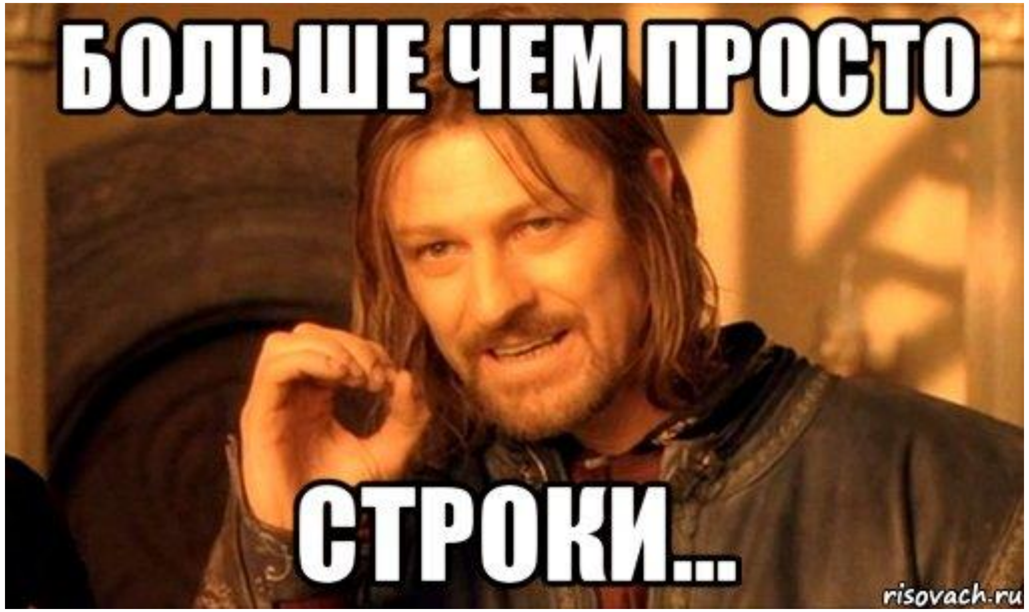
академия больших данных

mail.ru group

Базовые алгоритмы на строках

Шовкопляс Григорий

Введение в алгоритмы и структуры данных



Строки и популярные задачи

Основные определения

- Алфавит — конечное непустое множество символов
- Строка — конечная последовательность символов некоторого алфавита
- Полезные определения частей:
  - Префикс
  - Суффикс
  - Подстрока

Популярные задачи

- Сравнение подстрок
  - $O(|S|)$
- Поиск подстроки P в строке T
  - $O(|P||T|)$
- Все остальные сводятся плюс-минус к предыдущим

Хеширование строк

Полиномиальный хеш

- $h(s) = (s_0 \times p^0 + s_1 \times p^1 + \dots + s_{n-1} \times p^{n-1}) \; mod \; M$
- p – простое, чуть больше алфавита, M – "большое"
- На самом деле более удобно в будущем:
- $h(s) = (s_0 \times p^{n-1} + s_1 \times p^{n-2} + \dots + s_{n-1} \times p^0) \; mod \; M$
- Если хеши равны, то строки «равны»
- Какова вероятность коллизии?
- $P = \frac{n}{M}$
- По жизни этого достаточно

Полиномиальный хеш

- Хеш подстроки
- $h(s_{l..r}) = (s_l \times p^{r-l} + s_{l+1} \times p^{r-l-1} + \cdots + s_r \times p^0) \; mod \; M$
- Можно ли вычислить быстро?
- Предподсчет хеши на префиксах:
- $hash[i] = h(s_{0..i})$
- $hash[i] = (hash[i-1] \times p + s_i) mod \; M$
- $h(s_{l..r}) = \left(hash[r] - hash[l \; - 1] \times p^{r-l+1}\right) mod \; M$
- Получается можно найти хеш любой подстроки за O(1)

Полиномиальный хеш

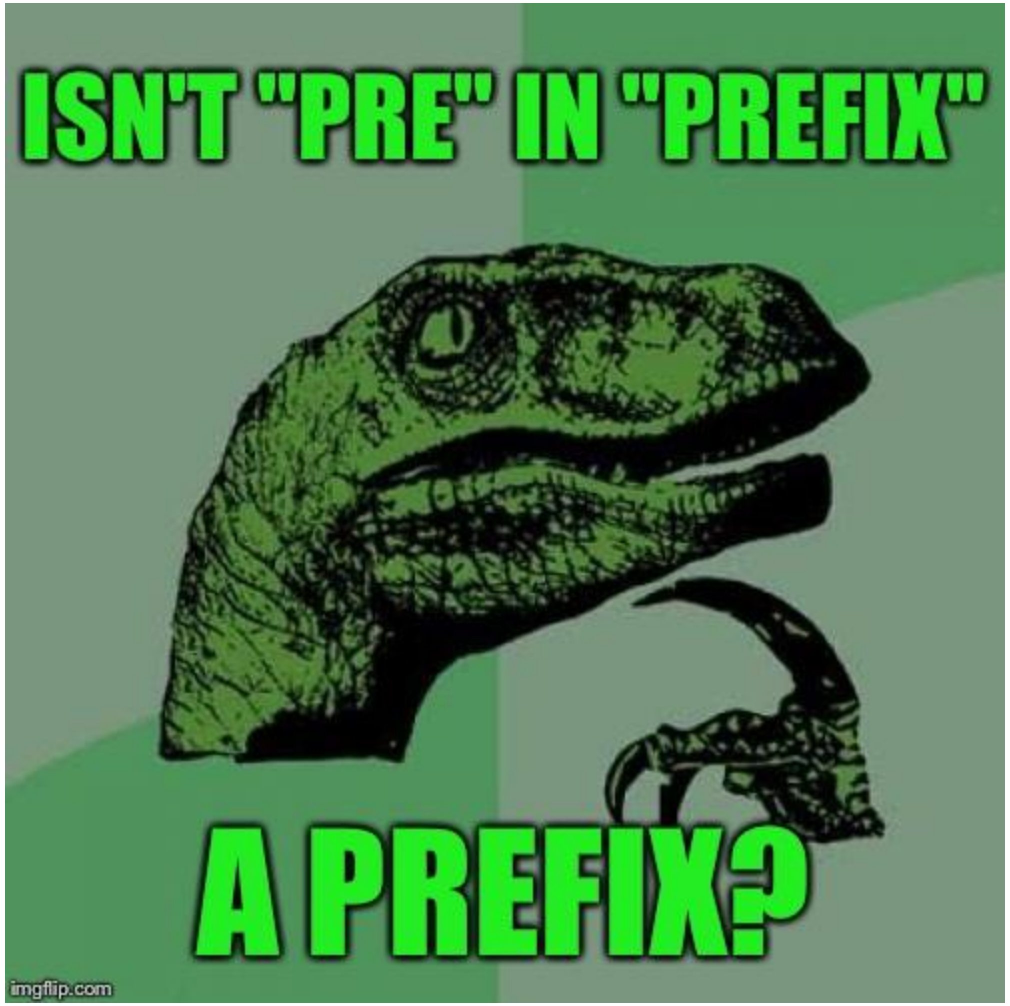
- Хеш подстроки

- Чего не хватает?
- l может быть равно 0!

```
getHash(l, r)
  if l == 0
    return hash[r]
  return (hash[r] - (hash[l-1] * powp[r-l+1])%M
                                          + M) % M

init(s)
  hash[0] = s[0]
  powp[0] = 1
  for i = 1 to |s| - 1
    hash[i] = (hash[i - 1] * p + s[i]) % M
    powp[i] = (powp[i - 1] * p) % M
```

Использование хешей

- Сравнение двух подстрок на равенство
  - $O(1)$
- Сравнение двух подстрок на больше/меньше
  - $O(\log|S|)$
- Поиск подстроки P в строке T
  - $O(|P| + |T|)$



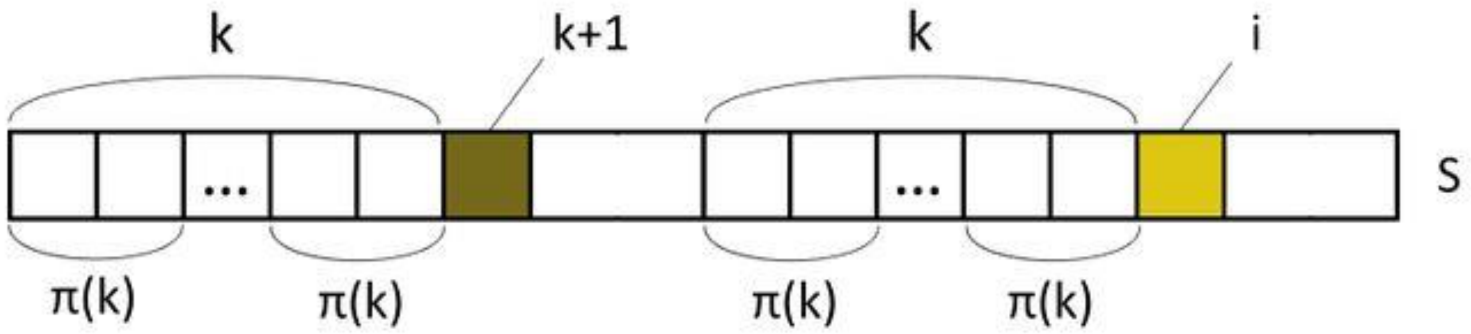
Префикс функция

Префикс функция

- Определим такую функцию $p(i)$ для каждого индекса строки
- $p(i) = \max\limits_{1..i}\{k: s_{0..k-1} = s_{i-k+1..i}\}$
- Если нет такого k, то $p(i) = 0$
- Наивно может посчитать за $O(n^3)$
- Для каждого индекса переберем, все k, сравним строки
- Можно соптимизирвать до $O(n^2)$
- Давайте научимся считать за $O(n)$

Префикс функция

- Заметим:
- $p[i+1] \leq p[i] + 1$
- Можно использовать информацию, полученную ранее
- $s[i+1] = s\big[p[i]\big] \Rightarrow p[i+1] = p[i]$
- А что делать, если $s[i+1] \neq s\big[p[i]\big]$?



Префикс функция

- Построение префикс-функции
- Почему работает за O(n)?

```
pfunction(s)
  p[0] = 0
  for i = 1 to |s| - 1
    k = p[i - 1]
    while k > 0 and s[i] != s[k]
      k = p[k - 1]
    if s[i] == s[k]
      k++
    p[i] = k
  return p
```

Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта

- Хотим быстро находить подстроку P в строке T
- Строим строку $S = P + \text{‘\#’} + T$
- Строим префикс функцию S
- Найдем $p[i] = |P|$
- Для каждого такого i будет вхождение с $i - |P|$ до i
- Время работы $O(|P| + |T|)$

Z-функция

Z-функция

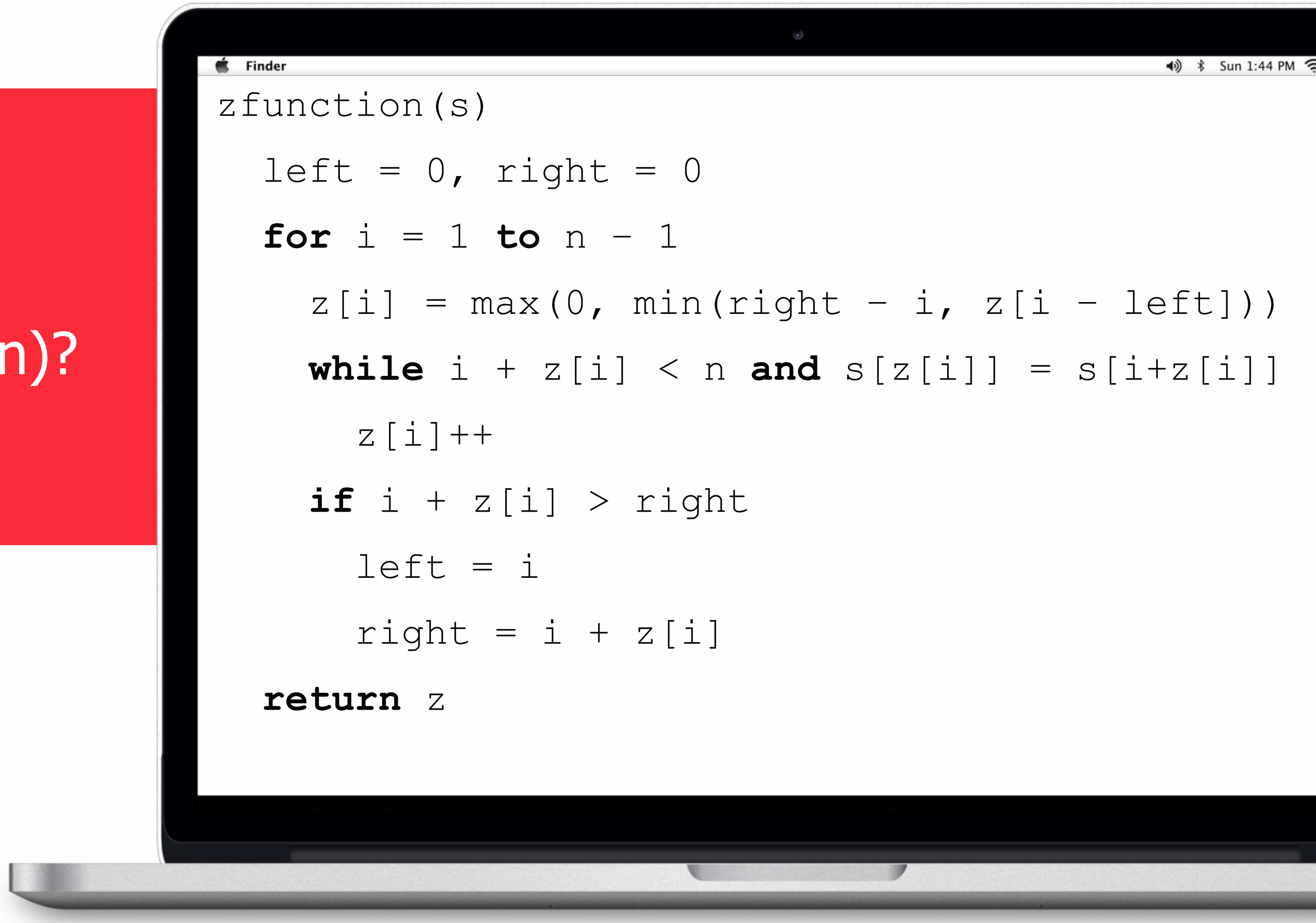
- Определим такую функцию z(i) для каждого индекса строки
- $z(i) = max\{k: s_{0..k-1} = s_{i..i+k-1}\}, z(0) = -$
- Если нет такого $k,$ то $z(i) = 0$
- Наивно может посчитать за $O(n^3)$
- Для каждого индекса переберем, все k, сравним строки
- Можно соптимизирвать до $O(n^2)$
- Давайте научимся считать за $O(n)$

Z-функция

- Z-блок – подстроку с началом в позиции i и длиной Z[i]
- Будем хранить Z-блок, с максимальной правой границей
- Пусть этот блок с $left$ до $right$
- $i > right$
- Ничего не знаем о позиции i и дальше, так что посчитаем в лоб
- $i \leq right$
- $z[i] \geq min(right - i, z[i - left])$
- Продлим в лоб

Z-функция

- Построение z-функции
- Почему работает за O(n)?




```
zfunction(s)
  left = 0, right = 0
  for i = 1 to n − 1
    z[i] = max(0, min(right − i, z[i − left]))
    while i + z[i] < n and s[z[i]] = s[i+z[i]]
      z[i]++
    if i + z[i] > right
      left = i
      right = i + z[i]
  return z
```

Бор (trie, префиксное дерево)

Бор

- Хотим хранить множество строк
  - insert(s)
  - contains(s)
- Хеш-таблицы
- Двоичное дерево
- Бор!

	Бор	Дерево	Хеш-таблица
Добавление элемента	$O(\|S\|)$	O(|S|logk)	$O(\|S\|)$
Получение ключей в отсортированном порядке	O(k)	O(k)	O(klogk)

Бор

- Дерево
- На ребрах написаны символы
- $next[u][c]$ – переход из вершины u по символу c
- Если в вершине заканчивается слово, пометим ее терминальной

Бор

- Структура + инициализация

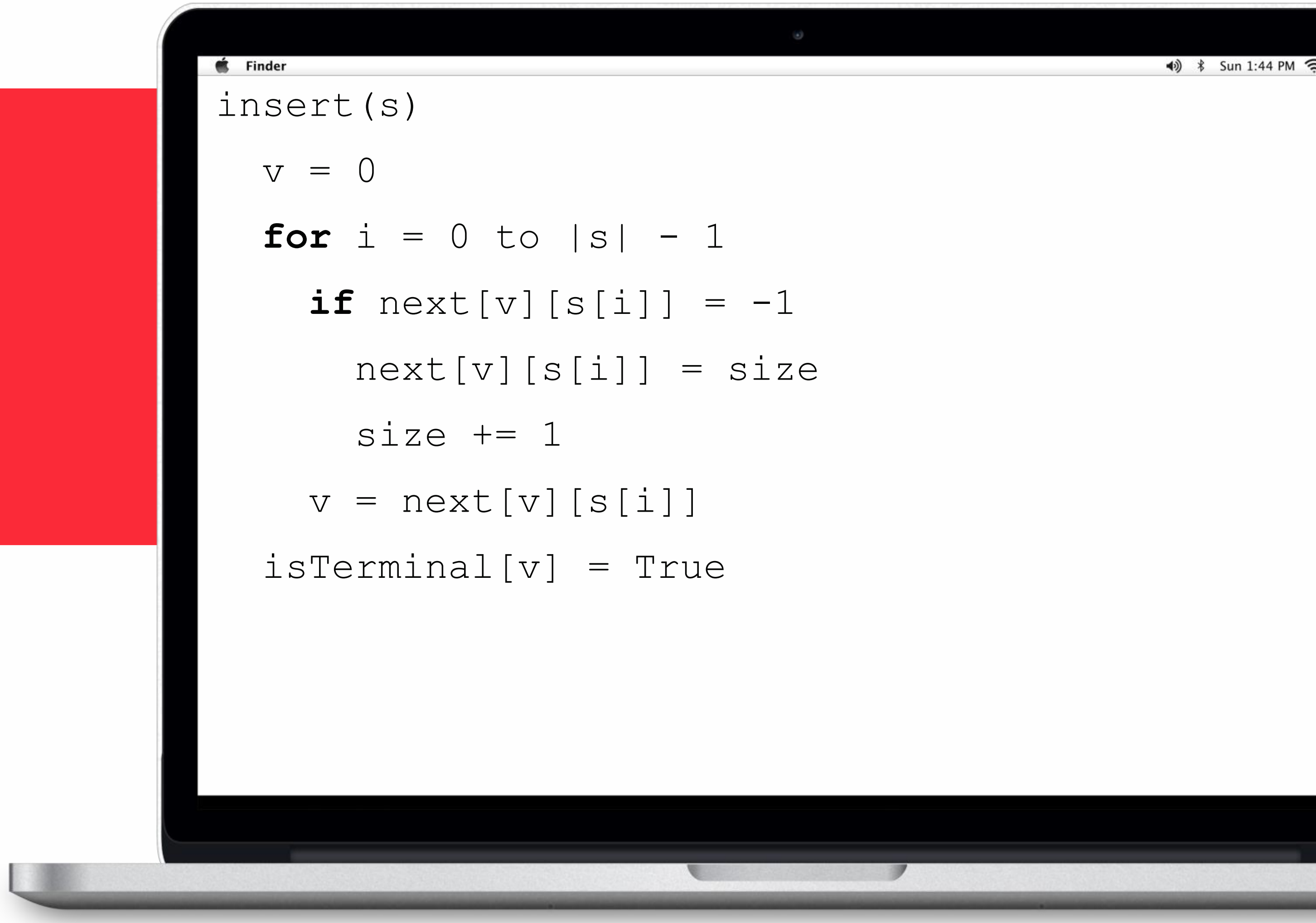
```
Trie:
  size
  next[][]
  isTerminal[]

init()
  size = 1
  for c in ALPHABET
    next[0][c] = -1
  isTerminal[0] = false
```

Бор

- Добавление
- Переходим в боре по символу строки, пока можем
- Если не можем, добавляем и переходим
- В конце помечаем терминальную
- Все!

Бор

- Добавление



Бор

- Поиск аналогично

```
contains(s)
  v = 0
  for i = 0 to |s| - 1
    if next[v][s[i]] = -1
      return False
    v = next[v][s[i]]
  if isTerminal[v]
    return True
  return False
```

Бор

- Сколько памяти?
- $O(|\sum s_i|)$

- Можем теперь за $O(\max|P_i| \times T)$ проверять есть ли одна из строк в тексте
- Можно соптимизировать до $O(T)$ см. алгоритм Ахо-Корасик

Все!