Алгоритмы

Харитонцев-Беглов Сергей

17 сентября 2021 г.

Содержание

1. Асимптотики и правила игры.

1.1. Условия

1 теория в неделю, 1 практика в неделю.

- Теор. часть.
  Дедлайн: вторник, 23:59. Потом придут исправления, которые надо сдать до пятницы ($\leqslant 23:59$). Сдача после дедлайна — понижение коэффициента. Два типа задач:
  1. Обязательные, $\Sigma = 10 - 15$
  2. Дополнительные. «Overprice»

  Домашки сдавать обязательно в $\TeX$, если вы не в группе А. Олемской. Дедлайны можно переносить, если вам тяжело/заболели, то можно попросить перенести дедлайн лично для вас. Но если делать так слишком часто, то это неоч :(.

- Контест. 3 три задач:
  1. Must Have. Если не сдал — пиши-пропало.
  2. Обязательные. Сумма маст хэвов и обязательных – 10-15 баллов.
  3. Дополнительные. «Overprice»

Не стоит сначала обращать внимание на мелочи. Пытайтесь вычленить основную идеи. Уже когда поймете её, стоит пытаться найти интересные случаи.

Как понять что вы поняли алгоритм? Сесть и подумать: можете ли вы прямо сейчас сесть и написать код. Если не можете, то надо задавать вопрос. **Думайте, перед тем, как писать**.

Обучение – интерактивный процесс, старайтесь включаться, если вы переходите в режим зрителя, то становится плохо.

1.2. Асимптотика

Как выбрать процессор? У процессора есть несколько остальных характеристик (примеры в скобках): количество ядер (8 ядер), частота (3.3 GHz), набор инструкций, битность (32/64).

У нас все алгоритмы однопоточные, поэтому для нас важна только частота.

+, -, * /	1 операция
a[i]	1 операция
if	1 операция
f(..)	1 операция
if	1 операция

TODO: Схема

Время $\to$ константа + асимптотика. Асимптотика, если просто, число операций, к которому стремится при увеличении количества входа.

Есть асимптотика используемого времени и памяти. Утверждается, что Время $\geqslant$ Память, потому что на выделение памяти тоже время (причем 1 ячейка = 1 операция). Второй момент, время довольно безгранично, а память конечна.

***Определение* 1.1.** $f = \mathcal{O}(g(n))$: $\exists C > 0 : \exists N : \forall n \geqslant N : f(n) \leqslant C \cdot g(n)$

***Определение* 1.2.** $f = \Theta(g(n))$: $\exists C_1 > 0, C_2 > 0 : \exists N : \forall n \geqslant N : C_1 \cdot g(n) \leqslant f(n) \leqslant C_2 \cdot g(n)$

***Определение* 1.3.** $f = o(g(n))$: $\forall C > 0 : \exists N : \forall n \geqslant N : f(n) \leqslant C \cdot g(n)$

Свойство. $f \pm o(f) = \Theta(f)$

Доказательство. $\exists N : \forall n \geqslant N : o(f) \leqslant \frac{1}{2}f \Rightarrow \frac{1}{2}f \leqslant f + o(f) \leqslant \frac{3}{2}f$ □

***Свойство транзитивности* Θ.** $\Theta(\Theta(f)) = \Theta(f)$

Доказательство. Внешняя и внутренняя Θ зажата константами, а значит можно сказать, что константы внутренней равны внешней. □

Лемма. $\forall P$: $P(n) = \Theta(n^{deg\ P})$

Доказательство. **TODO:** Я не успел, смотри конспекты Сережи. □

1.3. Умножение. Карацуба

$735 = 7 \cdot 100 + 3 \cdot 10 + 5 = 7x^2 + 3x + 5$, если $x = 10$. Пусть $n = x^2 + x +)$, а $m = 3x + 7$. Тогда $nm = (x^2 + x + 1)(3x + 7)$. Дальше для умножения многочленов можно просто раскрыть скобочки. Если записать это в коде, то получим:

```
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            c[i + j] += a[i] * b[j]
//          x^{i+j}     x^i  * x^j
```

Данный код работает за $\mathcal{O}(n^2)$. Долго человечество не могло решить задачу быстрее. Но Анатолий Карацуба придумал быстрее:

1. $n = m = 2^k$

2. Разобьем $A(x)$ и $B(x)$ на две половины: A_1 и A_2, B_1 и B_2.
   $A(x) \cdot B(x) = (A_1(x) + x^{\frac{n}{2}} \cdot A_2(x))(B_1(x) + x^{\frac{n}{2}} \cdot B_2(x)) = A_1B_1 + x^n \cdot A_2B_2 + x^{\frac{n}{2}} \cdot (A_1B_2 + A_2B_1)$.

TODO: схемы

Запишем время работы нашего алгоритма: $T(n) = 4T(\frac{n}{2}) + n$, что равно (магия пока что) $\Theta(n^2)$. Но давайте напишем алгоритм, работающий за $T(n) = 3T(\frac{n}{2}) + n \approx n^{1.6}$.

Для этого заметим, что $A_1B_2A_2B_1 = (A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2$. Здесь сложение многочленов это операция соответственного суммирования коэффициентов перед степенями.

```
Mul(A, B)
    A -> A1 A2
    B -> B1 B2
    c = Mul(A2, B2)
    d = Mul(A1, A2)
    e = Mul(A1 + A2, B1 + B2)
    ...
```

1.4. Мастер-Теорема

Теорема 1.1 (Мастер-Теорема)**.** Пусть $T(n) = aT(\frac{n}{b}) + n^c$, где $a > 0, b > 1, c \geqslant 0$.
Тогда:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & a > b^c. \\ \Theta(n^c), & a = b^c.\,. \\ \Theta(n^c \log n), & a < b^c \end{cases}$$

Пример. $T(n) = 2T(\frac{n}{2}) + n^2 = n^2 + 2(\frac{n}{2})^2 + 4(\frac{n}{4})^2 + \ldots = n^2(1 + \frac{1}{2} + \frac{1}{4} + \ldots) = n^2 \cdot 2 = \Theta(n^2)$

Доказательство. $T(n) = aT(\frac{n}{b}) + n^c = n^c + a(\frac{n}{b})^c + a^2(\frac{n}{b^2})^c + \ldots = n^{\cdot}(1 + \frac{a}{b^c} + (\frac{a}{b^c})^2 + \ldots)$. Тогда рассмотрим случаи:

- $a < b^c \Rightarrow \Theta(n^c)$
- $a = b^c \Rightarrow \Theta(n^c \log n)$
- $a > b^c$. Тогда под скобкой получается сумма вида $1 + x + x^2 + x^3 + \ldots$. Заметим, что сумма равна $\frac{(x^{k+1}-1)}{x-1}$, где -1 – константа, $x - 1$ – тоже. Тогда сумма равна $\Theta(x^{k+1}) = \Theta(x^k)$. Тогда получаем $\Theta(n^c \cdot (\frac{n}{b^c})^{\log_b n}) = \Theta(n^{\log_b a})$

□

Утверждение 1.2. $\forall a, b, c > 0$: $\log^a n < n^b < c^n$

Доказательство. Смотри в конспекте у Сережи. □

2. Линейные структуры данных. Стек

2.1. Ликбез по плюсам

Мне было вломас, сорян-борян (я контест закрывал).

2.2. Время

- `sqrt, pow, sin/cos, log` — медленные функции.
- `cin/cout` — довольно медленный ввод. Самый быстрый — `fread/fwrite`. Быстрый ввод-вывод реализован в `optimization.h`.
- Работа с памятью. `vector` и массивы работают за ±одинаковое время (разница в 1% чувствуется при очень больших n, но это только если тупо создания). Проблема, если выделяется/освобождается большое количество маленьких векторов.
- Доступ к памяти. Кеш. Есть несколько уровней памяти: регистры, кеш `L1`, кеш `L2`, кеш `L3`, ОЗУ, HDD. Каждый уровень работает в 2–10 раз больше (HDD раз в 100). Поэтому стоит делать последовательный доступ к элементам, а не случайный. Все из-за подгрузки в кеш.
- Деление и взятие по модулю.
- Рекурсия. Вход в рекурсия не тривиальная операция: нужно переложить регистры, чето-еще чето-еще....

А теперь быстрые операции:

- Операции с `char`, `int`. Все работает из-за векторизации: процессор умеет складывает сразу несколько чисел, которые не пересекаются.

2.3. Сумма на отрезке

Дан массив, поступают запросы $[l, r]$ — сумма на отрезке l, r. Запросов q. Решения:

1. Тупое решение $\mathcal{O}(n \cdot q)$.
2. Префиксные суммы. Насчитаем массив `F[i]` — сумма на префиксе длины i. Тогда запрос: `F[r] - F[l]`. Работает за $\mathcal{O}(n + q)$.

2.4. Массив и вектор

Массив — просто набор чисел, а вектор еще умеет динамически расширяться (добавляться в конец). Есть `init(n), get(i), set(i, x)`. Все работает за $\mathcal{O}$.

Простое решение: каждый раз перевыделять память. Работает за $\Theta(n^2)$.

Нормальное решение будем выделять $2n$ операций. Теперь добавление в конец работает за $\mathcal{O}(1)$, если размер меньше, чем аллоцированный размер. Иначе переаллоцируем на $2n$. В среднем работает за $\mathcal{O}(1)$.

2.5. Список

Для каждого элемента списка хранится два указателя: голова (`HEAD`) и хвост (`TAIL`). Ну и у самого списка есть и то, и другое. Можно подцепить к началу и к концу. Удаление из начала из конца.

Можно добавлять после iой позиции. Для этого его надо сначала найти.

2.6. Стек

Добавление в конец, убирание из конца. Реализация на массиве: просто поддерживаем конец.

2.7. Очередь

Добавление в конец, убирание из начала. Можно реализовать на односвязном списке. Можно на массиве, поддерживая два указателя: конец и начало.

2.8. Дек

Добавление в конец/начало, взять из начала/конца. Можно реализовать так же, как и вектор, просто теперь выделять не только в конце, но и перед. А можно просто двусвязный список. Преимущества массива: быстрота: кеш, меньше памяти. Преимущества списка: точная $\mathcal{O}(1)$.

2.9. Стек с минимумом

Просто теперь будет хранить не просто число x, но и $(x, \texttt{min}(x, \texttt{stack.head()}))$.

2.10. Очередь с минимумом

Реализуем на двух стеках с минимумом: добавляем в первый список. Теперь если надо вытащить, то проверяем сначала второй стек, если он пустой, то сразу переложим весь первый стек во второй. То есть перевернем первый стек. Минимум — минимум на двух стеках. Каждый элемент переживет 4 операции: добавление в 1 стек, удаление из 1, добавление во 2 стек, удавление из второго стека. На каждый элемент $\mathcal{O}(1)$ операций.

3. Структуры данных два

3.1. Амортизационные анализ

***Определение* 3.1.** Пусть у нас программа разбита на кусочки. Тогда t_i — реальное время работы. Тогда пусть есть какая-то функция ϕ. Амортизационное время работы $a_i = t_i + \overbrace{\varphi_{i+1} - \varphi_i}^{\Delta\varphi}$

Среднее время работы программы: $\frac{\sum t_i}{m} = \frac{\sum a_i}{m} - \frac{\Delta\varphi_i}{m} = \frac{\sum a_i}{m} - (\varphi_n - \varphi_0)$

Теорема 3.1. $\frac{\sum t_i}{m} \leqslant \max a_i$.

Доказательство. Очевидно. $\square$

3.2. Вектор. Реальное время работы.

$t_i = 1\ 1\ 1\ 1\ 1\ 1\ 1\ n$. Тогда $a_i = t_i + \varphi$. Интересно рассмотреть функцию $\varphi = -\texttt{size}$. Тогда операция без удвоения $a_i = n + (-n) = 0$, а операция у удвоением $a_i = 1 + 0 = 1$. Тогда $\max a_i = \mathcal{O}(1)$. Еще есть добавочка, так как наша φ — плохая, тогда среднее время работы равно $\max |a| + \frac{2m}{m}$

3.3. $a^2 + b^2 = N$ двумя указателями

Применим время работы к этой задаче. Хотим $a_i = t_i + \Delta\varphi = \mathcal{O}(1)$. В качестве функции φ можно выбрать B. $\varphi \geqslant 0$, но $\phi_0 \neq 0$. Ну тогда

$$\frac{\sum t_i}{m} \leqslant \max a_i + \frac{\max |\varphi|}{m} = \mathcal{O}(1).$$

Т.к. $\max |\varphi| = m = \sqrt{N}$, а $\max a_i = \mathcal{O}(1)$.