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

|  |  |  |
| --- | --- | --- |
| 1a) | 2,0 |  |
| 2a) | 2,0 |  |
| 3a) | 3,0 |  |
| 4a) | 3,0 |  |
|  | 10,0 |  |

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 durante toda a prova. **Aparelhos celulares ligados 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** **(2,0 pontos).** Uma tabela de dispersão (*hash table*) de tamanho 11 é implementada com *encadeamento externo* através da seguinte função de dispersão:   
Nela são inseridas 8 dados que possuem as seguintes chaves de busca (nesta ordem):

7,10,15,14,17,3,21,25

1. (0,5 ponto) Desenhe a estrutura de dados após a inserção destas chaves.
2. (1,0 ponto) Explique, com base no exemplo anterior, quais são os casos que devem ser considerados para implementar a operação de remoção de uma chave x:

remocao(x)

Entrada: um valor x de chave

Saída: NULL, se x não é encontrada

p, ponteiro para o elemento que contém x

1. (0,5 ponto) Qual a complexidade temporal, no pior caso, da operação de remoção? Explique sua resposta.

**Resposta**

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *x* | *h(x)* |  |  |  |  |
| 7 | **7** |
| 10 | **10** |
| 15 | **4** |
| 14 | **3** |
| 17 | **6** |
| 3 | 3 |
| 21 | 10 |
| 25 | 3 |

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |  | 14 |  | 3 |  | 25 |
| 4 |  |  | 15 |
| 5 |  |  |  |
| 6 |  |  | 17 |
| 7 |  |  | 7 |
| 8 |  |
| 9 |  |
| 10 |  |  | 10 |  | 21 |

1. Há 4 casos a considerar:

Caso 1: x não está na estrutura. Devolva NULL.

Caso 2: x está no único elemento de uma lista. Remova x e coloque NULL na entrada da tabela.

Caso 3: x está no primeiro elemento p de uma lista. Remova x e faça a tabela apontar para o elemento seguinte a p na lista.

Caso 4: Nenhum dos casos acima. Remova x da lista a que pertencia, como usual.

1. No pior caso, a função de remoção é *O*(*n*), onde *n* é o número de chaves na tabela, já que a tabela pode degenerar em uma única lista encadeada contendo todas as chaves.

**Questão 2** **(2,0 pontos).** Seja um conjunto de pessoas. Assuma que, em um dado instante, cada pessoa tem um único time de futebol preferido, mas que este time pode mudar ao longo do tempo. Considere o problema de agrupar as pessoas pelo seu time de futebol preferido de tal forma que seja rápido determinar se duas pessoas tem o mesmo time de futebol preferido ou não.

a) (0,5 ponto) Qual estrutura de dados, dentre as apresentadas em sala, seria mais eficiente para resolver o problema acima? Explique sua resposta.

b) (1,5 pontos) Defina em C a estrutura de dados escolhida no item (a) e implemente uma rotina que recebe como entrada a representação de duas pessoas e retorna 1, se as duas pessoas tem o mesmo time de futebol preferido, e 0, em caso contrário. A rotina não deverá chamar as operações definidas em sala para a estrutura escolhida.

**Resposta**

a) A estrutura mais apropriada seria uma partição dinâmica de conjuntos, implementada como uma floresta com ranking nos nós, com balanceamento de união e compressão de caminhos no Find, como no algoritmo Union-Find. Para determinar se duas pessoas tem o mesmo time de futebol preferido, basta realizar o Find para cada uma delas e verificar se os Finds retornam raízes iguais.

b)

forest\_node\* Find\_same(forest\_node\* node1, node2) {

forest\_node\* temp;

forest\_node\* root1 = node1;

forest\_node\* root2 = node2;

while (root1-> parent != NULL) {

root1 = root1-> parent;

}

while (node1->parent != root1) {

temp = node1->parent;

node1->parent = root1;

node1 = temp;

}

while (root2-> parent != NULL) {

root2 = root2-> parent;

}

while (node2->parent != root2) {

temp = node2->parent;

node2->parent = root2;

node2 = temp;

}

return root1 == root2;

}

**Questão 3** **(3,0 pontos).** Considere a árvore B de ordem 5 da Figura 1.

100

53 77 123 200

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

**Figura 1.** Uma árvore B.

Realize as operações dos item (a) e (b), utilizando sempre como árvore inicial a árvore da Figura 1. Indique os nós que sofrem modificações após cada operação, bem como a ocorrência de divisão, redistribuição ou concatenação:

1. (1,0 ponto) Inserção de 243.
2. (1,5 pontos) Remoção de 40.
3. (0,5 ponto) A árvore B da Figura 1 pode ser considerada uma árvore B de ordem distinta de 5? Explique sua resposta.

**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) Remoção de 40.

A chave 40 ocorre em uma folha e portanto pode ser removida.

100

53 77 123 200

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

100

***53*** 77 123 200

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

Concatene nós, com transferência de chave do pai para o nó à esquerda:

***100***

77 123 200

10 53 60 70 80 90 110 113 130 138 140 170 230 236 242 245

Concatene a raíz e os seus dois filhos:

77 100 123 200

10 53 60 70 80 90 110 113 130 138 140 170 230 236 242 245

1. Sim, a árvore pode ser considerada de ordem m=6, por exemplo, já que uma árvore B de ordem 6 pode ter no mínimo ⎡m/2⎤=⎡6/2⎤=3 filhos e no máximo 6 filhos, exceto a raiz, que pode ter no mínimo 2 filhos e no máximo 6 filhos. Além disto, todas as folhas tem no mínimo   
   (⎡m/2⎤–1)=(⎡6/2⎤–1)=2 chaves.

**Questão 4** **(3,0 pontos).**

0

1

2

3

4

5

6

7

**Figura 2.** Exemplo de um grafo.

Considere a representação de grafos por listas de adjacência em C abaixo:

typedef struct graph Graph;

typedef struct listNode ListNode;

struct listNode {

int vertex;

ListNode\* link;

};

struct graph {

int nv; /\* número de vértices do grafo \*/

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

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

};

* 1. (2,0 ponto) Implemente uma função em C que retorne o número de componentes conexas de um grafo não dirigido, utilizando como subrotina a rotina de busca em profundidade em grafos, modificada se for necessário. A função deve ter o seguinte protótipo:

int nconnected(Graph\* g);

onde g é um grafo representado como acima.

* 1. (1,0 ponto) Usando o grafo da Figura 2 como exemplo, argumente porque o seu algoritmo está correto.

**Resposta**

a)

0

1

2

3

4

5

6

7

a) (g->vis[n] = j indica que o nó n pertence à j-ésima componente conexa do grafo)

void dfs(Graph\* g, int N, int j){

ListNode\* w;

g->vis[N] = j;

for (w = g->vv[N]; w != NULL; w = w->link)

if (g->vis[w->vertex] == 0) dfs(g, w->vertex, j);

}

int nconectado(Graph\* g){

int i, cont=1;

for (i=1; i < g->nv; i++) g->vis[i]=0; /\* inicializa vis \*/

for (i=0, i < g->nv; i++)

if (g->vis[i] == 0){

dfs(g,i,cont);

cont++;}

return cont;

}  
b) O procedimento implementado em (a) opera da seguinte forma:

1. Marque todos os nós como não visitados.
2. Se há algum nó não foi visitado, escolha um nó não visitado e inicie novamente o procedimento em (2) para determinar a segunda componente conexa, e assim por diante.
3. Pare quando todos os nós tenham sido visitados. O número de componentes conexas será o número de vezes que o procedimento em (2) foi executado.

Este procedimento está correto pois o passo (2) marca todos os nós de cada componente conexa com um inteiro distinto, cont. No passo (3), o valor final da variável cont é retornado. Este valor será o número de componentes conexas.

Por exemplo, escolhendo-se o nó 0 como nó inicial, o passo (2) marcará com “1” os nós 1, 2 e 4. Em seguida, escolhendo-se o nó 3 como nó inicial, o passo (2) marcará com “2” apenas o nó 3. Por fim, escolhendo-se o nó 5 como nó inicial, o passo (2) marcará com “3” os nós 5, 6 e 7. O passo (3) retornará então “3” como o número de componentes conexas do grafo.