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

|  |  |  |
| --- | --- | --- |
| 1a) | 1.0 |  |
| 2a) | 3.0 |  |
| 3a) | 3.0 |  |
| 4a) | 2.0 |  |
| 5a) | 1.0 |  |
|  | 10.0 |  |

1. A prova é individual e sem consulta.
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.

**Questão 1** (1.0 pontos):

a) (0,5 ponto) Seja *T* uma tabela de dispersão (*hash table*) implementada com *encadeamento externo*. Assuma que *T* possui tamanho *n* e que a função de dispersão seja dada por:

Assuma que o total de elementos a serem inseridos seja de aproximadamente 600. Qual valor de *n* você escolheria? Explique sua resposta.

b) (0,5 ponto) Seja *T* uma tabela de dispersão (*hash table*) implementada com *encadeamento interior*. Assuma que *T* possui tamanho *n*. Defina uma função de dispersão adequada e simples (não é necessário escrever a função em C; basta defini-la matematicamente). Explique sua resposta.

Resp:

a) *n* = 23 x 29 = 667 seria um valor adequado pois é aproximadamente 10% a mais do que 600 e possui fatores primos grandes, maiores do que 20.

b) Uma boa função de dispersão seria

Ou seja, *h(x,j)* retorna a posição onde a chave *x* deve ser inserida, se possível, na *j-ésima* tentativa de inserir *x*. Neste caso, *p* não deve ser um submúltiplo de *n*.

**Questão 2** (3.0 pontos): Escreva um rotina em C que receba como entrada um *max heap* e uma percentagem *p*, modifique a prioridade *x* do maior elemento do heap para *p.x* e reorganize a estrutura para que mantenha as propriedades de um *max heap*. A rotina deve retornar 1 se a alteração ocorreu corretamente e 0 em caso contrário.

A sua implementação deve fazer o menor número possível de comparações e movimentações (e não deve usar as operações sobre heaps pré-definidas).

Assuma que o *max heap* seja definida como abaixo:

struct \_heap {

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

int pos; /\* proxima posicao disponivel no vetor \*/

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

};

typedef struct \_heap Heap;

Assuma ainda que a assinatura da rotina seja:

int heap\_altera(Heap\* heap, float p)

Resp:

int heap\_altera(Heap\* heap, float p)

{

if (heap->pos>0) {

heap->prioridade[0]=heap->prioridade[0]\*p;

corrige\_abaixo(heap);

return 1;

}

else {

printf("Heap VAZIO!");

return 0;

}

}

static void corrige\_abaixo(Heap\* heap){

int pai=0;

while (2\*pai+1 < heap->pos){

int filho\_esq=2\*pai+1;

int filho\_dir=2\*pai+2;

int filho;

if (filho\_dir >= heap->pos) filho\_dir=filho\_esq;

if (heap->prioridade[filho\_esq]>heap->prioridade[filho\_dir])

filho=filho\_esq;

else

filho=filho\_dir;

if (heap->prioridade[pai]<heap->prioridade[filho])

troca(pai,filho,heap->prioridade);

else

break;

pai=filho;

}

}

**Questão 3** (3.0 pontos): Considere o problema de representar o grupo sanguíneo de um conjunto de pessoas, que pode variar ao longo do tempo, de tal forma que seja rápido determinar se duas pessoas possuem o mesmo grupo sanguíneo.

a) (1,0 ponto) Qual estrutura de dados, dentre as apresentadas em sala, seria mais eficiente para resolver o problema acima? Explique sua resposta.

c) (2,0 pontos) Defina em C a estrutura de dados escolhida no item (a) e implemente uma rotina para determinar se duas pessoas possuem o mesmo grupo sanguíneo. A rotina não deverá chamar as operações definidas em sala para a estrutura escolhida.

Resp.

a) A estrutura mais apropriada seria uma partição dinâmica de conjuntos, implementada como uma floresta com ranking nos nós e com balanceamento de união e compressão de caminhos no Find (algoritmo Union-Find). Para determinar se duas pessoas possuem o mesmo grupo sanguíneo, basta realizar o Find para cada uma delas e verificar se os Finds retornam a mesma raiz.

b)

forest\_node\* Find\_same(forest\_node\* node1, forest\_node\* 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 4** (2.0 pontos): Considere o problema de representar 2 subconjuntos de um grupo *G* de pessoas, definidos como:

*G1* = { *x* ∈ *G* / *x* tem 21 anos ou mais }

*G2* = { *x* ∈ *G* / *x* é do sexo feminino }

a) (0,5 ponto) Qual estrutura de dados, dentre as apresentadas em sala, seria mais eficiente para representar os 2 subconjuntos acima e implementar operações sobre eles? Explique sua resposta.

c) (1,5 pontos) Defina em C a estrutura de dados escolhida no item (a) e implemente uma rotina que receba como entrada *G1* e *G2* e devolva o subconjunto *H* de *G* definido como:

*H* = { *x* ∈ *G* / *x* tem 21 anos ou mais e *x* não é do sexo feminino }

A rotina não deverá chamar as operações definidas em sala para a estrutura escolhida.

Resp:

a) A estrutura mais eficiente consiste em representar os conjuntos como vetores de bits. Desta forma seria possível representar os 2 conjuntos de forma compacta e implementar operações sobre eles usando as operações Booleanas nativas da linguagem C.

b) Definição da estrutura:

struct \_bitvector {

int max;

int \*vector;

};

typedef struct \_bitvector BitVector;

BitVector\* bvH(BitVector\* g1, BitVector\* g2)

{

int i,minab=(a->max < b->max)? a->max :b->max;

int num=(minab-1)/sizeof(int)+1; /\* numero de ints para max bits \*/

BitVector\* bv = (BitVector\*)malloc(sizeof(BitVector));

bv->max = minab;

bv->vector = (int\*)malloc(num\*sizeof(int));

for (i=0; i<num; i++)

bv->vector[i] = (a->vector[i])&(~(b->vector[i]));

return bv;

}

**Questão 5** (1.0 ponto):

a) (0,5 ponto) Considere a tabela de dispersão estendida, *T*, ***parcialmente*** representada abaixo. Quantos apontadores em *T* apontam para *B1* ? Caso haja mais de um apontador, em quais posições da tabela *T* eles estarão armazenados? Explique sua resposta.

*T*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =3* |  |  |  |
| 0 |  |  |  | *B1* |
| 1 |  |  |  | *j1 =2* | |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |

b) (0,5 ponto) Considere a tabela de dispersão estendida, *T*, representada na figura abaixo. Suponha que cada bloco contenha 2 chaves, como ilustrado na figura abaixo. Mostre como a estrutura ficará após a remoção de *K31* e *K32*, assumindo que a estrutura deve economizar espaço sempre que possível. Explique sua resposta.

*T*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =2* |  |  |  |
| 0 |  |  | *K11* | *B1* |
| 1 |  |  | *K12* | *j1 =2* | |
| 2 |  |  |  |  |
| 3 |  |  | *K21* | *B2* |
|  |  |  | *K22* | *j2 =2* | |
|  |  |  |  |  |
|  |  |  | *K31* | *B3* |
|  |  |  | *K32* | *j3 =2* | | |
|  |  |  |  |  |
|  |  |  | *K41* | *B4* |
|  |  |  | *K42* | *j4 =2* | |

Resp.:

a) Com *i = 3* e *j1 = 2*, há *p = 2(3-2)= 2* apontadores para *B1* . O segundo apontador estará em *T[4]* pois as chaves mapeadas em *T[0]* e *T[4]* possuem os 2 últimos bits iguais.

b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *i =2* |  |  |  |
| 0 |  |  | *K11* | *B1* |
| 1 |  |  | *K12* | *j1 =1* | |
| 2 |  |  |  |  |
| 3 |  |  | *K21* | *B2* |
|  |  |  | *K22* | *j2 =2* | |
|  |  |  |  |  |
|  |  |  | *K41* | *B4* |
|  |  |  | *K42* | *j4 =2* | |