Алгоритмы и структуры данных

Лекция 2. Сложность алгоритмов

Антон Штанюк (к.т.н, доцент)

18 февраля 2021 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

Содержание

Оценка сложности алгоритмов

Сложность алгоритмов

Сложность алгоритма помогает оценить затраты на его реализацию и определяется вычислительными мощностями, необходимыми для его выполнения.

Она часто измеряется двумя параметрами:

1. **T** - временная сложность
2. **S** - пространственная сложность

Сложность представляется как функция $f(n)$, где n - размер входных данных. Чаще всего под размером понимают *число* обрабатываемых элементов (размер массива данных).

Определим $f(n)$ как *рабочую функцию*, дающую верхнюю границу для максимального числа основных операций (сложения, сравнения и т.д.), которые должен выполнить алгоритм.

Асимптотическую сложность

Определим асимптотическую сложность алгоритма.

Если алгоритм обрабатывает входную последовательность размера **n** за время cn^2, то говорят, что временная сложность этого алгоритма - $\Theta(n^2)$

Найдутся такие константы $c_1, c_2 > 0$ и такое число n_0, что $c_1 n^2 \leqslant f(n) \leqslant c_2 n^2$ при всех $n \geqslant n_0$.

Вообще, если $g(n)$ - некоторая функция, то запись $f(n) = \Theta(g(n))$ означает, что найдутся такие $c_1, c_2 > 0$ и такое n_0, что $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ при всех $n \geqslant n_0$.

Запись $f(n) = \Theta(g(n))$ включает в себя две оценки - верхнюю и нижнюю. Их можно разделить (в данном случае нас больше интересует верхняя оценка). Говорят, что $f(n) = O(g(n))$, если найдется такая константа > 0 и такое число n_0, что $0 \leqslant f(n) \leqslant cg(n)$ для всех $n \geqslant n_0$.

Сложность алгоритмов

Также существуют определения:

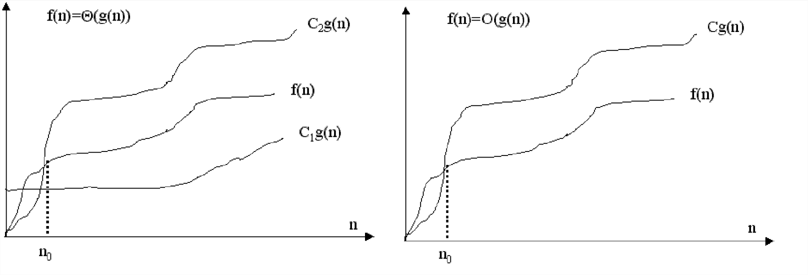
Функция $f(n)$ является $O[g(n)]$ (о-большое) для больших n, если:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = const \neq 0$$

Функция $f(n)$ является $o[g(n)]$ (о-малое) для больших n, если:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Сложность алгоритмов



Оценка сложности

Определяя асимптотическую сложность алгоритма, мы отбрасываем члены меньшего порядка суммы, которые при больших n становятся малыми, по сравнению с другими слагаемыми.

Пример

Если временная сложность алгоритма описывается как $T(n) = 4n^2 + 7n + 12$, то вычислительная сложность определяется, как $O(n^2)$.

Запись оценки сложности позволяет увидеть, как объем входных данных влияет на требования ко времени выполнения. Например, если $T(n) = O(n)$, то удвоение входных данных удвоит и время работы алгоритма. Если $T = O(2^n)$, то добавление одного бита к входным данным удвоит время выполнения.

Классификация по сложности

Классификация алгоритмов по сложности

Обычно алгоритмы классифицируют в соответствии с их временной сложностью. Можно выделить следующие их типы:

1. Постоянный - сложность оценивается как $O(1)$.
2. Логарифмический - сложность оценивается как $O(log(n))$
3. Линейный - оценка равна $O(n)$.
4. Квадратный - $O(n^2)$
5. Кубический, полиноминальный - $O(n^3), O(n^m)$.
6. Экспоненциальный - $O(t^{p(n)})$, t- константа, $p(n)$ - некоторая полиномиальная функция.
7. Факториальный - $O(n!)$. Обладает наибольшей временной сложностью среди всех известных типов.

Классификция по сложности

Логарифмическая сложность присуща алгоритмам, которые сводят большую задачу к набору меньших задач, уменьшая на каждом шаге размер задачи на постоянную величину. Например, двоичный поиск в массиве, когда на каждом шаге размер массива сокращается вдвое.

Линейное время выполнения свойственно тем алгоритмам, в которых осуществляется небольшая обработка каждого входного элемента.

Оценка $nlog(n)$ возникает в тех случаях, когда алгоритм решает задачу, разбивая её на меньшие подзадачи и решая их независимо друг от друга, а затем объединяя решение.

Классификция по сложности

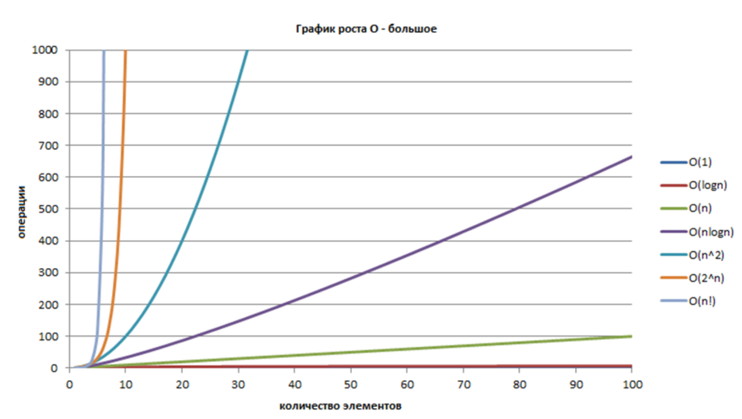
Квадратичное время выполнения свойственно алгоритмам, обрабатывающим все пары элементов данных.

Кубическое время соответсвует алгоритмам, которые обрабатывают все тройки элементов данных.

Экспоненциальное и факториальное время присуще алгоритмам, которые выполняют перебор всевозможных сочетаний элементов.

Классификция по сложности

Для того, чтобы визуально представить себе различную скорость роста функций, достаточно взглянуть на следующий график:



С ростом n временная сложность может стать настолько огромной, что это повлияет на практическую реализуемость алгоритма. Рассмотрим таблицу, в которой сравнивается время выполнения алгоритмов разных типов при $n = 10^6$, при условии, что единицей времени для компьютера является микросекунда (10^{-6}).

Тип	Сложность	Кол-во операций	Время при 10^6 операций в сек.
Постоянные	$O(1)$	1	1мкс
Линейные	$O(n)$	10^6	1 с
Квадратичные	$O(n^2)$	10^{12}	11.6 дн.
Кубические	$O(n^3)$	10^{18}	32000 лет
Экспоненциальные	$O(2^n)$	10^{301030}	в 10^{301006} раз больше времени существования Вселенной

Степени двойки

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536

2^{17}	$131,072$
2^{18}	$262,144$
2^{19}	$524,288$
2^{20}	$1,048,576$
2^{21}	$2,097,152$
2^{22}	$4,194,304$
2^{23}	$8,388,608$
2^{24}	$16,777,216$
2^{25}	$33,554,432$
2^{26}	$67,108,864$
2^{27}	$134,217,728$
2^{28}	$268,435,456$
2^{29}	$536,870,912$
2^{30}	$1,073,741,824$
2^{31}	$2,147,483,648$
2^{32}	$4,294,967,296$

Быстрая оценка больших чисел

$2^{24} = 2^{10} \times 2^{10} \times 2^{4} \approx 1000 \times 1000 \times 16 = 16$ млн. (ошибка 4.6%)

$2^{32} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{2} \approx 1000 \times 1000 \times 1000 \times 4 = 4$ млрд. (ошибка 6.9%)

Оценки времени

Часто приходится сталкиваться с показателями времени, выраженных в секундах. В следующей таблице содержится перевод секунд в другие единицы измерения времени

10^2 с.	1.7 мин
10^3 с.	16.7 мин
10^4 с.	2.8 часа
10^5 с.	1.1 дня
10^6 с.	1.6 недели
10^7 с.	3.8 мес
10^8 с.	3.1 года
10^9 с.	31 год
10^{10} с.	310 лет

Сложность проблем

Существует теория сложности, которая классифицирует не только сложность самих алгоритмов, но и сложность самих задач. Теория рассматривает минимальное время и объем памяти, необходимые для решения самого трудного варианта проблемы на теоретическом компьютере, или машине Тьюринга.

Проблемы, которые можно решить с помощью алгоритмов с полиномиальным временем, называют решаемыми, потому что при разумных входных данных обычно могут быть решены за разумное время (точное определение " разумности" зависит от конкретных обстоятельств). Проблемы, которые невозможно решить за полиномиальное время, называют нерешаемыми, потому что нахождение их решений быстро становится невозможным. Нерешаемые проблемы иногда называют трудными.

Алан Тьюринг доказал, что некоторые проблемы принципиально неразрешимы, то есть даже отвлекаясь от временной сложности, невозможно создать алгоритм их решения.

Классы сложности задач

Существует теория сложности, которая классифицирует не только сложность самих алгоритмов, но и сложность самих задач.

Выделяют несколько классов сложности алгоритмов:

- Класс P. Задачи, относящиеся к классу сложности P, могут быть решены за **полиномиальное** время. Класс P является одним из самых узких классов сложности.
  - Примерами алгоритмов класса P являются стандартные алгоритмы целочисленного сложения, умножения, деления, взятия остатка от деления, перемножения матриц, выяснение связности графов и некоторые другие.

Классы сложности задач

Класс NP. Это класс **недетминированно-полиномиальных** алгоритмов. Для них невозможно (или пока невозможно) создать реализацию, выполняющуюся за полиномиальное время, но возможно получить некоторое решение недетерминированным способом, а затем проверить его допустимость за полиномиальное время.

- Среди всех задач класса NP можно выделить ”самые сложные” — NP-полные задачи. Если мы научимся решать любую из них за полиномиальное время, то все задачи класса NP можно будет решить за полиномиальное время.

Примеры трудных задач

Примеры трудных задач

Примеры NP-полных задач:

1. Задача о выполнимости булевых формул
2. Кратчайшее решение «пятнашек» размера $n \times n$
3. Задача коммивояжёра
4. Проблема раскраски графа
5. Задача о вершинном покрытии
6. Задача о покрытии множества
7. Задача о клике
8. Задача о независимом множестве
9. Задача о рюкзаке
10. Сапер (игра)
11. Тетрис

Примеры трудных задач

Рассмотрим проблему вскрытия алгоритма шифрования по ключу. Временная сложность такого вскрытия пропорциональна числу возможных ключей, которое экспоненциально зависит от длины ключа. Если n - длина ключа, то сложность вскрытия грубой силой равна $O(2^n)$. В следующей таблице приведены параметры перебора с использованием суперкомпьютера Cray X1 (50 TFlop/s).

Длина ключа, бит	Кол-во вариантов	Время перебора
40 (5 символов)	10^{12}	0.02 с
48 (6 символов)	10^{14}	2 с
56 (7 символов)	10^{17}	33 мин
64 (8 символов)	10^{19}	55 час
80 (10 символов)	10^{24}	633 года

При n бит сложность равна 2^{56} и в этом случае вскрытие возможно за приемлемое время. При $n = 112$ бит сложность равна 2^{112} вскрытие становится невозможным.

Примеры трудных задач

Рассмотрим в качестве примера задачу комивояжера.

Задача комивояжера

Комивояжер должен объехать N городов с целью осуществления продажи своих товаров. Все N городов соединены дорогами по принципу ”каждый с каждым”. Известна стоимость проезда между двумя любыми городами. Найти оптимальный маршур движения так, чтобы побывать во всех городах и при этом иметь минимальные затраты на дорогу.

Исходная информация задана в виде перечня городов и соответствующей матрицы стоимостей, то есть двумерного массива с элементами C_{ij}, равными стоимости проезда из города i в город j. В данном случае матрица имеет N строк и N столбцов.

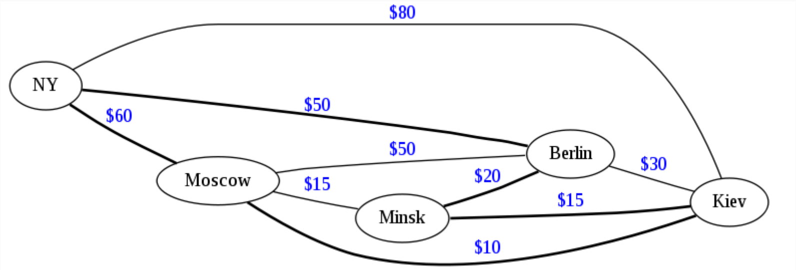
Следует также уточнить, что маршрут начинается и заканчивается в одном (базовом) городе и не может дважды проходить через один и тот же город.

Примеры трудных задач

<u>Решение</u> (метод грубого перебора).

Произвольно пронумеруем N городов целыми числами от 1 до N, причем базовый город имеет номер N. Каждый тур (один из возможных маршрутов) однозначно соответствует перестановке целых чисел $1, 2, ..N - 1$. Для каждой перестановки строим тур и определяем его стоимость. Обрабатывая все перестановки запоминаем маршрут, который имеет на текущий момент самую низкую стоимость. Если находится маршрут с меньшей стоимостью, то все дальнейшие сравнения осуществляем с ним.

Примеры трудных задач



Алгоритм является факториальным, с оценкой $O(n!)$. В задаче требуется найти $(N-1)!$ перестановок целых чисел.

Определение

Построим таблицу, иллюстрирующую вычислительную сложность алгоритма, предполагая, что производительность компьютера 1GFLOPS (1000000000 op/s).

Кол-во городов, N	Кол-во туров	T,c	T,дн	T,лет
2	1	2e-08	$\ll 1$	$\ll 1$
3	5	6e-08	$\ll 1$	$\ll 1$
4	23	2e-07	$\ll 1$	$\ll 1$
10	4e+06	4e-02	$\ll 1$	$\ll 1$
15	13e+12	1e+04	0.15	$\ll 1$
18	6e+15	6e+07	740	20
19	1e+17	1e+09	14000	390
20	2e+18	2e+10	280000	770

При количестве городов, равным 50, гипотетический компьютер с производительностью в 10^{18} оп/с будет решать задачу комивояжера 10^{42} с, или $3 * 10^{34}$ лет, что существенно больше времени существования Галактики.

Список литературы

Список литературы i

- Кормен Т.,Лейзерсон Ч., Ривест Р.
  Алгоритмы: построение и анализ
  МЦНМО, Москва, 2000
- **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.**
  2-е изд. — М.: «Вильямс», 2006
- **Википедия**
  Алгоритм
  http://ru.wikipedia.org/wiki/Алгоритм
- **Википедия**
  Список алгоритмов
  http://ru.wikipedia.org/wiki/Список_алгоритмов
- **Традиция**
  Задача коммивояжёра
  http://traditio.ru/wiki/Задача

Список литературы ii

- **Википедия**
  NP-полная задача
  http://ru.wikipedia.org/wiki/NP-полная
- **Серджвик Р.**
  Фундаментальные алгоритмы на C++. Части 1-4
  Diasoft,2001
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
  СПб.: ДиаСофтЮП, 2003
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Алгоритмы на графах
  СПб.: ДиаСофтЮП, 2003
- **Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.**
  Издательский дом «Вильямс», 2000

Список литературы iii

- **Кнут Д.**
  Искусство программирования, том 1. Основные алгоритмы
  3-е изд. — М.: «Вильямс», 2006
- **Кнут Д.**
  Искусство программирования, том 2. Получисленные методы
  3-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 3. Сортировка и поиск
  2-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений
  М.: «Вильямс», 2007

Список литературы iv

- **Кнут Д.**
  Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации
  М.: «Вильямс», 2007