Z-блоки

- Альтернативный подход к описанию структуры строк (Д. Гасфилд)
- *Определение*. Значение $Z_i(S)$ – длина наибольшей подстроки в *S*, которая начинается в позиции *i > 0* и совпадает с префиксом *S*.
  - Такая подстрока называется *Z-блоком*
  - При *i* = 0 полагают $Z_0(S) = 0$
  - Пример: *S* = $AABCAABXAAZ$

    0123456789

    - $Z_4(S)$ = 3 (*AABCAABXAAZ*)
    - $Z_5(S)$ = 1 (*AABCAABXAAZ*)
    - $Z_6(S)$ = 0 (*AABCAABXAAZ*)
    - $Z_7(S)$ = 0 (*AABCAABXAAZ*)
    - $Z_8(S)$ = 2 (*AABCAABXAAZ*)

Вычисление *Z*-блоков

- Задача: вычисление массива *Z*-значений *zp*[0..*n*-1] (*Z*-функция)
  - Наивный алгоритм: непосредственное сравнение подстрок S[i..j] с префиксом для *i* = 1, 2, …, *n*-1.
  - Сложность – $O(n^2) \sim$ (n-1) + (n-2) + … + 2 + 1

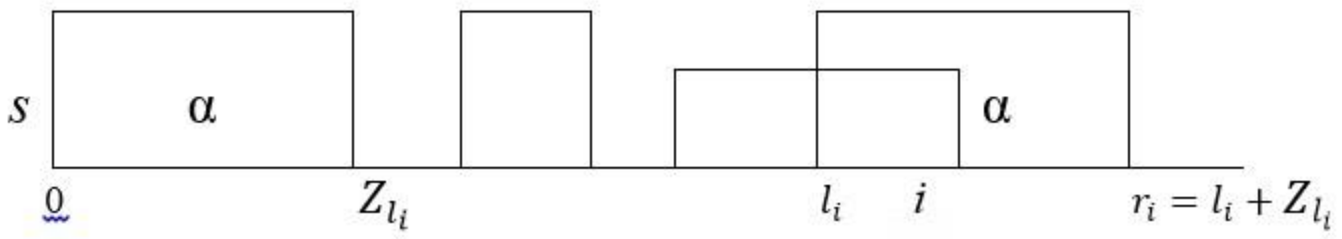
Массив *Z*-значений – наивный алгоритм

```
Naive-Z-Values (S, zp)
{
   n = strlen (S); zp[0] = 0;
   for  (i = 1; i < n; ++i)
   {
      j = i;
      while  (j  <  n && S[j] == S[j - i])  ++j;
      zp[i] = j - i;
   }
}
```

Свойства *Z*-блоков и их массива

- Позиция *i > 0* строки *S может быть* покрыта *Z*-блоками, начинающимися в позициях *j* ($0 \le j \le i$)
- r_i – позиция за правым концом всех *Z*-блоков, начинающихся не позднее *i*, то есть $r_i = \max\limits_{0 \le j \le i}(j + Z_j)$
- l_i – начальная позиция блока, соответствующего r_i, то есть любое *j*, на котором достигается *max*
- Таким образом, l_i и r_i – левая позиция и правая граница *Z*-блока, покрывающего *i*, если такие блоки есть
- Наличие *Z*-блока, покрывающего позицию *i* (на рисунке ниже – α), экономит сравнения, поскольку в *S* есть равный ему префикс, обработанный ранее.

Наличие *Z*-блока



- Предлагается эффективный алгоритм, последовательно вычисляющий *Z*-значения для *i* = 1, 2, …, *n*-1 ($Z_0 = 0$).
- $Z_0, Z_1, \ldots, Z_{i-1}$, а также l_{i-1} и r_{i-1}, используются для вычисления очередных Z_i, l_i и r_i
- Каждый раз требуются лишь последние значения l_{i-1} и r_{i-1}, поэтому достаточно хранить их как *l* и *r*

Вычисление очередных Z_i, *l*, *r*

(1) Случай $i \geq r$:

- Позиция *i* не покрыта ни одним *Z*-блоком
- «Честно» вычисляются Z_i, *l* и *r* : посимвольно сравниваются подстроки с позиций *i* и 0
- Длина совпадающей части дает Z_i, при этом полагают $l = i$, $r = l + Z_i$.

Вычисление очередных Z_i, *l*, *r*

(2) Случай *i* < *r*:

- Позиция *i* содержится в *Z*-блоке *S*[*l*..*r*-1] (α), совпадающем с префиксом *S*[0..Z_l-1]
- *S*[*i*] = *S*[*j*], где *j* = *i* - *l*. Подстрока *S*[*i*..*r*-1] (β) совпадает с *S*[*j*..Z_l-1].
- *Z*-блок в позиции *j* вычислен ранее (его длина – Z_j)
- Подстрока в позиции *i* совпадает с префиксом строки *S* длины min(Z_j, *r* - *i*).

Вычисление очередных Z_i, *l*, *r*

(2a) Случай *i* < *r*, Z_j < *r* - *i*:

- Вычисленный ранее *Z*-блок в позиции *j*, совпадающий с префиксом *S*, целиком (вместе с границей) содержится в β,
- Можно сразу положить Z_i = Z_j
- Значения *r* и *l* не меняются

Вычисление очередных Z_i, *l*, *r*

(2b) Случай *i* < *r*, $Z_j \geq r - i$:

- *Z*-блок в позиции *j* выходит за границу β (или примыкает к ней)
- Вся подстрока β гарантированно совпадает с префиксом *S*
- Для вычисления Z_i необходима проверка символов справа от β: сравниваются символы с позиции *r* и символы с позиции *r* - *i*.
- Пусть несовпадение случилось в позиции $k \geq r$. Тогда следует положить $Z_i = k - i$, $l = i$, $r = k$.

Алгоритм вычисления массива *Z*-значений

```
Prefix-Z-Values (S, zp)
{
  n = strlen (S); l = r = 0; zp[0] = 0;
  for  (i = 1; i < n; ++i)
  {
    zp[i] = 0;
    if (i >= r)
    { // Позиция i не покрыта Z-блоком – он вычисляется заново
      zp[i] = StrComp (S, n, 0, i); l = i; r = l + zp[i];
    }
```

Алгоритм вычисления массива *Z*-значений

```
    else
    { // Позиция i покрыта Z-блоком – он используется
      j = i – l;
      if (zp[j] < r - i) zp[i] = zp[j];
      else
      {
        zp[i] = r - i + StrComp (S, n, r - i, r); l = i; r = l + zp[i];
      }
    }
  }
}
```

Наибольшая длина совпадающих подстрок

StrComp (*S*, *n*, *i*1, *i*2)

```
{
   eqLen = 0;
   while  (i1  <  n && i2  <  n  && S[i1++] == S[i2++])  ++eqLen;
   return eqLen;
}
```

Сложность вычисления массива *Z*-значений

- Цикл *for* повторяется n-1 раз, что при отсутствии вложенных циклов составило бы $\Theta(n)$.
- *StrComp*() содержит цикл *while*, что угрожает квадратичной сложностью, однако оценим его общее число выполнений.
- Оно соответствует числу всех сравнений символов, взятых, например, из правых сравниваемых подстрок.
- Первое же «неудачное» сравнение прерывает цикл *while*, поэтому число всех таких сравнений оценивается как $O(n)$.
- Каждое «положительное» сравнение увеличивает *неубывающее* r. Поскольку всегда $0 \le r \le n$, то число таких увеличений не может превысить n.
- Таким образом, общее число повторений тела цикла *while* оценивается через $O(n)$, и рассматриваемый алгоритм имеет сложность $\Theta(n)$.

Массив *Z*-значений для суффиксов

- Массив *zp* содержит длины *Z*-блоков, начинающихся в данной позиции и совпадающих с *префиксом* строки *S.*
- Симметричная задача: вычисление массива *zs*, *i*-й элемент которого содержит длину максимальной подстроки, которая завершается в позиции *i* строки *S* и совпадает с ее суффиксом.
- Способ решения: переписать строку *S* в обратном порядке, построить массив *zp*, который в обратном порядке переписать в *zs.*
- Приведем непосредственный алгоритм, сложность аналогичная – $\Theta(n)$.

Вычисление массива *Z*-значений суффиксов

```
Suffix-Z-Values (S, zs)
{
  n = strlen (S); l = r = n-1; zs[n-1] = 0;
  for  (i = n - 2; i >= 0; --i)
  {
    zs[i] = 0;
    if (i <= l)
    { // Позиция i не покрыта Z-блоком – он вычисляется заново
      zs[i] = StrCompBack (S, i, n-1); r = i; l = r - zs[i];
    }
```

Вычисление массива Z-значений суффиксов

```
        else
        { // Позиция i покрыта Z-блоком – он используется
           j = n - (r + 1 - i);
          if (zs[j] < i - l)  zs[i] = zs[j];
          else
          {
            zs[i] = i - l + StrCompBack (S, l, n - i + l); r = i;  l = r - zs[i];
          }
        }
    }
}
```

Длина совпадающих обратных подстрок

```
StrCompBack (S, i1, i2)
{
   eqLen = 0;
   while  (i1  >=  0 && i2  >=  0  && S[i1--] == S[i2--])  ++eqLen;
   return eqLen;
}
```

Применение: поиск вхождений образца

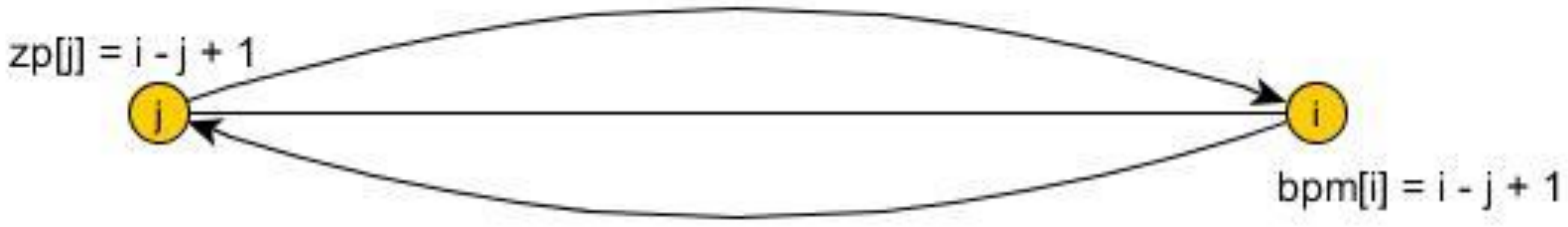
- Эффективный поиск вхождений образца *P* в текст *T*
  - Взять символ # не из алфавита *A* и построить строку *S* = *P#T*
  - Для строки *S* выполнить алгоритм построения массива *zp*
  - В *zp* найти все значения = *m* (длине *P*). Индекс *i* каждого из них (уменьшенный на *m*+1) укажет координату вхождения *P* в *T*.
- Симметричный способ
  - Построить строку *T#P* и для нее вычислить массив *zs*
  - В *zs* найти все элементы = длине *P.* Их индексы соответствуют правым координатам вхождений *P* в *T.*
- Сложность обоих алгоритмов – $\Theta(m+n)$
- В массивах достаточно хранить лишь *m элементов*

Две схемы – массивы граней и *Z*-блоков

- Обе описывают структуру строк
- Взаимосвязаны: содержат длины подстрок в *S*, совпадающих с префиксами *S*
- Основное отличие: в *zp* отмечаются начальные индексы таких подстрок, а в *bp* – конечные
- Вопрос о взаимном преобразовании

Массив *Z*-блоков в массив граней

- *Определение.* В строке *S* позиция *j* *отображается* в позицию *i* ($j \to i$), если $zp[j] > 0$ и $i = j + zp[j] - 1$.
- Очевидно, что $j \to i$, если *Z*-блок, начинающийся в позиции *j*, заканчивается в позиции $i + 1$. Таких *j* может быть несколько.
- *Теорема 1*. Для $\forall\ i > 0 : bpm[i] = zp[j] = i - j + 1$, где *j* – наименьшая из позиций, отображаемых в *i*. Если таких *j* нет, то $bpm[i] = 0$.



Теорема 1

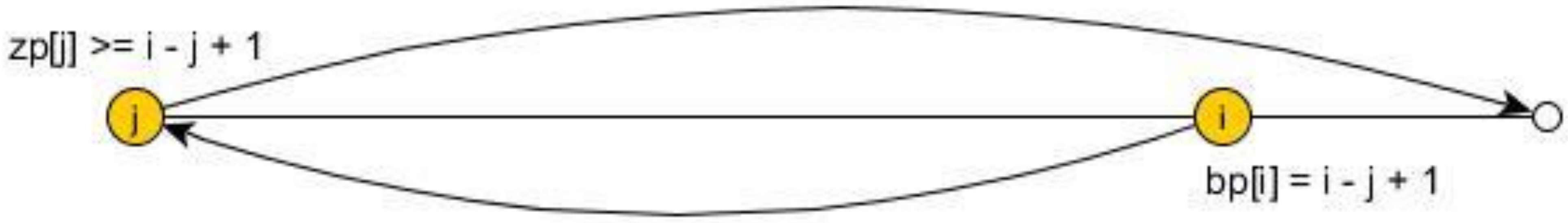
- Дает алгоритм вычисления массива *bpm* по массиву *zp*
- Алгоритм не обращается к исходной строке *S*
- Линейная сложность
- Просмотр – с конца массива, поэтому остается результат от наименьшего возможного *j*

Алгоритм: *zp в bpm*

```
ZP-to-BPM (zp, bpm, n)
{
   for  (i = 0; i < n; ++i)  bpm[i] = 0; // Инициализация
   for  (j = n - 1; j; --j)
   {
      i  = j + zp[j] - 1; bpm[i] = zp[j];
    }
}
```

Вычисление *bp* по *zp*

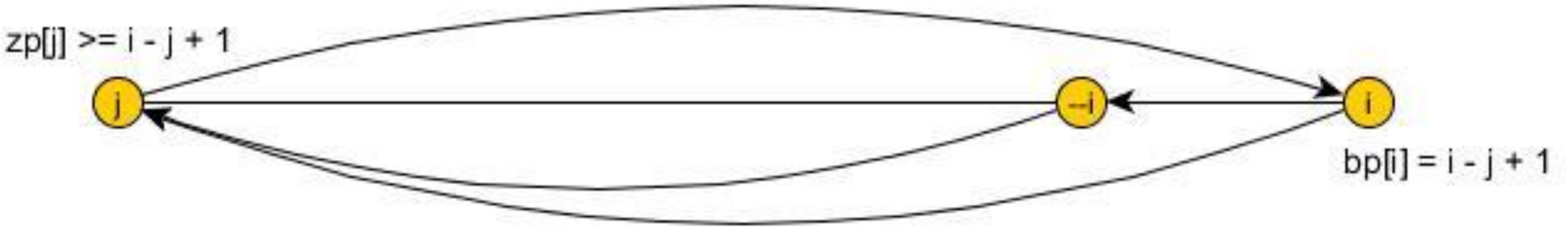
- Может быть использован алгоритм *BPM-to-BP.*
- Рассматривается вопрос о непосредственном вычислении *bp.*
- *Теорема 2*. Для ∀ $i > 0$: $bp[i] = i - j + 1$, где j – наименьшая из позиций, отображаемых в *i или дальше*. Если их нет, то $bp[i] = 0$.



- Замечание: величина $bp[i] = i - j + 1$ не зависит от разности $zp[j] - bp[i]$.

Теорема 2

- Дает алгоритм вычисления массива *bp* по массиву *zp.*
- Просматривая *zp* слева направо, для каждого *j* и *zp*[*j*] > *0* получаем позицию *i* = *j* + *zp*[*j*] - 1.



- Начиная с *i*, двигаясь обратно к *j*, для всех *bp*[*i*] = 0 записываем *bp*[*i*] = *i* - *j* + 1 и уменьшаем *i* на 1, т.е. фактически строим серию.
- При *bp*[*i*] ≠ 0 возврат прерывается, поскольку оно получено из меньшего *j*, имеющего по теореме 2 более высокий приоритет.

Алгоритм: *zp в bp*

```
ZP-to-BP (zp, bp, n)
{
  for  (i = 0; i < n; ++i)  bp[i] = 0; // Инициализация
  for  (j = 1; j < n; ++j)
    for  (i = j + zp[j] - 1; i >= j; --i)
    {
      if (bp[i]) break;
      bp[i] = i - j + 1;
    }
}
```

Сложность алгоритма *ZP-to-BP*

- Вложенный цикл угрожает квадратичной сложностью, однако оценим его общее число выполнений.
- Оно не превосходит количества срабатываний оператора *break* плюс количество присваиваний элементам *bp.*
- Каждое выполнение *break* продвигает счетчик внешнего цикла (это возможно не более $O(n)$ раз).
- Присваивание каждому элементу производится не более одного раза (всего в итоге – $O(n)$).
- Итого: алгоритм в целом имеет вычислительную сложность $\Theta(n)$.

Массив граней в массив Z-блоков

- Основа – алгоритм *Prefix-Z-Values*

- Вместо сравнений символов строки *S* используется массив *bp*
- Массив *bp* состоит из *серий* возрастающих на 1 целочисленных подпоследовательностей, разделенных нулями

Примеры

- *S*: C A C Z Z Z C A C A

  bp: 0 0 1 0 0 0 1 2 3 2

- *S*: A B X A B Z M A B X A B Z

  bp: 0 0 0 1 2 0 0 1 2 3 4 5 6

Проверка серий вместо сравнения

- Сравнение префикса S с подстрокой в позиции i $\sim$ длина серии в bp, стартующей со значения 1 в позиции i
- Сравнение части префикса в позиции j с подстрокой в позиции i $\sim$ длина серии, стартующей со значения j+1 в позиции i
- Замена функции *StrComp* (S, n, $i1$, $i2$) на *ValGrow* (bp, n, i, v) – проверка на серию в позиции i, начинающуюся со значения v
- Аналогично – линейная сложность

Алгоритм: *bp в zp*

```
BP-to-ZP (bp, zp, n)
{
   l = r = 0; zp[0] = 0;
   for  (i = 1; i < n; ++i)
   {
      zp[i] = 0;
      if (i >= r)
      { // Позиция i не покрыта Z-блоком – он вычисляется заново
         zp[i] = ValGrow (bp, n, i, 1); l = i; r = l + zp[i];
      }
```

Алгоритм: *bp в zp*

```
      else
      { // Позиция i покрыта Z-блоком – он используется
         j = i - l;
        if (zp[j] < r - i) zp[i] = zp[j];
        else
        {
          zp[i] = r - i + ValGrow (bp, n, r, r - i + 1); l = i; r = l + zp[i];
        }
      }
    }
  }
```

Проверка серии в заданной позиции

```
ValGrow (nArr, n, nPos, nVal)
{
   nSeqLen = 0;
   while  (nPos  <  n && nArr[nPos++] == nVal++)  ++nSeqLen;
   return nSeqLen;
}
```

Вывод

- Массивы граней и *Z*-значений имеют равные выразительные возможности: их взаимное преобразование может быть осуществлено без исследования символов исходной строки.