Pesquisa e Classificação de Dados - Trabalho da disciplina

Prof. Ricardo Oliveira - 2019/2

Enunciado

O trabalho consiste na implementação, na linguagem C, de um programa que manipula uma árvore B em disco, realizando as operações de busca e se inserção. O formato da árvore é definido a seguir:

Formato da Árvore em Disco

Cada nodo da árvore é armazenado em um arquivo em disco. Cada nodo tem um número identificador, e é representado no arquivo `ID.dat`, onde ID é seu identificador. O formato do arquivo deve *necessariamente* ser:

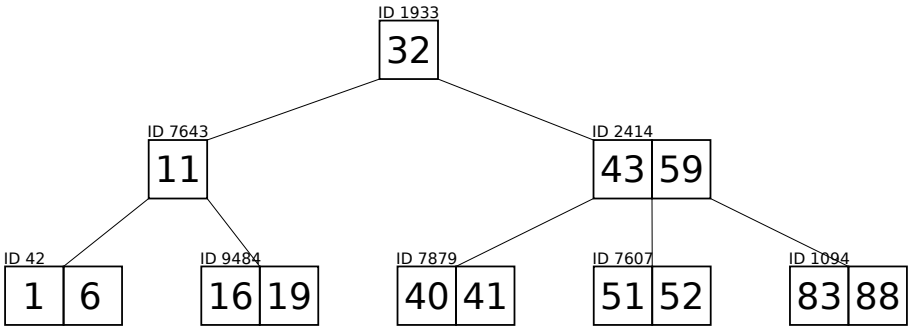
```
N
x0, x1, ...
f0, f1, ...
```

onde `N` é o número de chaves no nodo; `[x0, x1, ...]` é a lista de chaves no nodo, em ordem crescente; `f0, f1, ...` é a lista dos identificadores dos filhos do nodo, em ordem (ou uma lista de `N`+1 valores -1 para folhas).

Como exemplo, considere um nodo com ID 2414, contendo duas chaves 43 e 59, e três filhos com IDs 7879, 7607 e 1094. O arquivo `2414.dat` deve conter:

```
2
43 59
7879 7607 1094
```

A figura abaixo apresenta um exemplo completo de uma árvore B:



Os arquivos que a representam são:

- Arquivo `1933.dat`:

```
1
32
7643 2414
```

- Arquivo `7643.dat`:

```
1
11
42 9484
```

- Arquivo `2414.dat`:

```
2
43 59
7879 7607 1094
```

- Arquivo `42.dat`:

```
2
1 6
-1 -1 -1
```

- Arquivo `9484.dat`:

```
2
16 19
-1 -1 -1
```

- Arquivo `7879.dat`:

```
2
40 41
-1 -1 -1
```

- Arquivo `7607.dat`:

```
2
51 52
-1 -1 -1
```

- Arquivo `1094.dat`:

```
2
83 88
-1 -1 -1
```

Requisitos

No início de sua execução, seu programa deve ler do usuário a ordem/grau máximo D da árvore. Em seguida, seu programa deve ler o ID da raiz da árvore. Caso o arquivo respectivo (ID`.dat`) não exista, seu programa deve criar uma raiz vazia no arquivo ID`.dat` e imprimir a mensagem `Criada raiz vazia em` ID`.dat.` Caso o arquivo já exista, seu programa deve imprimir `Raiz em` ID`.dat encontrada.`

Em seguida, seu programa deve ler e processar uma sequência de comandos do usuário. Cada comando pode ser:

- `insere x`: Insere a chave `x` na árvore. Se a operação *split* for necessária, o ID do novo nodo criado deve ser gerado aleatoriamente (mas garanta que o arquivo ID`.dat` ainda não existe antes de sua criação). Se uma nova raiz for criada, seu programa deve imprimir `-- Nova raiz com ID [ID] --`, sendo `[ID]` o ID da nova raiz.

  Você pode assumir que nenhuma chave repetida será inserida na árvore.

- `busca x`: Busca a chave `x` na árvore. Se a chave for encontrada, seu programa deve imprimir `[x] encontrado`. Caso contrário, imprima `[x] nao encontrado`, sendo `[x]` a chave buscada.

- `fim`: encerra a execução do programa.

Durante qualquer operação, seu programa deve imprimir `Lido (...)` sempre que um nodo é lido do disco, e `Escrito (...)` sempre que um nodo é escrito em disco (onde `...` é a lista de chaves do nodo). Ao final de cada comando (tanto inserção quanto busca), seu programa deve imprimir o número de leituras e escritas feitas por ela.

Considere o seguinte exemplo de execução (é altamente recomendado acompanhar a execução na página `https://www.cs.usfca.edu/~galles/visualization/BTree.html`):

```
Digite Grau Maximo: 3
Digite ID raiz: 999
Escrito ()
Criada raiz vazia em 999.dat.

> insere 50
Lido ()
Escrito (50)
1 leitura(s), 1 escrita(s)

> insere 25
Lido (50)
Escrito (25 50)
1 leitura(s), 1 escrita(s)

> insere 70
Lido (25 50)
Escrito (25)
Escrito (70)
Escrito (50)
-- Nova raiz com ID 5557 --
1 leitura(s), 3 escrita(s)

> busca 55
Lido (50)
Lido (70)
55 nao encontrado.
2 leitura(s), 0 escrita(s)

> insere 55
Lido (50)
Lido (70)
Escrito (55 70)
2 leitura(s), 1 escrita(s)

> busca 55
Lido (50)
Lido (55 70)
55 encontrado.
2 leitura(s), 0 escrita(s)

> fim
```

Após a execução do programa, se os arquivos `.dat` não forem removidos do disco, deve ser possível "abrir" a mesma árvore B em uma nova execução do programa, fornecendo o ID de sua raiz:

```
Digite Grau Maximo: 3
Digite ID raiz: 5557
Raiz em 5557.dat encontrada.

> busca 70
Lido (50)
Lido (55 70)
70 encontrado.
2 leitura(s), 0 escrita(s)

> fim
```

A execução que gerou a árvore dada como exemplo na página 2 está no apêndice.

Implementação

Seu programa deve representar um nodo da árvore com a seguinte *struct*:

```
#define TAM 100
typedef struct {
    int n;              // numero de chaves no nodo
    int chaves[TAM];    // vetor de chaves
    int filhos[TAM];    // vetor de filhos
} nodo;
```

Deve haver uma única variável desta *struct* declarada em todo o programa (pode ser global). Ainda, **nenhum outro vetor deve ser declarado ou alocado** (com excessão possivelmente de uma string auxiliar para a função `fopen` e outra para leitura dos comandos do usuário). Em outras palavras, com excessão da *struct* (, das strings) e da *call stack*, seu programa deve ter complexidade de espaço $O(1)$ na memória principal.

Seu programa deve conter (ao menos) duas funções:

- `void le(int ID)`: abre o arquivo ID`.dat` e carrega o nodo na *struct* definida acima;
- `void escreve(int ID)`: escreve em disco o nodo presente na *struct*, no arquivo ID`.dat`.

Todos os nodos devem ser lidos e escritos usando apenas estas funções.

Orientações

- O trabalho pode ser feito por equipes de *até* 2 (dois) estudantes;
- Submeta, via *Moodle*, um pacote (zip ou tar.gz) contendo todo o código fonte do trabalho, além de um arquivo de texto (txt) onde conste:

  - O nome de todos os integrantes da equipe;
  - Toda informação que a equipe julgar relevante para a correção (como *bugs* conhecidos, detalhes de implementação, escolhas de projeto, etc.)

- Comente adequadamente seus códigos para facilitar a correção.
- Atenção: a correção será parcialmente automatizada, e a saída do programa será testada com outras entradas além das fornecidas como exemplo. *Siga* ***fielmente*** *o formato de saída dado nos exemplos*, sob pena de grande redução da nota;
- Certifique-se que seu programa funciona antes de submetê-lo;
- O trabalho deve ser entregue até **5 de Dezembro de 2019 (quinta-feira), 23:59**, via *Moodle*. É suficiente que o trabalho seja submetido por apenas um estudante da equipe;
- Trabalhos copiados ou plagiados receberão **todos** a nota 0 (**ZERO**).

Apêndice: Exemplo de execução completo

```
Digite Grau Maximo: 4
Digite ID raiz: 42
Escrito ()
Criada raiz vazia em 42.dat.

> insere 88
Lido ()
Escrito (88)
1 leitura(s), 1 escrita(s)

> insere 83
Lido (88)
Escrito (83 88)
1 leitura(s), 1 escrita(s)

> insere 6
Lido (83 88)
Escrito (6 83 88)
1 leitura(s), 1 escrita(s)

> insere 59
Lido (6 83 88)
Escrito (6)
Escrito (83 88)
Escrito (59)
-- Nova raiz com ID 7643 --
1 leitura(s), 3 escrita(s)

> insere 52
Lido (59)
Lido (6)
Escrito (6 52)
2 leitura(s), 1 escrita(s)

> insere 51
Lido (59)
Lido (6 52)
Escrito (6 51 52)
2 leitura(s), 1 escrita(s)

> insere 43
Lido (59)
Lido (6 51 52)
Escrito (6)
Escrito (51 52)
Lido (59)
Escrito (43 59)
3 leitura(s), 3 escrita(s)

> insere 41
Lido (43 59)
Lido (6)
Escrito (6 41)
2 leitura(s), 1 escrita(s)

> insere 40
Lido (43 59)
Lido (6 41)
Escrito (6 40 41)
2 leitura(s), 1 escrita(s)

> insere 32
Lido (43 59)
Lido (6 40 41)
Escrito (6)
Escrito (40 41)
Lido (43 59)
Escrito (32 43 59)
3 leitura(s), 3 escrita(s)

> insere 19
Lido (32 43 59)
Lido (6)
Escrito (6 19)
2 leitura(s), 1 escrita(s)

> insere 16
Lido (32 43 59)
Lido (6 19)
Escrito (6 16 19)
2 leitura(s), 1 escrita(s)

> insere 11
Lido (32 43 59)
Lido (6 16 19)
Escrito (6)
Escrito (16 19)
Lido (32 43 59)
Escrito (11)
Escrito (43 59)
Escrito (32)
-- Nova raiz com ID 1933 --
3 leitura(s), 5 escrita(s)

> insere 1
Lido (32)
Lido (11)
Lido (6)
Escrito (1 6)
3 leitura(s), 1 escrita(s)

> fim
```