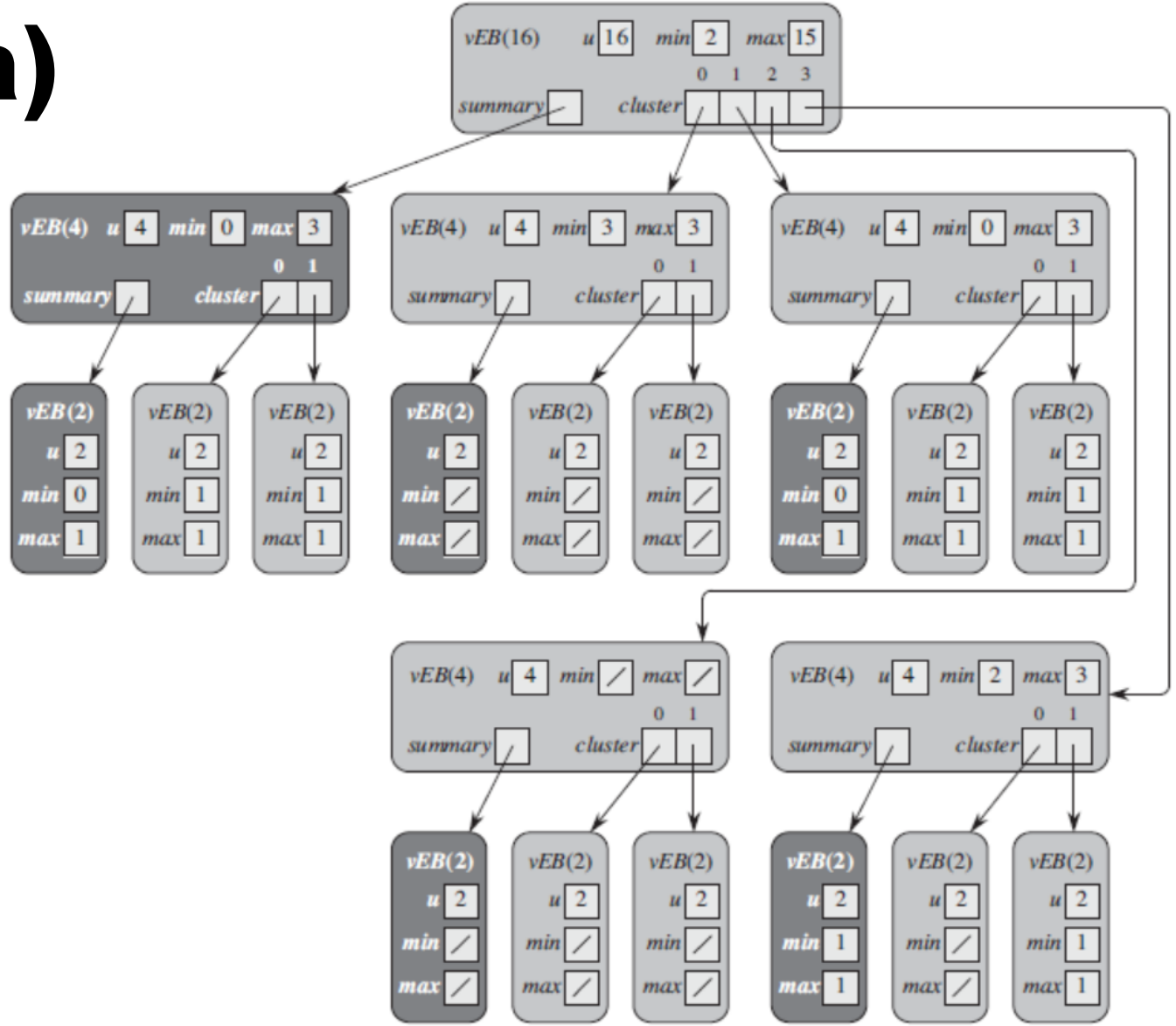
```
If Empty()
    return false
If X == min or X == max
    return true
return cluster[High(X)].Find(Low(X))
```



Insert (вставка)

На вход подается число X

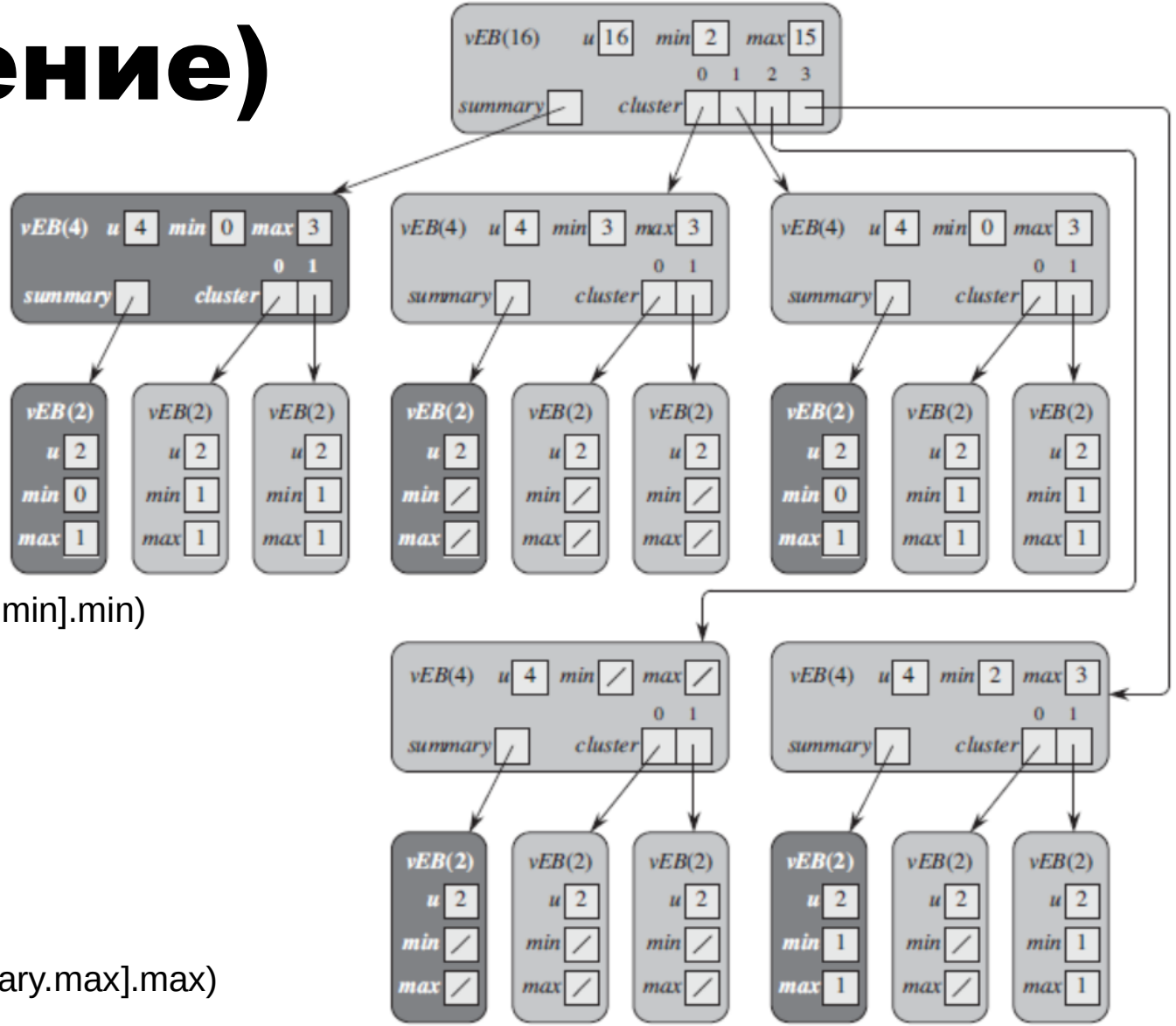
```
If Empty()
    min = X, max =  X
    return none
If X > min
    swap(X, min)
If USize > 2
    If cluster[High(X)].Empty()
        summary.Insert(High(X))
    cluster[High(X)].Insert(Low(X))
If X > max
    max = x
```



Remove (удаление)

На вход подается число X

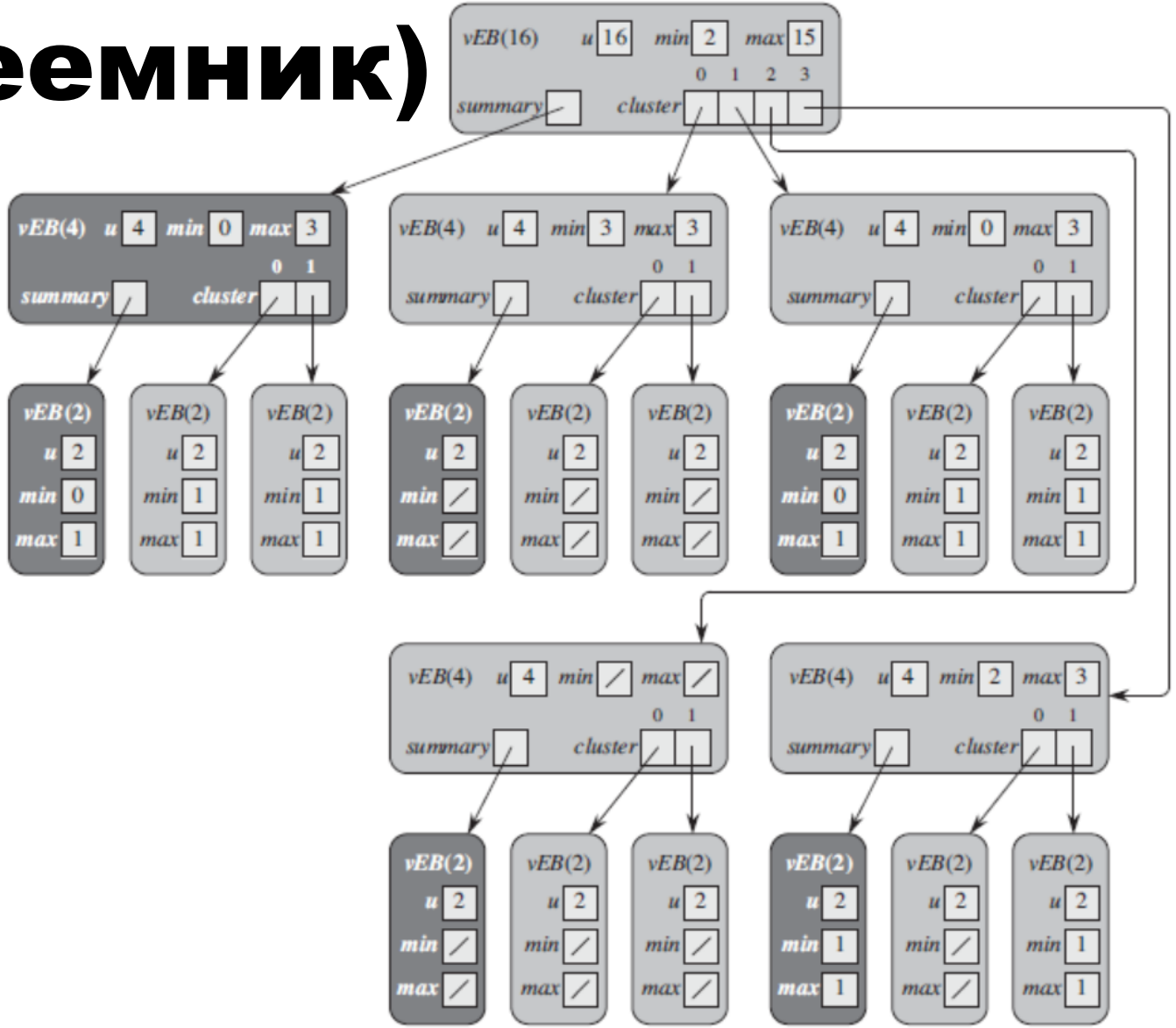
```
if min == X and max == X
    min = none
    max = none
    return
if min == X
    if summary.Empty()
        min = max
        return
    X = GenerateIndex(summary.min, cluster[summary.min].min)
    min = X
if summary.Empty()
    return
cluster[High(X)].Remove(Low(x))
if cluster[High(X)].Empty
 summary.Remove(High(x))
if max == X then
    if summary.Empty()
        max = min
        return
    max = GenerateIndex(summary.max, cluster[summary.max].max)
```



Successor (преемник)

На вход подается число X

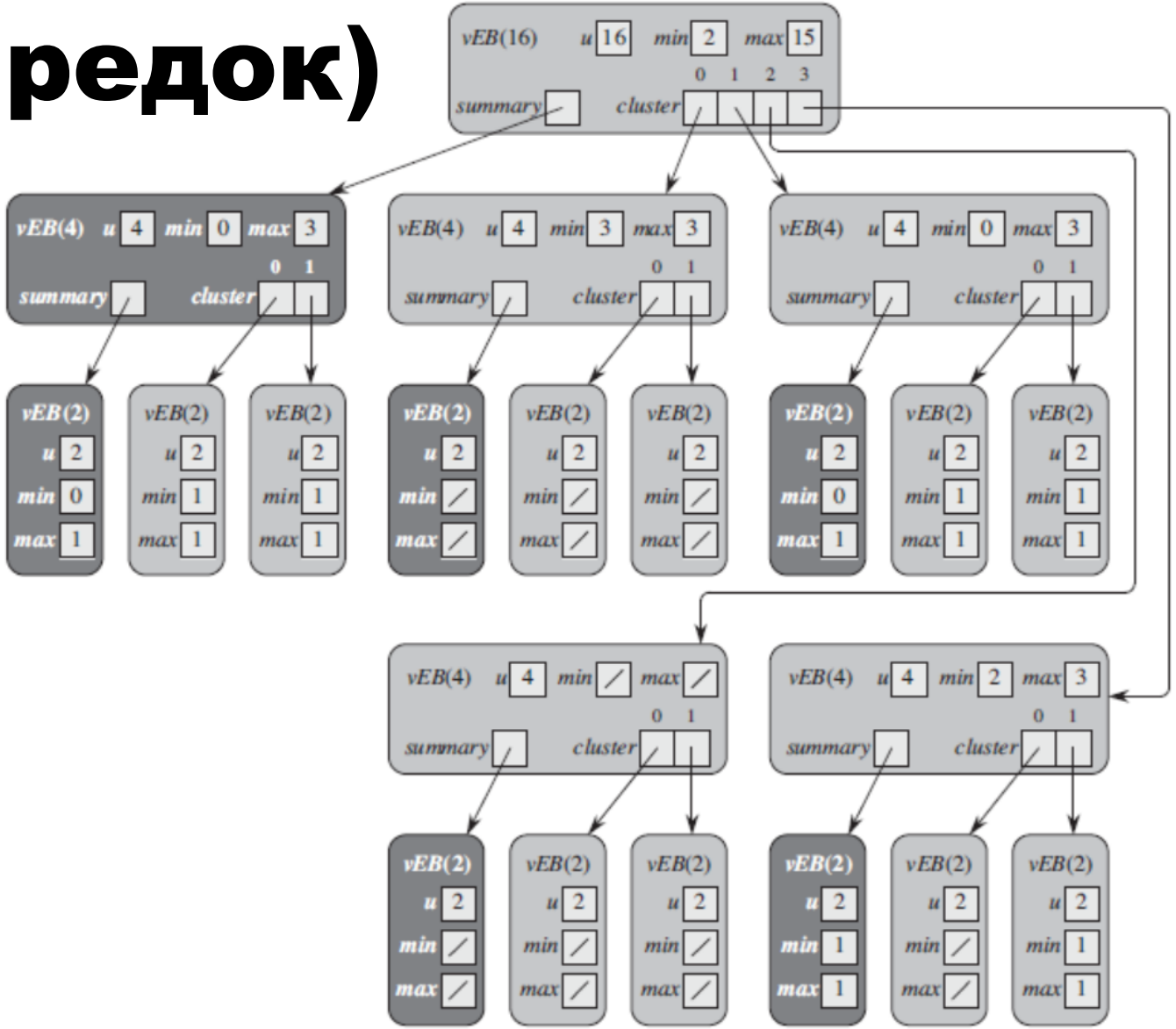
```
if USize = 2
    if X == 0 and max == 1
        return 1
    else
        return none
if min != none and x < min
    return min
else
    temp = cluster[High(X)].max
    if max != none and Low(X) < temp
        temp = cluster[High(X)].Successor(Low(X))
        GenerateIndex(High(X), temp)
    else
        temp = summary.Successor(High(X))
        If temp == none
            return none
        else
            GenerateIndex(temp, cluster[temp].min)
```



Predecessor (предок)

На вход подается число X

```
if USize = 2
    if X == 1 and min == 0
        return 0
    else
        return none
if max != none and X > max
    return max
else
    temp = cluster[High(X)].min
    if max != none and Low(X) > temp
        temp = cluster[High(X)].Predecessor(Low(X))
        GenerateIndex(High(X), temp)
    else
        temp = summary.Predecessor(High(X))
        If temp == none
            if min != none and X > min
                return min
            return none
        else
            GenerateIndex(temp, cluster[temp].max)
```



Тестирование

Для сравнения использовались контейнеры

std::set и **std::unordered_set**

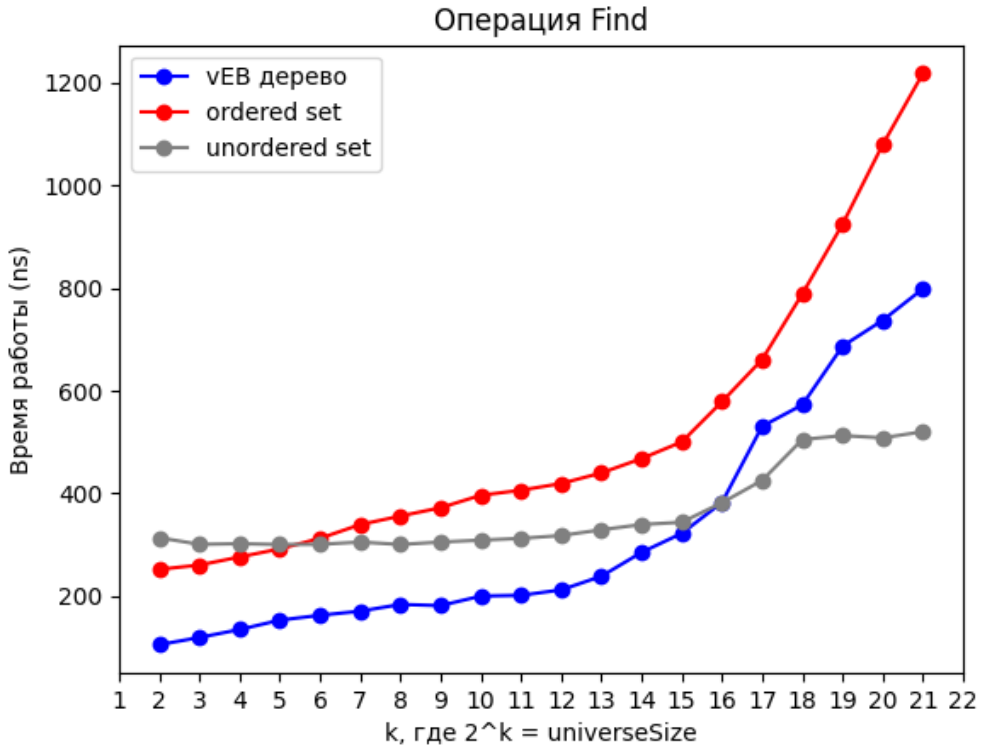
стандартной библиотеки шаблонов **STL**

и аналогичные **vEB дереву** функции.

Find

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

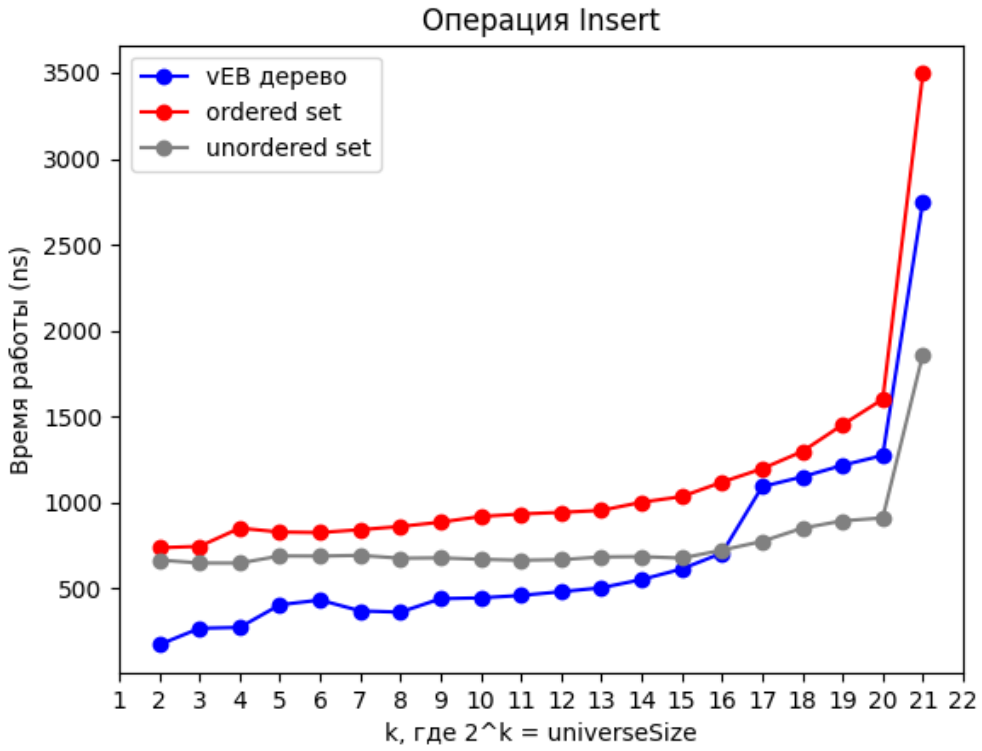
Из графика видно, что **vEB дерево** проигрывает в скорости **std::unordered_set** при больших значениях.



Insert

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

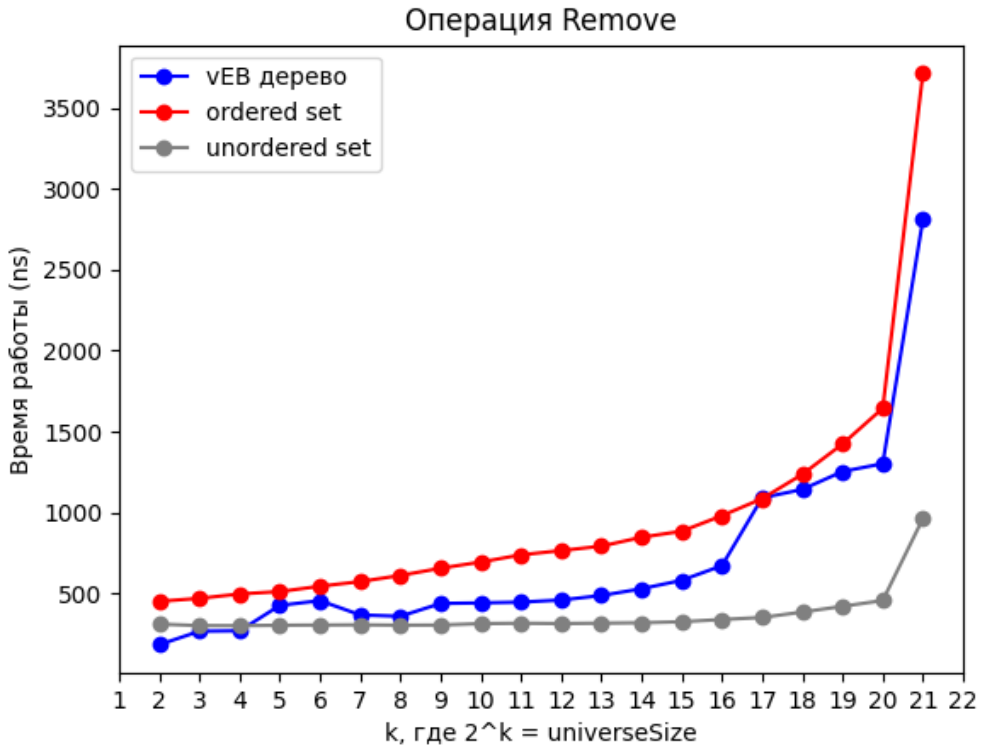
Из графика видно, что ситуация аналогична операции **Find**. **VEB дерево** проигрывает в скорости **std::unordered_set** при больших значениях.



Remove

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

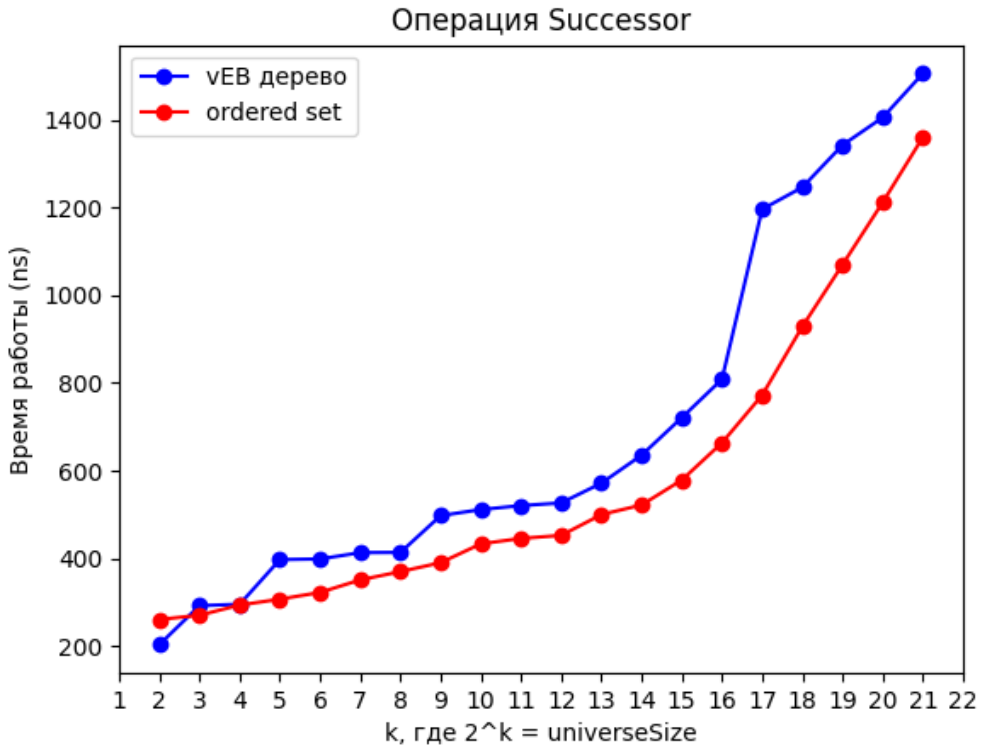
Из графика видно, что **vEB дерево** проигрывает в скорости **std::unordered_set** почти во всех случаях.



Successor

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

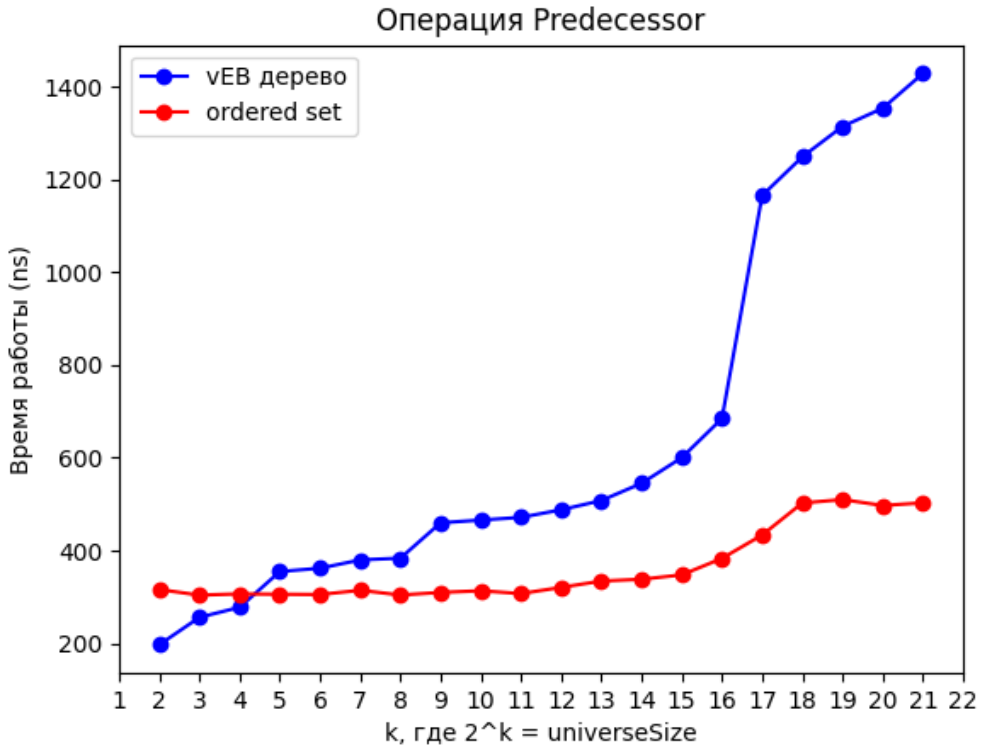
Из графика видно, что **vEB дерево** проигрывает в скорости **std::set.**



Predecessor

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

Из графика видно, что **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.