



**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.

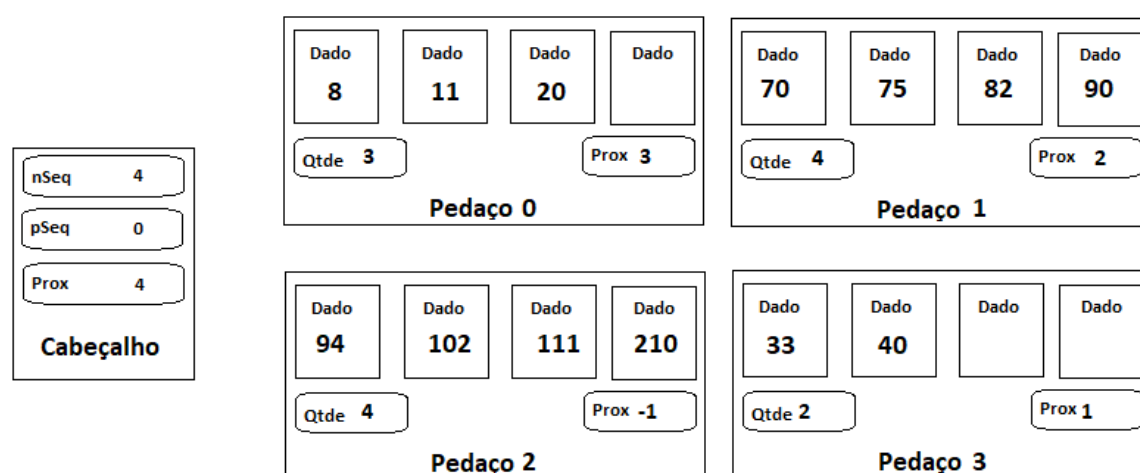


Figura 1 – Sequence Set

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+.

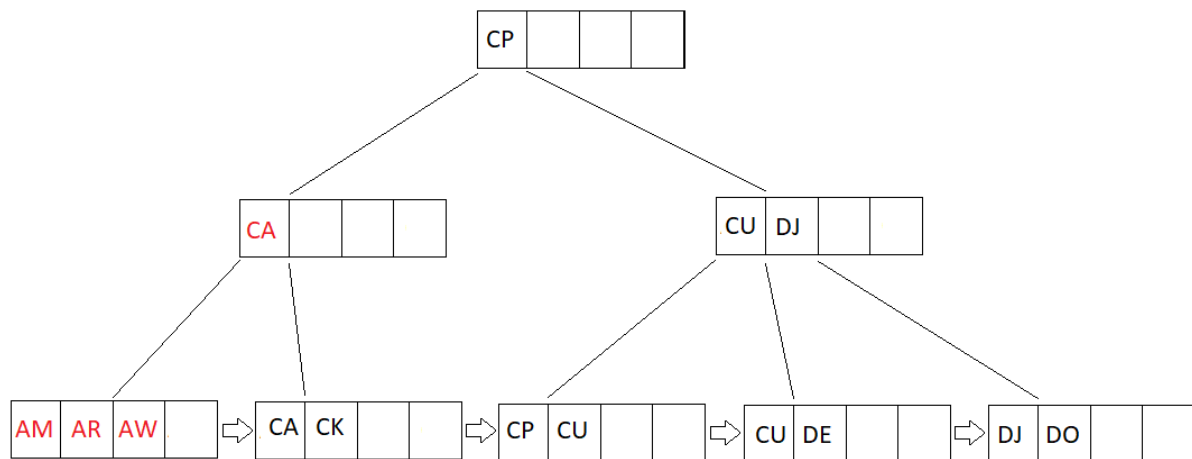


Figura 2 – Árvore B+

Fonte: do autor

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

1. A raiz tem 0,2 ou entre  $\lceil EM/2 \rceil$  e M filhos;
2. Todas as páginas, com exceção da raiz e das páginas terminais, têm no mínimo  $\lceil EM/2 \rceil$  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 percorrermos 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.

```
67 Ivysaur GRASS 405 60 62 63 80 80
34 Charmander FIRE 309 39 52 43 60 50
0 Squirtle WATER 314 44 48 65 50 64
69 Caterpie BUG 195 45 30 35 20 20
24 Kakuna POISON 205 45 25 50 25 25
78 Beedrill BUG 395 65 90 40 45 80
58 Pidgeotto NORMAL 349 63 60 55 50 50
62 Rattata NORMAL 253 30 56 35 25 35
64 Spearow NORMAL 262 40 60 30 31 31
5 Fearow NORMAL 442 65 90 65 61 61
45 Ekans POISON 288 35 60 44 40 54
81 Arbok POISON 448 60 95 69 65 79
27 Pikachu ELECTRIC 320 35 55 40 50 50
61 Raichu ELECTRIC 485 60 85 50 95 85
91 Sandshrew GROUND 300 50 75 85 20 30
95 Nidoran POISON 275 55 47 52 40 40
42 Clefairy FAIRY 323 70 45 48 60 65
27 Vulpix FIRE 299 38 41 40 50 65
36 Vulpix ICE 299 38 41 40 50 65
91 Diglett GROUND 265 10 55 25 35 45
4 Dugtrio GROUND 425 35 100 50 50 70
2 Meowth DARK 290 40 35 35 50 40
53 Persian DARK 440 65 60 60 75 65
92 Golduck WATER 500 80 82 78 95 80
82 Mankey FIGHTING 305 40 80 35 35 45
21 Primeape FIGHTING 455 65 105 60 60 70
16 Growlthe FIRE 350 55 70 45 70 50
18 Arcanine FIRE 555 90 110 80 100 80
95 Poliwhirl WATER 385 65 65 65 50 50
47 Abra PSYCHIC 310 25 20 15 105 55
26 Alakazam PSYCHIC 600 55 50 65 175 105
71 Machoke FIGHTING 405 80 100 70 50 60
38 Ponyta FIRE 410 50 85 55 65 65
69 Rapidash FIRE 500 65 100 70 80 80
12 Seel WATER 325 65 45 55 45 70
67 Grimer POISON 325 80 80 50 40 50
99 Hypno PSYCHIC 483 85 73 70 73 115
35 Krabby WATER 325 30 105 90 25 25
94 Voltorb ELECTRIC 330 40 30 50 55 55
3 Cubone GROUND 320 50 50 95 40 50
11 Hitmonlee FIGHTING 455 50 120 53 35 110
22 Lickitung NORMAL 385 90 55 75 60 75
33 Koffing POISON 340 40 65 95 60 45
73 Chansey NORMAL 450 250 5 5 35 105
64 Tangela GRASS 435 65 55 115 100 40
```

Figura 3 – Inserção de cem elementos no Sequence Set



```

5 Fearow NORMAL 442 65 90 65 61 61
45 Ekans POISON 288 35 60 44 40 54
81 Arbok POISON 448 60 95 69 65 79
27 Pikachu ELECTRIC 320 35 55 40 50 50
61 Raichu ELECTRIC 485 60 85 50 95 85
91 Sandshrew GROUND 300 50 75 85 20 30
95 Nidoran POISON 275 55 47 52 40 40
42 Clefairy FAIRY 323 70 45 48 60 65
27 Vulpix FIRE 299 38 41 40 50 65
36 Vulpix ICE 299 38 41 40 50 65
91 Diglett GROUND 265 10 55 25 35 45
4 Dugtrio GROUND 425 35 100 50 50 70
2 Meowth DARK 290 40 35 35 50 40
53 Persian DARK 440 65 60 60 75 65
92 Golduck WATER 500 80 82 78 95 80
82 Mankey FIGHTING 305 40 80 35 35 45
21 Primeape FIGHTING 455 65 105 60 60 70
16 Growlithe FIRE 350 55 70 45 70 50
18 Arcanine FIRE 555 90 110 80 100 80
95 Poliwhirl WATER 385 65 65 65 50 50
47 Abra PSYCHIC 310 25 20 15 105 55
26 Alakazam PSYCHIC 600 55 50 65 175 105
71 Machoke FIGHTING 405 80 100 70 50 60
38 Ponyta FIRE 410 50 85 55 65 65
69 Rapidash FIRE 500 65 100 70 80 80
12 Seel WATER 325 65 45 55 45 70
67 Grimer POISON 325 80 80 50 40 50
99 Hypno PSYCHIC 483 85 73 70 73 115
35 Krabby WATER 325 30 105 90 25 25
94 Voltorb ELECTRIC 330 40 30 50 55 55
3 Cubone GROUND 320 50 50 95 40 50
11 Hitmonlee FIGHTING 455 50 120 53 35 110
22 Lickitung NORMAL 385 90 55 75 60 75
33 Koffing POISON 340 40 65 95 60 45
73 Chansey NORMAL 450 250 5 5 35 105
64 Tangela GRASS 435 65 55 115 100 40
41 Kangaskhan NORMAL 590 105 125 100 60 100
11 Horsea WATER 295 30 40 70 70 25
53 Seaking WATER 450 80 92 65 65 80
68 Magmar FIRE 495 65 95 57 100 85
47 Pinsir BUG 500 65 125 100 55 70
44 Tauros NORMAL 490 75 100 95 40 70
62 Magikarp WATER 200 20 10 55 15 20
57 Ditto NORMAL 288 48 48 48 48 48
37 Vaporeon WATER 525 130 65 60 110 95

```

Figura 4 – Inserção de cem elementos no Sequence Set

```

37 Vaporeon WATER 525 130 65 60 110 95
59 Jolteon ELECTRIC 525 65 65 60 110 95
23 Flareon FIRE 525 65 130 60 95 110
41 Porygon NORMAL 395 65 60 70 85 75
29 Snorlax NORMAL 540 160 110 65 65 110
78 Mew PSYCHIC 600 100 100 100 100 100
16 Chikorita GRASS 318 45 49 65 49 65
35 Meganium GRASS 525 80 82 100 83 100
90 Cyndaquil FIRE 309 39 52 43 60 50
42 Typhlosion FIRE 534 78 84 78 109 85
64 Croconaw WATER 405 65 80 80 59 63
48 Sentret NORMAL 215 35 46 34 35 45
46 Cleffa FAIRY 218 50 25 28 45 55
5 Mareep ELECTRIC 280 55 40 40 65 45
90 Ampharos ELECTRIC 510 90 75 85 115 90
29 Bellossom GRASS 490 75 80 95 90 100
70 Sudowoodo ROCK 410 70 100 115 30 65
50 Sunkern GRASS 180 30 30 30 30 30
6 Umbreon DARK 525 95 65 110 60 130
1 Misdreavus GHOST 435 60 60 60 85 85
93 Unown PSYCHIC 336 48 72 48 72 48
48 Snubbull FAIRY 300 60 80 50 40 40
29 Teddiursa NORMAL 330 60 80 50 50 50
23 Slugma FIRE 250 40 40 40 70 40
84 Remoraid WATER 300 35 65 35 65 35
54 Octillery WATER 480 75 105 75 105 75
40 Phanpy GROUND 330 90 60 60 40 40
66 Porygon NORMAL 515 85 80 90 105 95
76 Tyrogue FIGHTING 210 35 35 35 35 35
31 Magby FIRE 365 45 75 37 70 55
8 Miltank NORMAL 490 95 80 105 40 70
44 Raikou ELECTRIC 580 90 85 75 115 100
39 Treecko GRASS 310 40 45 35 65 55
26 Mudkip WATER 310 50 70 50 50 50
23 Poochyena DARK 220 35 55 35 30 30
37 Wurmple BUG 195 45 45 35 20 30
38 Cascoon BUG 205 50 35 55 25 25
18 Seedot GRASS 220 40 40 50 30 30
82 Slakoth NORMAL 280 60 60 60 35 35
29 Whismur NORMAL 240 64 51 23 51 23
41 Makuhita FIGHTING 237 72 60 30 20 30
13 Electrike ELECTRIC 295 40 45 40 65 40
23 Volbeat BUG 430 65 73 75 47 85
97 Walmer WATER 400 130 70 35 70 35
4 Torkoal FIRE 470 70 85 140 85 70
79 Spinda NORMAL 360 60 60 60 60 60

```

Figura 5 – Inserção de cem elementos no Sequence Set

```

17 Castform ICE 420 70 70 70 70 70

```

Figura 6 – Inserção de cem elementos no Sequence Set

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

```

busca: (41,Porygon,NORMAL,395,65,60,70,85,75,)
b 0
busca: (0,Squirtle,WATER,314,44,48,65,50,64,)
b 62
busca: (62,Rattata,NORMAL,253,30,56,35,25,35,)
b 64
busca: (64,Tangela,GRASS,435,65,55,115,100,40,)
b 36
busca: (36,Vulpix,ICE,299,38,41,40,50,65,)
b 2
busca: (2,Meowth,DARK,290,40,35,35,50,40,)
b 18
busca: (18,Arcanine,FIRE,555,90,110,80,100,80,)
b 12
busca: (12,Seel,WATER,325,65,45,55,45,70,)
b 94
busca: (94,Voltorb,ELECTRIC,330,40,30,50,55,55,)
b 22
busca: (22,Lickitung,NORMAL,385,90,55,75,60,75,)
b 42
busca: (42,Clefairy,FAIRY,323,70,45,48,60,65,)
b 92
busca: (92,Golduck,WATER,500,80,82,78,95,80,)
b 26
busca: (26,Alakazam,PSYCHIC,600,55,50,65,175,105,)
b 99
busca: (99,Hypno,PSYCHIC,483,85,73,70,73,115,)
b 11
busca: (11,Horsea,WATER,295,30,40,70,70,25,)
b 47
busca: (47,Abra,PSYCHIC,310,25,20,15,105,55,)
b 90
busca: (90,Cyndaquil,FIRE,309,39,52,43,60,50,)
b 6
busca: (6,Umbreon,DARK,525,95,65,110,60,130,)
b 40
busca: (40,Phanpy,GROUND,330,90,60,60,40,40,)
b 4
busca: (4,Dugtrio,GROUND,425,35,100,50,50,70,)

```

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.

```

41
Busca: (41,Bulbasaur,GRASS,318,45,49,49,65,65,)
0
Busca: elemento não encontrado.
62
Busca: (62,Rattata,NORMAL,253,30,56,35,25,35,)
64
Busca: (64,Tangela,GRASS,435,65,55,115,100,40,)
36
Busca: (36,Vulpix,ICE,299,38,41,40,50,65,)
2
Busca: elemento não encontrado.
18
Busca: (18,Seedot,GRASS,220,40,40,50,30,30,)
12
Busca: (12,Seel,WATER,325,65,45,55,45,70,)
94
Busca: (94,Voltorb,ELECTRIC,330,40,30,50,55,55,)
22
Busca: (22,Lickitung,NORMAL,385,90,55,75,60,75,)
42
Busca: (42,Typhlosion,FIRE,534,78,84,78,109,85,)
92
Busca: (92,Golduck,WATER,500,80,82,78,95,80,)
26
Busca: (26,Alakazam,PSYCHIC,600,55,50,65,175,105,)
99
Busca: (99,Hypno,PSYCHIC,483,85,73,70,73,115,)
11
Busca: (11,Horsea,WATER,295,30,40,70,70,25,)
47
Busca: (47,Abra,PSYCHIC,310,25,20,15,105,55,)
90
Busca: (90,Ampharos,ELECTRIC,510,90,75,85,115,90,)
6
Busca: (6,Umbreon,DARK,525,95,65,110,60,130,)
40
Busca: (40,Phanpy,GROUND,330,90,60,60,40,40,)
4
Busca: (4,Torkoal,FIRE,470,70,85,140,85,70,)

```

Figura 8 – Busca na Árvore B+

A Figura 9 representa o percorrimto da árvore em ordem.

```

(257/1963745280)(15360/290)(3/320)(4/425)(4/470)(5/442)]->[(5/280)(6/525)(8/490)(11/455)(11/295)(12/325)]->[(13/295)(16/350)(16/318)(17/420)(18/555)(18/220)(1
314)]->[(21/455)(22/385)(23/525)(23/250)(23/220)(23/430)(24/205)]->[(26/600)(26/310)(27/320)(27/299)(29/240)]->[(29/540)(29/490)(29/330)(31/365)(33/340)]->[(3
309)(35/325)(35/525)(36/299)]->[(37/525)(37/195)(38/410)(38/205)]->[(39/310)(40/330)(41/318)(41/590)(41/395)(41/237)]->[(42/323)(42/534)(44/490)(44/580)(45/28
(46/218)(47/310)(47/500)]->[(48/215)(48/300)(50/180)(53/440)(53/450)(54/480)(57/288)]->[(58/349)(59/525)(61/485)(62/253)(62/200)]->[(64/262)(64/435)(64/405)(6
515)(67/405)(67/325)(68/495)]->[(69/195)(69/500)(70/410)(71/405)(73/450)(76/210)(78/395)(78/600)]->[(79/360)(81/448)(82/305)(82/280)(84/300)(90/309)(90/510)(9
300)(91/265)]->[(92/500)(93/336)(94/330)(95/275)(95/385)(97/400)(99/483)]

```

Figura 9 – Percorrimento em ordem na árvore B+

A Figura 10 representa a depuração da árvore.

```

Nivel da Arvore : [ 1 ]
Id : 5
Nome : Mareep
Tipo : ELECTRIC
Total : 280
Ataque : 55
Defesa : 40
At esp : 40
At def : 65
Velocidade : 45

Nivel da Arvore : [ 1 ]
Id : 13
Nome : Electrike
Tipo : ELECTRIC
Total : 295
Ataque : 40
Defesa : 45
At esp : 40
At def : 65
Velocidade : 40

Nivel da Arvore : [ 1 ]
Id : 21
Nome : Primeape

```

```

Nivel da Arvore : [ 1 ]
Id : 21
Nome : Primeape
Tipo : FIGHTING
Total : 455
Ataque : 65
Defesa : 105
At esp : 60
At def : 60
Velocidade : 70

Nivel da Arvore : [ 1 ]
Id : 26
Nome : Alakazam
Tipo : PSYCHIC
Total : 600
Ataque : 55
Defesa : 50
At esp : 65
At def : 175
Velocidade : 105

Nivel da Arvore : [ 0 ]
Id : 34
Nome : Charmander
Tipo : FIRE
Total : 309
Ataque : 39

```

```

Nivel da Arvore : [ 1 ]
Id : 26
Nome : Alakazam
Tipo : PSYCHIC
Total : 600
Ataque : 55
Defesa : 50
At esp : 65
At def : 175
Velocidade : 105

Nivel da Arvore : [ 0 ]
Id : 34
Nome : Charmander
Tipo : FIRE
Total : 309
Ataque : 39
Defesa : 52
At esp : 43
At def : 60
Velocidade : 50

Nivel da Arvore : [ 1 ]
Id : 34
Nome : Charmander
Tipo : FIRE
Total : 309
Ataque : 39
Defesa : 52

```

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.