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

|  |  |  |
| --- | --- | --- |
| 1a) | 1.0 |  |
| 2a) | 3.0 |  |
| 3a) | 3.0 |  |
| 4a) | 3.0 |  |
|  | 10.0 |  |

1. A prova é individual e sem consulta. Qualquer tentativa de “cola” resultará na anulação da prova de ambos os alunos envolvidos.
2. A interpretação faz parte da questão. Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
3. O tempo de prova é 1:45 h.
4. As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
5. A prova pode ser feita a lápis.

1) (1.0 ponto) Considere uma árvore de busca *A* com altura *h*. Prove qual é o menor e o maior número de chaves que podem ser armazenadas em *A*, se *A* for:

(a) (0.5 ponto) Uma árvore de busca binária.

(b) (0.5 ponto) Uma árvore-B de ordem 4.

As suas respostas devem ser cuidadosamente elaboradas.

Resposta:

(a) Uma árvore de busca binária de altura *h* tem no mínimo *n*=*h+1* chaves, quando cada nó só tiver 1 filho. A árvore terá no máximo *N=2h+1-1* chaves, quando cada nó tiver 2 filhos.

(b) Observe que uma árvore-B de ordem 4 satisfaz às seguintes propriedades:

* A raiz tem no mínimo 1 chave e 2 filhos.
* Todos os nós internos tem no mínimo ⎡4/2⎤=2 filhos e (⎡m/2⎤ -1)=1 chave.
* Todas as folhas tem no mínimo (⎡m/2⎤ -1)=1 chave.

Logo, a árvore-B de ordem 4 degenera em uma árvore de busca binária completa e balanceada. Assim, o número mínimo de chaves será:

*n = 2 x ⎡m/2⎤ h -1 = 2 x 2h-1 = 2h+1-1*

Observe ainda que uma árvore-B de ordem 4 com o maior número de chaves satisfaz às seguintes propriedades:

* A raiz e todos os nós internos tem 4 filhos e 3 chaves.
* Todas as folhas tem 3 chaves.

Logo, o número máximo de chaves de uma árvore-B de ordem 4 será:

*N = 3 x (1 + 4 + 42 +...+4h) = 4h+1-1*

2) (3.0 pontos) A *largura* de uma árvore de busca binária é definida como a diferença entre o maior e o menor valor dentre os valores de chave na árvore.

(a) (1,0 ponto) Implemente de forma não recursiva, visitando o menor número possível de nós, uma função que calcule a largura de uma árvore de busca binária. Por simplicidade, assuma que os valores de chave são inteiros. A função deve receber como entrada um apontador para a raiz da árvore e retornar a largura da árvore:

int abb\_largura (Abb\* r)**;**

Adote a seguinte estrutura para os nós:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* esq;

Abb\* dir;

};

(b) (2.0 pontos) Prove qual é o número mínimo e o número máximo de comparações necessárias para computar a largura de uma árvore de busca binária com *n* chaves.

Resp:

a)

int abb\_largura (Abb\* r){

Abb\* min, max;

if (r==NULL) return 0;

min = r;

while(min->esq != NULL)

min = min->esq;

max = r;

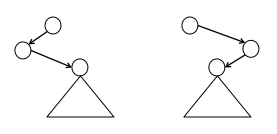
while(max->dir != NULL)

max = max->dir;

return (max->chave – min->chave);

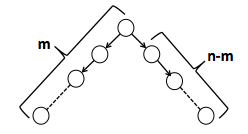
}

O número mínimo *k* de comparações ocorrerá quando a árvore tiver uma das duas configurações:



Neste caso, a árvore o número de comparações será igual a *k=3*.

O número máximo *K* de comparações ocorrerá quando a árvore estiver degenerada, da seguinte forma:



Logo, serão necessárias *K=m+1+(n-m)=n+1.*

3) (3.0 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 os nós que sofrem modificações (divisão, redistribuição ou concatenação) após cada operação:

1. (1.0 ponto) Inserção de 143 na árvore original.
2. (1.0 ponto) Remoção de 140 da árvore resultante do item (a).
3. (1.0 ponto) Remoção de 140 da árvore resultante do item (a), utilizando uma segunda alternativa para remover 140.

Resposta:

a) 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

b) Remoção de 140.

A chave 140 ocorre em um nó interior e portanto deve ser trocada com a sua sucessora (ou antecessora).

100

53 77 123 140 200

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

Trocando-se 140 com a sucessora 143, temos:

100

53 77 123 143 200

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

Removendo-se 140, tem-se um nó com menos chaves do que o número permitido (2 chaves):

100

53 77 123 143 200

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

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

100

53 77 123 143 230

10 40 60 70 80 90 110 113 130 138 170 200 236 243 245

c) Trocando-se 140 com a antecessora 138, temos:

100

53 77 123 138 200

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

Removendo-se 140, tem-se um nó com menos chaves do que o número permitido (2 chaves):

100

53 77 123 138 200

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

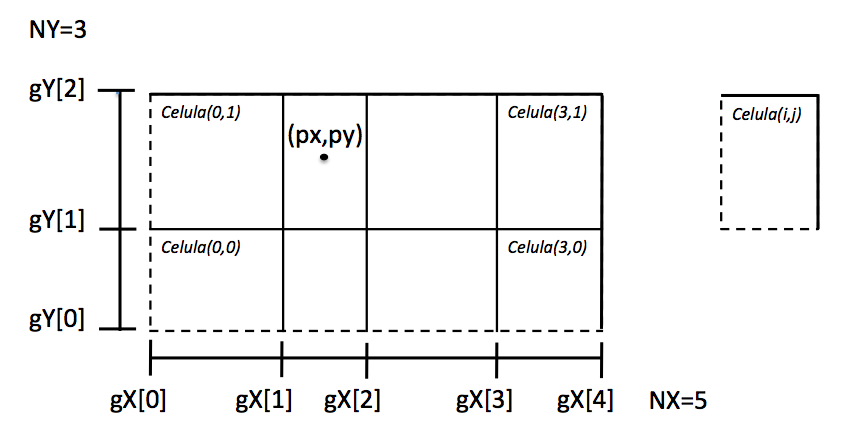
Combinando-se então o nó afetado com o seu irmão da esquerda (poderia ser o da direita), obtém-se:

100

53 77 138 200

10 40 60 70 80 90 110 113 123 130 143 170 230 236 243 245

4) (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 remova o ponto na grade. A função deve seguir o seguinte protótipo:

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

e deve retornar:

0 se o ponto não existir na grade

1 se o ponto foi removido

Resposta:

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

{ int i, j;

Lista\* ant = NULL;

Lista\* p;

if (x <= gX[0] || x > gX[NX-1] || y <= gY[0] || y > gX[NY-1]) return 0;

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

for(j = 1; j < NY && y > 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 \*/

ant = p;

p = p->prox;

}

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

return 0; /\* retorna 0 \*/

if (ant == NULL) /\* se achou o elemento \*/

g->prim[i-1][j-1] = p->prox; /\* retira do início da lista \*/

else ant->prox = p->prox; /\* ou do meio da lista \*/

free(p); /\* libera espaço ocupado pelo elemento \*/

return 1;

}