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

|  |  |  |
| --- | --- | --- |
| 1a) | 2,5 |  |
| 2a) | 2,5 |  |
| 3a) | 2,5 |  |
| 4a) | 2,5 |  |
|  | 10,0 |  |

**LEIA COM CUIDADO**

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 e guardados fora do alcance durante toda a prova. **Aparelhos celulares ligados ou de alguma forma visíveis 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 (2,5 pontos)**. Suponha que os carros em circulação em uma cidade sejam identificados por uma placa alfanumérica da forma CCDDDD, onde C representa um caractere não numérico e D é um dígito, de “0” a “9. Suponha que uma placa seja representada como uma cadeia de caracteres de comprimento 6, codificados em ASCII.

Considere uma tabela de *hash* para representar as placas dos carros em circulação na cidade.

a) (0,5 ponto) Suponha que a cidade possua 30.000 carros. Qual deve ser o tamanho, *tam*, da tabela de *hash*? Explique sua resposta.

b) (0,5 ponto) Escreva em C uma função de dispersão (função *hash*) que receba como argumentos:

* uma cadeia de caracteres, *placa*, representando uma placa de um carro
* 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 *placa* é mapeada.

c) (0,5 ponto) Argumente porque a sua implementação está correta e representa uma função de *hash* adequada.

d) (1,0 ponto) Suponha que o número de carros em circulação na cidade aumente em 100% e que um novo esquema de placas entre em vigor, acrescentando-se uma letra (escolhida aleatoriamente) ao início da placa. Qual seria o efeito desta nova situação na tabela de hash e o que deve ser modificado na função de *hash* e na tabela de *hash* para acomodar esta nova situação? Explique sua resposta.

**Resposta**

a) Para armazenar 30.000 placas, o tamanho da tabela de *hash* deve ser aproximadamente 10% maior, ou seja, 33.000. Podemos então tomar o tamanho da tabela de *hash* como sendo *tam* = 41\*37\*23 = 34.891.

Note que: (i) 41, 37 e 23 são números primos maiores do que 20; e (ii) *tam* > 30.000\*1,1 = 33.000. Portanto, *tam* seria um tamanho apropriado para a tabela de *hash*.

b)

1. int hash(char \*placa, int tam) {

2. int i, h=0;

3. for( i=0 ; i<6 ; i++ ) {

4. h = h + (int)( placa[i]) \* 10\*\*(3-i); }

5. return h%tam; }

c) A função de *hash* do item (b) está correta pois devolve um inteiro entre 0 e *tam*-1 (passo 5).

Ela é uma função de *hash* adequada pois considera todos os caracteres de uma placa ao computar a variável *h*. Além disto, como o tamanho da tabela possui as propriedades resumidas na letra (a), o passo 5 espalhará os códigos das placas adequadamente.

d) Ao duplicar o número de carros, a tabela de *hash* terá fator de carga *α* = 60.000/34.891 = 1,72, portanto maior do que 1. Uma nova tabela deverá então ser alocada com tamanho, digamos, *tam* = 41\*37\*47 = 71.299 (com as mesmas propriedades listadas na letra (a)).

A função de *hash* precisa ser modificada pois o tamanho da placa mudou. Além disto, cada placa deverá sr removida da tabela original e reinserida na nova tabela, usando a função de *hash* com segundo parâmetro igual a *tam* = 71.299. Para tal é necessário varrer toda a tabela original, sem usar a função de *hash*.

**Questão 2** **(2,5 pontos)** Considere a seguinte sequencia de inteiros: 90, 60, 30, 15, 45.

* 1. (1,0 ponto) Mostre, passo a passo, como o vetor armazenando um *heap* *mínimo* é construído pela inserção sucessiva destes 5 elementos, na ordem dada. Comente brevemente cada passo do algoritmo de inserção.
  2. (1,0 ponto) Mostre, passo a passo, como o vetor armazenando um *heap* *mínimo* é construído, para estes mesmos 5 elementos, mas usando o algoritmo eficiente para construção de heaps. Comente brevemente cada passo do algoritmo.
  3. (0,5 ponto) Os heaps resultantes do item (a) e do item (b) precisam ser iguais? Explique sua resposta.

Resposta:

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **Explicação** |
| 90 |  |  |  |  | Acrescente 90 depois do final do heap (que está vazio). | |
| 90 | 60 |  |  |  | Acrescente 60 depois do final do heap. | |
| 60 | 90 |  |  |  | Compare com o pai ((1-1)/2=0) e troque. | |
| 60 | 90 | 30 |  |  | Acrescente 30 depois do final do heap. | |
| 30 | 90 | 60 |  |  | Compare com o pai ((2-1)/2=0) e troque. | |
| 30 | 90 | 60 | 15 |  | Acrescente 15 depois do final do heap. | |
| 30 | 15 | 60 | 90 |  | Compare com o pai ((3-1)/2=1) e troque. | |
| 15 | 30 | 60 | 90 |  | Compare com o pai ((1-1)/2=0) e troque. | |
| 15 | 30 | 60 | 90 | 45 | Acrescente 45 depois do final do heap. | |
| 15 | 30 | 60 | 90 | 45 | Compare com o pai ((4-1)/2=1) e não troque. Pare. | |

b)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **Explicação (n=5)** |
|  |  | 90 | 60 | 30 | Aloque os n/2+1 primeiros elementos que ocorrem no conjunto | |
|  | 15 | 90 | 60 | 30 | Coloque o próximo elemento, 15, na posição n/2-1 | |
|  | 15 | 90 | 60 | 30 | Compare 15 com os seus filhos, nas posições 2i+1 e 2i+2 (com i=1).  Não é preciso trocar pois 15<60 e 15<40 | |
| 45 | 15 | 90 | 60 | 30 | Coloque o próximo elemento, 45, na posição 0 | |
| 15 | 45 | 90 | 60 | 30 | Compare 45 com os seus filhos, nas posições 2i+1 e 2i+2 (com i=0).  É preciso trocar 45 com 15 pois 15<45 e 15<90 | |
| 15 | 30 | 90 | 60 | 45 | Compare 45 com os seus filhos, nas posições 2i+1 e 2i+2 (com i=1).  É preciso trocar 45 com 30 pois 30<45 e 30<60 | |

c) Os heaps não precisam ser iguais. Por definição de min heap, basta que a prioridade do pai seja menor do que a dos filhos, mas a prioridade do filho à esquerda pode ser maior ou menor do que a do filho à direita.

**Questão 3 (2,5 pontos)**. Considere a árvore B de ordem 5 abaixo:

100

53 77 123 140 200

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

Realize as seguintes operações, indicando os nós que sofrem modificações (divisão, redistribuição ou concatenação) após cada operação:

1. (1,5 pontos) Remoção de 140 da árvore acima.
2. (1,0 ponto) Remoção de 140 da árvore acima, utilizando uma segunda alternativa para remover 140.

**Resposta**

a) 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

b) 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:

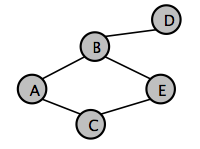
100

53 77 138 200

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

**Questão 4 (2,5 pontos).**

a) (0,5 ponto) Mostre como algoritmo de Dijkstra computará as distâncias do nó D para todos os outros nós do grafo da Figura 1. Use a Figura 2 como base para a sua resposta.



**Figura 1.** Grafo não dirigido.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | u | dist |  | u | dist |  | u | dist |  | u | dist |  | u | dist |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Inicialização Passo: Passo: Passo: Passo:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | u | dist |  | u | dist |  | u | dist |  | u | dist |  | u | dist |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Passo: Passo: Passo: Passo: Passo:

**Figura 2**. Apresentação esquemáticados estados do vetor dist e da variável u.

O *diâmetro* de um grafo não dirigido é definido como o maior valor da menor distância entre quaisquer pares de vértices pertencentes ao grafo.

b) (1,0 ponto) Explique como o algoritmo de Dijkstra pode ser usado para computar o diâmetro de um grafo não dirigido. Qual o custo, no pior caso, em termos do número de nós do grafo, da sua solução. Explique a sua resposta.

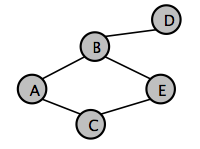
Dica: Considere todas as arestas com peso 1.

c) (1,0 ponto). Explique como o algoritmo de Floyd-Warshall pode ser usado para computar o diâmetro de um grafo não dirigido. Qual o custo, no pior caso, em termos do número de nós do grafo, da sua solução. Explique a sua resposta.

Dica: Considere todas as arestas com peso 1.

**Resposta**

a)



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | u | dist |  | u | dist |  | u | dist |  | u | dist |  | u | dist |
| A |  | ∞ |  |  | ∞ |  | A | 2 |  |  | 2 |  |  | 2 |
| B |  | ∞ |  | B | 1 |  |  | 1 |  |  | 1 |  |  | 1 |
| C |  | ∞ |  |  | ∞ |  |  | ∞ |  |  | 3 |  | C | 3 |
| D | D | 0 |  |  | 0 |  |  | 0 |  |  | 0 |  |  | 0 |
| E |  | ∞ |  |  | ∞ |  |  | 2 |  | E | 2 |  |  | 2 |

Inicialização Passo: 1 Passo: Passo: Passo:

b) Considere todas as arestas com peso 1. Desta forma, a menor distância entre 2 nós será o caminho de menor custo. Portanto, o algoritmo de Dijkstra determinará a menor distância entre um dado nó e todos os outros. Basta então:

1. Para cada nó do grafo:
   1. Execute o algoritmo de Dijkstra para cada um dos nós do grafo.
   2. Ao final de cada execução, verifique qual a maior distância encontrada.
   3. Mantenha a maior distância encontrada.
2. Retorne a maior distância encontrada.

O custo do algoritmo de Dijkstra (Passo 1a) é *O*(*n2*), onde *n* é o número de nós do grafo. Como o Passo 1 é executado para cada nó, o custo final será *O*(*n3*).

c) Novamente, considere todas as arestas com peso 1. Desta forma, a menor distância entre 2 nós será o caminho de menor custo. Portanto, o algoritmo de Floyd-Warshall determinará a menor distância entre os pares de nós. Basta então:

1. Execute o algoritmo de Floyd-Warshall para o grafo.
2. Retorne o maior elemento da matriz final.

O custo do algoritmo de Floyd-Warshall (Passo 1) é *O*(*n3*), onde *n* é o número de nós do grafo. O custo de determinar o maior elemento da matriz final (Passo 2) é *O*(*n2*). Logo, o custo final será *O*(*n3*).