МЕТОДЫ ОБНАРУЖЕНИЯ ЗАКОНОМЕРНОСТЕЙ С ПОМОЩЬЮ ЭВМ
(Вычислительные системы)
1981 год Выпуск 91

УДК 519.712.3

АДАПТИВНЫЙ АЛГОРИТМ ОТЫСКАНИЯ МАКСИМАЛЬНО ДЛИННОЙ ОБЩЕЙ ПОДПОСЛЕДОВАТЕЛЬНОСТИ

В.Д.Гусев, О.А.Кутненко

I. И с х о д н ы е п о н я т и я. Пусть A будет произвольная конечная символьная последовательность над алфавитом A_0. Введем следующие обозначения: $|A|$ - длина A, $A[i]$ - i-й элемент A, $A[i:j]$ - элементы A с i-го по j-й включительно.

Назовем U подпоследовательностью V, если существует монотонно возрастающая последовательность целых $r_1,\dots,r_2,\dots,r_{|U|}$ такая, что $U[i] = V[r_i]$ для $1 \le i \le |U|$. U является общей подпоследовательностью последовательностей A и B, если U - подпоследовательность как A, так и B. Максимально длинная общая подпоследовательность (МДП) есть общая подпоследовательность с наибольшим возможным числом элементов. Например, последовательности A=abcbdda и B=badbabd имеют МДП = abbd. Отметим, что сопоставляемые последовательности могут иметь несколько МДП, отличающихся как по составу элементов, так и по их расположению (при одинаковом составе). К примеру, последовательности A=abcdb и B=abbd имеют $(МДП)_1$ = abb и $(МДП)_2$ = abd.

Задача отыскания МДП связана с задачей вычисления введенного определенным образом расстояния между двумя символьными последовательностями [2]. Расстоянием между последовательностями $A = a_1a_2\dots a_m$ и $B = b_1b_2\dots b_n$ будем считать наименьшее число шагов, требующихся для перевода одной последовательности в другую, где под шагом понимается любая из следующих элементарных операций:

1) замена одного символа другим;

2) устранение одного символа из последовательности;

3) включение одного символа в последовательность.

Каждая из этих операций может иметь свою "стоимость". Если принять

стоимость второй и третьей операций равной I, а стоимость первой - 2, то длина p МДП будет связана с введенным таким образом расстоянием $\delta(A,B)$ соотношением $\delta(A,B) = |A| + |B| - 2p$.

Можно указать ряд содержательных задач, где возникает необходимость в отыскании МДП или вычислении расстояния между символьными последовательностями:

а) автоматическое обнаружение и коррекция ошибок в тексте;

б) вычисление мер близости между двумя ранжированиями N объектов (в частности, отыскание для заданной перестановки N чисел максимально длинной подпоследовательности с монотонно возрастающими членами);

в) нелинейная нормализация слов по темпу произнесения в задачах распознавания речи;

г) вычисление эволюционной близости между генетическими текстами (т.е. последовательностями нуклеотидов или аминокислот, представляющими молекулы нерегулярных полимеров - ДНК, РНК, белков).

Из приведенных примеров видно, что задача получения "быстрых" алгоритмов вычисления МДП весьма актуальна как из-за большого числа приложений, так и вследствие необходимости многократного применения указанных алгоритмов в отдельных приложениях.

2. А д а п т и в н ы е а л г о р и т м ы. В [1] для случая б е с к о н е ч н о г о алфавита получены верхние и нижние границы трудоемкости задачи отыскания МДП, пропорциональные N^2.*) В [2] описан алгоритм решения этой задачи, имеющий трудоемкость $O(N^2)$. Его можно считать "универсальным" в том смысле, что он применим к произвольным последовательностям и его трудоемкость не зависит от вида этих последовательностей.

Дальнейшие продвижения возможны в направлении разработки адаптивных алгоритмов, ориентированных на последовательности о п р е д е л е н н о й с т р у к т у р ы [3,4], либо на произвольные последовательности, но над к о н е ч н ы м а л ф а в и т о м [5]. Для многократно повторяемых задач относительно небольшой размерности актуальным является построение алгоритмов, которые при том же порядке роста сложности, что и у известных алгоритмов, имеют существенно меньшую мультипликативную постоянную в оценке трудоемкости.

---

*) При обзоре результатов, упоминаемых в данном и последующем разделах, ограничимся для простоты случаем двух последовательностей одинаковой длины N .

Сужение класса последовательностей позволяет понизить трудоемкость соответствующих алгоритмов (например, до $O(N\log N)$). Однако в наихудшем случае (когда на вход поступают последовательности не из заданного класса) их трудоемкость может оказаться большей, чем $O(N^2)$.

Ограничение мощности алфавита также позволяет в принципе строить алгоритмы с порядком роста трудоемкости, меньшим, чем N^2. Пример (пока единственный) алгоритмов такого рода представлен в [5]. Трудоемкость алгоритма равна $O(N^2/\log N)$, однако мультипликативная постоянная, соответствующая указанному порядку роста сложности, весьма велика. Этот же недостаток характерен для алгоритма, описанного в [4].

Целью данной работы является пополнение класса адаптивных алгоритмов за счет использования новых (более тонких, пошаговых) стратегий адаптации. Эффективность пошаговой стратегии адаптации показана на примере алгоритма Ханта и Шиманского [3], трудоемкость которого в зависимости от класса последовательностей на входе может варьировать от $O(N\log N)$ до $O(N^2\log N)$. Структурируя данные так же, как в указанном алгоритме, удается понизить трудоемкость в наихудшем случае до $O(N^2)$ (без дополнительных затрат памяти).

3. Краткое описание алгоритма Ханта и Шиманского. Ключевой структурой данных, используемой в алгоритме, является массив "пороговых значений" $T_{i,k}$ ($1 \le i \le N$, $1 \le k \le p$, p - длина МДП), где каждое $T_{i,k}$ есть наименьшее j, такое, что последовательности $A[1:i]$ и $B[1:j]$ содержат общую подпоследовательность длины k. Например, для последовательностей $A = abcbdda$ и $B = badbabd$ $T_{5,1} = 1$, $T_{5,2} = 3$, $T_{5,3} = 6$ (общая подпоследовательность - abb), $T_{5,4} = 7$ (общая подпоследовательность - abbd), $T_{5,5}$ - не определено. Каждое $T_{i,k}$ может интерпретироваться как указатель, определяющий, сколько членов последовательности B нужно взять, чтобы они образовали МДП длины k с первыми i элементами последовательности A.

Алгоритм базируется на следующих трех леммах.

ЛЕММА 1. Если $T_{i,1}, T_{i,2}, \ldots, T_{i,p}$ определены, то $T_{i,1} < T_{i,2} < \ldots < T_{i,p}$, т.е. каждая строка массива состоит из строго возрастающих чисел.

ЛЕММА 2. $T_{i,k-1} < T_{i+1,k} \le T_{i,k}$.

ЛЕММА 3.

$$T_{i+1,k} = \begin{cases} \text{наименьшему } j \text{ такому, что} \\ A[i+1] = B[j] \quad \text{и } T_{i,k-1} < j < T_{i,k}; \\ T_{i,k}, \text{ если такого } j \text{ не существует.} \end{cases}$$

Ниже приводится заимствованное из [3] описание алгоритма для случая последовательностей одинаковой длины. Массив пороговых значений (THRESH из [3]) и список возможных соответствий между одинаковыми элементами обеих последовательностей (MATCHLIST [3]) обозначены для краткости через TH и ML :

```
element array A[1:N],B[1:N];
integer array TH[0:N];
list array ML[1:N];
pointer array LINK[0:N];
pointer PTR;
comment Шаг I: построение списков возможных соответствий
между элементами массивов A и B ;
for i:= 1 step 1 until N do
ML[i]:= ⟨ j1,j2,...,jp ⟩ ,где  j1 > j2 > ... > jp  и
A[i] = B[jq] для  1 ≤ q ≤ p;
comment Шаг 2; задание начальных значений для массива TH;
TH[0]:= 0;
for i:= 1 step 1 until N do TH[i]:= N+1;
LINK[0]:= null;
comment Шаг 3: вычисление пороговых значений;
for i:= 1 step 1 until N do
for j , пробегающих значения из  ML[i], do
begin найти k такое, что  TH[k-1] < j ≤ TH[k];
if j < TH[k] then
begin TH[k]:= j;  LINK[k]:= newnode (i,j,LINK[k-1]):
end; end;
comment  Шаг 4: восстановление МДП в обратном порядке;
k:= максимальное k такое, что  TH[k] ≠ N+1;
PTR:= LINK[k];
while PTR ≠ null do
begin print (i,j)  - печать пары, выделяемой указателем PTR;
изменение значения PTR;
end
```

Корректность алгоритма следует из того, что $TH[k] = T_{i-1,k}$ для всех k в начале каждой итерации по i и $TH[k] = T_{i,k}$ для всех k в конце каждой итерации по i.

На первом шаге алгоритма для каждой позиции i последовательности A составляется список позиций j последовательности B, таких, что $A[i] = B[j]$. Список организуется в порядке убывания индекса j. Шаг I реализуется лексикографической сортировкой каждой последовательности (с последующим их слиянием) при условии сохранения для каждого элемента информации о номере позиции, в которой он располагается. Трудоемкость этого шага $O(N\log N)$, затраты памяти - $O(N)$.

Шаг 2 реализуется за время $O(N)$ при затратах памяти $O(N)$.

Два внешних цикла на третьем шаге должны рассматриваться как единый цикл по всем парам (i,j) таким, что $A[i] = B[j]$ (i увеличивается, j при фиксированном i уменьшается). Обозначим число таких пар через r:

$$r = \sum_{a_k \in A_0} f^{(1)}(a_k) \cdot f^{(2)}(a_k),$$

где $f^{(1)}(a_k)$ - частота встречаемости символа a_k в 1-й последовательности ($1 = 1,2$). Внешние циклы шага 3 индуцируют точно r выполнений тела цикла. Так как тело цикла включает один бинарный поиск в упорядоченном массиве TH длины N и несколько операций, время выполнения которых не зависит от N (в частности, операцию формирования очередного узла списка, необходимого для восстановления МДП), то трудоемкость шага 3 есть $O(N+r\log N)$. Нетрудно видеть, что параметр r может меняться для различных последовательностей в диапазоне от 0 до N^2. Соответственно этому в наихудшем случае шаг 3 имеет трудоемкость $O(N^2\log N)$.

На шаге 4 восстанавливается МДП, что требует (самое большее) $O(N)$ элементарных операций. Таким образом, основные затраты времени (а также и памяти*)) связаны с шагом 3 алгоритма. Поэтому мо -

*) Массив LINK после i-й итерации (шаг 3) содержит лишь з а г о л о в к и списков, каждый из которых позволяет восстановить МДП длины k ($k = 1,2,\ldots, k_{max}(i)$) последовательностей $A[1:i]$ и $B[1:j(k)]$, где $j(k)$ - минимально необходимое для существования МДП длины k число элементов из B. Сами списки, требующие в наихудшем случае памяти $O(r)$, из описания массивов авторами упущены, хотя в оценках памяти они фигурируют.

дификация затронула в основном именно этот шаг. Полная трудоемкость алгоритма Ханта и Шиманского есть $O((r+N)\log N)$ при затратах памяти $O(r)$ в наихудшем случае. Основные затраты памяти связаны не с оценкой длины МДП, а с хранением информации, необходимой для ее восстановления.

4. Адаптивная стратегия отыскания МДП. Пусть в общем случае длины последовательностей не совпадают ($M \neq N$). Предположим для определенности, что $M \leq N$, и организуем внешний цикл на шаге 3 алгоритма по $i = 1,2,\ldots,M$. Предлагаемая ниже адаптивная стратегия отыскания МДП основана на следующих предпосылках.

1) Ядро процедуры вычисления МДП разбивается на ряд однородных шагов (цикл по $i = 1,\ldots,M$ на этапе вычисления пороговых значений), каждый из которых целесообразно оптимизировать по отдельности.

2) Оба массива данных, которые обрабатываются на i-й итерации, являются строго упорядоченными. Для массива TH это вытекает из леммы 1. Список позиций $J = \{j_1, j_2, \ldots, j_{m_i}\}$, таких, что $A[i] = B[j_1] = B[j_2] = \ldots = B[j_{m_i}]$ и $m_i = f^{(2)}(A[i])$ упорядочен по убыванию при построении массива ML.

В [3] для сокращения объема вычислений используется лишь упорядоченность массива TH (при организации бинарного поиска). В рассматриваемой версии эффективно используется упорядоченность обоих массивов.

3) Нетрудно видеть, что длина p МДП ограничена сверху значением $N_{эф.} = \sum_{a_k \in A_0} \min\{f^{(1)}(a_k), f^{(2)}(a_k)\} \leq M$. Поэтому количество отличных от N+1 элементов массива TH не должно превышать $N_{эф.}$ и соответственно трудоемкость однократного бинарного поиска в этом массиве составит $]\log_2 N_{эф.}[$. Параметр $N_{эф.}$ легко вычисляется на основе массива ML. Таблица, задающая порядок поиска, также формируется на первом шаге алгоритма.

4) При переходе от i к i+1 значения $T_{i+1,k}$ получаются из значений $T_{i,k}$ в соответствии с леммой 3. При этом корректировке подвергается не вся строка $T_{i,k}$ (т.е. не весь массив TH), а лишь часть ее, ограниченная значениями $k_л(i)+1$ (слева) и $k_п(i)+1$ (справа).

Параметр $k_л(i)$ определим как максимальное k, такое, что для всех $k \leq k_л(i)$ выполняется $TH[k] = k$. Это означает, что при

$1 \le k \le k_{Л}(i)$ значения TH[k] образуют отрезок натурального ряда от I до $k_{Л}(i)$ и в соответствии с леммой 3 не найдется такого j, которое могло бы изменить любое из чисел этого отрезка. Заметим, что если после i-й итерации в общем случае в массиве TH сформируется произвольный отрезок натурального ряда (не обязательно начинающийся с единицы), то в ходе (i+1)-й итерации изменению может подвергнуться лишь крайний левый член этого ряда.

Нетрудно видеть, что с увеличением i параметр $k_{Л}(i)$ может лишь возрастать (но не больше чем на единицу на каждом шаге). Наибольшей скорости возрастания следует ожидать для пар последовательностей с м а л ы м алфавитом и равномерным распределением символов по обеим последовательностям. В частности, для идентичных последовательностей с $|A_0|=1$ имеем $k_{Л}(i) = i$, т.е. использование данного параметра дает наибольший эффект в наиболее тяжелой ($r = N^2$) для алгоритма [3] ситуации.

Параметр $k_{П}(i)$ определим как минимальное k, такое, что для всех $k > k_{П}(i)$ выполняется TH[k] = N+1. Нетрудно видеть, что $k_{П}(i) \le i$, причем $k_{П}(M) = p$ (длине МДП).

Параметры $k_{Л}(i)$ и $k_{П}(i)$ корректируются на шаге 3 после каждой итерации по i. Наибольший вклад в ограничение длины варьируемой части массива TH ($k^*(i) = k_{П}(i) - k_{Л}(i) + 1$) при малых i вносит параметр $k_{П}(i)$, а при больших - $k_{Л}(i)$.

На i-й итерации в алгоритме [3] m_i раз осуществляяся бинарный поиск в массиве длины N. Отсюда трудоемкость i-й итерации - $O(m_i \log_2 N)$. Очевидно, что такая стратегия выгодна лишь при малых значениях m_i; при больших же ($m_i \sim N$) имеем $T_i = O(N \log_2 N)$. Используя предпосылки I - 4, можно предложить следующую а д а п т и в н у ю стратегию поведения на каждой i-й итерации: если $m_i \log_2 N_{эф.} < m_i + k^*(i-1)$, то для формирования массива TH используется бинарный поиск, в противном случае массив TH формируется путем с л и я н и я двух у п о р я д о ч е н н ы х массивов - строки $T_{i-1,k}$ (от предыдущей итерации) и списка позиций $J = \langle j_1, j_2, \ldots, j_{m_i} \rangle$, где $A[i] = B[j_q]$ для всех $1 \le q \le m_i$.

Смысл подобной стратегии заключается в следующем. Во-первых, из описания алгоритма слияния (см. ниже) следует корректность (в смысле конечного результата) замены m_i-кратной процедуры бинарного поиска процедурой слияния. Во-вторых, трудоемкость процедуры слияния есть $O(m_i + k^*(i-1))$, т.е. в наихудшем случае, если даже на каждой итерации будет приниматься решение о слиянии, суммарная

трудоемкость составит $O(\sum_{i=1}^{M} m_i + \sum_{i=1}^{M} k^*(i))$. Поскольку $\sum_{i=1}^{M} m_i \le MN$, а $\sum_{i=1}^{M} k^*(i) \le \sum_{i=1}^{M} i = M(M+1)/2$, имеем для наихудшего случая трудоемкость $O(MN)$. При $M=N$ получим $T_{max} = O(N^2)$ вместо $O(N^2 \log N)$ в исходном алгоритме.

Заметим, что ввиду упорядоченности массивов TH и J процедуру бинарного поиска можно осуществлять не только среди элементов массива TH (при малых m_i), но и среди элементов массива J (при малых $k^*(i)$). Стратегия поведения на i-й итерации тогда будет выглядеть следующим образом: если $\min\{m_i \log_2 N_{эф}, k^*(i-1) \cdot \log_2 m_i\} < m_i + k^*(i-1)$, то бинарный поиск (в TH или J), в противном случае - слияние. Оценка трудоемкости в наихудшем случае при этом не изменится.

5. А л г о р и т м с л и я н и я. Традиционная процедура слияния [6] двух упорядоченных (по возрастанию, например) массивов "a" и "b" длины |a| и |b| соответственно заключается в получении из них упорядоченного массива "c" длины |a| + |b|. Слияние осуществляется переносом в "c" наименьшего из очередных (еще не перенесенных) элементов массивов "a" и "b".

Отличительные особенности предлагаемого алгоритма слияния: а) в слиянии участвуют лишь элементы массива TH с номерами от $k_л(i-1)$ до $k_п(i-1)+1$ и элементы массива J со значениями, большими, чем $k_л(i-1)$; б) не требуется дополнительных затрат памяти, поскольку роль результирующего массива "c" играет переформировываемый по ходу слияния массив TH; удлинения массива TH не происходит, поскольку (в силу леммы 3) из всех значений j, которые при слиянии могли бы попасть в интервал между значениями TH[k] и TH[k+1], следует удержать лишь наименьшее, заменив им значение TH[k+1]; в) при наличии нескольких значений j, попадающих в интервал между TH[k] и TH[k+1], обращение к процедуре "newnode" происходит лишь один раз (в отличие от алгоритма [3]), что позволяет сэкономить память и время; г) формирование результирующего массива ведется в сторону убывания значений, т.е. справа налево.

Схема слияния выглядит следующим образом. Пусть на очередном шаге сравниваются k-й элемент массива TH(TH[k]) и l-й элемент массива $J(J[l], 1 \le l \le m_i)$. Если $TH[k] \ge J[l]$, то в массиве TH ничего не меняется. Очередной меньший элемент из TH (TH[k-1])

и "старый" претендент из J образуют новую пару для сравнения и процесс продолжается.

В противном случае (когда TH[k] < J[1]) TH[k] сравнивается со значениями J[1+1], J[1+2] и т.д. (J[1] > J[1+1] > J[1+2] >...) до тех пор, пока не найдется $1^* \geq 1+1$, такое, что TH[k] ≥ J[1*]. В этом случае TH[k+1] принимает значение J[1*-1], вновь создан - ный узел ⟨i, J[1* - 1]⟩ включается в общий список узлов, исполь - зуемых для восстановления МДП (см. процедуру "newnode" на шаге 3), элементы TH[k-1] и J[1*] образуют новую пару для сравнения и процесс продолжается до исчерпания одного из массивов (TH или J).

Если исчерпан массив TH (т.е. $k=k_л$ (i-1)) и TH[k] ≥ J[1], то процедура слияния заканчивается сразу; в противном случае (TH[k] < J[1]) в последний раз осуществляется поиск 1* и форми - рование очередного узла списка, после чего процедура слияния так- же заканчивается.

Если исчерпан массив J (в процессе поиска 1*), то TH[k+1] принимает значение $J[m_i]$ (если TH[k] < $J[m_i]$) или $J[m_i-1]$ (ес- ли TH[k] ≥ $J[m_i = 1^*]$), после чего в обоих случаях формируется очередной узел списка и процедура слияния заканчивается.

Нетрудно видеть, что значение J[1* - 1] есть минимальное из всех возможных значений j таких, что A[i] = B[j] и $T_{i-1,k} < j < T_{i-1,k+1}$, т.е. условия леммы 3 выполнены и построенный таким образом алгоритм слияния является корректным. Заметим также, что варьируемость длины массива TH учитывается в алгоритме слияния естественным и эффективным образом (в отличие от варианта с бинар- ным поиском, когда таблицу, задающую порядок поиска, каждый раз пришлось бы вычислять заново). Процедура слияния не требует до- полнительных затрат памяти под результирующий массив, поскольку элементы массива J служат лишь для коррекции массива TH и участ- вуют в слиянии "фиктивно". Формирование массива TH справа налево, т.е. от больших значений к меньшим, соответствует порядку восста- новления МДП (от больших значений индексов i и j к меньшим). Слия- ние в обратном порядке потребовало бы дополнительных затрат по памяти (O(N)) и трудоемкости (O(N)).

6. Д о п о л н и т е л ь н ы е з а м е ч а н и я. 1. Если ал- фавит A_0 - конечный, причем $|A_0| < N$ (именно такая ситуация наи- более часто встречается на практике), первый шаг алгоритма, свя - занный с формированием массива ML, целесообразно реализовать сле- дующим образом. Заводится таблица с числом входов, равным числу

элементов алфавита. Обращение к таблице осуществляется по значению кода соответствующего символа. Каждому входу ставится в соответствие указатель на список позиций большей последовательности, содержащих данный символ. Линейным просмотром большей последовательности формируется массив ML. Меньшая последовательность, по которой организуется внешний цикл на шаге 3, просматривается лишь с целью определения частот $f^{(1)}(a_k)$ по всем a_k, входящим в нее. Таким образом, трудоемкость этого шага составит $O(N)$ при затратах памяти $O(N)$ вместо трудоемкости $O(N \log N)$ и той же памяти $O(N)$ в исходной версии.

При $|A_0| > N$ для построения ML можно использовать схему хеш-адресации [6]. Размер расстановочного поля определяется теперь не мощностью алфавита, а длинами анализируемых последовательностей. В отличие от предыдущего подхода в данной схеме возможны наложения, т.е. ситуации, когда двум и более разным символам соответствует одно и то же значение функции расстановки. Трудоемкость подхода "в среднем" $O(N)$ при затратах памяти $O(N)$. В наихудшем случае трудоемкость может оказаться нелинейной функцией от N, но этого, как правило, удается избежать подбором соответствующей функции расстановки (или суперпозиции нескольких функций).

2. Поскольку в алгоритме Ханта и Шиманского не описан этап запоминания узлов, необходимых для восстановления МДП, покажем, как это делается в предлагаемом алгоритме. Основное отличие от исходной версии, как уже упоминалось выше, состоит в том, что при наличии нескольких значений j, попадающих в полуинтервал $(T_{i-1,k}, T_{i-1,k+1}]$, в формировании нового узла участвует лишь одно значение, ближайшее к $T_{i-1,k}$. Таким образом, количество новых узлов, возникающих на i-й итерации, не может превысить длины варьируемой части массива TH, т.е. $k^*(i)$.

Поскольку $\sum_{i=1}^{M} k^*(i) \le \sum_{i=1}^{M} i = M(M+1)/2$, то суммарное количество узлов ограничено величиной $r^* = \min(r, M(M+1)/2)$.

Процедура запоминания узлов оперирует с двумя массивами: LINK[0 : M] и LIST[1:r*] . Значения LINK[k] после i-й итерации указывают порядковые номера узлов в массиве LIST, с которых начинается восстановление МДП длины k последовательностей A[1:i] и B[1:j(k)] , где $j(k)$ - минимально необходимое (для существования МДП длины k) число элементов из B . Массив LIST - список узлов. Каждый элемент его имеет структуру ⟨ i, j, LINK[k-1] ⟩ , где

i и j - номера позиций k-го элемента МДП в последовательностях A и B соответственно, а LINK[k-1] - ссылка на узел, содержащий информацию о (k-1)-м элементе МДП. Массив LIST заполняется последовательно по мере формирования новых узлов. Если, к примеру, он уже содержит (L-1) элемент и в ходе i-й итерации элемент TH[k] меняет свое значение (что эквивалентно формированию нового узла), то информация об этом узле заносится в LIST[L], а LINK[k] принимает значение L.

3. С учетом замечаний 1 и 2 полная трудоемкость модифицированного алгоритма не превышает $O(N + \min(M \cdot N, r \cdot \log_2 N_{эф.}))$ при затратах памяти $O(r^*)$.

Таким образом, на примере алгоритма Ханта и Шиманского показано, что введение пошаговой адаптации (в дополнение к используемой в [3] адаптации по параметру r) позволяет избежать неоправданных затрат по времени и по памяти в ситуациях, когда $r \sim N^2$, что гарантирует квадратичную трудоемкость в наихудшем случае. При всех других значениях r порядок трудоемкости не выше, чем в [3], а мультипликативная постоянная, как правило, меньше.

Литература

1. AHO A.V., HIRSCHBERG D.S., ULLMAN J.D. Bounds on the complexity of the longest common subsequence problem.- J.Assoc.Comput.Machinery, 1976, v.23, N 1, p.1-12.

2. WAGNER R.A., FISCHER M.J. The string-to-string correction problem.-J.Assoc.Comput.Machinery, 1974, v.21, N 1, p.168-173.

3. HUNT J.W., CZYMANSKI T.G. A fast algorithm for computing longest common subsequences. - Communs ACM, 1977, v.20, N 8, p. 350-353.

4. HIRSCHBERG D.S. Algorithms for the longest common subsequence problem.- J.Assoc.Comput.Machinery, 1977, v.24, N 4, p. 664 - 675.

5. MASEK W.J., PATERSON M.S. A faster algorithm computing string edit distances. - J.Comput.and System.Sci., 1980, v.20, N 1, p.18-31.

6. ЛАВРОВ С.С., ГОНЧАРОВ Л.И. Автоматическая обработка данных. Хранение информации в памяти ЭВМ. -М.: Наука, 1971. - 160 с.

Поступила в ред.-изд.отд.
5 ноября 1981 года