Busca de Animais em risco de extinção com o uso da Tabela Hash

Raul Corrêa Carneiro[1]
Email: raulccarneiro00@gmail.com

[1]Instituto de Educação Superior de Brasília (IESB)

Abstract. *This article discusses about the ability to search for individuals in a Hash Table, and a comparison between others Search Algorithms. Futhermore, this paper will also show how to solve collisions, as well as explain the processes involving Insertion and Search in a Hash Table.*

Resumo. *Esse artigo discorre sobre a capacidade de busca de indivíduos em uma Tabela Hash, além de compará-lo com outros Algoritmos de Busca. Nele também será abordado as resoluções de colisões, assim como a explicação dos processos de inserção e busca em uma Tabela Hash.*

1. Introdução

Desde os tempos remotos a relação entre os seres humanos e a natureza tem sido marcada pela constante degradação dos recursos naturais. Nesse cenário, a ação do Homem vem impulsionando a redução de animais em todos os continentes, provocando a extinção de certas espécies **[DIANA 2019]**, como o pássaro Ararinha-azul, o Rato-Candango, dentre outros.

Essa atitude torna-se cada vez mais presente no cenário global atual, com o aumento da desflorestação e da aniquilação de espécies em diversos ecossistemas. Devido à isso, a busca por um animal específico em risco acaba tornando-se cada vez mais difícil.

Com o intuito de estabelecer mecanismos de conservação da biodiversidade, organizações mundiais mantêm catalogados mais de 60.000 espécies de animais em situação de risco de total desaparecimento, relatando informações sobre o tamanho de certa população, sobre o estado dos habitats naturais e sobre ações humanas que ameaçam a extinção dessas espécies **[IUCN 2020]**. Tais registros são utilizados por órgãos governamentais e não-governamentais, a fim de estabelecer mecanismos para conservação da biodiversidade.

Tomando-se como base a complexidade e o número de indivíduos catalogados e a dificuldade em descobrir se uma determinada espécie da fauna se encontra em risco de extinção, o presente estudo utilizará a Tabela Hash. Tal ferramenta proporcionará a busca dentro de um banco de dados contendo animais em risco de extinção e informações sobre a situação atual.

2. Objetivo Geral

Realizar uma busca de uma determinada espécie de animal, através de um banco de dados, utilizando a Tabela Hash a fim de obter informações sobre sua situação atual de extinção.

3. Objetivo Específico

Além do objetivo principal do projeto, que é realizar uma busca em uma Tabela Hash, é de grande importância citar alguns objetivos que contribuem com o desenvolvimento do trabalho. Pode-se destacar:

A explicação dos conceitos e das aplicações da Tabela Hash;

As formas de como solucionar as colisões em uma Tabela Hash;

A apresentação dos dados coletados dos testes feitos na Tabela Hash;

A comparação com a Busca Linear, a Busca Binária e a Árvore Binária de Busca, mostrando a eficiência da Tabela Hash em relação a outros tipos de Algoritmos de Busca.

4. Referencial Teórico

4.1. Tabela Hash

Um banco de dados comum contém uma enorme lista de registros sobre determinado assunto. Normalmente, quando alguém deseja fazer uma busca sobre um registro específico dentro desse banco, o seu tempo de execução acaba sendo maior do que o esperado, devido à posição que o elemento sendo busca encontra-se no momento. Com isso, é necessário fazer uso de outros tipos de algoritmos de busca para otimizar o tempo de busca nesse mesmo banco.

A Hash Table **[CORMEN 2012]**, também conhecida como Tabela de Espalhamento, trata-se de uma estrutura de dados especial que armazena seu conteúdo dentro de um vetor, porém, cada indivíduo desse vetor apresenta um índice único, nomeada de chave, na qual é usada como uma espécie de identificação do registro.

O nome ”Hash Table” vem da função Hash, que consiste em uma técnica na qual é feita a distribuição de indivíduos em um vetor a partir de uma chave especial, o que leva a uma redução no tempo de busca de um determinado indivíduo. Normalmente, essa função é muito utilizada em bancos de dados que apresentam uma quantidade elevada de dados a serem armazenados.

A definição dessa chave especial é feita da seguinte maneira: suponha-se que será armazenado em uma Hash Table, de tamanho 15, números de 1 a 100. Para definir uma chave especial, é necessário fazer a divisão entre o valor cadastrado e o tamanho definido da tabela. Ao ser feita a divisão, o resto que sobrou da operação é definida como o índice, ou seja, a chave especial do registro, determinando a posição de cada dado no vetor. Essa operação é sempre feita quando ocorre uma inserção nova dentro de uma Hash Table. Um exemplo desse processo é mostrado na imagem a seguir:

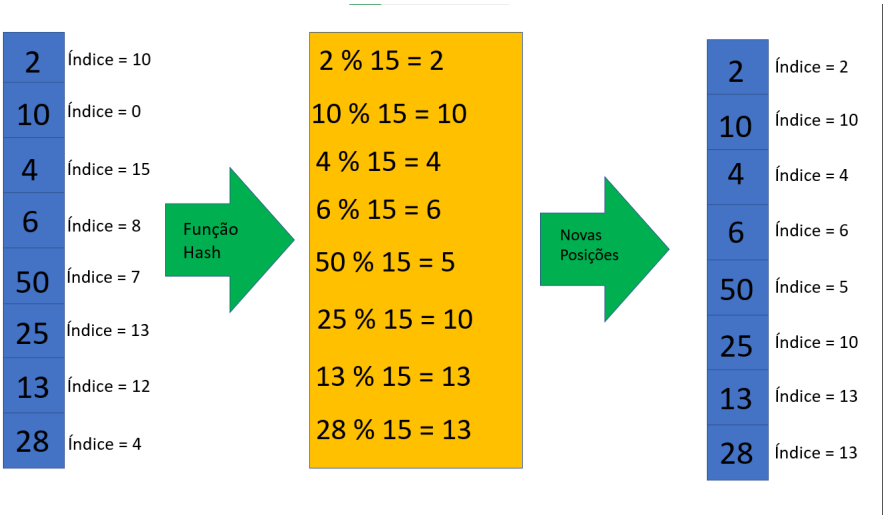

Figure 1. Função Hash em ação

Como explicado anteriormente, esse exemplo mostra como a função Hash funciona na prática, definindo novos índices para cada indivíduo a partir dos restos de cada divisão feita. No entanto, como mostrado na imagem acima, alguns indivíduos receberam índices iguais. Essa situação é nomeada de Colisão.

4.1.1. Colisões

Colisão trata-se de situações em que o cálculo feito em indivíduos diferentes geram resultados similares, ou seja, índices iguais. Essa situação pode ser resolvida em duas maneiras: usando o Endereçamento Aberto ou o Encadeamento Separado.

Normalmente usado em vetores com poucos registros, o Endereçamento Aberto apresenta duas formas de solucionar colisões: a Sondagem Linear e a Sondagem Quadrática. A Sondagem Linear funciona da seguinte maneira: o indivíduo que apresentar o mesmo índice de outro já registrado passa pela seguinte equação: **(h(X) + N)**, sendo o h(X) a Função Hash do número X e N o número a ser somado com o resultado da função.

Nessa equação, o índice resultante da Função Hash é somado ao número N, que inicialmente é representado pelo número 1, resultando em um novo índice na qual o registro será armazenado no vetor. Caso a posição representada por esse índice não estiver vazia, a equação é chamada novamente, com a diferença de que o valor N é somado por 1. Esse processo se repete até que uma posição vazia seja encontrada. Esse processo é mostrado no exemplo abaixo:

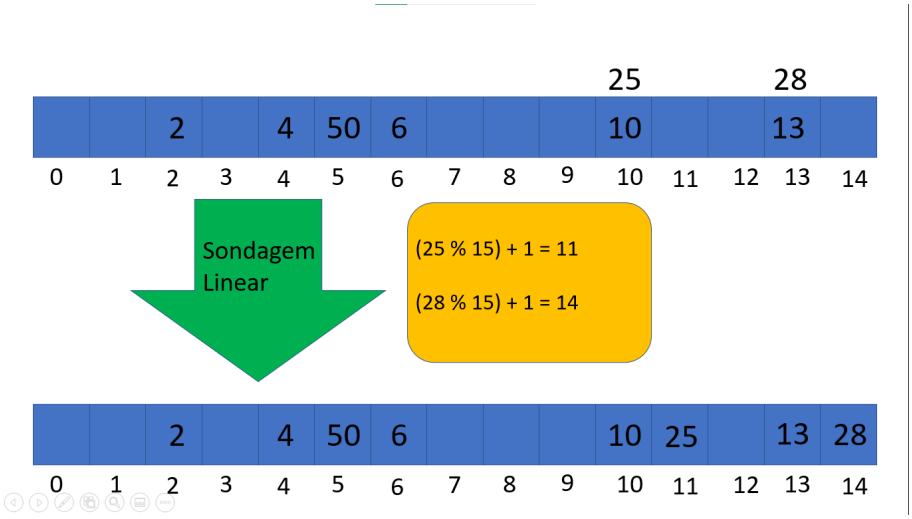

Figure 2. Colisão resolvida com Sondagem Linear

Como mostrado na imagem acima, dois indivíduos estavam em posições já ocupadas por dois registros. Devido à isso, seguindo a lógica da Sondagem Linear, a equação é chamada para os dois indivíduos, resultando na definição dos novos índices 11 e 14. Com isso, os indivíduos são inseridos nas posições de índice 11 e 14, respectivamente.

A Sondagem Quadrática apresenta outra forma de inserção, sendo representada pela seguinte fórmula: **h(X + N elevado ao 2)**, sendo X o número a ser inserido, N o número a ser somado com X e h() a representação da Função Hash . Nessa fórmula, o valor a ser inserido é somado a um outro número, inicialmente representado pelo 1, elevado ao quadrado. Com isso, o valor resultante dessa soma passa pela Função Hash, definindo o novo índice em que será feito a inserção do indivíduo. Caso a posição representado pelo índice estiver preenchida, o indivíduo passa pela mesma equação, com a única diferença que o valor N é somado por 1, resultando no novo índice. Esse processo se repete até que uma posição vazia seja encontrada. Esse processo é mostrado no exemplo abaixo:

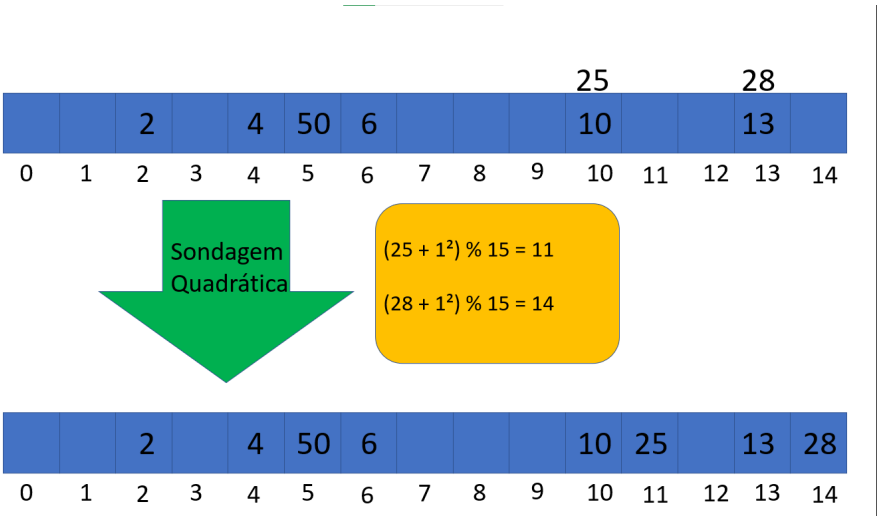

Figure 3. Colisão resolvida com Sondagem Quadrática

No exemplo acima, dois indivíduos estão em posições já ocupadas por outros dois registros. Com isso, usando o método da Sondagem Quadrática, os registros a serem inseridos, 25 e 28, passam pela equação, resultando nos novos índices 11 e 14. Após esse cálculo, os registros são armazenados nas posições localizadas nos índices resultantes, solucionando a colisão. No entanto, em vetores com muitos registros e pouco espaço, é necessário usar o outro método apresentado.

O Encadeamento Separado trata-se de um método usado em vetores com muitos registros, apresentando uma forma diferente de inserção. Ela funciona da seguinte maneira: quando ocorre uma inserção, os indivíduos são inseridos dentro de uma estrutura de dados suplementar, como a lista ligada ou a duplamente ligada, apontada pelo vetor. Em situações de colisão, o novo indivíduo é inserido no próximo espaço da lista, resultando em um novo elemento na mesma. Esse processo é mostrado no exemplo a seguir:

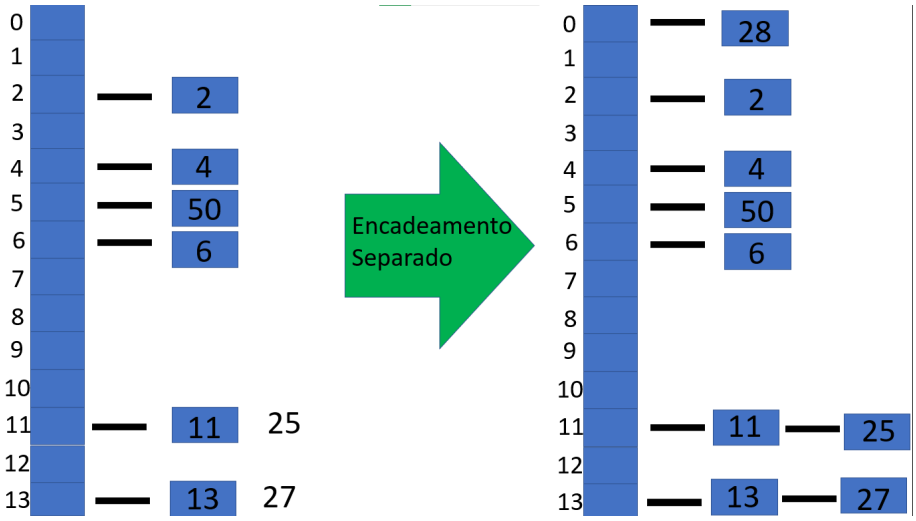

Figure 4. Colisão resolvida com Encadeamento Separado

Na imagem acima, assim como mostrado na Figura 2, dois indivíduos encontram-se em posições já ocupados por outros registros. No entanto, ao invés de procurarem por uma nova posição, eles são inseridos nos registros seguintes das listas ligadas nas posições 11 e 13, respectivamente. Normalmente, muitos optam pelo uso de uma Lista Duplamente Ligada, pela facilidade de locomoção em situações de busca e inserção.

Nos exemplos anteriores, os vetores apresentavam números como indivíduos, porém, quando se trata de strings, é necessário cumprir alguns passos a mais.

4.1.2. Função Hash com Strings

Uma String **[SCHILDT, 1997]** consiste em uma sequência de caracteres, normalmente usada para representar uma palavra ou frases em um programa. Cada carácter é representando por um código binário próprio, na qual é feita uma conversão, a partir da codificação ASCII, para transformar esse código em letras ou símbolos, possibilitando a leitura humana.

Nesse caso, a criação de uma Tabela Hash formada por Strings funcionará da seguinte maneira: para fazer o cálculo da função Hash, é necessário converter o código binário de cada carácter de uma string em um número de base decimal, possibilitando as operações de divisão futuramente. Logo após, é feita uma soma entre os números convertidos de cada string, e com isso, é feita uma divisão entre o resultado dessa soma e o tamanho do vetor, na qual o resto resultante será o novo índice desse indivíduo. Esse processo é mostrado no exemplo a seguir:

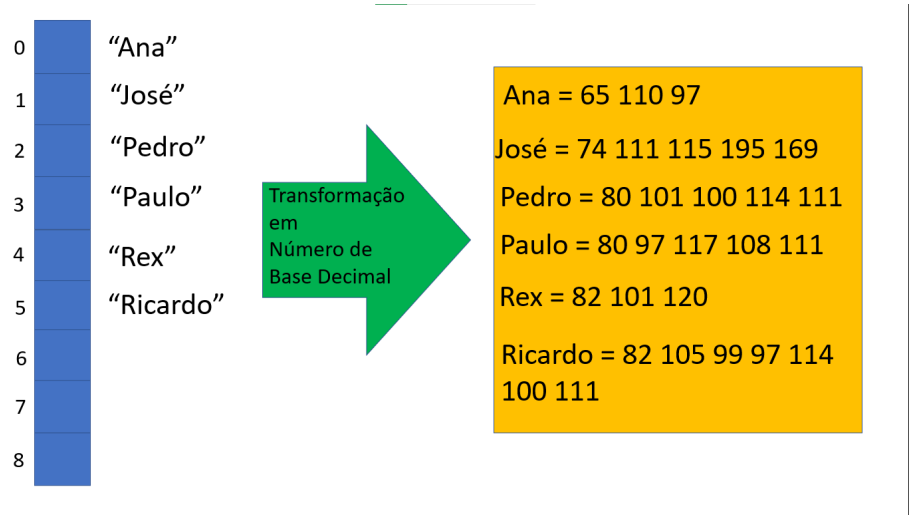

Figure 5. Transformação de Strings para números

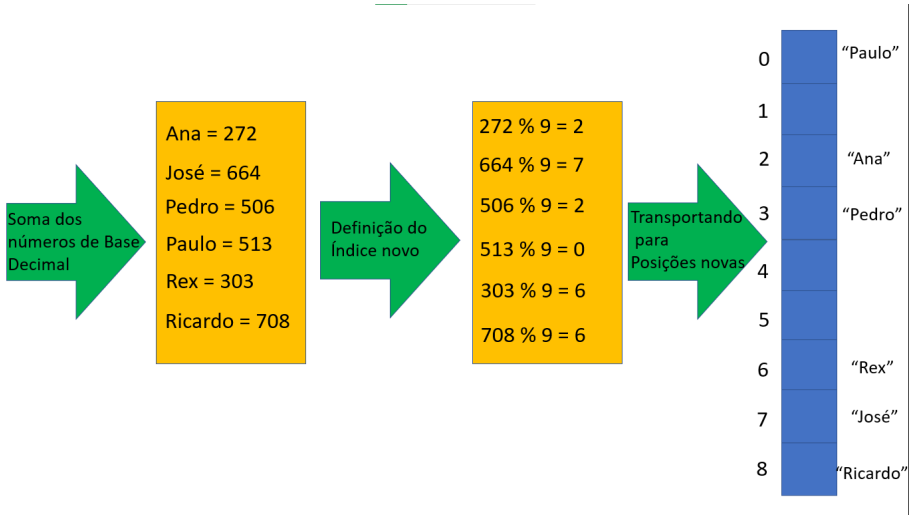

Figure 6. Função Hash e Inserção de Strings

Como mostrado no exemplo acima, as strings presentes no vetor são convertidas em números de base decimal, a partir do ASCII Code de cada uma. Após a conversão, é feita a soma desses números, na qual o resultado é usado para determinar o índice novo de cada elemento, a partir da divisão entre o número resultante e o tamanho do vetor. Com isso, os elementos são transferidos paras as novas posições do vetor, ao mesmo tempo resolvendo qualquer problema de colisão encontrado, finalizando a função Hash.

Em situações com colisões, os registros em String passam pelas mesmas condições de solução de colisões, no entanto, primeiramente é necessário fazer a sua conversão como mostrado anteriormente.

4.1.3. Busca na Tabela Hash

Com a solução das colisões, a busca na Tabela Hash torna-se mais fácil de ser executada. Esse processo funciona da seguinte maneira: ao ser decidido o registro a ser buscado, o registro passa pelo processo da Função Hash, sendo definido a posição em que esse registro se encontra na tabela. Após isso, a busca é iniciada.

No caso de uma Tabela Hash de Endereçamento Aberto, especificamente a de Sondagem Linear, a busca é iniciada a partir da posição definida pela Função Hash, na qual é feita uma comparação entre o registro buscado e o registro armazenado na posição atual, com o intuito de conferir se os registros são os mesmos. Caso os registros sejam diferentes, o indivíduo buscado passa pela equação da Sondagem Linear, continuando a busca até que o registro buscado seja encontrado. Esse processo é mostrado no exemplo a seguir:

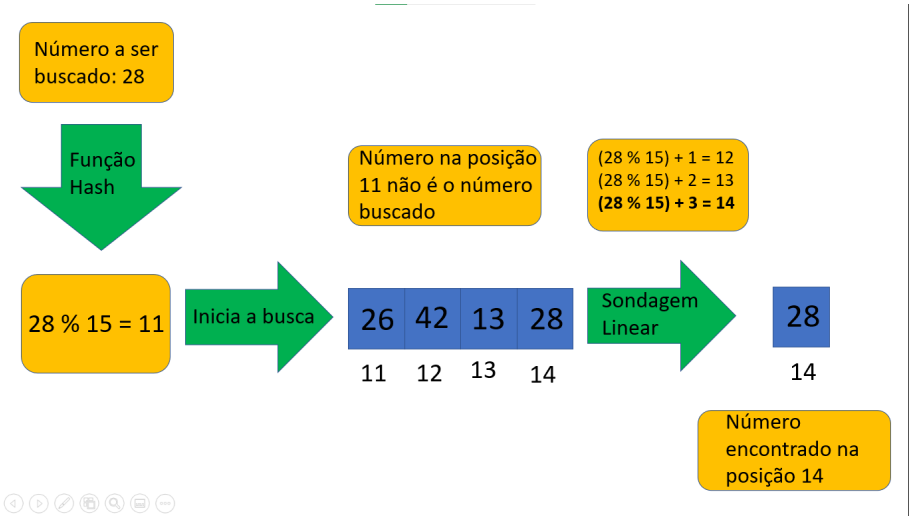

Figure 7. Busca em Tabela Hash de Sondagem Linear

No exemplo acima, o número sendo buscado passa pelo processo da Função Hash, sendo definido o índice da posição em que esse indivíduo estaria armazenado no vetor. Com isso, a busca é iniciada nesse mesmo local. Nesse caso, como o indivíduo passou pelo processo da Sondagem Linear, a busca continua nas posições resultantes da equação até que o alvo seja encontrado. Por fim, o número buscado é encontrado na posição 14, finalizando a função de busca.

Na Sondagem Quadrática, inicialmente, a busca ocorre da mesma forma que na Sondagem Linear, porém, caso o registro buscado não for o mesmo na posição atual, ele passa pela equação da Sondagem Quadrática, resultando na próxima posição a ser analisada na busca. Com isso, esse processo se repete até que o indivíduo buscado seja

encontrado. Esse processo se repete até que o indivíduo buscado seja encontrado. O exemplo a seguir apresenta como a busca funciona:

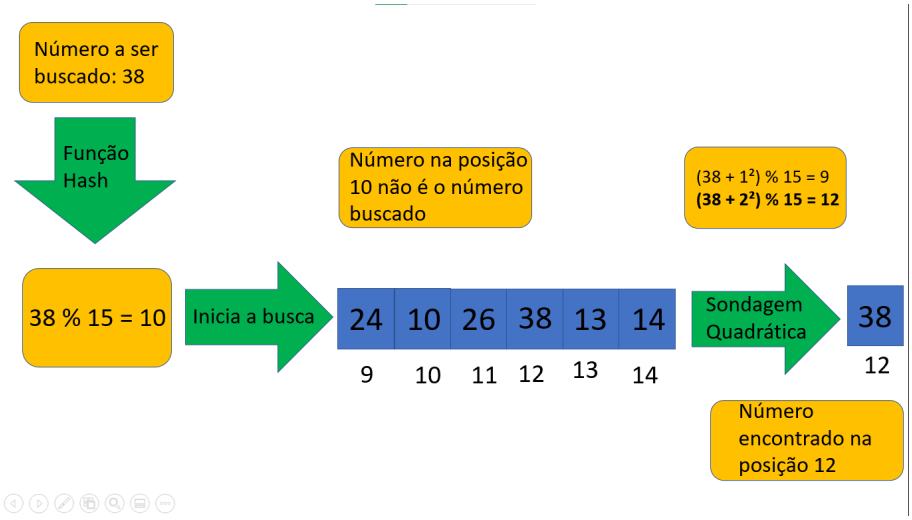

Figure 8. Busca em Tabela Hash de Sondagem Quadrática

No exemplo acima, o número buscado passa pelo processo da Função Hash, definindo o índice da posição em que estaria armazenado no vetor, iniciando a busca. No entanto, como o número nessa posição é diferente do indivíduo buscado, ele passa pelo processo da Sondagem Quadrática, continuando a busca nas novas posições até que o alvo seja encontrado. No final, o número buscado é encontrado na posição 12, finalizando a função.

No caso da Tabela Hash de Encadeamento Separado, a busca funciona da seguinte maneira: com o índice de busca definido, o registro buscado é comparado com cada registro das Listas Ligadas até que seja encontrado. Esse processo é mostrado no exemplo a seguir:

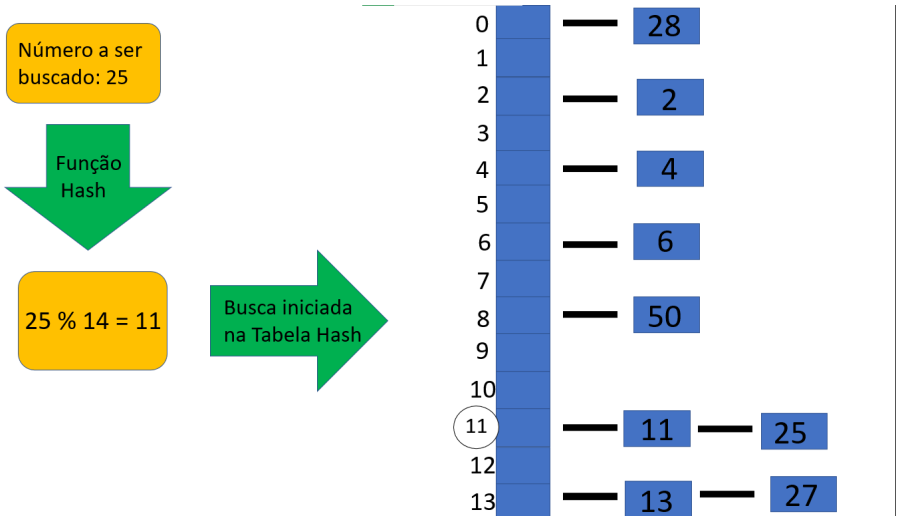

Figure 9. Busca em Tabela Hash de Encadeamento Separado (Parte 1)

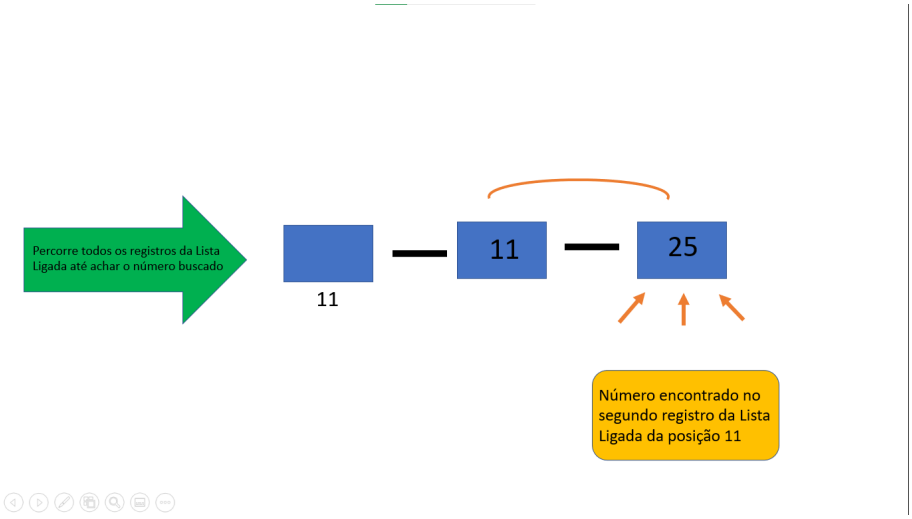

Figure 10. Busca em Tabela Hash de Encadeamento Separado (Parte 2)

No exemplo acima, o número buscado passa pela Função Hash, definindo o seu índice de busca, que nesse caso é o índice 11. Após isso, a posição de índice 11 da tabela é localizada, iniciando a busca dentro da mesma. Com isso, as comparações entre o registro atual e o registro buscado são feitas até que a mesma seja encontrado, finalizando a busca. Nesse caso, o indivíduo buscado é encontrado na segundo registro da Lista Ligada da posição de índice 11.

Os três algoritmos de busca apresentam a mesma complexidade: **O(1)** nos melhores casos e **O(N)** nos piores casos. No melhor caso, o indivíduo buscado será o primeiro registro encontrado inicialmente, enquanto no pior caso, ele será um dos últimos a ser encontrado, sendo o N a representação do números de registros inseridos na Lista Ligada da posição da Tabela Hash.

4.2. Algoritmos de Busca

Para mostrar o verdadeiro potencial da Tabela Hash, é necessário fazer comparações com alguns algoritmos especializados em busca, sendo eles a Busca Linear, a Busca Binária e a Árvore Binária de Busca. Esses três algoritmos apresentam características distintas, nas quais serão exploradas nos próximos tópicos.

4.2.1. Busca Linear

A Busca Linear **[SCHILDT, 1997]** consiste no método de busca mais simples entre as três, na qual a operação é considerada exaustiva, ou seja, a busca por um elemento ocorre em cada posição do vetor até que seja encontrado o indivíduo alvo. Esse processo é mostrado no exemplo a seguir:

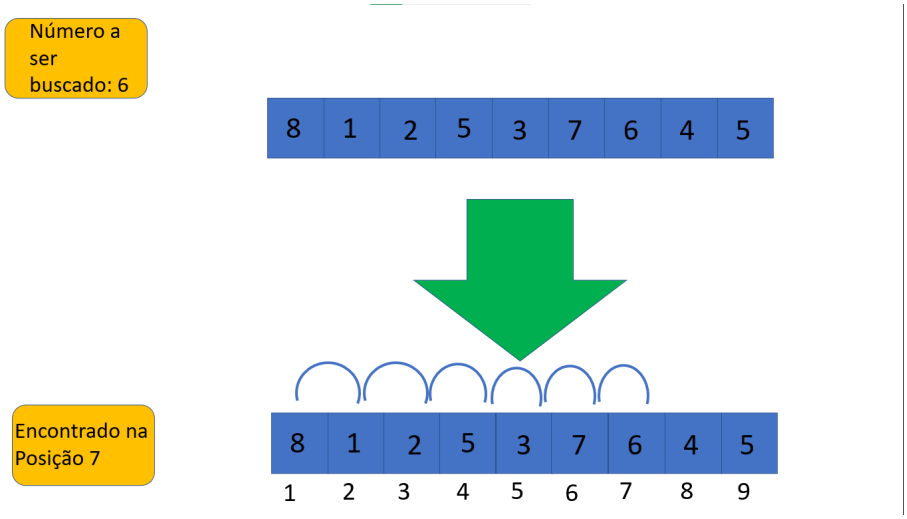

Figure 11. Aplicação da Busca Linear

Na imagem anterior, é feito uma Busca Linear, dentro de um vetor de tamanho 9, pelo número 6. Nesse caso, em cada posição desse vetor, é feito uma comparação entre o número buscado e o número presente na casa atual. Caso o elemento não seja o mesmo do número buscado, esse processo continua nas próximas posições até que seja encontrado o índice do elemento alvo.

Devido à sua simplicidade, a Busca Linear é utilizada frequentemente em casos que apresentam vetores de tamanho reduzido. No melhor caso, se o número buscado for o primeiro elemento do vetor, o algoritmo apresenta uma complexidade de **O(1)**. No entanto, no pior caso, o tempo de execução desse Algoritmo de Busca aumenta cada vez mais, apresentando uma complexidade de **O(N)**, sendo N o número de elementos no vetor. Devido à isso, o seu uso em torna-se menos efetivo em vetores com muitos registros.

4.2.2. Busca Binária

A Busca Binária **[SCHILDT, 1997]** consiste em um método de busca utilizado em vetores ordenados, na qual a busca é limitada a um local específico do vetor a partir da definição de um pivô.

Esse processo funciona da seguinte maneira: inicialmente, um cálculo é feita para definir um pivô, elemento que vai dividir o vetor em dois meios. Esse cálculo consiste na soma entre o índice inicial e o índice final de um vetor, ocorrendo uma divisão pela metade em seguida. Com isso, o índice resultante é escolhido como pivô.

Após essa definição, é feita uma comparação entre o número sendo buscado e o pivô. Caso o número seja maior que o elemento presente na posição do pivô, a busca é iniciada nas posições a frente do pivô até o fim do vetor. E caso o número seja menor que o pivô, a busca é iniciada no começo do vetor até a posição anterior a do pivô. Esse processo é mostrado com mais detalhes nas imagens a seguir:

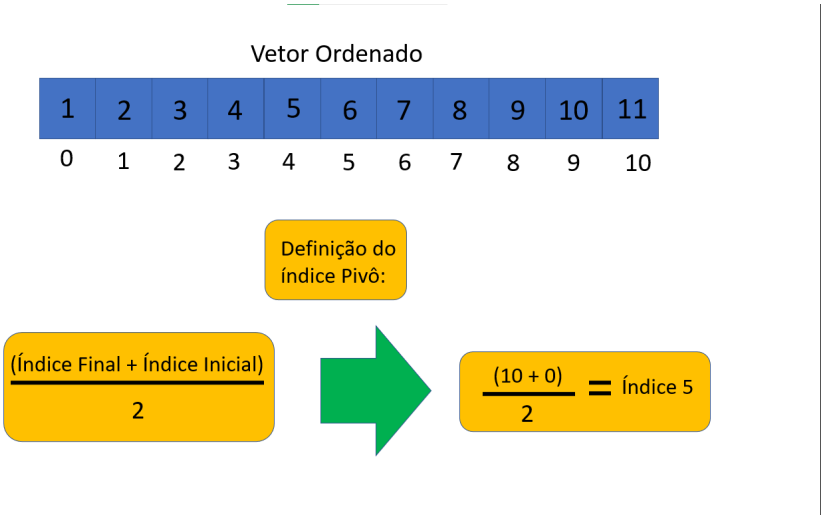

Figure 12. Definição do Pivô

Como mostrado na figura 12, o pivô é definido a partir da soma entre o índice 10 e o índice 0, que representam a posição inicial e final do vetor, logo ocorrendo a divisão, resultando na definição do pivô.

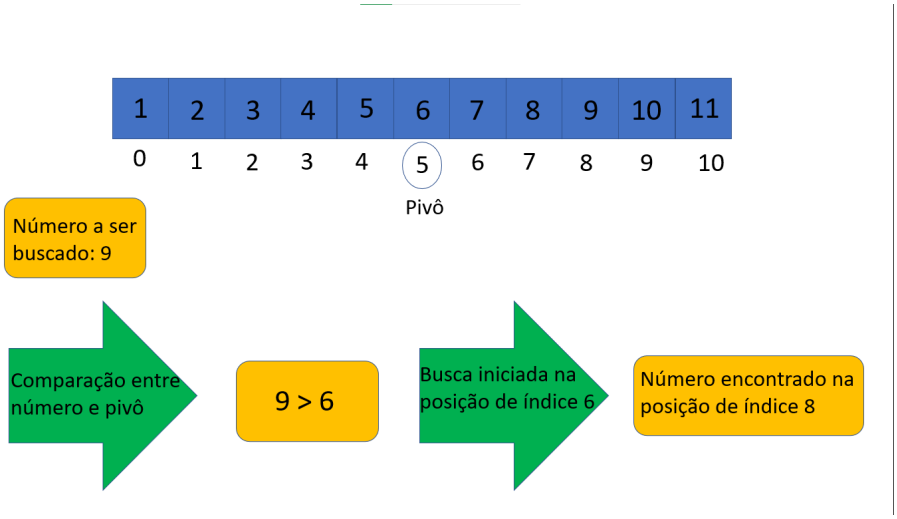

Figure 13. Aplicação da Busca Binária

Com isso, é feito uma comparação entre o número a ser buscado e o elemento presente na posição do pivô. Nesse caso, o número é superior ao do pivô, iniciando a busca a partir da primeira posição a frente do pivô, que por fim, encontra o mesmo na posição de índice 8, finalizando o processo.

Devido à sua funcionalidade em buscar elementos a partir da comparação entre os mesmos, a Busca Binária apenas pode ser usada em situações em que o vetor encontra-se ordenado, como explicado anteriormente.

No melhor caso, esse algoritmo apresenta uma complexidade de **O(1)**, ou seja, quando o elemento buscado estiver no meio do vetor. Já no pior caso, o algoritmo ap-

resenta uma complexidade de **O(log N)**, na qual existe a possibilidade que o indivíduo buscado não esteja presente no vetor.

4.2.3. Árvore Binária de Busca

A Árvore Binária de Busca **[CORMEN, 2012]** consiste em um algoritmo de busca utilizado em Árvores Binárias.

A Árvore Binária trata-se de um tipo de estrutura de dados ligada, na qual apresenta ”Nós” como objetos da mesma. Cada nó dessa árvore apresenta dois ponteiros, uma que aponta para o nó à esquerda e outra para o nó à direita. Esses nós são conhecidos como ”Nós Filhos”, enquanto o nó que aponta para eles é conhecido como ”Nó Pai”.

A definição das posições desses nós são feitas da seguinte maneira: ao ser feita a inserção de um novo registro, é feita uma comparação entre ele e o registro presente no nó pai. Caso o novo registro apresentar um valor maior que a do nó pai, ele é armazenado no nó apontado para a direita, e caso seja menor, ele é armazenado no nó apontado para a esquerda. Esse é processo é mostrado com mais detalhes na imagem a seguir:

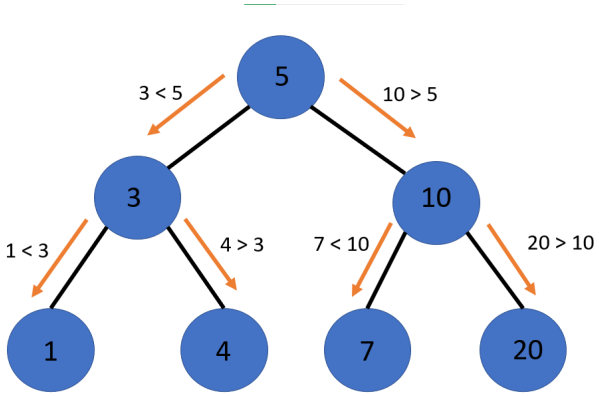

Figure 14. Inserção em uma Árvore Binária

Como mostrado na imagem acima, o processo de inserção é iniciada no nó raiz, representado pelo número 5. Nesse caso, ao ser feita a inserção, uma comparação entre o número a ser inserido e o nó pai ocorre, definindo se o número vai ser apontado para o nó esquerdo ou direito. Esse processo ocorre todas as vezes em que é feita uma inserção dentro da árvore. Devido à sua funcionalidade, a árvore binária é considerada uma estrutura de dados ordenada.

Dependendo da forma como é feita a inserção dos valores nos nós, essa estrutura pode ser considerada Balanceada ou não-balanceada. No caso da balanceada, a altura entre os nós permanece a mesma, enquanto na não-balanceada há uma diferença na altura, apresentando um desequilíbrio no número de nós em cada lado da árvore.

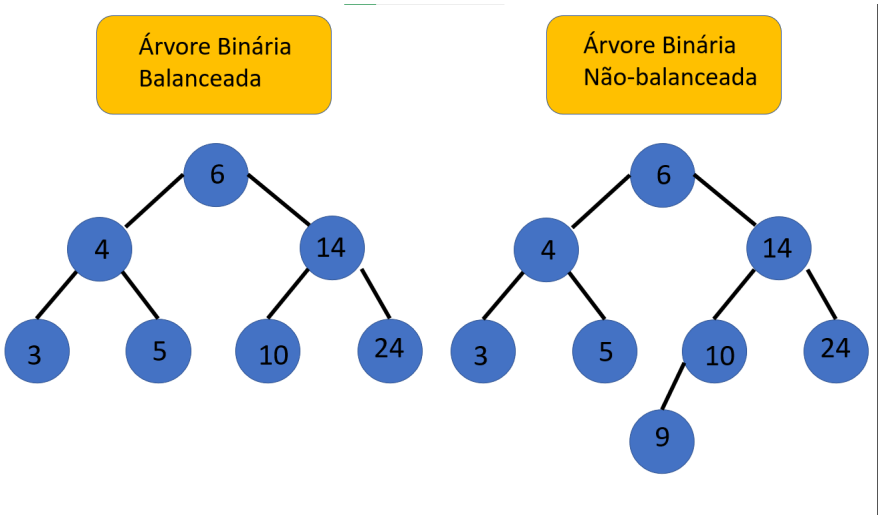

Figure 15. Exemplos de Árvore Binária Balanceada e não-balanceada

A Busca em uma Árvore Binária ocorre dessa forma: começando a partir do nó raiz, a busca é feita baseada na comparação entre o conteúdo de cada nó. Com isso, caso o número sendo buscado for menor que o nó, a busca segue pelo nó apontado a esquerda, na qual será feito a mesma comparação ao longo do caminho. Esse processo continua até que um nó seja igual ao número buscado, ou até que a estrutura não apresente mais registros. Esse processo é mostrado com mais detalhes na imagem a seguir:

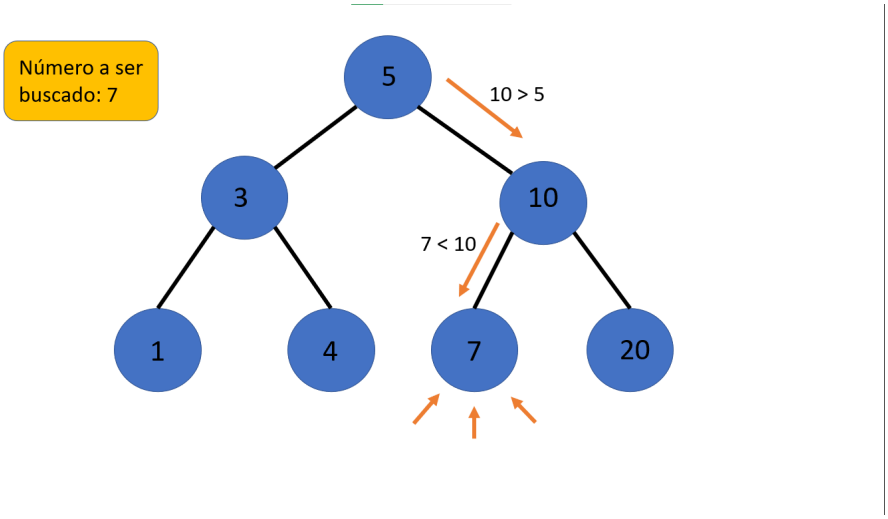

Figure 16. Busca em uma Árvore Binária

Como mostrado na imagem acima, a busca é iniciada a partir do nó raiz, na qual são feitas diversas comparações entre o número buscado e os números presentes nos nós até que o indivíduo buscado seja encontrado. Nesse caso, o número 7 é encontrado no nó filho a esquerda após a comparação entre o nó raiz e o nó pai, respectivamente.

Em um Árvore Binária não-balanceada, o algoritmo de busca apresenta uma complexidade de **O(H)** no pior caso, sendo o H a representação da altura da árvore. No melhor

caso, o algoritmo apresenta uma complexidade de **O(1)**, quando o indivíduo buscado é encontrado na raiz da árvore. Já na Árvore Binária balanceada, o algoritmo apresenta uma complexidade de **O(log N)** no pior caso, enquanto no melhor caso a busca é **O(1)**.

4.3. Nomenclatura do IUCN Red List

No banco de dados distribuído pela organização IUCN, todos os indivíduos registrados são classificados em determinadas menções, nas quais determinam a situação de risco em que cada animal se encontra no momento.

De acordo com a organização **[IUCN,2020]**, os animais podem ser classificados em 11 categorias, cada uma representando um certo grau de risco de extinção. Essas categorias são: Extinct (EX), Extinct in the Wild (EW), Regionally Extinct (RE), Critically Endangered (CR), Endangered (ER), Vulnerable (VN), Near Threatened (NT), Least Concern (LC), Data Deficient (DD), Not Applicable (NA) e Not Evaluated (NE).

Extinct: é a classificação usada para animais que se encontram desaparecidos totalmente no mundo.

Extinct in the Wild: é a classificação usada para espécies que se encontram extintos na natureza, porém, em estado de conservação em cativeiros.

Regionally Extinct: é a classificação usada para espécies que se encontram extintos em uma certa região, porém, ainda podem ser encontrados em um outro local.

Critically Endangered: é a classificação usada para espécies que se encontram em uma situação crítica de extinção.

Endangered: é a classificação usada para espécies que se encontram em risco de extinção.

Vulnerable: é a classificação usada para espécies que podem entrar em risco de extinção, causada pela destruição do habitat ou pelo aumento da caça da mesma.

Near Threatened: é a classificação usada para espécies que não se enquadram nas classificações acima, porém, estão próximos de participar de algumas delas.

Least Concern: é a classificação usada para espécies que não se encontram em risco de extinção em um futuro próximo, não sendo um alvo de prioridade para conservação no momento.

Data Deficient: é a classificação usada para espécies que não apresentam dados suficientes sobre o seu coletivo.

Not Applicable: é a classificação usada para espécies que não se aplicam em uma lista por algum motivo.

Not Evaluated: é a classificação usada para espécies que não passaram por uma avaliação.

5. Desenvolvimento

Para o desenvolvimento da Tabela Hash e dos outros Algoritmos de Busca, no presente estudo, foi utilizada a Linguagem C para a implementação dos mesmos. A sua escolha foi decidida devido a familiaridade e ao conhecimento amplo em relação a essa linguagem.

Ao longo do desenvolvimento do código, foram feitas as seguintes etapas:

A criação de um vetor de estrutura que armazena os registros originados do Banco de Dados utilizado durante o projeto;

A elaboração da estrutura da Tabela Hash e de suas funções;

A criação das funções de Busca Linear e Busca Binária;

E por fim, a criação da estrutura de uma Árvore Binária, assim como suas funções de Inserção e Busca.

Na primeira etapa, um vetor é criado para armazenar os registros do Banco de Dados, especificamente o código de identificação do indivíduo, o nome completo da espécie e a sua situação de risco atual. Com isso, em cada posição do vetor, uma estrutura formada por essas três categorias recebe cada dado, preenchendo o vetor até o último indivíduo.

Em seguida, é iniciada a criação da estrutura da Tabela Hash. Nesse caso, sua estrutura consiste em um vetor de Listas Ligadas, ou seja, em cada posição desse vetor, uma Lista Ligada é inserida nela. Essa estrutura de dados é utilizada para resolver problemas de colisões futuros, como explicado anteriormente no documento.

A função de inserção da Tabela Hash recebe o vetor de registros como parâmetro, na qual é feita o processo da função Hash, definindo a posição de cada registro dentro da Tabela Hash, ocorrendo a inserção logo após. Já na função de busca, o nome buscado passa pelas etapas da função Hash, definindo a posição na qual será feito a busca. Com isso, o nome buscado é comparado com todos os registros presentes naquela posição até que seja encontrado.

Na terceira etapa, as funções de Busca Linear e Busca Binária recebem o vetor de registros e o nome buscado como parâmetro, na qual são iniciadas as buscas pelo nome específico.

Já na última etapa, a elaboração da estrutura de uma Árvore Binária é iniciada. Nesse caso, ela apresentará em cada nó uma estrutura, que possui os mesmos registros dos outros casos e outro que representa o valor convertido do nome da espécie. Com isso, esse valor é utilizado para que seja feita a inserção dos registros nos nós, seguindo o processo da Árvore Binária.

Logo após, é feita a criação da função de busca, na qual será feito a comparação do indivíduo buscado com cada registro nos nós, até que seja encontrado.

5.1. Problemas e soluções

Ao longo do desenvolvimento do código, alguns problemas foram surgindo, dificultando os processos de busca na Tabela Hash e nos Algoritmos de Busca.

Um dos problemas encontrados foi na função de inserção na Tabela Hash. Nessa situação, todos os registros do Banco de Dados estavam sendo armazenados em cada posição do vetor, não respeitando os critérios de inserção impostos pela função Hash.

A resolução desse problema foi bem simples. A única mudança a se fazer era a forma como o a tabela estava recebendo. Nesse caso, ao invés de receber o resultado da função, a tabela era inserida como um dos parâmetros da mesma, o que solucionou o problema da inserção.

Outro problema encontrado foi durante o processo de inserção na Árvore Binária. Nesse caso, ao fazer a conferência dos registros inseridos na árvore, pode-se notar que nem todos eles tinham sido inseridos.

Para resolver esse problema, foi necessário adicionar mais uma condição durante a inserção, na qual caso o valor convertido de um nome for o mesmo de outro indivíduo presente no nó pai, esse novo registro seria inserido no nó filho da direita, continuando o processo.

6. Resultados dos testes

No presente trabalho, foram feitos diversos testes, em situações de diferentes tamanhos utilizados na confecção da Tabela Hash, com o intuito de mostrar uma comparação entre os dados em cada situação. Os tamanhos testados foram: tamanho de 13 posições, tamanho de 26 posições e tamanho de 52 posições, respectivamente.

Em cada caso, foram feitas 15 buscas, utilizando os mesmos registros buscados em cada teste. Para esses testes, foi utilizado um banco de dados com 743 registros.

Os dados coletados consistem no tempo de execução de cada Algoritmo de Busca presente no projeto: Busca na Tabela Hash, Busca Linear, Busca Binária e Busca na Árvore Binária não-balanceada. Para complementar a coleta de dados da Árvore Binária, também foi utilizada a Árvore Binária Balanceada.

Todos os dados coletados são mostrados nos três gráficos a seguir:

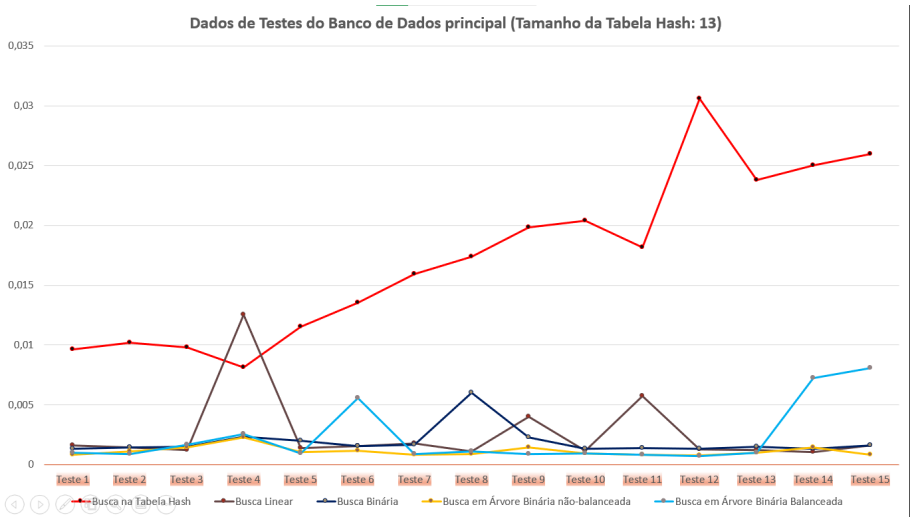

Figure 17. Dados de Testes do Banco de Dados principal - Tamanho da Tabela Hash: 13

Melhores resultados em cada teste (Caso 1):

Teste 1: 0,000832 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,00091 (Busca em Árvore Binária balanceada);

Teste 3: 0,001432 (Busca em Árvore Binária não-balanceada);

Teste 4: 0,002279 (Busca em Árvore Binária não-balanceada);

Teste 5: 0,000939 (Busca em Árvore Binária balanceada);

Teste 6: 0,00118 (Busca em Árvore Binária não-balanceada);

Teste 7: 0,000848 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,000888 (Busca em Árvore Binária não-balanceada);

Teste 9: 0,000872 (Busca em Árvore Binária balanceada);

Teste 10: 0,000919 (Busca em Árvore Binária balanceada);

Teste 11: 0,000811 (Busca em Árvore Binária balanceada);

Teste 12: 0,000721 (Busca em Árvore Binária balanceada);

Teste 13: 0,001004 (Busca em Árvore Binária não-balanceada);

Teste 14: 0,001068 (Busca Binária);

Teste 15: 0,00081 (Busca em Árvore Binária não-balanceada);

Piores resultados em cada teste (Caso 1):

Teste 1: 0,009664 (Busca na Tabela Hash);

Teste 2: 0,010206 (Busca na Tabela Hash);

Teste 3: 0,009793 (Busca na Tabela Hash);

Teste 4: 0,012527 (Busca Linear);

Teste 5: 0,011548 (Busca na Tabela Hash);

Teste 6: 0,013532 (Busca na Tabela Hash);

Teste 7: 0,015922 (Busca na Tabela Hash);

Teste 8: 0,017361 (Busca na Tabela Hash);

Teste 9: 0,019818 (Busca na Tabela Hash);

Teste 10: 0,020407 (Busca na Tabela Hash);

Teste 11: 0,018175 (Busca na Tabela Hash);

Teste 12: 0,030616 (Busca na Tabela Hash);

Teste 13: 0,023805 (Busca na Tabela Hash);

Teste 14: 0,025037 (Busca na Tabela Hash);

Teste 15: 0,025971 (Busca na Tabela Hash);

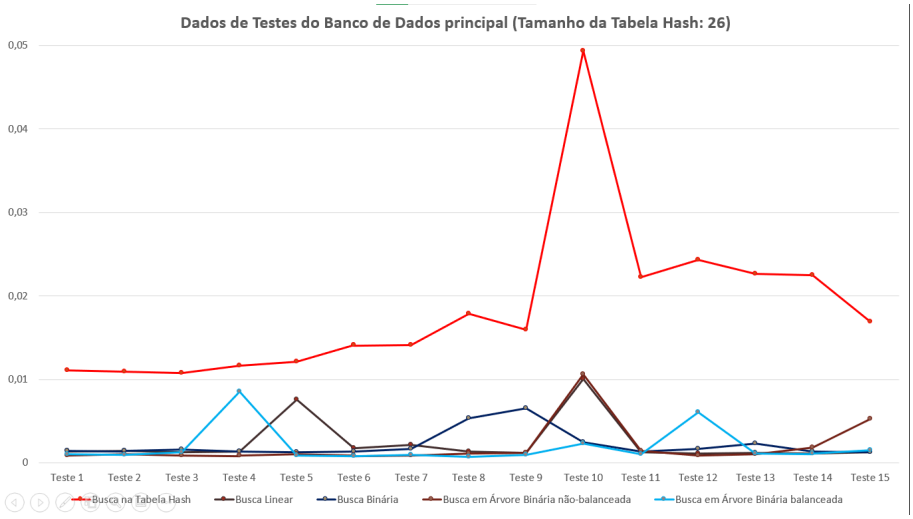

Figure 18. Dados de Testes do Banco de Dados principal - Tamanho da Tabela Hash: 26

Melhores resultados em cada teste (Caso 2):

Teste 1: 0,000903 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,000985 (Busca em Árvore Binária balanceada);

Teste 3: 0,000883 (Busca em Árvore Binária não-balanceada);

Teste 4: 0,000832 (Busca em Árvore Binária não-balanceada);

Teste 5: 0,000891 (Busca em Árvore Binária balanceada);

Teste 6: 0,000809 (Busca em Árvore Binária balanceada);

Teste 7: 0,000893 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,000722 (Busca em Árvore Binária balanceada);

Teste 9: 0,000923 (Busca em Árvore Binária balanceada);

Teste 10: 0,002291 (Busca em Árvore Binária balanceada);

Teste 11: 0,00101 (Busca em Árvore Binária balanceada);

Teste 12: 0,000873 (Busca em Árvore Binária não-balanceada);

Teste 13: 0,001035 (Busca em Árvore Binária não-balanceada);

Teste 14: 0,001084 (Busca em Árvore Binária balanceada);

Teste 15: 0,001227 (Busca Binária);

Piores resultados em cada teste (Caso 2):

Teste 1: 0,011116 (Busca na Tabela Hash);

Teste 2: 0,01093 (Busca na Tabela Hash);

Teste 3: 0,01072 (Busca na Tabela Hash);

Teste 4: 0,011649 (Busca na Tabela Hash);

Teste 5: 0,012134 (Busca na Tabela Hash);

Teste 6: 0,014134 (Busca na Tabela Hash);

Teste 7: 0,014143 (Busca na Tabela Hash);

Teste 8: 0,017877 (Busca na Tabela Hash);

Teste 9: 0,015972 (Busca na Tabela Hash);

Teste 10: 0,049395 (Busca na Tabela Hash);

Teste 11: 0,022271 (Busca na Tabela Hash);

Teste 12: 0,024304 (Busca na Tabela Hash);

Teste 13: 0,022665 (Busca na Tabela Hash);

Teste 14: 0,022522 (Busca na Tabela Hash);

Teste 15: 0,016896 (Busca na Tabela Hash);

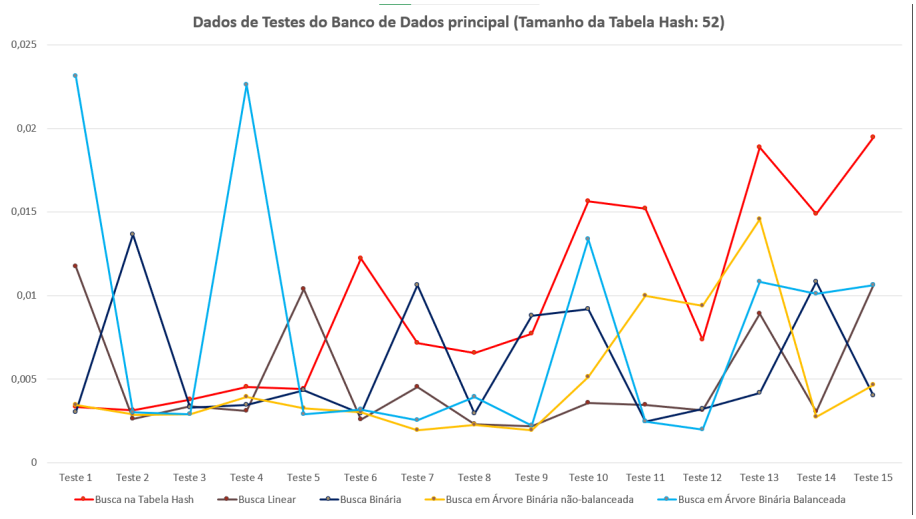

Figure 19. Dados de Testes do Banco de Dados principal - Tamanho da Tabela Hash: 52

Melhores resultados em cada teste (Caso 3):

Teste 1: 0,003004 (Busca Binária);

Teste 2: 0,00263 (Busca Linear);

Teste 3: 0,002902 (Busca em Árvore Binária balanceada);

Teste 4: 0,0031 (Busca Linear);

Teste 5: 0,002909 (Busca em Árvore Binária balanceada);

Teste 6: 0,002565 (Busca Linear);

Teste 7: 0,001959 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,002256 (Busca em Árvore Binária não-balanceada);

Teste 9: 0,00196 (Busca em Árvore Binária não-balanceada);

Teste 10: 0,003566 (Busca Linear);

Teste 11: 0,002466 (Busca em Árvore Binária balanceada);

Teste 12: 0,001975 (Busca em Árvore Binária balanceada);

Teste 13: 0,004174 (Busca Binária);

Teste 14: 0,002724 (Busca em Árvore Binária não-balanceada);

Teste 15: 0,004024 (Busca Binária);

Piores resultados em cada teste (Caso 3):

Teste 1: 0,023127 (Busca em Árvore Binária balanceada);

Teste 2: 0,0113643 (Busca Binária);

Teste 3: 0,003779 (Busca na Tabela Hash);

Teste 4: 0,022606 (Busca em Árvore Binária balanceada);

Teste 5: 0,010368 (Busca Linear);

Teste 6: 0,012227 (Busca na Tabela Hash);

Teste 7: 0,010619 (Busca Binária);

Teste 8: 0,00656 (Busca na Tabela Hash);

Teste 9: 0,008812 (Busca Binária);

Teste 10: 0,01563 (Busca na Tabela Hash);

Teste 11: 0,015208 (Busca na Tabela Hash);

Teste 12: 0,009397 (Busca na Árvore Binária não-balanceada);

Teste 13: 0,018856 (Busca na Tabela Hash);

Teste 14: 0,01489 (Busca na Tabela Hash);

Teste 15: 0,019455 (Busca na Tabela Hash);

Nos gráficos demonstrados, cada linha de uma cor representa o Algoritmo de Busca testado. Como exemplo: a Busca na Tabela Hash representada pela cor vermelha e a Busca Binária representada pela cor azul-escuro. Em cada teste, a linha que apresenta o menor valor é considerada o método que apresentou o melhor resultado na busca, enquanto o que tiver o maior valor é considerado o pior resultado.

Como mostrado nos gráficos, a Busca na Tabela Hash foi o método que apresentou os piores resultados, enquanto a Busca em Árvore Binária Balanceada apresentou os melhores resultados. No entanto, com o aumento de tamanho da Tabela Hash, os valores apresentados pela sua busca melhoraram. Como exemplo, podem-se destacar os piores resultados do teste 3 no caso 2 e 3.

No caso 2, esse método apresentou um resultado de 0,01072 segundos, enquanto no caso 3, ele apresentou um resultado de 0,003779 segundos. Nessa situação, pode-se notar que, com o aumento do tamanho da tabela, o tempo de execução da busca sofreu uma redução, tornando a função de busca mais rápida que no caso anterior. Em outros testes, essa mudança mostra-se mais aparente, principalmente nos resultados do caso 3, em que a Busca na Tabela Hash não apresenta o pior resultado entre os outros métodos de busca.

Para fazer uma análise um pouco mais aprofundada nesses casos, foram elaborados mais três gráficos de testes, utilizando um banco de dados maior que o banco utilizado inicialmente, apresentando um total de 26.142 registros. Nesse caso, os gráficos apresentam as mesmas características que os anteriores, incluindo as mudanças no tamanho da Tabela Hash.

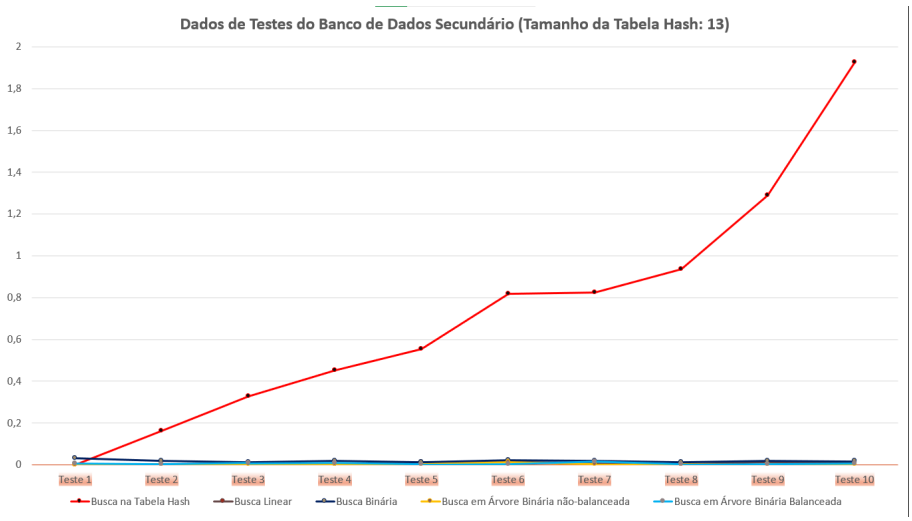

Figure 20. Dados de Testes do Banco de Dados Secundário - Tamanho da Tabela Hash: 13

Melhores resultados em cada teste (Caso 4):

Teste 1: 0,0007753 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,0018759 (Busca em Árvore Binária não-balanceada);

Teste 3: 0,0023987 (Busca em Árvore Binária não-balanceada);

Teste 4: 0,002465 (Busca em Árvore Binária não-balanceada);

Teste 5: 0,0023057 (Busca em Árvore Binária balanceada);

Teste 6: 0,0028992 (Busca em Árvore Binária balanceada);

Teste 7: 0,0018227 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,0019343 (Busca em Árvore Binária balanceada);

Teste 9: 0,0034579 (Busca em Árvore Binária balanceada);

Teste 10: 0,0034586 (Busca em Árvore Binária não-balanceada);

Piores resultados em cada teste (Caso 4):

Teste 1: 0,0312048 (Busca Binária);

Teste 2: 0,1627857 (Busca na Tabela Hash);

Teste 3: 0,3263688 (Busca na Tabela Hash);

Teste 4: 0,4519478 (Busca na Tabela Hash);

Teste 5: 0,5539413 (Busca na Tabela Hash);

Teste 6: 0,8184497 (Busca na Tabela Hash);

Teste 7: 0,8260912 (Busca na Tabela Hash);

Teste 8: 0,9346337 (Busca na Tabela Hash);

Teste 9: 1,2890516 (Busca na Tabela Hash);

Teste 10: 1,9261386 (Busca na Tabela Hash);

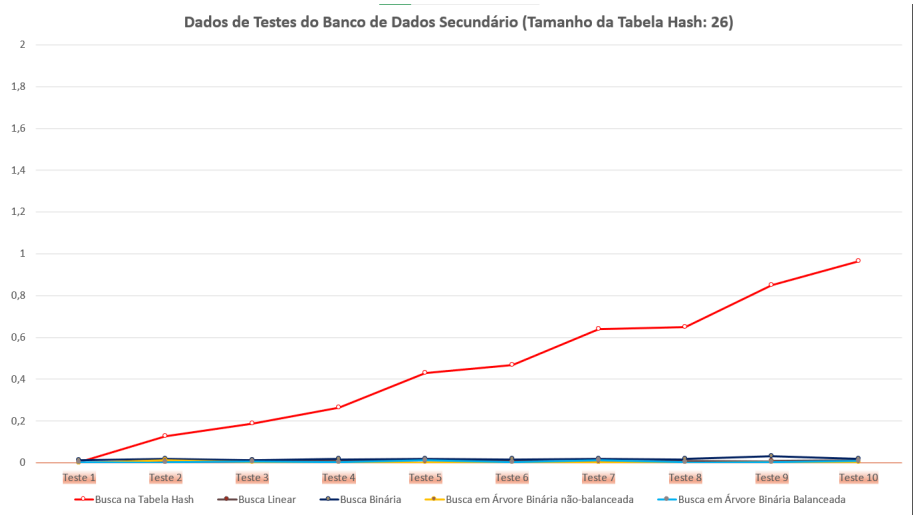

Figure 21. Dados de Testes do Terceiro Banco de Dados - Tamanho da Tabela Hash: 26

Melhores resultados em cada teste (Caso 5):

Teste 1: 0,0007059 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,0020083 (Busca em Árvore Binária balanceada);

Teste 3: 0,0032905 (Busca em Árvore Binária não-balanceada);

Teste 4: 0,0020297 (Busca em Árvore Binária balanceada);

Teste 5: 0,0029811 (Busca em Árvore Binária não-balanceada);

Teste 6: 0,0023361 (Busca em Árvore Binária balanceada);

Teste 7: 0,0029899 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,0020914 (Busca em Árvore Binária balanceada);

Teste 9: 0,0023433 (Busca em Árvore Binária não-balanceada);

Teste 10: 0,003926 (Busca em Árvore Binária não-balanceada);

Piores resultados em cada teste (Caso 5):

Teste 1: 0,0128077 (Busca Binária);

Teste 2: 0,1263684 (Busca na Tabela Hash);

Teste 3: 0,1865943 (Busca na Tabela Hash);

Teste 4: 0,2634283 (Busca na Tabela Hash);

Teste 5: 0,4305976 (Busca na Tabela Hash);

Teste 6: 0,467952 (Busca na Tabela Hash);

Teste 7: 0,6408624 (Busca na Tabela Hash);

Teste 8: 0,6492772 (Busca na Tabela Hash);

Teste 9: 0,850292 (Busca na Tabela Hash);

Teste 10: 0,9659353 (Busca na Tabela Hash);

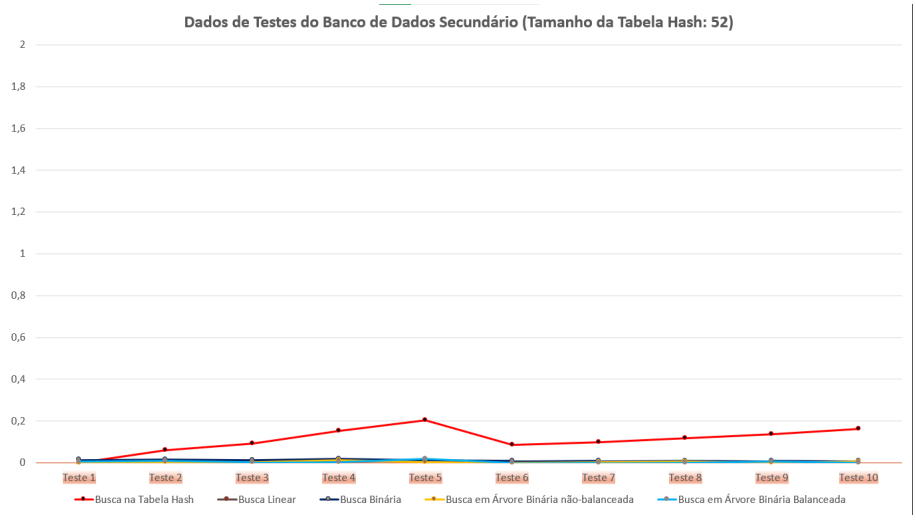

Figure 22. Dados de Testes do Terceiro Banco de Dados - Tamanho da Tabela Hash: 52

Melhores resultados em cada teste (Caso 6):

Teste 1: 0,0004907 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,0021097 (Busca em Árvore Binária não-balanceada);

Teste 3: 0,001971 (Busca em Árvore Binária balanceada);

Teste 4: 0,0040917 (Busca em Árvore Binária balanceada);

Teste 5: 0,001982 (Busca em Árvore Binária não-balanceada);

Teste 6: 0,0009218 (Busca em Árvore Binária não-balanceada);

Teste 7: 0,001008 (Busca em Árvore Binária balanceada);

Teste 8: 0,0010939 (Busca em Árvore Binária balanceada);

Teste 9: 0,0010028 (Busca em Árvore Binária não-balanceada);

Teste 10: 0,0007696 (Busca em Árvore Binária balanceada);

Piores resultados em cada teste (Caso 6):

Teste 1: 0,0140454 (Busca Binária);

Teste 2: 0,0593032 (Busca na Tabela Hash);

Teste 3: 0,0917587 (Busca na Tabela Hash);

Teste 4: 0,1525775 (Busca na Tabela Hash);

Teste 5: 0,2044444 (Busca na Tabela Hash);

Teste 6: 0,0857626 (Busca na Tabela Hash);

Teste 7: 0,0970384 (Busca na Tabela Hash);

Teste 8: 0,1173344 (Busca na Tabela Hash);

Teste 9: 0,1379406 (Busca na Tabela Hash);

Teste 10: 0,1634624 (Busca na Tabela Hash);

Como mostrado nos gráficos, a Busca na Tabela Hash continuou apresentando os piores resultados, porém, assim como nos casos anteriores, com o aumento do tamanho da Tabela Hash, os resultados apresentaram uma melhora notável. Pode-se destacar os testes 9 e 10 dos casos 4 e 6.

No caso 4, o teste 9 apresentou um tempo de execução de 1,2890516 segundos e o teste 10 apresentou um tempo de 1,9261386 segundos. Já no caso 6, o teste 9 e 10 apresentaram tempos de execução de 0,1379406 e 0,1634624 segundos, respectivamente. Com isso, pode-se notar que com o aumento do tamanho da Tabela Hash, os testes apresentam resultados melhores, conforme descrito nos casos anteriores.

Para complementar as análises feitas, foram realizados mais três séries de testes com o banco de dados de 26.142 registros, e utilizando na Tabela Hash os tamanhos 500.000, 5.000.000 e 100.000.000.

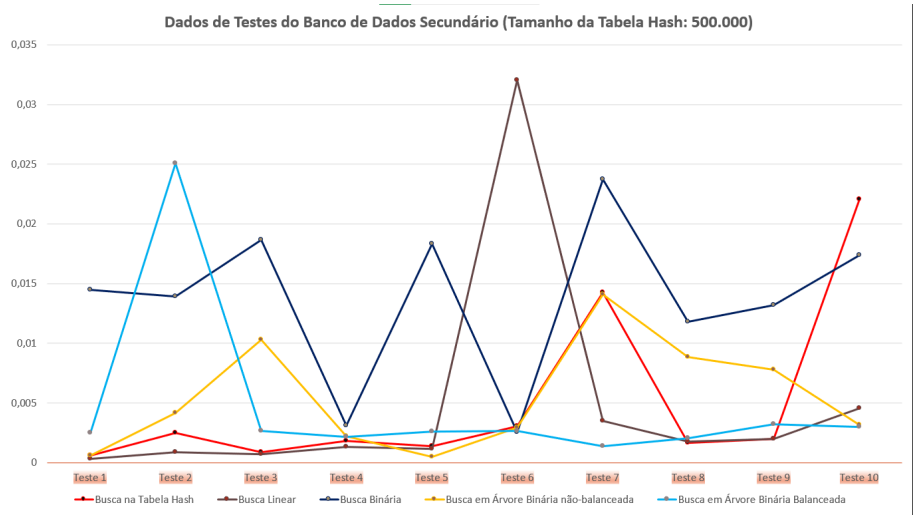

Figure 23. Dados de Testes do Banco de Dados Secundário - Tamanho da Tabela Hash: 500.000

Melhores resultados em cada teste (Caso 7):

Teste 1: 0,0003322000 (Busca Linear);

Teste 2: 0,0008897000 (Busca Linear);

Teste 3: 0,0007426000 (Busca Linear);

Teste 4: 0,0013150000 (Busca Linear);

Teste 5: 0,0004897000 (Busca em Árvore Binária não-balanceada);

Teste 6: 0,0025639000 (Busca Binária);

Teste 7: 0,0013744000 (Busca em Árvore Binária balanceada);

Teste 8: 0,0016666000 (Busca na Tabela Hash);

Teste 9: 0,0019743000 (Busca na Tabela Hash);

Teste 10: 0,0030136000 (Busca em Árvore Binária balanceada);

Piores resultados em cada teste (Caso 7):

Teste 1: 0,0144739000 (Busca Binária);

Teste 2: 0,0250797000 (Busca na Árvore Binária de Busca não-balanceada);

Teste 3: 0,0186435000 (Busca Binária);

Teste 4: 0,0030981000 (Busca Binária);

Teste 5: 0,0183227000 (Busca Binária);

Teste 6: 0,0320348000 (Busca Linear);

Teste 7: 0,0237586000 (Busca Binária);

Teste 8: 0,0118048000 (Busca Binária);

Teste 9: 0,0132326000 (Busca Binária);

Teste 10: 0,0220869000 (Busca na Tabela Hash);

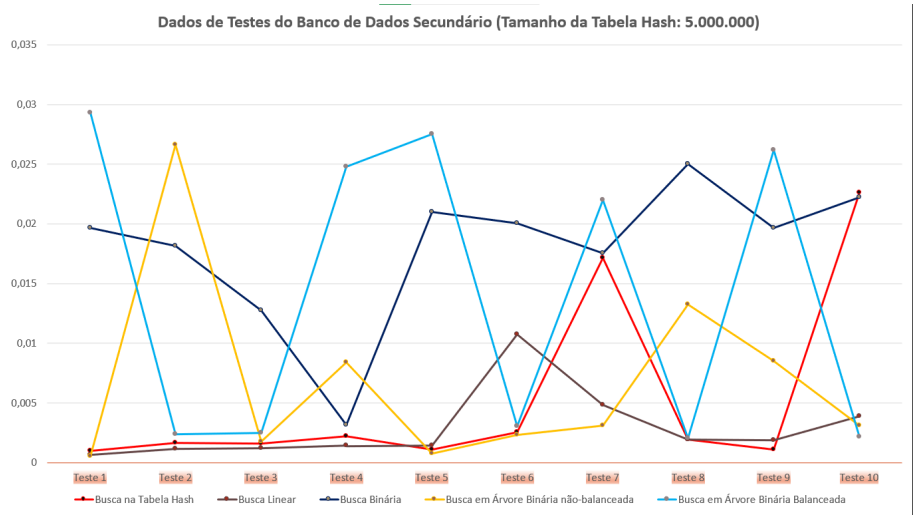

Figure 24. Dados de Testes do Banco de Dados Secundário - Tamanho da Tabela Hash: 5.000.000

Melhores resultados em cada teste (Caso 8):

Teste 1: 0,0005515000 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,0011680000 (Busca Linear);

Teste 3: 0,0012196000 (Busca Linear);

Teste 4: 0,0014430000 (Busca Linear);

Teste 5: 0,0007684000 (Busca em Árvore Binária não-balanceada);

Teste 6: 0,0023443000 (Busca em Árvore Binária não-balanceada);

Teste 7: 0,0031359000 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,0019325000 (Busca na Tabela Hash);

Teste 9: 0,0011222000 (Busca na Tabela Hash);

Teste 10: 0,0021642000 (Busca em Árvore Binária balanceada);

Piores resultados em cada teste (Caso 8):

Teste 1: 0,0293388000 (Busca em Árvore Binária balanceada);

Teste 2: 0,0266522000 (Busca na Árvore Binária de Busca não-balanceada);

Teste 3: 0,0127851000 (Busca Binária);

Teste 4: 0,0248087000 (Busca em Árvore Binária balanceada);

Teste 5: 0,0275087000 (Busca em Árvore Binária balanceada);

Teste 6: 0,0200700000 (Busca Binária);

Teste 7: 0,0175778000 (Busca Binária);

Teste 8: 0,0250253000 (Busca Binária);

Teste 9: 0,0262034000 (Busca em Árvore Binária balanceada);

Teste 10: 0,0226321000 (Busca na Tabela Hash);

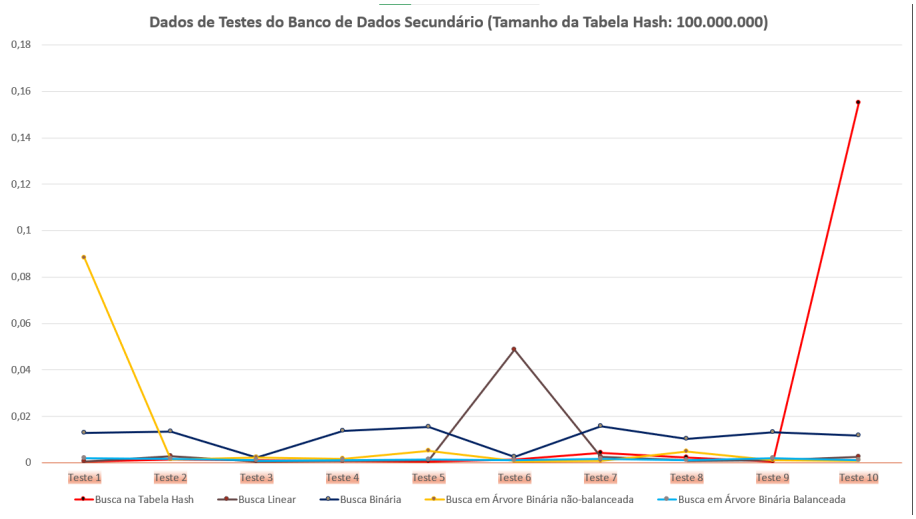

Figure 25. Dados de Testes do Banco de Dados Secundário - Tamanho da Tabela Hash: 100.000.000

Melhores resultados em cada teste (Caso 9):

Teste 1: 0,0004896000 (Busca na Tabela Hash);

Teste 2: 0,0014712000 (Busca na Tabela Hash);

Teste 3: 0,0006129000 (Busca Linear);

Teste 4: 0,0008001000 (Busca Linear);

Teste 5: 0,0006231000 (Busca na Tabela Hash);

Teste 6: 0,0008865000 (Busca em Árvore Binária não-balanceada);

Teste 7: 0,0009264000 (Busca em Árvore Binária não-balanceada);

Teste 8: 0,0009729000 (Busca Linear);

Teste 9: 0,0006553000 (Busca na Tabela Hash);

Teste 10: 0,0009971000 (Busca em Árvore Binária balanceada);

Piores resultados em cada teste (Caso 9):

Teste 1: 0,0883351000 (Busca em Árvore Binária não-balanceada);

Teste 2: 0,0134176000 (Busca Binária);

Teste 3: 0,0022683000 (Busca Binária);

Teste 4: 0,0136954000 (Busca Binária);

Teste 5: 0,0154199000 (Busca Binária);

Teste 6: 0,0487523000 (Busca Linear);

Teste 7: 0,0156933000 (Busca Binária);

Teste 8: 0,0102199000 (Busca Binária);

Teste 9: 0,0132356000 (Busca Binária);

Teste 10: 0,1550100000 (Busca na Tabela Hash);

Com o aumento colossal dos tamanhos utilizados na Tabela Hash, sua busca deixou de apresentar os piores resultados entre os outros algoritmos. Como exemplo, podem-se destacar o teste 8 dos casos 6 e 8.

No caso 6, a Busca na Tabela Hash apresentou um tempo de 0,1173344, considerado o pior resultado nesse teste em específico. No entanto, no caso 8, essa busca apresentou um tempo de 0,0019325000, sendo o melhor resultado no teste 8.

Com base nos testes realizados ao longo do experimento, pode-se notar que, com o aumento gradual do tamanho da Tabela Hash, sua busca obteve resultados cada vez melhores. Dessa maneira, com o aumento do tamanho, ocorre uma expansão número de posições disponíveis para inserção de registros , o quê por consequência, promove uma redução no número de colisões que podem acontecer ao longo da inserção.

Com a redução das colisões, as situações em que diversos registros se encontram na mesma posição diminuem, contribuindo com uma função de busca mais eficiente do que anteriormente demonstrado.

7. Conclusão

Diante dos estudos feitos e dos dados coletados sobre a Tabela Hash, pode-se concluir os seguintes pontos:

Em comparação aos outros algoritmos de busca, como a Árvore Binária, a Tabela Hash apresentou os piores resultados, em geral. No entanto, com o aumento gradual do tamanho da Tabela Hash, seu tempo de busca apresentou resultados melhores.

Nesse caso, com o aumento do tamanho, o número de posições disponíveis para a inserção de registros aumenta, levando a uma diminuição no número de colisões durante o processo de inserção, resultando em uma redução no número de situações em que muitos registros são inseridos no mesmo local.

Com isso, esse aumento no tamanho da Tabela Hash reduziu o tempo de execução da busca dentro da mesma, tornando em um algoritmo de busca mais eficiente comparado aos outros utilizados no experimento.

Por fim, a Tabela Hash permanece sendo o algoritmo de busca mais apropriado para banco de dados com muitos registros, tal como o de animais em risco de extinção em um continente inteiro. No entanto, é necessário estabelecer um tamanho adequado para as inserções dos registros, a fim de proporcionar um tempo de busca melhor ao longo do processo. Portanto, pode-se afirmar que dispomos de uma ferramenta a ser agregada na busca de informações para a redução do risco de extinção da fauna.

8. Bibliografia

8.1. Informações sobre os animais em extinção

IUCN RedList. Organização que fornece bancos de dados e informações sobre os animais em risco de extinção. IUCN RedList, 2020. Disponível em: https://www.iucnredlist.org/ - Acessado em: 16 de Setembro de 2020.

IUCN RedList. Informações sobre a classificação dos animais no banco de dados e as nomeclaturas usadas. IUCN RedList, 2020. Disponível em: https://portals.iucn.org/library/sites/library/files/documents/RL-2012-002.pdf - Acessado em: 22 de Setembro de 2020.

Ministério do Meio Ambiente. Artigo sobre a influência do homem no meio ambiente. EBC, 2015. Disponível em: https://memoria.ebc.com.br/infantil/voce-sabia/2015/03/homem-e-principal-agente-no-processo-de-extincao-de-animais - Acessado em: 16 de Setembro de 2020.

DIANA, Juliana. Artigo sobre exemplos de animais extintos no mundo. TodaMatéria, 2019. Disponível em: https://www.todamateria.com.br/animais-extintos/ - Acessado em: 16 de Setembro de 2020.

8.2. Banco de Dados

PANCHAL, Utsav. Vídeo sobre como inserir um arquivo CSV em um vetor de Structs. Canal Utsav Panchal, 2020. Disponível em: https://www.youtube.com/watch?v=uUcDSZ8CFhg - Acessado em: 13 de Outubro de 2020.

8.3. Tabela Hash

CORMEN, Thomas H. Algoritmos: Teoria e Prática. Elsevier Editora Ltda, 2012. Acessado em: 24 de Setembro de 2020.

TutorialPoint. Artigo sobre a estrutura e funcionalidades da Tabela Hash. TutorialPoint, ??. Disponível em: https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm - Acessado em: 24 de Setembro de 2020.

Computer Science. Explicação sobre a Tabela Hash e suas funções. Canal Computer Science, 2017. Disponível em: https://www.youtube.com/watch?v=KyUTuwz_b7Q - Acessado em: 26 de Setembro de 2020.

SCHILDT, Herbert. C Completo e Total. Makron Books do Brasil Editora Ltda, 1997. Acessado em: 28 de Setembro de 2020.

RANDERSON. Explicação sobre o processo da Sondagem Quadrática. Canal Randerson112358, 2016. Disponível em: https://www.youtube.com/watch?v=BoZbu1cR0no - Acessado em: 7 de Novembro de 2020.

8.4. Busca Linear

SCHILDT, Herbert. C Completo e Total. Makron Books do Brasil Editora Ltda, 1997. Acessado em: 28 de Setembro de 2020.

RICARTE, Ivan. Explicação sobre a complexidade da Busca Linear. Unicamp, 2003. Disponível em: http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node19.html - Acessado em: 6 de Novembro de 2020.

8.5. Busca Binária

SCHILDT, Herbert. C Completo e Total. Makron Books do Brasil Editora Ltda, 1997. Acessado em: 28 de Setembro.

CORMEN, Thomas H; BALKCOM, Devin. Artigo sobre a Busca Binária e sua aplicação. Khan Academy, 2015. Disponível em: https://pt.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search - Acessado em: 7 de Outubro de 2020.

BACKES, André. Explicação sobre a complexidade da Busca Binária. Slides Prof.André Backes, ??. Disponível em: http://www.facom.ufu.br/ backes/gsi011/Aula05-Busca.pdf - Acessado em: 6 de Novembro de 2020.

8.6. Árvore Binária de Busca

FEOFILOFF, Paulo. Artigo sobre a Árvore Binária de Busca e sua aplicação. IME-USP, 2007. Disponível em: https://www.ime.usp.br/ pf/algoritmos/aulas/bint.html - Acessado em: 8 de Outubro de 2020.

R,Haribalaji. Artigo sobre a construção de uma Árvore Binária usando valores de um vetor. GeeksforGeeks, 2019. Disponível em: https://www.geeksforgeeks.org/construct-complete-binary-tree-given-array/ - Acessado em: 28 de Outubro de 2020.

DUARTE, Dinho. Explicação sobre as funções de inserção e remoção de elementos, e sua estrutura. Canal Dinho Duarte, 2019. Disponível em: https://www.youtube.com/watch?v=owaBsGQheZM - Acessado em: 28 de Outubro de 2020.

CHIRAG. Artigo sobre como fazer a inserção de valores repetidos em uma Árvore Binária. GeeksforGeeks, 2019. Disponível em: https://www.geeksforgeeks.org/how-to-handle-duplicates-in-binary-search-tree/ - Acessado em: 28 de Outubro de 2020.

DUARTE, Dinho. Explicação sobre a função de busca de elementos em uma Árvore Binária, e sua estrutura. Canal Dinho Duarte, 2019. Disponível em:

https://www.youtube.com/watch?v=gR_-bnL628slist=PLsq8K378bmLNY9G5CVw6s3s9ogdW-GsSjindex=2 - Acessado em: 30 de Outubro de 2020.

RAI, Akanksha. Referência utilizada para a demonstração de testes utilizando a estrutura da Árvore Binária de Busca balanceada. GeeksforGeeks, 2020. Disponível em: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/ - Acessado em: 30 de Outubro de 2020.

SONG, Siang Wun. Explicação sobre a complexidade da Árvore Binária de Busca. IME/USP, ??. Disponível em: https://www.ime.usp.br/ song/mac5710/slides/06bst.pdf - Acessado em: 7 de Novembro de 2020.