Алгоритмы на строках
Часть 2

сборы минской области по информатике, февраль 2021

План

Проверка скобочных выражений

Вычисление арифметических выражений

Алгоритм Ахо-Корасик

Использованные источники

- [1] Habr - Электронный ресурс: Алгоритм Ахо-Корасик. Режим доступа: https://habr.com/ru/post/198682/.
- [2] MAXimal - Электронный ресурс: Алгоритм Ахо-Корасик. Режим доступа: https://e-maxx.ru/algo/aho_corasick.

Проверка скобочных выражений

Постановка задачи

- Дана последовательность из N круглых скобок. Выяснить, можно ли добавить в неё цифры и знаки арифметических действий так, чтобы получилось правильное арифметическое выражение.
- Правильная скобочная последовательность (англг. Correct Bracket Sequences) — частный случай скобочной последовательности, определяющийся следующими образами:
- Пустая строка — есть правильная скобочная последовательность;
- Пусть S — правильная скобочная последовательность, тогда (S) — правильная скобочная последовательность;
- Пусть S_1, S_2 — правильные скобочные последовательности, тогда $S_1 S_2$ есть правильная скобочная последовательность;

Постановка задачи

Примеры правильных скобочных последовательностей:

(())()

()

(((())))

Примеры НЕправильных скобочных последовательностей:

)()()

()(

((((

Решение

- Пусть $depth$ — это текущее количество открытых скобок. Изначально $depth = 0$.
- Будем двигаться по строке слева направо, если текущая скобка открывающая, то увеличим $depth$ на единицу, иначе уменьшим.
- Если при этом когда-то получалось отрицательное число, или в конце работы алгоритма $depth$ отлично от нуля, то данная строка не является правильной скобочной последовательностью, иначе является.

Решение

- Если допустимы скобки нескольких типов, то алгоритм нужно изменить.
- Вместо счётчика следует создать стек, в который будем класть открывающие скобки по мере поступления. Если текущий символ строки — открывающая скобка, то кладём его в стек, а если закрывающая — то проверяем, что стек не пуст, и что на его вершине лежит скобка того же типа, что и текущая, и затем достаём эту скобку из стека.
- Если какое-либо из условий не выполнилось, или в конце работы алгоритма стек остался не пуст, то последовательность не является правильной скобочной, иначе является.

Вычисление арифметических выражений

Постановка задачи

- Дана строка, представляющая собой математическое выражение, содержащее числа, скобки, различные операции. Требуется вычислить его значение за O (n), где n — длина строки.

Приоритет операций

- Операции в порядке убывания приоритета:
- () скобки
- * / деление и умножение
- + - сложение и вычитание
- В случае равных приоритетов операции выполняются по порядку, слева направо.

Решение задачи

- Для решения нам необходимо два стека – стек операций O, стек чисел N.
- Идем по строке S содержащей выражение, если встретили число – помещаем его в стек чисел.
- Если встретили оператор:
- В случае если последний оператор в стеке имеет больший приоритет чем рассматриваемый оператор то помещаем операцию в стек.
- Прежде чем добавить операцию с меньшим или равным приоритетом мы должны выполнить все операции с большими приоритетом (по определению приоритета).

Решение задачи

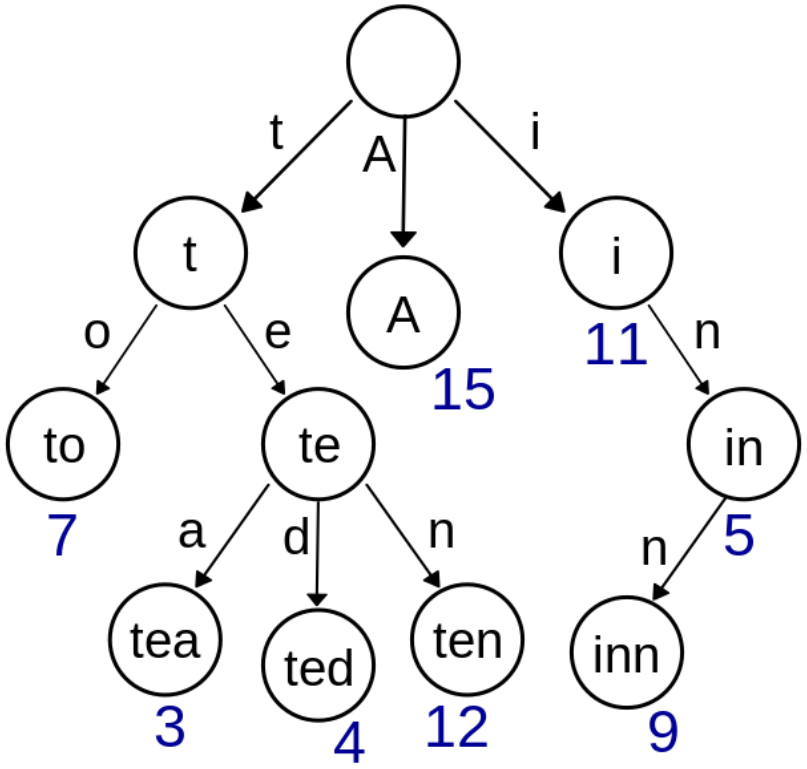
- Для выполнения операции необходимо достать оператор из стека операций, достать два числа из стека чисел, произвести соответствующую операцию между двумя числами, результат поместить в стек чисел.
- Открывающая скобка просто помещается в стек.
- При встрече закрывающей скобки мы должны выполнить все операции из стека, пока не достанем из стека открывающую скобку.
- После окончания парсинга строки необходимо выполнить операции пока стек операции не пуст.
- Результат вычисления будет храниться в стеке чисел.

Алгоритм Ахо-Корасик

Предисловие

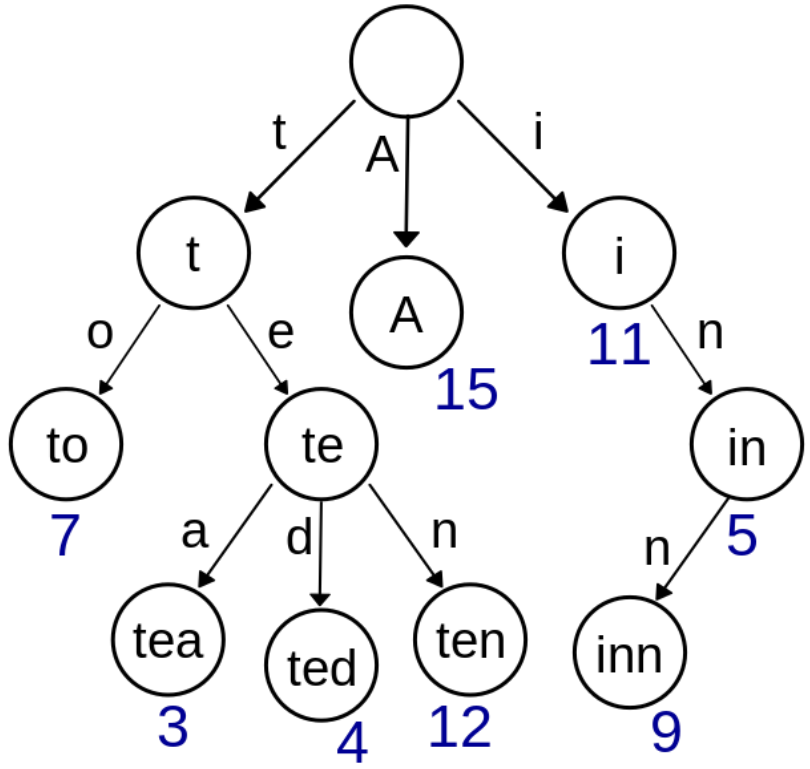
Бор

- Префиксное дерево, бор, луч, нагруженное дерево — структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах.
- Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной.
- Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.



Бор

- Рассмотрим в боре любой путь из корня:
- Выпишем подряд метки рёбер этого пути. В результате мы получим некоторую строку, которая соответствует этому пути. Если же мы рассмотрим любую вершину бора, то ей поставим в соответствие строку, соответствующую пути из корня до этой вершины.



Построение бора

- Каждая вершина бора также имеет флаг $leaf$, который равен $true$, если в этой вершине оканчивается какая-либо строка из данного набора.
- Соответственно, построить бор по данному набору строк — значит построить такой бор, что каждой $leaf$-вершине будет соответствовать какая-либо строка из набора, и, наоборот, каждой строке из набора будет соответствовать какая-то $leaf$-вершина.
- Бор по набору строк строится за линейное время относительно их суммарной длины.

Построение бора

- Структура узла:

```
struct node {
   node* next[p];
   bool leaf;
};
```

- Где `next` – массив указателей на вершины соответствующих символов или `nullptr` если таких строк нет.
- `p` – количество различных символов в строке (26 для латинских символов).
- `leaf` – показывает есть ли строка заканчивающаяся в этой вершине.
- Изначально бор состоит только из одной вершины – корня.

Построение бора

Рассмотрим функцию которая будет добавлять строку S в бор.

Начинаем в корне бора, проверяем, есть ли из корня переход по букве $S[0]$:

если переход есть, то просто переходим по нему в другую вершину, иначе создаём новую вершину и добавляем переход в эту вершину по букве $S[0]$. Затем мы, стоя в какой-то вершине, повторяем процесс для буквы $S[1]$, и т.д. После окончания процесса помечаем последнюю посещённую вершину флагом $leaf = true$.

```
void add(const string& s){
   node* ptr = root;
   for (int i = 0; i < s.length(); i++) {
      char c = s[i] - 'a';
      if (ptr->next[c] == nullptr)
         ptr->next[c] = new node;
      ptr = ptr->next[c];
   }
   ptr->leaf = true;
}
```

Постановка задачи

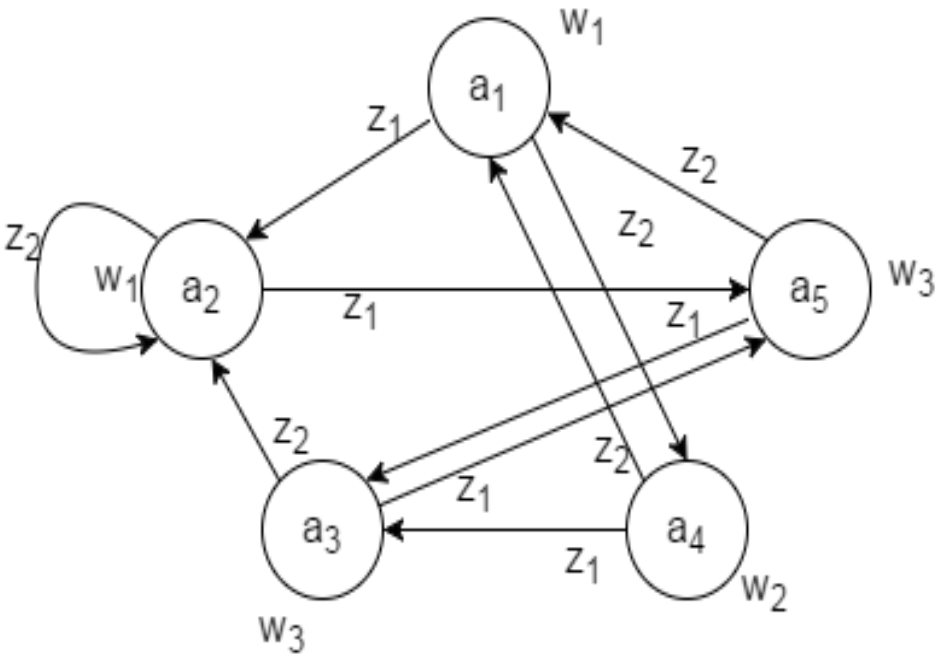
- Пусть дан набор из m строк размера k, необходимо найти все вхождения строк из набора в тексте t размера n.
- Асимптотика тривиального поиска составит $O(MNK)$.
- Ускорение с помощью префиксной функции (Z-функция или алгоритм КМП) позволит достичь асимптотики $O\ (M(N+K))$. Хэширование даст аналогичную асимптотику.
- Построение паттерна – конечного автомата Ахо-Корасик - позволяет решить задачу за линейное время $O(MK+N)$.

Пример задачи

- Алгоритм Ахо-Корасик применятся для быстрого поиска набора строк в тексте.
- Например поиск вирусов в файлах и программах по базе данных вирусных сигнатур представленных в виде автомата Ахо-Корасик.

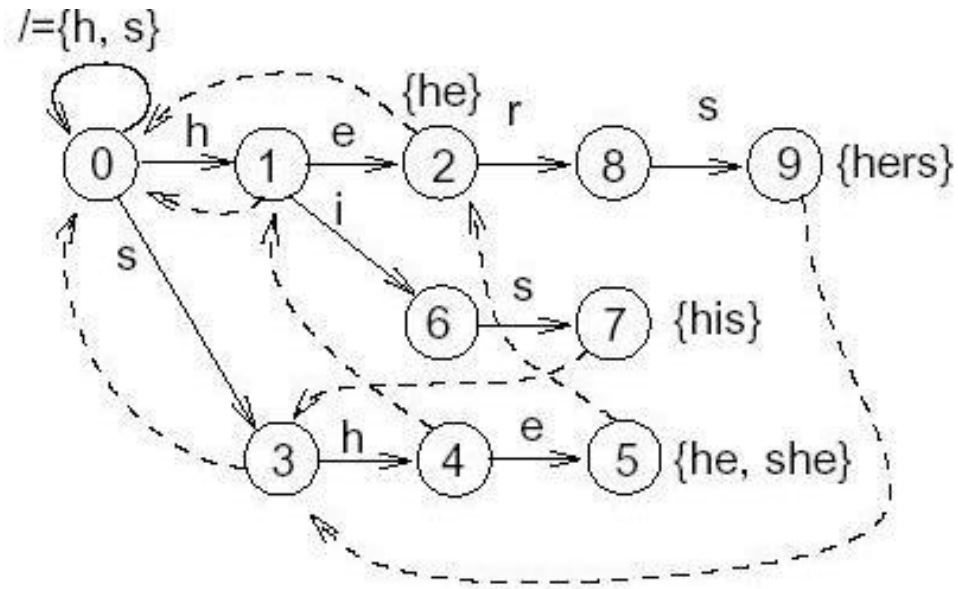
Конечный автомат

- Коне́чный автома́т (КА) — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных.
- Детерминированные КА — автоматы, в которых следующее состояние однозначно определяется текущим состоянием и выход зависит только от текущего состояния и текущего входа



Построение автомата Ахо-корасик

- Вершины бора можно понимать как состояния конечного детерминированного автомата.
- Находясь в каком-либо состоянии, мы под воздействием какой-то входной буквы переходим в другое состояние — т.е. в другую позицию в наборе строк. Например, если в боре (Рисунок) мы стоим в состоянии 4 (которому соответствует строка sh), то под воздействием буквы e мы перейдём в состояние 5 (she).

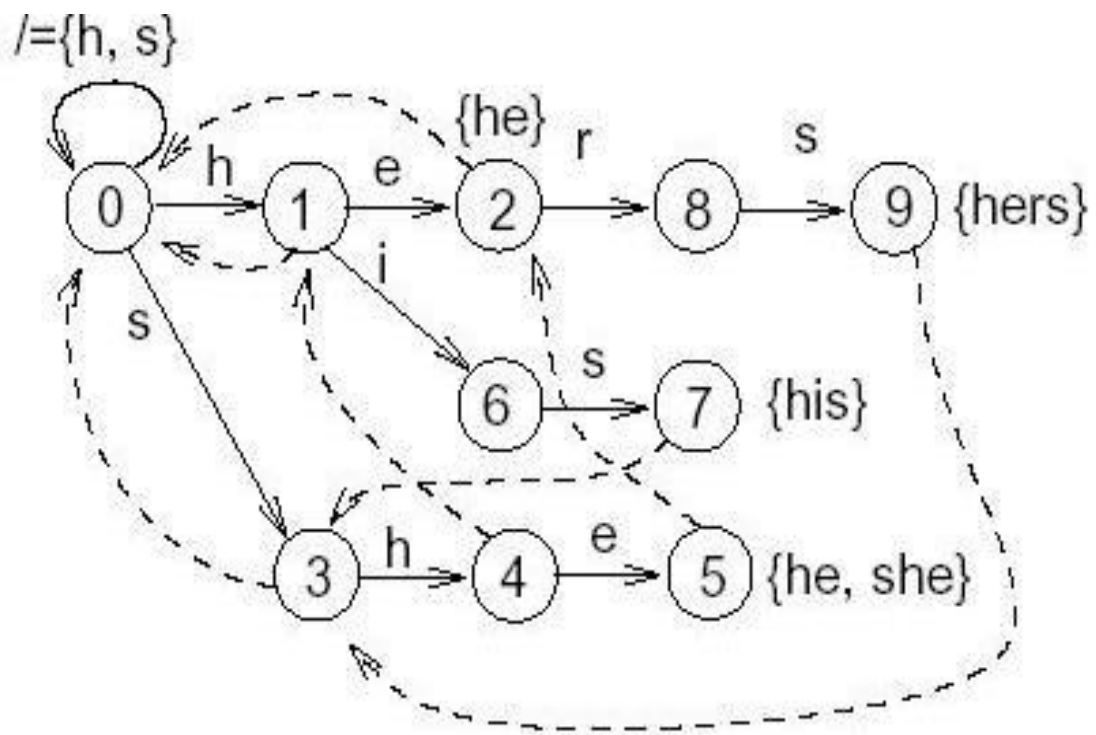


Построение автомата Ахо-корасик

- Если мы пытаемся выполнить переход по какой-либо букве, а соответствующего ребра в боре нет, то мы тем не менее должны перейти в какое-то состояние.
- Пусть мы находимся в состоянии p, которому соответствует некоторая строка t, и хотим выполнить переход по символу c. Если в боре из вершины p есть переход по букве c, то мы просто переходим по этому ребру. Если же такого ребра нет, то мы должны найти состояние, соответствующее наидлиннейшему собственному суффиксу строки t и выполнить переход по букве c из него.

Построение автомата Ахо-Корасик

- Например, под воздействием строки she мы перешли в состояние 5, являющееся листом. Тогда под воздействием буквы r мы вынуждены перейти в состояние 2, соответствующее строке he, и только оттуда выполнить переход по букве r.



Построение автомата Ахо-Корасик

- **Суффиксная ссылка** для каждой вершины v — это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине v. Единственный особый случай — корень бора; для удобства суффиксную ссылку из него проведём в себя же.
- Теперь мы можем переформулировать утверждение по поводу переходов в автомате так: пока из текущей вершины бора нет перехода по соответствующей букве (или пока мы не придём в корень бора), мы должны переходить по суффиксной ссылке.

Построение автомата Ахо-Корасик

- Заметим, что если мы хотим узнать суффиксную ссылку для некоторой вершины v, то мы можем перейти в предка p текущей вершины (пусть c — буква, по которой из p есть переход в v), затем перейти по его суффиксной ссылке, а затем из неё выполнить переход в автомате по букве c.
- Таким образом, задача нахождения перехода свелась к задаче нахождения суффиксной ссылки, а задача нахождения суффиксной ссылки — к задаче нахождения суффиксной ссылки и перехода, но уже для более близких к корню вершин. Мы получили рекурсивную зависимость, но не бесконечную, и, более того, разрешить которую можно за линейное время.

Построение автомата Ахо-Корасик

- Для каждой вершины необходимо хранить её предка p, а также символ ch, по которому из предка есть переход в нашу вершину. Также в каждой вершине будем хранить $node^* link$ — суффиксная ссылка (или $nullptr$, если она ещё не вычислена), и массив $node^* go[k]$ — переходы в автомате по каждому из символов (опять же, если элемент массива равен $nullptr$, то он ещё не вычислен).

Поиск набора строк в тексте

- Построим по данному набору строк бор. Будем теперь обрабатывать текст по одной букве, перемещаясь— по состояниям автомата. Изначально мы находимся в корне дерева.
- Пусть мы на очередном шаге мы находимся в состоянии v, и очередная буква текста c. Тогда следует переходить в состояние $go(v, c)$, тем самым либо увеличивая на 1 длину текущей совпадающей подстроки, либо уменьшая её, проходя по суффиксной ссылке.
- Если мы стоим в помеченной вершине ($leaf = true$), то имеется совпадение с тем образцом, который в боре оканчивается в вершине v.

Поиск набора строк в тексте

- Однако это далеко не единственный возможный случай достижения совпадения: если мы, двигаясь по суффиксным ссылкам, можем достигнуть одной или нескольких помеченных вершин, то совпадение также будет, но уже для образцов, оканчивающихся в этих состояниях.
- Простой пример такой ситуации — когда набор строк — это $\{dbac, abc, bc\}$, а текст — это $dabc$.
- Таким образом, за $O(n)$ можем найти номера всех образцов, для которых достигнуто совпадение, просто пройдя по суффиксным ссылкам от текущей вершины до корня. Однако это недостаточно эффективное решение, поскольку в сумме асимптотика получится $O(n\ Len)$.

Поиск набора строк в тексте

- Можно заметить, что движение по суффиксным ссылкам можно оптимизировать, предварительно посчитав для каждой вершины ближайшую к ней помеченную вершину, достижимую по суффиксным ссылкам (это называется "функцией выхода").
- Эту величину можно считать ленивой динамикой за линейное время. Тогда для текущей вершины мы сможем за $O(1)$ находить следующую в суффиксном пути помеченную вершину, т.е. следующее совпадение. Тем самым, на каждое совпадение будет тратиться $O(1)$ действий, и в сумме получится асимптотика $O(Len + Ans)$.
- Тогда $leaf\_link$ — это ближайший суффикс, имеющийся в боре, для которого $leaf = true$.

Использованные источники

- [1] Habr - Электронный ресурс: Алгоритм Ахо-Корасик. Режим доступа: https://habr.com/ru/post/198682/.
- [2] MAXimal - Электронный ресурс: Алгоритм Ахо-Корасик. Режим доступа: https://e-maxx.ru/algo/aho_corasick.