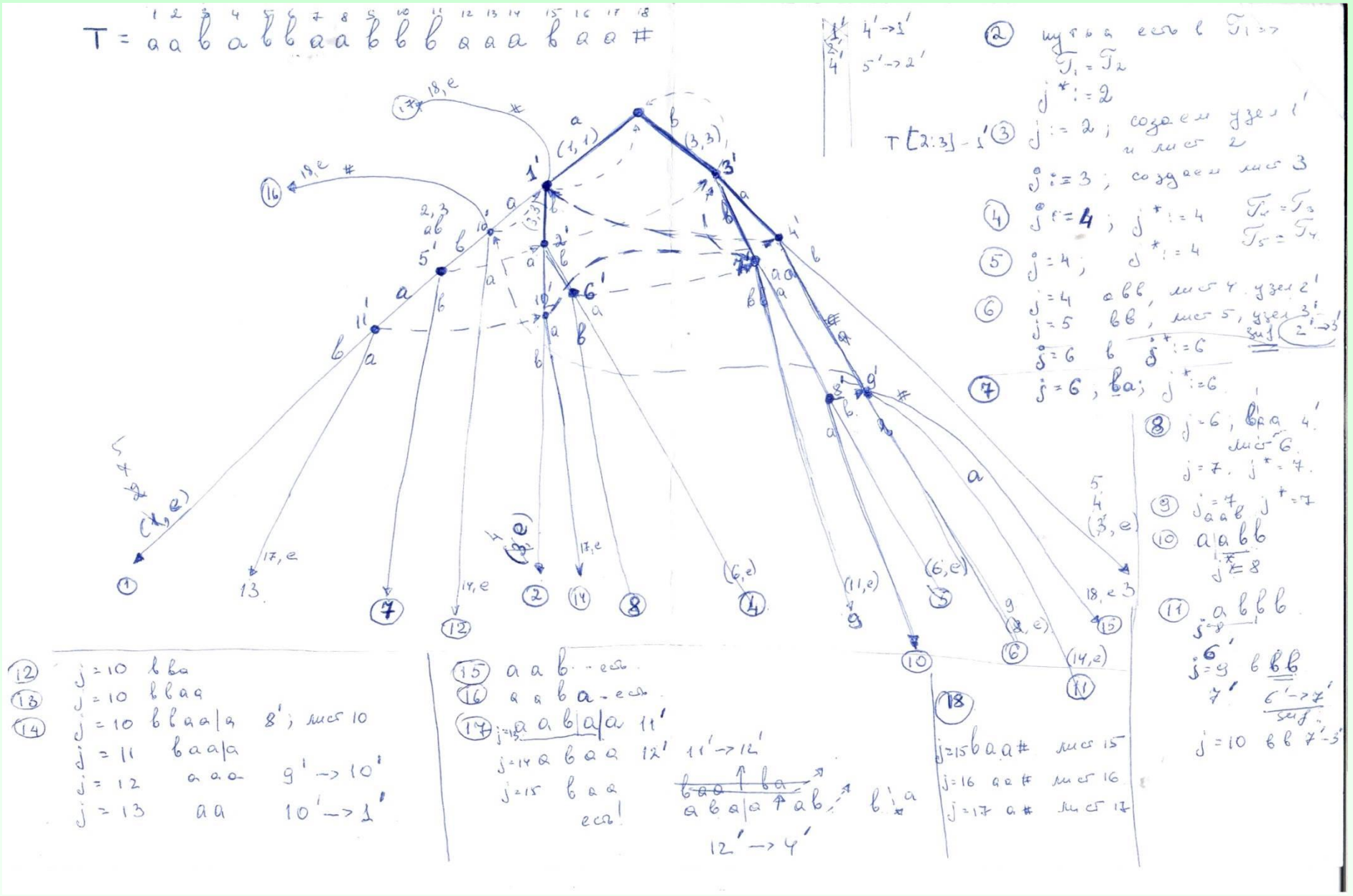
- **Пример.** $T = aababbaabbbaaabaa\#$.



Алгоритм построения суффиксного дерева с учетом замечаний

- Построить дерево Γ_1.
- $j^* := 1$;
- ***for*** $i := 1$ ***to*** N //фаза $i + 1$
- ***do begin*** // преобразовать Γ_i в Γ_{i+1}; для этого:
- e := $i + 1$;
- $j := j^*$;
- ***while*** (в текущем дереве нет пути $T[j : i + 1]$)
- ***do begin*** // обеспечить появление пути $T[j : i+1]$:
- для этого *найти конец существующего пути* $T[j : i]$ и,
- если необходимо, продолжить его элементом T_{i+1}
- $j := j + 1$;
- ***end***
- ***if*** (в текущем дереве есть путь $T[j : i + 1]$) ***then*** $j^* := j$;
- ***end***.

Осталось рассмотреть **вопрос поиска конца пути** $T[j : i]$.

Для поиска конца пути $T[j : i]$ используются ***суффиксные связи***. Они определяются для внутренних вершин суффиксного дерева.

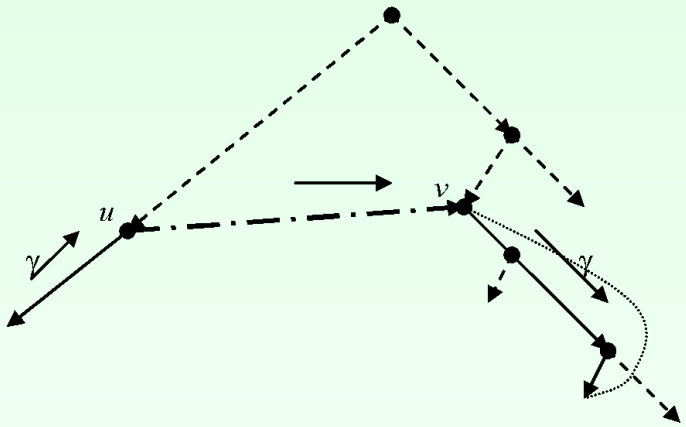
Пусть $a\beta$ – слово, $a \in \Sigma$, $\beta \in \Sigma^*$; u - вершина, путь из корня к "u" помечен $a\beta$;

v – вершина с путевой меткой β.

Указатель из u в v называется суффиксной связью и обозначается $v = \mathbf{suf}(u)$.

Как использовать суффиксные связи (в предположении, что они все существуют и все построены)?

Пусть $T[j : i] = a\,\beta\,\gamma$ ($a \in \Sigma$, $\beta \in \Sigma^*$, $\gamma \in \Sigma^*$) – последний рассмотренный путь в текущем дереве, (u, j) – дуга, помеченная γ. Следующее действие – поиск в текущем дереве конца строки $T[j + 1 : i\,] = \beta\,\gamma$. Если u – корень, просто спускаемся от корня по $\beta\,\gamma$. Если u – внутренняя вершина и есть суффиксная связь $v = \mathbf{suf}(u)$, то узел v имеет путевой меткой β, а γ должна помечать путь в поддереве v. От листа j идем вверх в узел u, далее по суффиксной связи в $v = \mathbf{suf}(u)$, затем от v вниз по пути с меткой γ. Конец пути $T[j + 1 : i\,] = \beta\,\gamma$ найден, можно приступать к удлинению этого пути элементом T_{i+1}.



- Осталось показать, что переход по суффиксной связи всегда возможен.
- **Теорема**. Суффиксные связи определены для всех внутренних вершин суффиксного дерева.
- **Лемма.** Если новая внутренняя вершина w с путевой меткой $a\beta$ добавляется к текущему дереву при рассмотрении $T[j : i + 1]$, то либо путь, помеченный β уже заканчивается во внутренней вершине текущего дерева, либо внутренняя вершина, соответствующая строке β будет создана при рассмотрении $T[j +1 : i + 1]$ (т.е. на следующем шаге той же фазы).
- **Док-во**. Новая вершина создается только правилом 2. Это означает, что в дереве есть путь $a\beta c$ ($c \neq t_{i+1}$, $a\beta = T[j : i + 1]$). Тогда в текущем дереве (на $(j + 1)$ шаге $(i + 1)$-й фазы есть путь βc. Если путь β продолжается не только "c", а еще каким-либо символом, то путь β приводит в вершину $u = \boldsymbol{suf}(w)$. Если путь β продолжается только "c", то на $(j + 1)$-м шаге $(i + 1)$-й фазы путь β должен быть продолжен символом t_{i+1}, т.е. вершина $u = \boldsymbol{suf}(w)$ будет создана.
- **Следствие.** В алгоритме Укконена любая вновь созданная вершина будет иметь суффиксную связь с концом следующего продолжения.
- **Следствие.** В любом неявном суффиксном дереве Γ_i, для любой внутренней вершины w с меткой $a\beta$ найдется внутренняя вершина $u = \boldsymbol{suf}(w)$ с путевой меткой β.

Окончательный вариант "действий" во внутреннем цикле алгоритма (построение пути $T[j : i+1]$) выглядит следующим образом:

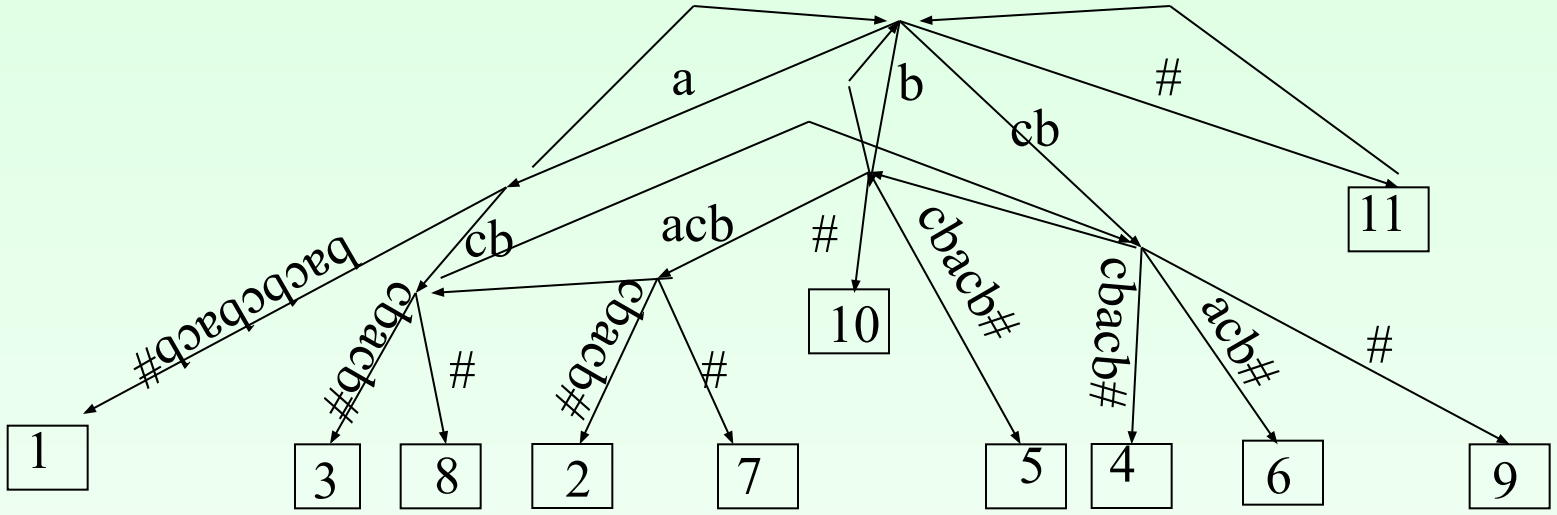
1. От конца пути $T[j – 1 : i]$ идем вверх до первой внутренней вершины v, имеющей исходящую суффиксную связь, либо до корня. Пусть γ – строка между v и концом пути $T[j – 1 : i]$ (возможно, пустая).
2. Если v – корень, пройти от корня по пути $T[j : i]$. Если v – внутренняя вершина, пройти по суффиксной связи из v в $u = \boldsymbol{suf}(v)$, затем от u вниз по пути с меткой γ. // Конец пути $T[j + 1 : i\ ] = \beta\ \gamma$ найден, можно приступать к удлинению этого пути элементом $T_{i+1.}$
3. Если строки $T[j : i + 1]$ в текущем дереве еще нет, выполняем действия по правилу 2, в частности создаем новую внутреннюю вершину w'.
4. Если в конце пути $T[j – 1 : i]$ была построена новая внутренняя вершина w, устанавливаем суффиксную связь w → w'.

Лемма. Пусть $v \rightarrow u$ – суффиксная связь, проходимая во время работы алгоритма Укконена. Если вершинная глубина v превосходит вершинную глубину u, то не более чем на единицу.

Теорема. Алгоритм Укконена строит суффиксное дерево для текста длины N за время $O(N)$.

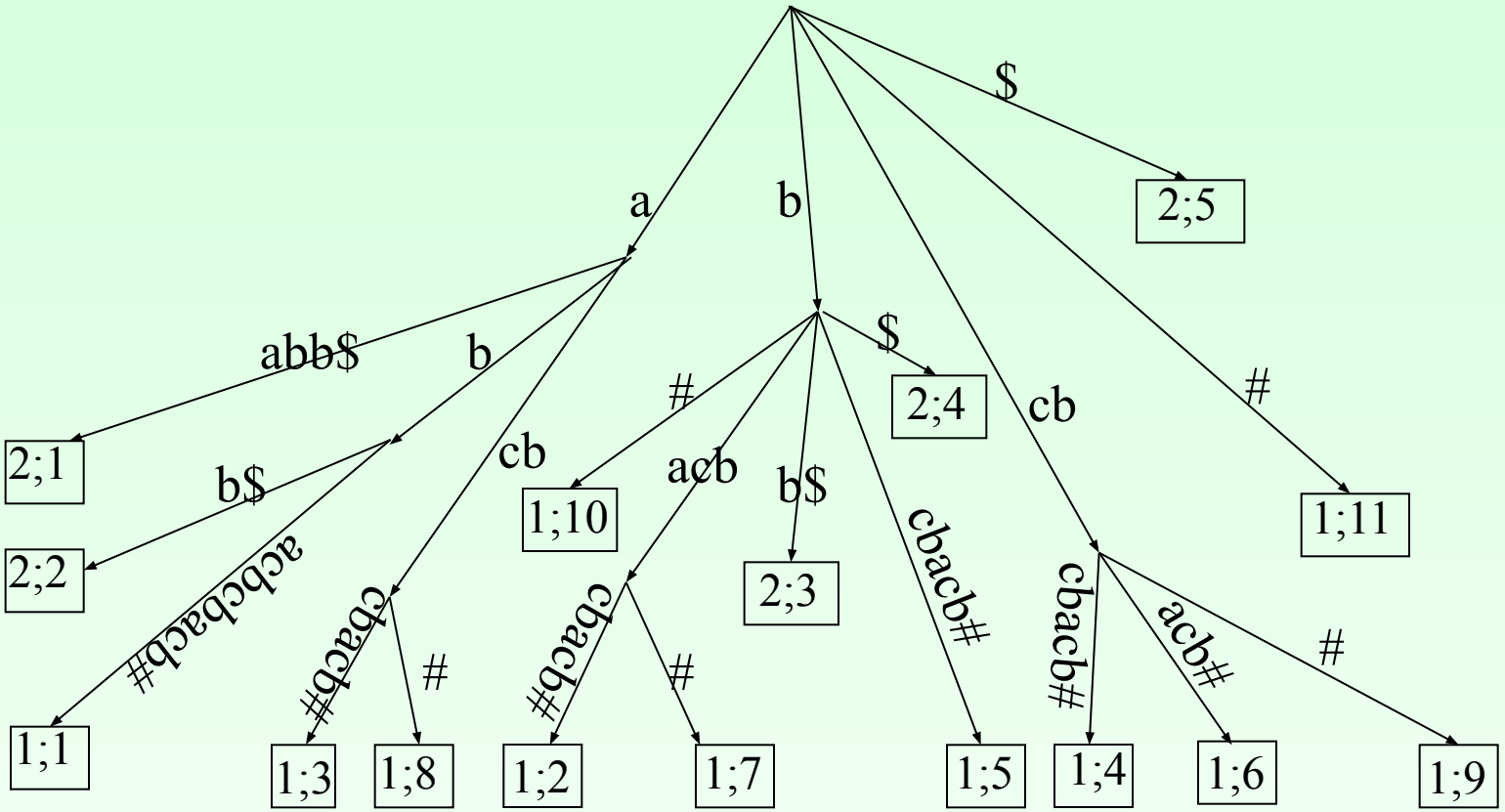
Задачи, решаемые с помощью суффиксного дерева:

- Поиск образца: P = bacb; P = bbc; T = abacbcbacb#
- Поиск множества образцов:
  Aho-Corasick или суффиксное дерево?
- Определение длины максимального повтора в тексте
- Обнаружение всех «нерасширяемых» повторов
- Наименьший *k*-повтор. Найти подстроку наименьшей длины, число вхождений которой в текст равно *k*.
- …
- Вычисление всех характеристик полного частотного спектра текста



Задачи, решаемые с помощью суффиксного дерева:

- **Наибольшая общая подстрока двух строк.** Обобщенное суффиксное дерево
  T_1 = abacbcbacb#; T_2 = aabb$;



Задачи, решаемые с помощью суффиксного дерева:

- Поиск палиндромов : $T_1 = T$; $T_2 = T^R$
- **Общие подстроки более чем двух строк**;
- **Задача загрязнения ДНК**. Даны строки S_1 и S_2: S_1 – вновь расшифрованная ДНК, S_2 – комбинация источников возможного загрязнения. Найти все подстроки S_2, которые встречаются в S_1 и длина которых не меньше заданного l;
- **Задача о праймере.** Дан набор строк $\{T_1 \ldots T_Z\}$ в алфавите Σ. Построить кратчайшую строку S над Σ, которая не встречается как подстрока ни в одном из $\{T_1 \ldots T_Z\}$.
- Другой вариант задачи: Задана строка P. Найти подстроки P, не встречающиеся как подстроки в $\{T_1 \ldots T_Z\}$.
- Или так: для каждого i вычислить кратчайшую подстроку P, начинающуюся с позиции i и не встречающуюся в качестве подстроки в $\{T_1 \ldots T_Z\}$.
- **Суффиксно-префиксные совпадения** всех пар строк (из заданного множества строк);
- **Задача о** наибольшем общем **«продолжении»**. Найти длину наибольшего общего префикса i-го суффикса строки S_1 и j-го суффикса строки S_2

- **Суффиксный массив** для строки T[1 : N] есть массив целых чисел от 1 до N, определяющий лексический порядок всех суффиксов стоки T.

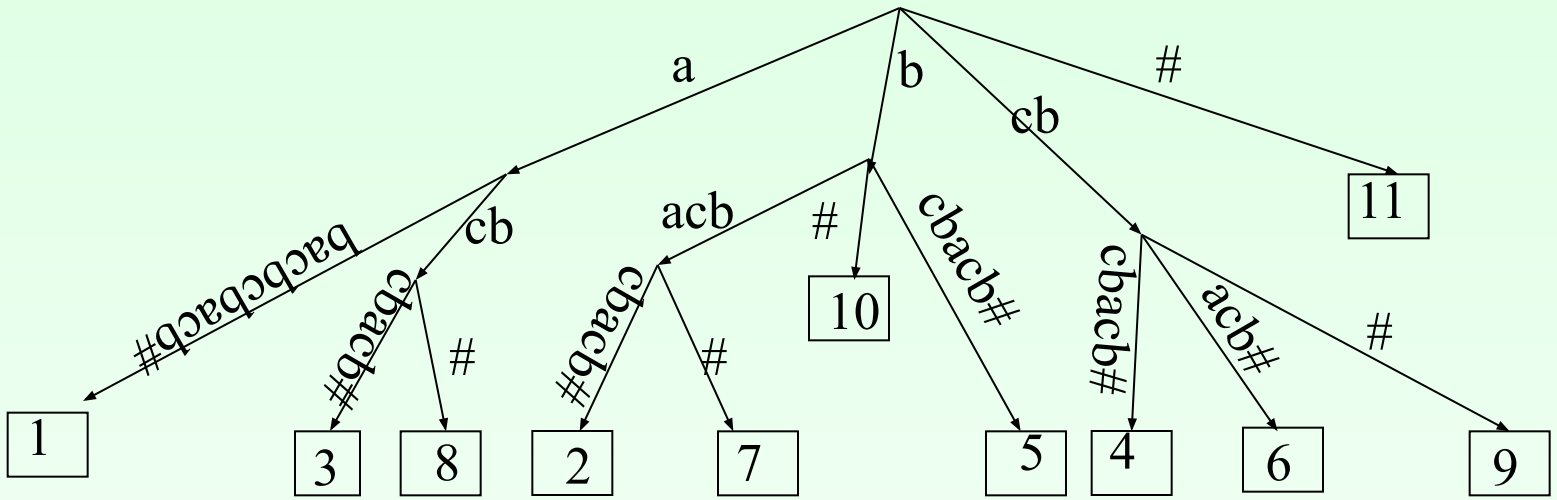
Пример T = mississippi Pos = (11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3)

11	**i**
8	**ippi**
5	**issippi**
2	**ississippi**
1	**mississippi**
10	**pi**
9	**ppi**
7	**sippi**
4	**sissippi**
6	**ssippi**
3	**ssissippi**

Построение суффиксного массива путём лексического обхода суффиксного дерева.

Пример: T = abacbcbacb (# < a < b < c)

Pos = (1, 8, 3, 10, 7, 2, 5, 9, 6, 4)



Задачи, связанные с поиском повторов.

Система антиплагиат

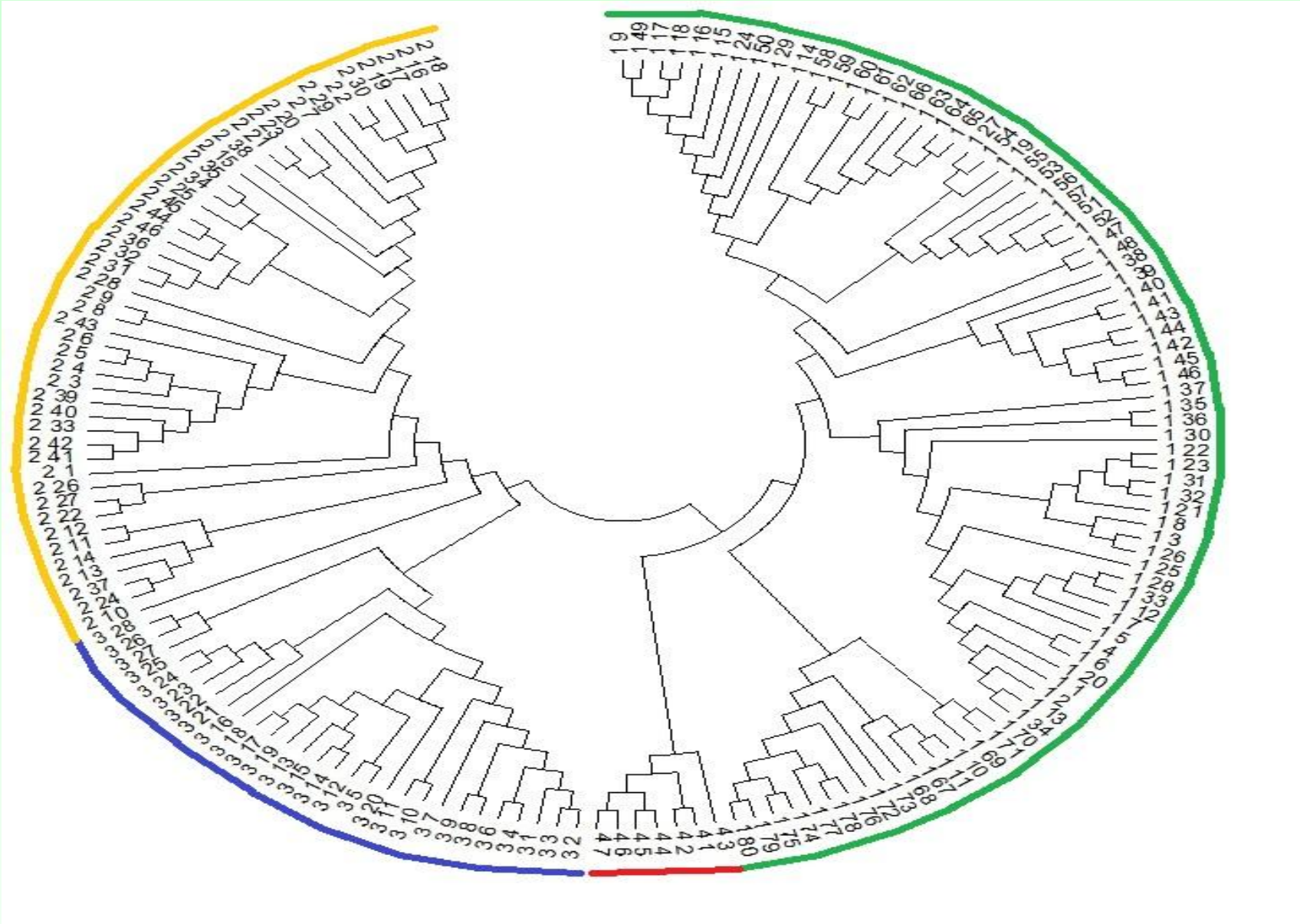
Поиск признаков для классификации последовательностей.

$\boldsymbol{K = \{\Pi_1, \Pi_2, \ldots \Pi_k\}}$ – множество текстов, разбитое на k классов

$\Pi_i = \{T_{i1}, T_{i2}, \ldots T_{imi}\}$ $(1 \le i \le k)$, m_i – количество текстов в i-м классе.

$\boldsymbol{\Phi_L(\Pi_i / K)}$ – множество «контрастных» L-грамм, присутствующих в <u>каждом</u> тексте i-го класса и отсутствующих во всех остальных текстах.

Вирус клещевого энцефалита (ВКЭ)



L-граммный анализ ВКЭ

- **Группа 886** ($m_4 = 7$): $L_{max}(\Phi(\Pi_4 / K)) = 491$; $L_{min}(\Phi(\Pi_4 / K)) = 7$

  $\Phi_7(\Pi_4 / K)$ = {cccgctg (поз. 2763 и 5221), atggcgc (поз. 3859 и 5517)}

- Класс 1 ($m_1 = 80$) : $L_{max}(\Phi(\Pi_1 / K)) = 17$; $L_{min}(\Phi(\Pi_1 / K)) = 6$

  $\Phi_6(\Pi_1 / K)$ = {agtaga}; F(agtaga, Π_1) = 182

- Класс 2 ($m_2 = 46$) : $L_{max}(\Phi(\Pi_2 / K)) = 35$; $L_{min}(\Phi(\Pi_2 / K)) = 7$

  $|\Phi_7(\Pi_2 / K)| = 8$;

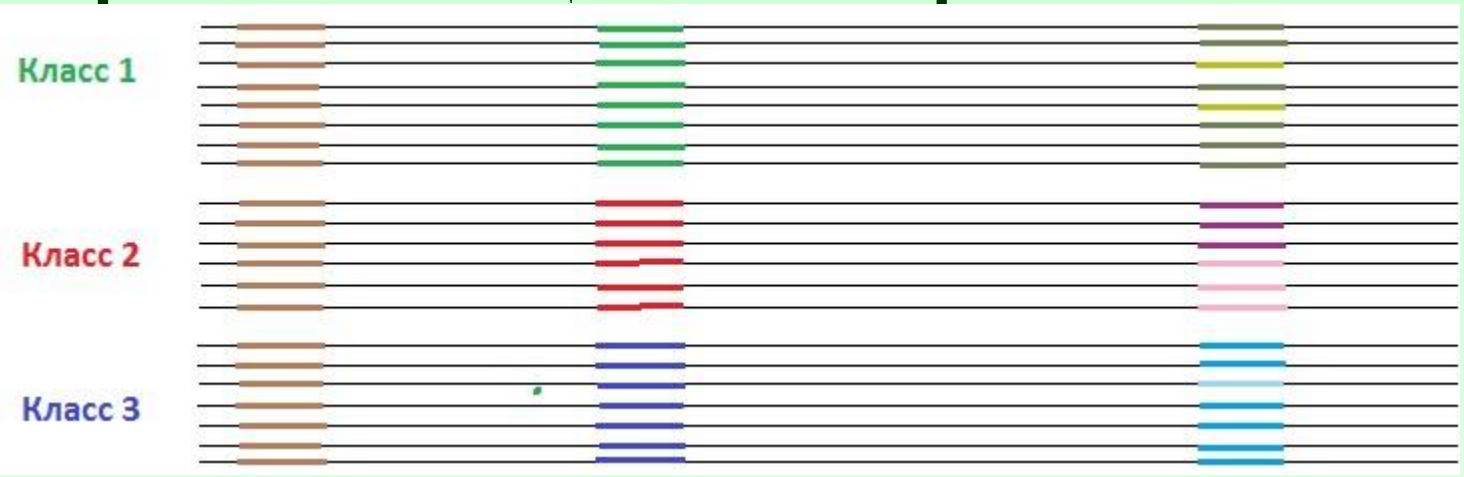
  $\Phi_7(\Pi_2 / K)$ = {gttattc, gcatctg, tgcacaa …}

- Класс 3 ($m_3 = 28$) : $L_{max}(\Phi(\Pi_3 / K)) = 35$; $L_{min}(\Phi(\Pi_3 / K)) = 8$

  $|\Phi_8(\Pi_3 / K)| = 9$;

  $\Phi_8(\Pi_3 / K)$ = {gggtggac (поз. 5029), ataagccg (поз. 6562), aggagagt (поз. 8020) … }

L-граммно-позиционный «срез»



$\Lambda_{L,pos}(\Pi_i)$ – множество различных L-грамм, расположенных в позиции pos в текстах класса Π_i. $M_{L,pos}(\Pi_i) = |\Lambda_{L,pos}(\Pi_i)|$.

- Позиция pos однозначно «классифицирующая», если

$$M_{L,pos}(\Pi_i) = M_{L,pos}(\Pi_j) = 1, \text{ но } \Lambda_{L,pos}(\Pi_i) \neq \Lambda_{L,pos}(\Pi_j) \quad (1 \le i, j \le k,\ i \neq j)$$

- В общем случае, позицию pos «классифицирующая», если

$$M_{L,pos}(\Pi_i) \ge 1,\ M_{L,pos}(\Pi_j) \ge 1,$$
$$\Lambda_{L,pos}(\Pi_i) \cap \Lambda_{L,pos}(\Pi_j) = \varnothing \qquad (1 \le i, j \le k,\ i \neq j).$$

«Устойчивые» позиции ВКЭ.

Позиция 4618 (H - гистидин) : cac в 1, 2, 3, *cat* (группа 886)

Позиция	Фрагмент РНК	Аминокислотная тройка
55	tcg-aaa-gag	S-K-E
73	aag-acg-cgt	K-T-R
115	ttg-atg-cgc	L-M-R
118	atg-cgc-atg	M-R-M
2269	atg-tcc-atg	M-S-M
2929	ctc-tgg-atg	L-W-M
4621	acg-atg-tgg	T-M-W
9382	aac-ata-aag	N-I-K

Примеры позиций, «характеризующих» класс:

3568 VAV	1	gtg-gca-gtt	3571 AVG	1	gca-gtt-ggg
	2	gtg-gca-gtg		2	gca-gtg-ggg
	3	gtg-gcg-gtg, gtt-gcg-gtg		3	gcg-gtg-ggg
	4	gta-gca-gtc		4	gca-gtc-ggg

Позиция 2350	AA .	класс	Кол-во текстов
gac-acc-gaa	DTE	1	44 из 80
gac-act-gaa	DTE		34 из 80
gat-act-gaa	DTE		1
gac-ccc-gaa	DPE		1
gac-acg-gaa	DTE	2	45 из 46
gac-aca-gaa	DTE		1
gac-acc-gag	DTE	3	25 из 28
gac-act-gag	DTE		3
gac-aca-gag	DTE	4	7 из 7

Расстояния между строками и меры близости

Расстояние Хэмминга: число несовпадений символов.

Пример: A = abbaeac; $d_{хэм} = 2$.

B = abdaecc;

Редакционное расстояние. В простейшем виде расстояние $d(A,B)$ между строками u и v определяется как минимальное число операций типа "вставка", "устранение" или "замена" символа, переводящих одну строку в другую.

Пример: A = abbaeac; B = bdedac

$A \rightarrow B$:

a	b	b	a	e	a	c
–	b	d	e	d	a	c

$d(A,B) = 4$ (1 дел., 3 зам).

$A \rightarrow B$:

a	b	b	a	e	–	a	c
–	b	d	–	e	d	a	c

$d(A,B) = 4$ (2 дел., 1 вст., 1 зам).

Хроника введения расстояний и мер сходства, основанных на идеях динамического программирования.

Год	Наименование (меры, расстояния, процедуры сравнения)	Авторы	Предметная область
1965	Метрика Левенштейна	Левенштейн В. И.	теория связи (двоичные коды)
1968	процедуры нелинейной нормализации речи по темпу	Слуцкер Г.С.	распознавание речи
1968	ДП - метод	Винцюк Т.К.	распознавание речи
1969	дискретный вариант меры Слуцкера	Загоруйко Н.Г., Величко В.М.	распознавание речи
1970	мера сходства АМ-последовательностей	Needleman S.B., Wunch S.B.	молекулярная биология
1974	редакционное расстояние	Wagner R.A., Fisher M.J.	лингвистика
1974	эволюционное расстояние	Sellers P.	молекулярная биология

Схема динамического программирования для вычисления редакционного расстояния:

$d(0,0) = 0;$

$d(i,0) = i,\ 1 \le i \le m;\ m = |A|;$

$d(0,j) = j,\ 1 \le j \le n;\ \ n = |B|;$

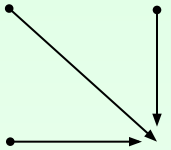
$$d(i,j) = \min \{d(i-1,\ j-1) + \gamma(a_i,b_j),\ d(i-1,\ j) + 1,\ d(i,\ j-1) + 1\}.$$

$$\gamma(a_i,b_j) = \begin{cases} 0, & a_i = b_j \\ 1, & a_i \neq b_j \end{cases}$$

B

	ε	a	b	b	a	e	a	c
ε	0	1	2	3	4	5	6	7
b	1	1	1	2	3	4	5	6
d	2	2	2	2	3	4	5	6
e	3	3	3	3	3	3	4	5
d	4	4	4	4	4	4	4	5
a	5	4	5	5	4	5	4	5
c	6	5	5	6	5	5	5	**4**

A



- Этап 1: заполнение матрицы динамического программирования.
- Этап 2: обратный ход. Построение выравнивания. $\forall$ $d(i,j)$ на этапе 1 запоминаем указатели на элемент, от которого реализовался переход ($d[i-1, j-1]$, $d[i-1, j]$ и /или $d[i, j-1]$; либо на этапе 2 делаем проверку $d[i, j] = d[i-1, j-1] + \gamma(a_i \rightarrow b_j)$?, или $d[i,j] = d[i-1, j] + \gamma(a_i \rightarrow \varepsilon)$?, или $d[i,j] = d[i, j-1] + \gamma(\varepsilon \rightarrow b_j)$?.

Трудоемкость и затраты памяти $O(n \times m)$.

Если требуется только вычислить редакционное расстояние (без выравнивания) можно хранить две строки: текущую и предыдущую.

Для построения выравнивания требуется знать всю матрицу.

–	**b**	**d**	**–**	**e**	**d**	**a**	**c**	**или**	**–**	**b**	**d**	**e**	**d**	**a**	**c**
	|					|	|		|				|	|
a	**b**	**b**	**a**	**e**	**–**	**a**	**c**		**a**	**b**	**b**	**a**	**e**	**a**	**c**

- В общем случае число редакционных операций можно расширить, например, добавить перестановку соседних символов или блочные вставки или устранения.
- Каждая операция может иметь свой "вес". В этом случае редакционное расстояние определяется как минимальная "стоимость" перевода A в B.

- Если к редакционным операциям добавлена операция перестановки соседних символов, схема динамического программирования выглядит следующим образом:

$$d[\,i,\,j\,] = \min\{\, d[i-1,\,j-1] + \gamma(a_i \to b_j),$$
$$d[i-1,\,j\,] + \gamma(a_i \to \varepsilon),$$
$$d[i,\quad j-1] + \gamma(\varepsilon \to b_j)$$
$$d[i-2,\,j-2] + \gamma(a_{i-1}\,a_i \to b_{j-1}\,b_j).\ \text{если}\ a_i\,a_{i-1} = b_{j-1}\,b_j\}$$

при тех же начальных условиях

$d(i,0) = i,\ 1 \le i \le m;\ m = |A|;$

$d(0,j) = j,\ 1 \le j \le n;\ n = |B|;$