Алгоритмы и структуры данных 1 модуль.

Андрей Тищенко @AndrewTGk

2024/2025

Контент

Лекция 3 сентября.

Выставление оценок

Накоп = 0.25Коллок + 0.25КР + 0.4ДЗ + 0.1РС.
Коллоквиум между 1 и 2 модулем. После коллока письменная контрольная работа (примерно в конце второго модуля).
ДЗ — контест.
РС — работа на семинаре.
Итог = Накоп или 0.5Накоп + 0.5Экз (экзамен можно не сдавать).
В первом случае накоп округляется, во втором — нет.

1 Структуры данных

Абстракный тип данных — определяется набор операций, но умалчивается реализация.
Структура данных — реализация абстаркного типа данных.

1.1 Линейные структуры данных

Массив

```
int a[20];
array<int, 20> a;
vector<int> a(20); // O(1) amortized
// amortized means average O(1), but O(n) is possible
```

1.2 Список

Виды списков

1. Односвязный (храним указатель на начало и конец, указатель на следующий элемент).
2. Двусвязный (аналогично односвязному, но также указатель на предыдущий).

1.3 Стек

Свойства: LIFO (last in, first out)

Реализация

Массив: простая реализация, так как переполнение невозможно.
Список: возвращаем head, добавление в head (список двусвязный).
deque: добавление и взятие элементов из начала или конца (покрывает функционал).

Виды стеков

1. Стек с минимумом (дополнительный стек, который хранит минимумы на префиксах).

1.4 Очередь

Свойства: FIFO (first in, first out)

Реализация

Массив: кладём элементы по очереди, после переполнения массива мы должны класть элементы в начало (храним указатель на начало и конец очереди).
Список: Возвращаем tail, добавление в head (список двусвязный).
deque: добавление и взятие элементов из начала или конца (покрывает функционал).
Два стека: кладём элементы в первый стек, если нужно взять элемент, то берём из второго стека. Если второй стек пустой, перекладываем все элементы во второй стек. Амортизированное $O(1)$.

1.5 Устройство вектора

Выделяет какое-то базовое количество памяти по умолчанию. Хранится указатель на начало, конец используемой пользователем памяти и конец аллоцированной памяти.
Когда конец используемой пользователем памяти совпадает с концом аллоцированной памяти, аллоцируется кусок памяти в 1.5 или 2 (зависит от реализации) раза больше. Получается, что при выполнении n пушбеков, вектор перезапишет себя не более $\log n$ раз. Всего будет переписано не более $1 + \dots + \frac{n}{2} + n \approx 2n = O(n)$.

2 Метод потенциалов анализа сложности

φ — функция подсчёта потенциала (зависит от параметров структуры данных).
$\varphi_0 \to \varphi_1 \to \varphi_2 \to \dots \to \varphi_n$
Определение: амортизированное время работы:

$$a_i = t_i + \Delta\varphi,\ \Delta\varphi = \varphi_{i+1} - \varphi_i$$

$\sum a_i = \sum t_i + (\varphi_n - \varphi_0) \Rightarrow \frac{\sum t_i}{n} = \frac{\varphi_0 - \varphi_n}{n} + \frac{\sum a_i}{n} \leqslant \frac{\varphi_0 - \varphi_n}{n} + \max(a_i)$
$\max(a_i)$, $\frac{\varphi_0 - \varphi_n}{n}$ хотим минимизировать, в наших силах выбирать функцию потенциала.
$\varphi_i = 2n_1$
push: $t_i = 1,\ a_i = 1 + 2 = 3$
pop: $t_i = 1$ или $t_i = 2n_1 + 1,\ a_i = 1$ или $a_i = 2n_1 + 1 + (0 - 2n_1) = 1$
Значит $\max(a_i) \leqslant 3$, при этом $\frac{\varphi_0 - \varphi_n}{n} \leqslant 0$, то есть амортизированное время работы:

$$\frac{\sum t_i}{n} \leqslant 3$$

Лекция 10 сентября.

3 Символы Ландау

Оценка сверху:

$$f(x) = O\big(g(x)\big) \Leftrightarrow \exists C > 0\ \exists x_0 \geqslant 0\ \forall x \geqslant x_0 :\ |f(x)| \leqslant C|g(x)|$$

Оценка снизу:

$$f(x) = o\big(g(x)\big) \Leftrightarrow \forall \varepsilon > 0\ \exists x_0\ \forall x \geqslant x_0 :\ |f(x)| \leqslant \varepsilon|g(x)|$$

Равенство функций:

$$f(x) = \Theta\big(g(x)\big) \Leftrightarrow \exists 0 < C_1 < C_2,\ \exists x_0\ \forall x \geqslant x_0 :\ C_1|g(x)| \leqslant |f(x)| \leqslant C_2|g(x)|$$

Примеры:

1. $3n + 5\sqrt{n} = O(n)$

2. $n = O(n^2)$. Оценка грубая, но правильная, потому что $n \leqslant n^2$. Лучше было бы понять, что $n = O(n)$

3. $n! = O(n^n)$

4. $\log n^2 = O(\log n)$

5. Пусть мы в задаче ввели параметр k, при этом оптимально, чтобы выполнялось соотношение $k \log k = n$. Как можно оценить k?
$k = O(?)$, обсудим на семинаре.

Задача

Найти асимптотику сортировки слиянием.

Пусть $T(n)$ — время, используемое для сортировки массива длины n.
Зная принцип работы этой сортировки можно сказать, что

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

Сформулируем **Мастер теорему**

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + O(n^c),\ a \in \mathbb{N},\ b \in \mathbb{R},\ b > 1,\ c \in \mathbb{R},\ c \geqslant 0 \\ O(1),\ n \leqslant n_0 \end{cases}$$

Разберём три случая:

1. $c > \log_b a:\ T(n) = O(n^c)$

2. $c = \log_b a:\ T(n) = O(n^c \log n)$

3. $c < \log_b a:\ T(n) = O(n^{\log_b a})$

На i-ом слое: $a^i \left(\frac{n}{b^i}\right)^c$
Листья (слой $\log_b n$): $a^{\log_b n}$ задач, сложность каждой равна 1
$T(n) \leqslant \sum\limits_{i=0}^{\log_b n} O\left(a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(\sum\limits_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(n^c \sum\limits_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i\right)$
Положим $q = \frac{a}{b^c}$, тогда:
$q < 1 \Leftrightarrow a < b^c \Leftrightarrow c > \log_b a,\ O\left(n^c \sum\limits_{i=0}^{\log_n b} q^i\right) = O\left(n^c \frac{1}{1-q}\right) = O\left(n^c\right)$
$q = 1 \Leftrightarrow O(n^c \log_b n)$
$q > 1$:
Докажем *лемму*:

$$\forall q > 1:\ 1 + q + \dots + q^n = O(q^n)$$

$\frac{q^{n+1}-1}{q-1} < \frac{q^{n+1}}{q-1} = \frac{q}{q-1} q^n = O(q^n)$
Тогда для $q > 1$: $O\left(n^c \left(\frac{a}{b^c}\right)^{\log_b n}\right) = O\left(n^c \frac{a^{\log_b n}}{b^{c \log_b n}}\right) = O\left(n^c \frac{a^{\log_b n}}{n^c}\right) =$
$= O\left(a^{\log_b n}\right) = O\left(n^{\log_b a}\right)$

Примеры

Сортировка слиянием:
$T(n) = 2T(\frac{n}{2}) + O(n^1) \Rightarrow$
$\Rightarrow a = 2,\ b = 2,\ c = 1 \Rightarrow \log_b a = c \land T(n) = O(n^c \log n) = O(n \log n)$
Бинарный поиск:
$T(n) = T(\frac{n}{2}) + O(1) \Rightarrow$
$\Rightarrow a = 1,\ b = 2,\ c = 0 \Rightarrow \log_b a = c \Rightarrow T(n) = O(n^c \log n) = O(\log n)$
Обход полного двоичного дерева с n вершинами:
$T(n) = 2T(\frac{n}{2}) + O(1) \Rightarrow$
$\Rightarrow a = b = 2,\ c = 0 \Rightarrow \log_2 2 > 0 \Rightarrow T(n) = O\left(n^{\log_b a}\right) = O(n^1) = O(n)$

4 Алгоритмы быстрого умножения

4.1 Алгортим Карацуба

Придумали в 1960 году. Этот алгоритм мотивировал людей искать более быстрые способы решения известных задач. (На коллоквиуме может пригодиться базовое знание алгоритма Фурье).

brute[1]

$A(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$. Будем называть это многочленом с n коэффициентами ((n)-член в дальнейшем).
$B(x) = b_0 + b_1x + \cdots + b_{m-1}b^m$ - (m)-член
$C(x) = A(x) \cdot B(x) = c_0 + c_1x + \cdots + c_{n+m-2}x^{n+m-2}$ - $(n+m-1)$-член
$c_k = \sum\limits_{i=0}^{k} a_i \cdot b_{k-i}$ - k-ый коэффициент в $C(x)$
Такое решение имеет асимптотику $O(n \cdot m)$. Мы будем писать алгоритм для перемножения многочленов одинаковых степеней, поэтому асимптотика будет $O\big(\max(n,\ m)^2\big)$, где $\max(n,\ m)$ - степень двойки.
$A(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} =$
$= \big(\underbrace{a_0 + a_1 + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}}_{A_0(x)}\big) + \big(\underbrace{a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x^1 + \cdots + a_{n-1}x^{\frac{n}{2}-1}}_{A_1(x)}\big)x^{\frac{n}{2}}$
Аналогично разбиваем $B(x) = B_0(x) + B_1(x)x^{\frac{n}{2}}$, тогда:
$A(x) \cdot B(x) = (A_0 + A_1x^{\frac{n}{2}})(B_0 + B_1x^{\frac{n}{2}}) = A_0B_0 + (A_1B_0 + A_0B_1)x^{\frac{n}{2}} + A_1B_1x^n$
Однако если это тупо перемножить, то мы ничего не выиграем:
$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$. По мастер теореме $a = 4,\ b = 2,\ c = 1 \Rightarrow$
$\Rightarrow 1 < \log_2 4 = 2 \Rightarrow T(n) = O(n^2)$
Методом подстановки получаем такую же асимптотику (опускаем O(n), так как здесь оно не сильно влияет).
$T(n) = 4T(\frac{n}{2}) = 4^2T(\frac{n}{2^2}) = \cdots = 4^kT(\frac{n}{2^k})$.
В какой-то момент $2^k = n \Rightarrow 4^k = n^2 \Rightarrow T(n) = n^2T(1) = n^2$
Идея Карацубы заключается в выполнении трёх умножений вместо четырёх.
Сделаем два умножения: $A_0B_0,\ A_1B_1$, далее алгебраические фокусы:
Делаем умножение $(A_0 + A_1)(B_0 + B_1) = A_0B_0 + A_1B_1 + A_0B_1 + A_1B_0$, тогда: $(A_1B_0 + A_0B_1) = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1$.
Этот метод работает быстрее, так как сложение и вычитание мы выполняем за линию, то есть за 2 сложения и два вычитания (4 линии) экономим одно умножение (квадрат).
Теперь $T(n) = 3T\left(\frac{n}{2}\right) + O(n) \overset{\text{M Th}}{\Longrightarrow} 1 < \log_2 3 \Rightarrow T(n) = O(n^{\log_2 3})$
Проверим методом подстановки: $n = 2^k:\ O(3^k) = O(3^{\log_2 n}) = O(n^{\log_2 3})$

4.2 Алгоритм Штрассена

Придуман в 1969 для умножения матриц.
Математически: $C = \underset{n\times n}{A} \cdot \underset{n\times n}{B}$, асимпотика $O(n^3)$
$C_{i\,j} = \sum\limits_{k=0}^{n-1} a_{i\,k} \cdot b_{k\,j}$.
Алгоритм заключается в следующем преобразовании:

$$\begin{pmatrix} a_{1\,1} & \vdots & a_{1\,2} \\ \cdots & . & \cdots \\ a_{2\,1} & \vdots & a_{2\,2} \end{pmatrix} \cdot \begin{pmatrix} b_{1\,1} & \vdots & b_{1\,2} \\ \cdots & . & \cdots \\ b_{2\,1} & \vdots & b_{2\,2} \end{pmatrix} = \begin{pmatrix} a_{1\,1}b_{1\,1} + a_{1\,2}b_{2\,1} & \vdots & a_{1\,1}b_{1\,2} + a_{1\,2}b_{2\,2} \\ \cdots & . & \cdots \\ a_{2\,1}b_{1\,1} + a_{2\,2}b_{2\,1} & \vdots & a_{2\,1}b_{1\,2} + a_{2\,2}b_{2\,2} \end{pmatrix}$$

$d = (a_{1\,1} + a_{2\,2})(b_{1\,1} + b_{2\,2})$
$d_1 = (a_{1\,2} - a_{2\,2})(b_{2\,1} + b_{2\,2})$

[1] Далее так будут называться решения в лоб или переборные решения
[0] M Th = Мастер Теорема

$d_2 = (a_{21} - a_{11})(b_{11} + b_{12})$
$h_1 = (a_{11} + a_{12})b_{22}$
$h_2 = (a_{21} + a_{22})b_{11}$
$v_1 = a_{22}(b_{21} - b_{11})$
$v_2 = a_{11}(b_{12} - b_{22})$
$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7 \approx 2,81})$

Алгоритм Копперсмита-Виноградова (1990)

Работает за $O(n^{2,3755})$. Является улучшением алгоритма Штрассена.

Алгоритм Алмана Вильямса (2020)

За $O(n^{2,3728})$.

Гипотеза Штрассена

$\forall \varepsilon > 0$ $\exists$ алгоритм $\exists N\ \forall n \geqslant N:\ O(n^{2+\varepsilon})$

4.3 Быстрые преобразования Фурье

$O(n \log n)$. Переменожаем n-члены, где n - степень двойки.
Храним многочлен в виде n его специально подобранных точек (по ним можно восстановить коэффициенты).
На коллоквиуме понадобится знать основную идею работу и асимптотику.

5 Вероятностные алгоритмы

Для детерминированных алгоритмов есть понятия:
Сложность — количество операций на данных размера n.
Сложность в среднем — математическое ожидание количества действий. Вероятностный алгоритм — использует генератор случайных чисел. Могут быть алгоритмы:

- Работающие без ошибок
- С односторонней ошибкой
- С двусторонней ошибкой

Ожидаемое время — математическое ожидание времени работы (на одном наборе данных)
Ожидаемая сложность — максимальное время работы на данных размера n.

Поиск k-ой порядковой статистики

Выбираем опорный элемент i. Пусть меньше него в массиве $m-1$ элемент, а больше него $n-m$ элементов.
Тогда $E\big(T(n)\big) = \sum\limits_{m=1}^{n} P(m) E\Big(T\big(\max(m-1,\ n-m)\big)\Big) + O(n) =$
$= \sum\limits_{m=\frac{n}{2}}^{n} \dfrac{1}{n} 2T(m) + O(n) = \dfrac{2}{n} \sum\limits_{m=\frac{n}{2}}^{n-1} E\big(T(n)\big) + O(n) =$
$= \dfrac{2}{n}\big(O(\dfrac{n}{2}) + O(n-1)\big) + O(n) = \dfrac{2}{n} O(\dfrac{3n^2}{8}) + O(n) = O(\dfrac{3}{4}n) + O(n) = O(n)$
Получаем $E\big(T(n)\big) = O(n)$ по индукции.

Детерминированный поиск медианы.

Делим массив на группы по 5 чисел, сортируем их за 7 сравнений (или за 10 действий пузырьком), тратим на это $7\frac{n}{5}$ действий.
Получаем $\frac{n}{5}$ медиан: $m_1, \ldots,\ m_{\frac{n}{5}}$ рекуррентно ищем медиану в них, пусть это m_s. Тогда:
$\frac{n}{10}$медиан $\leqslant m_s \leqslant \frac{n}{10}$медиан. Для каждой медианы слева справедливо, что есть два числа, меньше неё. Справа — есть два числа больше неё. Так как это были медианы в пятёрках чисел.
$T(n) = T\left(\frac{7n}{10}\right) + T(\frac{n}{5}) + O(n) \leqslant T\left(\frac{7n}{10}\right) + T(\frac{n}{5}) + C(n) \Rightarrow$
$\Rightarrow T(n) \leqslant 10C \cdot n \Rightarrow T(n) = O(n)$

Алгоритм Фреймвальдса

Умножаем $A \cdot B = C, \quad n \times n$
$v = \begin{pmatrix} 0 & 0 & & 0 \\ 1 ,& 1 ,& \cdots & 1 \end{pmatrix}$ случайные числа 0 или 1
$A \cdot B \cdot v = C \cdot v \Rightarrow O(n^2)$
Если получили равенство, то вероятность $A \cdot B \neq C$: $P_{\text{неудачи}} \leqslant \frac{1}{2}$

Доказательство

Положим $D = AB - C = \begin{pmatrix} d_{1\,1} & \ldots & d_{1\,n} \\ \ldots & \ldots & \ldots \\ d_{n\,1} & \ldots & d_{n\,n} \end{pmatrix}$, без ограничения общности в нём $d_{1\,1} \neq 0$
Тогда при домножении матрицы D на вектор v. Вероятность наличия на нужной позиции в v нуля есть $\frac{1}{2}$ (так как вектор состоит из 0 и 1). $Dv = \begin{pmatrix} d_{1\,1}v_1 + (d_{1\,2}v_2 + \cdots + d_{1\,n}v_n) \\ \ldots \\ \ldots \end{pmatrix}$

$\left(\frac{m}{2^n}\right)^k \leqslant \left(\frac{1}{2}\right)^k$

Лемма Шварца-Зиппеля

$f(x_1, \ldots,\ x_k)$ — многочлен степени n от k переменных.
Возьмём точку $Y = (y_1, \ldots,\ y_k)$ равновероятно из множества S (множество значений).
$P\big(f(y_1, \ldots,\ y_k)\big) \leqslant \frac{n}{|S|}$
Дерандомизация: положим $k = 1$ и смотрим $n + 1$ точку.

Лекция 1 октября.

Быстрая сортировка

Пусть изначальный массив $a = \{a_1, \ldots,\ a_b\}$
Хотим произвести операцию $a \rightsquigarrow b = \{b_1, \ldots,\ b_n\}$, где $b_1 \leqslant \cdots \leqslant b_n$.
$E\big(T(n)\big) = E\left(\sum\limits_{i=0}^{n-1}\sum\limits_{j=i+1}^{n}\delta_{i\,j}\right) = \sum\limits_{i=0}^{n-1}\sum\limits_{j=i+1}^{n}E(\delta_{i\,j}) = \sum\limits_{i=0}^{n-1}\sum\limits_{j=i+1}^{n}P(b_i \text{ сравнивается с } b_j)$
$\delta_{i\,j} = \mathrm{bool}(b_i \text{ сравним с } b_j)$
Рассмотрим массив b, посмотрим на b_i и b_j:

$$b_1 \leqslant b_2 \leqslant \cdots \leqslant \overbrace{b_i \leqslant \cdots \leqslant b_j}^{j-i+1 \text{ элемент}} \leqslant \cdots \leqslant b_n$$

Нас интересует выбор элемента из этих $j - i + 1$. Если выберем что-то между, то интересующие нас два сравниваться не будут. Если выберем один из них, то придётся сравнивать.
Вероятность выбора b_i или b_j из $j - i + 1$ равна $\frac{2}{j-i+1}$
Тогда $E\big(T(n)\big) = \sum\limits_{i=0}^{n-1}\sum\limits_{j=i-1}^{n}\dfrac{2}{j-i+1} = \sum\limits_{i=0}^{n-1}\sum\limits_{k=1}^{n-i}\dfrac{2}{k+1} = \sum\limits_{i=0}^{n-1}O(\log n) = O(n \log n)$

SkipList

Является стурктурой данных, придумал William Pugh в 1989 году.
Сначала представлял собой отсортированный список, первым элементом которого является $-\infty$, имеющий большое количество указателей в вершинах (у каждого второго был указатель для прыжка на 2, у каждого 4 — на 4 и т.д.), но возникали проблемы с удалением.
Появилась новая идея. Создать список "второго уровня", в который с вероятностью $\frac{1}{2}$ попадают элементы изначального списка ($-\infty$ и элемент сразу после неё попадает во все уровни).
У каждого элемента получившегося списка есть указатель на следующий элемент и на его копию на более низком уровне.
По получившемуся списку мы проходим в два раза быстрее, поэтому мы создаём аналогичный список, базируясь на втором уровне, продолжая это до тех пор, пока размер получившегося уровня не станет меньше либо равен 2.
В среднем на каждом уровне количество элементов уменьшается вдвое.

Поиск

В такой структуре данных поиск можно осуществлять бинарным поиском.

Удаление

В случае удаления элемента, он должен удаляться во всех уровнях. Пусть мы удаляем элемент x, тогда при рекурентном бинарном поиске этого элемента мы получим указатели $l < x \leqslant r$ для каждого уровня. Если r совпадает с x, то r удаляется как в обычно списке, иначе на этом и на всех уровнях выше x уже не будет, удаление завершено. Перебалансировку мы не производим, так как вероятность существенно уменьшить количество уровней крайне мала.

Вставка

Ищем позицию для вставки, после чего вставляем его на более высокий уровень с вероятностью $\frac{1}{2}$. В результате этой операции количество уровней могло увеличиться (в теории оно могло увеличиться более чем на 1, но на практике более чем на 1 уровень увеличивать смысла нет).

Оценка количества уровней

$P(\text{есть i-й уровень}) = 1 - \left(1 - \frac{1}{2^i}\right)^n \leqslant 1 - (1 - \frac{n}{2^i}) = \frac{n}{2^i}$
$i = 4\log_2 n \quad P(i) \leqslant \frac{n}{4\log_2 n} = \frac{n}{n^4} = \frac{1}{n^3}$, то есть вероятность сильно превзойти $\log_2 n$ очень мала.

Оценка оптимальной вероятности повышения

Пусть $p \neq \frac{1}{2}$, тогда $n,\ pn,\ p^2 n, \ldots,\ 1$ — размеры слоёв.
$\log_{\frac{1}{p}} n$ — количество слоёв.
Асимптотика: $O\left(\frac{1}{p}\log_{\frac{1}{p}} n\right)$

Алгоритм имитации отжига

Введём функционал качества $Q_0 \rightsquigarrow Q_1 \rightsquigarrow Q_2 \rightsquigarrow \ldots$.
Это некая функция, которую стоит минимизировать. Программа производит случайные изменения, после чего смотрит на изменение функционала.
$Q_{i+1} < Q_i \Rightarrow$ делаем изменение. $Q_{i+1} > Q_i \Rightarrow$ делаем изменение с вероятностью $p = e^{-\frac{Q_{i+1} - Q_i}{T_i}}$
T_i - некая убывающая функция (гипербола, линейная, что лучше подходит).

Лекция 8 октября.

6 Численное интегрирование

Подсчёт площади функции $f(x)$ на отрезке $[a,\ b]$.

Метод Монте-Карло

Составляем описанный прямоугольник для этой функции. Случайным образом помещаем в него точки. Далее составяем пропорцию: площадь прямоугольника отностися к площади фигуры так же как количество помещённых точек относится к количеству точек, попавших в нашу фигуру.

Метод прямоугольников

Поделим отрезок $[a,\ b]$ на n равных частей, далее скадываем площади S получившихся трапеций. $a = x_0 < x_1 < \dots < x_n = b,\ h = \frac{b-a}{n}$,

$$\begin{cases} S = h \cdot f(x_i) \text{ — метод левых прямоугольников} \\ S = h \cdot f(x_{i+1}) \text{ — метод правых прямоугольников} \\ S = h \cdot f\left(\frac{x_i + x_{i+1}}{2}\right) \text{ — метод центральных прямоугольников} \end{cases}$$

Метод трапеций

Разбиваем аналогично, но площадь считаем по формуле
$S = \frac{h}{s}\big(f(x_i) + f(x_{i+1})\big)$

Формула Симпсона

Разбиваем на равные отрезки, для каждого отрезка считаем:
$S = \frac{h}{6}\Big(f(x_i) + 4f\left(\frac{x_i+x_{i+1}}{2}\right) + f(x_{i+1})\Big)$

Подсчёт площади круга

Пусть окружность задаётся уравнением $(x-1)^2 + (y-5)^2 = 9$
Разобьём окружность на полуокружности: $y = 5 \pm \sqrt{9-(x-1)^2}$
Пусть $f_1(x) = 5 + \sqrt{9-(x-1)^2},\ f_2(x) = 5 - \sqrt{9-(x-1)^2}$
Тогда искомая площадь: $\displaystyle\int_{-2}^{4} f_1(x) - f_2(x)\, dx$
Для подсчёта этих интегралов стоит написать рекурретную функцию. Она должна на отрезках с недостаточной точностью добавлять новые точки, пока точность нас не устроит. (так называемая переменная сетка).

Сетка переменной плотности

Рассматриваем полученную на отрезке площадь, обозначим её S_1. Всё-таки поделим пополам и посчитаем полученную площадь, обозначим её S_2. Выбираем необходимую точность ε, если выполняется
$|S_1 - S_2| < \varepsilon$, то точку мы оставляем.

Задача 2 идея

Даётся экран, на нём расположено n фигур. Нужно найти площадь объединения этих фигур.

Возможные решения

0. Монте-Карло

1. Прямоугольники. Если центр прямоугольника лежит в фигуре, считаем, что весь прямоугольник лежит.

2. Квадродерево. Изначально разбиваем на большие прямоугольники. Каждый большой по необходимости разбиваем на маленькие (если квадрат частично лежит в какой-то фигуре). Тогда маленькие квадраты будут появляться на границах фигур, общая площадь граничных фигур будет $S \approx \sqrt{\varepsilon} \cdot Length$, где ε — площадь одного квадратика, $Length$ — длина всех фигур.

3. Вертикальный полосы переменной плотности + сканирующая прямая. Нарезаем вертикально, считаем площадь прямоугольников, полученных методом центральных прямогугольников. Делим отрезок пополам, считаем площидь но-новому, если разность площадей нас устраивает, прекращаем деление.

Лекция 15 октября

7 Декартово дерево

Декартово дерево хранит пары {ключ, приоритет}, в дальнейшем будет называться Vertex.
Можно визуализировать в декартовой системе координат, сопоставив каждой вершине {key, priority} точку {x, y}. При таком отображении увеличение ключа будет смещать вершину вправо, а увеличение приоритета будет поднимать вершину наверх.
Поиск по ключу — аналогичен поиску в двоичном дереве поиска (Binary Search Tree).
Поиск по приоритетам — поиск по куче (Heap).

7.1 История декартовых деревьев

7.1.1 Vuillemin (1980) Cartesian Tree

Вставка до листа в порядке случайной перестановки. Использует повороты.

Операции в структуре данных:

- Insert(Vertex) — вставляем как лист, затем поднимаем поворотами.
- Delete(Vertex) — спускаем нужную вершину поворотами, удаление листа.
- Split(key a) — выполняется Insert$\big($Vertex(a, $+\infty$)$\big)$, затем удаляется корень.
- Join(tree, tree, key a) — подвешивает деревья к Vertex(a, $+\infty$), потом эта вершина удаляется.

7.1.2 Seidel, Aragon (1996) treap (tree + heap)

Вставка в произвольном порядке (декартово дерево однозначно задаётся своими вершинами). Используются функции Split и Merge.
$(x,\, y),\ y \in R[0,\, 1]$ (равномерное распределение)
Можем посчитать математическое ожидание некоторых величин дерева:
$\mathrm{E}(depth[v]) = \mathrm{O}(\log n)$
$\mathrm{E}(height[v]) = \mathrm{O}(1)$
$\mathrm{E}(size[v]) = \mathrm{O}(\log n)$

- Insert(Vertex v) — выполняем Split(Vertex v), два дерева tl (ключи меньше v.key), tr(ключи больше v.key), выполняем Merge(tl, tr, v)
- Delete(Vertex v) — Split(Vertex v), получаем tl, tr, после чего выполянем Merge(tl, tr), не включая v в новое дерево.

7.2 Теорема 1

$$depth[x_l] = \sum_{i=1}^{n} a_{i,l},\ \text{где } a_{i,l} = \begin{cases} 1, & \text{если } x_i - \text{предок } x_l \\ 0, & \text{иначе} \end{cases} \qquad \text{(количество предков)}$$

$$size[x_l] = \sum_{j=1}^{n} a_{l,j},\ \text{где } a_{l\,j} = \begin{cases} 1, & \text{если } x_l - \text{предок } x_j \\ 0, & \text{иначе} \end{cases} \qquad \text{(количество детей)}$$

7.3 Следствие

$$\mathrm{E}(depth[x_l]) = \sum_{i=1}^{n} p_{i,l}, \quad \mathrm{E}(size[x_l]) = \sum_{j=1}^{n} p_{l,j}, \quad p_{i,j} = \mathrm{P}(a_{i,j} = 1),\ (\text{P это вероятность})$$

7.3.1 Лемма

Пусть T — декартово дерево (множество вершин), тогда:

$$\forall i,\ j \in \underline{|T|} \quad (v_i.prior = v_j.prior \Rightarrow i = j) \wedge a_{i,j} = 1 \Rightarrow v_i.prior = \max_{\min(i,\ j) \leqslant k \leqslant \max(i,j)} (v_k.prior)$$

То есть при отсутствии вершин с равными приоритетами, приоритет вершины с номером i, являющейся предком для вершины j, будет максимальным среди приоритетов всех вершин на отрезке $\begin{cases} [i,\ j],\ i \leqslant j \\ [j, i],\ j \leqslant i \end{cases}$

7.3.2 Следствие

$$p_{i,j} = \frac{1}{\big(|i-j|+1\big)}$$

Вероятность того, что одна вершина (с номером i) является максимальной среди всех вершин на отрезке с концами i и j.

7.3.3 Теорема 2

$$\mathrm{E}(depth[x_l]) = \frac{1}{|l-1|+1} + \frac{1}{|l-2|+1} + \dots + \frac{1}{1+1} + \frac{1}{1} + \frac{1}{1+1} + \dots + \frac{1}{|l-n|+1} =$$
$$= \mathrm{H}_l + \mathrm{H}_{n-l+1} - 1 < 1 + 2\ln n$$

За H_n обозначена сумма первых n членов гармонического ряда. Первое слагаемое — уменьшение знаменателя от l до 1, второе — увеличение знаменателя от 1 до $l-n+1$, единицу учли и там, и там, поэтому надо вычесть (третье слагаемое). Аналогично можем посчитать математическое ожидание $size[x_l]$:

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

Однако распределение глубины плотное, а распределение размера — всегда неравномерное.
Интересный факт из математики (Devroye): $\dfrac{H_n}{\ln n} \xrightarrow[n \to +\infty]{} \gamma \approx 4.311$

8 Zip tree

Создатели: Tarjan, Levy, Timmel (2017)
Задумка — улучшение декартова дерева путём уменьшения количества памяти, хранимого для приоритета. Теперь приоритеты лежат на отрезке $[1,\ \log_2 n]$ (вместо $[1,\ n^3]$), то есть на хранение можно выделять $\log_2 \log_2 n$ бит.
Аналог ДД, но вместо $R[0,\ 1]$ будем пользоваться геометрическим распределением приоритетов (в дальнейшем будем называть рангом):

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

Операции со структурой данных:

- Upzip = Split
- Zip = Merge
- Insert
- Delete

8.1 Утверждение

Zip Tree имеет единственный изоморфизм со Skip List.

8.2 Лемма

Zip Tree единственно (однозначно задаётся множеством вершин).

Свойства

$$P(v_i.rank = k) = \frac{1}{2^{k+1}} \Rightarrow P(v_i.rank < k) = \sum_{i=1}^{k} \frac{1}{2^i} = 1 - \frac{1}{2^k}$$

$$P(v_i.rank > k) = 1 - P(v_i.rank \leqslant k) = 1 - \left(\frac{1}{2^{k+1}} + 1 - \frac{1}{2^k}\right) = \frac{1}{2^{k+1}} \Rightarrow$$

$$\Rightarrow P(\max_i(v_i.rank) < k) = \big(P(v_i.rank < k)\big)^n = \left(1 - \frac{1}{2^k}\right)^n \geqslant 1 - \frac{n}{2^k} \Rightarrow$$

$$\Rightarrow P(\max_i(v_i.rank) \geqslant k) \leqslant \frac{n}{2^k}$$

$$P(v_{root}.rank \geqslant \log_2 n + C) \leqslant \frac{n}{2^{\log_2 n + C}} = \frac{1}{2^C}$$

$$P(v_{root}.rank \geqslant (C+1)\log_2 n) \leqslant \frac{n}{2^{(C+1)\log_2 n}} = \frac{n}{n^{C+1}} = \frac{1}{n^C}$$

8.3 Теорема 1

$$\mathrm{E}(v_{root}.rank) = 0 \cdot \mathrm{P}(v_{root}.rank = 0) + 1 \cdot \mathrm{P}(v_{root}.rank = 1) + \dots =$$

$$= 0 \cdot P_0 + 1 \cdot P_1 + \dots + \lceil \log_2 n \rceil P_{\lceil \log_2 n \rceil} + (\lceil \log_2 n \rceil + 1) P_{\lceil \log_2 n \rceil + 1} + \dots \leqslant$$

$$\leqslant \lceil \log_2 n \rceil \cdot \sum_{i=1}^{\infty} P_i + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + \dots =$$

$$= \lceil \log_2 n \rceil + 2 < \log_2 n + 3$$

8.4 Лемма 1

low предок — вершина, имеющая больший ранг, но меньший ключ (находятся слева сверху от вершины).
Для вершины x и её low предка y_l с наибольшим рангом справедливо:

$$\mathrm{E}(\#\text{low предков}) = 1 + (y_l.rank - x.rank) \leqslant 1 + y_l.rank$$

8.5 Лемма 2

$$\mathrm{E}(\#\text{low предков}) \leqslant \frac{1 + y_h.rank}{2}$$

8.6 Теорема 2

$$\mathrm{E}(depth[v]) = \frac{3}{2} \cdot \log_2 n + \mathrm{O}(1)$$

8.7 Теорема 3

$$\mathrm{E}(size[v], v.rank == k) \leqslant 3 \cdot 2^k - 1$$

$$\mathrm{E}(size[v]) \leqslant \frac{3}{2} \cdot \log_2 n + 2$$

Доказательства теорем не были разобраны на лекции (рекомендовали ознакомиться с фрагментом оригиналь
статьи про Zip Tree).

Сравнение глубины ДД и Zip дерева

ДД - $z \log_2 n \approx 1,38 \log_2 n$
Zip дерево - $1,5 \log_2 n$