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)** Defina o *comprimento* de um ramo de umaárvore binária (o que incluiu as árvores de busca binárias e as árvores AVL) como o número de nós que fazem parte do ramo.

a) **(0,5 ponto)** Qual o custo, no pior caso, para computar o comprimento do maior ramo de umaárvore binária com *n* nós. Explique a sua resposta.

b) **(1,0 ponto)** Implemente em C uma função que calcula o comprimento do maior ramo de uma árvore AVL com *n* nós de tal forma que a complexidade, no pior caso, da sua função seja *O*(*log*(*n*)). A função deverá ter o seguinte protótipo:

int avl\_longest(AvlNode\* node)**;**

e utilizar a estrutura:

typedef struct \_avl\_node AvlNode;

struct \_avl\_node {

void\* info;

int bf; /\* balance factor \*/

AvlNode\* parent;

AvlNode\* left;

AvlNode\* right;

};

c) **(0,5 ponto)** Argumente porque a implementação da função, apresentada no item (b), está correta.

d) **(0,5 ponto)** Argumente porque a implementação da função, apresentada no item (b), de fato possui complexidade, no pior caso, *O*(*log*(*n*)).

***Resposta***

a) O custo, no pior caso, será *O*(*n*) pois a árvore poderá degenerar em um único ramo com todos os *n* nós.

b)

1. int node\_longest(AvlNode\* node) {
2. int n=0;
3. while(node!=NULL) {
4. node = (node->bf>0) ? node->right : node->left;
5. n++;
6. }
7. return n;
8. }

c) O teste da linha 4 continua descendo pelo filho de node cuja subárvore tem o maior ramo: se node->bf>0 será o filho à direita; se node->bf<0 será o filho à direita; se node->bf=0 será indiferente pois as duas subárvores tem o maior ramo de igual comprimento. Portanto, se a árvore for AVL, o algoritmo corretamente computará o número de nós do maior ramo da árvore.

d) O comprimento do maior ramo de uma árvore binária é a altura mais 1. Mas altura de uma árvore AVL é proporcional a *log*(*n*). Logo, a implementação do item (b) possui custo, no pior caso, *O*(*log*(*n*)).

**Questão 2 (2,5 pontos)** Considere as árvores-B abaixo.

50

30 39 70 90

10 20 37 38 40 60 80 100

**Figura 1. Árvore B de ordem 3.**

100

53 77 123 200

10 40 60 70 80 90 110 113 130 138 140 170 230 236 242 245

**Figura 2. Árvore B de ordem 5.**

Realize as seguintes operações, indicando os nós que sofrem modificações (divisão, redistribuição ou concatenação) após cada operação. Justifique sempre porque não há mais modificações a fazer.

a) **(0,5 ponto)** Insira a chave 36 na árvore-B da Figura 1.

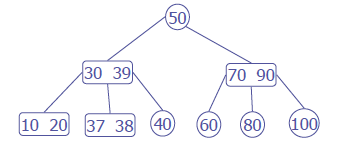
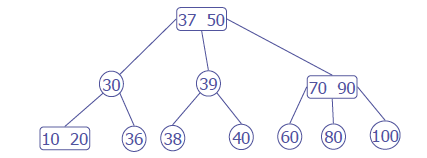
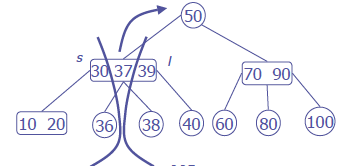
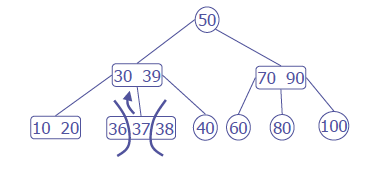
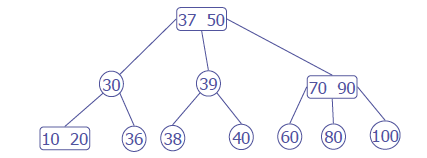
b) **(1,0 ponto)** Remova a chave 70 e depois a chave 90 da árvore-B da Figura 1.

c) **(0,5 ponto)** Insira a chave 143 na árvore-B da Figura 2.

d) **(0,5 ponto)** Remova a chave 100 da árvore-B da Figura 2.

***Resposta***

(a) Ver notas de aula.



(b)

50

30 39 70 90

10 20 37 38 40 60 80 100

A chave 70 não ocorre em uma folha. Troque 70 com o sucessor 80.

50

30 39 80 90

10 20 37 38 40 60 70 100

Remova a chave 70.

50

30 39 80 90

10 20 37 38 40 60 100

Remova o nó vazio, redistribuindo a chave 80 para a folha.

50

30 39 90

10 20 37 38 40 60 80 100

A chave 90 não ocorre em uma folha. Troque 90 com 100.

50

30 39 100

10 20 37 38 40 60 80 90

Remova a chave 90.

50

30 39 100

10 20 37 38 40 60 80

Redistribua as chaves 80 e 100.

50

30 39 80

10 20 37 38 40 60 100

c) Inserção de 143.

Pesquise o nó onde a chave 143 deve ser inserida.

100

53 77 123 200

10 40 60 70 80 90 110 113 130 138 140 170 230 236 242 245

***143***

Divida o nó e insira a chave do meio (140) no pai.

100

53 77 123 200

10 40 60 70 80 90 110 113 130 138 140 143 170 230 236 243 245

100

53 77 123 140 200

10 40 60 70 80 90 110 113 130 138 143 170 230 236 243 245

d) Remoção de 100.

A chave 100 ocorre em um nó interior e portanto deve ser trocada com a sua sucessora (ou antecessora).

100

53 77 123 200

10 40 60 70 80 90 110 113 130 138 143 170 230 236 243 245

Trocando-se 100 com a sucessora 110, temos:

110

53 77 123 200

10 40 60 70 80 90 100 113 130 138 143 170 230 236 243 245

Removendo-se 100, tem-se um nó com menos chaves do que o número permitido (2 chaves):

110

53 77 123 200

10 40 60 70 80 90 113 130 138 143 170 230 236 243 245

Combinando-se então as chaves do irmão da direita do nó afetado, obtém-se:

110

53 77 130 200

10 40 60 70 80 90 113 123 138 143 170 230 236 243 245

**Questão 3 (2,5 pontos)** Considere uma família de árvores definida de forma semelhante a árvores B, exceto que:

* Os nós possuem tamanho variável entre 128 e 256 bytes
* As chaves possuem tamanho variável, entre 22 bytes e 44 bytes
* Os ponteiros ocupam 4 bytes
* Em cada nó, há um campo a mais, de 4 bytes, indicando o número de chaves que o nó efetivamente armazena

1. **(1,0 ponto)** Qual é o maior número de chaves que uma árvore de altura 2 armazena (uma árvore que só tem a raiz possui altura 0, por convenção)? Explique cuidadosamente sua resposta.
2. **(1,0 ponto)** Qual é o número mínimo de chaves, estando todos os nós completamente preenchidos, que uma árvore de altura 2 armazena? Explique cuidadosamente sua resposta.
3. **(0,5 ponto)** A escolha dos tamanhos mínimo e máximo dos nós é adequada para o tamanho das chaves? Explique sua resposta.

***Resposta***

a) O maior número de chaves será atingido quando todas as chaves possuírem o menor tamanho e os nós, o maior tamanho. Seja *m* o número de chaves que podem ser acomodadas em um nó. Cada chave é acompanhada por um ponteiro, exceto a primeira, que é acompanhada por 2 ponteiros. Há ainda 4 bytes adicionais para indicar o número de chaves. Para o caso de todas as chaves terem o menor tamanho, 16 bytes, e os e os nós, o maior tamanho, 256 bytes, temos:

256 = (4 bytes para número de chaves) +

(4 bytes para o primeiro ponteiro) +

(4 bytes para cada ponteiro + 22 bytes de cada chave) \* *m*

*m* = (256-8)/26 = 248/26 = 9 chaves

Logo, cada nó terá 10 filhos. Portanto, como a árvore possui altura 2, o número máximo de chaves será:

*CM* = 9 + 9\*10 + 9\*10\*10 = 9 + 90 + 900 = 999

b) O menor número de chaves, quando os nós estão completos, será atingido quando todas as chaves possuírem o maior tamanho e os nós, o menor tamanho. Cada nó completo terá *p* chaves de tamanho máximo, onde *p* é dado por:

128 = 4 + 4 + (4 + 44) \* *p*

*p* = (128-8)/48 = 120/48 = 2

Logo, cada nó terá 3 filhos. Portanto, como a árvore possui altura 2, o número de chaves de tamanho máximo em nós completos será:

*Cm* = 2 + 2\*3 + 2\*3\*3 = 2 + 6 + 18 = 26

c) Os nós são demasiadamente pequenos. Mesmo no melhor caso - item (a), o número de chaves por nó é pequeno. No pior caso - item (b), a árvore degenera em uma árvore binária.

**Questão 4 (2,5 pontos)** Considere tabelas de dispersão estendidas tais que:

* Após aplicar a função de hash a uma chave, os bits são considerados da direita para a esquerda.
* Cada bloco pode conter um mínimo de 1 e um máximo de 3 chaves.

*T*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =3* |  |  |  |
| 0 |  |  |  | *B1* |
| 1 |  |  |  | *j1 = ??* | |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |

**Figura 3. Representação parcial de uma tabela de dispersão.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =2* |  |  |  |
| 0 |  |  | *K11* | *B1* |
| 1 |  |  | *K12* | *j1 =2* | |
| 2 |  |  |  |  | |
| 3 |  |  |  |  |
|  |  |  | *K21* | *B2* |
|  |  |  | *K22* | *j2 =2* | |
|  |  |  |  |  | |
|  |  |  |  |  |
|  |  |  | *K31* | *B3* |
|  |  |  | *K32* | *j3 =2* | | |
|  |  |  |  |  | | |
|  |  |  |  |  |
|  |  |  | *K41* | *B4* |
|  |  |  | *K42* | *j4 =2* | |
|  |  |  |  |  | |

**Figura 4. Representação esquemática de uma tabela de dispersão.**

a) **(0,5 ponto)** Considere a tabela de dispersão estendida, *T*, da Figura 3. Qual o valor de *j1* para o bloco *B1*? Explique sua resposta.

b) **(0,5 ponto)** Considere a tabela de dispersão estendida, *T*, da Figura 3. Quantos bits estão sendo considerados e qual ou quais são as possíveis combinações dos últimos bits das chaves que são armazenadas no bloco *B1*? Explique sua resposta.

c) **(0,5 ponto)** Suponha que um bloco *Bi* esteja completo. Explique todos os casos de inserção de uma nova chave *K* em *Bi*, supondo que não possam haver chaves duplicadas.

d) **(1,0 ponto)** Considere a tabela de dispersão estendida da Figura 4. Mostre como a estrutura ficará após as remoções, sucessivamente, de *K41* e *K42*. Explique sua resposta.

***Resposta***

a) Como *i = 3* e há *p = 2(3-2)= 2* apontadores para *B1*, temos que e *j1 = 2*.

b) Como *i = 3*, as chaves em *T*[0] terminam em “000” e as chaves em *T*[4] em “100”. Logo, como os apontadores de *T*[0] e de *T*[4] apontam para *B1*, este bucket pode possuir chaves terminando em “000” ou “100”.

c) Quando o bucket *B* está completo e *i-jB* > 0, o número de bits de relevância deve ser incrementado, um novo bucket *B’* deve ser criado e chaves redistribuídas entre *B* e *B’*. Este processo deve ser repetido até seja possível armazenar *K* ou   
*i-jB* = 0

Quando o bucket *B* está completo e *i-jB* = 0, a tabela de hash *H* deve ser duplicada, e o processo anterior repetido.

Mesmo que dois registros não possam ter a mesma chave (passada como entrada para a função de hash), o processo descrito anteriormente pode não ser suficiente para resolver o problema de overflow. Uma estratégia de encadeamento de buckets deverá ser então implementada.

d) Suponha que cada bloco possa conter um mínimo de 1 e um máximo de 3 chaves. Após a remoção de *K41* a estrutura e os buckets permanecem os mesmos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =2* |  |  |  |
| 0 |  |  | *K11* | *B1* |
| 1 |  |  | *K12* | *j1 =2* | |
| 2 |  |  |  |  | |
| 3 |  |  |  |  |
|  |  |  | *K21* | *B2* |
|  |  |  | *K22* | *j2 =2* | |
|  |  |  |  |  | |
|  |  |  |  |  |
|  |  |  | *K31* | *B3* |
|  |  |  | *K32* | *j3 =2* | | |
|  |  |  |  |  | | |
|  |  |  |  |  |
|  |  |  | *K42* | *B4* |
|  |  |  |  | *j4 =2* | |
|  |  |  |  |  | |

Porém, após a remoção de *K42*, o bucket *B4* fica vazio e deve ser descartado. O apontador de *T*[3] deve passar a apontar para o “companheiro” de *B4*, ou seja, devemos fazer *T*[3] = *T*[1]. Temos ainda que atualizar *j2 =*1 pois *B2* agora é apontado por dois apontadores.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =2* |  |  |  |
| 0 |  |  | *K11* | *B1* |
| 1 |  |  | *K12* | *j1 =*2 | |
| 2 |  |  |  |  | |
| 3 |  |  |  |  |
|  |  |  | *K21* | *B2* |
|  |  |  | *K22* | *j2 =*1 | |
|  |  |  |  |  | |
|  |  |  |  |  |
|  |  |  | *K31* | *B3* |
|  |  |  | *K32* | *j3 =*2 | | |
|  |  |  |  |  | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | |
|  |  |  |  |  | |