Модуль 1. Полный перебор

Лекция 1

Введение. Классы сложности. Генерация бинарных строк.

План

- Введение
  - Виды задач
  - Классы сложности
  - Сводимость
  - Класс NP
  - NP-трудность, NP-полнота
  - Подходы к решению NP-трудных задач
- Полный перебор вариантов решения
  - Генерация бинарных слов
  - Задание 1: Рюкзак (0-1 Knapsack)

Виды задач

- Распознавательные
- Вычислительные
- Оптимизационные

Виды задач

Распознавательная задача

- Формально: задача распознавания принадлежности входного слова заданному языку

Дано: $L \subseteq A^*, \quad x \in A^*$

Найти: верно ли, что $x \in L$?

- На практике: задача, в которой надо вернуть ответ „истина“ или „ложь“.

Виды задач

Вычислительная задача

- Множество входов: X
- Множество допустимых решений: S(x)
- Задача: для заданного $x \in X$ *найти* $s \in S(x)$

Виды задач

Оптимизационная задача

- Множество входов: X
- Множество допустимых решений: S(x)
- Функция стоимости решения: $c: S(x) \to R^+$
- Задача: для заданного $x \in X$ найти допустимое решение $s^* \in S(x)$:

  $\forall s \in S(x)$

  $c(s^*) \geq c(s)$ // максимизация

  $c(s^*) \leq c(s)$ // минимизация

- Обозначение: $s^* = opt(x)$; $c(s^*) = c^*(x)$

Сложность алгоритма

Сложность алгоритма = количество ресурсов, необходимых алгоритму для решения данного экземпляра задачи.

Наиболее важный ресурс — время. Измеряется количеством операций, выполняемых алгоритмом при решении задачи.

Сложность представляется в виде $O(f(n))$, где n — *размер задачи*, то есть количество бит в представлении исходных данных.

Для упрощения в качестве n часто берут более наглядный параметр: количество предметов, количество вершин/дуг в графе и т.п.

Сложность алгоритма

Эффективный алгоритм: имеет полиномиальную сложность O(n^k).

Экспоненциальный алгоритм: имеет сложность O(2^n) или более высокую, например O(n^n).

n	O(n)	O(2^n)
50	1,00 сек	1 сек
51	1,02 сек	2 сек
52	1,04 сек	4 сек
60	1,20 сек	17 мин
70	1,40 сек	12 суток
80	1,60 сек	34 года
90	1,70 сек	~ 35 000 лет

Сложность задачи

Для заданной задачи могут существовать алгоритмы разной сложности.

? Сложность задачи = сложность самого быстрого алгоритма, решающего эту задачу.

Теорема Блюма об ускорении: существует задача, для которой любой решающий её алгоритм можно экспоненциально ускорить.

Классы сложности

Класс сложности = множество задач, для каждой из которых существует решающий её алгоритм, имеющий указанную сложность.

P = множество задач, решаемых за полиномиальное время.

К сожалению, для многих практически важных задач пока не известны полиномиальные алгоритмы. Это те самые ***вычислительно сложные задачи***.

NP-трудные задачи

- Класс NP (Non-deterministic Polynomial)
  - Распознавательные задачи, решаемые недетерминированным алгоритмом за полиномиальное время.
  - Задачи, для которых решение может быть проверено (детерминированным алгоритмом) за полиномиальное время при наличии *сертификата*.

NP-трудные задачи

- Полиномиальная сводимость

Задача *R* полиномиально сводится к задаче *Q* <=> существует алгоритм A_R , обращающийся к алгоритму A_Q (алгоритм для *Q*), и решающий задачу *R* за полиномиальное время <u>без учёта времени работы</u> A_Q.

NP-трудные задачи

- NP-трудные задачи
  - Задача называется NP-трудной, если к ней полиномиально сводится любая задача $Q \in NP$.
  - Задача NP-полна, если она NP-трудна и принадлежит классу NP.

NP-трудные задачи

- $P \subseteq NP$. $P = NP$????
- Все NP-полные задачи полиномиально эквивалентны, т. е. полиномиально сводятся друг к другу.
- Ни для одной NP-полной задачи не известен полиномиальный алгоритм. И маловероятно, что будет обнаружен в обозримом будущем.

Что делать?

- Решать долго (за экспоненциальное время)
- Пытаться сократить время решения для входов, ожидаемых на практике
  - Полиномиальные в среднем алгоритмы
  - *Параметризованные* алгоритмы
- Решать приближённо
  - С гарантированной *оценкой точности*
  - Без гарантий, но как правило достаточно точно
  - С некоторой *вероятностью* ошибки

Метод полного перебора

- Полный перебор (Brute Force)
  - Последовательно генерировать все возможные решения
  - Для каждого сгенерированного решения x выполнять проверку на допустимость / оптимальность: *Process*(x)

Бинарные строки

Вход: натуральное число n.

Задача: последовательно сгенерировать все бинарные (битовые) строки длины n.

Бинарные строки

1. Массив A[1..*n*]
2. A := [0, 0, …, 0]
3. *ProcessBinaryStrings*(*A*, *n*)

Бинарные строки

ProcessBinaryStrings(*A*, *k*)

```
if k = 0 then Process(A)
else
    A[k] := 0;
    ProcessBinaryStrings(A, k-1);
    A[k] := 1;
    ProcessBinaryStrings(A, k-1);
```

Бинарные строки

Оценим качества алгоритма:

- Корректность: генерирует все требуемые комбинации и только их
- Оптимальность: временная сложность пропорциональна количеству комбинаций

Бинарные строки

Теорема. Алгоритм ProcessBinaryStrings корректен и оптимален

Доказательство

Корректность. Покажем, что для каждого $k \geq 1$ алгоритм вызывает *Process*(*A*) один раз для каждой бинарной строки.

Индукция по *k*. Для $k = 0$ справедливо. Допустим, что при вызове *ProcessBinaryStrings*(*A*, *k*-1) обрабатываются все бинарные строки длины (*k*-1).

Далее — анализируем, структуру алгоритма.

Бинарные строки

<u>Оптимальность</u>.

Пусть $T(n)$ — временная сложность алгоритма.

Тогда: $T(1) = c = \mathrm{const}$, $T(n) = 2T(n-1)+d$.

Получаем: $$T(n) = (c+d)2^{n-1} - d.$$

Таким образом, $T(n) = O(2^n)$.

То есть, алгоритм оптимален.

Задание 1

Задача «Рюкзак» (0-1 Рюкзак, Knapsack). 3 балла.

Дано:

- n предметов, для каждого задан вес w_i и стоимость c_i.
- предельный допустимый суммарный вес b.

Найти: $I \subseteq \{1,\ldots,n\}$, при котором $$c(I) = \sum_{i \in I} c_i \to max$$

при условии, что $$\sum_{i \in I} w_i \leq b$$

Задание 1

Тестовые наборы:
http://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html

Datasets:

P01 is a set of 10 weights and profits for a knapsack of capacity 165.

- p01_c.txt, the knapsack capacity.
- p01_w.txt, the weights of the objects.
- p01_p.txt, the profits of each object.
- p01_s.txt, the optimal selection of weights.

P02 is a set of 5 weights and profits for a knapsack of capacity 26.

- p02_c.txt, the knapsack capacity.
- p02_w.txt, the weights of the objects.
- p02_p.txt, the profits of each object.
- p02_s.txt, the optimal selection of weights.