

Методы анализа алгоритмов. Классы сложности Р и NP

Выполнил:
студент 2 курса магистратуры
05182м группы
Василевский А.В.

Алгоритм

- **Алгоритм** – любая корректно поставленная вычислительная процедура (отображения входа на выход)
- **Алгоритм** – инструмент решения **вычислительной задачи**
- **Экземпляр задачи** – задача + корректные исходные (входные) данные

Свойства алгоритмов

- **Дискретность.** Алгоритм – последовательность (конечных по времени выполнения) шагов.
- **Детерминированность.** Следующий шаг однозначно определяется состоянием системы.
- **Понятность** для исполнителя.
- **Завершаемость.** Конечное время работы.
- **Универсальность.** Применимость к различным экземплярам задачи.
- **Результативность** (корректность). (Корректный) результат работы.

Классификация алгоритмов

- Детерминированные/стохастические.
- Точные/приближенные.
- Корректные/частично корректные/некорректные.
- Последовательные/параллельные.
- По временной сложности ($O(1)$, $O(n)$, $O(\dots)$).
- По объему требуемых ресурсов (памяти) ($O(1)$, ...).

Методы анализа корректности

- **Инварианты** (циклов) – («ручной») анализ (циклических) алгоритмов на основе «урезанной» (конечной) математической индукции.
- **Полный перебор** – проверка алгоритма на всех возможных входных данных (тестирование методом черного ящика).
- Статический анализ (**data flow analysis**) – «умный» перебор, анализ возможных **классов** входных и выходных данных (напр. в компиляторах)
- Статический анализ (**формальная верификация**) – (полу)автоматический анализ корректности на основе инвариантов методами автоматического вывода и автоматического доказательства теорем (верификатор KeY для языка Java; Z3 Prover; Coq, Idris, etc.)

Анализ сложности

- **Цель** – определение временной и «пространственной» **сложности**.
- Модели:
 - Машина Тьюринга (RW-лента ячеек; команды чтения, записи, смещения вправо/влево)
 - Недетерминированная машина Тьюринга
 - RAM-машина (RO и WO-ленты + регистровая память)

RAM-машина

- Абстракция реального компьютера:
 - Последовательность входных команд (программа)
 - Память с произвольным доступом (регистры)
 - Набор инструкций (переходы, int/float-арифметика)
- Можно считать, что все элементарные инструкции имеют одинаковое время выполнения $t \sim O(1)$.

Анализ сложности

- **Время работы** алгоритма – сумма квантов времени элементарных инструкций. Зависит от **входных данных** (не только от размера, хотя прежде всего от него).
- Различают **лучшее, худшее и среднее** (ожидаемое) время работы алгоритма (при том же размере данных). Часто последние два имеют одинаковый порядок.
- Ожидаемое время зависит от специфики задачи (напр. поиск в БД заведомо не / существующих данных)

Анализ сложности

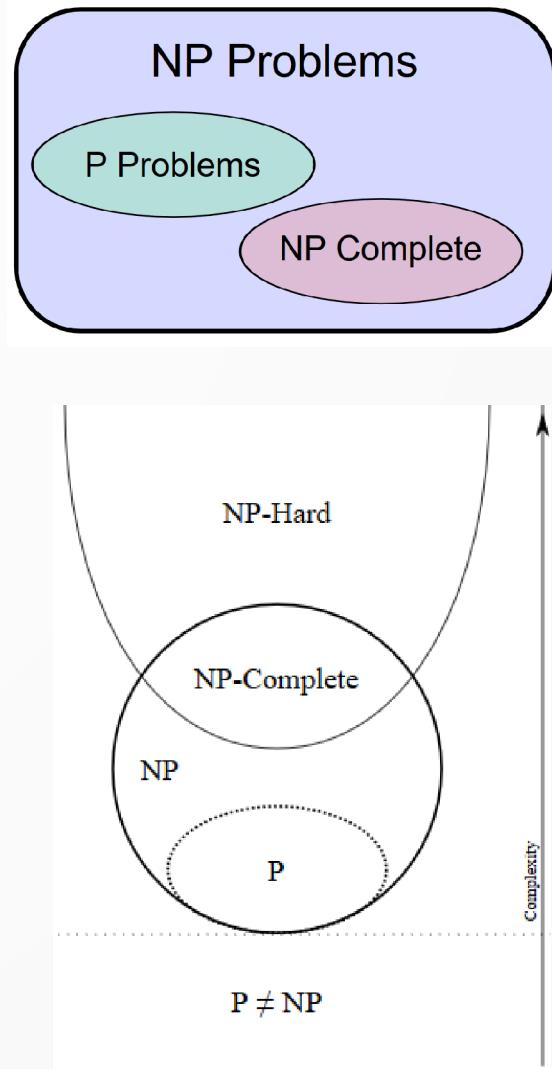
INSERTION-SORT(A)	<i>Стоимость</i>	<i>Повторы</i>
1 for $j = 2$ to $A.length$	c_1	n
2 $key = A[j]$	c_2	$n - 1$
3 // Вставка $A[j]$ в отсортированную последовательность $A[1..j - 1]$.	0	$n - 1$
4 $i = j - 1$	c_4	$n - 1$
5 while $i > 0$ и $A[i] > key$	c_5	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	c_6	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	c_8	$n - 1$

Лучший случай: $t_j = 1 \rightarrow$ сложность линейная

Худший случай: $t_j = j \rightarrow$ сложность квадратичная

Классы задач

- **P** – polynomial. Класс проблем, **разрешимых** за полиномиальное время.
- **NP** – nondeterministic P. Класс проблем, для которых **известный ответ** («сертификат» решения) **можно проверить** за полиномиальное время (сводится к другой задаче класса P **с ответом «да»/«нет»**).
- **NPC** – NP complete. Задача **перебора с ответом «да»/«нет»** – «композиция» NP-задач.
- **NPH** – NP hard. Задачи **перебора с произвольным ответом**. Самые сложные задачи (оптимизации).



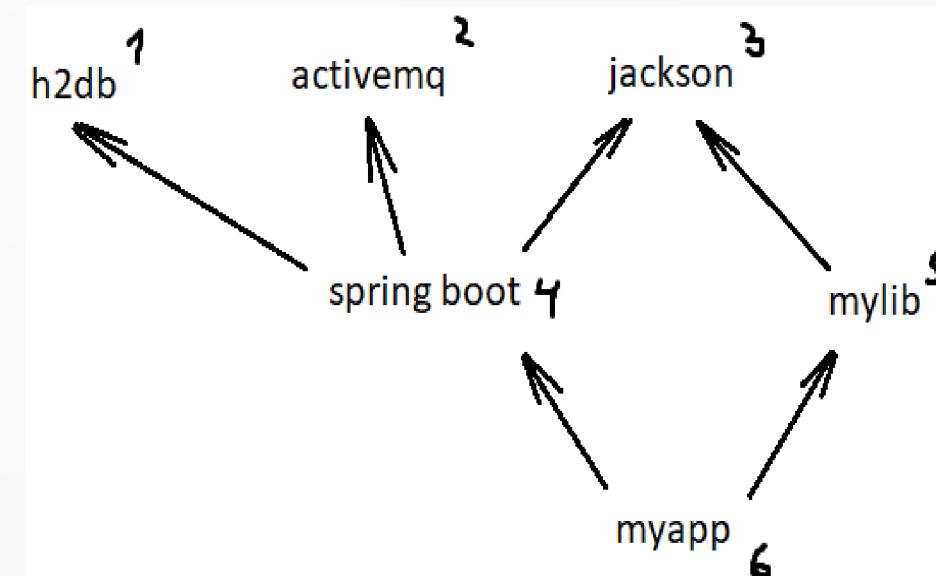
NP is Not in P

Пример 0 (sort)

- Задача: сортировка последовательности чисел.
 - Проверить, что данные числа отсортированы можно за $O(n)$, следовательно проблема сортировки – **NP**.
 - **Существуют** алгоритмы, решающие задачу за полиномиальное время, значит задача принадлежит классу **P**.

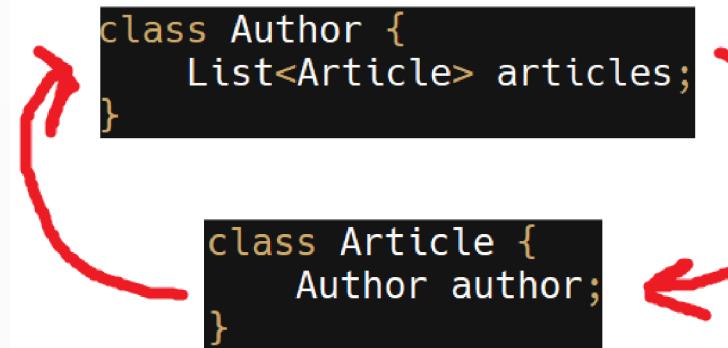
Пример 1 (toposort)

- Задача: определить порядок загрузки библиотек, если даны зависимости между ними.
 - Вариация задачи топологической сортировки вершин направленного ациклического графа (DAG)
 - Сложность алгоритма (Кана / Тарьяна) – $O(V+E)$ – полином 1-й степени – задача класса P.
 - Бонус:
 - Определяет **наличие** цикла(ов) в графе



Пример 2 (feedback arc set)

- Задача: определить порядок компиляции файлов, если даны зависимости между ними, причем **они могут быть циклическими**.
 - Реальный порядок обработки циклических зависимостей не так важен. Решение задачи – **размыкание** циклов (удаление ребер графа, пока не получится DAG). Проблема – FAS (\uparrow).
 - Задача **NP Complete**: (след. сама задача – минимум **NP**)
 - **Тест** «DAG ли это после удаления набора ребер» – **P**.
 - **Поиск** этого набора – **NPC** (перебор всех вариантов).
- Вариации: (и проверка каждого из них **тестом**)
 - размыкание deadlock-ов с минимальным «ущербом»
 - dependency injection с максимизацией быстродействия и т.д.



Проблема P \neq NP

- «Если можно быстро **проверить**, является ли X решением задачи Y, можно ли также быстро **найти** это решение?»
 - $\text{Prime1} * \text{Prime2} = \mathbf{X}$ – легко
 - $\text{Prime1} * \mathbf{X} = \mathbf{Y}$ – сложно, но легко проверить конкретный X.
- До сих пор не решена и актуальна. В случае $P = NP$ многие сложные задачи (перебора!) можно решить за полиномиальное время! Последствия катастрофические.

Типичные Р-проблемы

- **Сортировки и поиск**
- Математические операции [`sizeof(N) < inf`]
- **Обходы графов** (garbage collection, topological sort)
- Поиск **кратчайшего** пути ↑
- Regexp Matching [!без обратных ссылок!]

Некоторые NP-проблемы

- **Факторизация целых чисел.** Сертификат – набор (простых) множителей.
- **Дискретный логарифм** ($b^x = a$, $x = ?$). Сертификат – x .
- **(Изо)морфизмы** (графов, групп). Сертификат – функция-морфизм.

Типичные NPC-проблемы

- Boolean Satisfability Problem (**SAT**)
 - Обращается ли булева функция $f(x, \dots)$ в true хотя бы раз? Сертификат – само решение.
- Feedback Vertex/Arc Set (**FVS/FAS**, см. \leftarrow)
 - Найти **любой** набор вершин/ребер, превращающий произвольный граф в DAG. Сертификат – сам набор.
- Всевозможные **раскраски** графа
 - Раскрасить граф **любым из способов** так, чтобы смежные вершины/ребра имели разные цвета ($N < \inf$ цветов). Сертификат – сама раскраска.
- **Longest Path Problem**
 - Найти путь из вершины A в вершину B, **состоящий не менее чем из N вершин**. Сертификат – сам путь.

Типичные NPC-проблемы

- Всевозможные задачи о **клике** (полносвязном подграфе)
 - Найти клику **с не менее чем N вершинами**. Сертификат – сама клика.
- Задачи **упаковки** (контейнеры/«ранец»/«рюкзак»)
 - Имея набор контейнеров и набор предметов, упаковать их так, чтобы занять **максимум N контейнеров**. Сертификат – такая упаковка.
- **Сумма/разбиение подмножеств**
 - Найти подмножество множества, элементы которого суммируются в 0. Сертификат – такое подмножество.

Типичные NPH-проблемы

- Feedback Vertex/Arc Set (**FVS/FAS**, см. ←)
 - Найти **минимальный** набор вершин/ребер, превращающий произвольный граф в DAG...
- Всевозможные **раскраски** графа
 - Раскрасить граф **минимальным числом цветов** так, чтобы смежные вершины/ребра имели разные цвета ($N < \inf$ цветов)...
- **Longest Path Problem**
 - Найти **самый длинный** путь из вершины A в вершину B...
- Regexp Matching [**!с обратными ссылками!**]

Задачи (1+2)

- 1. Разработать алгоритм, проверяющий, является ли переданное в него число N простым.
 - **Базовая идея** [точно]: перебрать все делители $n < N$.
 - **Оптимизации** [точно]: методами задачи 2.
 - **Специальные знания** [с контролируемой ошибкой]: вероятностные методы на основе теории чисел. [не рассматриваются]
- 2. Дано число N . Написать функцию, возвращающую список простых чисел до N включительно.
 - **Базовая идея**: как в 1, но перебирать ранее найденные простые делители.
 - **Решето**: как в 1, но еще и вычеркивать все кратные найденным.
 - **Оптимизации** [решето + специальные знания]: уменьшение пространства поиска методами теории чисел [только простейшие] и простыми эвристиками.

Задачи (1+2)

- 1. Оптимизации:
 - Исключить все четные числа сразу
 - Заметить, что перебирать делители достаточно до \sqrt{N}
 - *Оптимизации памяти делением на подрешета*
- 2. Простейшая реализация в лоб
 -

```
1 std::vector<int> P(N); P.push_back(2);
2 for (int n = 3; n < N; n += 2) {
3     bool prime = true;
4     for (int p = 0; P[p]*P[p] <= n; ++p) {
5         if ((n % P[p]) == 0) { prime = false; break; }
6     }
7     if (prime) P.push_back(n);
8 }
```

Задачи (1+2)

- 3. Простейшее решето (Эратосфена):

```
10 std::vector<bool> S(N, true);
11 for (int n = 4; n < N; n += 2) S[n] = false;
12 for (int n = 3; n*n < N; n += 2)
13     if (S[n]) // если n не вычеркнуто
14         for (int m = n*n; m < N; m += n) S[m] = false;
```

- Асимптотика первой реализации: $O(N \sqrt{N})$ [точнее еще $\ln(\ln(N))$ – из теории чисел]
- Асимптотика второй реализации:
 - Вычеркиваем $N/2, N/3, N/5, \dots, N/p, \dots, p < \sqrt{N}$ чисел за итерацию $\Rightarrow \sum(1/x) \sim \ln(x) + c$
 - Асимптотика $O(N \ln \sqrt{N})$ [точнее $O(N \ln \ln \sqrt{N})$ – из теории чисел]

Литература

- [1] Томас Х. Кормен и др. Часть 1. Основы // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — 1296 с.
- [2] Томас Х. Кормен и др. Глава 34. NP-полнота // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — 1296 с.
- [3] Википедия, Wikipedia
- [4] Хабрахабр: «Доказательство некорректности алгоритма сортировки Android, Java и Python»
- [5] Alfred V. Aho et.al. Compilers: principles, techniques, and tools, 2nd edition – Pearson, «Addison Wesley», 2007 – 1009 p.