Estratégia Concursos

Aula 00

Ministério da Fazenda (ATA - Assistente Técnico Administrativo) Passo Estratégico de Informática

Autor:
Thiago Rodrigues Cavalcanti

04 de Junho de 2024

1. Conhecimentos básicos de linguagens de programação relativos a Lógica e Estrutura de programação

Sumário

APRESENTAÇÃO

Olá Senhoras e Senhores,

Eu me chamo Thiago Cavalcanti. Sou funcionário do Banco Central do Brasil, passei no concurso em 2010 para Analista de Tecnologia da Informação (TI). Atualmente estou de licença, cursando doutorado em economia na UnB. Também trabalho como professor de TI no Estratégia e sou o analista do Passo Estratégico de Informática.

Tenho graduação em Ciência da Computação pela UFPE e mestrado em Engenharia de Software. Já fui aprovado em diversos concursos tais como ANAC, BNDES, TCE-RN, INFRAERO e, claro, Banco Central. A minha trajetória como concurseiro durou pouco mais de dois anos. Neste intervalo, aprendi muito e vou tentar passar um pouco desta minha experiência ao longo deste curso.

O QUE É O PASSO ESTRATÉGICO?

O Passo Estratégico é um material escrito e enxuto que possui dois objetivos principais:

a) orientar revisões eficientes;

b) destacar os pontos mais importantes e prováveis de serem cobrados em prova.

Assim, o Passo Estratégico pode ser utilizado tanto para **turbinar as revisões dos alunos mais adiantados nas matérias, quanto para maximizar o resultado na reta final de estudos por parte dos alunos que não conseguirão estudar todo o conteúdo do curso regular.**

Em ambas as formas de utilização, como regra, **o aluno precisa utilizar o Passo Estratégico em conjunto com um curso regular completo.**

Isso porque nossa didática é direcionada ao aluno que já possui uma base do conteúdo.

Assim, se você vai utilizar o Passo Estratégico:

a) **como método de revisão**, você precisará de seu curso completo para realizar as leituras indicadas no próprio Passo Estratégico, em complemento ao conteúdo entregue diretamente em nossos relatórios;

b) **como material de reta final**, você precisará de seu curso completo para buscar maiores esclarecimentos sobre alguns pontos do conteúdo que, em nosso relatório, foram eventualmente expostos utilizando uma didática mais avançada que a sua capacidade de compreensão, em razão do seu nível de conhecimento do assunto.

Seu cantinho de estudos famoso!

Poste uma foto do seu cantinho de estudos nos stories do Instagram e nos marque:

@passoestrategico

Vamos repostar sua foto no nosso perfil para que ele fique famoso entre milhares de concurseiros!

ANÁLISE ESTATÍSTICA

A análise estatística estará disponível a partir da próxima aula.

ROTEIRO DE REVISÃO E PONTOS DO ASSUNTO QUE MERECEM DESTAQUE

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Para revisar e ficar bem preparado no assunto, você precisa, basicamente, seguir os passos a seguir:

Lógica de Programação

De forma simples e direta, **lógica de programação** é o modo como se escreve um programa de computador, um algoritmo. Podemos definir "lógica de programação" como a elaboração de sequências de ações para atingir um determinado objetivo. O processo envolve o uso de dispositivos lógicos, como estruturas condicionais (if/else) e de repetição (for/while). Já **um algoritmo** é uma sequência de passos para se executar uma função. Para entendermos melhor, podemos fazer um paralelo fora da computação, com uma receita de bolo.

Na receita, devem-se seguir os passos para o bolo ficar pronto e sem nenhum problema. Na informática, os programadores escrevem as "receitas de bolo" (algoritmos) de modo que o computador leia e entenda o que deve ser feito, ao executar o algoritmo. Para isto é necessária uma **linguagem de programação**.

Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa. Sua implementação pode ser feita por um computador, por outro tipo de autômato ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros. Tal diferença pode ser

reflexo da complexidade computacional aplicada, que depende de estruturas de dados adequadas ao algoritmo. Por exemplo, um algoritmo para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça enquanto outro algoritmo especifica que você deve primeiro vestir a calça e depois as meias e os sapatos. Fica claro que o primeiro algoritmo é mais difícil de executar que o segundo apesar de ambos levarem ao mesmo resultado.

A linguagem de programação é como uma língua normal, um grupo de palavras com significados. No caso da programação, a maioria das linguagens é escrita em Inglês. Estas linguagens fazem o computador assimilar cada comando e função de um algoritmo, depois executar cada função.

Na hora de programar alguns passos são indispensáveis, como Declarar Variáveis. Variáveis podem ser escritas por letras ou números, que representam um valor que pode ser mudado a qualquer momento. Cada variável tem um espaço na memória para armazenar seus dados. Porem existem vários tipos de dados, sendo os mais comuns:

- Numérico: todo e qualquer tipo número, positivo ou negativo
- Reais: podem ser positivos ou negativos e decimais.
- Caractere: São os textos. Qualquer número pode entrar aqui, porém não terá função matemática.

Diante de todo o exposto, podemos definir lógica de programação como a capacidade de dividir o problema em partes menores é uma etapa essencial da lógica de programação e precisa ser levada em consideração quando nos deparamos com qualquer exercício/desafio.

Linguagem de Programação

A linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

Não entendeu nada? Vamos a outra definição.

Linguagem de Programação é uma linguagem escrita e formal que especifica um conjunto de instruções e regras usadas para gerar programas (software).

A definição ainda está confusa?

Vamos imaginar o computador como uma super calculadora, capaz de fazer cálculos muito mais rápido que nós, mas para isso devemos dizer para o computador o que deve ser calculado e como deve ser calculado. A função das linguagens de programação é exatamente essa, ou seja, servir de um meio de comunicação entre nós e os computadores.

Desde os primórdios da computação foram sendo desenvolvidas várias linguagens e adaptadas conforme os computadores evoluíram. Com isso, as linguagens de programação foram divididas em **quatro gerações** desde o início da década de 50 até os dias atuais.

Na **primeira geração**, a programação era feita através de linguagem de máquina e a Assembly, que dependiam de um hardware/software para que as tarefas pudessem ser executadas.

Na **segunda geração**, começaram a ser desenvolvidas linguagens mais modernas que fizeram sucesso no mercado e ampliaram o uso de bibliotecas de software. As linguagens que marcaram essa geração foram Fortran (usada por engenheiros e cientistas), Cobol (usada em aplicações comerciais), Algol (usada para suporte às estruturas de controle e tipos de dados) e Basic (usada para propósitos acadêmicos).

Na **terceira geração**, as linguagens de programação também são chamadas de linguagens estruturadas, caracterizada pela sua enorme clareza e estruturação na organização dos dados. Através delas foi possível atribuir recursos inteligentes, criar sistemas distribuídos, etc. São classificadas em:

- linguagens de propósito geral - baseadas na linguagem Algol e podem ser utilizadas para várias aplicações, desde propósitos científicos à comerciais. Ex.: C, Pascal, Modula-2;
- linguagens especializadas - desenvolvidas apenas para uma determinada aplicação. Ex.: linguagem Lisp, usada em engenharia de software para a manipulação de listas e símbolos; APL criada para manipulação de vetores; Forth criada para o desenvolvimento de softwares para microprocessadores, etc.
- linguagens orientadas a objetos - com mecanismos baseados na semântica e sintática dando apoio à programação orientada a objetos. Ex.: Smalltalk, C++, Delphi, etc.

Na **quarta geração**, também chamadas de linguagens artificiais, houve a criação de sistemas baseados em inteligência artificial. Elas foram divididas em:

- linguagem de consulta - criadas para a manipulação de base de dados gerenciando uma grande quantidade de informações armazenadas em arquivos;
- linguagens geradoras de programas - são linguagens 4GL, que possibilitam a criação de programas complexos, da terceira geração e possuem um nível mais alto que as linguagens de quarta geração;
- outras linguagens - são aquelas usadas em sistemas de suporte à decisão, modelagem de sistemas, linguagens para protótipos, etc.

Na década de 90 e nos anos 2000 surgiram linguagens populares como: Python, Java, Ruby, Javascript, PHP, Delphi, C#, etc. Dando início a uma quinta geração de linguagens de programação.

Existem dois tipos de linguagens de programação: as de baixo nível e as de alto nível. Os computadores interpretam tudo como números em base binária, ou seja, só entendem zero e um. As linguagens de baixo nível são interpretadas diretamente pelo computador, tendo um resultado rápido, porém é muito difícil e incômodo se trabalhar com elas.

Já as linguagens de alto nível são mais fáceis de se trabalhar e de entender, as ações são representadas por palavras de ordem (exemplo: faça, imprima, etc.) geralmente em inglês. Elas foram feitas assim para facilitar a memorização e a lógica. As linguagens de alto nível não são interpretadas diretamente pelo computador,

sendo necessário traduzi-las para linguagem binária. A partir daqui entra em ação um programa chamado **compilador**.

Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Em geral, um compilador não produz diretamente o código de máquina, mas sim um programa em linguagem simbólica semanticamente equivalente ao programa em linguagem de alto nível. O programa em linguagem simbólica é então traduzido para o programa em linguagem de máquina através de montadores.

Quando programamos em uma linguagem de programação de alto nível primeiramente criamos um arquivo de texto comum contendo a lógica do programa, ou seja, é onde falamos ao computador como deve ser feito o que queremos. Este arquivo de texto é chamado de código-fonte, cada palavra de ordem dentro do código-fonte é chamada de instrução. Após criarmos o código-fonte devemos traduzir este arquivo para linguagem binária usando o compilador correspondente com a linguagem na qual estamos programando. O compilador irá gerar um segundo arquivo que chamamos de executável ou programa, este arquivo gerado é interpretado diretamente pelo computador.

Para desempenhar suas tarefas, um compilador deve executar dois tipos de atividade. A primeira atividade é a análise do código fonte, onde a estrutura e significado do programa de alto nível são reconhecidos. A segunda atividade é a síntese do programa equivalente em linguagem simbólica. Embora conceitualmente seja possível executar toda a análise e apenas então iniciar a síntese, em geral estas duas atividades ocorrem praticamente em paralelo.

Estrutura de Dados

Na criação de um programa em qualquer linguagem são abordados os seguintes itens: bloco de programa; declaração de variáveis; sintaxe dos comandos.

Em linguagens de programação, o tipo de dados de uma variável define o conjunto de valores que a variável pode assumir. Por exemplo, uma variável do tipo *lógico* pode assumir o valor *verdadeiro* ou *falso*.

Uma declaração de variável em uma linguagem como **C** ou **Pascal** especifica:

- O conjunto de valores que pode assumir.
- O conjunto de operações que podemos efetuar.
- A quantidade de bytes que deve ser reservada para ela.
- Como o dado representado por esses bytes deve ser interpretado (por exemplo, uma cadeia de bits pode ser interpretada como um inteiro ou real...).

Então, tipos de dados podem ser definidos como métodos para interpretar o conteúdo da memória do computador.

Podemos ver o conceito de **Tipo de Dados** de uma outra perspectiva: não em termos do que um **computador** pode fazer (exemplo: interpretar os bits), mas em termos do que os usuários desejam fazer (exemplo: somar dois inteiros).

Este conceito de Tipo de Dado "divorciado" do hardware é chamado **Tipo Abstrato de Dado - TAD**.

Dessa forma definimos **Estrutura de Dados - ED** como um método particular de se implementar um **TAD**.

A implementação de um Tipo Abstrato de Dado escolhe uma Estrutura de Dados para representá-lo. Cada ED é construída dos tipos primitivos (inteiro, real, char, etc.) ou dos tipos compostos (array, registro, etc.) de uma linguagem de programação.

Não importa que tipo de dados estaremos trabalhando, a primeira operação a ser efetuada em um TAD é a criação. Depois, podemos realizar inclusões e remoções de dados. A operação que varre todos os dados armazenados num TAD é o percurso, podendo também ser realizada uma busca por algum valor dentro da estrutura.

Exemplos de **Tipo Abstrato de Dado - TAD**:

- Lineares: Listas Ordenadas; Pilhas; Filas; Deques
- Não lineares: Árvores; Grafos

Listas

As listas são estruturas formadas por um conjunto de dados de forma a preservar a relação de ordem linear entre eles. Uma lista é composta por nós, que podem conter um dado primitivo ou composto (em cada nó).

Cada elemento (ou nó) da lista tem a seguinte estrutura:

➔ um atributo com o valor do elemento, e

➔ um atributo com uma referência para o próximo elemento da lista.

É importante observar que a ordem dos elementos na lista é totalmente relevante e esses elementos são todos do mesmo tipo.

Representação:

L_1	L_2	L_3	...	L_{n-1}	L_n

Onde L_1 é o primeiro elemento da lista e L_n é o último elemento da lista.

Formas de Representação:

a) Sequencial

Uma lista representada de forma sequencial é um conjunto de registros onde estão estabelecidas regras de precedência entre seus elementos. O sucessor de um elemento ocupa posição física subsequente.

A implementação de operações pode ser feita utilizando *array* e *registro*, associando o elemento a(i) com o índice i (mapeamento sequencial)

Essa representação tem como característica os elementos serem armazenados em posições consecutivas. Para a inserção de um elemento na posição a(i) causa o deslocamento para direita do elemento a(i) até o último elemento; e para a eliminação do elemento a(i) é necessário o deslocamento à esquerda do elemento a(i+1) até o último elemento.

Como vantagens dessa representação podemos descrever o acesso direto indexado a qualquer elemento da lista e o tempo constante para acessar o elemento *i*.

Já as desvantagens são: movimentação quando um elemento é inserido ou eliminado e o tamanho máximo é pré-definido.

b) Encadeada

Esta estrutura é tida como uma sequência de elementos encadeados por ponteiros, ou seja, cada elemento deve conter, além do dado propriamente dito, uma referência para o próximo elemento da lista.

Há duas alternativas para implementação de operações de listas encadeadas: utilizando **arrays estáticos** ou **variáveis dinâmicas**. Entretanto, não há vantagem na implementação da lista encadeada com array, uma vez que permanece o problema do tamanho máximo da lista predefinido e necessita de mais espaço para armazenar os endereços dos elementos sucessores. Portanto, são utilizadas variáveis dinâmicas como forma de implementação para as nossas listas encadeadas.

Antes de prosseguirmos, é importante entendermos o conceito e utilização de variáveis dinâmicas e de ponteiros (apontadores).

As linguagens de programação modernas tornaram possível explicitar não apenas o acesso aos dados, mas também aos endereços desses dados. Isso significa que ficou possível a utilização de ponteiros explicitamente

implicando que uma distinção notacional deve existir entre os dados e as referências (endereços) desses dados. Os valores para ponteiros são gerados quando dados correspondentes a seus tipos são alocados/desalocados dinamicamente.

Pilhas

As pilhas são listas onde a inserção de um novo item ou a remoção de um item já existente se dá em uma única extremidade, no topo. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

Pilhas são também conhecidas como listas LIFO (*last in first out*).

A implementação de pilhas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas.

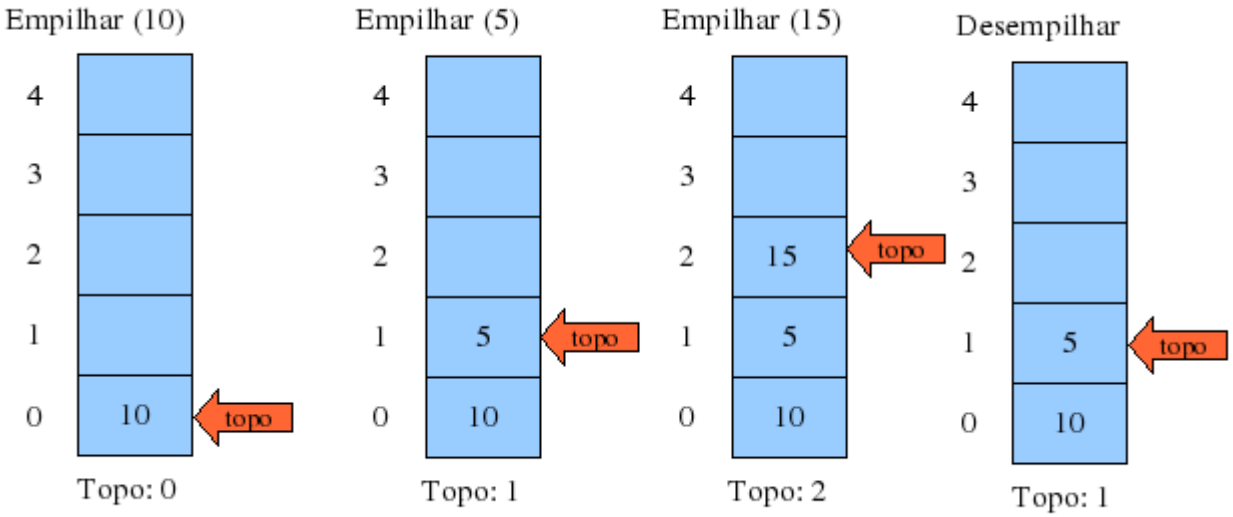
Numa pilha, a manipulação dos elementos é realizada em apenas uma das extremidades, chamada de topo, em oposição a outra extremidade, chamada de base.

Operações com Pilha:

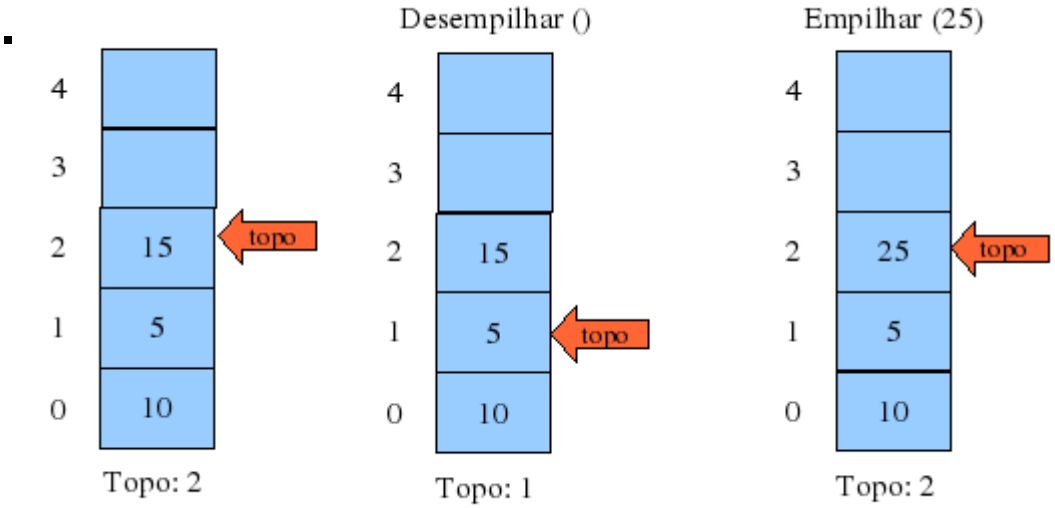
Todas as operações em uma pilha podem ser imaginadas como as que ocorre numa pilha de pratos em um restaurante ou como num jogo com as cartas de um baralho:

- criação da pilha (informar a capacidade no caso de implementação sequencial - vetor);
- empilhar (push) - o elemento é o parâmetro nesta operação;
- desempilhar (pop);
- mostrar o topo;
- verificar se a pilha está vazia (isEmpty);
- verificar se a pilha está cheia (isFull - implementação sequencial - vetor).

Supondo uma pilha com capacidade para 5 elementos (5 nós).

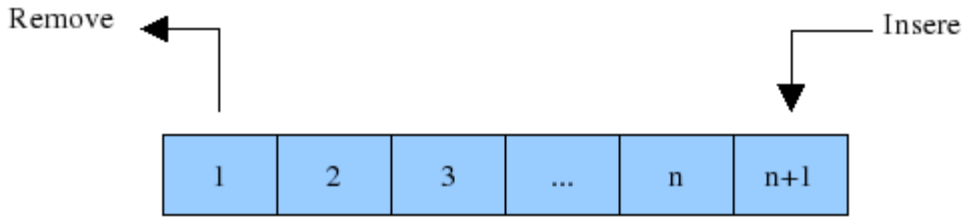


Na realidade a remoção de um elemento da pilha é realizada apenas alterando-se a informação da posição do topo.



Filas (Queue)

São estruturas de dados do tipo FIFO (first-in first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.



A implementação de filas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas.

Operações com Fila:

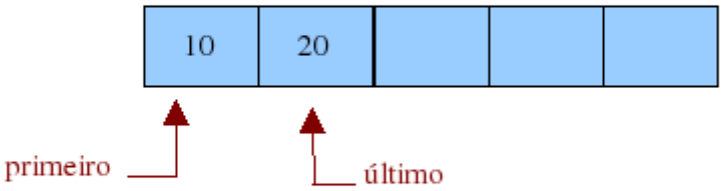
Todas as operações em uma fila podem ser imaginadas como as que ocorre numa fila de pessoas num banco, exceto que os elementos não se movem na fila, conforme o primeiro elemento é retirado. Isto seria muito custoso para o computador. O que se faz na realidade é indicar quem é o primeiro.

- criação da fila (informar a capacidade no caso de implementação sequencial - vetor);
- enfileirar (enqueue) - o elemento é o parâmetro nesta operação;
- desenfileirar (dequeue);
- mostrar a fila (todos os elementos);
- verificar se a fila está vazia (isEmpty);
- verificar se a fila está cheia (isFull - implementação sequencial - vetor).

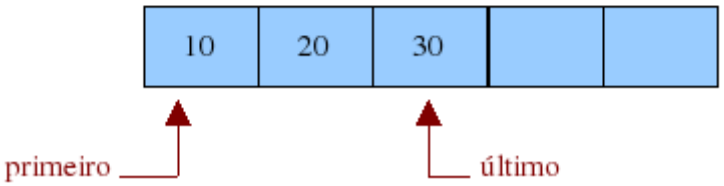
Supondo uma fila com capacidade para 5 elementos (5 nós).

Insere(10)

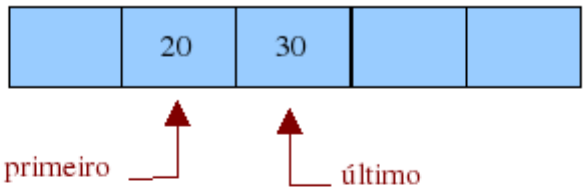
Insere (20)



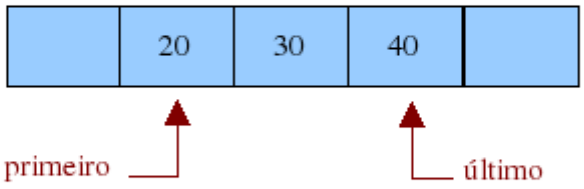
Insere(30)



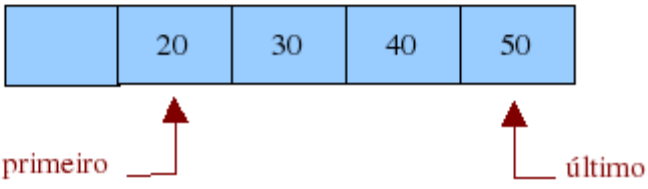
Remove()



Insere (40)

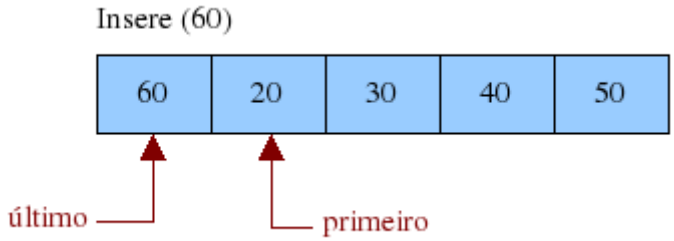


Insere (50)

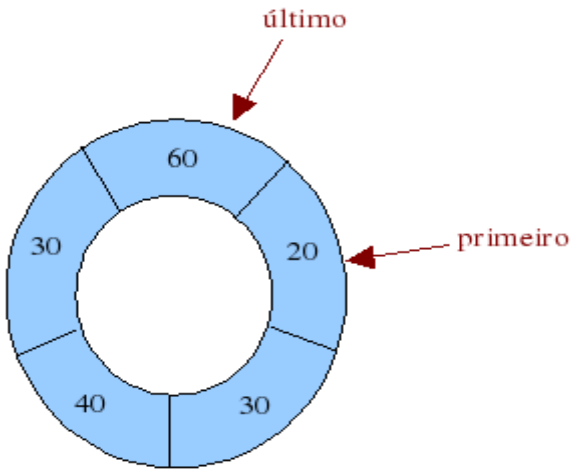


Na realidade a remoção de um elemento da fila é realizada apenas alterando-se a informação da posição do último.

Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o inicio do vetor, resultando numa fila circular.



Desta forma a fila simula uma representação circular:



Árvores

Anteriormente estudamos a organização dos dados de uma maneira linear, onde a propriedade básica é a relação sequencial mantida entre seus elementos.

Agora, estudaremos uma outra forma de organizar os dados, chamada genericamente de "não-linear". Essa forma permite que outros tipos de relação entre dados possam ser representados, como por exemplo, hierarquia e composição.

Um dos exemplos mais significativos de estruturas não-lineares é a árvore.

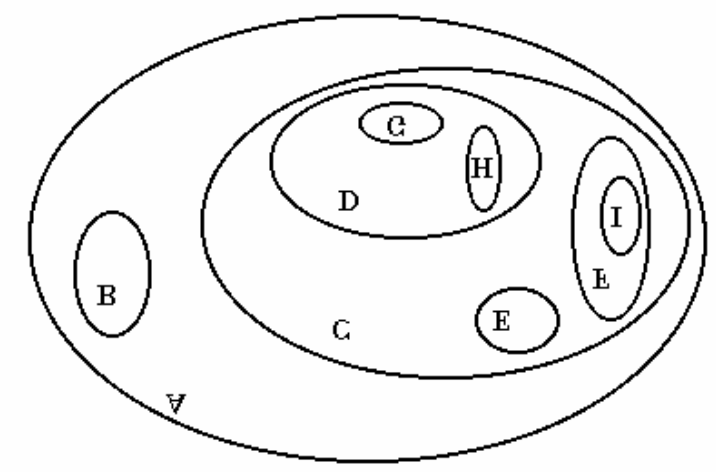
Representação Gráfica

As três formas mais comuns de representação gráfica de uma árvore são:

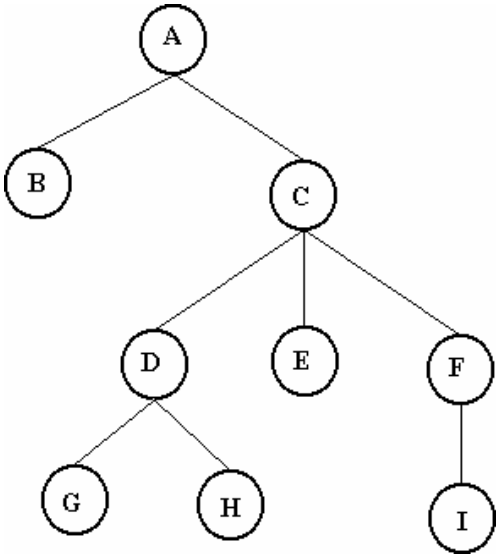
1) Representação por parênteses aninhados

(A (B) (C (D (G) (H)) (E) (F (I))))

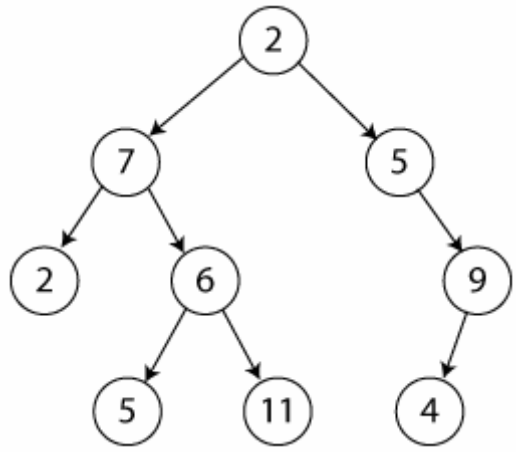
2) Diagrama de inclusão



3) Representação hierárquica



A figura abaixo mostra dois exemplos de árvores. Do lado esquerdo, temos uma curiosa árvore na natureza. Do lado direito, uma representação esquemática de uma árvore cujos “nós” contêm os números 2, 7, 5, etc.



Compiladores

Compiladores são programas de computador capazes de traduzir um código-fonte escrito em uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível.

Código-fonte é um conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica.

Código-objeto representa a versão compilada e montada de um arquivo de código-fonte.

- Ele não é diretamente um arquivo executável, na medida em que ele deve ser ligado com outros códigos-objeto para criar um executável.

Pré-processador é um programa que recebe um texto e efetua conversões léxicas, tais como substituição de macros, inclusão condicional, inclusão de ficheiros e exclusão de comentários.

- Diretivas de Compilação (Ex: #define, #include, etc) são comandos que não são compilados, dirigidos ao pré-processador e executados pelo compilador antes do processo de compilação em si.

Processo de compilação

Dividido em duas fases:

- Análise ou Front-End
  - Análise Léxica
    - Um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos.
    - Inicialmente, ele descarta comentários, espaços em branco, marcas de formatação, etc; depois ele efetivamente agrupa os caracteres em tokens.
    - A Análise Léxica é uma forma de verificar um alfabeto, ou seja, conseguimos verificar se existe ou não algum caractere que não faz parte do alfabeto.

  - Análise Sintática
    - Também conhecida como Análise Gramatical ou Parsing.
    - Analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo uma determinada gramática formal.
    - Ela pega os tokens resultantes do processo de análise léxica e joga em uma estrutura hierárquica, como uma árvore.
    - O analisador sintático consulta a tabela de símbolos para verificar a presença de variáveis definidas pelo programador.
    - Um erro sintático é um caso em que as "frases" do programa estão mal formuladas.
      - ➔ Exemplo de erros sintáticos: parênteses que abrem, mas não fecham; duas instruções sem um ponto-e-vírgula entre elas; uma palavra-chave sendo usada numa posição inesperada; etc.
      - ➔ Os erros de sintaxe são sempre detectados em tempo de compilação/parse.

  - Análise Semântica
    - Entre as principais atividades, estão a checagem de tipos, verificação de fluxos de controle e verificação de unicidade de declaração de variáveis. Essa fase também é encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração.
    - Erros semânticos podem ser detectados tanto durante a compilação quanto durante a execução. Exemplos de erros semânticos: dividir um número por uma string; criar uma classe que herda de si mesma; dividir zero por zero; etc.

        - O Analisador Semântico usa a árvore de análise como entrada e verifica se os tipos de dados são apropriados usando informação da tabela de símbolos.

    - Geração de Código Intermediário
        - Nesta fase, ocorre a transformação da árvore sintática em uma representação intermediária do código fonte. Esta linguagem intermediária é mais próxima da linguagem objeto do que o código fonte, mas ainda permite uma manipulação mais fácil do que se o código Assembly ou código de máquina fosse utilizado.

- Síntese ou Back-End
    - Otimização de Código Intermediário
        - Nesta fase, examina-se o código intermediário produzido durante a fase anterior com objetivo de produzir, através de algumas técnicas, um código que execute com bastante eficiência o programa.
        - Utilizam-se técnicas que detectam padrões dentro do código produzido e os substitui por códigos mais eficientes.

    - Geração de Código Final
        - Última fase do processo de compilação.
        - Ocorre a geração do código de montagem (ainda não é o código-objeto, apesar de muitos compiladores já realizarem a montagem).
        - Um código-objeto não é imediatamente executável, visto que ainda há código binário a ser incluído no programa, tais como bibliotecas.

Assembler

Montador (Assembler)

- O processo de Montagem traduz um programa escrito em linguagem de montagem (Ex.: Assembly) em um equivalente em linguagem de máquina (com algumas referências).

Cross-assembler:

- É executado em um computador com um processador diferente daquele para o qual se está gerando código.

Macro-assembler

- Dispõe de recursos de macro, efetuando a expansão do código cada vez que uma macro for encontrada.

Micro-assembler

- Permite a escrita de micro-instruções, definindo-se assim um conjunto de instruções de um processador microprogramável.

Meta-assembler

- É um assembler que pode montar programas para vários processadores diferentes.

Assembler de um passo

- Varre o programa-fonte apenas uma vez, gerando o código (deve existir alguma forma de se revolver as referências adiante).

Assembler de dois passos

- Varre o programa-fonte duas vezes para gerar o código, podendo assim resolver automaticamente as referências adiante.

Ligador, Carregador e Bibliotecas

Ligadores são programas que recebem como entrada um conjunto de arquivos-objeto, bibliotecas padrão, arquivos de controle, parâmetros diversos e bibliotecas compartilhadas e os unem em um módulo, denominado Módulo de Carga, que é posteriormente carregado em memória (eventualmente, pode ser mais de um módulo).

- Ele é responsável por resolver referências internas e externas.

Carregador (Loader)

- São programas que transferem o código de máquina de um módulo objeto para a memória e encaminham o início de sua execução.

Tipos básicos de carregador

- Carregadores binários
  - Também chamados de carregadores absolutos
  - São mais simples e apenas copiam o arquivo em formato binário para a memória.
  - Devem sempre serem carregados na mesma área de memória para serem executados corretamente.
- Carregadores realocáveis
  - Podem ser colocados em qualquer local da memória para execução.

Bibliotecas

- São uma coleção de funções e definições utilizadas no desenvolvimento de software para algum propósito específico.
- Elas provêm serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular.
- O Ligador é o responsável, em geral, por fazer referências entre os executáveis e as bibliotecas.
- Tipos de bibliotecas
  - Bibliotecas estáticas
    - Todo o código da biblioteca é copiado e inserido dentro do binário final.
  - Bibliotecas dinâmicas (ou compartilhadas)
    - Não são inseridas em sua totalidade, elas são apenas referenciadas no binário final.

- Mais lentas que a biblioteca estática.

Interpretadores

Interpretadores processam um comando de cada vez, ao contrário dos compiladores, que leem todo o arquivo de código fonte antes de produzir uma sequência binária.

Interpretadores geralmente são muito mais lentos do que compiladores

A detecção de erros em interpretadores é geralmente limitada à verificação da sintaxe da linguagem e dos tipos de variáveis

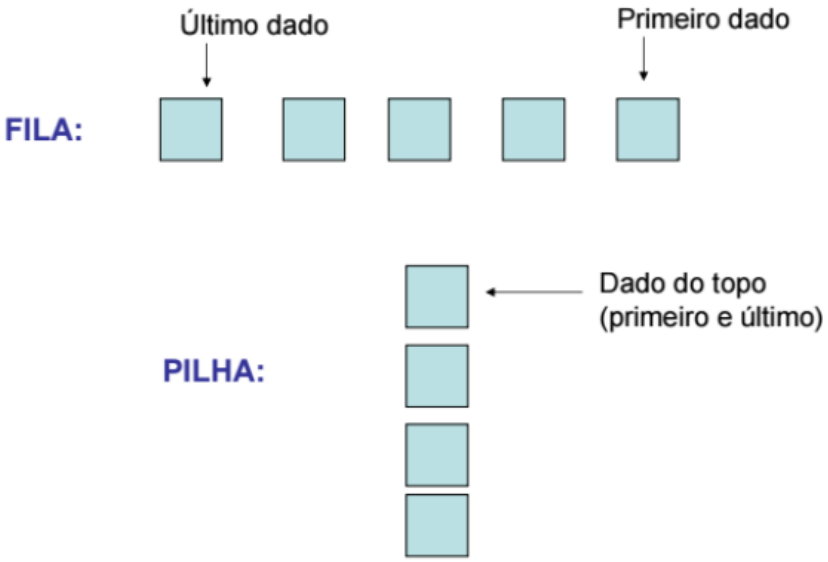
APOSTA ESTRATÉGICA

A ideia desta seção é apresentar os pontos do conteúdo que mais possuem chances de serem cobrados em prova, considerando o histórico de questões da banca em provas de nível semelhante à nossa, bem como as inovações no conteúdo, na legislação e nos entendimentos doutrinários e jurisprudenciais[1].

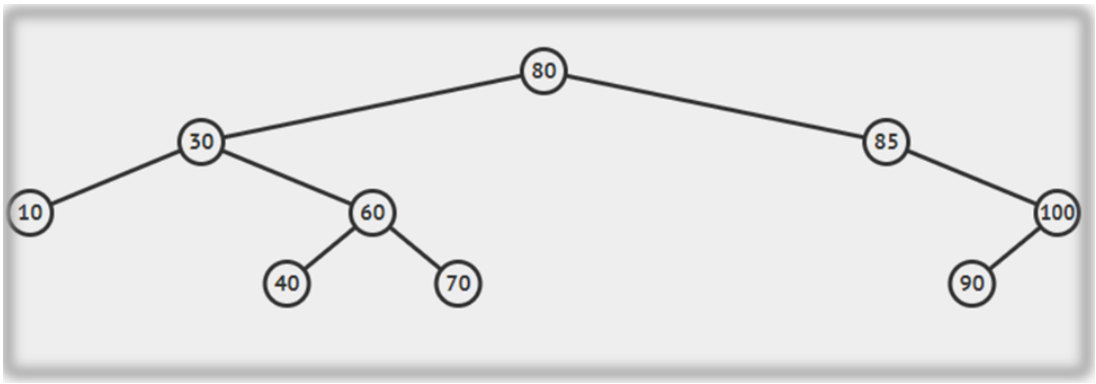
Dentro do assunto “Estrutura de Dados e Algoritmos”, os pontos que acreditamos que possuem mais chances de serem cobrados pela banca são os listados abaixo:

Fila e **Pilha:**

[1] Vale deixar claro que nem sempre será possível realizar uma aposta estratégica para um determinado assunto, considerando que às vezes não é viável identificar os pontos mais prováveis de serem cobrados a partir de critérios objetivos ou minimamente razoáveis.



Árvore binária de busca

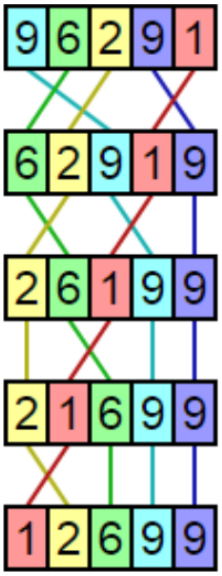


Formas de percorrer a árvore:

- Pré-ordem: 80, 30, 10, 60, 40, 70, 85, 100, 90.
- In-ordem: 10, 30, 40, 60, 70, 80, 85, 90, 100.
- Pós-ordem: 10, 40, 70, 60, 30, 90, 100, 85, 80.

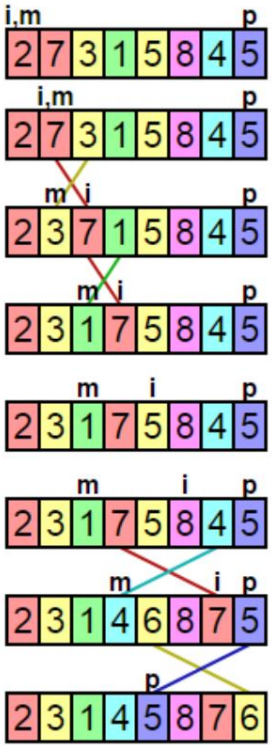
Métodos de ordenação

- **Bubble Sort:** a cada iteração do método, percorremos a lista a partir de seu início comparando cada elemento com seu sucessor, trocando-se de posição se houver necessidade.



Melhor Caso	Caso Médio	Pior Caso
$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^2)$

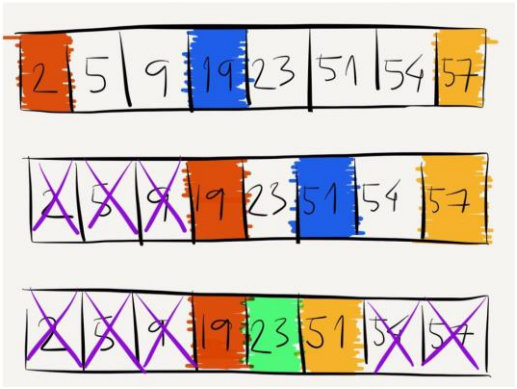
- **Quick Sort**: escolhe um pivô e divide os elementos em 2 conjuntos – maiores e menores que o pivô. Em seguida, repete para os conjuntos.



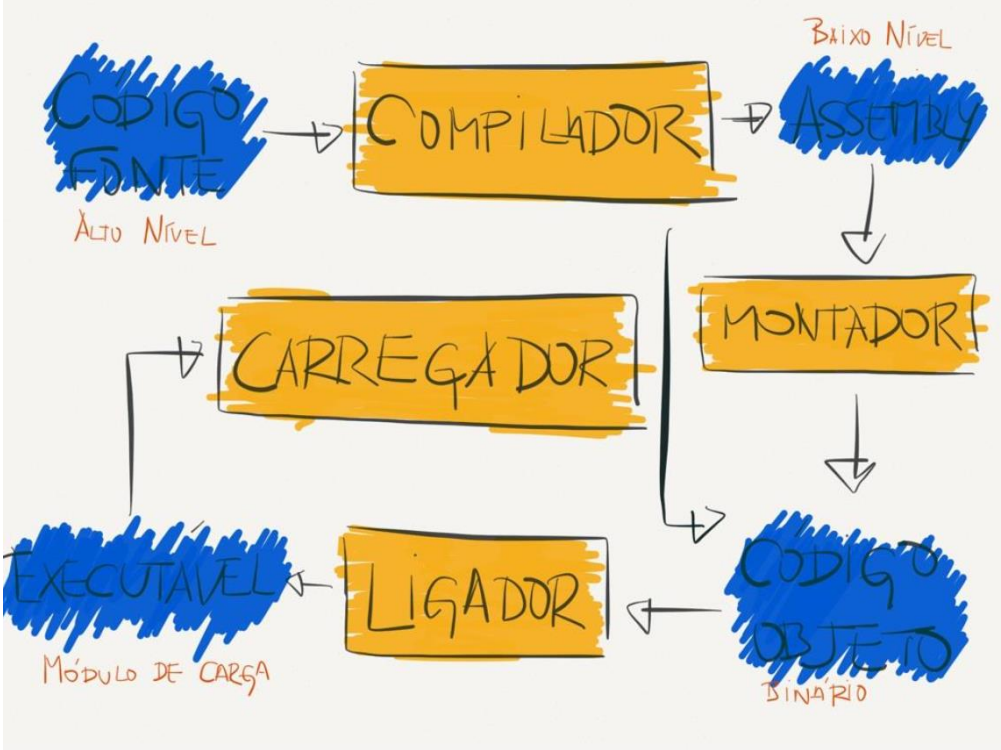
Melhor Caso	Caso Médio	Pior Caso
$O(n \log n)$	$O(n \log n)$	$O(n^2)$

Algoritmos de Busca

- **Busca Binária** considera que a lista está ordenada e consiste em:
  - Visita o elemento central da lista e compara com o elemento procurado.
    - Se o elemento central for igual, encerra a busca.
    - Se o elemento central for maior do que o elemento procurado, repete a operação considerando os elementos à esquerda do elemento central como uma nova lista.
    - Se o elemento central for menor do que o elemento procurado, repete a operação considerando os elementos à direita do elemento central como uma nova lista.



Compiladores



QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.

1.

Considere, hipoteticamente, que um Técnico do TRE-SP tem, em seu computador, a seguinte organização de um diretório:

Principal: Dados

Dentro de Dados: Técnicos Práticos

Dentro de Técnicos: Árvores Hash Recursão Filas Pilhas

Dentro de Práticos: Programas AFazer Prontos

Dentro de Prontos: Eleições Urnas

Dentro de Programas: Corretos ComErro

Dentro de ComErro: Urgentes Pendentes Antigos

A estrutura de dados

a) fila é a mais adequada para representar este diretório.

b) pilha é a mais adequada para representar este diretório.

c) árvore binária, ao armazenar este diretório, terá Dados na raiz e nós com grau 2, 3, 5 e folhas.

d) árvore, que consegue armazenar este diretório, é de ordem 5.

e) hashing, ao armazenar este diretório, não terá colisões na tabela de dispersão

Comentários

Uma árvore é uma estrutura de dados usada principalmente para representar dados com uma relação hierárquica entre seus elementos. Os elementos de uma árvore são denominados "nós" e existe um nó especial chamado "Raiz da árvore" (não é filho de nenhum elemento).

O número máximo de filhos de um elemento é chamado ordem da árvore. Vamos analisar a organização do diretório:

- Dados
  - Técnicos
    - Árvores
    - Hash
    - Recursão
    - Filas
    - Pilhas
  - Práticos
    - Programas
      - Corretos
      - ComErro
        - Urgentes
        - Pendentes
        - Antigos
- AFazer
- Prontos
  - Eleições
  - Urnas

Podemos ver que a pasta "Técnicos" possui cinco subpastas. Dessa forma, a ordem dessa árvore será 5.

Gabarito: alternativa D.

2.

Estruturas de dados básicas, como as pilhas e filas, são usadas em uma gama variada de aplicações. As filas, por exemplo, suportam alguns métodos essenciais, como o

a) enqueue(x), que insere o elemento x no fim da fila, sobrepondo o último elemento.

b) dequeue(), que remove e retorna o elemento do começo da fila; um erro ocorrerá se a fila estiver vazia.

c) push(x), que insere o elemento x no topo da fila, sem sobrepor nenhum elemento.

d) pop(), que remove o elemento do início da fila e o retorna, ou seja, devolve o último elemento inserido.

e) top(), que retorna o elemento do fim da fila sem removê-lo; um erro ocorrerá se a fila estiver vazia.

Comentários

Fila (queue) é uma estrutura de dados baseada na política FIFO (first in, first out), ou seja, o primeiro objeto inserido na fila é também o primeiro a ser removido. Nessa estrutura, o elemento a ser removido é o que foi inserido a mais tempo. Segundo Cormen, a operação INSERT, chamada ENQUEUE, adiciona um elemento ao final da fila, e a operação DELETE, chamada de DEQUEUE, remove o elemento no início da fila.

A operação DEQUEUE só pode ser aplicada se a fila não estiver vazia, para evitar o erro de underflow ou fila vazia.

Gabarito: alternativa B.

3.

Assinale a opção que apresenta a denominação da estrutura de dados constituída por um conjunto de elementos individualizados, em que cada um dos elementos — com exceção dos elementos inicial e final — referencia sempre outros dois, um que o antecede e outro que o sucede.

a) lista circular

b) grafo

c) lista duplamente encadeada

d) árvore

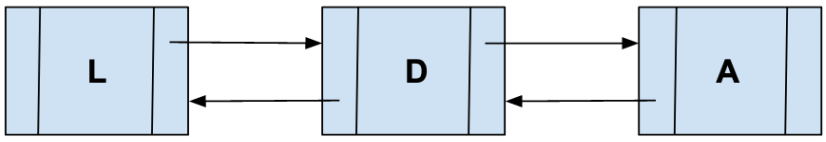
e) pilha

Comentários

A chave para responder a questão é o trecho “cada um dos elementos – com exceção dos elementos inicial e final – referencia sempre outros dois, um que o antecede e outro que o sucede”. A estrutura de dados que tem essa característica é a lista duplamente encadeada.

Lista encadeada é uma estrutura de dados dinâmica formada por uma sequência encadeada de elementos.

A lista duplamente encadeada é uma lista em que cada elemento (ou nó) possui um campo para a informação e dois campos para endereços, um para guardar o endereço do nó anterior e outro para guardar o endereço do nó posterior, permitindo, assim, que a lista seja percorrida em dois sentidos. Nenhuma das outras alternativas possui uma estrutura de dados com essa característica.

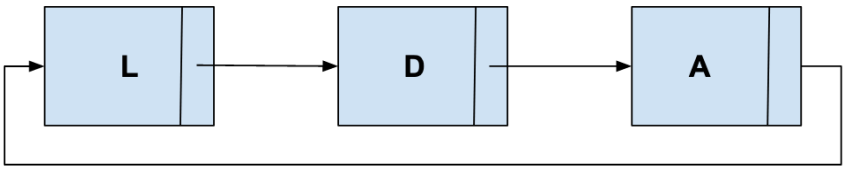


Lista duplamente encadeada

Portanto, a alternativa correta é a letra C.

Agora vamos entender porque as outras alternativas estão incorretas:

A) Incorreta. A lista circular é uma lista encadeada linear cujos elementos sempre possuem o endereço do próximo nó (o endereço nunca é nulo).

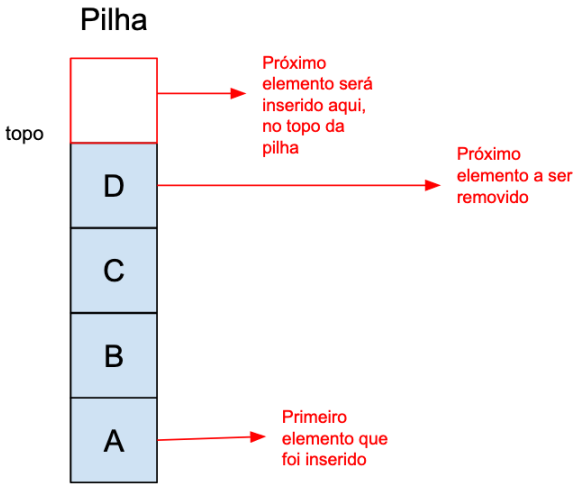


Lista circular

B) Incorreta. Grafo é uma estrutura de dados que consiste em um conjunto de nós (ou vértices) e um conjunto de arcos (ou arestas).

D) Incorreta. Árvore é uma estrutura de dados hierárquica que possui um único elemento raiz, com zero ou mais subárvores ligadas a ele.

E) Incorreta. Pilha é uma estrutura de dados dinâmica e ordenada. Os elementos são inseridos e removidos da mesma extremidade (topo).



4.

Uma estrutura de dados em que o primeiro elemento inserido seja o primeiro elemento a ser retirado é denominada

a) pilha.

b) matriz.

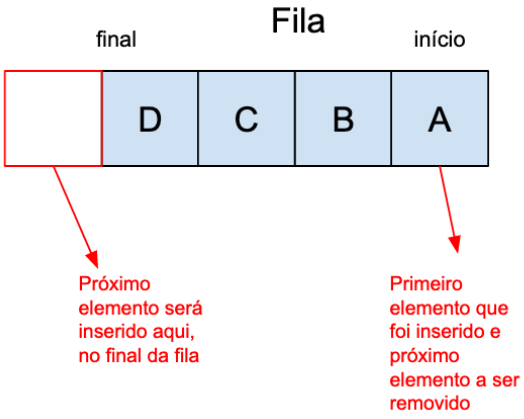
c) árvore binária.

d) fila.

e) lista.

Comentários

Fila é uma estrutura de dados dinâmica e ordenada. Os elementos são inseridos em uma extremidade (final da fila) e removidos na outra extremidade (início da fila). Ou seja, o primeiro elemento inserido será o primeiro elemento a ser retirado



Portanto, a alternativa correta é a letra D.

Agora vamos entender porque as outras alternativas estão incorretas:

A) Incorreta. Pilha é uma estrutura de dados dinâmica e ordenada. Os elementos são inseridos e removidos da mesma extremidade (topo). Logo, o primeiro elemento inserido será o último a ser retirado (LIFO – Last In First Out – último a entrar é o primeiro a sair).

B) Incorreta. Matrizes são estruturas de dados homogêneas – armazenam uma lista de valores do mesmo tipo. Elas possuem mais de uma dimensão. Logo, é preciso mais de um índice para identificar cada elemento. Em matrizes bidimensionais, por exemplo, são necessários dois índices: um para a linha e outro para a coluna. Não há uma ordem para acessar os elementos, eles podem ser acessados diretamente em qualquer ordem.

C) Incorreta. Árvore é uma estrutura de dados hierárquica que possui um único elemento raiz, com zero ou mais subárvores ligadas a ele. Na árvore binária, todos os nós têm, no máximo, dois filhos. O primeiro elemento inserido não é necessariamente o primeiro a ser retirado.

E) Incorreta. Lista encadeada é uma estrutura de dados dinâmica formada por uma sequência encadeada de elementos. O primeiro elemento inserido não é necessariamente o primeiro a ser retirado.

5.

Um compilador está verificando se uma variável foi declarada somente uma vez, se foi declarada antes do seu primeiro uso, se foi declarada e nunca foi usada e se os tipos de dados em uma expressão aritmética são compatíveis. Essas verificações são realizadas na fase de

a) análise sintática.

b) geração do código.

c) análise léxica.

d) análise semântica.

e) otimização do código.

Comentários

A) Incorreta. A análise sintática analisa uma sequência de entrada para determinar sua estrutura gramatical, segundo uma determinada gramática formal.

B) Incorreta. Geração de código pode ser de código intermediário ou código final.

Geração de Código Intermediário: nesta fase, ocorre a transformação da árvore sintática em uma representação intermediária do código fonte. Esta linguagem intermediária é mais próxima da linguagem objeto do que o código fonte, mas ainda permite uma manipulação mais fácil do que se o código Assembly ou código de máquina fosse utilizado.

Geração de Código Final: ocorre a geração do código de montagem (ainda não é o código-objeto, apesar de muitos compiladores já realizarem a montagem).

C) Incorreta. Um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos.

D) Correta. Entre as principais atividades da análise semântica, estão a checagem de tipos, verificação de fluxos de controle e verificação de unicidade de declaração de variáveis. Essa fase também é encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração.

E) Incorreta. Na fase Otimização de Código Intermediário, examina-se o código intermediário produzido durante a fase anterior com objetivo de produzir, através de algumas técnicas, um código que execute com bastante eficiência o programa.

6.

Na compilação, a análise consiste em três fases. Em uma das fases, os caracteres ou tokens são agrupados hierarquicamente em coleções aninhadas com significado coletivo. Essa fase envolve o agrupamento dos tokens do programa fonte em frases gramaticais, que são usadas pelo compilador, a fim de sintetizar a saída. Usualmente, as frases gramaticais do programa fonte são representadas por uma árvore gramatical.

a) sintática.

b) semântica.

c) léxica.

d) binária.

e) linear.

Comentários

A) Correta. A análise sintática pega os tokens resultantes do processo de análise léxica e joga em uma estrutura hierárquica, como uma árvore. O analisador sintático consulta a tabela de símbolos para verificar a presença de variáveis definidas pelo programador.

B) Incorreta. Entre as principais atividades da análise semântica, estão a checagem de tipos, verificação de fluxos de controle e verificação de unicidade de declaração de variáveis. Essa fase também é encarregada de analisar a utilização dos identificadores e de ligar cada uma delas a sua declaração.

C) Incorreta. Um analisador léxico (ou Scanner) lê o código-fonte caractere por caractere e divide o código escrito em símbolos léxicos chamados tokens, guardados em uma tabela de símbolos. A Análise Léxica é uma forma de verificar um alfabeto, ou seja, conseguimos verificar se existe ou não algum caractere que não faz parte do alfabeto.

D) Incorreta. No processo de compilação, não existe a fase análise binária.

E) Incorreta. No processo de compilação, não existe a fase análise linear.

...

Forte abraço e bons estudos!

"Hoje, o 'Eu não sei', se tornou o 'Eu ainda não sei'"

(Bill Gates)

Thiago Cavalcanti

Face: www.facebook.com/profthiagocavalcanti
Insta: www.instagram.com/prof.thiago.cavalcanti
YouTube: youtube.com/profthiagocavalcanti

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.