Pesquisa Digital: Uma comparação entre os algoritmos Trie e Patricia

Marcos Ramon Paulino Resende[1]

[1]UESPI - Universidade Estadual do Piauí Campus Piripiri
Av. Pres. Castelo Branco, 180 - Petecas, Piripiri - PI, 64260-000

maramramon10@gmail.com

Resumo. *Os algoritmos de pesquisa digital estão relacionados a uma árvore de pesquisa, onde ela é uma estrutura de dados desenvolvida para que a inserção e remoção de dados se torne muito mais eficiente, visando uma finalidade de otimizar o uso da memória, seja ela primária ou secundária.O objetivo da pesquisa em memória primária é buscar e encontrar um ou mais episódio de registros com suas respectivas chaves iguais à chave de pesquisa.Diante disso, o principal objetivo desse trabalho é fazer uma comparação e analise entre os algoritmos de pesquisa digital, Trie e Patricia, assim verificar e apresentar como funciona e a analise assintótica de busca, inserção e remoção de registros produzido por cada um deles.*

1. Introdução

Diante da importância de como recuperar informações a partir de uma parte grande de informações previamente armazenada, sendo que assas informações são divididas em registros, onde cada um desses registros possui uma determinada chave, é necessário usar uma pesquisa em memória primária. Essa pesquisa é dividida em pesquisa sequencial, pesquisa binária, arvores de pesquisa, pesquisa digital e transformação de chave(Hashing) [1,2].

O objetivo da pesquisa em memória primária é buscar e encontrar um ou mais episódio de registros com suas respectivas chaves iguais à chave de pesquisa [3].Nesse sentido, quando a chave é encontrada é notório ver que a pesquisa ocorre com sucesso, caso contrario a pesquisa é sem sucesso dentro de um conjunto de registros que é chamando de arquivo ou tabela [3].

Dentro desse contexto pesquisa, existe uma profusão enorme de métodos de pesquisa. Para saber qual o método adequado a ser usado em uma determinada aplicação depende principalmente do acervo dos dados envolvidos e do arquivo estar sujeito a frequentes inserções e retiradas, ou do conteúdo do arquivo ser aproximadamente fixo [5,6].

Consequentemente, é importante salientar que os algoritmos de pesquisa deve ser considerados como tipo abstrato de dados, com um conjunto de operações associadas a uma estrutura de dados, de tal forma que haja uma independência de implementação para as operações [5,7].

Algumas dessas operações mais comuns incluem principiar a estrutura de dados, buscar um ou mais registros com uma determinada chave, somar um novo registro, subtrair um registro específico, ordenar um arquivo para obter todos os registros em ordem de acordo com a chave e ajuntar dois arquivos para formar um arquivo maior [8,11].

Um nome usualmente utilizado para representar uma estrutura de dados para pesquisa é dicionário. Ele é um tipo abstrato de dados com as operações de inicializa, pesquisa, insere e retira [9]. Tem uma semelhança com o dicionário da língua portuguesa, onde as chaves são as palavras e os registros são as entradas associadas com cada palavra, cada entrada contém a definição, pronúncia, sinônimo e outras informações associadas com a palavra [10].

Diante disso, o principal objetivo desse trabalho é fazer uma comparação e analise entre os algoritmos de pesquisa digital, Trie e Patricia, assim verificar e apresentar como funciona e a complexidade assintótica de busca, inserção e remoção de registros produzido por cada um deles.

Para o esclarecimento desse trabalho a organização estrutural esta dividida da seguinte forma: na seção 2 é apresentado alguns trabalhos relacionados a Trie e Patricia, a seção 3 é comentando sobre Pesquisa Digital e explanado o algoritmo Trie e Patricia, a seção 4 apresenta a comparação entre os algoritmos com os resultados obtidos e a seção 5 mostra as conclusões do presente trabalho.

2. Trabalhos Relacionados

Durante o desenvolvimento deste artigo foram estudados alguns trabalhos que auxiliaram e reforçaram o conhecimento sobre os algoritmos Trie e Patricia. Os trabalhos estudados apresentam modelos variados de algoritmos e mostra a usabilidade de cada um deles.

O trabalho de [BODON et al. 2005], investiga um algoritmo APRIORI baseado em trie para minerar itens frequentes sequências em um banco de dados transacional. Ele examina a estrutura de dados, implementação e recursos algorítmicos com foco principalmente naqueles que também surgem na mineração frequente de conjuntos de itens. A análise leva em consideração as propriedades dos processadores modernos (hierarquias de memória, pré-busca, previsão de ramificação, tamanho da linha de cache, etc.), a fim de compreender melhor os resultados dos experimentos.

Para a sua pesquisa [PIETRACAPRINA et al. 2003], estudou o algoritmo Patricia com otimizações onde apresentam um algoritmo de profundidade, PatriciaMine, que descobre todos os conjuntos de itens frequentes em um conjunto de dados, para um determinado limite de suporte. O algoritmo é baseado na memória principal, ele é eficiente em termos de espaço em um conjuntos de dados.

[SHAFIEI et al. 2013] apresenta em seu artigo uma implementação não bloqueadora de Patricia trie para um sistema assíncrono compartilhado de sistema de memória usando Compare Swap. O trie implementa um conjunto linearizável e suporta três operações de atualização: inserir adiciona um elemento, excluir remove um elemento e substituir substitui um elemento por outro. A operação de substituição é interessante porque muda duas localizações diferentes do trie atomicamente. Se todas as operações de atualização modificarem diferentes partes do trie, eles são executados simultaneamente.

3. Pesquisa Digital

Os algoritmos de pesquisa digital estão relacionados a uma árvore de pesquisa, onde ela é uma estrutura de dados desenvolvida para que a inserção e remoção de dados se torne muito mais eficiente, visando uma finalidade de otimizar o uso da memória, seja ela primária ou secundária.

Adentro de estruturas utilizadas em árvore de pesquisa tem uma delas que é a pequisa digital, onde baseia-se na representação de palavras, conhecidas como chaves, como uma sequencia de dígitos ou caracteres, atuando como um índice de dicionário, em que a primeira palavra são determinada todas as páginas que contêm as palavras iniciadas por aquela letra e assim por diante.

A seguir são abordados os conceitos e analise dos algoritmos Trie e Patricia quanto a complexidade assintótica de busca, inserção e remoção de registro, para que se tenha entendimento do sistema de pesquisa digital desse trabalho.

3.1. Trie

Uma trie, ou árvore de prefixos, é estruturada do tipo árvore ordenada, onde pode ser utilizada para armazenar um array comparativo, logo suas chaves são normalmente cadeias de caracteres. Ela é basicamente usada na recuperação de dados e sua pronuncia é do inglês retrieval (recuperação), é [tri] (”tree”), alguns usam o [tra] (”try”).

Uma TRIE é um tipo de árvore m-ária onde seus nodos são vetores de m componentes, as chaves são formadas por campos correspondentes aos caracteres ou dígitos.

O tamanho das chaves inseridas é quem indica o tamanho de uma Trie, no qual cada nó equivale a uma letra da palavra e, consequentemente, vai gerar novas ramificações. Um exemplo simples esta representado e pode ser observado na Figura 1, pois são utilizados bits para a representação das letras inseridas na árvore.

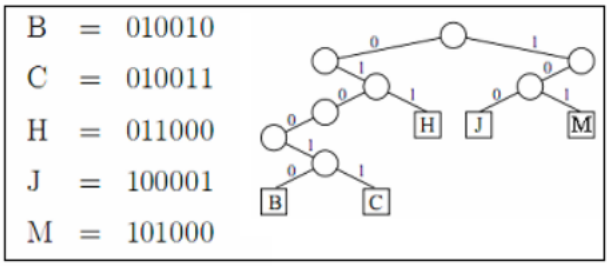

Figure 1. Formato de uma Trie com seus caracteres representados por bits [5]

3.1.1. Busca

A busca de uma determinada chave é feita caractere por caractere. Diante disso a sua complexidade é de O(n) no pior caso,ou seja, ela possui uma complexidade linear.

- Se o primeiro caractere pertence a árvore, verifica o próximo e caso todos os caracteres pertençam em sequencia a Trie a chave pertence a árvore. Representado e pode ser observado na Figura 2

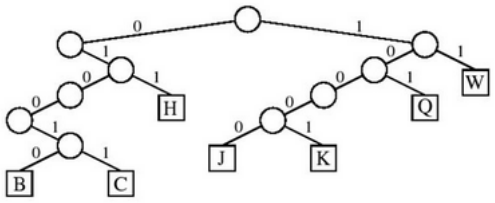

Figure 2. A chave W que é igual a 11 pertence a Trie [5]

- Se o caractere não pertence a árvore a chave não pertence a Trie. Representado e pode ser observado na Figura 3.

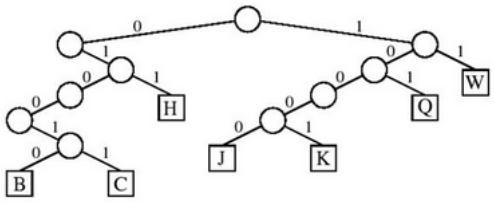

Figure 3. A chave R que é igual a 00 não pertence a Trie [5]

3.1.2. Inserção

A inserção de uma determinada chave faz-se uma busca pela palavra a ser inserida. A sua complexidade é de O(n) no pior caso, também linear.

- Se ela já estiver na Trie nada é feito durante a inserção. Representado e pode ser observado na Figura 4.

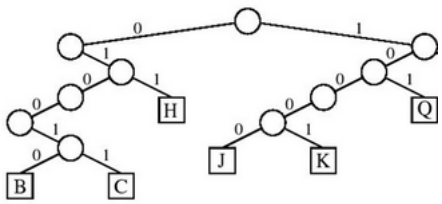

Figure 4. A chave Q que é igual a 101 já esta na Trie [5]

- Se caso contrário, é recuperado o último nó N da maior substring da palavra a ser inserida e assim o complemento dos caracteres da chave são inclusos na Trie a partir do nó N. Representado e pode ser observado na Figura 5.

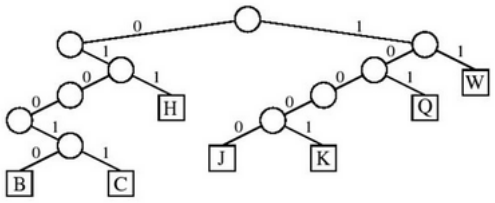

Figure 5. Inserir a chave W que é igual a 11 na Trie [5]

3.1.3. Remoção

Na remoção busca-se a chave a ser removida a partir da folha e são removidos todos os nós que tem apenas um filho e com isso a complexidade é O(n), linear também.

- A busca da chave w que é igual a 11 na Trie. Representado e pode ser observado na Figura 6.

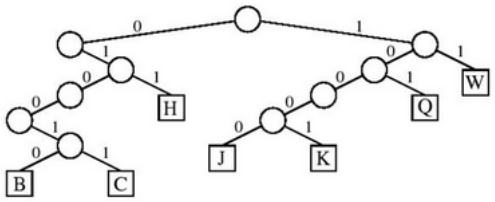

Figure 6. Ainda com a chave na busca [5]

- Chave removida com sucesso. Representado e pode ser observado na Figura 7.

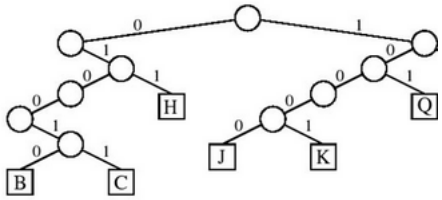

Figure 7. Sem a chave apos a remoção [5]

3.2. Patricia

A Patricia (Practical Algorithm To Retrieve Information Coded In Alphanumeric) é um método de pesquisa digital, tem com o objetivo de melhorar a recuperação de informações em arquivos de um volume grande, no sentido de que cada nó da árvore guarde o índice do próximo bit a ser testado, ou seja, o índice armazenado no nó mostra o próximo bit diferente entre as palavras, para que assim evite uma pesquisa bit a bit. Ao ser inserido

um determinado número nas duas estruturas, o tamanho da dela é mais otimizado do que a Trie. Diate disso, esses fatores contribuem no tempo de pesquisa e também é notório uma economia de memória bastante considerável.

Esse algoritmo é uma representação comprimida de uma Trie, pois os nós que teriam apenas um filho são inseridos nos seus antecessores. Geralmente as árvores Trie possuem um grupo disperso de chaves, desse modo, muitos nós possuem apenas um sucessor. Com tudo isso as Trie obtenham um custo grande de espaço, ela usa cada uma dos lados de uma chave, por vez, para determinar a sub-árvore, em vez disso a árvore Patricia seleciona um elemento da chave para determinar a sub-árvore.

Usando os exemplos vistos anteriores, pode-se observar na Figura 1 como fica uma Patricia. Representada e pode ser observada na Figura 8.

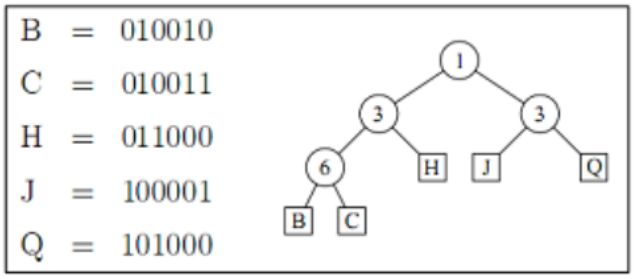

Figure 8. Formato de uma Patricia com seus caracteres representados por bits [5]

3.2.1. Busca

A busca de uma determinada chave é feita da seguinte forma:

- Compara o caractere na posição X com o caractere Y. Se menor ou igual segue o ramo a esquerda, caso contrario segue o ramo a direita, é repetido até chegar numa folha, contendo a complexidade de O(log2 n) no pior caso,ou seja, ela é uma complexidade logarítmica. A demostração pode ser representada e observada na Figura 9

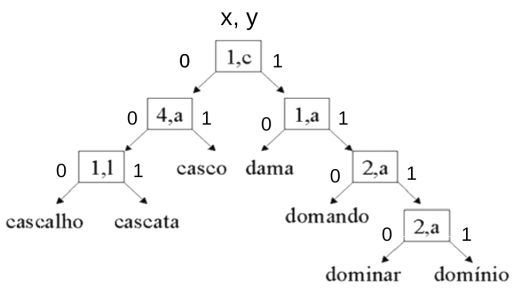

Figure 9. Busca em uma árvore Patricia [12]

3.2.2. Inserção

A inserção de uma determinada chave faz-se uma verificação no caractere de mesma posição, aloca o nó para a nova palavra e atualiza o acumulador, com a complexidade de O(log n) para o pior caso, também é logarítmica.

- Demostração de um exemplo em uma Patricia. Representada e pode ser observada na Figura 10.

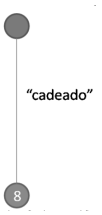

Figure 10. Árvore Patricia [12]

- Inserção de uma determinada chave em uma Patricia. Representada e pode ser observada na Figura 11.

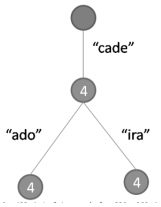

Figure 11. Inserção da chave cadeira [12]

3.2.3. Remoção

Na remoção é o oposto da operação de inserção, busca-se a chave a ser removida e apaga a chave da árvore, como o pai terá apenas um filho a chave dele deve ser apagado e os nós pai e irmão do nó removido são agrupados em um único nó, com a complexidade O(log n), é logarítmica.

- Árvore Patricia antes da remoção. Representada e pode ser observada na Figura 12.

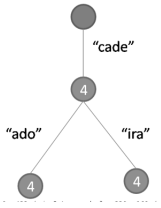

Figure 12. Ainda com a chave na busca para a remoção [12]

- Chave removida com sucesso. Representada e pode ser observada na Figura 13.

Figure 13. Sem a chave apos a remoção [12]

4. Resultados

Os resultados apresentam uma diferença entre o algoritmo de Patricia e o algoritmo Trie não em uma pequena mais em uma grande quantidade de dados. Pois foi verificado no pior caso em relação a busca, inserção e remoção de chaves. Diante das análises dos algoritmos e estudos relacionados o algoritmo de Patricia é mais rápido do que o algoritmo Trie em uma grande quantidade de dados em qualquer uma das operações.

Observando os resultados dos testes feitos para ambos os algoritmos, podemos notar que o algoritmo Trie é linear na suas operações. Já o algoritmo Patricia se torna mais eficiente por conta de ser logarítmico.

5. Conclusões

Diante das análises feitas dos algoritmos Trie e Patricia, observou-se que na busca por uma chave o algoritmo Patricia é mais rápido do que o algoritmo Trie em uma grande quantidade de dados. Na inserção de uma chave, o algoritmo Patricia é mais rápido do que o algoritmo Trie assim como na remoção de uma determinada chave. Consequentemente com o aumento do número de entradas o algoritmo de Patricia sera mais rápido do que o algoritmo Trie.

References

[1] AL-SUWAIYEL, M.; HOROWITZ, Ellis. Algorithms for trie compaction. ACM Transactions on Database Systems (TODS), v. 9, n. 2, p. 243-263, 1984.

[2] AOE, Jun-Ichi; MORIMOTO, Katsushi; SATO, Takashi. An efficient implementation of trie structures. Software: Practice and Experience, v. 22, n. 9, p. 695-721, 1992.

[3] ATHAIDE, Noel et al. Trie partitioning in distributed PC based routers. In: 2009 First International Communication Systems and Networks and Workshops. IEEE, 2009. p. 1-10.

[4] BODON, Ferenc. A trie-based APRIORI implementation for mining frequent item sequences. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. 2005. p. 56-65.

[5] BODON, Ferenc; RÓNYAI, Lajos. Trie: an alternative data structure for data mining algorithms. Mathematical and Computer Modelling, v. 38, n. 7-9, p. 739-751, 2003.

[6] CHATTERJEE, Tanusree; RUJ, Sushmita; DASBIT, Sipra. LowSHeP: Low-overhead forwarding and update Solution in ndn with Hexadecimal Patricia trie. In: 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). IEEE, 2019. p. 1-6.

[7] CROCHEMORE, Maxime; LECROQ, Thierry. Trie. 2009.

[8] HERLIHY, Thomas J. Repetti Maurice P. A Case Study in Optimizing HTM-Enabled Dynamic Data Structures: Patricia Tries.

[9] KNIESBURGES, Sebastian; SCHEIDELER, Christian. Hashed Patricia Trie: Efficient longest prefix matching in peer-to-peer systems. In: International Workshop on Algorithms and Computation. Springer, Berlin, Heidelberg, 2011. p. 170-181.

[10] KNOLLMANN, Till; SCHEIDELER, Christian. A Self-stabilizing Hashed Patricia Trie. In: International Symposium on Stabilizing, Safety, and Security of Distributed Systems. Springer, Cham, 2018. p. 1-15.

[11] NAVARRO, Gonzalo. Indexing text using the Ziv–Lempel trie. Journal of Discrete Algorithms, v. 2, n. 1, p. 87-114, 2004.

[12] PIETRACAPRINA, Andrea. Mining frequent itemsets using patricia tries. 2003.

[13] SHAFIEI, Niloufar. Non-blocking Patricia tries with replace operations. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems. IEEE, 2013. p. 216-225.