


[PERLIK](#) [БЛОГ](#) [КОМАНДЫ](#) [ПОПЫТКИ](#) [ГРУППЫ](#) [СОРЕВНОВАНИЯ](#)

Блог пользователя Perlik

Вытаскиваем дерамиду по неявному ключу из недр C++.

 Автор [Perlik](#), 6 лет назад, 

Всем привет! Не буду долгих предисловий тут писать, поэтому сразу к делу. Читая книгу Мейерса "Эффективное использование STL", я вдруг наткнулся на упоминание о наличии в некоторых версиях STL структуры данных [rope](#). Если кратко, то эта структура данных позволяет быстро вырезать/вставлять куски массива в произвольные позиции, аналогично декартовому дереву по неявному ключу (с аналогичной сложностью — подробности смотрите в статье на вики). Она иногда используется для обработки сверхдлинных строк.

Как выяснилось, горе действительно реализована в некоторых версиях STL, например в [SGI STL](#). Сразу замечу, что это наиболее полная документация по классу горе, которую мне удалось найти в сети. А теперь давайте разыщем горе в GNU C++. Поскольку кто-то ранее находил расширенную версию красно-черных деревьев в GNU, я подумал, почему бы и горе где-нибудь не завалиться. Для тестинга я взял вот [эту](#) задачу, в которой у меня уже была сдана дерамида по неявному ключу. После недолгого гугления получилось следующее решение:

```
#include <iostream>
#include <cstdio>
#include <ext/rope> //заголовочный файл с rope
using namespace std;
using namespace __gnu_cxx; //пространство имен, в котором находится класс горе
и все, что с ним связано
int main()
{
    ios_base::sync_with_stdio(false);
    rope<int> v; //используем как самый обычный stl контейнер
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i);
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope<int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope<int>::iterator it = v.mutable_begin(); it != v.mutable_end();
    ++it)
        cout << *it << " ";
    return 0;
}
```

Оно без всяких проблем зашло и уложилось в секунду, что в 2 раза медленнее варианта с рукописным декартовым деревом, но при этом с чуть более оптимальным расходом памяти. Полную документацию по всем методам можно найти по вышеприведенной ссылке на SGI STL. Судя по всему, в GNU полностью аналогичная реализация. Visual C++ как у нас не поддерживает rope.

→ Обратите внимание

Соревнование идет
[Microsoft Q# Coding Contest - Summer 2020 - Warmup](#)
 01:01:10
[Зарегистрироваться >](#)

До соревнования
[Codeforces Round #650 \(Div. 3\)](#)
 23:36:10

Нравится

 Like 3 people like this. [Sign Up](#) to see what your friends like.

→ obr.n198

 Рейтинг: **1160**
 Вклад: 0

- [Настройки](#)
- [Блог](#)
- [Команды](#)
- [Попытки](#)
- [Группы](#)
- [Переписка](#)
- [Соревнования](#)



obr.n198

→ Лидеры (рейтинг)

№	Пользователь	Рейтинг
1	MiFaFaOvO	3681
2	Um_nik	3567
3	tourist	3520
4	maroonrk	3421
5	apiadu	3397
6	300iq	3317
7	ecnerwala	3309
8	Benq	3283
9	LHiC	3229
10	TLE	3223

[Страны](#) | [Города](#) | [Организации](#)
[Всё →](#)

→ Лидеры (вклад)

№	Пользователь	Вклад
1	Errichto	194
1	antontrygubO_o	194
3	pikmike	181
4	vovuh	177
5	Ashishgup	171
6	Radewoosh	169
7	Um_nik	166
7	tourist	166
9	ko_osaga	163
10	McDic	162

Что касается применения, то есть несколько особенностей. Из документации SGI STL следует, что класс `gore` плохо дружит с изменениями отдельных элементов последовательности (поэтому методы `begin()` и `end()` возвращают `const_iterator`. Чтобы получить обычный итератор, необходимо вызывать `mutable_begin()` и `mutable_end()` соответственно.). Также нельзя просто взять и присвоить значение `i`-ому элементу последовательности (см. код ниже для подробностей). Но насколько это "плохо", я не знаю. По идее за классический логарифм от числа элементов. В то же время конкатенация (операция `+=`) вообще работает за $O(1)$ (не считая затрат на создание объекта справа, конечно).

Особенности индексирования можно увидеть в этом коде: <http://pastebin.com/U8rG1tfu>. Поскольку разработчики очень не хотят, чтобы мы меняли отдельные элементы контейнера, оператор `[]` возвращает ссылку на константу, но специальный метод все же есть. Данный код работает столько же, сколько и предыдущий. И да, забыл упомянуть, что все итераторы `RandomAccess`, что не может не радовать.

Если кто захочет потестировать контейнер на более серьезных задачах, расскажите в комментариях. По моему ощущению, у нас теперь есть быстрый массив с логарифмическим временем выполнения всех операций и как обычно большой константой :)

UPD: сдал еще [задачу](#). Здесь мало запросов, но последовательность подлиннее. Зашло тоже без проблем. Еще наткнулся на такую особенность, что `erase` может принимать только 1 индекс типа `size_t` (хотя метода с такой сигнатурой нет). И я не очень понимаю, что он делает в этом случае. Из-за этого можно случайно набазить, когда нужно удалять 1 элемент.

с++, rope, treap

+285

Perlik

6 лет назад

48



Комментарии (48)

[Написать комментарий?](#)

6 лет назад, # | ☆



Break-Neck

Объясните дураку: в статье на Википедии оцепятка и в определении веса имеется ввиду "длина своей строки и сумма **длинн подстрок** в своем левом поддереве"? Если нет, то объясните, почему там на картинке цепочка $9 \rightarrow 9$, если при переходе на верхний уровень 6 и 3 должны были учестся еще раз?

→ Ответить

6 лет назад, # | ☆



Break-Neck

"В то же время конкатенация (операция `+=`) вообще работает за $O(1)$ (не считая затрат на создание объекта справа, конечно)." — либо это правда, и они не балансируют дерево (со всеми вытекающими), либо все за $O(\log N)$.

Еще смущает то, что оно проигрывает декартовому дереву — там рандомизированный псевдо-баланс, здесь же при балансировке должно быть все в рамках. Неужели балансировка занимает столько времени, что это дает такое замедление? Но ведь другие деревья (RB, AVL) выигрывают у декартового. Вообще, непонятно.

→ Ответить

6 лет назад, # | ☆

← Rev. 3

+10

действительно сверхдлинные строки можно хранить. [этот](#) код в запуске отрабатывает за 300 мс

upd. [этот](#) за 140мс

upd2. [292E - Копирование данных](#) сдать не удалось

→ Ответить

6 лет назад, # | ☆

0

Попробовал вариант с `replace` тоже не прокатило. Так что не

Все →

→ Найти пользователя

Хэндл:

Найти

→ Прямой эфир

eva_fan → [Mifafaovo VS Tourist?](#)

MikeMirzayanov → [Эксперимент: обучение программированию с нуля через задачи](#)

kazakhThunder → [Need help in proving time complexity of code.](#)

Stepavly → [Codeforces Round #650 \(Div. 3\)](#)

aop → [Vasya and good Sequences](#)

mohammedehab2002 → [Codeforces round #649 editorial](#)

Cache → [Invitation to CodeChef June Long Challenge 2020](#)

Nickolas → [Announcement: Microsoft Q# Coding Contest – Summer 2020](#)

hungrytired → [Love of CF with 1337?](#)

shejibri → [One problem with several strange and pretty solutions.](#)

Jellyman102 → [An Efficient \(and quite common\) Way to Navigate Grid Problems \[C++\]](#)

LM10_Piyush → [Why std::cout gave me runtime error?](#)

Jellyman102 → [An Explanation of ckmin and ckmax \[C++\]](#)

vovuh → [Разбор Codeforces Round #642 \(Div. 3\)](#)

Stepavly → [Codeforces Round #644 \(Div. 3\) Editorial](#)

pikmike → [Разбор Educational Codeforces Round 88](#)

BledDest → [Kotlin Heroes: Episode 4 — Editorial](#)

mohammedehab2002 → [Codeforces Round #649 \(Div.2\)](#)

TheOneYouWant → [Editorial — Codeforces Round #646](#)

pikmike → [Разбор Educational Codeforces Round 89](#)

Okrut → [Codeforces Round #647 Editorial](#)

FastestFinger → [Editorial — Codeforces Round #648](#)

arujbansal → [An Introduction To Difference Arrays](#)

fantactic → [Здравствуй, не могли бы подсказать как удалить свой аккаунт?](#)

pikmike → [Разбор Educational Codeforces Round 86](#)

Детальнее →



Perlik

тестировал вариант с `erase`, тоже не прокатило. так что не все так хорошо) А там обычный treap заходит?

→ [Ответить](#)



Bugman

6 лет назад, # ^ | ☆

treap тоже не будет заходить, так как в задаче не вырезал/вставил, а скопировал/заменял.

→ [Ответить](#)



Perlik

6 лет назад, # ^ | ☆

Интересно, как оно ведет себя в тех задачах, где проходит treap. Я просто давно уже не участвую и кроме приведенной в посте задачи не помню задачи на неявный treap.

→ [Ответить](#)



AlexanderBolshakov

6 лет назад, # ^ | ☆

Будет. Вот [здесь](#) все очень подробно объяснили, от себя мне добавить нечего.

→ [Ответить](#)



Bugman

6 лет назад, # ^ | ☆

мм, ну ок. просто для меня персистентное декартово дерево — уже не декартово дерево.

→ [Ответить](#)

6 лет назад, # | ☆

← Rev. 2 ▲ +14 ▼



MikhailRubinchik.ru

На сайте мосЧФ параллельно с реальным чф можно играть в онлайн. Так вот на мосчф была задача на обработку строк (кажется, в 2009), мы её сдали с помощью горе как раз :)

Хотя говорят, что там заходил метсру (квадрат с маленькой константой)

→ [Ответить](#)



pva701

6 лет назад, # | ☆

не зашло

→ [Ответить](#)



Perlik

6 лет назад, # ^ | ☆

А вот здесь декартово у меня залетело. По-видимому при 10^5 запросах и ограничении в секунду горе лучше не использовать.

→ [Ответить](#)

6 лет назад, # | ☆

▲ +16 ▼



MikhailRubinchik.ru

Пока руки не дошли прочитать статью на вики или код stl. Кто-то это сделал? На каком конкретно бинарном дереве реализована структура горе?

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ 0 ▼



Perlik

Вот [здесь](#) наиболее полное описание реализации, что мне удалось найти. Вот какое именно бинарное дерево используется, тут не сказано. Однако, в коде (в `gorimpl.h`) я нашел список чисел Фибоначчи рядом с `_S_max_rope_depth` и `_S_min_len` функций `_S_balance`. Я не в курсе, как реализуется горе, но может это АВЛ?

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ 0 ▼

ДРП как-то жестко по неявному ключу. Вот спейс-дерево



Avitella

А вот как-то жестко по полному ключу. Вот сплей-дерево умеет и за $O(1)$ конкатенироваться и все остальное перечисленное.

P.S. не знаю на чем горе, просто флужу

→ [Ответить](#)

6 лет назад, # [^](#) | [☆](#)

▲ 0 ▼



Perlik

Никто не будет писать splay, потому что у него хорошая только амортизированная оценка. Да и возможно есть некоторые нюансы по сравнению с обычными деревьями поиска, препятствующие использованию splay.

→ [Ответить](#)



SergeyLazarev

6 лет назад, # [^](#) | [☆](#)

▲ +16 ▼

Хорошая амортизированная оценка — это, вообще-то, хорошо. У добавления элемента в вектор тоже только амортизированная оценка константная.

→ [Ответить](#)

6 лет назад, # [^](#) | [☆](#) ▲ 0 ▼



Perlik

Да, это я ляпнул, не подумав. Но все равно никогда не видел, чтобы Splay-дерево использовалось в какой-либо стандартной библиотеке. Оно хоть и изящней всех остальных, но проигрывает тому же красно-черному, например.

→ [Ответить](#)

6 лет назад, # [^](#) | [☆](#) -34 ▼

→ [Ответить](#)



Break-Neck

Комментарий скрыт по причине большого числа негативных отзывов о нем, нажмите [здесь](#) для его просмотра

6 лет назад, # [^](#) | [☆](#)

▲ 0 ▼

Вот сплей-дерево умеет и за $O(1)$ конкатенироваться и все остальное перечисленное.



Zlobober

$O(\log size)$ ведь? То есть я однажды слышал, что якобы *merge*-и делаются за амортизированные $O(1)$, но не очень понимаю, с чего бы быть такому. Можно поподробнее?

→ [Ответить](#)

6 лет назад, # [^](#) | [☆](#) Rev. 2

▲ 0 ▼



Avitella

На самом деле я не знаю такую амортизированную оценку. Я просто подумал, что нам ничего не мешает создать вершину-корень, в которой будет пустая строка и прицепить слева и справа то, что конкатенируем. Свойства вроде как не нарушаются.

P.S. Нашел такую ссылку:
<http://habrahabr.ru/post/144736/>

→ [Ответить](#)

**PavelKunyavskiy**

6 лет назад, # | 0

Да. Но во первых плодятся лишние вершины на пустые строки, во вторых растёт потенциал, который есть в оценке. Поэтому все-таки лог.

→ [Ответить](#)**Avitella**

6 лет назад, # 0



Это еще почему?
Оценка конкатенации
— явно константа.

→ [Ответить](#)

6 лет назад, # +3



То, что ты предложил, просто ломает все остальные асимптотики. Ты же не мёрджишь так декартовы деревья? Попробуй приконкатенировать 10^5 одноэлементных кусочков по очереди следующий к предыдущему. Если следовать твоему алгоритму, получится бамбук глубины 10^5 .

→ [Ответить](#)**Zlobober**

6 лет назад, # 0



Амортизированное время = реальное + изменение потенциала в правильную сторону. На этом строится оценка Splay в общем-то. Потенциал, который есть в доказательстве, которое я видел увеличивается на $O(\log n)$ при merge. Это даёт логорифмическое время в этом смысле.

→ [Ответить](#)**PavelKunyavskiy**

6 лет назад, # |

← Rev. 2 +97

There is also an order-statistics tree in the SGI library included in STL.

Sample code: <http://ideone.com/QuiYER>

→ [Ответить](#)**adelnoble****MikhailRubinchik.ru**

6 лет назад, # |

Оо

→ [Ответить](#)

+43



Perlík

6 лет назад, # ^ | ☆

▲ 0 ▼

Cool! I have also found the patricia trie and splay tree there as well as the binomial heap and some other data structures, but it seems to me that they are incomplete.

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ +87 ▼

Well, Last night I was investigating contents of pb_ds (Politics based data structures) library, included by default in g++. I've found that implementations of binary search trees there are almost as we need: they support traveling with node iterators (such as moving to the left/right child, dereferencing and everything we need), splitting, mergeing and even containing additional information in each node with its recalculation after every change of tree structure (such as size of subtree, minimum or maximum)!

Here are two examples of what we can do with this library.

[First one](#). Basic examples of usage. Look over it, it's very interesting! It also compiles here on Codeforces.

[Second one](#) with solution of RMQ problem using this tree. When I was writing it I was thinking like "That's really great! That can replace treap as default binary search tree that people often write on contests! That's very cool!". But...

The main problem is that splitting is implemented in linear time because of very stupid bottleneck: after split they set the size of second subtree as

```
std::distance(other.begin(), other.end()) , that works in linear time. =(
```



Zlobober

There is also implementation of 4 types of heap. They support iterators to fixed elements of the heap, that makes us possible to write, for example, Dijkstra with heap algorithm. But here is a [benchmark](#), that shows that it is still 20-30% slower then usual Dijkstra+set implementation.

[Patricia tree](#) was a surprise for me, essentially after I looked in wikipedia and found a compressed trie (!) picture. But in fact it is another associative container for storing keys in increasing order:

Patricia trees have excellent lookup performance, but they do so through maintaining, for each node, a miniature "hash-table". Their space efficiency is low, and their modification performance is bad.

(from [official documentation of that library](#))

There is also forward-linked list and bunch of hashmaps (I didn't tested yet).

That trees are really what we could use in our contests, If it hadn't linear-time split. I've written an e-mail to the developers of that library with some questions about why they didn't implement it more efficient. Waiting for the answer from them.

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ 0 ▼



adelnobil

What's their point of doing this in Linear time when it can be done in constant time? This is really weird! Nice effort by the way I really appreciate it :)

I also found a skip-list implementation, it allows finding, searching, deleting all in logarithmic time.

→ [Ответить](#)



adelnobel

6 лет назад, # ^ | ☆

▲ 0 ▼

Excuse me, but in which file did you find the linear split method ? As from what I can see in this page

http://gcc.gnu.org/onlinedocs/libstdc++/ext/pb_ds/tree_based_containers.html

Scroll to the end of the page right to (Additional Methods)

You will find this: "These methods are efficient — red-black trees are split and joined in poly-logarithmic complexity; ordered-vector trees are split and joined at linear complexity. The alternatives have super-linear complexity."

So only ordered-vector trees are splitted in linear time, you should be alright if you use red-black tree as the underlying data structure by specifying the `rb_tree_tag` when creating the object?

I'll look into the implementation of the split method inside..

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ +9 ▼

No, split will be anyway linear-time. After tree-depend implementation of split function there is a call of general function for all binary trees `split_finish(other)`, where follows next line:

```
...
other.m_size = std::distance(other.begin(),
other.end());
...
```



Zlobober

([/ext/pb_ds/detail/bin_search_tree_/split_join_fn_imps.hpp](#), #134).

Of course `std::distance` for iterators of this tree works in linear time (it shifts first iterator until it is equal to the second iterator). You can run my solution for RMQ on big test to ensure that I'm right. It's surprising, but in this place official documentation is wrong.

→ [Ответить](#)



adelnobel

6 лет назад, # ^ | ☆ ▲ 0 ▼

Aha I see...

→ [Ответить](#)

6 лет назад, # ^ | ☆

▲ +3 ▼

Did the authors reply by now? I think it was a design issue because to update the tree size quickly one needs the subtree size augmentation that is only provided by the

`tree_order_statistics_node_update` mixin.



niklasb

We *can* work around the problem by overloading

`std::distance`, but it's not pretty:

https://github.com/niklasb/contest-algos/blob/master/stl_splay_tree.cpp Overall it takes a lot of code to reimplement order statistics, so it's probably not really worth the effort, since a manually coded treap seems to be much faster and doesn't require a lot more code.

→ [Ответить](#)



klip

6 лет назад, # ^ | ☆

▲ 0 ▼

Is it possible to construct order-statistics tree quickly or use something like `emplace_hint`?

→ [Ответить](#)



h8ter

6 лет назад, # | ☆

▲ 0 ▼

В данной [статье](#) написано, что "splay дерево отличная основа для структуры данных горе", но я никак не могу понять как можно реализовать горе используя splay дерево (или другое дерево, содержащее информацию во всех узлах, а не только в листьях). Что именно должно содержаться в узлах splay дерева?

→ [Ответить](#)

TORRES

6 лет назад, # | ☆

▲ 0 ▼

можно ли решать с помощью сия чуда [эту](#) и [эту](#) задачи? Если так подумать, то в роупе можно хранить какие-то структуры данных, например дерево отрезков, или же при изменениях порядка элементов будет полностью нарушаться корректность надстройки над частями контейнера?

→ [Ответить](#)

TORRES

6 лет назад, # ^ | ☆

▲ 0 ▼

да, наверно фигню сказал...

→ [Ответить](#)

Barichek

4 года назад, # | ☆

▲ 0 ▼

Есть ли структура типа горе которая способна разворачивать подотрезок массива за $O(\log N)$?

→ [Ответить](#)

adamant

4 года назад, # ^ | ☆

▲ +14 ▼

Переворот на отрезке можно реализовать с помощью двух горе. В одной храним прямой массив, в другой — развернутый. Перевернуть — то же самое, что поменять соответствующие отрезки местами.

→ [Ответить](#)

Barichek

4 года назад, # ^ | ☆

▲ 0 ▼

Спасибо.

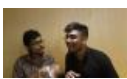
→ [Ответить](#)

SpyCheese

4 года назад, # | ☆

▲ +16 ▼

Правда ли, что эта штука персистентная?

→ [Ответить](#)

mochow

4 года назад, # | ☆

▲ 0 ▼

What if we want to detach a block and set it anywhere else after reversing the block? Can we do this efficiently?

→ [Ответить](#)

Rajan_sust

3 года назад, # | ☆

▲ -20 ▼

@ Perlik, Can I get the pdf link of "Effective STL"-by Meyers.

→ [Ответить](#)

Sukarna_Paul

5 месяцев назад, # ^ | ☆

▲ 0 ▼

<http://www.uml.org.cn/c/%2B%2B/pdf/EffectiveSTL.pdf>

→ [Ответить](#)

Exo

3 года назад, # | ☆

← Rev. 2

▲ -11 ▼

OH WOW

→ [Ответить](#)

3 года назад, # | ☆

▲ -11 ▼

HELLO FROM THE OTHER SIDEEEE

→ [Ответить](#)



Ego

[←](#) [VIBESINIB](#)23 месяца назад, <#> | ☆[←](#) Rev. 3 ▲ 0 ▼

дополню информацию про erase, где аргументом выступает один индекс типа size_t. Решение было найдено после изучения этой функции внутри дерева. А именно:



Draevich

```
// Erase, single character
void
erase(size_t __p)
{ erase(__p, __p + 1); }
```

думаю здесь все понятно: он убирает с позиции __p __p+1 элемент

[→ Ответить](#)

ahmed_drawy

20 месяцев назад, <#> | ☆

▲ 0 ▼

any english link for this article ?

[→ Ответить](#)

[Codeforces](#) (c) Copyright 2010-2020 Михаил Мирзаянов
Соревнования по программированию 2.0
Время на сервере: 15.06.2020 19:52:49^{UTC+5} (f3).
Десктопная версия, переключиться на [мобильную](#).
[Privacy Policy](#)

При поддержке



УНИВЕРСИТЕТ ИТМО