САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра алгебры и теории чисел

О вычислительной сложности определения вложимости сжатых текстов

Дипломная работа студента 511-ой группы
Лифшица Юрия Михайловича

Научный руководитель,
д. ф.-м. н., профессор ..... /Ю. В. Матиясевич/
кафедры алгебры и теории чисел /подпись/

Рецензент,
к. ф.-м. н., доцент ..... /Э. А. Гирш/
кафедры информатики /подпись/

“Допустить к защите”,
заведующий кафедрой, ... /А. В. Яковлев/
д. ф.-м. н., профессор /подпись/

Санкт-Петербург
2005

Аннотация

В работе рассматривается известная задача обработки сжатых текстов. Мы изучаем следующий вопрос (**Вложимость**): является ли один сжатый текст подпоследовательностью в другом сжатом тексте? До сих пор вычислительная сложность этой задачи была неизвестна. В этой работе мы докажем что **Вложимость** NP- и co-NP-трудна.

1 Введение

В связи со стремительным ростом объемов информации все большее внимание уделяется алгоритмам обработки сжатых объектов **без разархивирования**. Изучается обработка сжатых изображений, деревьев, схем. Одним из центральных понятий является сжатие строки (слова). В 1977 году Лемпель и Зив предложили свою модель архивирования [13], которая получила широкое распространение. В девяностые годы для построения алгоритмов стали использовать модель, основанную на грамматиках - *прямолинейные программы* (Straight-Line Programs). Мы будем пользоваться прямолинейными программами.

1.1 Основные понятия и задачи

Мы рассматриваем конечный алфавит Σ, словом называется любая последовательность его букв, множество всех слов традиционно обозначается как Σ^*.

Базовым понятием данной работы являются *прямолинейные программы*. Неформально, прямолинейная программа – это контекстно-свободная грамматика, порождающая только одно слово.

Определение 1. Прямолинейной программой называется контекстно-свободная грамматика $\mathcal{P}$, в которой нетерминальные символы $X_1, \ldots, X_m$ упорядочены (X_m – стартовый символ), и где у каждого не-терминального символа есть только одно правило: $X_i \to a$, где a — терминал, или $X_i \to X_j X_k$ для некоторых $j, k < i$.

Для прямолинейной программы $\mathcal{P}$ мы будем обозначать $eval(\mathcal{P})$ единственное слово, которое описывается $\mathcal{P}$. Таким образом $eval(\mathcal{P}) = eval(X_m)$.

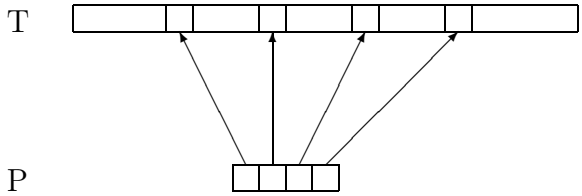
Поясним происхождение термина “прямолинейная программа”. Дело в том, что любой текст, представленный с помощью вышеописанных контекстно-свободных грамматик, может быть порожден с помощью программ, использующих в точности один оператор — оператор присваивания.

Основными задачами, которые изучаются для сжатых текстов, являются следующие:

- **Сжатое Равенство**: дано два сжатых текста, требуется определить, совпадают они или нет.
- **(Простой) Сжатый Поиск Подстроки**: дан шаблон и сжатый текст, требуется определить, является ли шаблон подстрокой текста, и при положительном ответе найти первое вхождение.
- **Полный Сжатый Поиск Подстроки**: дан **сжатый** шаблон и сжатый текст, требуется определить, является ли шаблон подстрокой текста, и при положительном ответе найти первое вхождение.
- **Принадлежность Языку**: зафиксирован некоторый язык. Требуется по данному сжатому слову определить, принадлежит оно языку или нет.

Вычислительная сложность этих задач рассматривалась в целом ряде работ [3, 8, 10, 5]. Первые три задачи имеют полиномиальную сложность. Принадлежность регулярному языку также решается за полиномиальное время, а принадлежность контекстно-свободному уже оказывается PSPACE-полной [5]. После изучения этих базовых задач потребовалось найти эффективные алгоритмы и для других постановок. Одной из ближайших к **Сжатому Поиску Подстроки** является следующая задача:

- **Сжатый Поиск Подпоследовательности** (для краткости — **Вложимость**). Дана сжатая строка P (шаблон) и сжатая строка T (текст). Требуется определить, являются ли буквы шаблона подпоследовательностью в строке T (обозначение — $P \hookrightarrow T$).



Как ни странно, долгое время вычислительная сложность **Вложимости** оставалась совершенно неясной. До недавнего времени не было ни одной разумной оценки сложности этой задачи, она могла оказаться как решаемой за полиномиальное время, так и PSPACE-полной. В данной работе удалось получить нижние оценки, показав, что задача не только NP-трудна, но и (при предположении NP≠co-NP) лежит вне класса NP.

1.2 Предыдущие исследования

В работе [3], развивающей идеи [7], был построен полиномиальный алгоритм для задачи **Полного Сжатого Поиска Подстроки**. Кроме того, авторы этой работы представили полиномиальный алгоритм проверки принадлежности сжатого слова регулярному языку.

Недавно удалось доказать [6], что принадлежность регулярному языку является P-полной.

Еще одним важным результатом является эффективное приближенное построение в [11] по данному тексту минимальной прямолинейной программы (ПП), порождающей этот текст. Более формально, построен алгоритм, работающий за время $O(n \cdot \log|\Sigma|)$, который по тексту длины n строит $O(\log n)$-приближение минимальной ПП, порождающей этот текст. Это означает, что поиск адекватного (близкого к максимально возможному) сжатия в модели прямолинейных программ может быть выполнен эффективно.

Кроме того, следует упомянуть работу [1], в которой задачи **Сжатого Поиска Подстроки** обобщены на двумерные тексты. Как оказалось, при этом резко возрастает вычислительная сложность. **Простой Сжатый Поиск Подстроки** является NP-полным, а **Полный Сжатый Поиск Подстроки** является Σ_2^P-полным.

Строки являются элементами свободного конечно-порожденного моноида. В работе [5] изучается обработка сжатых элементов различных классов конечно-порожденных моноидов. В зависимости от ограничений

на моноид были получены доказательства полноты задачи **Сжатого Равенства** для классов P, co-NP, PSPACE и EXPSPACE.

Недавно найдены (см. [4]) приложения алгоритмов обработки сжатых текстов для анализа диаграмм последовательностей сообщений (message sequence charts) .

В качестве обзоров по обработке сжатых текстов можно порекомендовать [9, 12].

1.3 Мотивация и полученные результаты

В случае обычных текстов, сложность поиска подпоследовательности линейна, то есть эта задача не сложнее, чем поиск подстроки. Так как был получен полиномиальный алгоритм поиска подстроки в сжатых текстах, встал вопрос о существовании аналогичного алгоритма для поиска подпоследовательности. При естественном предположении (P≠NP) в нашей работе получен отрицательный ответ на этот вопрос.

1.4 Схема доказательств

В статье представлено два основных результата. Мы начнем со сведения известной NP-полной задачи (**Сумма Размеров**) к задаче **Вложимости**. Таким образом мы получим NP-трудность последней. Следующим шагом будет сведение **Вложимости** к **Невложимости** и наоборот. Немедленным следствием из этой симметричности станет co-NP-трудность изучаемой задачи.

2 NP-трудность

Напомним формулировку известной NP-полной задачи **Сумма Размеров** (см. [2]):

- Дана двоичная запись натуральных чисел $w_1, \ldots, w_n, t$. Требуется определить, существуют ли $x_1, \ldots, x_n \in \{0, 1\}$ такие, что $\sum_{i=1}^{n} x_i \cdot w_i = t$.

Теорема 1. Сумма Размеров *сводится к* **Вложимости**.

Доказательство. Итак пусть $t, \bar{w} = \langle w_1, \ldots, w_n \rangle$ – входные данные **Суммы Размеров** (будем считать $n > 1$). Мы построим такие прямолинейные программы $\mathcal{F}$ и $\mathcal{G}$, что подмножество $\bar{w}$ с суммой t существует тогда и только тогда, когда $eval(\mathcal{F}) \hookrightarrow eval(\mathcal{G})$.

Введем обозначения $s = w_1 + \cdots + w_n$, $N = 2^n s$. Каждому подмножеству $\bar{w}$ можно поставить в соответствие строчку $x = \overline{x_1 \ldots x_n}$ длины n из нулей и единиц, то есть целое число от 0 до $2^n - 1$. Введем обозначение $x \circ \bar{w} = \sum_{i=1}^{n} x_i w_i$, фактически, $x \circ \bar{w}$ дает сумму подмножества $\bar{w}$, кодируемого числом x.

Конструкция:

$$G = G_0^{5N} \qquad G_0 = G_1 G_2 G_3 G_4$$

$$G_1 = \prod_{x=0}^{2^n-1} (10^s) = (10^s)^{2^n} \qquad G_2 = 0^{2N}$$

$$G_3 = \prod_{x=0}^{2^n-1} (0^{x \circ \bar{w}} 10^{s - x \circ \bar{w}}) \qquad G_4 = 0^{t+1}$$

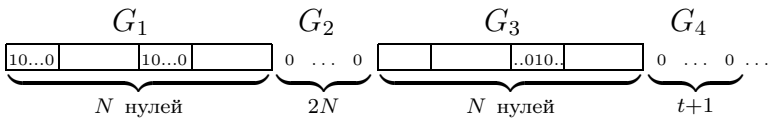
$$F = F_0^{5N-1} \qquad F_0 = 10^{3N+t} 10^{N+1}$$

Мы использовали знак $\prod$ для обозначения конкатенации соответствующих слов в порядке изменения индекса от нижнего предела к верхнему.

Этими равенствами мы определили тексты F и G. Ниже мы докажем, что они могут быть порождены прямолинейными программами $\mathcal{F}$ и $\mathcal{G}$. Но сначала убедимся, что вложимость текстов F и G равносильна существованию подмножества $\bar{w}$ с суммой t.

Равносильность существования подмножества с заданной суммой и вложимости.

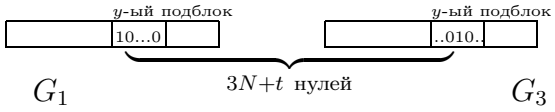
"Существует подмножество $\Rightarrow$ вложимость". Пусть x — код подмножества $\bar{w}$ с суммой t, то есть $x \circ \bar{w} = t$. Рассмотрим начало G, т.е. $G_1 G_2 G_3 G_4 G_1 \ldots$, и выделим следующее вложение F_0: отметим 1 в x-ом блоке G_1, затем $3N + t$ нулей, далее (в точности потому что $x \circ \bar{w} = t$) стоит 1 и отметив следующие $N + 1$ нулей мы окажемся перед x-ой единицей во втором G_1, таким образом имея $5N$ блоков $G_0 = G_1 G_2 G_3 G_4$ мы сможем вложить $5N - 1$ копий F_0. Что, собственно, и требовалось проверить: $F \hookrightarrow G$.



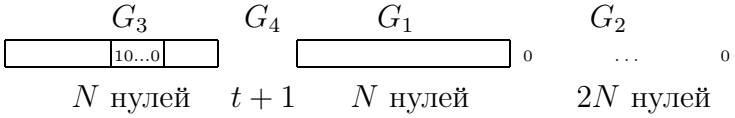
"Вложимость $\Rightarrow$ существует подмножество". Пусть $F \hookrightarrow G$. Предположим также, что ответ в **Сумме Размеров** отрицательный. Выведем из этих двух утверждений противоречие.

Мы будем искать нули в G, которые не задействованы во вложении F в G. Это вложение представляет собой $5N-1$ непересекающихся вложений F_0 в G. Слово F_0 содержит две единицы, между которыми расположено ровно $3N+t$ нулей.

Убедимся, что в G нет двух единиц, между которыми расположено в точности столько же нулей, от противного. Рассмотрим два случая: первая единица находится в блоке G_1 (скажем, в подблоке номер y). Тогда, сдвинувшись на $3N+t$ нулей, мы попадем в y-ый подблок G_3. Если бы в этом подблоке после t нулей стояла единица, то мы бы получили $y \circ \bar{w} = t$ — противоречие с несуществованием подмножества с суммой t.



Второй случай: левая единица лежит в блоке G_3. Тогда, сдвинувшись на $3N+t$ нулей мы попадем внутрь блока G_2, где вообще нет единиц.



Следовательно, при каждом вложении F_0 хотя бы один ноль (между двух единиц) остается незадействованным. Осталось лишь оценить с двух сторон количество нулей в G. По построению, их там $5N \cdot (4N+t+1)$. С другой стороны, каждое слово F_0 содержит $4N+t+1$ нулей и еще хотя бы $5N-1$ ноль не задействован. Таким образом общее количество нулей в G должно быть не меньше, чем $(5N-1) \cdot (4N+t+1) + 5N - 1 = 5N \cdot (4N+t+1) + (N-t-2) > 5N \cdot (4N+t+1)$. Поясним последнее неравенство: $N = s2^n \geq 4s > t+2$. То есть мы получили больше нулей в G, чем там есть по построению, что и дает нам противоречие.

Реализация F и G в виде прямолинейных программ. Заметим, что, за одним исключением, F и G строятся из 0 и 1 только с помощью полиномиального числа конкатенаций и возведения в степень (двоичная запись всех использованных степеней – полиномиального размера). Такие конструкции напрямую реализуются сжатыми словами полиномиального размера. Единственным нетривиальным блоком является G_3. Конструкция сжатой версии G_3 полиномиального размера впервые была предложена Маркусом Лори в работе [5]. Повторим ее здесь:

Мы хотим построить прямолинейную программу, вычисляющую

$$G_3 = \prod_{\bar{x}=0}^{2^n-1} (0^{x \circ \bar{w}} 10^{s - x \circ \bar{w}})$$

Вот эта конструкция:

$$S_1 \to 10^{s+w_1}1$$

$$S_{k+1} \to S_k 0^{s-s_k+w_{k+1}} S_k$$

Мы возьмем S_n как стартовый символ и докажем, что он вычисляет в точности G_3, с помощью индукции.

Утверждение: $eval(S_k) = \left(\prod_{\bar{x} \in \{0,1\}^k \setminus \{\bar{1}_k\}} (0^{\bar{x} \cdot \bar{w}_k} 10^{s - \bar{x} \cdot \bar{w}_k}) \right) 0^{s_k} 1.$

Для $k = 1$ утверждение сводится к равенству $10^{s+w_1}1 = 0^0 10^{s-0} 0^{s_1} 1$. Для $k+1 \le n$ мы получим следующую цепочку равенств:

$$eval(S_{k+1}) = \left(\prod_{\bar{x} \in \{0,1\}^{k+1} \setminus \{\bar{1}_{k+1}\}} (0^{\bar{x} \cdot \bar{w}_{k+1}} 10^{s - \bar{x} \cdot \bar{w}_{k+1}}) \right) 0^{s_{k+1}} 1 =$$

$$\underbrace{\left(\prod_{\bar{x} \in \{0,1\}^k} (0^{\bar{x} \cdot \bar{w}_k} 10^{s - \bar{x} \cdot \bar{w}_k}) \right)}_{eval(S_k) 0^{s-s_k}} \underbrace{\left(\prod_{\bar{x} \in \{0,1\}^k \setminus \{\bar{1}_k\}} (0^{\bar{x} \cdot \bar{w}_k + w_{k+1}} 10^{s - \bar{x} \cdot \bar{w}_k - w_{k+1}}) \right) 0^{w_{k+1}} 0^{s_k} 1}_{0^{w_{k+1}} eval(S_k)} =$$

$$eval(S_k) 0^{s - s_k + w_{k+1}} eval(S_k) = eval(S_{k+1}),$$

что доказывает утверждение.

Осталось заметить, что при $k = n$

$$eval(S_n) = \prod_{\bar{x} \in \{0,1\}^n} \left(0^{\bar{x} \cdot \bar{w}} 10^{s - \bar{x} \cdot \bar{w}}\right) = G_3.$$

Полиномиальность сведения. Для завершения доказательства необходимо убедиться, что построение прямолинейных программ, вычисляющих F_1 и G_1 полиномиально относительно размера входных данных задачи **Subset Sum**. Чтобы убедиться в этом, достаточно заметить, что в построении F_1 и G_1 мы использовали лишь конкатенации и возведение в степени, являющиеся результатами арифметических действий над t и $w_1, \ldots, w_n$. □

Следствие 1. Вложимость *NP-трудна.*

3 co-NP-трудность

Теорема 2. Вложимость *сводится (по Карпу) к* **Невложимости**. **Невложимость** *сводится к* **Вложимости**.

Доказательство. Мы будем доказывать следующее утверждение: существует полиномиальный алгоритм, позволяющий для любых сжатых слов F и G построить сжатые слова F_1 и G_1 (естественно, что их размер может быть лишь полиномиально больше, чем у F и G), такие что

$$F \hookrightarrow G \Leftrightarrow F_1 \not\hookrightarrow G_1. \qquad (*)$$

В случае унарного алфавита обе задачи лежат в P. Будем считать, что в алфавите хотя бы 2 буквы. Заметим, что за полиномиальное время можно вычислить последнюю букву F и приписать к G другую букву (получив G') и при этом $F \hookrightarrow G \Leftrightarrow F \hookrightarrow G'$. Так что, не умаляя общности, можно считать, что F и G заканчиваются на разные буквы.

Пусть $F = f_1 \ldots f_k$, $G = g_1 \ldots g_m$. Для каждой буквы алфавита a введем обозначение $X_a = (\Sigma/a)^{m+1}$, где (Σ/a) – конкатенация всех букв алфавита, кроме a, в любом порядке.

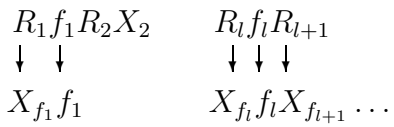
Конструкция:

$$F_1 = G = g_1 \ldots g_m$$

$$G_1 = X_{f_1} f_1 \ldots X_{f_k} f_k$$

Проверка свойства $(*)$: Мы можем представить G в следующем виде: $R_1 f_1 \ldots R_l f_l R_{l+1}$, где R_i не содержит буквы f_i. Утверждение $F \hookrightarrow G$ равносильно выполнению равенства $l = k$ для нашего представления.

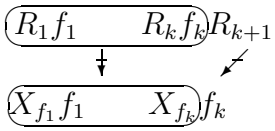
Если $l < k$, то $F_1 \hookrightarrow G_1$, так как для каждого $1 \le i \le l+1$ выполнено $R_i \hookrightarrow X_{f_i}$ и $f_i = f_i$.



Если $l = k$, то $R_{l+1} \neq \varnothing$, так как g_m по предположению не равно f_k. По индукции можно убедиться в том, что

$$R_1 f_1 \ldots R_i f_i \not\rightarrow X_{f_1} f_1 \ldots X_{f_i}.$$

Действительно, при $i = 1$ это следует из факта $f_1 \not\rightarrow X_{f_1}$. Переход $i \to i+1$: если бы вложение существовало бы для $i+1$, то, так как $f_{k+1} \not\rightarrow X_{f_{k+1}}$, то f_{k+1} попадало бы не правее f_k, а значит было бы вложение и для i, что противоречит предположению индукции. Рассмотрим наше утверждение при $i = k$ и в сумме с фактом $R_{k+1} \not\rightarrow f_k$ мы получим $F_1 \not\rightarrow G_1$.



Полиномиальность конструкций: Заметим, что X_a строится за полиномиальное от $\log m$ время. Для построения G_1 остается лишь добавить правила вида $A \to X_a a$ для всего алфавита и использовать в конструкции эти нетерминальные символы вместо соответствующих терминалов. □

Следствие 2. Вложимость *co-NP-трудна.*

Доказательство. Согласно Теореме 1 **Вложимость** NP-трудна. Следовательно, так как мы используем сведение по Карпу, **Невложимость** - co-NP-трудна. Так как **Невложимость** сводится к **Вложимости** (Теорема 2), то **Вложимость** также является co-NP-трудной. □

4 Выводы и открытые вопросы

В работе рассмотрена задача поиска подпоследовательности в сжатых текстах. Удалось получить нижние оценки ее вычислительной сложности. Мы построили доказательство (при предположении NP≠co-NP), что эта задача лежит за пределами класса NP. С другой стороны, известна

только тривиальная верхняя оценка сложности — легко построить алгоритм из класса PSPACE. Главным открытым вопросом остается сближение этих оценок.

Список литературы

[1] Piotr Berman, Marek Karpinski, Lawrence L. Larmore, Wojciech Plandowski and Wojciech Rytter. *On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts*, Journal of Computer and Systems Science, vol. 65, number 2, pp. 332–350, 2002.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman, 1979.

Русский перевод: М. Гэри и Д. Джонсон *Вычислительные машины и труднорешаемые задачи*, Москва, Мир, 1982.

[3] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski and Wojciech Rytter. *Efficient Algorithms for Lempel-Ziv Encoding (Extended Abstract)*, Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996), Springer-Verlag, LNCS 1097, pp. 392–403, 1996.

[4] Blaise Genest and Anca Muscholl. *Pattern Matching and Membership for Hierarchical Message Sequence Charts*, In Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002), Springer-Verlag, LNCS 2286, pp. 326–340, 2002.

[5] Markus Lorhey. *Word problems on compressed word*, ICALP 2004, Springer-Verlag, LNCS, 3142, pp. 906–918, 2004.

[6] N. Markey and Ph. Schnoebelen. *A PTIME-complete matching problem for SLP-compressed words*, Information Processing Letters, vol. 90, number 1, pp. 3–6, 2004.

[7] Wojciech Plandowski. *Testing Equivalence of Morphisms on Context-Free Languages*, Second Annual European Symposium on Algorithms (ESA'94), Utrecht (The Netherlands), Springer-Verlag, LNCS 855, pp. 460–470, 1994.

[8] Wojciech Plandowski and Wojciech Rytter. *Complexity of Language Recognition Problems for Compressed Words*, Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, Springer-Verlag, pp. 262–272, 1999.

[9] Wojciech Rytter. *Algorithms on Compressed Strings and Arrays*, Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99), Springer-Verlag, LNCS 1725, pp. 48–65, 1999.

[10] Wojciech Rytter. *Compressed and fully compressed pattern matching in one and two dimensions*, Proceedings of the IEEE, vol. 88, number 11, pp. 1769–1778, 2000.

[11] Wojciech Rytter. *Application of Lempel-Ziv factorization to the approximation of grammar-based compression*, Theoretical Computer Science, vol. 302, number 1–3, pp. 211–222, 2003.

[12] Wojciech Rytter. *Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input*, Proceedings of the 31st International Colloquium on Automata, Languages and Programming(ICALP 2004), Springer-Verlag, LNCS 3142, pp. 15–27, 2004.

[13] J. Ziv and A. Lempel. *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory, vol. 23, number 3, pp. 337-343, 1977.