Descrição

Criar uma Árvore B+ que manipula dados da memória primária.

Dados da memória

Cria-se várias instâncias do tipo Número:

```
Class NUMERO {
 long long int indice;
 String nomeDoNumero;
 Bool éPar;
 Bool éPrimo;
 String observacao;
}
```

Árvore de Números: adicionar

Usa-se a árvore para manipular os números:

```
ARVORE b;
NUMERO * n1 = new Numero(60);
n1.setObservacao(“O minuto tem 60 segundos”);
b.adicionar(n);
```

Árvore de Números: remover

Usa-se a árvore para manipular os números:

```
Numero * n2 = b.remover(444);
delete n2;
```

Árvore de Números: buscar

Usa-se a árvore para manipular os números:

```
Numero * n3 = b.buscar(123);
cout << n3;
```

Árvore de Números: atualização

Usa-se a árvore para manipular os números:

Numero * n4 = b.buscar(60);

n4.setObservacao(“A hora tem 60*60 segundos”);

Nós da árvore

```
struct{
 char tipo; (1 byte)
 int contador; ( mín 2 byte)
 tipo_no* ponteiros;          ponteiros = new tipo_no[n]  (4 bytes * n)
 long long int * indice;      indice = new long long int[n-1] ( mín 8 bytes * (n-1) )
}
```

Total = 1 + 2 + 4n + 8(n-1)

Obs: mín = depende do SO, do processador, etc.

Grau da árvore

Tamanho do bloco = 4Kb

$1 + 2 + 4n + 8(n-1) = 4Kb$

$12n - 5 = 4Kb$

$12n = 4096 – 5$

$12n = 4091$

$n = \lfloor 340.92 \rfloor = 340$

Opcional que vale 10

Acesso sequencial

```
Numero n = b.busca(10);
enquanto( n != null && n <= 100){
 cout << n;
 n = b.proximo(n);
}
```