Оценка эффективности алгоритма поиска наибольшей общей подстроки с помощью суффиксного автомата.

I. Сравним эффективность следующих алгоритмов поиска наибольшей общей подстроки:
   1) Алгоритм, использующий динамическое программирование.
   2) Алгоритм, построенный с помощью суффиксного автомата.

II. Сравнение проведем по следующим параметрам:
   1) Зависимость используемой памяти от количества символов в наименьшей строке.
   2) Зависимость времени работы от количества символов в наименьшей строке.

III. Оценим относительную эффективность алгоритмов.

I. Краткое описание алгоритмов:
   1) <u>Суффиксный автомат.</u>
      1. Построение.

Алгоритм онлайновый, т.е. будет добавляет по одному символу строки s, перестраивая соответствующим образом текущий автомат.

Чтобы достичь линейного потребления памяти, в каждом состоянии мы будем хранить только значение length, suffix и список переходов из этого состояния. Есть начальное состояние terminal, для которого length = 0, suffix=-1, даёт сылку на фиктивное состояние.

Создадим новое состояние длины length+1. Если из него нет перехода по букве c, то добавляем его, и затем переходим по суффиксной ссылке, снова проверяя — если нет перехода, то добавляем. Если в какой-то момент случится, что такой переход уже есть, то если длина перехода равна 1, то связываем эти два состояния, нет – «клонируем» состояние.

Если ни разу не случилось, что переход по букве c уже имелся, и мы так и дошли до фиктивного состояния -1 (в которое мы попали по суффиксной ссылке из начального состояния t_0), то мы можем просто присвоить $link(cur) = 0$ и выйти.

      2. Алгоритм поиска.

Построим суффиксный автомат по наименьшей строке S.

Будем теперь идти по строке T, и для каждого префикса искать наидлиннейший суффикс этого префикса, встречающийся в S.

Введём две переменные: текущее состояние и текущую длину. Изначально совпадение пустое.

На i-м шаге рассматриваем символ $T[i]$ и пересчитываем ответ для него. Есть переход по символу – просто увеличим длину на единицу. Нет – укорачиваем совпадающую часть.

Ответом на задачу будет максимум из значений l за всё время обхода.

2) Динамическое программирование.

Решение задачи поиска наибольшей общей подстроки для двух строк s_1 и s_2, длины которых m и n соответственно, заключается в заполнении таблицы A_{ij} размером $(m+1) \times (n+1)$ по следующему правилу, принимая, что символы в строке нумеруются от единицы.

$$\begin{cases} A_{0j} = 0, & j = 0 \dots n, \\ A_{i0} = 0, & i = 0 \dots m, \\ A_{ij} = 0, & s_1[i] \neq s_2[j], i \neq 0, j \neq 0, \\ A_{ij} = A_{i-1j-1} + 1, & s_1[i] = s_2[j], i \neq 0, j \neq 0. \end{cases}$$

Максимальное число A_{uv} в таблице это и есть длина наибольшей общей подстроки, сама подстрока:

$s_1[u - A_{uv} + 1] \dots s_1[u]$ и $s_2[v - A_{uv} + 1] \dots s_2[v]$.

II. Оценочные графики.

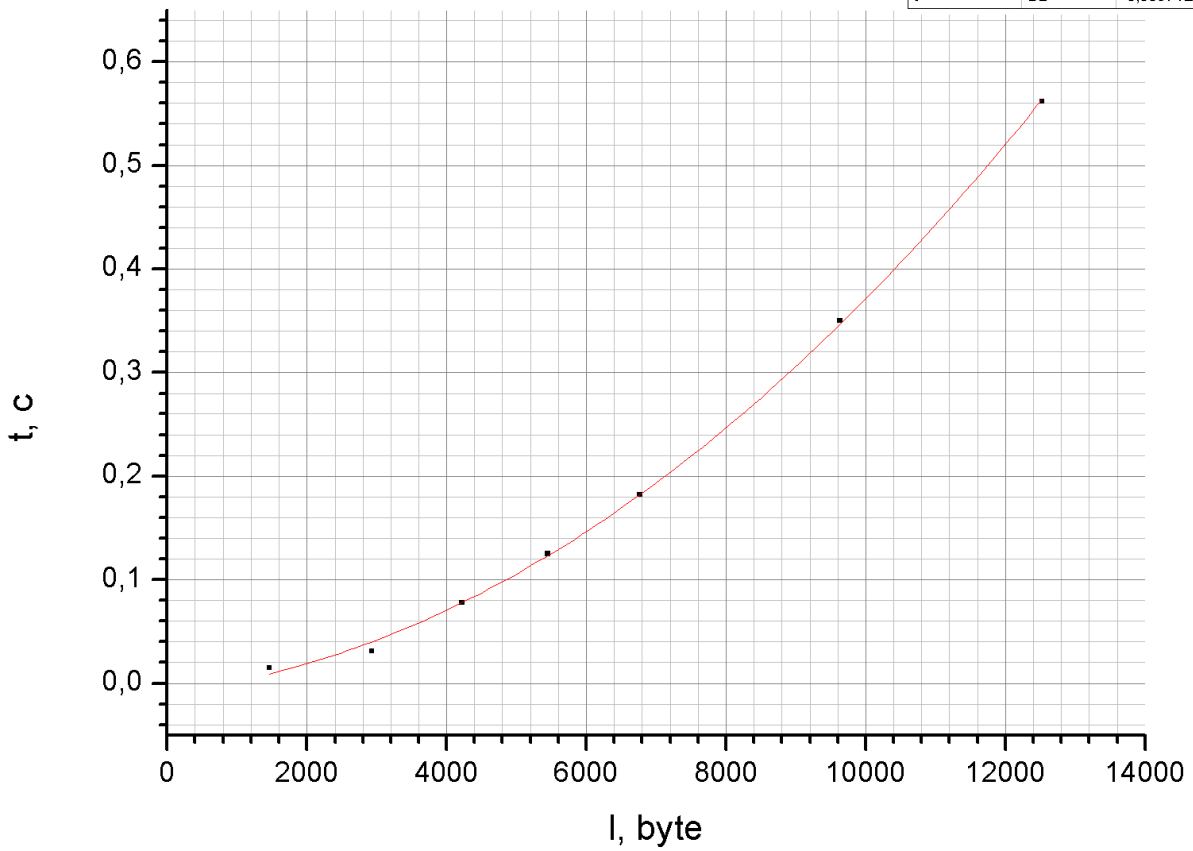
Наблюдения показали, что от длины искомой подстроки время не зависит ни в одном случае при возрастании общей подстроки от нескольких букв до последовательностей в 2200+ символов.

Рассмотрим зависимости времени работы программы от длины файла.

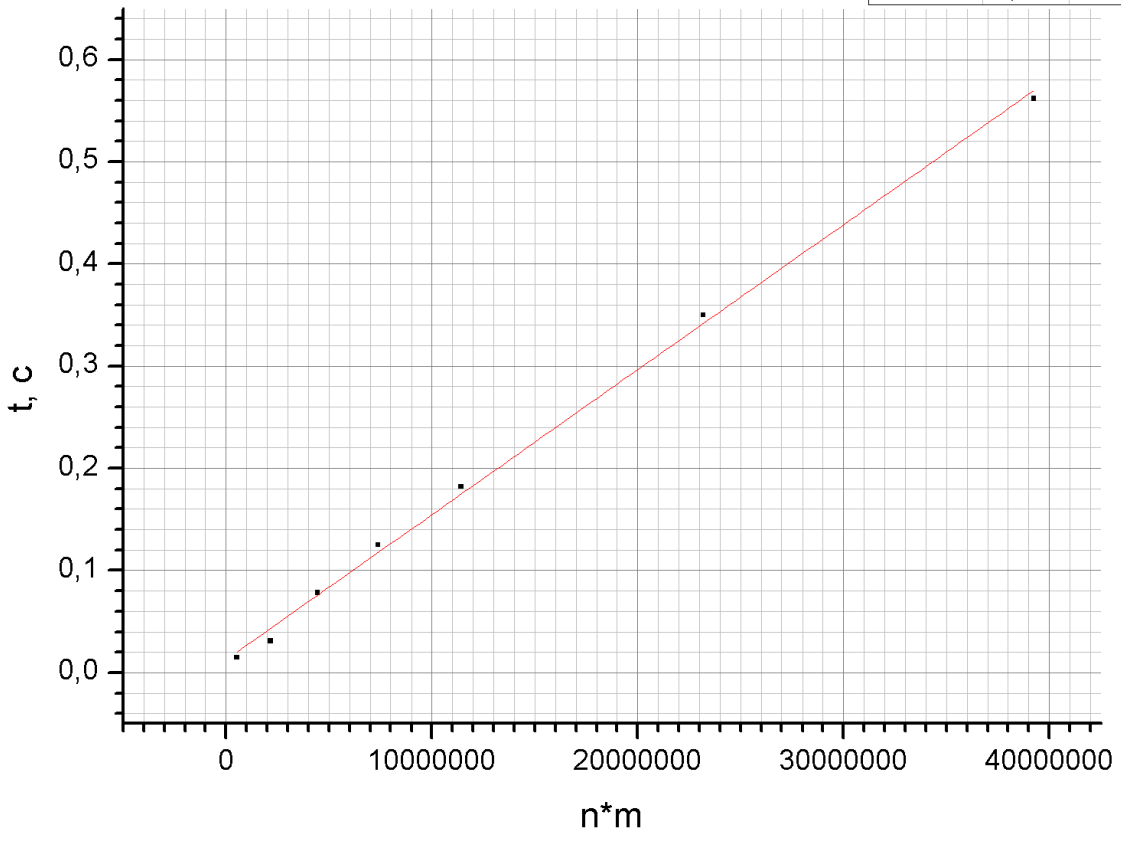
Первый рассматриваемый алгоритм – алгоритм с использованием динамического массива.

	n	m	nm(X)	t(Y)	A
Name			n*m		
Units					
ments					
1	661	803	530783	0,015	1464
2	1464	1466	2,14622E6	0,031	2930
3	2012	2210	4,44652E6	0,078	4222
4	2931	2518	7,38026E6	0,125	5449
5	3591	3178	1,14122E7	0,182	6769
6	4871	4759	2,31811E7	0,35	9630
7	6235	6294	3,92431E7	0,562	12529

Model	Polynomial		
Adj. R-Square	0,99917		
		Value	Standard Error
t	Intercept	-0,00839	0,00787
t	B1	7,58978E-6	2,65422E-6
t	B2	3,03974E-9	1,83659E-10



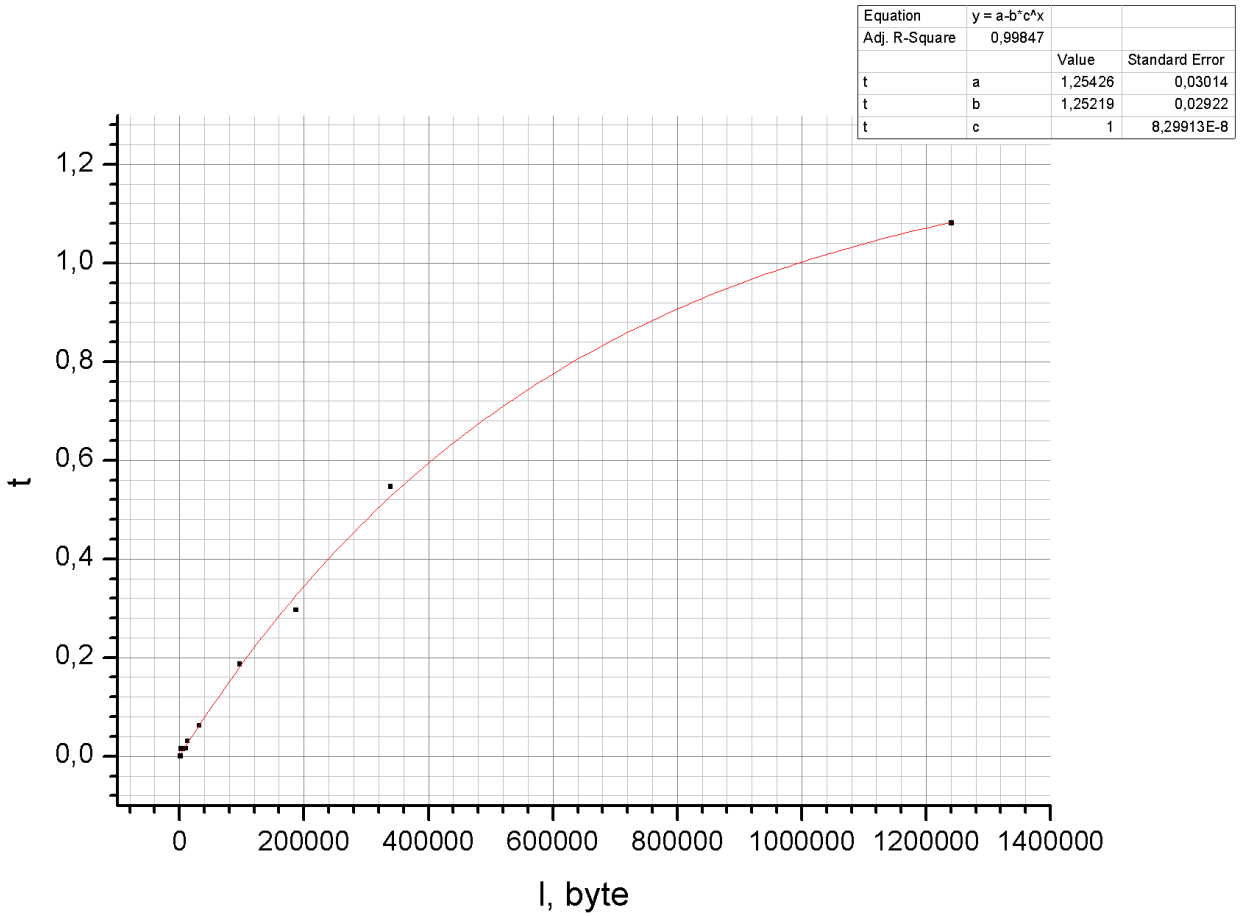
Equation	y = a + b*x		
Adj. R-Square	0,99788		
		Value	Standard Error
t	Intercept	0,01266	0,00483
t	Slope	1,41994E-8	2,67223E-10



Для данных графиков мы видим, что время выполнения зависит линейно от произведения длин двух строк, то есть ассимптотика алгоритма O(n*m), где n и m – длины сравниваемых строк.

Следующий алгоритм – поиск наибольшей общей подстроки с помощью суффиксного автомата.

	n	m	nm(X)	t(Y)
ame			n+m	
Jnits				
ents				
1	661	803	1464	0
2	1464	1466	2930	0,015
3	2012	2210	4222	0,015
4	2931	2518	5449	0,015
5	3591	3178	6769	0,016
6	4871	4759	9630	0,016
7	6235	6294	12529	0,031
8	12519	19045	31564	0,062
9	48626	48146	96772	0,187
10	93723	93415	187138	0,297
11	176542	162345	338887	0,547
12	623487	617294	1,24078E6	1,081



На данном графике можно заметить, что зависимость времени выполнения от длины файла – логарифмическая. Ассимптотика алгоритма может выглядеть как O(log(n+m)). Даже предположив «удачность» крайнего теста можно увидеть, что зависимость в худшем случае линейная, т.е. ассимптотика алгоритма – O(n+m).

III. Относительная эффективность алгоритмов.

Из данных выше можно сделать вывод о том, что алгоритм, использующий суффиксное дерево, работает быстрее, что особенно видно на объемах данных, подходящих к файлам длиной в мегабайт. Память, занимаемая динамическим массивом пропорциональна n*m, память же, занимаемая суффиксным деревом – пропорциональна 2n (из алгоритма построения), где n – длина кратчайшей строки, что доказывает оптимальность решения в плане задействованной памяти.

Таким образом, суффиксный автомат позволяет решать задачи со строками с минимальными затратами ресурсов (памяти и времени).