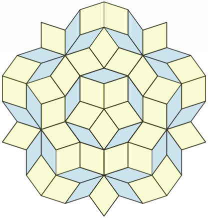

Сложность вычислений и бенчмаркинг

Летняя практика, Переславль–Залесский

4–8 июля, 2022 г.

Воспоминание I: Сложность в худшем случае

- $f(x) = O(g(x)) \Leftrightarrow \exists C(f(x) < C \cdot g(x))$ для достаточно больших x;
- $f(x) = \Omega(g(x)) \Leftrightarrow \exists c(f(x) > c \cdot g(x))$ для достаточно больших x;
- $f(x) = \Theta(g(x)) \Leftrightarrow \exists c, C(C \cdot g(x) > f(x) > c \cdot g(x))$ для достаточно больших x.

Для теоретически обоснованных оценок важно выбрать дискретный параметр x (размер структуры данных, количество определённого рода операций, и т.д.). Параметр должен быть абстрактным (в терминах алгоритма), а не физическим.

Почему не время / память?

- Колебания значений из-за посторонних процессов (вмешательство фоновых программ, неявное распараллеливание). Можно отчасти решить включением специального тестирующего скрипта.
- Зависимость от компилятора и набора оптимизаций в нём. Эта зависимость делает ненадёжным также подсчёт конкретных низкоуровневых инструкций.

Колебания, вносимые в пункте 1, можно сгладить применением аппроксимационных оценок, т.к. распараллеливание в общем случае даёт сублинейный выигрыш. Колебания, вносимые в пункте 2, могут привести к изменению асимптотики.

Техника оценки сложности

- Вариант 1 (индуктивный): найти какую-нибудь меру сложности и оценить, как она уменьшается пошагово;

Сортировка пузырьком чисел от 1 до N. Сопоставим каждому массиву A размера N меру неупорядоченности: кортеж, где $\mathrm{N}-i$-ый элемент — это число элементов $A[j]$, таких что $|A[j]-j| = i$. Например, массиву $(3\ 4\ 2\ 1)$ соответствует кортеж $\langle 1,\ 2,\ 1,\ 0\rangle$; отсортированному массиву — кортеж $\langle 0,\ \ldots,\ \mathrm{N}\rangle$. Тогда мера неупорядоченности будет убывать лексикографически с каждым шагом сортировки. Это грубая оценка, т.к. допускает перепрыгивания типа $\langle 1,\ \ldots,\ \mathrm{N}-1\rangle \to \langle 0,\ \mathrm{N},\ \ldots,\ 0\rangle$, каких не может быть при реальной сортировке. Но если заметить, что уменьшение каждого элемента кортежа не может быть сделано более, чем N раз, то получается уже адекватная оценка $O(\mathrm{N}^2)$.

Техника оценки сложности

- Вариант 1 (индуктивный): найти какую-нибудь меру сложности и оценить, как она уменьшается пошагово;
- Вариант 2 (конструктивный): найти худший случай и оценить количество шагов для него.

Та же самая мера неупорядоченности подсказывает, как найти худший случай. Для любого массива размера N не более, чем 2 элемента имеют отличие в позициях от значений на $N-1$. Далее по индукции. После чего достаточно посчитать количество шагов сортировки на построенном массиве.

Данное построение не обосновывает, почему хуже, чем этот случай, не бывает, однако позволяет построить полноценный бенчмарк для «достаточно плохих» случаев.

Техника оценки сложности

- Вариант 1 (индуктивный): найти какую-нибудь меру сложности и оценить, как она уменьшается пошагово;
- Вариант 2 (конструктивный): найти худший случай и оценить количество шагов для него.

На практике при работе со сложными структурами данных точные «худшие случаи» почти никогда не ищут. Намного проще работать с «достаточно плохими» относительно выбранной меры сложности[a].

[a] Тем более, что точные худшие случаи часто являются патологиями, которые никогда не встречаются при реальном применении алгоритма.

Сложность «в среднем»

- Требует анализа распределения данных и адекватно работает для простых структур (массивы, произвольные графы).
- Очень плохо работает для сложных структур: например, «случайная» (т.е. случайно порождённая) программа содержит грубые семантические ошибки с вероятностью 1. Случайно порождённое уравнение в словах не имеет решений, причём это устанавливается тривиальными алгоритмами с вероятностью 1.
- На практике сложность в среднем оценивается на фрагментах реальных программ, либо требует порождения специальных бенчмарков по нетривиальным алгоритмам.

Воспоминание II: рекуррентные соотношения

Если функция $f(x)$ определяется через равенство $f(x) = G(f(h_1(x)), \ldots, f(h_k(x)), x)$, то говорят, что f задаётся рекуррентным соотношением. Если $f(x) = G\left(f\left(h_1(x)\right), \ldots, f\left(h_k(x)\right)\right)$, то рекуррентность называется однородной. Если $\forall i (h_i(x) = x - s_i)$, а G определяет линейную комбинацию вызовов f, то рекуррентность называется линейной.

- Возникают при оценках сложности по индукции.
- Описывают сложность рекурсивных алгоритмов.

Воспоминание II: рекуррентные соотношения

Если функция $f(x)$ определяется через равенство $f(x) = G(f(h_1(x)), \ldots, f(h_k(x)), x)$, то говорят, что f задаётся рекуррентным соотношением. Если $f(x) = G\left(f\left(h_1(x)\right), \ldots, f\left(h_k(x)\right)\right)$, то рекуррентность называется однородной. Если $\forall i(h_i(x) = x - s_i)$, а G определяет линейную комбинацию вызовов f, то рекуррентность называется линейной.

- Переборные EXPTIME-алгоритмы: линейные рекуррентности по размеру входных данных. Например, $f(n) = 2 \cdot f(n-1) + n^2$.
- Субэкспоненциальные алгоритмы: как линейные (например, $f(n) = 2 \cdot f(n-1) - f(n-2)$), так и нелинейные (например, $f(n) = 2 \cdot f\left(\frac{n}{2}\right) + n$).

Мастер–теорема («разделяй и властвуй»)

Пусть функция $f(n)$ описывается следующим рекуррентным соотношением:

$$f(n) = a \cdot f\left(\frac{n}{b}\right) + g(n)$$

Тогда сложность $f(n)$ оценивается так:

- если $g(n) = O(n^c)$, где $c < \log_b a$, тогда $f(n) = \Theta(n^{\log_b a})$;
- если $g(n) = \Theta(n^{\log_b a})$, тогда $f(n) = \Theta(n^{\log_b a} \cdot \log n)$;
- если $g(n) = \Omega(n^c)$, где $c > \log_b a$, причём асимптотически верно, что для некоторой величины $t < 1$ $a \cdot g\left(\frac{n}{b}\right) \leqslant t \cdot g(n)$, тогда $f(n) = \Theta\left(g(n)\right)$.

Сочетание условий $c > \log_b a$ и $a \cdot g\left(\frac{n}{b}\right) \leqslant t \cdot g(n)$ выполняется почти всегда.

Экспоненциальные рекуррентности

Классика жанра анализа программ: алгоритм полиномиален по размеру графа, а граф экспоненциален от входных данных. Переход к оценке по размеру входа порождает рекуррентные соотношения уже другой формы, асимптотику которых, однако, также можно оценить с помощью мастер-теоремы.

- Пример оценки:

$$f(n) = 3 \cdot f(n-1) + 2^n$$

Как поступить с двумя рекурсивными вызовами: $f(n) = a_1 \cdot f(n-k_1) + a_2 \cdot f(n-k_2) + g(n)$?

Экспоненциальные рекуррентности

- Частный случай:

$$f(n) = 3 \cdot f(n-1) + 2^n$$

- Общий случай:

$$f(n) = a \cdot f(n-1) + b^n$$

Метод замены переменных: вводим функцию g такую, что $g(b^n) = f(n)$, и затем осуществляем замену b^n на n'.

Как поступить с двумя рекурсивными вызовами: $f(n) = a_1 \cdot f(n-k_1) + a_2 \cdot f(n-k_2) + g(n)$?

Пример рекуррентности

Даны слова $S_1 = a_1 \dots a_n$, $S_2 = b_1 \dots b_m$. Найти их максимальную общую подпоследовательность.

Это NP-трудная задача. Будем анализировать очень наивный алгоритм её решения, связанный с полным перебором всех возможных подпоследовательностей:

- Находясь на i-й позиции в S_1, ищем все подпоследовательности суффикса $a_i \dots a_n$ и слова S_2, включающие a_i. То есть мы либо успешно сопоставляем a_i с b_1 и решаем задачу для строк $a_{i+1} \dots a_n$ и $b_2 \dots b_m$, либо решаем задачу для строк $a_i \dots a_n$ с $b_2 \dots b_m$ так, что a_i сопоставляется с каким-нибудь b_k.
- Сдвигаем i на 1 вправо.

Рекуррентное соотношение, дающее оценку количества перебираемых случаев, выглядит следующим образом: $g(n+1, m+1) = g(n, m+1) + g(n+1, m)$.

Пример рекуррентности

Даны слова $S_1 = a_1 \dots a_n$, $S_2 = b_1 \dots b_m$. Найти их максимальную общую подпоследовательность.

Рекуррентное соотношение, дающее оценку количества перебираемых случаев, выглядит следующим образом: $g(n+1,\, m+1) = g(n,\, m+1) + g(n+1,\, m)$.

Положим $g(n,\, 1) = g(1,\, n) = n$ (побуквенный перебор), и оценим для начала $h_2(n) = g(2,\, n)$. Получаем соотношение: $h_2(n+1) = n + 1 + h_2(n)$. То есть $h_2(n) = O(n^2)$.

Очевидно, $h_{i+1}(n) = h_i(n) + h_{i+1}(n-1)$, из чего можно вывести $h_i(n) = O(n^i)$. На практике эта оценка несколько лучше: $O\left(\frac{n^i}{i!}\right)$.

Неудобство заключается в том, что для построения этой оценки мы вынуждены пользоваться знаниями о сумме ряда $1^i + 2^i + \dots n^i$, а хотелось бы уметь выводить их в общем виде.

Пример рекуррентности

Даны слова $S_1 = a_1 \dots a_n$, $S_2 = b_1 \dots b_m$. Найти их максимальную общую подпоследовательность.

Рекуррентное соотношение, дающее оценку количества перебираемых случаев, выглядит следующим образом: $g(n+1,\ m+1) = g(n,\ m+1) + g(n+1,\ m)$.

Заметим, что если положить $g(n,\ 1) = g(1,\ n) = q$ (проверка на вхождение буквы в конечный словарь стоимостью в константу q), то все оценочные полиномы сразу станут на 1 степень меньше. И хотя диагональная функция $h_n(n) = g(n, n)$ всё равно растёт экспоненциально быстро, использование словаря даже только для базисных случаев даёт заметный выигрыш в скорости.

Формальные степенные ряды

$\mathbb{K}$ — какое-нибудь (числовое) поле. Существует взаимно-однозначное соответствие между списками элементов из $\mathbb{K}$ и многочленами с коэффициентами из $\mathbb{K}$. Т.е. представим списки чисел как «многочлены» (с бесконечным числом коэффициентов):

$$\sum_{n=0}^{\infty} a_n x^n$$

Приятная неожиданность

- Покоэффициентное сложение, коммутативное и ассоциативное.
- Умножение, коммутативное и ассоциативное.

Формальные степенные ряды

Производящая функция последовательности $\{a_n\}$ — это формальный степенной ряд:

$$\sum_{n=0}^{\infty} a_n x^n$$

Приятная неожиданность

- Покоэффициентное сложение, коммутативное и ассоциативное.
- Умножение, коммутативное и ассоциативное.
- Расширение алгебры многочленов! Частное и производная.

Значения, принимаемые x — не численные, а комбинаторные.

Краткое представление ряда

Пусть $P(x)$ — полином степени n. Рассмотрим «обращённый» полином $P^R(x) = P(\frac{1}{x}) \cdot x^n$ (тот же самый, но упорядоченный в перевёрнутом порядке).

Что получится, если формально разделить в столбик 1 на $(-k \cdot x + 1)^R$?

Получаем способ краткой записи формального ряда (списка) в виде частного обратных полиномов.

Способы краткого представления

- ПФ для частного $\frac{1}{1-k\cdot x}$ — это $\sum\limits_{i=0}^{\infty} k^i x^i$.
- Умножение на n-ую степень x — это сдвиг на n.
- ПФ для частного $\frac{P(x)}{Q(x)}$, где $Q(x)$ — многочлен без кратных и мнимых корней, получается с помощью разложения в сумму простых дробей со знаменателями вида $1 - k \cdot x$.

Упражнение: превратить рекуррентность для чисел Фибоначчи в частное вида $\frac{1}{P(x)}$ (или подсмотреть в интернете), а потом разделить 1^R на $P(x)^R$ в столбик.

Теорема о рациональных ПФ

Квазиполином — это сумма вида $\sum \beta_i \cdot x^{q_i} \cdot {k_i}^{\alpha_i \cdot x}$.

Следующие утверждения относительно последовательности $\{a_n\}$ эквивалентны:

- ПФ $\{a_n\}$ является рациональной (может быть представлена в виде частного полиномов $P(x)/Q(x)$), где $Q(x) = 1 - c_1 \cdot x - \cdots - c_k \cdot x^k$;
- $f(i) = a_i$ описывается квазиполиномом, основания экспонент которого определяются корнями $Q(x)$;
- $\{a_n\}$ описывается линейной рекуррентностью $a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} + \cdots + c_k \cdot a_{n-k}$.

Более точно, если $Q(x) = (1 - q_1 \cdot x)^{t_1} \cdot \ldots (1 - q_k \cdot x)^{t_k}$, то $a_n = O(q_1^n \cdot n^{t_1 - 1} + \cdots + q_k^n \cdot n^{t_k - 1})$. Случай мнимых корней $Q(x)$ в лекции не рассматривается.

Алгоритм приведения к дроби

Ищем краткое представление для $G = \sum\limits_{j \geqslant 0} a_j x^j$.

Строим уравнения на коэффициенты производящей функции и умножаем на подходящие степени x:

$$
\begin{array}{ll}
a_k = c_1 \cdot a_{k-1} + c_2 \cdot a_{k-2} + \cdots + c_k \cdot a_0 & (\cdot x^0) \\
a_{k+1} = c_1 \cdot a_k + c_2 \cdot a_{k-1} + \cdots + c_k \cdot a_1 & (\cdot x^1) \\
\ldots & \\
a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} + \cdots + c_k \cdot a_{n-k} & (\cdot x^{n-k}) \\
\ldots &
\end{array}
$$

Алгоритм приведения к дроби

Ищем краткое представление для $G = \sum_{j \geqslant 0} a_j x^j$.

$$
\begin{array}{ll}
a_k = c_1 \cdot a_{k-1} + c_2 \cdot a_{k-2} + \cdots + c_k \cdot a_0 & (\cdot x^0) \\
a_{k+1} = c_1 \cdot a_k + c_2 \cdot a_{k-1} + \cdots + c_k \cdot a_1 & (\cdot x^1) \\
\ldots & \\
a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} + \cdots + c_k \cdot a_{n-k} & (\cdot x^{n-k}) \\
\ldots &
\end{array}
$$

Теперь складываем все эти уравнения:

$$\sum_{j \geqslant k} a_j x^{j-k} = c_1 \cdot \sum_{j \geqslant k-1} a_j x^{j-k+1} + \cdots + c_k \cdot \sum_{j \geqslant 0} a_j x^j$$

Алгоритм приведения к дроби

Ищем краткое представление для $G = \sum_{j\geqslant 0} a_j x^j$.

$$\sum_{j\geqslant k} a_j x^{j-k} = c_1 \cdot \sum_{j\geqslant k-1} a_j x^{j-k+1} + \cdots + c_k \cdot \sum_{j\geqslant 0} a_j x^j$$

Чтобы стало возможным подставить G, умножаем сумму на x^k:

$$\sum_{j\geqslant k} a_j x^j = c_1 x \cdot \sum_{j\geqslant k-1} a_j x^j + \cdots + c_k x^k \cdot \sum_{j\geqslant 0} a_j x^j$$

Алгоритм приведения к дроби

Ищем краткое представление для $G = \sum_{j\geqslant 0} a_j x^j$.

$$\sum_{j\geqslant k} a_j x^j = c_1 x \cdot \sum_{j\geqslant k-1} a_j x^j + \cdots + c_k x^k \cdot \sum_{j\geqslant 0} a_j x^j$$

Выражаем результат в терминах G и нескольких первых значений a_i:

$$G - \left(\sum_{j<k} a_j x^j\right) = c_1 x \cdot \left(G - \sum_{j<k-1} a_j x^j\right) + \cdots + c_k x^k \cdot G$$

Итог:

$$G = \frac{\sum_{j<k} a_j x^j - c_1 x \cdot \sum_{j<k-1} a_j x^j - \cdots + c_{k-1} x^{k-1} a_0}{1 - c_1 x - \cdots - c_k x^k}$$

Пример оценки

Применим формальные ряды для оценки скорости роста $a_n = h_2(n) = n + h_2(n-1)$.

Берём начальное значение h_2: $h_2(0) = 1$, чтобы получить $a_1 = h_2(1) = g(2,1) = 2 = a_0 + 1$.
Соотношение имеет дополнительное слагаемое (n), поэтому придётся повторять конструкцию суммы по шагам.

$$\sum_{j\geqslant 1} a_j x^j = x \cdot \sum_{j\geqslant 0} a_j x^j + x \cdot \sum_{j\geqslant 0} (j+1)\cdot x^j.$$

Пример оценки

Применим формальные ряды для оценки скорости роста $a_n = h_2(n) = n + h_2(n-1)$.

Берём начальное значение h_2: $h_2(0) = 1$, чтобы получить $a_1 = h_2(1) = g(2,1) = 2 = a_0 + 1$.

$$\sum_{j\geqslant 1} a_j x^j = x \cdot \sum_{j\geqslant 0} a_j x^j + x \cdot \sum_{j\geqslant 0} (j+1) \cdot x^j.$$

Ряд $\sum\limits_{j\geqslant 0} (j+1) \cdot x^j = 1 + 2 \cdot x + 3 \cdot x^2 + \dots$ — можно заметить, что это производная знакомого нам ряда $1 + x + x^2 + \dots + x^n + \dots$.

В терминах частных, $\left(\frac{1}{1-x}\right)' = \frac{1}{(1-x)^2}$. Дальше всё просто.

$$H_2 - 1 = x \cdot H_2 + \frac{x}{(1-x)^2}$$

Пример оценки

Применим формальные ряды для оценки скорости роста $a_n = h_2(n) = n + h_2(n-1)$.

Берём начальное значение h_2: $h_2(0) = 1$, чтобы получить $a_1 = h_2(1) = g(2,1) = 2 = a_0 + 1$.

$$H_2 - 1 = x \cdot H_2 + \frac{x}{(1-x)^2}$$

$$H_2 = \frac{1 - x + x^2}{(1-x)^3}$$

Пример оценки

Применим формальные ряды для оценки скорости роста $a_n = h_2(n) = n + h_2(n-1)$.

Берём начальное значение h_2: $h_2(0) = 1$, чтобы получить $a_1 = h_2(1) = g(2,1) = 2 = a_0 + 1$.

$$H_2 = \frac{1 - x + x^2}{(1-x)^3}$$

Разделив 1 на $1 - 2x + x^3$ в столбик, мы действительно можем убедиться, что получаются нужные оценки. Заодно из неоднородной рекуррентности $a_n = a_{n-1} + n$ мы построили однородную: $a_n = 3 \cdot a_{n-1} - 3 \cdot a_{n-2} + a_{n-3}$.
Чтобы оценить рост функции h_2, воспользуемся теоремой о рекуррентных соотношениях. Корень у знаменателя единственный, это 1, его кратность равна 3, поэтому $h_2(n) = O(1^n \cdot n^2) = O(n^2)$. Что и требовалось доказать.

Связь ПФ и М.- Т.

Оценить асимптотику роста n-ого коэффициента производящей функции, заданной следующей рекуррентностью:

$$f(n) = a \cdot f(n-1) + b^{c \cdot n}$$

Вывести из этой оценки хотя бы один частный случай Мастер–Теоремы.

Выбор системы тестов

- Почти однородная: при разрастании изменяется количество применений частей алгоритма, но не их порядок или выборка;
- Накрывающая: для каждого подалгоритма, для которого возможно подобрать свою систему тестов, изолированную от прочих, желательно это сделать;
- Слепая: системы тестов должны быть подобраны исходя из теоретических свойств алгоритма, без подгонки под конкретную реализацию.

Задача из ВКР

Размер графа разбора конъюнктивной грамматики в зависимости от номера теста:

Номер теста	тип A		тип B		тип C	
	Узлы	Ветви	Узлы	Ветви	Узлы	Ветви
1	39	55	39	55	39	55
2	58	86	82	124	124	203
3	81	129	130	205	270	491
4	108	186	181	292	490	969
5	139	259	235	385	799	1701
6	174	350	292	484	1212	2759
7	213	461	352	589	1744	4223
8	256	594	415	700	2410	6181
9	303	751	481	817	3225	8729
10	354	934	550	940	4204	11971

Метод конечных разностей

- Строим разности между соседними элементами в ряду оценок.
- Продолжаем этот процесс над разностями, и т.д. по необходимости.
- Если алгоритм полиномиален, процесс сходится по вертикали — на итерации, соответствующей степени полинома, получаются константы.
- Если алгоритм экспоненциален, процесс сходится по диагонали — между j-ой позицией в j-ом столбике и $j+1$-ой позицией в $j+1$-ом разности стабилизируются.
- Если алгоритм имеет логарифмический множитель, это нужно определить по номерам тестов, на которых происходит скачок. И дальше исследовать фрагмент, где логарифм сохраняет значение.