**Описание идеи и реализации алгоритма Бойера-Мура**

Прежде чем переходить к описанию алгоритма, определим формат строковых данных, с которыми будем работать. Для реализации возможности работы с произвольным алфавитом реализован модуль abc.cpp. В данном модуле присутствует функция int abc\_set(char \*s), которая принимает на входе строку с перечислением всех символов используемого алфавита и задаёт соответствие между кодировкой ASCII и внутренней кодировкой, используемой алгоритмом. Неапример, если подать на вход строку “ABXYZ”, то внутренний код «A» будет равен 0, «B» - 1, «X» - 2 и тд…

Строки в алгоритме представлены массивами целых чисел. Преобразование из строки в массив и обратно осуществляются с помощью функций int abc\_str\_to\_code( char\* src, int\*\* dst ) и void abc\_code\_to\_str( int\* src, char\*\* dst, int len ) соответственно. Внедрение своей кодировки не является необходимым, но позволяет сделать алгоритм более универсальным и «чистым», а также не формировать таблицу плохих символов для всех символов ASCII.

На рисунке 1 изображено представление входных данных алгоритма Бойера-Мура. Имеется строка str, длинны str\_len и шаблон tem длинны tem\_len. Алгоритм ищет все вхождения шаблона в строку. Буквой p обозначено текущее смещение шаблона относительно строки.

C

str:

B

A

A

B

C

A

C

A

A

B

C

A

tem:

str\_len

tem\_len

p

*Рисунок 1. Описание исходных данных*

Начнём с того, как работает самый простой алгоритм поиска подстроки в строке. Сначала мы совмещаем начала строки с началом шаблона (p=0). Затем в цикле, начиная с первого символа шаблона и до последнего сравниваем соответствующие символы строки и шаблона. Если все символы совпали, значит подстрока найдена. После того, как мы нашали или не нашли совпадение шаблона со строкой, смещаем шаблон на одну позицию вправо (p++) и повторяем операцию до тех пор, пока правая граница шаблона не достигнет конца строки.

Теперь немного изменим этот алгоритм. Шаблон будем всё так-же двигать слева на право, а сравнивать символы шаблона со строкой справа налево (от последнего к первому). Это никак не повлияет на работу алгоритма. Тем самым мы получили заготовку для Алгоритм Бойера-Мура.

Алгоритм Бойера-Мура использует две эвристические функции, которые позволяют в некоторых случаях смещать шаблон более, чем на один шаг вправо, что экономит вычислительные ресурсы. Эти функции независимы и могут использоваться по отдельности или совместно. Каждая из этих функций выдаёт минимальное безопасное смещение шаблона вправо, такое, что не будет пропущено не одно вхождение шаблона в строку. Если функции используются совместно, то выбирается наибольшее значение смещения.

Сначала рассмотрим функцию плохого символа. Представим ситуацию, как на рисунке 2. При p=1 мы начали сравнивать строку с шаблонам с последнего символа и сразу нашли несовпадение. Это означает, что шаблон не совпадает со строкой при p=1. По идее мы должны сдвинуть шаблон на один шаг вправо и начать следующее сравнение, НО мы видим, что в исходной строке на месте не совпавшего символа стоит «С», а в шаблоне последний и предпоследний символ не равны «С». Это значит, что и при p=2 шаблон не будет совпадать со строкой. Первое совпадение возможно только в том случае, если на месте последнего символа строки окажется «С», а это произойдет только тогда, когда шаблон сдвинется на две позиции вправо, поскольку это пододвинет последнее вхождение символа «С» шаблона к последнему символу строки также равному «С».

A

str:

B

D

D

A

C

A

B

С

B

C

A

B

tem:

str\_len

tem\_len

p

*Рисунок 2. Плохой символ*

Мы можем заранее определить для каждого символа алфавита на каком смещении от конца шаблона он встречается первый раз. Для символа «C», это значение равно 2. Теперь каждый раз, когда мы встретим в качестве последнего символа строки, символ «С», мы сможем сместить шаблон на две позиции вправо, поскольку для совпадения последний символ строки «С» должен оказаться над символом «С» шаблона. Если мы сместим не до последнего вхождения «С» в шаблон, а до первого, то есть риск пропустить одно из вхождений. Определим значения сдвига для всех символов алфавита, рисунок 3. Обратим внимание на две детали. Мы не учитываем последний символ «B» - это не имеет смысла, поскольку если последний символ строки равен «B», то полученное смещение равно 0. Мы уже проверили совпадение шаблона со строкой на позиции p и вне зависимости от результата нет смысла оставаться на месте. Именно поэтому мы игнорируем последний символ и используем смещение предпоследнего. Во вторых в шаблоне отсутствует символ «D». Ну тут всё просто. Раз этого символа нету в шаблоне, то можно сдвинуть шаблон на всю его длину. Логика в том, что если мы встретили символ, которого не было в шаблоне, то пока последний символ строки будет стоять над шаблоном, вхождения не будет и нужно сдвинуть шаблон на всю его длину.

С

B

C

A

B

tem:

A

bad table (BT):

1

B

3

C

2

D

5

*Рисунок 3. Bad table*

С этим разобрались. Но что будет, если не совпавший символ будет не первым, а как показано на рисунке 4?