Комбинаторные Алгоритмы

Ильнур Шугаепов*, Михаил Чернявский†, Александр Соколов‡

Весна, 2016

*ilnur.shug@gmail.com, СПбАУ НОЦНТ РАН
†chemike47@gmail.com, Университет ИТМО
‡sookooliki@outlook.com, Университет ИТМО

Содержание

1 Лабораторные работы

Замечание 1.1. Ознакомьтесь с правилами выполнения лабораторных работ (см. раздел 2) и правилами сдачи (см. раздел 3).

Замечание 1.2. Прежде чем предпринять попытку сдать очередную лабораторную работу, убедитесь в том, что вы знаете весь теоретический минимум (см. раздел 5), связанный с данной лабораторной работой.

1.1 Сортировки за $\mathcal{O}(n^2)$

`Deadline: 5.03`

1. Пузырьковая;
2. Вставками (простыми, бинарными);
3. Выбором.

1.2 Сортировки за $\mathcal{O}(n \log n)$

`Deadline: 2.04`

1. Быстрая;
2. Кучей (двоичная, $d-$heap, leftist);
3. Слиянием (простое, многопутевое, выбор из дерева).

1.3 Сортировки за $\mathcal{O}(n)$

`Deadline: 16.04`

1. Подсчетом;
2. Поразрядная;
3. Блочная.

1.4 Поиск

`Deadline: 7.05`

1. Бинарный поиск;
2. Поиск в дереве (простое бинарное дерево поиска, АВЛ, Splay).

1.5 Хеш-функции и хеш-таблицы

`Deadline: 28.05`

1. Динамическая хеш-таблица (разрешение коллизий методом цепочек);
2. Открытая адресация.

2 Правила выполнения ЛР

2.1 Сортировка

(см. Лабораторная Работа 1.1, 1.2, 1.3)

1. Для выполнения лабораторной работы необходимо сгенерировать тестовые файлы (используя генераторы случайных чисел), содержащие целые числа, в количестве от 2^6 до 2^{20} (можно и больше), при этом количество элементов в следующем файле в два раза больше чем в предыдущем.

   Замечание 2.1. Рекомендуется создать несколько наборов тестовых данных. *Например*: с маленьким количеством различных элементов, с большим / маленьким количеством инверсий и.т.д.

2. Реализовать алгоритмы используя один из следующих языков программирования: `C++`, `C#`, `C`, `Python`.

3. Для каждого тестового файла из набора выполнить сортировку данных. Вычислить среднее время сортировки по одному файлу.

4. Построить график зависимости времени сортировки от количества элементов в файле.

5. Проверить существование исходных данных таких, что время работы алгоритма сильно увеличивается по сравнению со временем работы алгоритма на случайных данных.

6. Выполнить сравнение алгоритмов.

Содержание отчета

1. Исходный код генератора исходных данных;

2. Программный код (можно не вставлять весь, если есть возможность показать с машины);

3. Объяснение (анализ) полученных результатов;

4. Сравнительный анализ алгоритмов.

Замечание 2.2. Отчет может не содержать математических основ и описания алгоритмов, так как Ваше знание теор. мина (см. раздел 5) будет проверено в процессе сдачи ЛР.

2.2 Поиск

(см. Лабораторная Работа 1.4, 1.5)

1. Для выполнения лабораторной работы необходимо сгенерировать тестовые файлы (используя генераторы случайных чисел), содержащие целые числа, в количестве от 2^6 до 2^{20} (можно и больше), при этом количество элементов в следующем файле в два раза больше чем в предыдущем.

2. Для каждого тестового файла из набора выполнить следующие действия:

   (a) Поиск элементов, которые гарантированно имеются во входных данных. Выполнить поиск каждого элемента. Вычислить среднее время поиска одного элемента.

   (b) Поиск элементов, которые гарантированно не имеются в исходных данных. Вычислить среднее время поиска одного элемента.

3. Построить график зависимости времени поиска одного элемента от количества элементов в файле.

4. Проверить существование исходных данных таких, что время работы алгоритма сильно увеличивается по сравнению со временем работы алгоритма на случайных данных.

5. Выполнить сравнение алгоритмов:

   (a) Сравнить графики;

   (b) Сравнить асимптотические оценки.

Содержание отчета (см. раздел 2.1)

3 Правила сдачи ЛР

1. Сдавая конкретную ЛР, Вы должны до крайнего срока (включительно) получить зачет по теоретическому минимуму (см. раздел 5);

2. После того как вы получили зачет по теормину, преподаватель выдает Вам несколько задач на защиту (количество и сложность задач по усмотрению преподавателя).
   По задачам на защиту Вы должны получить зачет не позднее крайнего срока выполнения следующей за данной лабораторной работой;

3. Решения задач должны быть оформлены в `.tex` (см. пример оформления решения в разделе 4). Для этого можно воспользоваться, например, `sharelatex.com`.

4 Примеры оформления решений задач

Данный раздел предназначен для того, чтобы продемонстрировать ряд примеров по оформлению решений задач.
Темы задач не имеют отношения к курсу, поэтому нет необходимости в детальном изучении самих решений.
Исходный `.tex` файл с примерами доступен по ссылке `Example`. Данный файл позволит Вам быстро оформить Ваше решение в виде `.tex` файла, так как содержит все необходимые элементы верстки.
Настоятельно рекомендуется изучить файл по ссылке.

4.1 Пример 1

Условие Пусть дано дерево $T = \langle V, E \rangle$. Для каждого ребра вычислить, сколько путей проходит через него. Придумать алгоритм, который работает за линейное время.

Решение Рассмотрим дерево $T = \langle V, E \rangle$. Подвесим T за произвольную вершину $r \in V$. Будем решать задачу методом динамического программирования.
Пусть

$$d(v) = |\{\pi = \langle v_1 = v, v_2, \dots, v_k \rangle \colon \forall i = \overline{2, k}(v_{i-1} = p(v_i)) \wedge v_k \in T_v \setminus \{v\}\}|.$$

То есть $d(v)$- количество путей в поддереве T_v от вершины v к ее потомкам. Пусть $\mathcal{C} = \{c_i \colon p(c_i) = v\}_{i=1}^{k}$ - множество дочерних узлов вершины v. Теперь покажем, как $d(v)$ можно получить через $d(c_i)$:

$$d(v) = \sum_{i=1}^{k}(d(c_i) + 1) = k + \sum_{c \in \mathcal{C}} d(c).$$

Нас интерисует $d(v, c_i)$ - количество путей проходящих через ребро (v, c_i):

$$d(v, c_i) = \sum_{\substack{j=1,\\ j \neq i}}^{k}(d(c_i) \cdot d(c_j) + 1) = d(c_i)\sum_{j=1}^{k} d(c_j) + (k-1) - d^2(c_i).$$

Замечание 4.1. Согласно данному выше определению, $d(v, c_i)$ - описывает количество путей, проходящих через ребро (v, c_i) при условии, что пути находятся в поддереве T_v. Следовательно, $d(v, c_i)$ дает ответ на задачу *только* если $v = r$.

Пусть r - корень, выполним переподвешивание дерева T за c_i, тогда произойдут следующие изменения:

$$\begin{aligned} d(r) &\leftarrow d(r) - d(c_i) - 1, \\ d(c_i) &\leftarrow d(c_i) + d(r) + 1. \end{aligned}$$

Ясно, что переподвешивание такого вида мы можем выполнить за $\mathcal{O}(1)$.
Пусть мы уже посчитали для все вершин $d(v)$, это, очевидно. можно сделать с

помощью DFS за $\mathcal{O}(|V|)$. Кроме того, в процессе вычисления мы также посчитали $d_\Sigma(v) = \sum\limits_{c \in \mathcal{C}} d(c)$.

Algorithm 4.1.1

procedure $\textsc{hang}(r, c)$ ▷ r – текущий корень, c – новый
  $d_\Sigma(r) \leftarrow d_\Sigma(r) - d(c)$
  $d(r) \leftarrow d(r) - d(c) - 1$
  $d_\Sigma(c) \leftarrow d_\Sigma(c) + d(r)$
  $d(c) \leftarrow d(c) + d(r) + 1$
procedure $\textsc{dp}(r)$
  for $c \in \mathcal{C}(r)$ **do**
    $d(r, c) \leftarrow d(c) d_\Sigma(r) + |\mathcal{C}| - 1 - d^2(c)$
  for $c \in \mathcal{C}(r)$ **do**
    $\textsc{hang}(r, c)$ ▷ подвешиваем за c
    $\textsc{dp}(c)$
    $\textsc{hang}(c, r)$ ▷ подвешиваем за r

Так как мы посетим каждое ребро, то переподвешиваний будет $\mathcal{O}(|E|) = \mathcal{O}(|V|)$, каждое за $\mathcal{O}(1)$, следовательно, временная сложность предложенного алгоритма $\mathcal{O}(|V|)$.

4.2 Пример 2

Условие Дано дерево $T = \langle V, E \rangle$. Выбрать на нем 2 различные вершины так, чтобы сумма расстояний до ближайшей из выбранных вершин была минимальна (т.е., если мы назовем выбранные вершины a и b, то нужно минимизировать величину $\sum_{v \in V} \min(dist(v, a), dist(v, b))$). Балл за задачу здесь зависит от эффективности предложенного алгоритма.

Решение Пусть $c_1, c_2 \in V \colon c_1 \neq c_2 \wedge \sum\limits_{v \in V} \min\{d(c_1, v), d(c_2, v)\} \to \min$. Тогда ясно, что существует разбиение вершин $V = V_1 \cup V_2 \colon V_1 \cap V_2 = \emptyset \wedge \forall v \in V_1 (d(c_1, v) < d(c_2, v))$.
Рассмотрим разрез $C(V_1, V_2) = \{e = (v, u) \in E \colon v \in V_1 \wedge u \in V_2\}$.

Лемма 4.2.1. $|C(V_1, V_2)| = 1$.

Доказательство. Предположим, что $|C(V_1, V_2)| > 1$, то есть $(v_1, u_1), (v_2, u_2) \in C(V_1, V_2) \colon v_1 \neq v_2$. Пусть $c_1 \rightsquigarrow v_1 u_1 \rightsquigarrow c_2$ - простой путь. Заметим, что путь $c_1 \rightsquigarrow v_2$ не должен содержать вершину c_2, так как в противном случае $d(c_1, v_2) > d(c_2, v_2) \Rightarrow v_2 \in V_2$. Получаем, что между c_1 и c_2 есть не менее двух путей, отичающихся хотябы в одной вершине, что противоречит тому, что в дереве между любой парой вершин существует единственный простой путь. □

Таким образом, мы свели исходную задачу к следующей:

Задача 4.1 (*1-median problem*)**.**
В дереве $T = \langle V, E\rangle$ найти $c \in V : f(c) := \sum\limits_{v\in V} d(c, v) \to \min$.

Будем решать по аналогии с задачей 4.1. Пусть T подвешено за r, $k(v)$- количество путей из v в поддереве T_v:

$$f(r) = \sum_{c\in\mathcal{C}(r)} (1 + f(c) + k(c)) = k(r) + \sum_{c\in\mathcal{C}(r)} f(c).$$

Мы уже знаем, что $k(r)$ можно пересчитать за $\mathcal{O}(1)$ при переподвешивании T за $c \in \mathcal{C}(r)$, не трудно понять, что $f(r)$ мы также можем пересчитать за $\mathcal{O}(1)$.

Algorithm 4.2.2

function 1-MEDIAN(T)
  $r \leftarrow root(T)$
  $ans \leftarrow r$
  $f \leftarrow f(ans)$
  for $c \in \mathcal{C}(r)$ **do**
    HANG(r, c) ▷ подвешиваем за c
    $(r', f') \leftarrow$1-MEDIAN(T_c)
    if $f' < f$ **then**
      $f \leftarrow f'$
      $ans \leftarrow r'$
    HANG(c, r) ▷ подвешиваем за r
  return (ans, f)

С помощью алгоритма 4.2.2, исходную задачу мы можем решить следующим образом:

Algorithm 4.2.3

function 2-MEDIAN(T)
  $(c_1, c_2) \leftarrow (-1, -1)$
  $d \leftarrow \infty$
  for $(v, u) \in E$ **do**
    $(d_v, a) \leftarrow$ 1-MEDIAN(T_v)
    $(d_u, b) \leftarrow$ 1-MEDIAN(T_u)
    if $d_v + d_u < d$ **then**
      $d \leftarrow d_v + d_u$
      $(c_1, c_2) \leftarrow (a, b)$
  return (c_1, c_2)

Сложность алгоритма, очевидно, $\mathcal{O}(|V|^2)$.

5 Теоретический минимум

Основные структуры данных: очередь, стек, дек, моделирование очередии с помощью двух стеков, односвязный список, двусвязный список; Асимптотические обозначения и формальные определения: $\mathcal{O}, \Theta, \Omega, o, \omega$; Временная и пространственная сложность алгоритма: в худшем случае, в среднем (см. [3]); Вычисление чисел Фибоначчи: экспоненциальный рекурсивный алгоритм, полиномиальный алгоритм, более детальный анализ; Количество перестановок: с повторениями и без; Количество сочетаний: с повторениями и без, рекуррентные соотношения и их комбинаторный смысл.

5.1 Сортировки за $\mathcal{O}(n^2)$

Для каждого алгоритма: количесто сравнений и обменов, устойчивость; перестановки (симметрическая группа), инверсии, таблица перестановок, обратная перестановка; восстановление перестановки по таблице инверсий за $\mathcal{O}(n \log n)$; построение таблицы инверсий за $\mathcal{O}(n \log n)$; матричное представление перестановок.

5.1.1 Пузырьковая

5.1.2 Вставками

Простые вставки; Бинарные вставки.

5.1.3 Выбором

Простой выбор; Выбор из дерева.

5.2 Сортировки за $\mathcal{O}(n \log n)$

Нижняя оценка $\Omega(n \log n)$ для сортировки сравнениями.

5.2.1 Быстрая

Различные варианты процедуры partition; Доказательство $T_{QS}(n) = \mathcal{O}(n \log n)$ в среднем; Доказательство $T_{QS}(n) = \mathcal{O}(n^2)$ в худшем случае; IntroSort; K - ая порядковая статистика, поиск за $\mathcal{O}(n)$ в среднем, одновременный поиск min и max; QuickSort3, массивы с маленьким количеством различных элементов; Глубина дерева рекурсии; tail-recursion elimination.

5.2.2 Кучей

Построение пирамиды за $\mathcal{O}(n)$; Очереди с приоритетами; Частичная сортировка; $d-$heap; Построение худшего случая для пирамидальной сортировки: количество сравнений; Leftist heap; Устойчивость; Коды Хаффмена.

5.2.3 Слиянием

Многопутевое слияние, выбор из дерева; Разделяй и властвуй: алгоритм Карацубы, алгоритм Штрассена; Рекуррентные соотношения: основная теорема, методы решения; Устойчивость; Внешняя сортировка.

5.3 Сортировки за $\mathcal{O}(n)$

5.3.1 Подсчетом

Устойчивость.

5.3.2 Поразрядная

5.3.3 Блочная

Доказательство $T_{BS}(n) = \mathcal{O}(n)$ в среднем.

5.4 Поиск

5.4.1 Бинарный поиск

Тернарный поиск; d - поиск: анализ времени поиска.

5.4.2 AVL tree

Количество балансировок в худшем случае при вставке, удалении; операция merge для АВЛ деревьев; количество узлов в минимальном АВЛ дереве; Сохранение свойств при поворотах.

5.4.3 Splay tree

Амортизационный анализ: групповой анализ, бухгалтерский учет, метод потенциалов, динамический массив; доказательство выполнения n операций за $\mathcal{O}(n \log n)$; операция splay за $\mathcal{O}(\log n)$; реализация основных операций через операцию splay.

5.5 Хеш-функции и хеш-таблицы

Хеш-функции: метод деления, метод умножения, универсальное хеширование*; Хеш-таблицы: разрешение коллизий с помощью цепочек, анализ хеширования с цепочками, рехеширование; Открытая адресация: линейное исследование, квадратичное исследование, двойное хеширование.

Список литературы

[1] Т.Кормен, Ч.Лейзерсон, Р.Ривест. Алгоритмы: построение и анализ. Москва: Издательство МЦНМО, 2000.

[2] Д.Кнут. Искусство программирования. Т. 3. Сортировка и поиск. М.—СПб.—Киев: ИД «Вильямс», 2001.

[3] С.А. Абрамов. Лекции о сложности алгоритмов. Москва: Издательство МЦНМО, 2009.

[4] А.Х. Шень. Программирование: теоремы и задачи. Москва: Издательство МЦНМО, 2012.

[5] С.Дасгупта, Х.Пападимитриу, У.Вазирани Алгоритмы. Москва: Издательство МЦНМО, 2014.

[6] M. Weiss. Data Structures and Algorithm Analysis in Java. 2012.