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)** 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, indicando os fatores de balanceamento dos nós que provocam as rotações e a forma da árvore a cada rotação **(sem esta explicação a questão não será corrigida)**.

(a) (1,0 ponto) Operação: **Inserção da chave 6**.

5

3

10

12

7

(b) (1,0 ponto) Operação: **Remoção da chave 7, trocando-se 7 com a sua antecessora.**

5

7

2

10

6

1

3

9

11

(c) (0,5 ponto) Suponha que imediatamente após a inserção da chave 6 no item (a), seja inserida a chave 4. Baseando-se neste cenário, explique se, e como, seria possível otimizar o processo de inserção.

***Resposta***

(a) Inserção de 6.

2

5

-1

3

10

12

7

6

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:

2

5

3

7

10

1

6

12

* ***Rotação à esquerda*** a partir do nó rotulado com 5, resultando na árvore:

0

5

3

7

10

12

6

0

b) Remoção de 7:

5

2

7

1

3

10

6

11

9

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

5

2

6

1

3

10

11

9

2

0

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

-1

6

0

9

(c) Suponha que não sejam feitas as rotações indicadas no item (a) após a inserção da chave 6. Note que, após inserir a chave 4, a árvore volta a ficar balanceada. Em geral, seria possível otimizar o processo de inserção atrasando-se o processo de relanceamento após uma inserção e esperando-se pela próxima inserção, que pode rebalancear a árvore sem necessidade de rotações.

**Questão 2 (2,5 pontos)** Considere a árvore B de ordem 5 abaixo:

100

53 77 123 200

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

Realize as seguintes operações, indicando as transformações (divisão, redistribuição ou concatenação) executadas e a forma da árvore após cada transformação:

1. (0,5 ponto) Inserção de 243 na árvore original.
2. (1,0 ponto) Remoção de 123 da árvore original, trocando-se 123 pela sua antecessora.
3. (1,0 ponto) A troca de 123 pela sua antecessora, exigida no item (b), seria a melhor forma de remover 123? Explique a sua resposta exibindo a alternativa escolhida e comparando-a com a alternativa do item (b).

***Resposta***

a) Inserção de 243.

Pesquise o nó onde a chave 243 deve ser inserida.

100

53 77 123 200

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

***243***

Divida o nó e insira a chave do meio (242) no pai.

100

53 77 123 200

10 40 60 70 80 90 110 113 130 138 140 170 230 236 242 243 245

100

53 77 123 200 242

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

b) A chave 123 ocorre em um nó interior e deve ser trocada com a sua antecessora, conforme pedido.

100

53 77 123 200

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

Trocando-se 123 com a antecessora 113, e removendo-se 123, temos:

100

53 77 113 200

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

Removendo-se 123, tem-se um nó com menos chaves do que o número permitido (2 chaves). Redistribuindo-se então as chaves do irmão da direita do nó afetado, obtém-se:

100

53 77 130 200

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

c) A chave 123 também pode ser trocada com a sua sucessora. De fato, trocando-se 123 com a sucessora 130, e removendo-se 123, temos:

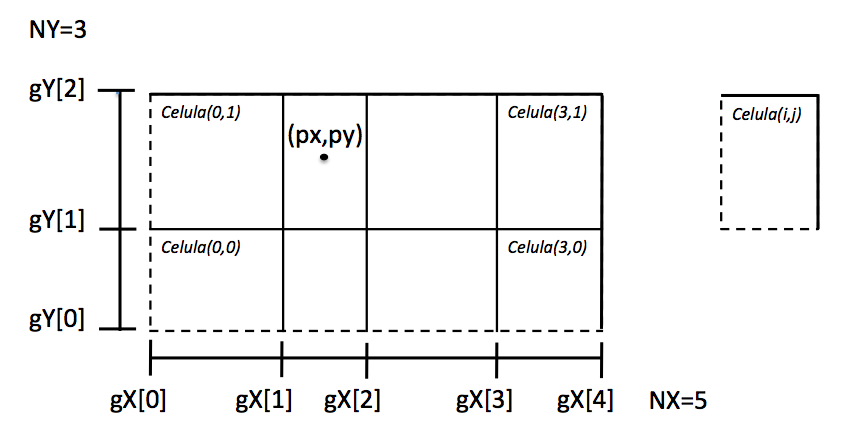
100

53 77 130 200

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

Note que esta alternativa é melhor do que a considerada no item (b) pois não necessita outras transformações para rearrumar a árvore-B.

**Questão 3 (2,5 pontos)** Considere uma grade não regular bidimensional tal que (ver figura abaixo): as bordas lateral esquerda e inferior não pertencem à grade; para cada célula, as bordas lateral esquerda e inferior não pertencem à célula.



Cada célula da grade armazena uma lista encadeada de pontos.

Considere os seguintes tipos que representam a grade:

typedef struct lista Lista;

struct lista {

float x, y; /\* ponto na lista \*/

Lista\* prox; /\* ponteiro para próximo elemento da lista \*/

};

typedef struct grade Grade;

#define NX 200 /\* número de elementos do vetor gX \*/

#define NY 135 /\* número de elementos do vetor gY \*/

struct grade {

float gX[NX]; /\* vetor que define as divisões no eixo X da grade \*/

float gY[NY]; /\* vetor que define as divisões no eixo Y da grade \*/

Lista\* prim[NX][NY]; /\* lista por célula (inicializada com NULL) \*/

};

Implemente uma função que receba como entrada uma grade *g*, com a definição acima, e um ponto *(px, py)*, como na figura, e pesquise se o ponto ocorre na grade. A função deve seguir o seguinte protótipo:

int pesq(Grade\* g, float px, float py);

e deve retornar:

0 se o ponto não existir na grade, ou estiver fora da grade

1 se o ponto existir na grade

***Resposta***

int pesq(Grade\* g, float x, float y)

{ int i, j;

Lista\* p;

if (x <= g->gX[0] || x > g->gX[NX-1] ||

y <= g->gY[0] || y > g->gX[NY-1]) return 0;

for(i = 1; i < NX && x > g->gX[i]; i++) /\* calcula posição x da célula \*/

for(j = 1; j < NY && y > g->gY[j]; j++) /\* calcula posição y da célula \*/

p = g->prim[i-1][j-1];

while(p != NULL && (p->x != x || p->y != y)) { /\* procura ponto na lista \*/

p = p->prox;

}

if (p == NULL) /\* se não achou o elemento na lista \*/

return 0; /\* retorna 0 \*/

else return 1; /\* senão retorna 1 \*/

}

**Questão 4 (2,5 pontos)** Considere uma tabela de dispersão estendida, parcialmente representada na figura abaixo, onde

i = número de bits de relevância

h = função de hash

H = tabela de hash, armazenando apenas ponteiros para os buckets

B = um bucket arbitrário

jB = informação adicional sobre B

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

i

H

|  |
| --- |
|  |
|  |
|  |

jB

B

* 1. (0,5 ponto) Seja k = h(K) o valor da função de hash para uma chave K. Escreva uma expressão em C que mapeia k e i em um valor x tal que H[x] aponta para o bucket onde K deve ser armazenada.
  2. (0,5 ponto) Suponha que o bucket B esteja completo e que existam exatamente 2 entradas de H, H[x] e H[y], apontando para B e que x < y. Ao inserir uma nova chave K em B, haverá overflow de B e um novo bucket B’ será criado; H[x] continuará apontando para B e H[y] apontará para B’.

Sob que condições a criação de B’ não resolverá o problema do overflow do bucket B? Explique sua resposta.

* 1. (0,5 ponto) Quando a situação em (b) ocorre, explique como a estrutura deve ser atualizada para resolver o problema.
  2. (1,0 ponto) Suponha que N seja o número máximo de bits de relevância que podem ser considerados. Explique qual novo problema poderá ocorrer no item (c) e como deve ser resolvido.

***Resposta***

a) Assuma que os bits de relevância sejam os bits mais à direita. A chave K tal que k=h(K) deve ser inserida no bucket apontado pela entrada H[x], onde x = (k % pow(2.0,i)).

Esta expressão está correta, mesmo quando i=0. Neste caso, x = (k % pow(2.0,0)) = 0.

b) Seja ***K*** o conjunto de chaves originalmente em B, acrescido da chave K. A criação de B’ não resolverá o problema de overflow quando cada chave L em ***K*** satisfizer à expressão   
(h(L) % pow(2.0,i)) = x. Ou, equivalentemente, quando nenhuma chave L em ***K*** satisfizer à expressão (h(L) % pow(2.0,i)) = y.

c) Quando a situação em (b) ocorre, a tabela de hash H deve ser duplicada, o número de bits de relevância incrementado de 1 e nova tentativa de mover chaves de B para B’ deve ser realizada. Este processo deve ser repetido até seja possível armazenar todas as chaves em ***K***.

d) Mesmo aumentando o número de bits considerado, o processo descrito no item (c) pode não ser suficiente para resolver o problema de overflow, quando o limite N do número de bits de relevância é atingido. De fato, todas as chaves em ***K*** podem possuir os mesmos N últimos bits.

Uma estratégia de encadeamento de buckets deverá então ser implementada.