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)**

a) **(1,5 pontos)** Defina em pseudo-código (ou seja, não é necessário apresentar o código em C) um algoritmo não recursivo que recebe como entrada um ponteiro *r* para a raiz de uma árvore binária de busca e uma chave *K* e retorna um ponteiro *n* para o nó *N* da árvore tal que a chave armazenada em *N* seja a *menor chave armazenada na árvore que é maior ou igual* a *K*. Se *N* não existir, o algoritmo deve retornar NULL. Por simplicidade, assuma que os valores de chave são inteiros.

Assuma a seguinte estrutura para os nós de uma árvore binária de busca:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* pai;

Abb\* esq;

Abb\* dir;

};

b) **(1,0 ponto)** Argumente porque a sua implementação está correta.

***Resposta***

a)

Entrada: *r* – um ponteiro para a raiz de uma árvore binária de busca

*K* – um inteiro positivo

Saída: *n* – um ponteiro para o nó *N*, se houver, definido como acima, ou o ponteiro nulo

1. Se a árvore for vazia, pare e retorne NULL;
2. Pesquise se a chave *K* existe na árvore, mantendo também um ponteiro *p* para o pai *P* do nó sendo visitado;
3. Se existir um nó *N* na árvore cuja chave é *K*, retorne um ponteiro para *N* e pare;
4. Se a chave de *P* for maior do que *K*, retorne *p* e pare.
5. Suba na árvore a partir de *p* até encontrar o primeiro ancestral *Q* cuja chave seja maior que do que a chave do seu filho ou chegar à raiz.
6. Retorne um ponteiro para *Q*, ou NULL, se o loop de (5) tiver chegado à raiz.

b)

* O algoritmo do item (a) pesquisa se existe um nó *N* na árvore com a chave *K*. Se existir, um ponteiro para *N* será a resposta correta.
* Caso não exista um nó *N* na árvore com a chave *K*, o algoritmo simula a inserção de um nó *M* com a chave *K* (como filho do nó *P*) e pesquisa o sucessor deste nó.
* Como o nó fictício *M* seria uma folha, não tem uma subárvore à direita.
* Logo, o sucessor do nó fictício *M* é computado subindo na árvore a partir do pai *P* de *M* e procurando o primeiro ancestral *Q* cuja chave seja maior que do que a chave do seu filho. Este ancestral *Q* conterá a chave que seria a sucessora de *K* na árvore, ou seja, a menor chave maior do que *K*.
* O algoritmo retorna um ponteiro para *Q*. Se a busca chegar na raiz (ancestral NULL), o algoritmo retorna NULL. Neste caso, a chave *K* é maior do que qualquer chave armazenada na árvore.

**Questão 2 (2,5 pontos)**

a) **(1,5 pontos)** Implemente em C *de forma não recursiva e eficiente* uma função que calcula a altura de uma árvore AVL, com o seguinte protótipo:

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

visitando o menor número possível de nós. Utilize a estrutura:

typedef struct \_avl\_node AvlNode;

struct \_avl\_node {

void\* info;

int bf; /\* balance factor \*/

AvlNode\* parent;

AvlNode\* left;

AvlNode\* right;

};

b) **(1,0 ponto)** Argumente porque a sua implementação está correta e é eficiente.

***Resposta***

a)

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

b) O teste da linha 4 continua descendo pelo filho de node cuja subárvore tem a maior altura: 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 a mesma altura. Portanto, se a árvore for AVL, o algoritmo corretamente computará a altura da árvore.

A implementação é eficiente pois não precisa computar a altura das duas subárvores, mas apenas daquela que sabidamente (via o fator de balanceamento) tem a maior altura.

**Questão 3 (2,5 pontos)** Considere as árvores abaixo.

(i)

50

30 39 70 90

10 20 37 38 40 60 80 100

(ii)

50

30 33 70 90

10 20 31 32 41 60 81 89 94 100

Para os itens (a) a (c), descreva passo a passo todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique sempre porque não há mais modificações a fazer.

a)(0,5 ponto) Assuma que a árvore (i) é uma árvore B de ordem 3. Insira a chave 36.

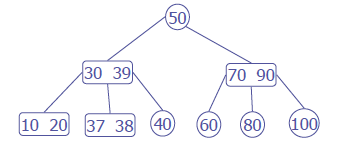
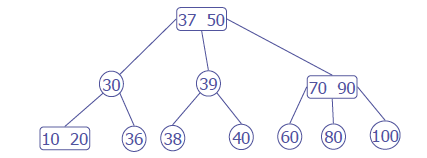
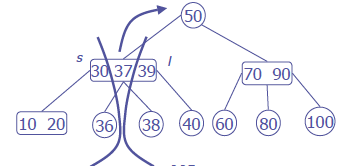
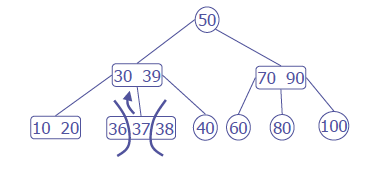
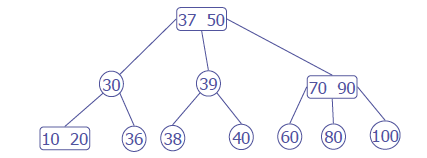
b)(1,0 ponto) Assuma que a árvore (i) é uma árvore B de ordem 3. Remova a chave 70 e depois a chave 90.

c) (0,5 ponto) Assuma que a árvore (ii) é uma árvore B de ordem 3. Remova a chave 50.

d) (0,5 ponto) As árvores (i) e (ii) podem ser consideradas uma árvore B de ordem *n*, onde *n* > 3? Explique a sua resposta.

***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) Troque 50 com a sua sucessora 60.

60

30 33 70 90

10 20 31 32 41 50 81 89 94 100

Remova a chave 50.

60

30 33 70 90

10 20 31 32 41 81 89 94 100

Redistribua as chaves 70 e 81.

60

30 33 81 90

10 20 31 32 41 70 89 94 100

d) As árvores (i) e (ii) poderão ser consideradas uma árvore B de ordem *n*=4. De fato, em uma árvore B de ordem 4, os nós internos poderão ter um mínimo de *f*=⎡4/2⎤=2 filhos e um mínimo de   
*k*=(*f*-1)=1 chaves e as folhas poderão ter um mínimo de *k*=1 chaves.

**Questão 4 (2,5 pontos)**

a) **(0,5 ponto)** Considere a tabela de dispersão estendida, *T*, parcialmente representada abaixo. Quantos apontadores em *T* apontam para o bloco *B1* ? Caso haja mais de um apontador, além do ilustrado na figura, em quais posições da tabela *T* eles estarão armazenados? Explique sua resposta.

*T*

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

b) **(0,5 ponto)** Considere a tabela de dispersão estendida representada na figura abaixo. Suponha que cada bloco possa conter até 3 chaves. Mostre como a estrutura ficará após a remoção de *K21*, assumindo que a estrutura deve economizar espaço sempre que possível. Explique sua resposta.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *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* | |
|  |  |  |  |  | |

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

d) **(1,0 ponto)** Suponha que cada bloco possa conter um mínimo de 2 chaves e um máximo de 3 chaves. Assuma que a tabela possua 8 entradas. Qual o número mínimo de chaves que uma tabela de dispersão estendida com esta configuração pode conter? Explique sua resposta.

***Resposta***

a) Com *i = 3* e *j1 = 2*, há *p = 2(3-2)= 2* apontadores para *B1* . O segundo apontador estará em *T[6]* pois as chaves mapeadas em *T[2]* e *T[6]* possuem os 2 últimos bits iguais.

b) 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.

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

c) Após a remoção de *K21*, a chave *K22* deve ser movida para *B4* e o bloco *B2* descartado. De fato, os blocos *B2* e *B4* são “companheiros”, no sentido de que as chaves mapeadas em *T[1]* e *T[3]* possuem o último bit igual.

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

d)Suponha que cada bloco possa conter um mínimo de 2 chaves e um máximo de 3 chaves. Assuma   
que a tabela possua 8 entradas. Então a estrutura deve conter um mínimo 4 blocos (ver cenário abaixo):

* 1 bloco apontado por 1 posição na tabela com 2 chaves
* 1 bloco apontado por 4 posições na tabela com 2 chave
* 1 bloco apontado por 2 posições na tabela com 2 chave
* 1 bloco apontado por 1 posição na tabela com 2 chave

Assim, o número mínimo de chaves será *kmin* = 4 \* 2 = 8 chaves.

***Cenário para forçar a tabela a ter 8 entradas, com um mínimo de blocos.***

1) Estrutura é criada e K1, K2 e K3 são inseridas no Bloco 1:



2) K4 seria inserida no Bloco 1, causando a duplicação da tabela, a alocação de um novo bloco, Bloco 2, e a redistribuição das chaves entre os Blocos 1 e 2.



3) K5 e K6 seriam inseridas no Bloco 1, causando a duplicação da tabela, a alocação de um novo bloco, Bloco 3, e a redistribuição das chaves entre os Blocos 1 e 3.



4) K7 e K8 seriam inseridas no Bloco 1, causando a duplicação da tabela e a alocação de um novo bloco, Bloco 4, e a redistribuição das chaves entre os Blocos 1 e 4.

