Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2,5 |  |
| 2a) | 2,5 |  |
| 3a) | 2,5 |  |
| 4a) | 2,5 |  |
|  | 10.0 |  |

**LEIA COM CUIDADO**

1. A prova é individual e sem consulta.
   1. **Qualquer tentativa de “cola” resultará na anulação da prova do aluno ou dos alunos envolvidos**.
   2. Os aparelhos celulares deverão permanecer desligados e guardados fora do alcance durante toda a prova. **Aparelhos celulares ligados ou de alguma forma visíveis serão tratados como tentativa de “cola”**.
2. A interpretação faz parte da questão.
   1. **Não há perguntas durante a prova.**
   2. Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
3. O tempo de prova é 1:45 h.
4. **Após o início da prova, não será possível sair e voltar à sala.**
5. As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
6. A prova pode ser feita a lápis.

**Questão 1 (2,5 pontos)** Implemente, de forma não recursiva, uma função que calcule o *antecessor* de um nó em uma árvore de busca binária. Por simplicidade, assuma que os valores de chave são inteiros. A função deve ter o seguinte protótipo:

Abb\* abb\_antecessor (Abb\* r)**;**

A função recebe como entrada um apontador para o nó que se deseja achar o antecessor e deve retornar um apontador para o antecessor, se houver, ou o apontador nulo, em caso contrário. O código deve ser completo, ou seja, deve incluir a função e todas as outras que por ventura forem por ela chamadas.

Adote a seguinte estrutura para os nós:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* esq;

Abb\* dir;

};

***Resposta***

*Caso 1* - *r* possui sub-árvore esquerda: o antecessor será o máximo da sub-árvore esquerda de *r*.

*Caso 2* - *r* não possui sub-árvore esquerda: o antecessor de *r* será o nó *p* tal que *r* é o mínimo da sub-arvore à direta de *p* (ou seja, *p* é o antecessor de *r* sse *r* é o sucessor de *p*). Neste caso, a estrutura deve incluir um ponteiro para o pai:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* pai;

Abb\* esq;

Abb\* dir;

};

A solução é semelhante à rotina para computar o sucessor de um nó apresentada nas notas de aula, apenas invertendo direita com esquerda e mínimo por máximo.

Abb\* abb\_antecessor (Abb\* r) {

if (r==NULL) return NULL;

else if (r->esq!=NULL) /\* caso 1: retorna o máximo da sub-árvore esq. \*/

return abb\_max(r->esq);

else { /\* caso 2: retorna o ancestral mais próximo   
 que seja menor que seu filho \*/

Abb\* p = r->pai;

while (p!=NULL && r==p->esq) {

r = p;

p = p->pai;

}

return p;

}

}

Abb\* abb\_max (Abb\* r) {

if (r==NULL) return NULL;

while(r->dir!=NULL)

r=r->dir;

return r;

}

**Questão 2** **(2,5 pontos)** Em cada um dos casos abaixo indique qual ou quais rotações devem ser feitas para rebalancear a árvore AVL após a operação indicada. Explique a sua resposta, usando os fatores de balanceamento, e indique a forma da árvore a cada rotação.

(a) (1,0 ponto) Operação: **Inserção da chave 7**.

5

3

10

12

6

(b) (1,5 ponto) Operação: **Remoção da chave 7**. **Apresente 2 formas distintas para remover 7.**

5

7

2

10

6

1

3

9

11

***Resposta***

(a) Inserção de 7

5

3

10

12

6

7

Na árvore acima, a raiz possui fator de balanceamento +2 e o nó rotulado com 10 tem fator de balanceamento -1. Logo, deve-se fazer uma ***rotação direita-esquerda***.

* ***Rotação à direita*** a partir do nó rotulado com 10, resultando na árvore:

5

3

6

10

7

12

* ***Rotação à esquerda*** a partir do nó rotulado com 6, resultando na árvore:

5

3

6

10

12

7

b) Remoção de 7:

5

2

7

1

3

10

6

11

9

* Primeira forma: Troca-se 7 com seu sucessor 9, e remove-se 7:

5

2

9

1

3

10

6

11

A árvore está balanceada.

* Segunda forma: Troca-se 7 com seu antecessor 6, e remove-se 7:

5

2

6

1

3

10

11

9

O nó rotulado com 6 possui fator de balanceamento +2 e o nó rotulado com 10 tem fator de balanceamento 0. Logo, deve-se fazer uma ***rotação à esquerda*** no nó rotulado com 6.

5

2

10

1

3

11

9

6

**Questão 3** (2,5 pontos) Considere a árvore B de ordem 5 (todo nó, exceto a raiz, tem no mínimo 2 chaves e no máximo 4 chaves) abaixo:

100

53 77 123 200 242 301

10 40 60 70 80 90 110 113 130 138 140 170 230 236 250 270 350 370

(a) Realize cada uma das seguintes operações sobre a árvore acima. Indique os nós que sofrem modificações após cada operação, bem como a ocorrência de divisão, redistribuição ou concatenação:

(i) (1,0 ponto) Inserção de 143.

(ii) (1,0 ponto) Remoção de 77.

(b) (0,5 ponto) Existe alguma forma alternativa à remoção de 77 apresentada no item (a-ii)? A forma alternativa é mais eficiente? Explique sua resposta.

***Resposta***

a) Inserção de 143.

* Pesquise o nó onde a chave 143 deve ser inserida.

100

53 77 123 200 242 301

10 40 60 70 80 90 110 113 130 138 140 170 230 236 250 270 350 370

***143***

* Divida o nó e insira a chave do meio no pai, duas vezes.

100

53 77 123 200 242 301

10 40 60 70 80 90 110 113 130 138 143 170 230 236 250 270 350 370

***140***

100

53 77 ***200*** 123 140 242 301

10 40 60 70 80 90 110 113 130 138 143 170 230 236 250 270 350 370

100 200

53 77 123 140 242 301

10 40 60 70 80 90 110 113 130 138 143 170 230 236 250 270 350 370

b) Remoção de 77.

* Troque 77 com a sua sucessora 80 e remova 77.

100

53 **80** 123 200 242 301

10 40 **60 70** **90** 110 113 130 138 140 170 230 236 250 270 350 370

* Concatene nós, com transferência de chave do pai para o nó resultante da concatenação:

**100**

53 **123** 200 242 301

10 40 **60 70 80 90** 110 113 130 138 140 170 230 236 250 270 350 370

* Redistribua chaves entre os nós de segundo nível, com transferência da chave do pai:

**123**

53 **100** 200 242 301

10 40 60 70 80 90 110 113 130 138 140 170 230 236 250 270 350 370

(c) Poderíamos remover 77 trocando-a com a sua antecessora, a chave 70. Porém, esta alternativa também resultaria em um nó com apenas 1 chave, que também deveria ser concatenado com um vizinho, etc.

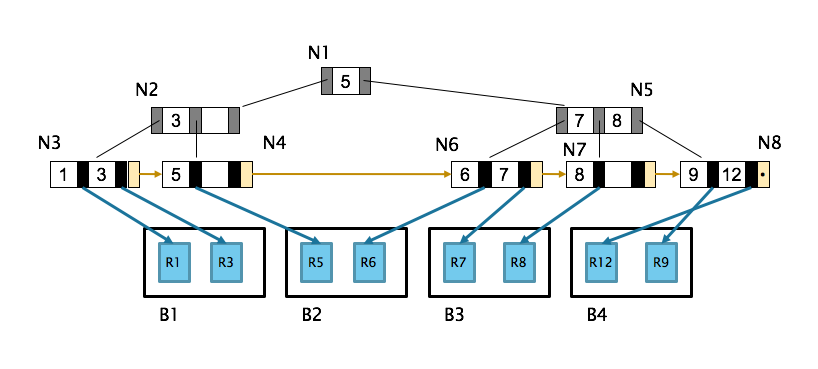
**Questão 4 (2,5 pontos).** Considere a árvore B+ de ordem 3 (ou seja, cada nó, exceto a raiz, pode ter no mínimo 1 chave e no máximo 2 chaves) e um buffer pool capaz de armazenar 5 nós ou blocos de dados, indistintamente. Assuma que o buffer pool é gerenciado de tal forma que os nós ou blocos de dados há mais tempo sem serem usados são retirados do buffer pool, quando necessário.

(a) (1,0 ponto) Assuma que o buffer pool está inicialmente vazio, mas mantém o estado da pesquisa anterior quando uma nova pesquisa é iniciada. Mostre os estados por que passa o buffer pool quando são recuperados os dados associados às seguintes chaves, nesta ordem: 3, 5, 6. Explique sua resposta.

(b) (1,0 ponto) Assuma que o buffer pool está inicialmente vazio. Mostre os estados por que passa o buffer pool quando são recuperados os dados associados às chaves no intervalo [5,8]. Use o algoritmo mais eficiente possível. Explique sua resposta.

(c) (0,5 ponto) Na árvore B+ da Figura 1, os dados estão organizados nos blocos de dados de forma eficiente? Explique sua resposta.

Em todos os itens, refira-se aos nós e blocos de dados usando os rótulos indicados na figura abaixo.



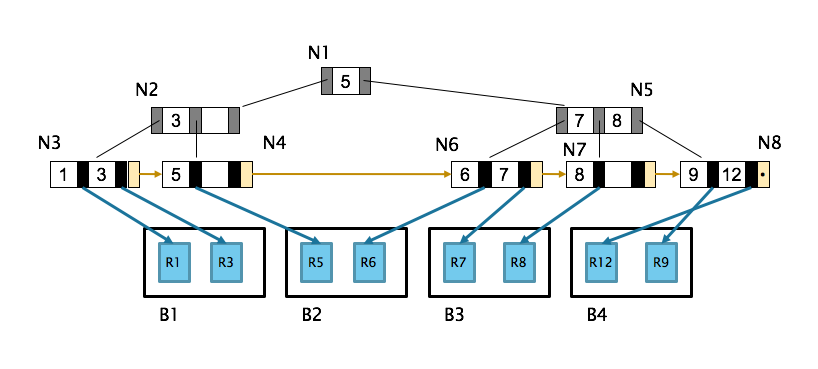
**Figura 1**. Árvore B+.

Use a tabela abaixo para descrever as suas respostas.

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Buffer Pool | Acesso à memória secundária | Número total de acessos |
|  | (vazio) |  |  |
| (a) Pesquisa da chave 3 |  |  |  |
| (a) Pesquisa da chave 5 |  |  |  |
| (a) Pesquisa da chave 6 |  |  |  |
| (b) Pesquisa no intervalo [5,8] |  |  |  |

***Resposta***

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Buffer Pool | Acesso à memória secundária | Número total de acessos |
|  | (vazio) |  |  |
| (a) Pesquisa da chave 3 | N1  N2 N1  N3 N2 N1  B1 N3 N2 N1 | 1  1  1  1 | 4 |
| (a) Pesquisa da chave 5 | N1 B1 N3 N2  N2 N1 B1 N3  N4 N2 N1 B1 N3  B2 N4 N2 N1 B1 | 0  0  1  1 | 2 |
| (a) Pesquisa da chave 6 | N1 B2 N4 N2 B1  N5 N1 B2 N4 N2  N6 N5 N1 B2 N4  B2 N6 N5 N1 N4 | 0  1  1  0 | 2 |
| (b) Pesquisa no intervalo [5,8]  Pesquise a primeira chave, 5  Recupere os dados associados a 5  Pesquise 6 no mesmo nó que 5  Navegue para a folha seguinte  Recupere os dados associados a 6  Pesquise 7 no mesmo nó que 6  Recupere os dados associados a 7  Pesquise 8 no mesmo nó que 7  Navega para a folha seguinte  Recupere os dados associados a 8 | N1  N2 N1  N4 N2 N1  B2 N4 N2 N1  N4 B2 N2 N1  N6 N4 B2 N2 N1  B2 N6 N4 N2 N1  N6 B2 N4 N2 N1  B3 N6 B2 N4 N2  N6 B3 B2 N4 N2  N7 N6 B3 B2 N4  B3 N7 N6 B2 N4 | 1  1  1  1  0  1  0  0  1  0  1  0 | 7 |



(c) Na árvore B+ da Figura 1, os dados estão organizados nos blocos de dados de forma eficiente pois dados de chaves contíguas estão no mesmo bloco na medida do possível, permitindo varrer continuamente os dados associados a chaves em um intervalo.