

2 курс ПМИ, осень 2024/25
Практика по алгоритмам #9
Архиваторы и хеширование
19 ноября

Собрано 18 ноября 2024 г. в 15:11

Содержание

1. Архиваторы и хеширование	1
2. Разбор задач практики	3
3. Домашнее задание	6
3.1. Обязательная часть	6
3.2. Дополнительная часть	6

Архиваторы и хеширование

1. Эффективная замена

Алгоритм сжатия **литературного текста** для $|\Sigma| = 127$: выберем слово w , дадим ему код 128, сожмём текст 7-битовым сжатим, каждую новую встречу w будем кодировать одним символом. Как передать само w ? Как выбрать оптимальное w на практике?

2. Обратное к MTF

Вспомните, что такое MTF и придумайте обратное преобразование. За $\mathcal{O}(n \cdot k)$. За $\mathcal{O}(n \log k)$.

```

1 void MTF(vector<int> &text):
2     int k = *max_element(all(text)) + 1; // размер алфавита
3     vector<int> ord(k); iota(all(ord), 0); // тождественная перестановка
4     for (size_t i = 0; i < text.size(); i++):
5         int c = text[i], p = ord-1[c]; // позиция c в алфавите
6         text[i] = p;
7         for (int i = p - 1; i >= 0; i--)
8             ord[i + 1] = ord[i];
9         ord[0] = c;

```

3. LZSS на практике

Представьте себе идеальный LZSS для 7-битного алфавита, который умеет находить $\max n$, для такого $\min p$, и которому нужно закодировать или очередной символ, или пару (n, p) .

Придумайте точный способ кодирования – какие биты чему отвечают, сколько бит хранить. Про оптимальность думайте на примере текста длины $\approx 1\text{mb}$, но помните, что ваш способ должен быть универсальным. Как можно применить сюда Хаффмана?

4. LZSS сжимает википедию

Как пожать 16 гигабайт входных данных и не умереть?

Что улучшилось/ухудшилось в решении предыдущей задачи?

5. Обратное преобразование Барроуза-Уилера

Дан последний столбец отсортированного массива циклических сдвигов строки. Восстановить первую строку массива.

a) $\mathcal{O}(n^3)$ (хоть как-нибудь)

b) $\mathcal{O}(n)$ CountSort-ом последнего столбца. На примере aabaxbх.

Перерыв

6. Универсальные и независимые

Семейство хэш-функций $\mathcal{H} = \{h : X \rightarrow Y\}$ называется универсальным, если:

$$\forall x_1, x_2 \in X, x_1 \neq x_2 : \Pr_{h \in \mathcal{H}} [h(x_1) = h(x_2)] \leq \frac{1}{|Y|}$$

Семейство хэш-функций $\mathcal{H} = \{h : X \rightarrow Y\}$ называется k -независимым, если

\forall различных $x_1, x_2, \dots, x_k \in X$, и \forall возможно совпадающих $y_1, y_2, \dots, y_k \in Y$ выполняется:

$$\Pr_{h \in \mathcal{H}} \left[\bigwedge_{i=1}^k h(x_i) = y_i \right] = \frac{1}{|Y|^k}$$

- Докажите, что любое 2-независимое семейство хэш-функций является универсальным.
- Докажите, что любое $k+1$ -независимое семейство хэш-функций является k -независимым.
- Приведите пример 2-независимого семейства для случая $|Y| \in \mathbb{P}$.

7. Словари и пары (хеши пар)

Пусть ваш язык программирования имеет встроенный ассоциативный массив $\text{int64} \rightarrow \text{int}$.
`unordered_map<int64_t,int> m; m[2] = 3;`

Пользуясь только функциональностью, описанной выше, добавьте в свой язык ассоциативный массив от пары $\langle u, v \rangle$ двух k -битных целых:

- $k = 32$. Простое решение.
- $k = 64$. Чем плохо решение $\langle u, v \rangle \rightarrow (u2^{64} + v) \bmod p$?
- $k = 64$. Решение универсальным семейством.
- $k = 64$. Решение полиномиальным хешированием.
- (*) $k = 64$. Вспомните про $\mathbb{F}_{2^{64}}$. Как улучшить предыдущее решение?

8. Вертикальная симметрия

- На плоскости даны n точек $x_i, y_i \in \mathbb{Z}$. Есть ли у них вертикальная ось симметрии? $\mathcal{O}(n)$.
- Точки добавляются/удаляются. После каждого запроса за $\mathcal{O}(\log n)$ проверять симметрию.

9. Упрощаем универсальное семейство

Задача: как можно упростить $(ax + b) \bmod p \bmod n$ для $p = n$? (большого простого n)

10. (*) Разбить строку на 2 почти палиндрома

Проверить за $\mathcal{O}(|s|)$, можно ли строку s представить в виде конкатенации двух почти палиндромов. Почти палиндром – строка, в которой можно заменить не более одного символа, чтобы получился палиндром.

11. (*) Валим хеш-таблицу с открытой адресацией

Задача: постройте контрпример к хеш-таблице с открытой адресацией и хеш функцией $x \rightarrow x \bmod n$ для 30-битного x и простого $n \approx 10^6$. А если n случайно?

12. (*) Почти совпадения — 2

Даны n словарных слов и m слов текста. Суммарная длина всех $n + m$ слов равна L . Скажем, что слова похожи, если можно из каждого удалить не более одной буквы, чтобы они стали равны. Найдите за $\mathcal{O}(L)$ для каждого слова текста кол-во похожих слов в словаре.

13. (*) Жесть

Есть множество X и 2-независимое семейство хеш-функций $\mathcal{H} = \{f : X \rightarrow \mathbb{F}_2^n\}$. Определим похожесть двух множеств $A \subset X$ и $B \subset X$ как $d(A, B) = |A \cap B| / |A \cup B|$. Пусть $2^n \gg |A \cup B|^2$.

- Придумайте, как, случайно выбрав из \mathcal{H} k функций, вероятно оценить $d(A, B)$ с точностью порядка $\frac{1}{\sqrt{k}}$ за $\mathcal{O}(k(|A| + |B|))$ вычислений хеш-функций, используя оценку:

$$d(A, B) \simeq \Pr_{h \in \mathcal{H}} [\min h(A) = \min h(B)].$$

Возможно, придется ввести дополнительные требования на семейство.

- Придумайте, как добиться того же результата с одной хеш-функцией и $\mathcal{O}(|A| + |B|)$ вычислений. Возможно, придется ввести дополнительные требования на семейство.

Разбор задач практики

1. Эффективная замена

Слово w можно передать за $|w|+1$ символов = $7(|w|+1)$ бит.

С таким способом передачи мы минимизируем $|w| - |w| \cdot \text{count}(w)$.

Если при подсчёте $\text{count}(w)$ забить на перекрытия, проще всего решается суфф.деревом.

Если учитывать перекрытия честно, воспользуемся литературностью: $|w| \leq 30$.

Переберём длину, внутри линейный проход с хеш-мэпом:

```
FOR(i) { h = hash(i,i+len); if (end[h] <= i) k[h]++, end[h]=i+len; }
```

Честное решение: $|w|\text{count}(w) \leq n \Rightarrow$ или $|w| \leq \sqrt{n}$, или $\text{count}(w) \leq \sqrt{n} \Rightarrow$ переберём все маленькие длины, а для каждого $\text{count}(w)$ запустим бинпоиск для $|w| \rightarrow \max$.

2. Обратное к МТФ

По индукции мы в каждый момент времени знаем перестановку ord .

Модифицируем лишь две строки исходного кода.

```
1 for (size_t i = 0; i < text.size(); i++):
2     int c = ord[text[i]]; // было: c = text[i]
3     text[i] = c; // было: text[i] = ord-1[c]
4     ...
```

Чтобы работало за $\mathcal{O}(n \log k)$, нужно дерево по неявному ключу, ссылки $c \rightarrow \text{node}[c]$ и подъём по дереву для определения позиции.

3. LZSS на практике

Сперва пишем один бит. $0 \Rightarrow \langle n, p \rangle$, $1 \Rightarrow c$.

$|p| = 20$ (не более длины текста).

Улучшение: $|p| = \lceil \log_2 i \rceil$ ($p \leq i$, i – текущая позиция).

$|n| \leq 20$, но на практике меньше. Мы ориентируемся на сжатие в ≈ 3 раза $\Rightarrow |n| \approx 5$.

На пару $\langle n, p \rangle$ мы тратим 25 бит \Rightarrow при $n \leq 3$ выгоднее просто писать символы.

Как подобрать $|n|$? Или угадать, или перебрать и сравнить длины кодов. Перебирать можно умно: тройичный поиск + окрестность найденной точки. При подборе $|n|$ важно, что мы можем один раз пустить LZSS, а потом уже смотреть много разных $|n|$.

Хаффманом следует сжимать исходные символы и n . Получится маленький алфавит.

Не нужно хранить ничего дополнительного: прочитали хаффманом или $1c$ или $0n$, если первый бит 0, это n и далее из следующих $|p|$ бит берём p .

4. LZSS сжимает википедию

Проблема: если строить суффмассив от большого файла, уходит много памяти и огребаем сплошные кеш-мисы.

Решение. Бъём на блоки. Блоки сжимаем независимо.

Какого размера 2^k блок оптимально использовать? При уменьшении k с одной стороны уменьшается $|p|$ (см.выше), с другой стороны уменьшается число текста, который можно переиспользовать \Rightarrow растёт число пар $\langle n, p \rangle$.

5. Обратное преобразование Барроуза-Уилера

- a) $\mathcal{O}(n^3)$. Первый столбец = сортировка последнего. Два первых = сортировка... и т.д.
 b) $\mathcal{O}(n^2)$. Восстановим всю таблицу.

Пусть уже знаем первые k столбцов (изначально 0). Тогда знаем все подстроки из $k + 1$ буквы, входящие в зацикленную строку, так как есть последний столбец.

Эти подстроки отсортированы по суффиксам длины k , можно досортировать стабильным подсчетом за $\mathcal{O}(n)$. Получили $k + 1$ первых столбцов.

- c) $\mathcal{O}(n)$. [\[Конспект Зк\]](#) [\[Конспект ИТМО\]](#), [\[Википедия\]](#). Задачу можно сдать на [тимусе](#).
 Сортируем последний столбец подсчётом (стабильным), получим первый.

$c[i]$ – символы первого столбца, $p[i]$ – их позиции в последнем.

Алгоритм:

```
1 for (int i = start, j = 0; j < n; j++):
2   out << c[i]; // вывести очередной символ строки
3   i = p[i]; // перейти к следующему
```

6. Универсальные и независимые

- a) Просто подставим в определения. Задача на понимание определений.
 b) Просто подставим в определения. Задача на понимание определений.
 c) 2-независимое семейство: $h_{a,b}(x) = ax + b$ (можно $a = 0$).
 d) k -независимое семейство: $h_{a_0, \dots, a_{k-1}} = a_0 + xa_1 + x^2a_2 + \dots$. Доказательство обоих пунктов: интерполяция многочлена $A(x_1) = y_1, A(x_2) = y_2, \dots, A(x_k) = y_k$ даёт единственный $A = [a_0, \dots, a_{k-1}] \Rightarrow$ вероятность его выпадения = $\frac{1}{P^k}$.

7. Словари и пары (хеши пар)

`unordered_map<int64_t, int> m; m[hash(a,b)] = 3.`

Функция `hash` должна быть с достаточно малой вероятностью коллизии.

`int64_t hash(int a, int b) { return ((int64_t)a << 32) + b; }`

$\langle u, v \rangle \rightarrow (u2^{64} + v) \bmod p$ плоха: легко сойтись коллизия $w = uv$, 128-битные w и $w + p$.

Решение #1: `w=(u<<64)+v; return (aw+b)%P%264`; ($P \approx 2^{128}$, простое). Нужен `int256`.

Решение #2 (полиномиальным хешированием): `return (u*x+v)%P`; x – случайное, P – простое. Чтобы работало $\forall u, v$, нужно $P > 2^{64}$. Иначе работает только для $\max u, v < P \leq 2^{64}$.

Вспомним про $\mathbb{F}_{2^{64}}$: `return u*x+v`; где `*` и `+` – операции над $\mathbb{F}_{2^{64}}$. Полиномиальное хеширование над $\mathbb{F}_{2^{64}}$ решает проблемы с переполнением, теперь не нужен `int128`.

8. Вертикальная симметрия

Нужно увидеть ось $z = \frac{1}{2}(x_{min} + x_{max})$. $\mathcal{O}(n \log n)$: `sort()` + циклом `for` проверить. $\mathcal{O}(n)$: сложить всё в хеш-таблицу и для каждой (x, y) за $\mathcal{O}(1)$ проверить что отражённая $(2z - x, y)$ тоже есть в наборе.

Для каждого y получаем отдельную задачу: точки = дерево по x , внутри которого в каждой вершине полиномиальный хеш от массива $dx_i = x_{i+1} - x_i$. Нужно проверить, что хеш первой половины массива равен хешу развёрнутой второй половины.

9. Упрощаем универсальное семейство

Очевидно упрощаем до $(ax + b) \bmod p$. Но теперь $+b$ даёт лишь циклический сдвиг \Rightarrow тоже не нужно. Итого $h_a(x) = ax \bmod p$, $\{h_a\}$ – универсальное семейство $[0, p) \rightarrow [0, p)$. Нужно ли такое? С одной стороны нет, т.к. отображение при $n = p$ не сжимающее. С другой стороны эта функция неплохо работает для хеш-таблицы с открытой адресацией (см. следующую задачу).

10. (*) Разбить строку на 2 почти палиндрома

Предподсчёт: Манакер + хеши/Z.

Решение: перебираем разрез, в каждой половине ищем позицию ошибки Манакером, проверяем остаток на равенство. Проверять остаток проще всего хешами, а можно вместо хешей предподсчитать для левой половины $Z(s\#\text{rev}(s))$, аналогично для правой.

11. (*) Валим хеш-таблицу с открытой адресацией

Возьмём $k = \frac{n}{4}$. Добавим $1, 2, 3, \dots, k$ и k случайных чисел. Первые k чисел дадут отрезок из k подряд идущих занятых ячеек. При попадании в первые $\frac{1}{2}k$ из них `add` будет работать за $\Theta(k)$, для каждого из следующих k `add` такое событие происходит с вероятностью $\frac{1}{8} \Rightarrow E[\text{времени add}] = \Theta(n)$.

12. (*) Почти совпадения — 2

Почти также как «почти-совпадения-1». В каждом словарном слове перебираем позицию ошибки i , за $\mathcal{O}(1)$ считаем хеш для слова с пропуском в позиции i . Все хеши заботливо кладём в хеш-таблицу. Когда приходит запрос, перебираем позицию ошибки, ищем в хеш-таблице совпадения, складываем. Остаётся проблема «почему словарные слова мы не учли два раза?». Учесть два раза могли только словарное слово равное запросу, заифаем.

13. (*) Жесть

?

Домашнее задание

3.1. Обязательная часть

1. (1+1) Совершенная хеш-функция

Является ли семейство из одной совершенной хеш-функции:

(a) универсальным? (b) 2-независимым?

2. (2) Хеш пар

Нам даны $0 \leq u, v < P$, P – простое.

Мы хешируем пару $h_x(\langle u, v \rangle) = (u \cdot x + v) \bmod P$. Покажите, что $\mathcal{H} = \{h_x\}$ (полиномиальные хеши для пары) – универсальное семейство хеш-функций $[0, P^2) \rightarrow [0, P)$.

3. (2) Равенство меняющихся строк

Нужно быстро отвечать на запросы «сравнить на равенство две подстроки», «поменять символ строки», «вставить символ в строку», «удалить символ из строки».

4. (1+2) Количество семейств

а) (1) Сколько существует различных функций $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$?

б) (2) Для каких n и m семейство всех функций $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ является универсальным? 2-независимым?

3.2. Дополнительная часть

1. (2) Подпалиндромы на отрезках

Дана строка s и q запросов «самый длинный подпалиндром на отрезке строки $[l..r]$ ».

Решить за $\mathcal{O}((|s| + q) \cdot \text{poly}(\log))$.

2. (3) Жизнь и жесть

Дано начальное состояние в игре «Жизнь» в виде битовой матрицы $N \times N$. Нас интересует, сколько останется живых клеток на поле после большого количества итераций $T \gg N$. Для простоты можно считать N и T степенями двойки.

а) Определим функцию e , сопоставляющую тайлу размера $2^n \times 2^n$ кусок размера $2^{n-1} \times 2^{n-1}$ из его середины на 2^{n-2} шагов в будущем. Есть ли рекуррентное соотношение для e ?

б) Придумайте, как с помощью хеширования решить нашу задачу быстрее, чем наивной симуляцией.