Компактное представление автомата Ахо-Корасик

Никита Сивухин

25.09.2017

1 RAM модель

Все алгоритмы анализируются в рамках стандартной `RAM`-модели с размером машинного слова $\omega = \Omega(\log N)$, где N — суммарный размер входа в битах. Это означает, что доступ на чтение и запись к последовательным $O(\omega)$ битам оперативной памяти осуществляется за константное время, а также базовые арифметические и логические операции над последовательными $O(\omega)$ битами ($O(1)$ машинными словами) также осуществляется за константное время.

2 Введение в задачу

Задача поиска вхождений фиксированного набора слов в текст является одной из ключевых задач стрингологии. Алгоритмы, эффективно решающие эту задау, применяются в различных областях: биоинформатика, разработка движков полнотекстового поиска, разработка сканеров вирусов и т.п.

Формализуем условие задачи, которую мы хотим решить

Задача 1 (Dictionary matching problem)**.** Задано множество слов $D = \{s_1, s_2, \cdots s_d\}$ над алфавитом A размером σ. Нужно для произвольного текста T эффективно находить все вхождения слов из D в данный текст.

Существует два распространенных подхода к решению этой задачи: адаптация алгоритма Рабина-Карпа, использующая хэширование и построение автомата Ахо-Корасик. Мы рассмотрим второй способ и его возможные улучшения.

3 Автомат Ахо-Корасик

Формально автомат Ахо-Корасик можно определить следующим образом

Определение 3.1. Автомат Ахо-Корасик $M = (A, Q, q_0, T, \delta)$, построенный по набору слов D над алфавитом A, обладает следующими свойствами:

- Каждому состоянию $v \in V$ соответствует префикс некоторого слова $w \in D$, который мы будем обозначать как $\mathit{prefix}(v)$
- Начальному состоянию q_0 соответсвует пустая строка
- Множество $\{\mathit{prefix}(t) \mid t \in T\}$ совпадает с множеством D

- Для произвольной строки s определим множество

$$Suff(s) = \{v \mid prefix(v) \text{ суффикс строки } s\}$$

  В таком случае $\delta(q, c) = v$, где v — состояние из $Suff(prefix(q) + c)$, которому соответствует наидлиннейшая строка $prefix(v)$

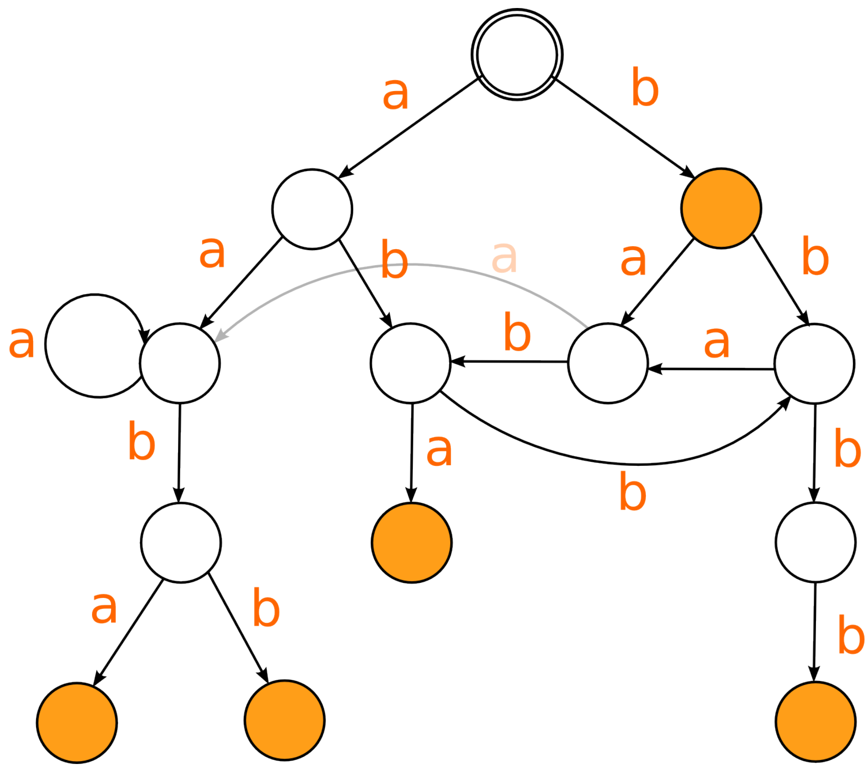

Рис. 1: Автомат Ахо-Корасик для множества строк $D = \{aaba, aabb, aba, b, ba, bbbb\}$ (для глубоких вершин указаны не все переходы)

Несложно показать, что для произвольной строки s автомат закончит свою работу в таком состоянии $v = M(s)$, что $prefix(v)$ — это наидлиннейший суффикс строки s, который присутствует во множестве $\{prefix(v) \mid v \in Q\}$. Тогда алгоритм поиска вхождений строк из D в произвольный текст T выглядит следующим образом:

1. Обрабатываем текст T автоматом, получая последовательность состояний

$$q_0 \to q_1 \cdots \to q_{|T|}$$

2. Для каждого состояния q из последовательности находим множество строк из D, являющихся суффиксом строки $prefix(v)$. Данное множество можно представить как $R(v) = Suff(prefix(v)) \cap T$

Для решения второй задачи потребуется дополнительная структура, построенная над состояниями автомата. Посчитаем для каждого состояния q ссылку $report(q)$ вещущую в такое состояние q', что

- $q' \in T$ или $q' = q_0$

- $prefix(q')$ — суффикс $prefix(q)$

Легко заметить, что строки множества $R(v)$ образуют линейный порядок относительно отношения «быть суффиксом», поэтому с помощью ссылок *report* можно перечислить элементы множества $R(v)$ за время $O(|R(v)|)$ (при перечислении строки множества мы будем использовать соответствующее ей состояние).

Также нам нужно сохранить информацию о терминальных состояниях и длинах строк. Для этого просто сохраним битовый массив $term(q)$, в котором будет $|D|$ единиц, маркирующих терминальные состояния и массив длин терминальных состояний $termLen(q)$.

Оценим количество памяти, требующееся для хранения структуры. Обозначим за n количество состояний в автомате Ахо-Корасик. Видно, что если суммарная длина строк из D равна m, то $n \leq m+1$. На практике данная оценка не достигается и n существенно меньше m, т.к. многие слова имеют общий префикс.

Если явно хранить переходы автомата, то для этого потребуется $O(n\sigma \log n)$ бит памяти. Однако данное ограничение может быть понижено до $O(n \log n)$ бит, если хранить переходы автомата неявно.

Оставим от автомата только каркас — такие переходы $\delta(q,c) = q'$, что $prefix(q') = prefix(q) + c$. Ясно, что данный каркас образует дерево (для каждого состояния $q \neq q_0$ можно однозначно определить предка). Данные список переходов в дальнейшем будем обозначать как

$$trans(q,c) = \begin{cases} \delta(q,c) & \texttt{если } \ prefix(\delta(q,c)) = prefix(q) + c \\ \Lambda & \texttt{иначе} \end{cases}$$

Чтобы иметь возможность восстановить произвольный переход автомата посчитаем суффиксную ссылку для каждого состояния $link(q) = q'$, где q' — состояние из множества $Suff(q) \setminus \{q\}$, имеющее наидлиннейшее значение $prefix(q')$. Несложно показать, что если $trans(q,c) = \Lambda$, то $\delta(q,c) = \delta(link(q),c)$. Пользуясь этим свойством можно неявно восстановить все переходы автомата:

$$\delta'(q,c) = \begin{cases} trans(q,c) & \texttt{если } \ trans(q,c) \neq \Lambda \\ \delta'(link(q),c) & \texttt{иначе} \end{cases}$$

Покажем, что вычисление последовательности состояний для префиксов строки T с помощью рекурсивной процедуры вычисления $\delta'(q,c)$ на самом деле выполняется за $O(|T|)$. Посчитаем отдельно количество переходов по ребрам дерева $trans(q,c)$ и количество рекурсивных переходов к $\delta'(link(v),c)$. Ясно, что переходов по ребрам дерева ровно $|T|$. Заметим тогда, что каждый переход первого типа увеличивает глубину вершины дерева ровно на 1, в то время как каждый рекурсивный переход уменьшает её хотя бы на 1. Значит рекурсивных переходов суммарно не больше чем $|T|$.

Таким образом мы уменьшили память структуры с $O(n\sigma \log n)$ до $O(n \log n)$.

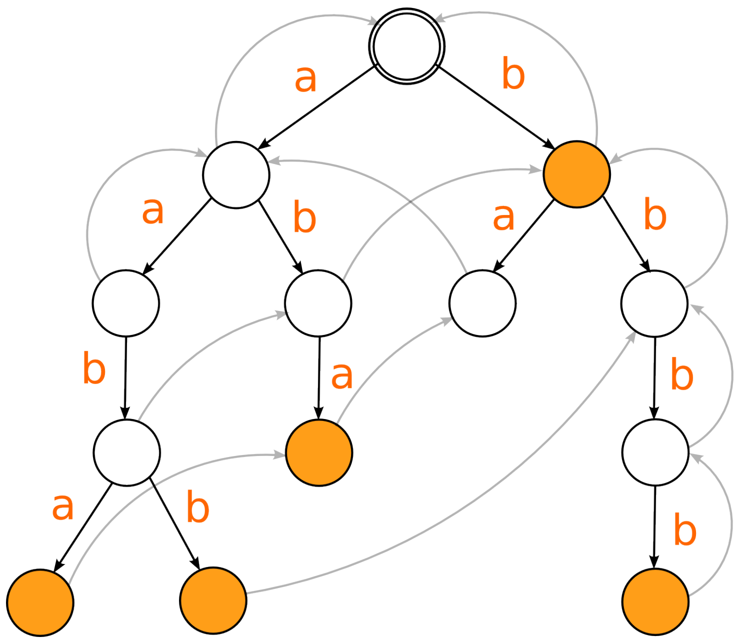

Рис. 2: Переходы *trans* и суффиксные ссылки *link* построенные над состояниями Q автомата

4 Succinct структуры данных

Модное направление `Computer Science`. Объемы данных, которые нужно эффективно обрабатывать, растут и теперь уже константа «под знаком O» становится критичной. Для работы с ними нужны компактные индексы, которые стараются представить структуру данных максимально сжато, при этом сохраняя способность эффективно отвечать на поисковые запросы. Более формально, структура данных для некоторого набора данных называется `succinct`, если она занимает $Z + o(Z)$ бит памяти, где Z - теоретический минимум количества информации необходимый для хранения данных. При этом данная структура сохраняет способность эффективно отвечать на различные запросы.

Пример 4.1. Пусть нам нужно сохранить произвольное подмножество размера k над множеством $[1 \ldots n]$, при этом имея возможность перечислять элементы нашего подмножества за $O(k)$. Одним из вариантов решения этой задачи может являться следующая структура данных:

- Сохраним n, k используя $O(\log n)$ бит
- Перечислим подряд все элементы подмножества в порядке возрастания, используя $k\lceil \log n \rceil \leq k \log n + k$ бит памяти

Лемма 1. *Для любых* $n, k \in \mathbb{N}, k \leq n$ *выполняются неравенства:*

$$k \log \frac{n}{k} \leq \log \binom{n}{k} \leq k \log \frac{n}{k} + k \log e$$

Где $\log$ *— двоичный логарифм*

Оценим теоретический минимум Z пользуясь леммой 1, необходимый для сохранения произвольного подмножества: $Z = \log \binom{n}{k} \leq k \log \frac{n}{k} + k \log e = k \log n - k \log k + k \log e$

Видно, что при $k = o(n)$ наше представление является `succinct`, т.к. $k \log k = o(k \log n)$. Однако при $k = O(n)$ структура перестает быть `succinct`.

Лемма 2. *Существует такая структура данных, хранящая произвольное k-элементое подмножество S над множеством $U = [0 \dots n-1]$, использущая $\log \binom{n}{k} + o(k)$ бит памяти и способная отвечать за константное время на следующие запросы:*

- *$select(i)$ — найти i-ый элемент подмножества в порядке возрастания*
- *$rank(x)$ — посчитать количество элементов подмножества S меньших x, в случае если $x \in S$, либо вернуть Λ в случае, если $x \notin S$*

Мы также будем без доказательства пользоваться следующими результатами:

Лемма 3. *Над любым деревом с n вершинами можно построить такую структура данных, которая использует $2n + o(n)$ бит памяти и позволяет находить предка любой вершины за константное время.*

Лемма 4. *Для произвольного непомеченного упорядоченного корневого дерева на n вершинах с d внутренними вершинами существует структура данных, использующая $d(\log \frac{n}{d} + O(1))$ бит памяти и позволяющая находить предка любой вершины за константное время.*

5 Компактное представление

В статье `Belazzougui` описано компактное представление структур алгоритма Ахо-Корасик, занимающее $n(\log \sigma + 3.443 + o(1)) + |D|(3 \log \frac{m}{|D|} + O(1))$ бит памяти и позволяющее находить все вхождения слов из словаря в текста T за время $O(|T| + occ)$.

Разберем, каким образом `Belazzougui` добился такого успеха.

Ключевой идеей, необходимой для достижения компактности структуры без потери её эффективности, является правильная нумерация вершин бора. Занумеруем вершины бора числами из диапазона $[0 \dots n-1]$ в порядке, соответствующем лексикографическому порядку строк $prefix(v)^r$. Обозначим за $num(v)$ — порядковый номер вершины.

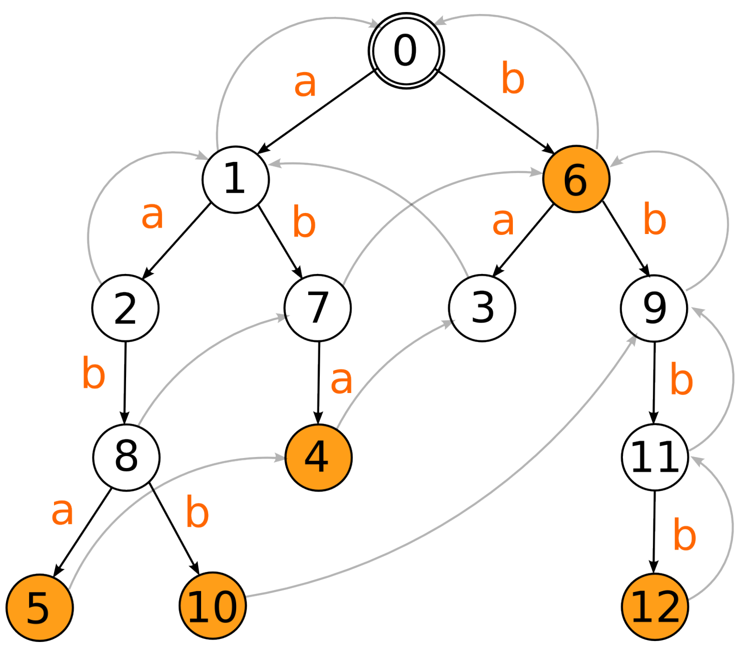

Рис. 3: Нумерация вершин в порядке сортировки относительно строки на пути от вершины к корню

Данный порядок кажется менее естественным, чем обычный порядок относительно сортировки строк $\mathit{prefix}(v)$, однако в рамках работы алгоритма Ахо-Корасика данный порядок совершенно естественен. Посмотрим внимательно на дерево суффиксных ссылок: в данной нумерации вершины дерева пронумерованы в порядке обхода в глубину этого дерева.

5.1 Сжимаем переходы $\boldsymbol{trans}(q, c)$

Поставим также в соответствие каждой вершине бора уникальную пару $\mathit{ord}(v) = (\mathit{char}(v), \mathit{num}(\mathit{parent}(v)))$. $\mathit{char}(v)$ и $\mathit{parent}(v)$ — принимают такие значения, что $\mathit{trans}(\mathit{parent}(v), \mathit{char}(v)) = v$. Для корня q_0 положим $\mathit{char}(v) = \$$ и $\mathit{parent}(q_0) = q_0$.

Чтобы сжать переходы бора заметим, что если $\mathit{trans}(q, c) = q'$, то

$$\mathit{num}(q') = |\{x \mid x \in Q \text{ и } \mathit{ord}(x) < (c, \mathit{num}(q))\}|$$

Видно, что данная операция является операцией rank на множестве пар (c, num). Чтобы иметь возможность воспользоваться леммой 2 представим пару (c, num) как одно число вида $cn + \mathit{num}$. Построим $\mathit{rank}/\mathit{select}$ структуру из леммы 2 и научимся с помощью нее эмулировать переходы по ребрам бора $\mathit{trans}(q, c)$.

Для перехода из состояния, которому соответствует номер $\mathit{num}(q)$ по символу c достаточно посчитать $\mathit{rank}(cn + \mathit{num}(q)) = \mathit{num}(\mathit{trans}(q, c))$.

Таким образом мы представили структуру бора, используя $\log\binom{\sigma n}{n} \leq n(\log\sigma + 1.443 + o(1))$ бит памяти.

5.2 Сжимаем суффиксные ссылки

Т.к. суффиксные ссылки образуют дерево, помеченного в порядке обхода в глубину, то можно воспользоваться леммой 5 и представить его с помощью $2n + o(n)$ бит памяти.

5.3 Сжимаем ссылки $\boldsymbol{report}(q)$

Несложно показать, что ссылки $\mathit{report}(q)$ также образуют дерево с метками соответствующими порядку обхода этого дерева в глубину. Также понятно, что внутренней вершиной в этом дереве может быть только терминальное состояние автомата. Таким образом, внутренних вершин в дереве не больше чем $|D|$. Тогда мы можем воспользоваться леммой 4 и представить дерево с помощью $|D|(\log\frac{n}{|D|} + O(1))$ бит памяти.

5.4 Сжимаем $\boldsymbol{term}$(q)

Т.к. в битовом массиве-индикаторе для терминальных состояний всего $|D|$ единиц, то его можно компактно представить с помощью структуры из леммы 2, используя $\log\binom{n}{|D|} + o(|D|) \leq |D|(\log\frac{n}{|D|} + O(1))$ бит.

5.5 Сжимаем $\boldsymbol{termLen}$(q)

Нам нужно сохранить $|D|$ чисел, сумма которых равна m и уметь получать i-ое число за $O(1)$. Эту задачу можно элегантно решить с помощью кодирования Элиаса-Фано используя всего лишь $|D|\log\frac{m}{|D|} + O(|D|)$ бит памяти.

Таким образом, алгоритм потребляет $n(\log\sigma + 3.443 + o(1)) + |D|(3\log\frac{m}{|D|} + O(1))$ бит памяти.

6 Еще более компактное представление

Хоть представление `Belazzougui` и `succinct`, но в нем все еще можно оптимизировать константы. Можно заняться оптимизированием элемента суммы $1.443n \approx n\log(e)$, однако данное направление связано с улучшением наработок в сфере *rank/select* структур над алфавитами с произвольным размером. Можно попытаться уменьшить затраты на хранения дерева суффиксных ссылок, которые в оригинальном алгоритме составляют $2n$ бит. Мы рассмотрим второе направление оптимизации.

Лемма 5. *Дано непомеченное упорядоченно корневое дерево с n вершинами и подмножество вершин W. Можно построить структуру, занимающую $|W|(\log\frac{n}{|W|}+O(1))+\log n$ бит памяти, способную отвечать на запросы $parent(v)$ для всех $v \in W$, либо сигнализировать, что $v \notin W$, в случае чего $parent(v) = \Lambda$.*

Доказательство. Ключевая идея доказательствоа состоит в том, чтобы заменить исходное дерево на дерево с большим числом листев, для которого значения предков для выделенных вершин такие же, как в оригинальном дереве.

Пронумеруем вершины в порядке обхода в глубину. Назовем вершину важной, если она корень или хотя бы один из её сыновей принадлежит множеству W. Построим новое дерево следующим образом: для каждой вершины v найдем ближайшего важного предка p. Тогда в новом дереве p будет непосредственным родителем вершины v. Также, всех сыновей для внутренних вершин упорядочим согласно их номерам.

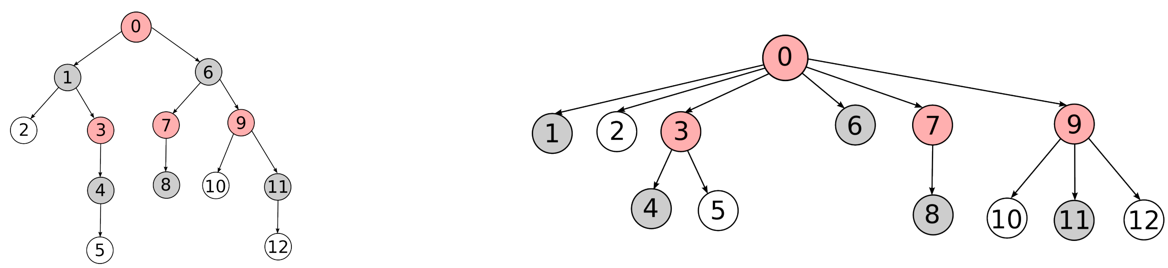

Рис. 4: Упрощение дерева. Серым выделены вершины множества W, красным — важные вершины

Докажем несколько свойств нового дерева:

- Для любой вершины нового дерева v все её непосредственные дети обладают меньшим номером. Данное свойство очевидно из построение

- Для любой вершины нового дерева v, номера вершин её поддерева образуют непрерывный отрезок. Ясно, что если вершина v не является важной, то её поддерево состоит из одной вершины и утверждение верно. Для важных вершин поддерево в новом дереве содержит только те вершины, которые были в оригинальном дереве. Значит в этом случае утверждение тоже верно

- Обход в глубину для нового дерева последовательно посещает вершины в порядке $0, 1, \cdots n-1$.

  Доказательство. Ясно, что для доказательства достаточно показать, что обход посещает вершины в порядке возрастания номеров. Рассмотрим две произвольные последовательные вершины в обходе в глубину: A и B. Рассмотрим три возможных случая перехода из A в B:

  1. A — родитель B. Но тогда $A < B$ по первому свойству
  2. A — брат B. Тогда $A < B$, т.к. мы упорядочивали детей всех внутренних вершин согласно их номерам
  3. A — последняя вершина некоторого поддерева вершины v такой, что v и B — братья. Но т.к. обе вершины A и v принадлежат некоторому непрерывному отрезку $[l \dots r]$ (по свойству 2), а $v < B$, то и $A < B$

  □

Т.к. обход нового дерева совпадает со старым обходом, а каждая вершина из W осталась соединена со своим старым предком, то если мы построим структуру данных из леммы 4 над новым деревом, а также сохраним бит вектор, маркирующий вершины из множества W, то мы сможем эффективно отвечать на запросы поиска родителя вершины из W.

При этом данная структура будет занимать $(|W|+1)(\log \frac{n}{|W|+1} + O(1)) \leq |W|(\log \frac{n}{|W|} + O(1)) + \log n$ бит памяти. □

Лемма 6. *Пусть суффиксные ссылки в алгоритме Ахо-Корасик посчитаны только для некоторого множества вершин W такого, что для любой вершины v существует предок на расстоянии $K = O(1)$ из множества W. В таком случае можно адаптировать алгоритм Ахо-Корасик без ухудшения асимптотики $O(|T| + occ)$.*

Доказательство. Опишем рекурсивную процедуру вычисления перехода в автомате из состояния q по произвольной строке s

$$\delta(q, s) = \begin{cases} q & \texttt{если } s = \lambda \\ \delta(trans(q, s[0]), s[1 \dots]) & \texttt{если } trans(q, s[0]) \neq \Lambda \\ \delta(link(nearParent(q)), path(nearParent(q), q) + s) & \texttt{если } link(q) \neq \Lambda \end{cases}$$

Где $nearParent(q)$ — ближайший предок вершины q из множества W, а $path(v, u)$ — строка, получаемая выписыванием символов ребер бора на пути между вершинами v и u.

Заметим, что если мы можем прочитать произвольный символ текста T за $O(1)$ и s — это некоторая подстрока текста, то можно заменить её на пару указателей границ подстроки $l \dots r$. В таком случае данная процедура требует $O(\log n)$ бит памяти и соответсвует нашим ограничениям (если бы мы реально хранили s в виде двусвязного списка, то алгоритм стал бы потреблять $O(n \log n)$ бит памяти, что неприемлимо).

Оценим время работы алгоритма для случая $|s| = 1$. Покажем, что для вычисления перехода из вершины v в вершину u по нашему алгоритму необходимо $O(K\Delta h) = O(\Delta h)$ времени, где $\Delta h = |h(v) - h(u)|$, а $h(v)$ — высота вершины в боре. Для этого оценим количество переходов каждого типа:

- Переход первого типа случается только один раз
- Переход по суффиксной ссылке случается не больше Δh раз, т.к. в случае такого перехода высота конечной вершины u уменьшается хотя бы на 1
- Переход по ребрам бора не может случиться больше $K\Delta h$ раз, т.к. длина s увеличивается только при переходе по суффиксной ссылке, и следовательно не превосходит $K\Delta h$

Научимся вычислять ближайшего непомеченного предка $p = nearParent(v)$ и путь между ними за $O(|h(v) - h(nearParent(v))|) = O(K) = O(1)$. Для этого достаточно научиться вычислять предка вершины и символ на ребре из предка в вершину за $O(1)$. Воспользуемся для этого сжатым массивом ребер бора $trans(q, c)$. Из его устойства несложно понять, что вершине с номером $cn + p$ соответствует предок с номером p. Причем соответствующую пару ord для предка мы можем узнать, выполнив поиск p-ого элемента в нашем битовом массиве. Аналогичным образом можно вычленить и значение символа на ребре.

Таким образом наш алгоритм работает за $O(\Delta h)$ на итерацию, что совпадает со временем работы обычного алгоритма Ахо-Корасик. □

Теорема 1. *Для любого положительного $\epsilon \leq 2$ можно построить структуру, заменяющую дерево суффиксных ссылок и занимающую $\epsilon n + o(\epsilon n)$ бит памяти, при этом поиск вхождений слов в текст будет занимать $O(|T|(\epsilon^{-1}\log\epsilon^{-1}) + occ)$*

Доказательство. Найдем такое натуральное K, что $\frac{\log K}{K} \leq \epsilon$. Несложно показать, что $K = \Theta(\epsilon^{-1}\log\epsilon^{-1})$. Обозначим за $level(i)$ — множество вершин бора, высота которых сравнима с i по модулю K. Ясно, что существует такое j, что $|level(j)| \leq \frac{n}{K}$. В таком случае сохраним суффиксные ссылки только для вершин множества $level(j)$ используя $\frac{n}{K}(\log K + O(1)) + \log n \leq \epsilon n(1 + \frac{1}{\log\epsilon^{-1}} + o(1)) = \epsilon n + o(\epsilon n)$ бит памяти (лемма 6). Ясно, что при таком распределении множества сохраненных вершин выполняются условия леммы 5, поэтому асимптотика решения увеличится до $O(K|T| + occ) = O((\epsilon^{-1}\log\epsilon^{-1})|T| + occ)$. □