REO 7

IMPLEMENTAÇÃO DE ÁRVORE B+ A PARTIR DE CÓDIGO DE SEQUENCE SET

CLÁUDIO MANUEL DOS REIS JUNIOR

GABRIEL NATHAN ALMEIDA SILVA

IGOR CUNHA FERREIRA

LAVRAS - MG

1. INTRODUÇÃO

As estruturas de dados podem ser utilizadas para manipular arquivos binários, o que muitas vezes não representa uma atividade fácil, visto que na maioria dos casos são necessárias diversas inserções, remoções, alterações, buscas, ordenações, etc (Uchôa et al.., 2021). Por isso, utiliza-se o Sequence Set, que é um arquivo binário que se encontra dividido em pedaços de tamanhos iguais, cada um contendo uma sequência de registros ordenados (Uchôa *et al.*, 2021). A Figura 1 representa um Sequence Set.

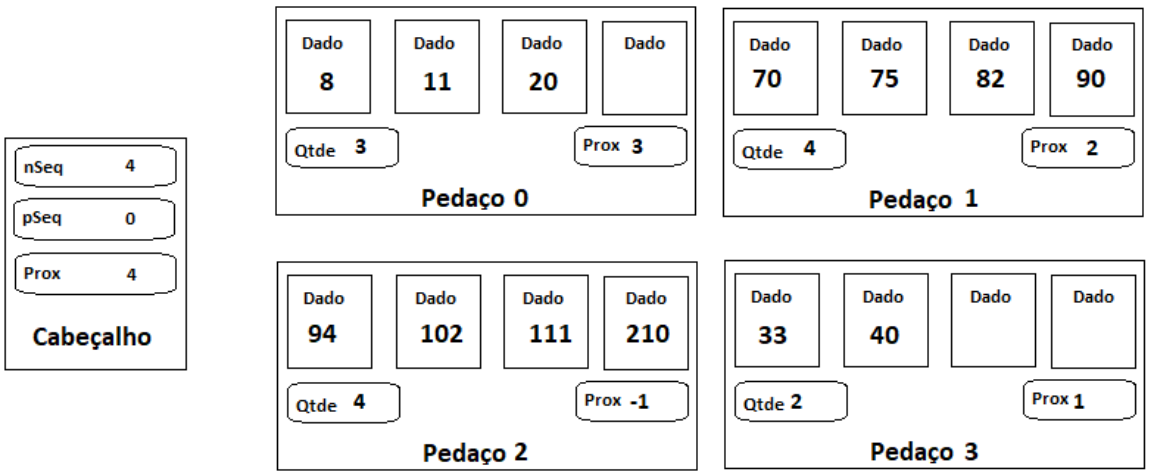

Figura 1 – Sequence Set

Fonte: adaptado de Uchôa, 2021

Através da Figura 1 é possível perceber que o arquivo está dividido em pedaços, cada pedaço está ordenado guardando o número de chaves e apontando para o próximo. Além disso, o cabeçalho do Sequence Set possui como informações padrões o número de pedaços, a primeira sequência e o próximo pedaço que será utilizado. Assim na Figura 1, o número de pedaços é quatro, a primeira sequência é o pedaço zero e o próximo pedaço é o quatro.

Para acessar efetivamente os registros do arquivo binário pode-se utilizar a Árvore B+, que é uma variante da Árvore B que permite acesso sequencial mais eficiente aos registros do arquivo (Falcão et al., 2021). Árvores B+ são semelhantes às árvores B em todos os níveis, exceto ao de folha. Ao nível terminal as páginas são encadeadas umas às outras formando uma lista denominada sequence set. As páginas terminais podem até ser armazenadas em outro tipo de meio e sua ordem pode ser diferente da ordem das demais páginas. Não é preciso que as páginas terminais tenham campos de ponteiros para descendentes. Por outro lado, as não-terminais(indexadoras) não necessitam de ponteiros para registro do arquivo principal (Ferraz,

2003). Portanto, a Árvore B+ representa a combinação da Árvore B com o Sequence Set, pois ela armazena os dados na folha, que são Sequence Sets, e os índices são armazenados em nós intermediários, que representa uma Árvore B. Portanto, a Árvore B permite a busca rápida enquanto o Sequence Set o acesso sequencial dos arquivos (Uchôa et al., 2021). A Figura 2 representa uma Árvore B+.

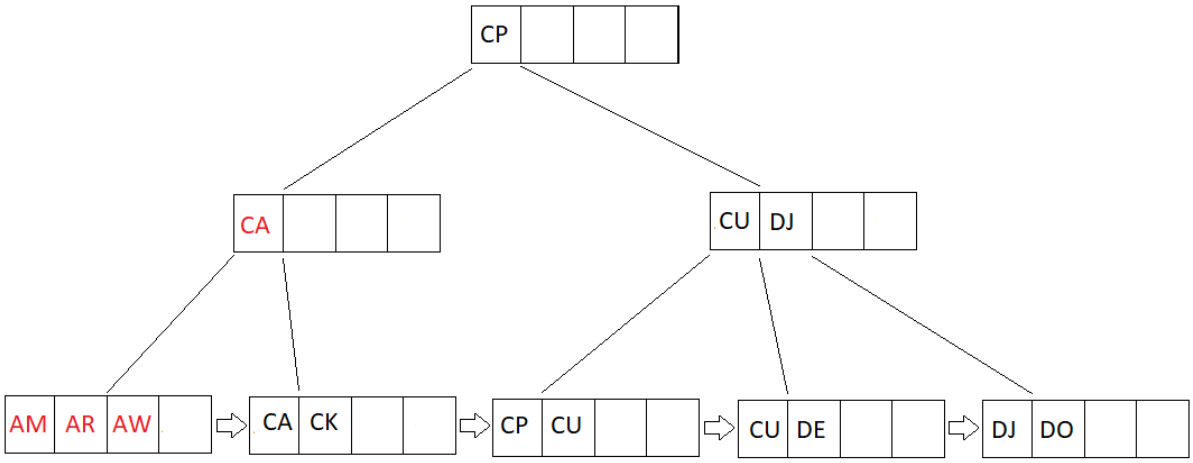

Figura 2 – Árvore B+

Fonte: do autor

As árvores B+ apresentam algumas propriedades, como por exemplo, em uma Árvores B+ de Ordem M:

1. A raiz tem 0,2 ou entre [EM/2] e M filhos;

2. Todas as páginas, com exceção da raíz e das páginas terminais, têm no mínimo [EM/2] e no máximo M filhos;

3.Todas as páginas terminais aparecem no mesmo nível, ou seja, estão à mesma distância da raiz;

4.Uma página terminal com K filhos tem k-1 chaves;

5. Páginas terminais representam o conjunto de sequência de dados do arquivo e são concatenadas juntos. (SEQUENCE SET);

Assim, o objetivo desse trabalho é realizar a combinação do Sequence Set e da Árvore B para a implementação de uma Árvore B+, sendo que o Sequence Set representa as folhas da árvore e a Árvore B o índice.

2. METODOLOGIA

Para a composição do programa, dividiu-se o código em classes, sendo essas: Pacote, SequenceSet, Noh e ArvoreBMais. Além disso, para criação de um objeto, utilizaram-se duas structs, que foram nomeadas como Pokemon e CabecalhoArqSS.

2.1. Struct Pokemon

A struct Pokemon foi criada para armazenar as características de um objeto, o Pokemon, que será manipulado na Árvore B+.

As características foram denominadas através de nove atributos, um atributo do tipo unsigned, denominado “id”, dois atributos do tipo char, sendo um nomeado como “nome” e outro como “tipo”, sendo que ambos são vetores com capacidade de armazenamento de até quinze posições. Por fim, seis atributos do tipo unsigned nomeados como “total”, “ataque”, “defesa”, “at_esp”, “def_esp” e velocidade, respectivamente.

É importante ressaltar que se optou por usar como chave o atributo “id” ao invés do atributo “nome”, devido a alguns empecilhos encontrados na execução da árvore.

2.2. Classe Pacote

A classe Pacote, representada pelas folhas da árvore na Figura 2, também representa as páginas do Sequence Set e os objetos dessa classe são criados e manuseados pela classe SequenceSet.

Para isso, essa classe possui quatro métodos, sendo o “inserir” responsável por fazer a inserção de um novo dado no Sequence Set, o “imprimir” foi criado para imprimir os elementos do pacote, o método “chaveEhMaiorQueTodos” recebe uma chave e retorna um valor do tipo booleano verdadeiro caso a chave recebida seja maior que todos os elementos do pacote e o método “chaveEhMenorQueTodos” que realiza o processo contrário com relação ao “chaveEhMaiorQueTodos”, cujo o valor do tipo booleano retorna verdadeiro caso a chave passada por parâmetro for menor que todos os elementos do pacote.

2.3. Classe SequenceSet

Para criar o cabeçalho do SequenceSet criou-se a struct CabecalhoArqSS que armazena quatro atributos do tipo unsigned denominados como “capacidadeMaxPacote”, “capacidadeMinPacote”, “posiçãoMeio” e “numPacotes”, respectivamente. Além disso, ela guarda dois atributos do tipo inteiro nomeados como “posPrimeiroPacote” e “proxPosicaoVazia”.

Para a construção do Sequence Set se utiliza o método “sequenceset”. Primeiramente, o método recebe o arquivo e verifica se o arquivo existe e caso exista, lê o cabeçalho e verifica se os dados são consistentes com a configuração do Sequence Set e em seguida se atualiza os dados do Sequence Set de acordo com o cabeçalho do arquivo. Caso os dados não estejam de acordo as configurações do Sequence Set, gera-se um erro e se o arquivo de entrada ainda não existir, se inicializa um novo Sequence Set e salva o cabeçalho no arquivo.

O destrutor do Sequence Set é representado pelo método “~sequenceset” e esse método atualiza o cabeçalho, caso o objeto atual não seja mais desejado pelo usuário.

Para saber se as configurações utilizadas no cabeçalho são iguais as utilizadas no arquivo, utiliza-se o método “cabecalhoEhConsistente” que retorna um valor booleano verdadeiro ou falso.

Para atualizar os atributos do cabeçalho é utilizado o método “atualizaCabecalhoNoArquivo”.

Para ler os arquivos de um pacote em uma posição desejada pelo usuário utiliza-se o método “lerPacoteDoArquivo”.

Para gravar pacotes, através de uma posição e um pacote solicitado pelo usuário, utiliza-se o método “gravarPacoteNoArquivo”.

Para procurar algum pacote vazio disponível no meio do arquivo se utiliza o método “encontrarProxPosDisponivel”, que retorna a posição disponível para se inserir dados em um pacote vazio. Caso não haja nenhum pacote vazio disponível no meio do arquivo, a próxima posição disponível será uma posição a mais que a ultima do arquivo.

Para inserir um dado no Sequence Set, se utiliza o método “inserirDado”, que recebe um dado a ser inserido e uma posição. Posteriormente procura-se a posição do pacote para a inserção do dado, através do método “encontrarPacoteParaInsercao”. Caso o pacote encontrado

tenha espaço, o dado é inserido, caso o pacote esteja cheio, se faz a divisão do pacote, utilizando o método “dividirPacote”, e o dado é inserido no novo pacote com espaço vazio.

O método “encontrarPacoteParaInsercao” retorna dois elementos: um pacote, que já está carregado, em que será feita a inserção, e a posição relativa desse pacote para facilitar a gravação posterior após as alterações. O método recebe um pacote recém criado e o dado para busca.

O método “dividirPacote” copia a metade superior dos dados de um pacote atual para um pacote novo e retorna o pacote novo.

Para verificar se algum elemento está no Sequence Set utiliza-se o método “buscar” que recebe uma chave, procura qual pacote poderia conter essa chave e verifica se ela está no pacote. Para isso, utiliza-se o método “buscaBinaria” que implementa uma função de busca binária recursiva.

Para imprimir os elementos do Sequence Set utiliza-se o método “imprimir” e para realizar a depuração da classe, utiliza-se o método “depurar”.

2.4.Classes ArvoreBMais e Noh

Primeiramente criaram-se quatro variáveis constantes para estabelecer o máximo de dados, o mínimo de dados, o meio dos dados e o máximo de filhos para cada folha da árvore.

Em seguida, criou-se a classe Noh que tem como atributos privados uma variável booleana denominada “folha”, uma variável do tipo unsigned chamada “num”, uma variável do tipo Pokemon denominada “vetorDeDados” com capacidade de até o tamanho da variável constante “máximoDeDados”, um ponteiro de Noh denominado “filhos” com capacidade máxima de até o tamanho da variável constante “maximoDeFilhos”, uma variável do tipo unsigned denominada “vetorDeDados” com capacidade igual a variável constante “maximoDeFilhos”. A classe Noh possui dois métodos apenas: o método construtor e o método destrutor, que constrói e destrói o Noh, respectivamente.

A Árvore B+ é composta por nós e para a construção da árvore utilizou-se o método “ArvoreBMais” que recebe um arquivo e aponta a raiz para nulo. Já o método destrutor, foi denominado como “~ArvoreBMais” e deleta a raiz.

Para verificar se uma chave está ou não presente em determinado pacote, utiliza-se o método “buscaArquivo” que recebe uma chave e uma posição e retorna um pacote se a chave for encontrada ou “-1” caso a chave não for encontrada.

Para inserir um novo dado na árvore, utiliza-se o método “insere” que recebe um dado do tipo Pokemon e tem como função inseri-lo na árvore. Caso a árvore esteja vazia aloca-se um Noh folha para a raiz e insere o dado na raiz, se a raiz estiver ocupada, verifica-se se a raiz possui espaço e se ela possuir espaço se procura a posição de inserção e se insere o dado na raiz. Caso contrário, com a raiz sem espaços vazios, divide-se a raiz, utilizando o método “insereAux” e insere o elemento na posição e promove o elemento do meio.

O método “insereAux” é um método auxiliar recursivo chamado quando é necessário dividir um Noh, utilizando o método “divideNoh”, que recebe como parâmetro um ponteiro de Noh, um dado do tipo Pokemon e um dado a ser promovido. A função desse método é dividir o Noh cheio, criando um novo Noh, e promover o elemento do meio.

O método “divideNoh” é responsável por fazer a divisão de um Noh, que se encontra cheio, e copiar os elementos da metade superior para o novo Noh.

O método “insereEmNohFolhaNaoCheio” é responsável por procurar a posição para inserção de um novo dado em um Noh que ainda possui espaço e inserir o Noh no local.

O método “insereEmNohIntermediarioNaoCheio” busca a posição de inserção de um dado em um Noh intermediário que ainda possui espaço e realiza a inserção.

Os métodos “atualizaEmOrdem” e “atualizaEmOrdemAux” são responsáveis por fazer a atualização do nível da árvore.

Os métodos “percorreEmOrdem” e “percorreEmOrdemAux” são responsáveis por percorrerem a árvore em ordem crescente, imprimindo o valor do “id” e seu respectivo nível.

Para buscar uma chave na árvore utiliza-se o método “busca” que retorna uma mensagem de erro caso a árvore esteja vazia ou chama o método “buscaAux” passando por parâmetro a raiz e a chave que deseja ser encontrada e retorna o “id”.

Os métodos “imprimir” e “imprimirAux” imprimem a árvore e os dados contidos na árvore em ordem.

3. RESULTADOS

3.1.Sequence Set

A inserção de cem elementos são representados pelas Figuras 3, 4, 5 e 6, respectivamente.

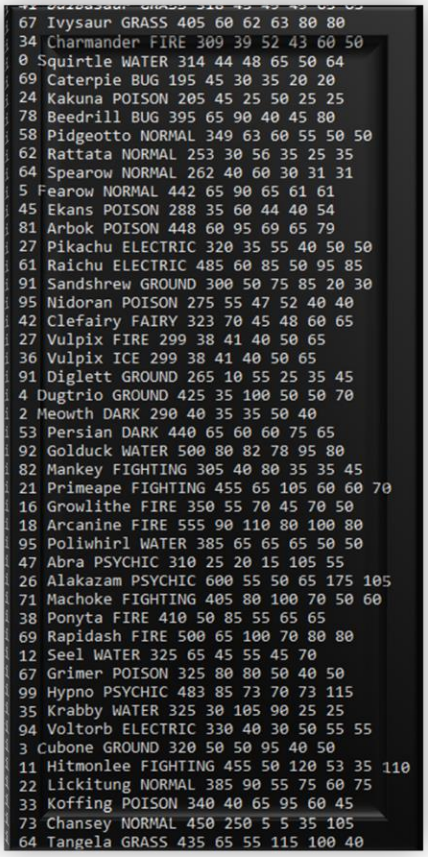

Figura 3 – Inserção de cem elementos no Sequence Set

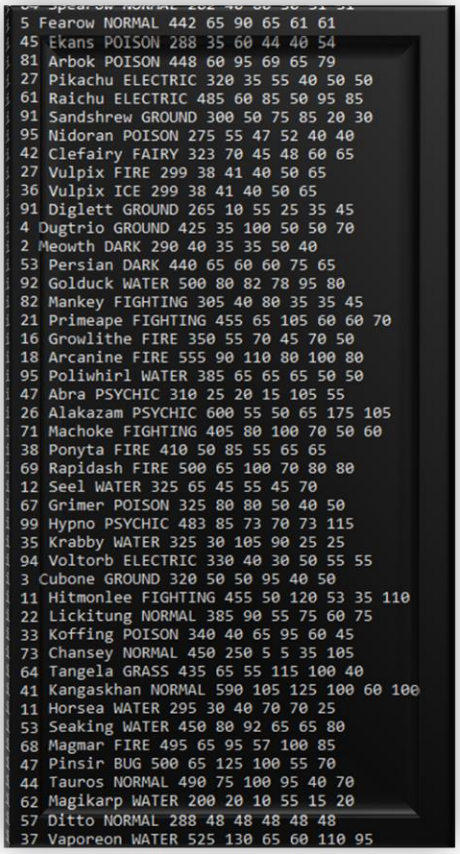

Figura 4 – Inserção de cem elementos no Sequence Set

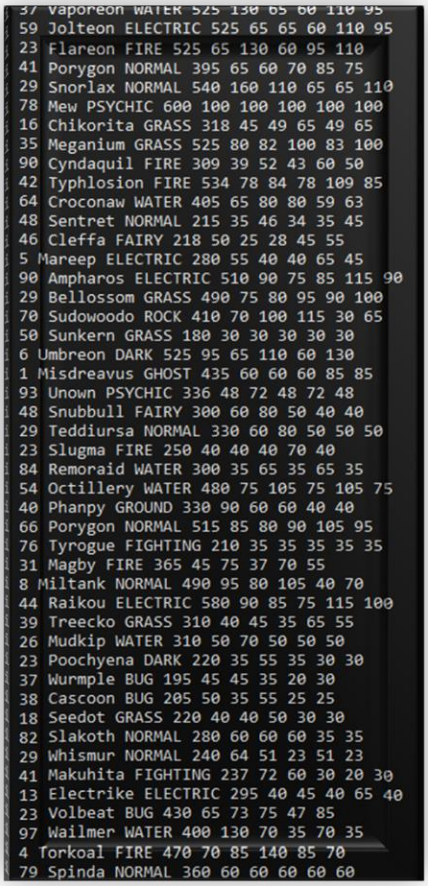

Figura 5 – Inserção de cem elementos no Sequence Set

Figura 6 – Inserção de cem elementos no Sequence Set

A busca de vinte elementos no Sequence Set é representada pela Figura 7.

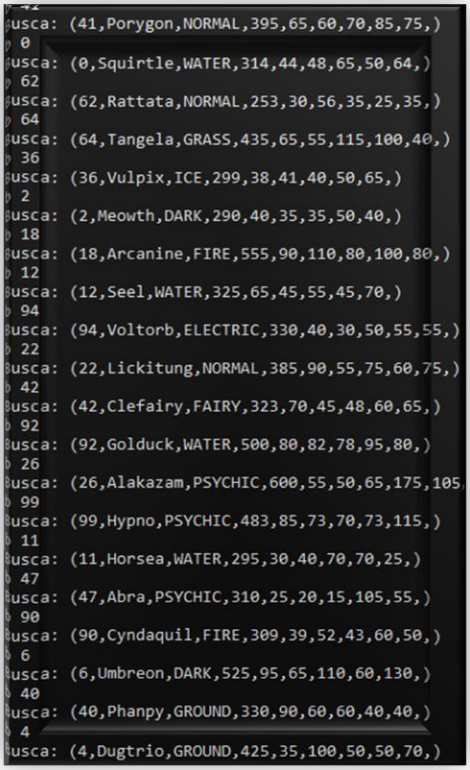

Figura 7 – Busca de vinte elementos no Sequence Set

3.2.Árvore B+

Na Árvore B+ não foi feita a inserção manual, pois o programa está lendo as inserções diretamente do arquivo “imput”. Os valores inseridos na Árvore B+ foram idênticos aos valores inseridos no Sequence Set, assim como os valores de busca. A busca na Árvore B+ é representada pela Figura 8.

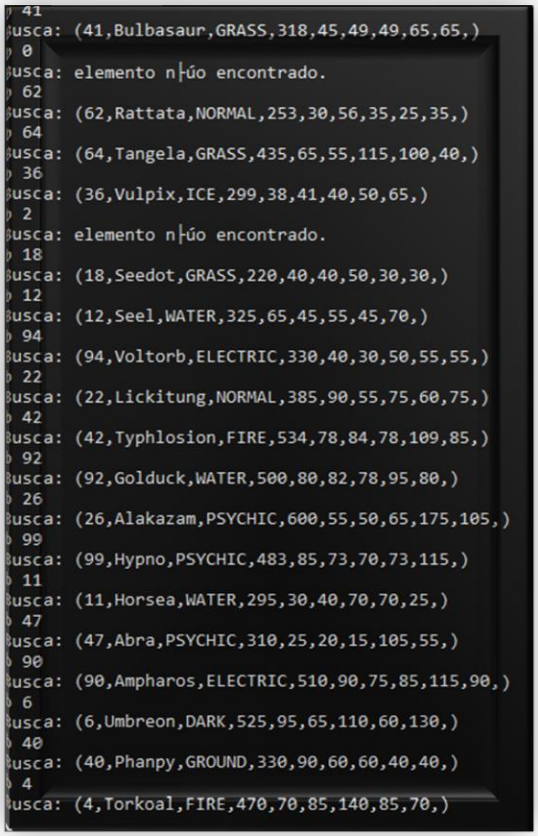

Figura 8 – Busca na Árvore B+

A Figura 9 representa o percorrimento da árvore em ordem.

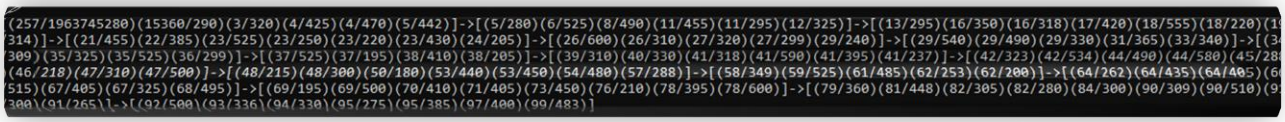

Figura 9 – Percorrimento em ordem na árvore B+

A Figura 10 representa a depuração da árvore.

Já como são muitos dados mostramos apenas alguns , para não ficar muito grande a depuração no relatório

Figura 10 – Depuração da árvore

4. CONCLUSÃO

O objetivo principal do trabalho foi adicionar um índice ao Sequence Set para facilitar a busca de dados, que representa uma Árvore B, e realizar a combinação ao Sequence Set. Diante dos resultados apresentados é possível perceber que esse objetivo foi cumprido e observou-se como vantagens da Árvore B+ a eficiência da busca e da inserção dos valores.

Além disso, foi proposto que o código suportasse a inserção de um dado qualquer armazenado no Sequence Set, a busca de um dado completo a partir da chave, a impressão dos dados do Sequence Set por ordem da chave e a depuração para a verificação da estrutura da Árvore B+. A partir dos resultados apresentados, é possível perceber que todos esses objetivos específicos foram realizados, porém alguns problemas foram encontrados. Como mencionado na metodologia, ao utilizar o atributo "nome" como chave, como havia sido proposto, deparou-se com erros e por isso optou-se por utilizar o atributo "id" como a chave. Além disso, ao implementar a busca, deparou-se com erros em alguns casos executados.

5. REFERÊNCIAS

FERRAZ, N. Inhaúma. **PROGRAMAÇÃO COM ARQUIVOS**, 2003, 1ª edição.

UCHÔA, Joaquim; GREGHI, Juliana; RAMOS, Renato. **Estrutura de Dados Baseados em Listas.** Slides 92-95.

UCHÔA, Joaquim; GREGHI, Juliana. **Árvore B.** Slide 38.