Ordenação em Memória Secundária, com Sequence Set e Árvore B+

[1]Chrystian Arriel Amaral, [2]Débora Rossini Martins Cardoso, [1]Thiago Salles Santos

[1]Turma 10A do curso de Ciência da Computação, Universidade Federal de Lavras, C.P. 3037, 37200–000, Lavras, MG, Brasil.
[2]Turma 14B do curso de Sistemas de Informação, Universidade Federal de Lavras, C.P. 3037, 37200–000, Lavras, MG, Brasil.

16 de agosto de 2020

Aplicando a estrutura Sequence Set, para um conjunto de dados de um banco de dados, é gerado um arquivo binário com estruturas em uma sequência ordenada por duas chaves, sendo facilmente possível expandir a quantidade de dados armazenados, sem perder a ordenação dos dados. E com o uso da Árvore B+ de índices, a busca fica mais eficiente e o processo mais rápido.

1 Introdução

Algoritmos de ordenação são usados em Computação para ordenar/organizar um conjunto de dados de acordo com a necessidade. Existem vários métodos de ordenação, que são escolhidos de acordo com diversos fatores – tamanho do arquivo, custo computacional, dentre outros.

No presente trabalho, o algoritmo de ordenação dos elementos de um banco de dados é o Sequence Set, que ordena os dados e armazena em um arquivo binário. O algoritmo estrutura os dados com base em um cabeçalho ao início do arquivo e blocos (packets), os quais são criados com a necessidade de mais inserção de elementos.

A estrutura promete uma facilidade em inserir mais e mais elementos, sem grande trabalho de realocação e movimentação de todos os elementos para manter a ordenação.

Para auxiliar na busca desses elementos, em meio a uma grande quantidades de blocos, a estrutura Árvore B+ cria uma árvore de índices, criando eficiência de busca dos elementos na sequência do Sequence Set.

2 Sequence Set

A quantidade de dados tratados por um computador, pode ser estrondosa, chegando milhares de dados em um só problema. Logo, para que seja cada vez mais fácil a manipulação desses dados, as estruturas de dados entram em foco. Na realidade da computação existem uma diversidade de estruturas, as quais cada uma tem seus pontos positivos e negativos, levando a aplicações melhores em alguns problemas e em outros nem tanto. Com isso, no contexto de dados em memória secundária, temos a estrutura Sequence Set como candidata a solução de problemas neste contexto.

O Sequence Set é, de certa forma, uma derivação dos vetores expansíveis, porém com à aplicação em arquivos binários. Igualmente nós vetores expansíveis, temos no Sequence Set, pacotes interligados em um esquema de lista, onde o pacote anterior está interligado ao pacote sucessor a ele. O que define essa ordem são os elementos presentes dentro dos pacotes, os quais estão ordenados com base em uma chave especifica (a chave é definida de acordo com o problema).

Os pacotes possuem um pequeno cabeçalho, o qual está presente qual o próximo pacote, qual a quantidade de elementos esse pacote possui e o vetor com esses elementos. Também é possível ter qual o pacote anterior, no cabeçalho.

Além dos pacotes, o Sequence Set possui o cabeçalho, o qual orienta sobre qual é o primeiro pacote, a quantidade de pacotes e o próximo pacote a ser criado, ou que está vazio, de acordo com o caso. É possível ter, também, no cabeçalho, qual o último pacote da ordenação, tendo um acesso direto ao último elemento.

A ordenação gerada pela estrutura, com fácil redimencionamento, é o verdadeiro valor do Sequence Set, pois permite ter dados ordenados em arquivos binários de uma forma mais eficiente e menos trabalhosa com a possibilidade de inserir muitos outros elementos. Vale lembrar que essa ordenação é em cima da estrutura, em suas ligações entre pacotes e seu cabeçário, pois o arquivo em si pode estar desordenado. Mas, pelo o poder da estrutura, fica desnecessário locomover diversos elementos de posição, por exemplo em uma inserção, para obter elementos ordenados.

3 Árvore B+

Dentre as estruturas de dados, existe um subconjunto delas que derivam da ideia de uma árvore, onde temos uma raiz e seus galhos consecutivos, chegando, no final, nas folhas. No contexto de estruturas, chamamos cada galho e folha de "noh", sendo que este também é uma raiz de uma subárvore consecutiva da raiz principal. Vale lembrar que o crescimento desta árvore é convencionado para baixo.

As características principais dessas árvores é o fato de cada noh ser menor ou maior que sua raiz, sendo que,

normalmente, nohs com valor menor, ficam a esquerda da raiz e nohs com valor maior, ficam a direita. Sendo o maior exemplo disso a Árvore Busca Binária.

No entanto, a Árvore BB não é eficiente para todos os problemas, pois ela pode crescer de forma desbalanceada. Com isso temos as variações que resolvem desbalanceamentos de nohs, na árvore. Como exemplo, há as Árvores AVL e Rubro Negra.

Mas isso não resolve todos os contextos, pois com uma quantidade muito grande de dados, a árvore tende a ficar ineficiente. Isso pois vai haver muitos nohs a serem linkados, ou seja, muitas subárvores a serem criadas e verificadas.

Com isso surge o contexto da Árvore 2-3, 2-3-4 e B, pois lidam com muitos dados. Suas características diferenciais são que cada noh possui mais de 2 "filhos", o que gera bem menos nohs a cada nível da árvore. Focando na Árvore B, temos outra característica diferencial, a qual é de trabalhar com uso da memória secundária.

A Árvore B possui variações, sendo elas as Árvores B+ e B*. Focando em na Árvore B+, temos a característica dela ter todos os seus dados nas folhas, subindo para outros somente aqueles que ligam uma folha a outra, mantendo a característica binária de busca. De forma técnica, temos que a árvore somente carrega os índices dos dados brutos, os quais estão em blocos na memória secundária. Com isso, é possível manter a estrutura da árvore em memória primária, armazenando os índices para busca e deixando os dados brutos em blocos na memória secundária - o que seria o Sequence Set.

Dessa forma, a Árvore B+ traz o poder de eficiência em busca, para tratamento de dados ordenados em memória secundária. O que, por consequência, gera a possibilidade de ligar a estrutura Sequence Set com a Árvore B+, ganhando eficiência no sistema.

4 Desenvolvimento do Sistema

Para o uso eficiente de arquivos binários e ordenar uma sequência de dados, os quais são de um banco de dados de captura de pacotes de rede, foi desenvolvido um sistema que utiliza a estrutura Sequence Set. E, para maior eficiência de busca e, consequentemente, de inserção, também foi implementado uma estrutura de Árvore B+, a qual armazena as chaves e índices de bloco, do primeiro elemento de cada bloco, a partir do segundo.

O sistema faz a leitura do arquivo "captura_pacotes.csv", onde possui uma imensa quantidade de pacotes de redes, sendo cada linha um pacote. Cada pacote possui os campos indice, tempo, origem, destino, protocolo, tamanho e informação. Logo, dado a diversidade de campos, é possível ordenar em função de diferentes chaves e também por mais de uma chave, em caso de chaves iguais. Para esse sistema, foi utilizado a chave primária destino (destination) e a chave secundária índice.

A cada pacote lido do arquivo "captura_pacotes.csv", é transferido para estrutura Sequence Set, a qual vai inserir no arquivo binário "arq_comDados.bin"em um bloco, sendo o bloco determinado pela ordenação dos elementos (pacotes de rede) já presentes no arquivo. Assim, ao final da leitura de todos os pacotes de rede do banco de dados, o arquivo binário vai ter esses dados em ordem, com base na estrutura Sequence Set.

Durante o processo de inserção dos pacotes de rede, no Sequence Set, a cada divisão dos elementos de um bloco cheio, com um novo bloco, a Árvore B+ recebe o primeiro elemento do novo bloco, atualizando assim sua estrutura de busca pelos índices de blocos. Esse processo recebe o nome de "promovê", pois realiza uma "elevação"de nível do elemento, na árvore.

Foi realizado duas versões do sistema, onde uma versão possui sua árvore totalmente em memória primária, ganhando eficiência em processamento. A árvore trabalha com sequências de nohs, em cada galho, interligados por ponteiros, o que é uma ideia generalizada de lista, possibilitando menos uso da memória principal.

A outra versão possui sua árvore totalmente em memória secundária, o que reduz o uso de memória primária e garante que a árvore permaneça salva, mesmo após fechar o programa.

Cada versão possui três bibliotecas (.h), e um arquivo principal (.cpp), os quais são: "leCSV.h", "sequence_set.h", "arvoreBplus.h"e "main.cpp". A biblioteca "time.h", foi utilizada com a intenção de exibir o tempo de busca e leitura. Não foi utilizado Makefile por inexperiência dos membros do grupo, com o método.

- leCSV.h: A biblioteca possui a definição da class "leCSV", a qual faz leitura do arquivo "captura_pacotes.csv", estruturando em uma struct "pacote"cada informação em seu campo específico na struct. Ao final a class insere o elemento (pacote) no Sequence Set.

  O algoritmo é totalmente em cima do disponibilizado para o projeto.

- sequence_set.h: É a biblioteca que está todas definições e funções da class "SequenceSet". É ali que está todo o trabalho da estrutura Sequence Set, para ordenar e tratar o arquivo binário com os dados. É onde está instanciado o objeto "arvoreBplus".

- "arvoreBplus.h": É a biblioteca que está todas definições e funções da class "arvoreBplus". É ali que está todo o trabalho da estrutura Arvore B+, para buscar os índices dos blocos, pelas chaves. Também é onde esta a definicão do processo de estruturação da árvore. Como há duas versões, as funções e metodos são diferentes para cada "arvoreBplus.h".

- main.cpp: É o primeiro contato com o usuário, tendo os menus e opções iniciais de tratamento do arquivo. É onde está instanciado os objetos "SequenceSet"e "leCSV".

Para buscar uma melhor eficiência em uso de memória secundária, foi pensado na inserção com e sem a função "empresta", a qual possibilita passagem de elementos para blocos anteriores ou posteriores, em caso de bloco atual cheio.

4.1 Leitura do banco de dados

A leitura do banco de dados é realizada pela class leCSV, onde ela recebe, em seu construtor, os parâmetros: o objeto SequenceSet que vai ser utilizado, o inicio do trecho para leitura e o final do trecho para leitura, sendo o trecho com base no índice de cada linha dentro do csv. Também recebe uma booleana "caso", a qual define a utilização ou não da função "empresta", na inserção, que deve ser passada a função de inserção da class SequenceSet. Ao final de cada leitura de um elemento (pacote), do csv, é chamado a função para inserção do SequenceSet, passando como parâmetro o pacote setado e o "caso". O processo se repete até o fim do trecho.

4.2 A composição da class SequenceSet

A class SequenceSet possui funções para inserir elementos, buscar elementos, alterar elementos, setar novos packets e cabeçalho, realizar comparação de elementos, de realizar troca de elementos entre packets (empresta), ler e escrever no arquivo binário "arq_comDados.bin"e de realizar divisões de packets (criando um novo e dividindo os elementos com o antigo). Ela também contém como atributo um objeto "arvoreBplus", uma variável "MAXelements", do tipo unsigned, um objeto "Cabecalho"e duas structs, "Packet"e "Cabecalho".

Os atributos:

- Packet: É a estrutura que representa o bloco do sequence set, onde tem os campos: quantidade de elementos no vetor, próximo bloco, bloco anterior e o vetor de elementos (pacotes).
- Cabecalho: É a estrutura que representa o cabeçalho do sequence set, onde tem os campos: quantidade de packets, quantidade de elementos, próximo packet a ser criado (ou vazio) para inserção, primeiro packet da sequência e o último packet da sequência.
- MAXelements: Mantém o valor total de elementos por packet.
- cabecalho: Utilizada para receber o cabeçalho do arquivo em diferentes momentos do do código.

Cada função e suas propriedades:

- criaCabecalho: Seta as informações iniciais do cabeçalho, atribuindo a um objeto "Cabecalho"passado como parâmetro.
- criaPacket: Seta as informações de um packet, com base nos parâmetros passados, sendo eles: ID do packet, packet anterior, o packet posterior.
- insereNoVetor: É utilizada para inserir um novo elemento no vetor interno de um packet, realizando um procedimento de busca pela posição correta, com base nas chaves de ordenação.
- dividePacket: Quando um packet está cheio (e não houve "empresta"), é criado um novo packet, ou utilizado um já vazio no arquivo, para inserir a metade posterior dos elementos do packet cheio, liberando espaço para a inserção.
- empresta: realiza a passagem de um elemento para o packet posterior ou anterior.
- insereNoPacket: Realiza a análise das possibilidades de inserção no packet passado como parâmetro.

  Em caso do packet não cheio, é realizado a inserção direta do elemento no packet (chama a função insereNoVetor).

  Em caso de packet cheio, com a utilização do empresta, é analisado a possibilidade de liberar espaço passando para o packet anterior ou posterior. É feito chamada da função de atualização da árvore, passando como parametro o antigo elemento que estava na árvore e também o novo que iria substituí-lo. Caso não for possível realizar o "empresta", por packet anterior e posterior cheios, é realizado a divisão do packet.

  Já sem a utilização do empresta, com o packet cheio, é feito diretamente a divisão do packet.

  Sempre que ocorre divisão de um packet, é passado para a função "promove"da class arvoreBplus, o primeiro elemento do novo packet criado, para que a árvore fique atualizada.
- compara: Realiza a comparação entre elementos, com base nas chaves de ordenação (destino e índice). Em caso de ordenação, é feito comparação com as duas chaves. Já em outros casos, é feito comparação somente da primeira chave, possibilitando utilizar a função em outros momentos.
- lePacketArq e escrevePacketArq: Abre o arquivo e escreve, ou lê, um packet na posição passada como parâmetro.
- leCabecalhoArq e escreveCabecalhoArq: Abre o arquivo e escreve, ou lê, o cabeçalho no início do arquivo.
- comparaIgualdade: Igualmente ao "compara", realiza comparação entre elementos, no entanto com base em um campo específico do pacote.

- buscaElementos: Com base no campo, a função busca os elementos com o mesmo campo, criando um arquivo txt com o resultado. Em caso de chave primária (destino), nesta função, a busca é mais eficiente, devido a ordenação e a árvore. As demais devem percorrer todo o arquivo.

- buscaEficiente: Recebendo como parâmetro as chaves e utilizando a árvore de índices, é retornado o packet, por referência, e a posição no packet do elemento em questão.

- pegaElementoPorIndice: Ela faz a verificação de índices iguais, evitando possibilidade de dois elementos totalmente iguais.

- SequenceSet (construtor): Ele verifica a existência de arquivo, com os dados, pré existente.

  Em caso de árvore em memória, existindo um arquivo, é chamado a função "criaArvore". Já na versão da árvore em arquivo, caso exista arquivo de dados, é verificada a existência de um arquivo com a árvore. Caso não exista, é criado a árvore com a função "criaArvore".

- setInicial: Cria o arquivo binário, o cabeçalho e o primeiro packet, o que deixa pronto para inserção de elementos. Chama a função setInicial da árvore, em caso de árvore em arquivo.

- insereElemento: Insere um elemento passado como parâmetro, na sequência, utilizando outras funções para isso. Chama a busca da árvore, a qual devolve o índice do packet a ser inserido.

- criaTxt: Cria um arquivo txt com o cabeçalho do arquivo e todos os packets e seus elementos, tornando possível visualizar o resultado dentro do arquivo binário.

- atualiza: Recebe do usuário qual o índice a ser buscado e atualizado. Utiliza a função "buscaEficiente"para buscar. Possui interface para o usuário definir o que será atualizado e receber o novo dado.

- busca: Recebe do usuário qual o campo e seu valor, a ser buscado na sequência;

- insere: Cria um novo elemento (pacote), com as informações passadas pelo usuário, para que seja inserido na sequência.

- CriaNovoTxt: possibilita criar um novo txt da árvore ou do sequence set.

4.3 Composição da class arvoreBplus (em memória)

A class arvoreBplus possui funções para promover um elemento, atualizar os indice e as chaves de um elemento, e também buscar um packet do sequence set, que contenha ou possa receber um elemento. Porém para realizar suas operações necessita de classes auxiliares, que são as class galho, e a class noh.

4.3.1 A classe Noh

A class noh, possui como funcionalidade ser o objeto q irá representar um pacote do Packet na árvore. seus objetos armazenam as chaves para as comparações, os índice dos packet ao qual elas representam, além dos ponteiros dos galhos, que se encontram abaixo. Ela é amiga das classes galho, arvoreBplus.

Os atributos:

- destino: Armazena a chave primaria.

- indice: Armazena a chave secundaria.

- nohAnterior: Ponteiro de noh, que aponta para o noh que se encontra atrás dele no galho.

- nohPosterior: Ponteiro de noh, que aponta para o noh que se encontra a frente dele no galho.

- galhoAnterior: Ponteiro de galho, que aponta para o galho abaixo dele que possui elementos inferiores, as suas chaves destino e índice.

- galhoPosterior: Ponteiro de galho, que aponta para o galho abaixo dele que possui elementos superiores ou iguais, as suas chaves destino e índice.

- anteriorPacket: Armazena o índice do packet, que possui valores inferiores, as suas chaves destino e índice.

- inidicePacket: Armazena o índice do packet, que possui valores superiores ou iguais, as suas chaves destino e índice.

Cada função e suas propriedades:

- noh(construtor): Recebe um pacote, e os índice dos packets anteriores e posteriores, copia as chaves primaria e secundaria, setar os índices, e aterra todos os ponteiros.

- atualizaIndiceAnteriorNoh: Sobrescreve o índice anterior, pelo novo índice recebido como parâmetro pela a função.

- atualizaPacoteNoh: Sobrescreve o destino e o índice, pelo os dados destino e índice do pacote recebido como parâmetro.

4.3.2 A classe Galho

A class galho, atua como uma lista de nohs. Ela é amiga da classe arvorebplus.

Os atributos:

- primeiroNoh: Ponteiro de noh, que aponta para o noh com menor chave entre todos os nohs que ele administra.
- pai: Ponteiro de galho, que aponta para um galho pai.
- limiteNohs: Armazena a quantidade máxima de nohs q o galho pode ter.
- quantElements: Armazena a quantidade de nohs que ele tem.

Cada função e suas propriedades:

- galho(construtor): setar os ponteiros como NULL, define o limiteNohs como 80, e a quantElements como 0.
- comparaNohs: Recebe por parâmetro e compara dois nohs a partir de suas duas chaves e retorna um inteiro, que segue a seguinte regra: retorna valor menor que zero quando segundo parâmetro é maior, retorna igual a zero quando ambos os parâmetros possuem valor igual, retorna maior q zero quando primeiro parâmetro é maior.
- insereNoGalho: Recebe o noh que será inserido no galho, e chama o processamento de divide e promoveNoh do galho, quando um galho atinge a quantidade limite de nohs estabelecida, ele retornará o pai quando utilizado, do caso contrário ele retorna NULL.
- divideGalho: Divide o galho que chamou a função, e retorna um galho com a metade direita do valores do galho de origem.
- promoveNoh: Promove o primeiro noh de um galho para o seu pai, caso não exista pai, ele cria um, e retorna o pai do galho.
- atualizaPonteiros: Atualiza o ponteiro de galhoAnterior do noh posterior do pai que acabou de receber um elemento, pois foi criado um galho entre seus ponteiros de anterior e posterior galho, caso esse noh posterior ao galho exista também.

4.3.3 A classe ArvoreB+

A class arvoreBplus, ministra a classe galho, e defini a lógica de uma árvore B+.

Os atributos:

- raiz: Ponteiro de galho, para indiciar qual galho representa a raiz.

Cada função e suas propriedades:

- promove: Recebe um pacote vindo do SequecenSet, após a ocorrência de uma divisão de um Packet, ele cria um noh, chama o buscaGalho, para saber em que galho inserir, depois de descobrir insere no galho referente, e analisa o retorno para saber se existe uma nova raiz.
- buscaAux: Método recursivo que irá busca dentre os galhos, em qual packet o elemento deverá ser inserido ou se encontra.
- busca: Método que irá preparar os parâmetros de busca para o buscaAux e retornar o valo obtido por ele.
- atualizaAux: Método recursivo que irá retornar o noh com as chaves requisitadas.
- atualizaPacoteArvore: Atualiza um noh, trocando os valores das chaves, pelos novos valores correspondentes, ele é chamado pelo SequenceSet quando ocorre algum tipo de empresta seja anterior ou posterior, pois como ocorre a alteração do primeiro elemento de um Packet por causa do empresta ele deverá ser atualizado na árvore, para descobrir o noh q irá atualizar ele chama o atualizaAux.
- atualizaIndiceArvore: Atualiza um noh, trocando os valores dos índices de packets anteriores, pelo novo packet. ele é chamado pelo SequenceSet quando ocorre uma divisão e o packet gerado por essa divisão irá ocupar no SequenceSet uma posição entre dois packets, então se deve atualizar o anteriorPacket do noh que representa o primeiro valor do packet posterior a esse que recebeu a metade direita dos dados.
- buscaGalho: Método recursivo que irá retornar o galho mais baixo em nível de árvore, na qual o noh será inserido ou encontrado.

4.4 Composição da class arvoreBplus (em arquivo)

A class arvoreBplus possui funções para promover um elemento, atualizar as chaves de um elemento e buscar um packet do sequence set, que contenha ou possa receber um elemento. Também possui funções para buscar chaves dentro da árvore, buscar chaves dentro de um noh específico e para comparar chaves. Também há funções para escrever nohs e cabeçalho no arquivo "arq_comArvore.bin", funções para setar novos nohs e para dividir nohs cheios. Ela também contém como atributo um objeto "cabecalhoArvore", uma variável "MAXElements", do tipo int, uma struct "cabecalhoArvore", uma struct "Chaves"e uma struct "noh".

Os atributos:

- "cabecalhoArvore": É a estrutura que representa o cabeçalho do arquivo com a árvore, onde tem os campos: quantidade de nohs, quantidade de chaves, próximo noh a ser criado para inserção e a raiz da árvore.
- "Chaves": É a estrutura que junta as duas chaves de ordenação de elementos do sequence set.
- "noh": É a estrutura que representa o noh, onde fica todas as chaves e os índices para o próximo noh ou o packet do sequence set, em caso de nohs folhas. Possui os campos: chaves, sendo um vetor de structs "Chaves", um para os índices, um indicador do pai, um indicador de folha, ou não folha, um id (índice do noh) e a quantidades de chaves no noh.
- "MAXElements": Mantém o valor total de chaves por noh.
- cabecalho: Utilizada para receber o cabeçalho do arquivo em diferentes momentos do código.

Cada função e suas propriedades

- criaCabecalho: Igualmente no sequence set, seta as informações iniciais do cabeçalho, atribuindo a um objeto "cabecalhoArvore"passado como parâmetro.
- criaNoh: Seta as informações do noh, com base nos parâmetros passados, sendo eles: ID do noh, noh pai e o noh a ser atribuído.
- leNohArq e escreveNohArq: Escreve, ou lê, o noh no arquivo com a árvore.
- leCabecalhoArq e escreveCabecalhoArq: Escreve, ou lê, o cabeçalho no início do arquivo com a arvore.
- criaChave: seta as chaves, passadas como "pacote", em uma struct "Chaves".
- comparaChaves: Compara as chaves de duas structs "Chaves". Retorna "1"para caso onde a chave A, primeiro parâmetro, é maior que a chave B, segundo parâmetro. Caso A seja menor, retorna -1"e igualdade retorna "0".
- buscaNoNoh: Faz busca de uma chave dentro de um noh, retornando a posição no vetor ou -1", em caso de não encontrar a chave.
- buscaNaArvore: Realiza busca de uma chave pela árvore. Retorna a posição dentro do vetor que ela estiver e, por referência, o noh que ela está. Caso não encontre, retorna -1";
- insereNoNoh: Insere uma chave, na posição correta, dentro de um noh passado como parâmetro. Também insere no vetor de índices do noh, os vetores anterior e posterior, passados como parâmetros junto com a chave.
- atualizaChave: Procura uma chave já existente na árvore e à atualiza, colocando novas chaves. Isso ocorre por causa de processo de "empresta"realizado no sequence set.
- divideNoh: Realiza a divisão de um noh, passando metade dos elementos para um novo noh e subindo a primeira chave do novo noh, para o pai de ambos. Em caso de não existir pai, é criado um novo, mantendo uma raiz da subárvore.
- buscaPacket: Realiza a busca de um packet para um elemento do sequence set, utilizando buscaNaArvore para passar pela árvore e chegar na folha correspondente, na chave correspondente e passar o índice anterior ou posterior.
- promove: Recebe um elemento do sequence set a ser "promovido", transformando em struct chave e inserindo no noh correto, de acordo com o buscaNaArvore.
- criaTxt: Cria um arquivo txt com os nohs e o cabeçalho da árvore, possibilitando visualizar brevemente a árvore. Não organiza pois grava na ordem que está no arquivo da árvore.
- testeDeArquivo: Verifica a existência de um arquivo de árvore, retornando "true"se existir e caso contrário, "false".
- setInicial: Igualmente o sequence set, cria um arquivo binario para a árvore, um cabeçalho e um noh inicial, sendo ele setado como raiz no cabeçalho.

4.5 Fluxo da Inserção

Para a inserção na sequência, gerada pelo Sequence Set, é primeiramente passado um novo elemento a função "insereElemento". Ocorre essa passagem tanto pelo leCSV, quando lê o banco de dados, tanto pela função "insere", depois de setar um novo pacote pelo usuário. Também é passado o tipo de inserção, com ou sem "empresta".

Recebendo o pacote e o tipo de inserção, a função seta o cabeçalho do arquivo no atributo "cabecalho". Assim sabemos onde está o começo e o fim da sequência, no arquivo. Depois é realizada a busca do packet ideal, pela árvore de índices, que retorna o índice do packet. Assim, é possível ler o packet no arquivo e passar para a função "insereNoPacket"o packet, o elemento a ser inserido e o tipo de inserção.

Nesse momento é analisado se o packet está cheio ou não. Em caso de não estar, chama insereNoVetor, que busca a posição no vetor a ser inserido o elemento, deslocando as demais já existentes.

Porém, estando cheio o packet, analisa o tipo de inserção. Em caso de inserção com "empresta", pega o packet anterior e verifica se está cheio, chamando "empresta"em caso de não estar. Nesse caso, atualiza a árvore,

passando o antigo elemento e o novo elemento. O mesmo ocorre para o packet posterior, caso o anterior estiver cheio, também atualizando a árvore. Em último caso, estando cheio os dois, realiza divisão do packet, chamando a função "dividePacket", a qual cria um novo, para dividir os elementos do packet cheio, recebendo sua metade posterior. Nesse caso, a função "promove"da árvore recebe o primeiro elemento do novo packet, mantendo os índices e chaves da árvore atualizados.

O caso de não usar "empresta", chama diretamente a "dividePacket", em caso de packet cheio, também realizando "promove".

4.6 A main e o Menu de Tratamento

No arquivo principal main, temos os menus e os objetos SequenceSet e leCSV instanciados. Depois de instanciar o SequenceSet, e receber na variavel "caso"a existência de arquivo já existente, a função "menuInicial"pede ao usuário pergunta ao usuário se deve ou não realizar uma nova leitura. Caso não for realizada, vai direto para o menu de tratamento. Não existindo arquivo, realiza diretamente a leitura.

Para realizar a leitura, a função leituraDoCSV pergunta ao usuário se deseja utilizar "empresta"na inserção e qual o trecho a ser lido, onde o trecho padrão é dos índices 519091 a 605605.

Entrando na seção do Menu de Tratamento, pela função "menuTratamento", pega a opção de tratamento (inserir, buscar, atualizar, criar novo Txt de Sequence Set, finalizar, realizar nova leitura e, em caso de árvore em arquivo, criar novo Txt da árvore) da sequência e realiza o caso correspondente.

Finalizando o programa, a função "menuFinaliza"exibe informações do grupo do projeto.

4.6.1 Inserir

Chama a função "inserir"dá class SequenceSet, a qual cria um novo elemento, sendo as informações passadas pelo usuário e depois realizando o processo de inserção com a função "insereElemento".

4.6.2 Buscar

Chama a função "busca"da class SequenceSet, a qual pede o campo e o valor a ser buscado, que depois, pela função "buscaElemento", acha todos os elementos e escreve seu packet e sua posição no vetor, em um arquivo txt. Em caso de busca eficiente, utiliza a função "buscaEficiente"e exibe o elemento.

4.6.3 Atualizar

Chama a função "atualiza"da class SequenceSet, a qual pega a chave primaria e secundaria do elemento desejado e realizar alteração do elemento, escrevendo seu packet novamente no arquivo depois.

5 Resultados e Discussão

O sistema tem grande eficiência em organizar e tratar os dados, tornando possível redimensionamento da sequencia sem grande trabalho de locomover cada elemento, por todo o arquivo.

Foi realizado teste com o trecho padrão e um trecho menor. Os testes foram realizados na versão em arquivo e na versão em memoria. Também, para fins de comparação, foi realizado teste com uma versão que utiliza somente o Sequence Set, sem uma arvore de índices.

Com a utilização da função "empresta", na inserção, foi possível analisar uma grande redução no número de packets criados, ganhando eficiência em memória secundária. No entanto, o número de condições a serem realizadas aumenta, o que pode interferir no tempo de processamento.

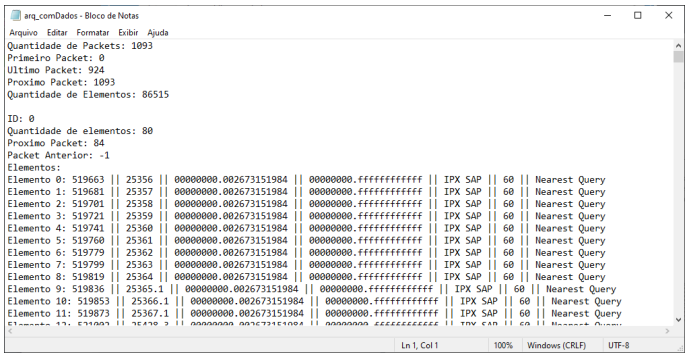

Figura 1: Print do resultado da inserção, do trecho padrão, com a função "empresta"

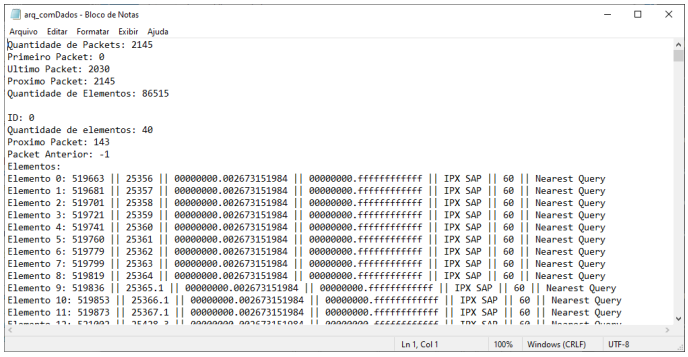

Figura 2: Print do resultado da inserção, do trecho padrão, sem a função "empresta"

Em questão de eficiência de tempo, o sistema com a Árvore B+ conseguiu reduzir, consideravelmente, o tempo de inserção e busca.

Na tabela da figura 3, é possível ver o tempo, em segundos, de cada versão, com e sem empresta.

A versão em memória tem o melhor tempo, porém a versão em arquivo não perde muito, relativamente. O tempo a questão do tempo pode estar relacionada com

		Árvore B+ em Memória	Árvore B+ em Arquivo	Sequence Set
Trecho padrão	Com Empresta	544.344 segundos	752.027 segundos	7249.81
	Sem Empreta	491.781 segundos	562.459 segundos	-
1 a 10000	Com Empresta	58.945 segundos	84.003 segundos	159.714 segundos
	Sem Empresta	51.89 segundos	65.556 segundos	211.065 segundos

Figura 3: Tabela com o tempo de inserção das versões, com e sem empresta.

a implementação do sequence set, que pode ter pesado o processamento com muitas condições. No entanto é visível a diferença com e sem a árvore de índices.

6 Conclusão

Para se ordenar e tratar uma grande quantidade de dados, com diferentes características cada, a estrutura Sequence Set é de grande utilidade, pois torna possível inserir cada vez mais elementos, sem muito problema de realocação. Em banco de dados muito grandes, os quais o processamento direto em memória primária fica inviável, a estrutura se torna uma opção.

Com o auxílio da Árvore B+, o processo fica além de otimizado em realocação, eficiente em tempo de processamento, pois árvore agiliza visivelmente o tempo.

E como visto nos resultados, uma otimização da inserção, utilizando o "empresta", é possível reduzir mais o espaço gasto em memória secundária, sem perder o poder de ordenação da estrutura.

Com otimização da implementação da árvore, e do sequence set, existe possibilidade de um tempo ainda melhor de inserção, mas com a implementação atual já é possível ver o poder de busca que a árvore fornece, reduzindo o tempo.

Referências

[1] DEITEL, Harvey M.; DEITEL, Paul J. C++: como programar. 5. ed. São Paulo, SP: Pearson Prentice Hall, 2006. xlii, 1163 p.

[2] KOFFMAN, Elliot B. Objetos, abstração, estrutura de dados e projeto usando C++. Rio de Janeiro LTC 2008.

[3] Slides "*Estruturas baseadas em Listas*", do Prof. Joaquim Uchôa, Profa. Juliana Greghi e Prof. Renato Ramos, disponibilizado para a disciplina GCC216 do Departamento de Ciência da Computação da Universidade Federal de Lavras. **Acesso em 20 de julho de 2020**.

[4] Slides "*Árvore B*", do Prof. Joaquim Uchôa e Profa. Juliana Greghi, disponibilizado para a disciplina GCC216 do Departamento de Ciência da Computação da Universidade Federal de Lavras. **Acesso em 15 de agosto de 2020**.

[5] Video-aula "*Estruturas de Dados - Aula 10 - Parte 3 - Sequence Sets* ", do Prof. Joaquim Uchôa, disponibilizado para a disciplica GCC216 do Departamento de Ciência da Computação da Universidade Federal de Lavras. **Acesso em 20 de julho de 2020**.

Endereço eletrônico: https://www.youtube.com/watch?v=5KGM $PLY_X MpiC9C3Yn - T5Rg0f0te7X8aVUOO0index = 27t = 0s$