Второй курс, осенний семестр 2021/22
Практика по алгоритмам #8

Mix про строки и сжатие

8 ноября

Собрано 20 декабря 2021 г. в 09:18

Содержание

Mix про строки и сжатие

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?

   (*) Предложите честное решение корневой.

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 ≈1mb, но помните, что ваш способ должен быть универсальным. Как можно применить сюда Хаффмана?

4. **LZSS сжимает википедию**

   Как пожать 16 гигабайт входных данных? Как использовать это для улучшения в предыдущей задаче?

5. **Обратное преобразование Барроуза-Уилера**

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

   a) $\mathcal{O}(n^3)$
   b) $\mathcal{O}(n^2)$
   c) $\mathcal{O}(n)$ на примере `aabaxbx`.

Перерыв

6. **Равенство меняющихся строк**

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

7. **Словари и пары (хеши пар)**

   Пусть ваш язык программирования имеет встроенный ассоциативный массив `int64` → `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:

   a) $k = 32$. Простое решение.
   b) $k = 64$. Чем плохо решение $\langle u, v \rangle \to (u2^{64} + v) \bmod p$?
   c) $k = 64$. Решение универсальным семейством.
   d) $k = 64$. Решение полиномиальным хешированием.
   e) `(*)` $k = 64$. Вспомните про $\mathbb{F}_{2^{64}}$. Как улучшить предыдущее решение?
   f) Получите из «полиномиальных хешей для пары» универсальное семейство.

8. **Вертикальная симметрия**

   a) На плоскости даны n точек $x_i, y_i \in \mathbb{Z}$.
   Проверьте за $\mathcal{O}(n)$, есть ли у них вертикальная ось симметрии.

   b) Точки добавляются/удаляются. После каждого запроса за $\mathcal{O}(\log n)$ проверять симметрию.

9. **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.

10. **Универсальные и независимые**

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

    $$\forall x_1, x_2 \in X, x_1 \neq x_2 : \mathop{\mathbf{Pr}}_{h \in \mathcal{H}} [\, h(x_1) = h(x_2) \,] \leqslant \frac{1}{|Y|}$$

    Семейство хэш-функций $\mathcal{H} = \{h : X \to Y\}$ называется k-независимым, если
    $\forall$ различных $x_1, x_2, \cdots, x_k \in X$, и $\forall$, возможно, совпадающих $y_1, y_2, \cdots, y_k \in Y$ выполняется:

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

    a) Докажите, что любое 2-независимое семейство хэш-функций является универсальным.
    b) Докажите, что любое $k+1$-независимое семейство хэш-функций является k-независимым.
    c) Приведите пример 2-независимого семейства для случая $|Y| \in \mathbb{P}$.
    d) Приведите пример k-независимого семейства для случая $|Y| \in \mathbb{P}$.
    e) `(*)` Приведите пример k-независимой функции. Докажите.

11. **Совершенная хеш-функция**

    Является ли семейство из одной совершенной хеш-функции:

    a) универсальным?
    b) 2-независимым?

12. **Упрощаем универсальное семейство**

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

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

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

14. (*) **Количество семейств**

a) (*) Сколько существует различных функций $\mathbb{F}_2^n \to \mathbb{F}_2^m$?

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

c) (**) Может ли существовать конечное универсальное семейство функций $X \to \mathbb{F}_2^m$, если X бесконечно?

15. (*) **Подпалиндромы на отрезках**

Дана строка s и q запросов «самый длинный подпалиндром на отрезке строки $[l..r]$».
Решить за $\mathcal{O}((|s| + q) \cdot \texttt{poly}(\log))$.

16. (*) **Полиномиальные хеши универсальны?**

Пусть модуль m, по которому берётся многочлен в полиномиальном хешировании не фиксирован, а выбирается из некоторого конечного набора простых чисел. Пусть точка x для полиномиального хеширования выбирается из множества $\{1, 2, \ldots, m-1\}$. Докажите, что такое семейство полиномиальных хеш-функций не является универсальным.

17. (*) **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:

a) какое-нибудь похожее слово словаря;

b) кол-во похожих слов в словаре.

Разбор задач практики

1. **Эффективная замена**

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

   С таким способом передачи мы минимизируем $|w| - |w| \cdot count(w)$.
   Если при подсчёте $count(w)$ забить на перекрытия, проще всего решается суф.деревом.

   Если учитывать перекрытия честно, воспользуемся литературностью: $|w| \leqslant 30$.
   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: $|w|count(w) \leqslant n \Rightarrow$ или $|w| \leqslant \sqrt{n}$, или $count(w) \leqslant \sqrt{n} \Rightarrow$ переберём все маленькие длины, а для каждого $count(w)$ запустим бинпоиск для $|w| \to \max$.

2. **Обратное к MTF**

   По индукции мы в каждый момент времени знаем перестановку `ord`.
   Модифицируем лишь две строки исходного кода.

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

   Чтобы работало за $\mathcal{O}(n \log k)$, нужно дерево по неявному ключу, ссылки $c \to node[c]$ 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: $|p| = \lceil \log_2 i \rceil$ ($p \leqslant i$, i – текущая позиция).
   $|n| \leqslant 20$, но на практике меньше. Мы ориентируемся на сжатие в $\approx$3 раза $\Rightarrow |n| \approx 5$.
   На пару $\langle n, p \rangle$ мы тратим 25 бит $\Rightarrow$ при $n \leqslant 3$ выгоднее просто писать символы.

   Как подобрать $|n|$? Или угадать, или перебрать и сравнить длины кодов. Перебирать можно умно: троичный 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. Бьём 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 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:

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

6. **Равенство меняющихся строк**

   Храним хеш в вершинах дерева поиска/дерева отрезков, для 3-го запроса нужен неявный ключ. Запрос на отрезке «посчитать хеш отрезка».
   Операция склеивания для пересчёта хеша: $h[l,r) = h[l,m)p^{r-m} + h[m,r)$.

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; }`

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

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

   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 $\max u, v < P \leqslant 2^{64}$.

   «Полиномиальных хеши для пар» дают $\Pr_x[\text{коллизии}] \leqslant \frac{\deg}{P} = \frac{1}{P} \Rightarrow$
   $\{\langle 0,P\rangle, \langle 1,P\rangle, \ldots \langle P-1,P\rangle\}$ – универсальное семейство для сжатия $[0, P^2) \to [0, P)$.

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

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. **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#rev(s))`, аналогично для правой.

10. **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 $= \frac{1}{P^k}$.
    e) `(*)` ?

11. **Совершенная хеш-функция**

    Является ли семейство из одной совершенной хеш-функции:

    a) универсальным? конечно, **да** (вероятность ошибки 0).
    b) 2-независимым? конечно, **нет** (не сюръекция).

12. **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 $[0, p) \to [0, p)$. 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).

13. `(*)` **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[времени `add`] $= \Theta(n)$.

14. `(*)` **Количество семейств**

    ?

15. `(*)` **Подпалиндромы на отрезках**

    Решение #1: манакер + бинпоиск по ответу x + максимум z_i на отрезке $[l+x, r-x]$.

    Решение #2: разобьём точкой $m = \frac{1}{2}(l + r)$ задачу на две одинаковых, для $i \leqslant m$ имеем $\min(z_i, i-l) \to \max$: при $i \leqslant m, z[i] \leqslant i-l$ имеем $z_i \to \max$ и наоборот.

16. `(*)` **Полиномиальные хеши универсальны?**

    ?

17. `(*)` **Почти совпадения — 2**

    ?