Algoritmo Linear para Detecção de Unidades de Repetição em *Tandem Arrays*

Luiz H. B. Lago[1], **Bruna da Silva Righi**[1]

[1]Universidade Federal de Santa Maria (UFSM)
97.105-900 – Santa Maria – RS – Brazil

lhlago@inf.ufsm.br, righi.bruna@acad.ufsm.br

Abstract. *Detecting repetitive sequences is relevant in many areas of knowledge, such as genetics, data compression, and text processing. In this article, a solution to the problem of identifying the smallest sequential repetition in a string is addressed. Tandem arrays are a special group of strings made out of the repetition of smaller non-empty strings. The algorithm analyzed in this article has its focus on detecting the smallest repetitive unit of a tandem array in linear time, according to the size of the input string. It also validates the functionality of the algorithm.*

Resumo. *A detecção de repetições é relevante em diversas áreas, como em genética, compressão de dados e processamento de texto. Neste artigo é abordada uma solução para o problema de identificar a menor sequência repetida em uma string. Tandem arrays são um grupo especial de string formadas pelas repetições de strings menores não-vazias. O algoritmo analisado neste artigo tem enfoque na detecção da menor unidade de repetição de um tandem array em tempo linear, em relação ao tamanho da string de entrada. Também é feito uma validação da funcionalidade do algoritmo.*

1. Introdução

Em genética, sequências repetitivas de ácidos nucleicos, também chamadas de *tandem arrays*, são sequências de DNA que aparecem adjacentes, seja de forma direta ou inversa. Essas repetições possuem utilidade no mapeamento do genoma humano, na identificação de doenças, e em estudos forenses e populacionais, e são computáveis a partir da identificação de padrões [C. S. Góes 2005, Sokol et al. 2007]. Além disso, a identificação de sequências repetidas é relevante para áreas como compressão de dados, na identificação de *runs* [Peterson and Davie 2022], e para edição e processamento de textos no geral [Aho 1990]. Este artigo apresenta um algoritmo vantajoso para a computação de sequências repetidas, que pode ser utilizado principalmente para a detecção de *tandem arrays*.

2. Definição do Problema

Seja $\sum$ um alfabeto finito cujos elementos são denominados letras; uma *string* S é uma sequência finita de letras de $\sum$. O comprimento de S é o número de ocorrências de letras na sequência, denotado por $|S|$. Dado um índice i e um índice j, com $0 \leq ij \leq |S|$, denotamos por $S[i : j]$ uma *substring* de S que começa na i-ésima posição e termina na

j-ésima posição [Nakamura et al. 2013]. Por um padrão de consistência com o pseudo-código, iremos adotar o primeiro índice da *string* como sendo de valor 0.

Um *tandem array* é uma *string* S que pode ser representada como a seguinte concatenação $UU...U = S$, onde U é uma *substring* de S chamada **unidade de repetição**. De forma mais simplificada, podemos representar um *tandem array* usando a *Repetition Representation (of a String)* (RRS)[Nakamura et al. 2013], no qual utilizamos o formato U^k ao invés de $UU..U$, onde U repete k vezes.

Com as definições já estabelecidas, o problema pode ser especificado da seguinte forma: dado uma *string* S, determinar se ela é um *tandem array*, e se for, determinar uma das **unidades de repetição** U tal que $|U|$ seja o menor possível.

3. Solução

Para a comparação dos algoritmos aqui presentes, será utilizada a notação *Big O*, que define o comportamento limitante de uma função, ou seja, a quantidade de operações realizadas por cada algoritmo quando a entrada cresce tendendo ao infinito [Mala and Ali 2022].

3.1. Algoritmo ingênuo

Conforme especificado, U é uma substring de S, e por ser periódico, pode ser representado usando RRS por $U^k = S$, e de forma trivial, é possivel perceber que $U = S[0 : T]$ para algum $T \in \{1, 2, ...|S|\}$. Dessa forma, o problema pode ser reduzido a encontrar o menor valor de T válido. O **algoritmo ingênuo** verifica cada possível T comparando todos os índices em S segundo a relação $S[i \bmod T] = S[i]$. Esse algoritmo apresenta complexidade temporal $O(N^2)$ em função do tamanho da *string*. É possível notar que o algoritmo parte do princípio que a *string* de entrada é um *tandem array* e tenta encontrar uma resposta mesmo que não exista, e caso a entrada não seja um *tandem array*, ele irá retornar $T = |S|$.

3.2. Algoritmo proposto

A solução aqui proposta busca reduzir o número de verificações necessárias, reaproveitando as verificações anteriores para confirmar se determinada *substring* é de fato válida. Dessa forma, quando $S[i \bmod T] \neq S[i]$, descartamos a possibilidade de $S[0 : T]$ ser a resposta para o problema e incrementamos em um valor de T para continuar a checagem.

```
1: function SOLUTION(string S[N])
2:     T ← 1
3:     for i ← 0 to 2N do
4:         while (S[i mod T] ≠ S[i mod N]) do
5:             T ← T + 1
6:         end while
7:     end for
8:     return S[0 : T]
```

O algoritmo apresenta um tempo de execução linear $O(N)$ em relação ao tamanho da *string* de entrada. Isso pode ser identificado pelo laço externo limitado por $|S|$, enquanto o laço interno não reinicia a cada iteração, mas continua a execução a partir do

último valor de T, incrementando-o progressivamente até atingir $|S|$, visto que a condição $S[i \bmod T] = S[i \bmod N]$ sempre será verdadeira para $T = N$. Dessa forma, o laço externo itera $2N$ vezes, enquanto o laço interno itera em no máximo N vezes.

4. Prova

Para verificar a correção do algoritmo, dividimos em dois casos a serem analisados: **Caso A** onde a *string* de entrada não é um *tandem array*, e **Caso B** no qual a entrada é.

4.1. Caso A: entrada não é *tandem array*

Esse caso pode ser contornado caso forcemos que a entrada seja um *tandem array*, e isso pode ser obtido ao concatenar a *string* de entrada a ela mesma. Na solução proposta, o mesmo resultado é obtido ao multiplicarmos o número de iterações no laço externo em 2 e aplicarmos $\bmod|S|$ quando acessamos o índice da *string* de entrada.

4.2. Caso B: entrada é *tandem array*

Para esse caso, T representa qualquer um dos valores que podemos encontrar para a unidade de repetição U tal que $U = S[0 : T]$ e $U^k = S$. Definindo T_{min} como o tamanho da menor *substring* possível, é inferível que, quando $T = T_{min}$, a condicional do laço interno nunca será verdadeira, então o valor de T nunca será incrementado acima do primeiro T_{min}. Outro ponto que podemos induzir é que, como T cresce a partir de 1, o primeiro T_{min} encontrado será automaticamente a menor unidade de repetição, visto que, se existir outro T_{min} para a entrada, ele terá que ser maior do que o primeiro.

Se fosse possível que o algoritmo encontrasse um valor $T < T_{min}$, significaria que $S[0 : T]$ nunca violou a condição $S[i \bmod T] = S[i \bmod N]$ ao longo da execução após a última ativação do laço interno. Isso implica que alguma *substring* de $S[0 : T]$ é sufixo da *string* de entrada.

Além disso, como T é menor que T_{min}, o segmento $S[0 : T]$ é um prefixo de $S[0 : T_{min}]$. No entanto, para que $S[0 : T]$ pudesse ser um período válido da *string* completa, ele precisaria ser capaz de reconstruir $S[0 : T_{min}]$ repetindo-se integralmente dentro de $S[0 : T_{min}]$. Isso levaria a uma contradição: se $S[0 : T]$ fosse um sufixo de $S[0 : T_{min}]$ e também um período da entrada, então T_{min} não seria a menor unidade de repetição, pois um período menor teria sido encontrado. Como T_{min} já foi definido como o menor período válido, não pode existir um $T < T_{min}$ que satisfaça a condição do laço interno sem ser rejeitado.

5. Avaliação

Foram feitos testes de desempenho para demonstrar a linearidade temporal do algoritmo proposto e compará-lo com a versão ingênua e uma implementação usando o algoritmo KMP [Knuth et al. 1977], o qual também resolve o problema em tempo linear. Os resultados são visualizáveis na Fig. 1. Para a realização dos testes, foi utilizado um notebook Aspire A315, com 8 GB de memória RAM. Foram utilizadas 26 entradas de 100 até 24 mil caracteres, armazenadas em diferentes arquivos, sendo todas *tandem arrays*. Para cada entrada, foi feita a média entre 4 testes. O algoritmo proposto obteve um desempenho similar ao algoritmo usando KMP no geral.

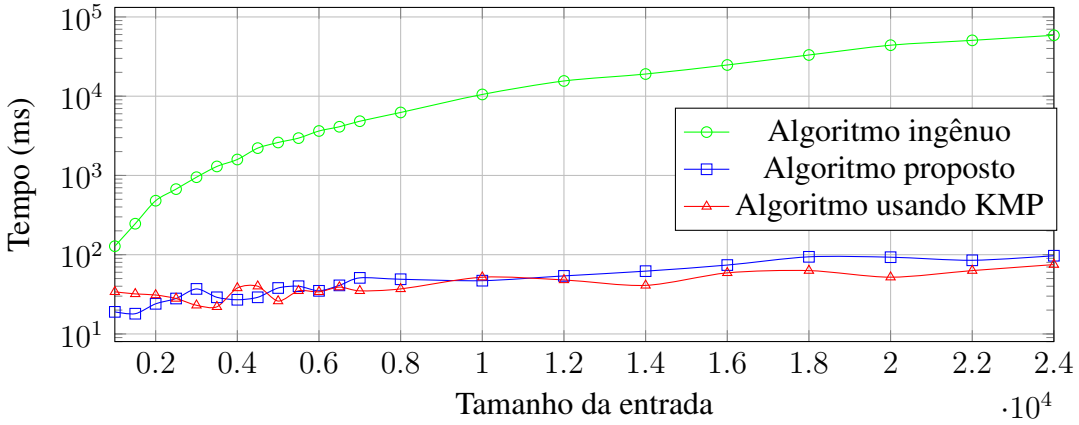

Figura 1. Comparação de tempo de execução

6. Conclusão

Em suma, foi demonstrada a execução do algoritmo para encontrar a menor unidade de repetição em um *tandem array*, garantindo que ele sempre retorne o menor período válido. Embora o algoritmo não traga uma proposta completamente nova, ele oferece uma solução eficiente, similar ao algoritmo KMP [Knuth et al. 1977], que também opera em tempo linear, porém com uma implementação simplificada. Além disso, a abordagem pode ser adaptada para o problema de *Minimum Repetition Representation (MRR) of a Tree* [Nakamura et al. 2013], onde a identificação de padrões repetitivos auxilia na compactação e análise estrutural de árvores.

Referências

Aho, A. V. (1990). Chapter 5 - algorithms for finding patterns in strings. In Van Leeuwen, J., editor, *Algorithms and Complexity*, Handbook of Theoretical Computer Science, pages 255–300. Elsevier, Amsterdam.

C. S. Góes, A. (2005). Análise de regiões polimórficas do DNA com o objetivo de estabelecer vínculos genéticos, identificar restos mortais ou realizar perícias criminais. *Revista do Biomédico*, 65:22–23.

Knuth, D. E., Morris, Jr., J. H., and Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350.

Mala, F. and Ali, R. (2022). The big-O of mathematics and computer science. *Journal of Applied Mathematics and Computation*, 6:1–3.

Nakamura, A., Saito, T., Takigawa, I., Kudo, M., and Mamitsuka, H. (2013). Fast algorithms for finding a minimum repetition representation of strings and trees. *Discrete Applied Mathematics*, 161(10):1556–1575.

Peterson, L. L. and Davie, B. S. (2022). 7 - end-to-end data. In Peterson, L. L. and Davie, B. S., editors, *Computer Networks (Sixth Edition)*, The Morgan Kaufmann Series in Networking, pages 554–605. Morgan Kaufmann, 6th edition edition.

Sokol, D., Benson, G., and Tojeira, J. (2007). Tandem repeats over the edit distance. *Bioinformatics*, 23(2):e30–e35.