

Алгоритмы

Лекции

Савва Чубий, БПИ233

2024–2025

2024-09-03

Введение	4
Структуры данных	4
Линейные структуры данных	4
Стек и очередь	4
Список	4
Стек с минимумом	4
Очередь через два стека	5
Асимптотика	5
Вектор	5
Асимптотика	5
Метод потенциалов	5
Для стека	6

2024-09-10

Символы Ландау	6
Примеры	6
Мастер-теорема	6
Доказательство	7
Примеры	7
Merge sort	7
Бинарный поиск	7
Обход полного двоичного дерева	7
Обобщение	8

2024-09-17

Информация	8
Алгоритм Карацубы	8
Длинная арифметика	9
Алгоритм Штрассена	9
Аналоги Штрассена	9
Fast Fourier Transform (FFT)	10

2024-09-24

Детерминированные и вероятностные алгоритмы	10
Детерминированные алгоритмы	10
Вероятностные алгоритмы	10
k -ая порядковая статистика (вероятностный)	10
k -ая порядковая статистика (детерминированный)	11
Алгоритм Фрейвалдса	11
Лемма Шварца-Зиппеля	11
Дерандомизация	11
<hr/>	
2024-10-01	
Время работы quicksort-a	12
Skip List	12
Модификации Skip List-a	13
Метод имитации отжига	13
<hr/>	
2024-10-08	
Численное интегрирование	13
Метод Монте-Карло	13
Детерминированный метод	14
Пример	14
Сетки переменной плотности	14
Задача (похожая на 2-ую и 3-ю из конкурса)	15
<hr/>	
2024-10-15	
Декартово дерево ¹	15
Версия 1980 (offline)	15
Версия 1996	15
Split-Merge	16
Split	16
Merge	16
Insert	16
Delete	16
Повороты	16
Insert	16
Delete	16
Split	16
Merge (Join)	17
Теоремы	17
Теорема 1	17
Следствие	17
Лемма	17
Теорема	17
Zip-дерево	17
Теоремы	18
Формулки	18
Теорема	19
Лемма	19
Теорема	19

¹а.к.а ДД, Treap, Дерамиды, Пиво, Курево

Теорема (без доказательства)	19
Сравнение с ДД	19

2024-09-03

Введение

Игорь Борисович

$$\text{Накоп} = 0.25 \cdot \text{Кол} + 0.25 \cdot \text{КР} + 0.4 \cdot \text{ДЗ} + 0.1 \cdot \text{Сем}$$

$$\text{Итог} = \begin{cases} \lfloor \text{Накоп} \rfloor, & \text{if НЕ идти на экзамен} \\ 0.5 \cdot \text{Накоп} + 0.5 \cdot \text{Экз}, & \text{if идти на экзамен} \end{cases}$$

Контесты на 1–2 недели

Структуры данных

Опр. Абстрактный тип данных — определяем, какие операции делает структура, но не определяем конкретную реализацию

Контейнеры:

- Последовательные (напр, вектор)
- Ассоциативные (напр, map)
- Адаптеры (не имеют итераторов)

Линейные структуры данных

Стек и очередь

Стек	Очередь
LIFO	FIFO

Реализации:

- Массив
- Список
- Deque
- (для очереди) на двух стеках

Список

Односвязный:

- `begin()` указывает на первый эл-т
- каждый элемент указывает на следующий
- `end()` указывает в пустоту

Двусвязный:

- каждый элемент указывает ещё и на прошлый эл-т

Список может быть зациклен

Зацикленный список может иметь незацикленное начало

Стек с минимумом

Помимо основного стека поддерживаем стек минимумов (на префиксе)

st	min_st
2	2
5	3
3	3
6	4
4	4

Минимум в стеке – `min_st.top()`

Очередь через два стека

Имеем два стека: `st1` и `st2`

Push:

`st1.push(x)`

Pop:

if `st2` is empty:

переложить весь `st1` в `st2`

`st2.pop()`

Асимптотика

аморт. $O(1)$

Над каждым элементом совершается не более 3 операций:

1. Положить в `st1`
2. Переложить из `st1` в `st2`
3. Вытащить из `st2`

Вектор

1. Изначально выделяется память под несколько эл-в
2. Можем push-ить, пока `v.size() < v.capacity()`
3. Когда место кончается, вектор выделяет в два раза больше памяти и копирует туда элементы
4. При удалении `capacity()` не меняется

Асимптотика

аморт. $O(1)$

На n операций уходит $n + \frac{n}{2} + \frac{n}{4} + \dots + 1 \rightarrow 2n = O(n)$ копирований

Метод потенциалов

Метод подсчета асимптотики

$$\varphi_0 \rightarrow \varphi_1 \rightarrow \dots \rightarrow \varphi_n$$

Опр. Потенциал – функция от наших структур данных

Опр. Аморт. время работы – $a_i = t_i + \Delta\varphi$

$$\sum a_i = \sum (t_i + \Delta\varphi) = \sum t_i + (\varphi_n - \varphi_0)$$

$$\frac{\sum t_i}{n} = \frac{\varphi_0 - \varphi_n}{n} + \frac{\sum a_i}{n} \leq \frac{\varphi_0 - \varphi_n}{n} + \max(a_i)$$

Хотим минимизировать $\max(a_i)$ и $\frac{\varphi_0 - \varphi_n}{n}$

——— Для стека ———

$$\varphi_i := 2n_1$$

push	pop
$t_i = 1$	$t_i = 1$ или $2n_1 + 1$
$a_i = 1 + 2 = 3$	$a_i = 1$ или $2n_1 + 1 + (0 - 2n_1) = 1$

2024-09-10

——— Символы Ландау ———

Опр. $f(x) = O(g(x)) :$

$$\exists C > 0 \exists x_0 \geq 0 : \forall x \geq x_0 : |f(x)| \leq C|g(x)|$$

Опр. $f(x) = o(g(x)) :$

$$\forall \varepsilon > 0 \exists x_0 : \forall x \geq x_0 : |f(x)| \leq \varepsilon|g(x)|$$

Опр. $f(x) = \Theta(g(x)) :$

$$\exists 0 < C_1 \leq C_2 \exists x_0 : C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

——— Примеры ———

1. $3n + 5\sqrt{n} = O(n)$
2. $n = O(n^2)$
3. $n! = O(n^n)$
4. $\log n^2 = O(\log n)$
5. $k \log k = n \Rightarrow k = O(?)$

——— Мастер-теорема ———

$T(n)$ — время работы (количество операций)

Теор. Пусть

$$a \in \mathbb{N}, b \in \mathbb{R}, b > 1, c \geq 0$$

$$T(n) = \begin{cases} O(1) & \text{if } n \leq n_0 \\ aT\left(\frac{n}{b}\right) + O(n^c) & \text{otherwise} \end{cases}$$

тогда:

$$T(n) = \begin{cases} O(n^c) & \text{if } c > \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^{\log_b a}) & \text{if } c < \log_b a \end{cases}$$

— Доказательство —

Мак глубина = $\log_b n$

На i -ом слое: $a^i \cdot \left(\frac{n}{b^i}\right)^c$ операций

В листьях (слой $\log_b n$): $a^{\log_b n}$ операций

$$T(n) = \sum_{i=0}^{\log_b n} O\left(a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(\sum_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i\right)$$

Пусть $q = \frac{a}{b^c}$

При $q < 1 \Leftrightarrow a < b^c \Leftrightarrow \log_b a < c$:

$$O\left(n^c \sum_i q^i\right) \leq O\left(n^c \sum_i^{\infty} q^i\right) = O\left(n^c \cdot \frac{1}{1-q}\right) = O(n^c)$$

При $q = 1$:

$$O(n^c \cdot \log_b n)$$

При $q > 1$:

$$\begin{aligned} O\left(n^c \cdot \left(\frac{a}{b^c}\right)^{\log_b n}\right) &= O\left(n^c \cdot \frac{a^{\log_b n}}{b^{c \cdot \log_b n}}\right) = O\left(n^c \cdot \frac{a^{\log_b n}}{n^c}\right) = \\ &= O(a^{\log_b n}) = O\left(a^{\frac{\log_a n}{\log_a b}}\right) = O\left(n^{\frac{1}{\log_a b}}\right) = O(n^{\log_b a}) \end{aligned}$$

— Примеры —

· MERGE SORT ·

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

$$a = 2$$

$$b = 2$$

$$c = 1$$

$$\log_2 2 = 1 \Rightarrow T(n) = O(n^c \log n) = O(n \log n)$$

· Бинпоиск ·

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

$$a = 1$$

$$b = 2$$

$$c = 0$$

$$\log_2 1 = 0 \Rightarrow T(n) = O(n^c \log n) = O(\log n)$$

· Обход полного двоичного дерева ·

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

$$a = b = 2$$

$$c = 0$$

$$\log_2 2 > 0 \Rightarrow T(n) = O(n^{\log_b a}) = O(n^1) = O(n)$$

—— Обобщение ——

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^c \cdot \log^k n)$$

$$c = \log_b a \Rightarrow T(n) = O(n^c \log^{k+1} n)$$

2024-09-17

—— Информация ——

Коллоквиум предварительно в начале второго модуля (2 ноября, 1–4 пары)

Все задачи в констесте стоят одиночного

—— Алгоритм Карацубы ——

Алгоритм перемножения двух многочленов (или чисел)

$$A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1}$$

$$C(x) = A(x)B(x) = c_0 + c_1x + \dots + c_{n+m-2}x^{n+m-2}$$

bruteforce (в столбик) за $O(n^2)$:

$$c_k = \sum_{i=0}^k a_i \cdot b_{k-i}$$

В Карацубе лучше останавливаться при $\deg \approx 16$ и перемножать в столбик

Добиваем многочлены до одинаковой длины и до степени двойки.

Разобьем многочлен на два:

$$\begin{aligned} A(x) &= a_0 + a_1x + \dots + a_{n-1}x^{n-1} \\ &= \underbrace{\left[a_0 + a_1x + \dots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} \right]}_{A_0(x)} + \underbrace{\left[a_{\frac{n}{2}} + \dots + a_{n-1}x^{\frac{n}{2}-1} \right]}_{A_1(x)} x^{\frac{n}{2}} \\ &= A_0(x) + A_1(x)x^{\frac{n}{2}} \end{aligned}$$

$$B(x) = B_0(x) + B_1(x)x^{\frac{n}{2}}$$

Перемножим (складываем за линию, перемножаем рекурсивно):

$$A(x)B(x) = (A_0 + A_1x^{\frac{n}{2}})(B_0 + B_1x^{\frac{n}{2}}) = A_0B_0 + (A_1B_0 + A_0B_1)x^{\frac{n}{2}} + A_1B_1x^n$$

Найдем асимптотику:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n^2)$$

Так перемножать не выгодно. Проблема в четырех произведениях.

Сокращаем число произведений до трех:

$$(A_0 + A_1)(B_0 + B_1) = \underbrace{A_0B_0 + A_1B_1}_{\text{уже знаем}} + \underbrace{A_0B_1 + A_1B_0}_{\text{сможем найти}}$$

Найдем новую асимптотику:

$$T(n) = \underbrace{3T\left(\frac{n}{2}\right)}_{\text{на умножения}} + \underbrace{O(n)}_{\text{на сложения}} \Rightarrow T(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$$

Так перемножать значительно быстрее.

—— Длинная арифметика ——

$$2105789 = 9 + 8x + 7x^2 + 5x^3 + x^5 + 2x^6 \big|_{x=10}$$

$$a, b < 10^{1000}$$

Нужно делать перенос разряда.

Можно сменить систему счисления для ускорения в константу раз. Удобно брать $x = 10^n$.

—— Алгоритм Штрассена ——

Обобщение Карацубы на матрицы

$$\text{brutforce за } O(n^3): C_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

Размер матрицы: $n = 2^k$

Пилим матрицу на четыре куса. Куски будут перемножаться, как обычные матрицы.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Можно посчитать не за 8, а за 7 умножений

Посчитаем сложность:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7}) \approx O(n^{2.81})$$

Выгодно только для очень больших матриц

· Аналоги Штрассена ·

Год	Название	Асимптотика
1990	Коперсмита-Виноградова	$O(n^{2.3755})$
2020	Алмана-Вильямса	$O(n^{2.3728})$

Гипотеза Штрассена: $\forall \varepsilon > 0 : \exists \text{ алгоритм} : \forall n \geq N : O(n^{2+\varepsilon})$

— FAST FOURIER TRANSFORM (FFT) —

Сложность $O(n \log n)$, но с большой константой

Основной принцип: храним многочлен, как список его значений в некоторых точках. Знаем $A(x_0), A(x_1), \dots, A(x_{n-1})$

Коэффициенты при умножении меняются нетривиально, а значения в точках — намного проще, если удачно выбрать точки: $x_i = \omega^i$, где $\omega \in \mathbb{C}$ или $\omega \in \mathbb{Z}_p$.

Проблема: переход в double.

2024-09-24

— Детерминированные и вероятностные алгоритмы —

— Детерминированные алгоритмы —

Опр. Сложность — максимальное время работы на данных размера n .

Опр. Сложность в среднем — математическое ожидание количества действий.

Для конечномерных:

$$E = \sum_{x \in \chi} P(X) \cdot \text{cut}(x)$$

Для бесконечномерных: *какой-то интеграл*

От бесконечномерного случая часто можно перейти к конечномерному. Например, в случае сортировок делать сжатие координат (превращать)

— Вероятностные алгоритмы —

Опр. Вероятностные алгоритмы — алгоритмы, которые при одних выходных данных могут иметь разное время работы или разный вывод. Используют генератор случайных чисел.

Виды вероятностных алгоритмов:

- **Без ошибки:** всегда выдает правильный ответ
- **С односторонней ошибкой:** ошибается только в одну сторону

Пример: вероятностные алгоритмы проверки на простоту

- **С двусторонней ошибкой:** ошибается в обе стороны

Опр. Ожидаемое время работы — математическое ожидание времени работы (для конкретного набора входных данных)

Опр. Ожидаемая сложность — максимальное ожидаемое время на данных размера n .

— k -ая порядковая статистика (вероятностный) —

Выбрали случайный опорный элемент, разделили массив на две части по опорному:

$$\underbrace{\dots}_{m-1 \text{ элемент}} \leq x_i \leq \underbrace{\dots}_{n-m \text{ элементов}}$$

Медиана будет либо опорным элементом, либо элементов в большем куске.

Оценим ожидаемое время работы. Если массив разбился на куски по $m-1$ и $n-m$

$$T(n) = \underbrace{T(\max(m-1, n-m))}_{\text{в худшем случае ищем в большем куске}} + \underbrace{O(n)}_{\text{на разделение по опорному}}$$

Итого:

$$\begin{aligned} E(T(n)) &= \sum_{m=1}^n P(n) E(T(\max(m-1, n-m))) + O(n) = \sum_{m=\frac{n}{2}}^n \frac{1}{n} \cdot 2E(T(m)) + O(n) = \\ &= \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} E(T(m)) + O(n) = \frac{2}{n} \left(O\left(\frac{n}{2}\right) + \dots + O(n-1) \right) + O(n) = \\ &= \frac{2}{n} O\left(\frac{3n^2}{8}\right) + O(n) = O\left(\frac{3}{4}n\right) + O(n) = O(n) \end{aligned}$$

—— k -ая порядковая статистика (детерминированный) ——

1. Делим массив на чанки размера 5
2. Сортируем каждый чанк: $\frac{7}{5}n$ действий
3. Берем медиану каждого чанка: $m_1, \dots, m_{\frac{n}{10}}$
4. Ищем медиану медиан рекурсивно
5. Используем найденное число в виде опорного элемента в прошлом алгоритме

$$T(n) = \underbrace{T\left(\frac{7n}{10}\right)}_{\text{прошлый алгоритм}} + \underbrace{T\left(\frac{n}{5}\right)}_{\text{рекурсия}} + \underbrace{O(n)}_{\text{разделение}} \rightarrow T(n) = O(n)$$

—— Алгоритм Фрейвалдса ——

Правда ли, что $A \cdot B = C$? (A, B и C даны)

Берем случайный вектор из 0 и 1: $v = (\frac{1}{0}, \frac{1}{0}, \dots, \frac{1}{0})$

Если $AB = C$, то $A \cdot (B \cdot v) = C \cdot v$

Алгоритм с односторонней ошибкой. Если получили равенство, то вероятность неудачи не больше одной второй

Можно повторит процедуру и улучшить вероятность. За k испытаний получаем вероятность $P_{\text{неуд}} \leq \frac{1}{2^k}$, а сложность $O(kn^2)$.

—— Лемма Шварца-Зиппеля ——

$f(x_1, \dots, x_k)$ — многочлен степени n

Считаем, что умеем находить значение f в точке

Хотим проверить, является ли он тождественным нулем

1. Берем случайный набор данных $(y_1, \dots, y_k) \in S^k$
2. Для ненулевого $f : P(f(y_1, \dots, y_n) = 0) \leq \frac{n}{|S|}$

—— Дерандомизация ——

Преобразование вероятностного алгоритма в детерминированный

Для леммы Шварца-Зиппеля и $k = 1$ достаточно проверить $n + 1$ разную точку

2024-10-01

— Время работы QUICKSORT-a —

Случайно выбираем опорный элемент x

Мысленно отсортируем массив:

$$a_1, a_2, \dots, a_n \longrightarrow b_1, b_2, \dots, b_n$$

Опорный элемент x сравнивается со всем и исключается из работы. Значит, два элемента никогда не сравниваются больше одного раза.

Пусть $\delta_{ij} = \text{int}(b_i \text{ сравнивали с } b_j)$. Для времени работы считаем количество сравнений:

$$E(T(n)) = E\left(\sum_{i=0}^{n-1} \sum_{j=i+1}^n \delta_{ij}\right) = \sum \sum E(\delta_{ij}) = \sum \sum P(b_i \text{ сравнивали с } b_j)$$

Два элемента b_i и b_j НЕ будут сравниваться, если между ними когда-то выбирался опорный. Они будут сравниваться только, если среди элементов b_i, \dots, b_j первым был выбран i -ый или j -ый.

$$E(T(n)) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=0}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} = \sum_{i=0}^{n-1} 2 \underbrace{\left(\frac{1}{2} + \frac{1}{3} + \dots + (n-i+1)\right)}_{O(\log n)} = O(n \log n)$$

— SKIP LIST —

Вероятностная структура данных на основе list-a и операциями, как у дерева поиска

Элементы лежат по возрастанию. Есть фиктивные элементы $-\infty$ и $+\infty$ в начале и конце.

$$-\infty \rightarrow -4 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 12 \rightarrow 15 \rightarrow +\infty$$

Хотим прыгать по списку большими шагами, а не только шагами по 1.

Делаем новый список («уровень»), в который включаем элементы данного с вероятностью $P = \frac{1}{2}$. Бесконечности всегда переходят на новый уровень.

В каждом элементе храним указатель направо и вниз. Отдельно храним указатель на $-\infty$ верхнего уровня.

Таблица 4. Структура списка

	↓													
Уровень 2:	$-\infty$	→	-4	→								7	→	$+\infty$
Уровень 1:	$-\infty$	→	-4	→				3	→	7	→		15	→ $+\infty$
Уровень 0:	$-\infty$	→	-4	→	0	→	1	→	3	→	7	→	12	→ 15 → $+\infty$

Операции:

- Поиск: спускаемся по дереву
- Удаление: удаляем из всех слоев
- Добавление: случайно выбираем в каких слоях элемент будет, а в каких — нет (количество слоев может увеличиться), потом добавляем.

Способы реализации:

- multiple nodes: несколько уровней с node-ами
- fat nodes: один уровень, но в каждой node-е несколько указателей

Преимущество перед деревом:

- Легко пишется
- Легко распараллеливается (можно вставлять несколько элементов одновременно)
- Легко печатается

$$P(\text{есть } i\text{-ый уровень}) = 1 - \left(1 - \frac{1}{2^i}\right)^n \underset{\text{неравенство бернулли}}{\leq} 1 - \left(1 - \frac{n}{2^i}\right) = \frac{n}{2^i}$$

$$i = 4 \log_2 n : P(i) \leq \frac{n}{2^{4 \log_2 n}} = \frac{n}{n^4} = \frac{1}{n^3}$$

Модификации SKIP LIST-a

- $p \neq \frac{1}{2}$.
 - количество слоев: $\log_{\frac{1}{p}} n$
 - $O\left(\frac{1}{p} \log_{\frac{1}{p}} n\right)$
 - Лучший вариант $p = \frac{1}{e}$, но на практике, вероятно, бесполезно.
- Можно сделать только два слоя: получим корневую декомпозицию

Метод имитации отжига

Пытаемся минимизировать некоторую величину – функционал качества (например, какой-нибудь путь в графе, расставить на доску ферзей, которые друг друга не бьют).

Будем пытаться улучшить значение функционала: $Q_0 \rightarrow Q_1 \rightarrow \dots$

Плохой способ

Будем случайно генерировать новое состояние и переходить на него только, если оно лучше.

Не работает, так как можно попасть в локальный минимум, а не глобальный.

Вводим понятие температуры T_i , функции, которая как-то убывает с каждой итерацией.

Делаем случайное, небольшое изменение. Если функционал стал меньше, то переходим, иначе переходим с вероятностью:

$$P = e^{-\frac{Q_{i+1} - Q_i}{T_i}}$$

Идея в том, что изначально (когда T_i большое) у нас плохое состояние и не страшно его «потерять», перепрыгнув в другое. Потом (когда T_i маленькое), состояние более хорошее и перепрыгивать мы хотим меньше.

2024-10-08

Численное интегрирование

Дана функция $y = f(x)$. Хотим посчитать $\int_a^b f(x)$.

Метод Монте-Карло

Работает, но сходится медленно

Возьмем $\sin(x)$, $x \in [0, \pi]$.

Будем генерировать точки в прямоугольнике $x \in [0, \pi]$; $y \in [0, \pi]$ и смотреть, сколько попало под график.

—— Детерминированный метод ——

Разбиваем отрезок на равные кусочки:

$$a = x_0 < x_1 < \dots < x_n = b$$

$$x_{i+1} - x_i = h = \frac{b - a}{n}$$

Один кусок ($x \in [x_i, x_{i+1}]$) — криволинейная трапеция. Её можно приближать разными фигурами:

- Прямоугольником:
 - $S = h \cdot f(x_i)$ — метод левых прямоугольников
 - $S = h \cdot f(x_{i+1})$ — метод правых прямоугольников
 - $S = h \cdot f\left(\frac{x_i + x_{i+1}}{2}\right)$ — метод средних прямоугольников
- Трапецией:
 - $S = \frac{h}{2}(f(x_i) + f(x_{i+1}))$
- Криволинейной трапецией (с параболой сверху):
 - $S = \frac{h}{6}(f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1}))$ — формула Симпсона

У формулы Симпсона сходимость на два порядка выше

· Пример ·

Ищем площадь круга:

$$(x - 1)^2 + (y - 5)^2 = 9$$

Разбиваем на две полуокружности: верхнюю ($f_1(x)$) и нижнюю ($f_2(x)$)

$$S = \int_{-2}^4 (f_1(x) - f_2(x)) dx$$

—— Сетки переменной плотности ——

Интегрируем верхнюю половину окружности. Будем использовать метод левых прямоугольников. Если разрезать на четыре части, то два центральных кусочка дают хорошее приближение, а два крайних — очень плохое.

Вывод: сетка с постоянным шагом — часто плохой вариант.

Алгоритм:

- Делим на несколько частей
- Проверяем, на каких кусках получили хорошее приближение, а на каких плохое.

Для этого всё же измельчаем сетку в этом месте и проверяем, сильно ли изменилось приближение. Если слабо, то приближение хорошее:

$$|S_1 - S_2| < \varepsilon$$

- Там, где получили плохое приближение, запускается от него рекурсивно.

Метод хорошо сходится, но мы плохо можем оценивать приближение.

——— Задача (похожая на 2-ую и 3-ю из контеста) ———

Есть набор фигур на экране. Найти площадь объединения

Методы решения (по увеличению эффективности):

1. **Монте-Карло**: плохо сходится

2. **Сетка**:

Разрежем на сеточку (по вертикали и горизонтали). Смотрим на центр: считаем, что если входит центр, то входит весь прямоугольник

3. **Квадродерево**:

- Режем на сетку. Смотрим на каждый квадратик.
- Если квадратик заполнен полностью или полностью пустой, то сразу добавляем его в ответ
- Иначе (если заполнен частично), то продолжаем рекурсивно.
- Останавливаемся, если получили квадратик площади меньше ϵ

4. **Вертикальные полосы переменной плотности + Сканлайн**

- Режем на полосы, которые дают элементарные огибающие. Можно порезать и сильнее.
- Каждая фигура идет от начала до конца полосы (то есть по горизонтали начинается и заканчивается либо на границе полосы, либо вне её).
- Внутри каждой полосы, заменяем каждую фигуру на прямоугольник, дальше сканлайном ищем площадь пересечения.
- Потом делим вертикальную линию пополам, проверяем хорошее ли получилось приближение, если плохое, запускаемся дальше

2024-10-15

——— Декартово дерево² ———

Хотим реализовать set.

Каждая вершина дерева хранит пару (x, y) – (ключ, приоритет).

Приоритет – внутренняя информация для балансировки.

Предполагаем, что x -ы и y -и уникальны.

- По ключам (x) – двоичное дерево поиска
- По приоритетам (y) – куча (с максимумом вверху)

——— Версия 1980 (OFFLINE) ———

В этой версии приоритетов нет

Вставка (как в простом бинарном дереве) в порядке случайной перестановки

——— Версия 1996 ———

Вставка в произвольном порядке, но с приоритетами

²a.k.a ДД, Treap, Дермида, Пиво, Курево

Берем y из равномерного распределения: $y \in U[0, 1]$ — получаем нулевую вероятность совпадения приоритетов

Характеристики:

- Глубина вершины: $E(\text{dep}[v]) = O(\log n)$
- Высота вершины: $E(h[v]) = O(1)$
- Размер поддерева вершины: $E(\text{sz}[v]) = O(\log n)$

Есть два способа реализации:

- с поворотами
- через split-merge

—— SPLIT-MERGE ——

· SPLIT ·

«Режем» дерево вертикальной прямой: `pair<T, T> split(T tree, int line_x).`

Все ключи левого дерева меньше прямой, все ключи правого — больше

· MERGE ·

Есть два дерева такие, что у левого все ключи меньше, чем у правого.

Merge объединяет их в одно дерево: `T split(T left, T right).`

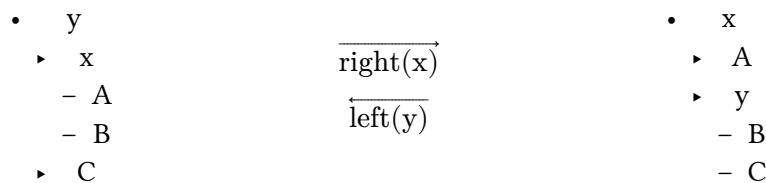
· INSERT ·

```
tl, tr = split(t, x)
merge(tl, merge(T({x, y}), tr))
```

· DELETE ·

- Двумя split-ами отрезаем все элементы меньше x и все элементы больше x .
- merge-им два эти дерева

—— Повороты ——



· INSERT ·

- Вставка до листа
- Подъем поворотами

· DELETE ·

- Поворотами опускаем в лист
- Удаляем лист

· SPLIT ·

Делаем `split(a)`

- `insert({a, +inf})` — новый элемент окажется в корне
- Левое и правое поддерево корня — нужные деревья

· MERGE (JOIN) ·

- Подвесим деревья к {a, +inf}
- Сделаем delete(a)

—— Теоремы ——

· Теорема 1 ·

Пусть ключи отсортированы: $x_1 < \dots < x_n$

Хотим узнать $\text{dep}[x_l] = \sum_{i=1}^n A_{i,l}$ — где $A_{i,l} = \text{int}(x_i - \text{предок } x_l)$

$$\text{sz}[x_l] = \sum_{j=1}^n A_{l,j}$$

Следствие

$$a_{i,l} = P(x_i - \text{предок } x_j)$$

$$E(\text{dep}[x_l]) = \sum_{i=1}^n a_{i,l} = E(\text{sz}[x_l]) = \sum_{i=1}^n a_{l,i}$$

Лемма

Пусть все приоритеты разные (это происходит с вероятностью = 1)

$$x_i - \text{предок } x_j \Leftrightarrow \text{prior}(x_i) = \max_{\min(i,j) \leq k \leq \max(i,j)} \text{prior}(x_k)$$

$$a_{i,j} = \frac{1}{|i-j|+1}$$

Теорема

$$\ln n := \log_e n$$

$$\lg n := \log_2 n$$

$$\begin{aligned} E(\text{dep}[x_l]) &= \frac{1}{|l-1|+1} + \frac{1}{|l-2|+1} + \dots + \frac{1}{1+1} + \frac{1}{1} + \frac{1}{1+1} + \dots + \frac{1}{|n-l|+1} = \\ &= \{H_l - \text{сумма гармонического ряда до } l\} = \\ &= H_l + H_{n-l+1} - 1 < \{\ln(n) < H_n < \ln(n) + 1\} < 1 + 2 \ln n \end{aligned}$$

$$E(\text{sz}[x_l]) < 1 + 2 \ln n$$

$$\frac{h_n}{\ln n} \rightarrow \gamma, n \rightarrow +\infty, \text{ где } \gamma \approx 4.311$$

—— Zip-дерево ——

Цель создания — чтобы ранг (приоритет) занимал меньше бит. В zip-дереве он имеет значение от 1 до $\lg n$ (т.е. столько же, сколько и слоев) и занимает $\lg \lg n$ бит.

Берем не равномерное распределение, а геометрическое. Это логично т.к. количество вершин на соседних уровнях различается примерно в два раза.

$$P(\text{rank} = 0) = \frac{1}{2}, P(\text{rank} = 1) = \frac{1}{4}, \dots, P(\text{rank} = k) = \frac{1}{2^{k+1}}$$

Теперь многие приоритеты совпадают.

Пусть у v есть два сына l и r :

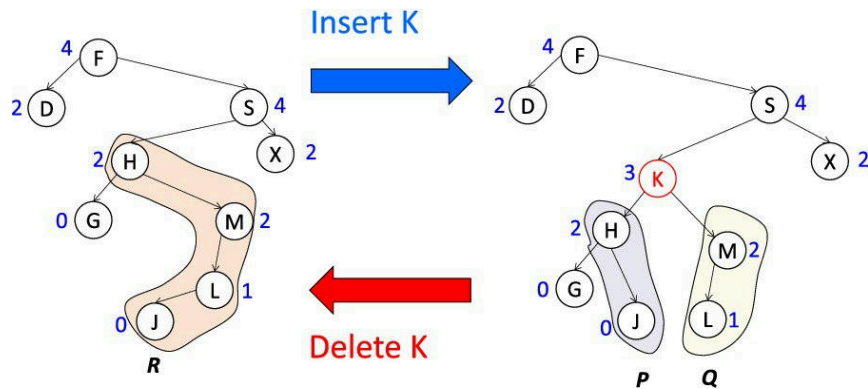
- $\text{rank}(l) < \text{rank}(v)$
- $\text{rank}(r) \geq \text{rank}(v)$

Дерево будет перекошено влево.

$$E(\text{rank}) = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} + \dots \Rightarrow 2E = E + 1 \Rightarrow E = 1$$

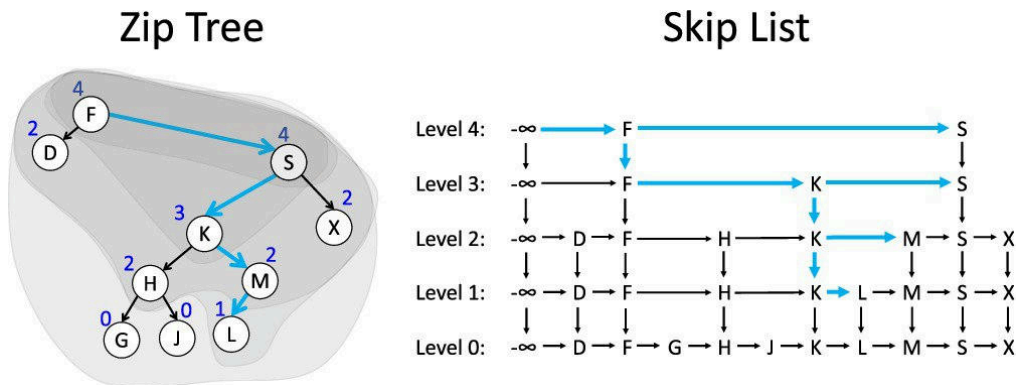
Операции unzip и zip — аналоги split и merge

Рис. 1. Вставка и удаление в zip-дереве



zip-tree имеет естественный изоморфизм со skip-list: Вершина, которая лежит в k -ом уровне, но не лежит в $(k + 1)$ -ом в skip-list-е имеет $\text{rank} = k$.

Рис. 2. Изоморфизм zip-tree и skip-list



zip-tree с данными ключами и приоритетами единственно.

— Теоремы —

· Формулки ·

$$r_i := \text{rank}(i)$$

$$rr := \text{rank}(\text{root})$$

$$P(r_i = k) = \frac{1}{2^{k+1}} \Rightarrow P(r_i < k) = \frac{1}{2} + \dots + \frac{1}{2^k} = 1 - \frac{1}{2^k}$$

$$P(r_i > k) = \frac{1}{2^k} \Rightarrow P\left(\max_i r_i < k\right) = \left(1 - \frac{1}{2^k}\right)^n \underset{\text{по н-ву Бернулли}}{\geq} 1 - \frac{n}{2^k}$$

$$P(\max r_i \geq k) \leq \frac{n}{2^k}$$

$$P(\text{rr} \geq \ln n + C) \leq \frac{n}{2^{\ln n + C}} = \frac{1}{2^C}$$

$$P(\text{rr} \geq (c+1) \ln n) \leq \frac{1}{n^C}$$

· Теорема ·

$$\begin{aligned} E(\text{rr}) &= 0 \cdot P(\text{rr} = 0) + 1 \cdot P(\text{rr} = 1) + \dots = \\ &= 0 \cdot P_0 + 1 \cdot P_1 + \dots + \lceil \lg n \rceil P_{\lceil \lg n \rceil} + (\lceil \lg n \rceil + 1) P_{\lceil \lg n \rceil + 1} + \dots \leq \\ &\leq \lceil \lg n \rceil \cdot \sum_{i=1}^{\infty} P_i + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \dots = \lceil \lg n \rceil + 2 < \lg n + 3 \end{aligned}$$

· Лемма ·

- low — предки x -а с ключом, **меньше** чем у x -а
- high — предки x -а с ключом, **больше** чем у x -а

y_l — самый высокий из low y_h — самый высокий из high

$$E(\# \text{ low}) = 1 + (\text{rank}(y_e) - \text{rank}(x)) \leq 1 + \text{rank}(y_l) \leq \lg n + 4$$

$$E(\# \text{ high}) \leq \frac{1 + \text{rank}(y_h)}{2}$$

· Теорема ·

Из прошлых лемм:

$$E(\text{dep}[v]) = \frac{3}{2} \lg n + O(1)$$

· Теорема (без доказательства) ·

$$E(\text{sz}[v], \text{rank}(v) == k) \leq 3 \cdot 2^k - 1$$

$$E(\text{sz}[v]) \leq \frac{3}{2} \lg n + 2$$

—— Сравнение с ДД ——

	ДД	Zip
глубина	$2 \ln n = 1.3863 \lg n$	$1.5 \ln n$