Exercícios da Live de Revisão de Estruturas de Dados

Prof. Douglas Maioli

Professor Douglas Maioli

Link da Live: https://youtu.be/TflnCL2BOkU

1) Complete o seguinte texto abaixo com a palavras (último, primeiro, final, início, FIFO e LIFO).
   "A estrutura de dados Pilha é do tipo _____ em que o último elemento a ser inserido é o _____ a ser removido e o primeiro inserido é o _____ removido, pois nessa estrutura a inserção é feita no _____ e a remoção é feita no _____ . Já a estrutura de dados Fila é do tipo _____ em que o primeiro elemento a ser inserido é o _____ a ser removido, pois nessa estrutura a inserção é feita no _____ e a remoção é feita no _____ ."

2) Dados as seguintes complexidades de tempos computacionais
   a) $O(n) - Tempo\ Linear$
   b) $O(1) - Tempo\ Constante$
   c) $O(\log(n)) - Tempo\ Logarítmico$
   d) $O(2^n) - Tempo\ Exponencial$

   Complete com a, b, c ou d, conforme a complexidade, do pior caso, de cada método abaixo:
   I) Inserção e Remoção em Vetores: _____
   II) Inserção e Remoção em Listas Encadeadas: _____
   III) Inserção em Fila com Vetores: _____
   IV) Inserção em Fila Dinâmica: _____
   V) Remoção em Fila Dinâmica: _____
   VI) Remoção em Pilha Dinâmica: _____

3) Dados as seguintes complexidades de tempos computacionais
   a) $O(n) - Tempo\ Linear$
   b) $O(1) - Tempo\ Constante$
   c) $O(\log(n)) - Tempo\ Logarítmico$
   d) $O(2^n) - Tempo\ Exponencial$

   Complete com a, b, c ou d, conforme a complexidade de cada método abaixo:
I) Busca Sequencial: _____
II) Busca Binária (em vetor ordenado): _____
III) Busca em Árvore Binária de Busca balanceada: _____
IV) Busca em Árvore Binária de Busca totalmente desbalanceada: _____
V) Cada rotação em uma Árvore AVL: _____

Professor
Douglas Maioli

4) Diga de cada afirmação abaixo se é verdadeira ou falsa, em relação ao C++:
   a) ( ) Em um vetor alocado estaticamente é necessário informar seu tamanho (em tempo de compilação) antes de utilizá-lo e não é possível modificar esse tamanho durante a execução. Além disso seus elementos ocupam regiões contíguas na memória.
   b) ( ) Em um vetor alocado dinamicamente é necessário informar seu tamanho (em tempo de execução) antes de utilizá-lo, mas é possível modificar esse tamanho durante a execução. Além disso seus elementos ocupam regiões contíguas na memória.
   c) ( ) Em um vetor é possível armazenar elementos de diferentes tipos.
   d) ( ) Em uma lista encadeada não é necessário informar seu tamanho antes de utilizá-lo, já que temos a liberdade de inserir elementos dentro da capacidade de memória do computador, porém seus elementos ficam dispersos na memória, o que implica que o acesso a seus elementos, na maioria das vezes, não se dê em tempo constante.

5) Sobre a Tabela Hash com teste linear, diga qual afirmação é verdadeira ou falsa:
   a) ( ) A função de remoção inicia a busca no índice indicado pela função hash, e somente para quando encontra um espaço “vazio” ou quando encontra o elemento a ser removido, nesse caso, ela o remove e o marca o espaço como “disponível”.
   b) ( ) A função de busca inicia no índice indicado pela função hash, e somente para quando encontra um espaço “vazio” ou quando encontra o elemento buscado.
   c) ( ) A função de inserção inicia a busca no índice indicado pela função hash, e somente para quando encontra um espaço “vazio” ou “disponível”, inserindo o elemento neste espaço.
   d) ( ) As chaves (entradas da função hash) que são os elementos que vamos inserir na tabela hash podem ser de qualquer tipo, enquanto o valor (saída da função hash) devem ser inteiros, pois estes vão indicar o índice do vetor para iniciar a busca.

6) Complete o seguinte texto abaixo com a palavras (encadeamento separado, tratamento de colisões, lista encadeada).
   “Na tabela hash, quando utilizamos a estratégia de _____, utilizando _____, utilizamos um espaço de memória adicional, como uma _____ para alocar todos as chaves com o mesmo valor na função hash, incluindo as colisões. Lembrando que independente do tratamento de colisões escolhido, é essencial termos uma boa função hash, que diminua o número de colisões.”

7) Utilizando a tabela e a fórmula abaixo da função hash, calcule qual seria o valor (saída) da função hash, quando damos como entrada a chave "IRA".
Obs.: $(c_0, c_1, \dots, c_{k-1})$ são os valores correspondentes de cada letra da chave dada, com k letra. Utilize $a = 2$ e $n = 17$.

$$f(c_0, c_1, \dots, c_{k-1}) = (c_0 . a^{k-1} + c_1 . a^{k-2} + \cdots + c_{k-2} . a^1 + c_{k-1}) mod\ n$$

A	B	C	D	E	F	G	H	I
10	11	12	13	14	15	16	17	18
J	K	L	M	N	O	P	Q	R
19	20	21	22	23	24	25	26	27
S	T	U	V	W	X	Y	Z	SPACE
28	29	30	31	32	33	34	35	36

8) Dado o seguinte código, responda se cada afirmação abaixo é verdadeira ou falsa.

```
void Hash::insertItem(Aluno aluno) {
  int location = getHash(aluno);
  structure[location] = aluno;
  length++;
}
void Hash::deleteItem(Aluno aluno) {
  int location = getHash(aluno);
  structure[location] = Aluno();
  length--;
}
```

a) Estas funções tratam bem das colisões; ( )
b) Caso tenha colisão, podemos ter a inserção de um elemento sobre outro elemento já inserido, o que causaria dois erros: o elemento removido não seria mais possível buscá-lo e o "length" aumentaria 1, sem de fato ter aumentado a quantidade de elementos; ( )
c) Caso tenha colisão, podemos deletar um elemento diferente do que queremos deletar. ( )

Professor
Douglas Maioli

9) Construa a Árvore Binária seguindo os seguintes passos (e a ordem dada):
Insere os elementos 29, 20, 40, 10, 7, 3
Remove os elementos 7, 3
Insere os elementos 5, 60, 30, 50
Remove o elemento 29

10) Imprima a árvore binária do exercício 9, utilizando o percurso de pré-ordem, in-ordem e pós-ordem.

11) Em relação a árvore binária do exercício 9, responda:
A profundidade no nó 30: _____
A altura do nó 30: _____
A profundidade do nó 40: _____
A altura do nó 40: _____
A altura da Árvore: _____

12) Calcule o fator de balanceamento de cada nó na Árvore Binária do exercício 9:

13) Construa uma árvore AVL, fazendo a inserção dos elementos na seguinte ordem:

$$30,\ 20,\ 40,\ 10,\ 5,\ 60,\ 50$$

14) Dada a seguinte matriz de adjacências do grafo ponderado G, calcule o grau de cada vértice:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 7 & 1 & 0 & 4 \\ 3 & 0 & 9 & 0 & 2 \\ 0 & 0 & 0 & 5 & 0 \\ 2 & 8 & 2 & 0 & 6 \\ 0 & 0 & 3 & 0 & 0 \end{pmatrix} \end{array}$$

15) Complete as lacunas com uma das opções abaixo:

a) SearchTree::getSucessor

b) SearchTree::getPredecessor

```
void _____(NodeType* tree, Aluno& data)
{
  tree = tree->esquerda;
  while (tree->direita != NULL)
    tree = tree->direita;
  data = tree->aluno;
}

void _____'(NodeType* tree, Aluno& data)
{
  tree = tree->direita;
  while (tree->esquerda != NULL)
    tree = tree->esquerda;
  data = tree->aluno;
}
```

16) Em cada item (I a IV) temos as funções Inserir (Push, Enqueue) e Remover (Pop, Dequeue), responda de qual código essas funções pertencem:

a) Pilha com Vetor (Stack)

b) Pilha com Lista Encadeada (Dynamic Stack)

c) Fila com Vetor (Queue)

d) Fila com Lista Encadeada (Dynamic Queue)

I)

```
void _____(ItemType item)
{
  if (!isFull()){
    structure[length] = item;
    length++;
  } else {
    throw " List  is already full!";
  }
}
```

```
ItemType _____()
{
  if (!isEmpty()){
    ItemType aux = structure[length - 1];
    length--;
    return aux;
  } else {
    throw " List  is empty!";
  }
}
```

II)

```
void_____(ItemType item)
{
  if (!isFull()){
    structure[back % MAX_ITEMS] = item;
    back++;
  } else {
    throw " List  is already full!";
  }
}
```

```
ItemType _____()
{
  if (!isEmpty()){
    front++;
    return structure[(front-1) % MAX_ITEMS];
  } else {
    throw " List  is empty!";
  }
}
```

III)

```
void _____(ItemType newItem)
{
  if (!isFull()) {
    NodeType* newNode;
    newNode = new NodeType;
    newNode->info = newItem;
    newNode->next = NULL;
    if (rear == NULL)
      front = newNode;
    else
      rear->next = newNode;
    rear = newNode;
  } else {
    throw " List  is already full!";
  }
}
```

```
ItemType _____()
{
  if (!isEmpty()) {
    NodeType* tempPtr;
    tempPtr = front;
    ItemType item = front->info;
    front = front->next;
    if (front == NULL)
      rear = NULL;
    delete tempPtr;
    return item;
  } else {
    throw " List  is empty!";
  }
}
```

IV)

```
void _____(ItemType item){
  if (!isFull()){
    NodeType* location;
    location = new NodeType;
    location->info = item;
    location->next = structure;
    structure = location;
  } else {
    throw " List  is already full!";
  }
}
```

```
ItemType _____(){
  if (!isEmpty()) {
    NodeType* tempPtr;
    tempPtr = structure;
    ItemType item = structure->info;
    structure       = structure->next;
    delete tempPtr;
    return item;
  } else {
    throw " List  is empty!";
  }
}
```