MAXimal

home
algo
bookz
forum
about

добавлено: 11 Jun 2008 10:35
редактировано: 26 Apr 2012 2:00

Z-функция строки и её вычисление

Содержание [скрыть]

- Z-функция строки и её вычисление
  - Примеры
  - Тривиальный алгоритм
  - Эффективный алгоритм вычисления Z-функции
  - Реализация
  - Асимптотика алгоритма
  - Применения
    - Поиск подстроки в строке
    - Количество различных подстрок в строке
    - Сжатие строки
  - Задачи в online judges

Пусть дана строка s длины n. Тогда **Z-функция** ("зет-функция") от этой строки — это массив длины n, i-ый элемент которого равен наибольшему числу символов, начиная с позиции i, совпадающих с первыми символами строки s.

Иными словами, $z[i]$ — это наибольший общий префикс строки s и её i-го суффикса.

Примечание. В данной статье, во избежание неопределённости, мы будем считать строку 0-индексированной — т.е. первый символ строки имеет индекс 0, а последний — $n-1$.

Первый элемент Z-функции, $z[0]$, обычно считают неопределённым. В данной статье мы будем считать, что он равен нулю (хотя ни в алгоритме, ни в приведённой реализации это ничего не меняет).

В данной статье приводится алгоритм вычисления Z-функции за время $O(n)$, а также различные применения этого алгоритма.

Примеры

Приведём для примера подсчитанную Z-функцию для нескольких строк:

- $"aaaaa"$:

$$z[0]=0,\\ z[1]=4,\\ z[2]=3,\\ z[3]=2,\\ z[4]=1.$$

- $"aaabaab"$:

$$z[0]=0,\\ z[1]=2,\\ z[2]=1,\\ z[3]=0,\\ z[4]=2,\\ z[5]=1,\\ z[6]=0.$$

- $"abacaba"$:

$$z[0] = 0,$$
$$z[1] = 0,$$
$$z[2] = 1,$$
$$z[3] = 0,$$
$$z[4] = 3,$$
$$z[5] = 0,$$
$$z[6] = 1.$$

Тривиальный алгоритм

Формальное определение можно представить в виде следующей элементарной реализации за $O(n^2)$:

```
vector<int> z_function_trivial (string s) {
	int n = (int) s.length();
	vector<int> z (n);
	for (int i=1; i<n; ++i)
		while (i + z[i] < n && s[z[i]] == s[i+z[i]])
			++z[i];
	return z;
}
```

Мы просто для каждой позиции i перебираем ответ для неё $z[i]$, начиная с нуля, и до тех пор, пока мы не обнаружим несовпадение или не дойдём до конца строки.

Разумеется, эта реализация слишком неэффективна, перейдём теперь к построению эффективного алгоритма.

Эффективный алгоритм вычисления Z-функции

Чтобы получить эффективный алгоритм, будем вычислять значения $z[i]$ по очереди — от $i = 1$ до $n - 1$, и при этом постараемся при вычислении очередного значения $z[i]$ максимально использовать уже вычисленные значения.

Назовём для краткости подстроку, совпадающую с префиксом строки s, **отрезком совпадения**. Например, значение искомой Z-функции $z[i]$ — это длиннейший отрезок совпадения, начинающийся в позиции i (и заканчиваться он будет в позиции $i + z[i] - 1$).

Для этого будем поддерживать **координаты $[l; r]$ самого правого отрезка совпадения**, т.е. из всех обнаруженных отрезков будем хранить тот, который оканчивается правее всего. В некотором смысле, индекс r — это такая граница, до которой наша строка уже была просканирована алгоритмом, а всё остальное — пока ещё не известно.

Тогда если текущий индекс, для которого мы хотим посчитать очередное значение Z-функции, — это i, мы имеем один из двух вариантов:

- $i > r$ — т.е. текущая позиция лежит **за пределами** того, что мы уже успели обработать.

Тогда будем искать $z[i]$ **тривиальным алгоритмом**, т.е. просто пробуя значения $z[i]=0$, $z[i]=1$, и т.д. Заметим, что в итоге, если $z[i]$ окажется >0, то мы будем обязаны обновить координаты самого правого отрезка $[l;r]$ — т.к. $i+z[i]-1$ гарантированно окажется больше r.

- $i \le r$ — т.е. текущая позиция лежит внутри отрезка совпадения $[l;r]$.

  Тогда мы можем использовать уже подсчитанные **предыдущие** значения Z-функции, чтобы проинициализировать значение $z[i]$ не нулём, а каким-то возможно бОльшим числом.

  Для этого заметим, что подстроки $s[l \ldots r]$ и $s[0 \ldots r-l]$ **совпадают**. Это означает, что в качестве начального приближения для $z[i]$ можно взять соответствующее ему значение из отрезка $s[0 \ldots r-l]$, а именно, значение $z[i-l]$.

  Однако значение $z[i-l]$ могло оказаться слишком большим: таким, что при применении его к позиции i оно "вылезет" за пределы границы r. Этого допустить нельзя, т.к. про символы правее r мы ничего не знаем, и они могут отличаться от требуемых.

  Приведём **пример** такой ситуации, на примере строки:

  $$"aaaabaa"$$

  Когда мы дойдём до последней позиции ($i=6$), текущим самым правым отрезком будет $[5;6]$. Позиции 6 с учётом этого отрезка будет соответствовать позиция $6-5=1$, ответ в которой равен $z[1]=3$. Очевидно, что таким значением инициализировать $z[6]$ нельзя, оно совершенно некорректно. Максимум, каким значением мы могли проинициализировать — это 1, поскольку это наибольшее значение, которое не вылазит за пределы отрезка $[l;r]$.

  Таким образом, в качестве **начального приближения** для $z[i]$ безопасно брать только такое выражение:

  $$z_0[i] = \min(r-i+1, z[i-l]).$$

  Проинициализировав $z[i]$ таким значением $z_0[i]$, мы снова дальше действуем **тривиальным алгоритмом** — потому что после границы r, вообще говоря, могло обнаружиться продолжение отрезка совпадение, предугадать которое одними лишь предыдущими значениями Z-функции мы не могли.

Таким образом, весь алгоритм представляет из себя два случая, которые фактически различаются только **начальным значением** $z[i]$: в первом случае оно полагается равным нулю, а во втором — определяется по предыдущим значениям по указанной формуле. После этого обе ветки алгоритма сводятся к выполнению **тривиального алгоритма**, стартующего сразу с указанного начального значения.

Алгоритм получился весьма простым. Несмотря на то, что при каждом i в нём так или иначе выполняется тривиальный алгоритм — мы достигли существенного прогресса, получив алгоритм, работающий за линейное время. Почему это так, рассмотрим ниже, после того, как приведём реализацию алгоритма.

Реализация

Реализация получается весьма лаконичной:

```
vector<int> z_function (string s) {
	int n = (int) s.length();
	vector<int> z (n);
	for (int i=1, l=0, r=0; i<n; ++i) {
		if (i <= r)
			z[i] = min (r-i+1, z[i-l]);
		while (i+z[i] < n && s[z[i]] == s[i+z[i]])
			++z[i];
		if (i+z[i]-1 > r)
			l = i,  r = i+z[i]-1;
	}
	return z;
}
```

Прокомментируем эту реализацию.

Всё решение оформлено в виде функции, которая по строке возвращает массив длины n — вычисленную Z-функцию.

Массив $z[]$ изначально заполняется нулями. Текущий самый правый отрезок совпадения полагается равным $[0;0]$, т.е. заведомо маленький отрезок, в который не попадёт ни одно i.

Внутри цикла по $i = 1 \ldots n-1$ мы сначала по описанному выше алгоритму определяем начальное значение $z[i]$ — оно либо останется нулём, либо вычислится на основе приведённой формулы.

После этого выполняется тривиальный алгоритм, который пытается увеличить значение $z[i]$ настолько, насколько это возможно.

В конце выполняется обновление текущего самого правого отрезка совпадения $[l;r]$, если, конечно, это обновление требуется — т.е. если $i + z[i] - 1 > r$.

Асимптотика алгоритма

Докажем, что приведённый выше алгоритм работает за линейное относительно длины строки время, т.е. за $O(n)$.

Доказательство очень простое.

Нас интересует вложенный цикл $\rm while$ — т.к. всё остальное — лишь константные операции, выполняемые $O(n)$ раз.

Покажем, что **каждая итерация** этого цикла $\rm while$ приведёт к увеличению правой границы r на единицу.

Для этого рассмотрим обе ветки алгоритма:

- $i > r$

  В этом случае либо цикл $\rm while$ не сделает ни одной итерации (если $s[0] \neq s[i]$), либо же сделает несколько итераций, продвигаясь каждый раз на один символ вправо, начиная с позиции i, а после этого — правая граница r обязательно обновится.

  Поскольку $i > r$, то мы получаем, что действительно каждая итерация этого цикла увеличивает новое значение r на единицу.

- $i \le r$

  В этом случае мы по приведённой формуле инициализируем значение $z[i]$ некоторым числом z_0. Сравним это начальное значение z_0 с величиной $r - i + 1$, получаем три варианта:

  - $z_0 < r - i + 1$

    Докажем, что в этом случае ни одной итерации цикл while не сделает.

    Это легко доказать, например, от противного: если бы цикл while сделал хотя бы одну итерацию, это бы означало, что определённое нами значение z_0 было неточным, меньше настоящей длины совпадения. Но т.к. строки $s[l \ldots r]$ и $s[0 \ldots r - l]$ совпадают, то это означает, что в позиции $z[i - l]$ стоит неправильное значение: меньше, чем должно быть.

    Таким образом, в этом варианте из корректности значения $z[i - l]$ и из того, что оно меньше $r - i + 1$, следует, что это значение совпадает с искомым значением $z[i]$.

  - $z_0 = r - i + 1$

    В этом случае цикл while может совершить несколько итераций, однако каждая из них будет приводить к увеличению нового значения r на единицу: потому что первым же сравниваемым символом будет $s[r + 1]$, который вылазит за пределы отрезка $[l; r]$.

  - $z_0 > r - i + 1$

    Этот вариант принципиально невозможен, в силу определения z_0.

Таким образом, мы доказали, что каждая итерация вложенного цикла приводит к продвижению указателя r вправо. Т.к. r не могло оказаться больше $n - 1$, это означает, что всего этот цикл сделает не более $n - 1$ итерации.

Поскольку вся остальная часть алгоритма, очевидно, работает за $O(n)$, то мы доказали, что и весь алгоритм вычисления Z-функции выполняется за линейное время.

Применения

Рассмотрим несколько применений Z-функции при решении конкретных задач.

Применения эти будут во многом аналогичным применениям префикс-функции.

Поиск подстроки в строке

Во избежании путаницы, назовём одну строку **текстом** t, другую — **образцом** p. Таким образом, задача заключается в том, чтобы найти все вхождения образца p в текст t.

Для решения этой задачи образуем строку $s = p + \# + t$, т.е. к образцу припишем текст через символ-разделитель (который не встречается нигде в самих строках).

Посчитаем для полученной строки Z-функцию. Тогда для любого i в отрезке $[0; length(t) - 1]$ по соответствующему значению $z[i + length(p) + 1]$ можно понять, входит ли образец p в текст t, начиная с позиции i: если это значение Z-функции равно $length(p)$, то да, входит, иначе — нет.

Таким образом, асимптотика решения получилась $O(length(t) + length(p))$. Потребление памяти имеет ту же асимптотику.

Количество различных подстрок в строке

Дана строка s длины n. Требуется посчитать количество её различных подстрок.

Будем решать эту задачу итеративно. А именно, научимся, зная текущее количество различных подстрок, пересчитывать это количество при добавлении в конец одного символа.

Итак, пусть k — текущее количество различных подстрок строки s, и мы добавляем в конец символ c. Очевидно, в результате могли появиться некоторые новые подстроки, оканчивавшиеся на этом новом символе c (а именно, все подстроки, оканчивающиеся на этом символе, но не встречавшиеся раньше).

Возьмём строку $t = s + c$ и инвертируем её (запишем символы в обратном порядке). Наша задача — посчитать, сколько у строки t таких префиксов, которые не встречаются в ней более нигде. Но если мы посчитаем для строки t Z-функцию и найдём её максимальное значение $z_{\max}$, то, очевидно, в строке t встречается (не в начале) её префикс длины $z_{\max}$, но не большей длины. Понятно, префиксы меньшей длины уже точно встречаются в ней.

Итак, мы получили, что число новых подстрок, появляющихся при дописывании символа c, равно $len - z_{\max}$, где len — текущая длина строки после приписывания символа c.

Следовательно, асимптотика решения для строки длины n составляет $O(n^2)$.

Стоит заметить, что совершенно аналогично можно пересчитывать за $O(n)$ количество различных подстрок и при дописывании символа в начало, а также при удалении символа с конца или с начала.

Сжатие строки

Дана строка s длины n. Требуется найти самое короткое её "сжатое" представление, т.е. найти такую строку t наименьшей длины, что s можно представить в виде конкатенации одной или нескольких копий t.

Для решения посчитаем Z-функцию строки s, и найдём первую позицию i такую, что $i + z[i] = n$, и при этом n делится на i. Тогда строку s можно сжать до строки длины i.

Доказательство такого решения практически не отличается от доказательства решения с помощью префикс-функции.

Задачи в online judges

Список задач, которые можно решить, используя Z-функцию:

- UVA #455 **"Periodic Strings"** [сложность: средняя]
- UVA #11022 **"String Factoring"** [сложность: средняя]