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

|  |  |  |
| --- | --- | --- |
| 1a) | 2,0 |  |
| 2a) | 3,0 |  |
| 3a) | 2,0 |  |
| 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) Considere um município onde os veículos são identificados por placas. Cada placa é uma cadeia de símbolos de comprimento 8, onde cada símbolo ou é uma letra do alfabeto ou é um dos 10 dígitos. Por exemplo, “X9Y0BCD9” é uma possível placa. Porém, nem todas as possíveis combinações de letras e números são de fato utilizadas.

Considere uma tabela de hash para representar as placas de veículos que de fato estão em uso.

a) (0,5 ponto) Suponha que o município possua hoje 100.000 veículos. Qual deve ser o tamanho, *tam*, da tabela de hash? Explique sua resposta.

b) (1,0 ponto) Escreva em C uma função de dispersão (função *hash*) que receba como argumentos:

* um *string* de comprimento 8, *cod*, representando a placa de um veículo
* um inteiro *tam* que define o tamanho da tabela *hash*

e retorne um inteiro que identifica a entrada na tabela de *hash* para a qual *cod* é mapeada.

Argumente porque a sua implementação está correta e representa uma função de hash adequada.

c) (0,5 ponto) Suponha que se espera que o número de veículos cresça 20% nos próximos 5 anos. Diante desta nova informação, o que deve ser modificado nas letras (a) e (b) para que não seja necessário reconstruir a estrutura de dados nos próximos 5 anos?

**Resposta**

**Questão 2** (3,0 pontos)

Considere a representação de conjuntos utilizando vetores de bits, definidos como:

struct \_bitvector {

int max; /\* número máximo de elementos do conjunto \*/

int \*vector;

};

typedef struct \_bitvector BitVector;

a) (1,5 ponto) Implemente em C uma rotina que recebe como entrada dois conjuntos *A* e *B* e devolve o conjunto *D* = *A* – *B*. A rotina não deverá chamar as operações definidas em sala para vetores de bits.

Nota: *A* – *B*, a *diferença* de *A* e *B*, é o conjunto de todos os elementos de *A* que não estão em *B*.

b) (0,5 ponto) Qual a complexidade, no pior caso, da rotina implementada no item (a), medida em termos do tamanho dos vetores de bits. Explique sua resposta.

c) (0,5 ponto) Assuma que os conjuntos *A* e *B* estão representados como tabelas de hash. Explique como seria implementada a operação de diferença. Esta implementação seria mais ou menos eficiente do que a descrita no item (a)?

d) (0,5 ponto) Qual seria a complexidade, no pior caso, desta segunda implementação da operação de diferença, medida em termos do número de elementos dos conjuntos. Explique sua resposta.

**Resposta**

**Questão 3** (2,0 pontos) Considere a representação de partições por florestas, com as otimizações da operação de union através do balanceamento por altura e da operação de find por compressão de caminhos. Caso haja empate na operação de union, escolha o menor inteiro para raíz.

1. (1,5 ponto) Considere o conjunto {1,2,3,4,5}. Começando com a floresta representando a partição {{1},{2},{3},{4},{5}}, desenhe a sequência de florestas obtidas sucessivamente pelas operações:

union(1,2), union(3,4), union(3,5), find(4), union(1,3), find(4), find(4)

1. (0,5 ponto) Quantas comparações são realizadas, ao todo, pelas 3 execuções da operação find(4) da sequencia acima? Explique a sua resposta.

**Resposta**

**Questão 4** (3,0 pontos)

a) (1,0 ponto) Considere o conjunto de prioridades *P* = { 6.2, 7.2, 7.1, 6.0, 7.1 }. Mostre como construir um *max heap*, utilizando um vetor, da forma mais eficiente possível, para organizar este conjunto de prioridades.

b) (0,5 ponto) Suponha que o elemento com a segunda maior prioridade do *heap* esteja na segunda posição do vetor. É possível remover do *heap* o elemento com a segunda maior prioridade copiando a terceira posição do vetor para a segunda, a quarta para a terceira e assim por diante? Explique a sua resposta.

c) (1,5 ponto) Escreva em C uma função que recebe como argumento um *heap*, organizado como no item (a), e remove do *heap* o elemento com a segunda maior prioridade, usando o menor espaço e o menor tempo possível. A rotina deve retornar 1 se a alteração ocorreu corretamente e 0 em caso contrário. Assuma que o *heap* seja definido como abaixo:

struct \_heap {

int max; /\* tamanho máximo do heap \*/

int pos; /\* próxima posicao disponível no vetor \*/

float\* prioridade; /\* vetor das prioridades \*/

};

typedef struct \_heap Heap;

Assuma ainda que a assinatura da rotina seja:

int heap\_altera(Heap\* heap)

A rotina não deve chamar as operações de inserção ou remoção de um heap apresentadas em sala, mas poderá usar uma das operações auxiliares apresentadas em sala.

**Resposta**