Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2.5 |  |
| 2a) | 2.5 |  |
| 3a) | 2.5 |  |
| 4a) | 2.5 |  |
|  |  |  |

1. **A prova é individual e sem consulta.**
2. **A interpretação faz parte da questão.**
3. **O tempo de prova é 1:45 h.**
4. **As respostas devem seguir as questões.**
5. **Caso precise de rascunho use o verso da folha.**
6. **Caso parte da resposta esteja no verso, indique claramente este fato.**
7. **(2.5 pontos)**
   1. **(2.0 pontos)** Complete o código da função, parcialmente apresentado abaixo, para que realize inserção de chaves em uma tabela de dispersão, utilizando desempate interno de colisões:

**#define MAX 8 /\* tamanho da tabela de hash \*/**

**#define DEC 2**

**#define VAZIO (-1)**

**int hash[MAX];**

**int insere (int x)**

**{**

**int k, pos;**

**pos = x % MAX;**

**(COMPLETAR)**

**return (-1); /\* não inseriu \*/**

**}**

b) **(0.5 ponto)** Escreva a expressão matemática da função de dispersão H(x,k) implicitamente utilizada na rotina acima. Por que H(x,k) não é uma boa função de dispersão? Justifique sua resposta.

Resp.:

**2) (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

B = um bucket arbitrário

jB = informação adicional sobre B

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

i

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

jB

H

B

* 1. **(2.0 pontos)** Suponha que o bucket *B* está completo. Explique todos os casos de uma inserção de uma nova chave *K* em *B*, supondo que possam haver chaves duplicadas
  2. **(0.5 ponto)** Suponha que cada bucket contenha *b* chaves. Qual o número mínimo, *nB*, de buckets e o tamanho mínimo, *hmin*, da tabela *H* para acomodar *N* chaves, supondo que não hajam chaves duplicadas?

Resp.:

**3)** **(2.5 pontos)**

a) **(0.5 ponto)** Insira a chave 31 na árvore B de ordem 4 abaixo:

50

30 70 90

10 20 39 40 60 80 100

Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

b) **(0.5 ponto)** Insira a chave 31 na árvore 2-3 abaixo:

50

30 70 90

10 20 39 40 60 80 100

Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

d) **(0.5 ponto)** Remova a chave 70 da árvore B de ordem 4 abaixo:.

50

30 33 70 90

10 20 31 32 40 41 55 60 81 80 94 100

Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

c) **(1.0 ponto)** Remova a chave 70 da árvore B de ordem 3 abaixo:.

50

30 70 90

10 20 40 60 80 100

Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

Resp.:

**4) (2.5 pontos)**

* 1. **(0.5 ponto)** Explique como determinar quantas componentes conexas um grafo não dirigido possui. Use a figura abaixo para ilustrar a sua solução.

0

1

2

3

4

5

6

7

* 1. **(2.0 pontos)** Implemente uma função em C que determine o número de componentes conexas de um grafo não dirigido. A função deve ter o seguinte protótipo:

int nconnected(Graph\* G);

Considere que o grafo está representado como listas de adjacências:

typedef struct graph Graph;

typedef struct listNode ListNode;

struct listNode {

int vertex;

ListNode\* link;

};

struct graph {

int nv; /\* número de vértices no grafo \*/

int\* vis; /\* vis[i]=1 sse vertice i foi visitado \*/

ListNode\*\* vv; /\* vv[i] aponta p/ lista de vértices adjacentes a i \*/

};

Resp.: