UNIP
EDUCAÇÃO, POR SUA ESCOLHA

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTITUCIONAL ESPECÍFICO

TOMO 11

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 11

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Língua Portuguesa e Literatura, Especialista em Recursos Gramaticais para Revisão Textual, Engenheira Química e Licenciada em Matemática, com Aperfeiçoamento em Estatística e MBA em Engenharia Econômica. Professora titular da Universidade Paulista.

Tiago Guglielmeti Correale

Doutor e Mestre em Engenharia Elétrica e Engenheiro Elétrico - ênfase em Telecomunicações. Professor titular da Universidade Paulista.

Questões 1 e 2

Questão 1.[1]

Leia o texto a seguir.

Quando um computador é multiprogramado, ele geralmente tem múltiplos processos ou threads que competem pela CPU ao mesmo tempo. Essa situação ocorre sempre que dois ou mais processos estão simultaneamente no estado pronto. Se somente uma CPU se encontrar disponível, deverá ser feita uma escolha de qual processo executar em seguida. A parte do sistema operacional que faz a escolha é chamada de escalonador, e o algoritmo que ele usa é o algoritmo de escalonamento.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 3. ed., São Paulo: Pearson, 2010 (com adaptações).

Considerando que em ambientes diferentes são necessários algoritmos diferentes de escalonamento, garantindo assim que seja maximizado o uso de seus recursos, assinale a opção que apresenta um algoritmo de escalonamento seguido do tipo de ambiente no qual deva ser implementado.

A. Primeiro a chegar, último a sair (*first in, last out* - FILO); propício para sistemas de tempo real.
B. Escalonamento por taxas monotônicas (*rate monotonic scheduling* - RMS); propício para sistemas em lote.
C. Tarefa mais curta primeiro; propício para sistemas interativos.
D. Escalonamento por chave circular (*round-robin*); propício para sistemas de tempo real.
E. Escalonamento por prioridades; propício para sistemas interativos.

Questão 2.[2]

Durante parte do tempo, um processo está ocupado realizando computações internas e outras coisas que não levam a condições de corrida. No entanto, às vezes, um processo tem de acessar uma memória compartilhada ou arquivos, ou realizar outras tarefas críticas que podem levar a corridas. Essa parte do programa onde a memória compartilhada é acessada é chamada de **região crítica** ou **seção crítica**. Se conseguíssemos arranjar as coisas de maneira que dois processos jamais estivessem em suas regiões críticas ao mesmo tempo, poderíamos evitar as corridas. Embora essa exigência evite as condições de corrida, ela não é suficiente para garantir que processos em paralelo cooperem de modo correto e eficiente usando dados compartilhados. Precisamos que quatro condições se mantenham para chegar a uma boa solução.

1. Dois processos jamais podem simultaneamente estar dentro de suas regiões críticas.
2. Nenhuma suposição pode ser feita a respeito de velocidades ou de número de CPUs.

[1]Questão 9 – Enade 2021.

[2]Questão 17 – Enade 2021.

3. Nenhum processo executando fora de sua região crítica pode bloquear qualquer processo.

4. Nenhum processo deve ser obrigado a esperar eternamente para entrar em sua região crítica.

Em um sentido abstrato, o comportamento que queremos é mostrado na figura a seguir.

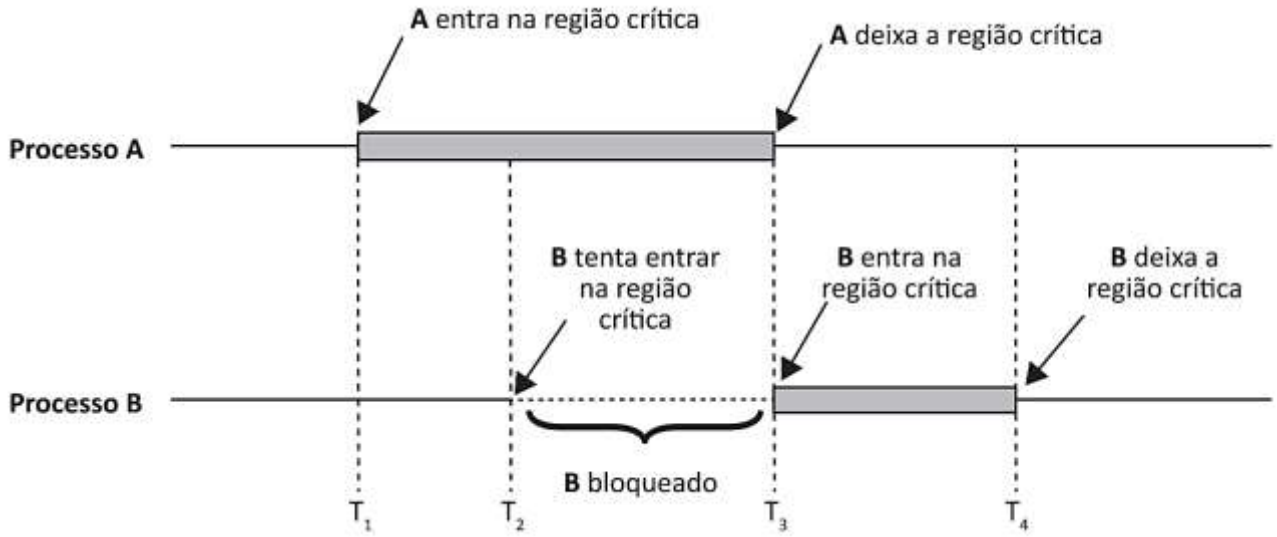


Figura. Exclusão mútua usando regiões críticas.
TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 4.ed. Versão para Biblioteca Virtual Pearson. São Paulo: Pearson Education do Brasil, p. 83, 2016 (com adaptações).

Considerando o texto e a figura, avalie as asserções e a relação proposta entre elas.

I. Em algumas situações, a exclusão mútua pode ser obtida por meio da desabilitação da interrupção controlada pelo Sistema Operacional, não sendo permitido que o seu controle seja feito pelo usuário.

PORQUE

II. A desabilitação da interrupção é uma técnica que pode impedir que o processador que está executando um processo em sua região crítica seja interrompido para executar outro código, sendo mais eficiente em sistemas de multiprocessadores devido a quantidade de processos concorrentes.

A respeito dessas asserções, assinale a opção correta.

A. As asserções I e II são proposições verdadeiras, e a asserção II justifica a I.
B. As asserções I e II são proposições verdadeiras, e a asserção II não justifica a I.
C. A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
D. A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
E. As asserções I e II são proposições falsas.

1. Introdução teórica

1.1. Processador

Também conhecido pelo termo em inglês *Central Processing Unit* (CPU), o processador é o elemento central de um sistema computacional, sendo responsável pela execução de operações, pelo processamento dos dados do computador e pelo controle das operações de entrada e saída de dados. Boa parte dos sistemas computacionais tem por base a arquitetura de Von Neumann, que divide o computador em uma área de processamento (CPU), uma área de armazenamento de dados (memória) e em operações de entrada e saída. Os processadores nativos dessa arquitetura seguem uma lógica bastante parecida para funcionar, já que executam instruções sequenciais armazenadas na sua memória, gerando um resultado como saída.

A CPU é composta:

- pela "Unidade Lógica Aritmética" (ULA), responsável pelos cálculos matemáticos binários do processador; pelos registradores;
- pelos barramentos e pela unidade de controle, com o gerenciamento da sequência de entrada e saída do processador.

Frequentemente, as CPUs também têm uma pequena quantidade de memória auxiliar interna chamada de memória cache, em que as instruções, os endereços e os resultados temporários ficam armazenados durante os cálculos. Adicionalmente, é comum que a memória cache esteja dividida em diferentes níveis, com capacidades de armazenamento e velocidades variadas. As ligações entre os componentes da CPU são realizadas por barramentos, que funcionam como estradas por onde dados, instruções e endereços são conduzidos.

Os processadores atuais podem conter diversos núcleos (parte central da CPU), que funcionam como unidades independentes de processamento. A maioria das CPUs utilizadas atualmente funciona como um circuito digital síncrono, requerendo um sinal de *clock* para o sincronismo das operações. Esses sinais são tipicamente uma série de pulsos (como uma onda quadrada), com taxa ou velocidade medida em Hertz (Hz), que é uma unidade de frequência.

Veja a figura 1.

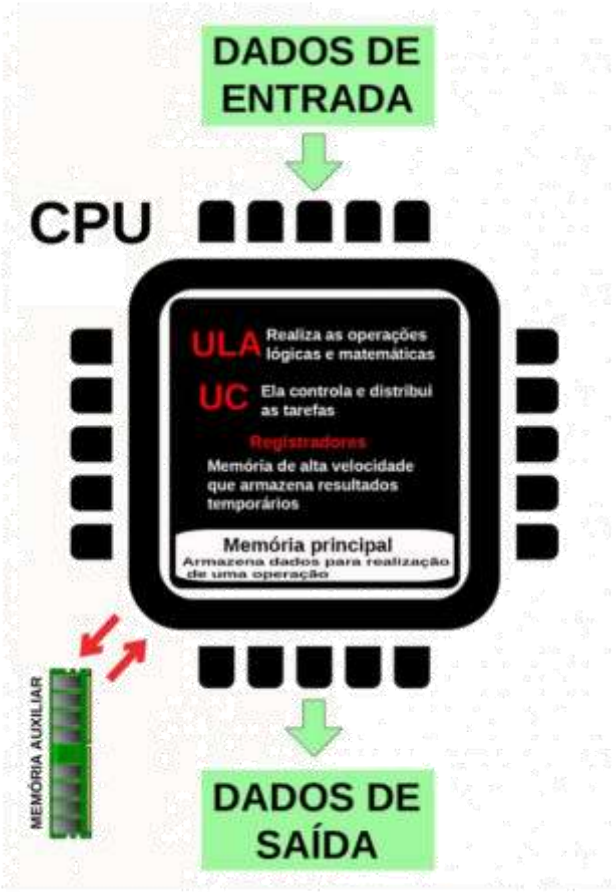


Figura 1. CPU.
Disponível em <https://materiasparaconcursos.com.br/wp-content/uploads/2020/10/pinterest-CPU-683x1024.jpg>. Acesso em 11 jul. 2025 (com adaptações).

Do ponto de vista da execução das instruções, existem diversas tecnologias disponíveis no mercado. No passado, uma CPU física podia executar uma única instrução de cada vez, sendo chamadas de CPUs escalares. Com o tempo, surgiram tecnologias mais avançadas, como CPUs superescalares e CPUs capazes de trabalhar com SMT (*Simultaneous Multithreading*). Nesses casos, uma única CPU física pode executar mais de uma instrução simultaneamente. Para orientar o consumidor na compra dessas tecnologias, alguns fabricantes utilizam os termos "núcleos físicos" e "núcleos lógicos". Ainda que a analogia não seja perfeita, podemos pensar, de forma superficial, que o núcleo lógico se refere à ideia básica de uma CPU que executa uma instrução de cada vez, enquanto um núcleo físico é composto de vários núcleos lógicos.

Precisamos agora entender a associação entre núcleos e processos em execução. Sabemos que um processo (ou uma tarefa) é composto por várias instruções e dados. Se considerarmos um caso simples de uma CPU escalar que não disponha de nenhuma tecnologia de SMT (*Simultaneous Multithreading*) ou se considerarmos apenas núcleos lógicos simples, poderemos observar que cada núcleo lógico deve estar associado a um único processo, no

máximo, em dado instante (um núcleo poderia estar ocioso, caso não existissem instruções a serem executadas).

Contudo, do ponto de vista do usuário de um computador, a impressão é bastante diferente. Muitas vezes, um usuário pode ter a sensação de que um computador está executando um número de tarefas muito maior do que o número de núcleos lógicos disponíveis na máquina. Isso ocorre devido à elevada velocidade de execução e da alternância entre diferentes tarefas. Diversos recursos e fatores ajudam o processador a desempenhar tarefas de forma mais otimizada. Por exemplo:

- a memória cache aumenta o desempenho do processador ao armazenar instruções e dados que são frequentemente utilizados;
- o uso de pipelines melhora a eficiência ao permitir a execução de instruções em diferentes partes do processador;
- instruções avançadas como *Single Instruction, Multiple Data* (SIMD) e *Simultaneous Multithreading* (SMT) permitem a execução eficiente de operações em paralelo e a manipulação de várias *threads* simultaneamente.

Além disso, tecnologias avançadas, como execução fora de ordem, previsão de ramificação e virtualização, também contribuem para a eficiência e a diminuição do consumo de energia.

1.2. Escalonamento de processos

Um dos conceitos mais importantes na área de sistemas operacionais é o conceito de processo. Em sua definição mais básica, dizemos um processo é um programa em execução. Dessa forma, um processo é um programa que está sendo executado pelo microprocessador (CPU) e é gerenciado pelo sistema operacional. Para poder gerenciar os processos em execução, o sistema operacional associa parâmetros, estados e recursos aos processos. É importante termos em mente que uma única aplicação pode ser composta por vários processos e que, em alguns casos, um mesmo programa pode gerar vários processos.

Há, essencialmente, quatro situações geradoras de processos, explicadas a seguir.

1. Processo de inicialização: logo que iniciamos um computador (por exemplo, quando ligamos uma máquina), o processo de inicialização (popularmente chamado de *boot*) costuma gerar uma série de processos, especialmente aqueles ligados diretamente ao funcionamento do sistema operacional.

2. Forma programática: é possível escrever um programa no qual um processo gera outros processos.
3. Criação de processo: o usuário do computador pode criar um novo processo (isso acontece, por exemplo, quando executamos um programa, como um jogo ou um editor de texto).
4. Programas especiais: são os chamados de programas em lote (*job*, em inglês), que podem gerar vários processos ao longo da sua execução.

Com essas quatro possibilidades, percebemos que existem várias situações capazes de gerar processos em execução. Isso significa que o número de processos sendo gerenciados pelo sistema operacional em dado instante pode ser bem grande, provavelmente bem maior do que o número de núcleos (físicos ou lógicos) de uma máquina. Na realidade, é bastante comum o cenário em que uma máquina comum, com um único usuário utilizando a máquina de forma simples, tenha mais de uma centena de processos em diversos estados de execução. Em servidores, esse número costuma ser ainda maior.

Assim, temos um cenário no qual o número de processos que requerem recursos de processamento é muito maior do que o número de recursos disponíveis. O sistema operacional deve gerenciar essa assimetria e distribuir o tempo de processamento da forma mais adequada a determinado cenário de utilização. Essa é a função do escalonador de processos do sistema operacional.

Frequentemente, o sistema operacional escalona os processos concorrentes que são executados por meio do compartilhamento de tempo (*time sharing*). Isso acontece em sistemas operacionais multiprogramados, também chamados de preemptivos, cuja característica principal é o revezamento da execução de vários processos de forma parcial e em grande velocidade.

Do ponto de vista do usuário, no revezamento de processos, devem ser levados em conta dois detalhes: (1) um processo executado é parte de uma aplicação; (2) cada processo não precisa necessariamente ser executado até o fim para passar pelo processo de revezamento. Isso gera para o usuário a sensação de que todos os processos – e consequentemente todas as aplicações – são executados ao mesmo tempo. Por isso, sistemas operacionais que utilizam o conceito de *time sharing* são chamados de multitarefas.

Para gerenciar a execução dos diversos processos, o escalonador do sistema operacional associa diversos estados a cada um dos processos ao longo da sua vida. Nesse contexto, há basicamente cinco estados possíveis para um processo, indicados a seguir.

1. **Novo**
2. **Pronto**
3. **Em Execução**
4. **Em Espera**
5. **Encerrado**

O primeiro estado, chamado de "Novo", é associado a um processo assim que ele é criado. O estado "Pronto" é associado aos processos que estão prontos para serem executados, mas que aguardam a sua associação a um núcleo para a sua execução. Os processos cujas instruções estão sendo executados em algum núcleo são associados ao estado "Em Execução". Quando um processo está aguardando alguma informação para continuar o processamento (por exemplo, quando ele depende de uma informação vinda da rede ou uma interação do usuário), ele é associado ao estado "Em Espera". Finalmente, o estado "Encerrado" é associado ao processo que vai finalizar a sua execução.

Existem vários tipos de processos com diferentes perfis de execução. Por exemplo, há processos cuja execução é caracterizada por grande quantidade de cálculos, com pouca troca de informação entre dispositivos externos. Dizemos que esses tipos de processos são vinculados ao processador (*CPU bound*). Mas há processos que apresentam um perfil de execução bastante diferente: eles dependem de muitos dados vindos de outros dispositivos, como discos rígidos ou da rede, mas exigem pouco esforço por parte do processador. Dizemos que esses processos são vinculados à entrada/saída (*Input/Output bound* ou *I/O bound*).

Há vários algoritmos que regem a preempção, como o *First In/Last Out* (FILO), *First In/First Out* (FIFO) e o *Round Robin*. Em todos eles, para chegar ao tempo total que um processo ocupa do início ao fim da execução, devemos somar o *quantum* de cada execução do processo com o tempo de contexto de cada uma dessas execuções.

1.3. Threads

Uma *thread*, em termos de programação e sistemas operacionais, é a menor unidade de execução em um processo. Dado que um processo é um programa em execução, dizemos que uma *thread* é uma subdivisão desse processo que pode executar operações independentes. Cada *thread* em um processo compartilha os mesmos recursos e o mesmo espaço de memória, mas tem seu próprio contador de programa, seus próprios registradores e sua própria pilha de execução.

Multithreading é a técnica de programação em que múltiplas *threads* são utilizadas para realizar tarefas concorrentes em um único processo. A principal vantagem do uso de *multithreads* está na capacidade de realizar operações simultâneas, o que pode resultar em melhor desempenho e responsividade de um programa.

Apesar de as *multithreads* serem amplamente utilizadas em programação concorrente para melhorar a eficiência e a responsividade dos programas, o desenvolvimento de software *multithreaded* apresenta desafios, e os programadores devem estar cientes das implicações da concorrência para evitar problemas como condições de corrida e *deadlocks*.

1.4. Condições de corrida

Em processadores ou sistemas multitarefas, uma condição de corrida ocorre quando a saída de um programa depende da ordem de execução de operações concorrentes. Essas condições podem levar a resultados inesperados e indesejados devido à concorrência não controlada entre diferentes threads ou processos. As condições de corrida são um tipo comum de problema em sistemas concorrentes e podem ser difíceis de identificar e corrigir. Existem alguns cenários típicos que podem resultar em condições de corrida, mencionados a seguir.

- Leitura-escrita concorrente: quando várias *threads* ou processos tentam ler e escrever dados compartilhados simultaneamente, podem ocorrer condições de corrida. Se uma *thread* estiver lendo um dado enquanto outra estiver escrevendo nele, o resultado pode depender da ordem em que as operações são executadas.
- Escrita-escrita concorrente: duas ou mais operações de escrita concorrentes em uma variável compartilhada podem causar condições de corrida. A última escrita pode sobrescrever as alterações feitas por uma escrita anterior, resultando em um estado inconsistente.
- Atualização não atômica: operações que envolvem a leitura, a modificação e a gravação de dados (como incremento) podem ser suscetíveis a condições de corrida se não forem realizadas de maneira atômica. Se duas *threads* tentarem incrementar uma variável simultaneamente, o valor final pode não ser o esperado.
- Acesso não sincronizado a recursos compartilhados: quando várias *thread*s têm acesso a recursos compartilhados sem a devida sincronização, problemas podem surgir. Por exemplo, se duas threads tentarem acessar uma lista ligada simultaneamente, a estrutura da lista pode ficar corrompida.

- Concorrência não controlada: a falta de mecanismos de sincronização adequados, como semáforos, mutexes ou monitores, pode resultar em execução concorrente não controlada, levando a condições de corrida.

Para exemplificarmos um problema de condição de corrida, vamos pensar no funcionamento de um servidor de impressão. Um servidor de impressão recebe um trabalho de impressão (por exemplo, um documento de texto ou uma imagem) e deve controlar o envio desse trabalho para uma impressora. Considere o contexto de um escritório, no qual existem vários computadores conectados em rede, mas apenas uma única impressora. Os diversos computadores devem compartilhar a única impressora disponível utilizando o servidor de impressão (atualmente, muitas impressoras têm um servidor de impressão incorporado e podem ser diretamente conectadas na rede).

Considere o cenário no qual duas impressões, A e B, são enviadas uma em sequência da outra, mas em uma velocidade muito maior do que a velocidade de impressão da impressora. A impressão A tem seu pedido reservado na memória do serviço de *spool* (nome do serviço de impressão). Durante o funcionamento normal, o servidor de impressão deve primeiro processar a impressão A até a sua conclusão e, posteriormente, processar a impressão B até o seu fim. Contudo, imagine que exista um *bug* (erro) no servidor de impressão e que o servidor permita que parte da impressão A seja misturada, de forma intercalada, com a impressão B, de forma que os dois trabalhos fiquem completamente misturados. Esse cenário de "confusão" exemplifica, de forma material, o problema do compartilhamento de recursos.

Um cenário similar a esse pode ocorrer em um sistema multiprogramado ou em um programa que execute várias *threads* simultaneamente, mesmo que o recurso compartilhado seja apenas uma posição de memória. No caso de programas com problemas de condição de corrida, um caso patológico típico envolve vários processos entrando simultaneamente na sua região crítica e sobrescrevendo uma mesma região de memória compartilhada, o que pode gerar um comportamento inesperado, como falhas ou corrupção dos dados.

A tabela 1 mostra algumas técnicas de exclusão mútua que buscam solucionar a ocorrência de situações desse tipo.

Tabela 1. Soluções possíveis para a condição de corrida.

Algoritmos	Descrição	Vantagens	Desvantagens
Exclusão mútua com espera ocupada	Quando um processo entra na zona crítica, impede que outros entrem	Evita a condição de corrida	Consumo de processamento: os processos concorrentes ficam constantemente verificando se a zona crítica está liberada
Exclusão mútua com inibição de interrupções	O processo inibe todas as interrupções logo após entrar em uma zona crítica, e as habilita ao sair	O processo não é interrompido mesmo se acabar a sua fatia de tempo de execução	Sistema operacional monousuário: a capacidade de inibir interrupções pode ser mal-usada pelos processos
Exclusão mútua com variáveis de travamento	Antes de entrar na zona crítica, o processo verifica uma variável (V = há processos na zona crítica ou F = não há processos na zona crítica) para ver se já existe outro processo manipulando a memória compartilhada		Consumo de processamento na verificação da variável Se dois processos testarem a variável ao mesmo tempo podem entrar juntos na zona crítica
Exclusão mútua com estrita alternância	Testa uma variável (vez) e define qual processo deve entrar na zona crítica Se vez == 'A', então é a vez do processo A. Se vez == 'B', é a vez do processo B	Evita que dois ou mais processos entrem ao mesmo tempo na região crítica	Consumo de processamento: teste contínuo da variável Um processo só terá vez novamente se o outro entrar e sair da sua zona crítica Esquema inadequado quando um processo é mais lento do que o outro
Exclusão mútua com bloqueio e desbloqueio	Bloqueia a execução do processo quando a sua entrada na zona crítica não for permitida	Não desperdiça processamento	

2. Análise das alternativas e das asserções

Questão 1

A – Alternativa incorreta.

JUSTIFICATIVA. Um sistema de tempo real é aquele que deve executar determinada tarefa em tempo previsível e, em geral, o mais rapidamente possível. Um sistema em tempo real falha quando não executa uma tarefa dentro dos limites pré-estabelecidos. Como exemplo dessa situação, considere um carro autônomo que detectou um obstáculo em seu caminho, mas não parou a tempo, causando um acidente. Esse é um exemplo catastrófico de um sistema de tempo real que não funcionou. Um algoritmo de escalonamento do tipo FILO é

aquele que atende as requisições (ou tarefas) mais recentes, deixando as primeiras requisições por último. Não é o algoritmo ideal para sistemas de tempo real, pois uma requisição inicial enviada para um algoritmo FILO poderia ter sua execução postergada para o atendimento de novas requisições menos prioritárias.

B – Alternativa incorreta.
JUSTIFICATIVA. O RMS é um algoritmo baseado em prioridades fixas que ordena os processos com base na sua frequência, sendo que os processos com a maior frequência têm prioridade maior. Portanto, esse tipo de algoritmo não é apropriado para sistema em lote, mas sim para sistemas de tempo real, que precisam executar uma tarefa em um momento ou prazo especificado.

C – Alternativa incorreta.
JUSTIFICATIVA. Um algoritmo de escalonamento cuja tarefa mais curta termine primeiro não é adequado para sistemas interativos, que esperam a interação do usuário para algumas operações. O motivo é que não há vantagem alguma na realização de tarefas curtas, uma vez que vai ser a interação do usuário que determinará o andamento do processo. Muitas vezes, tarefas mais longas podem ser priorizadas, já que a interação do usuário pode demorar.

D – Alternativa incorreta.
JUSTIFICATIVA. Sistemas do tipo round-robin realizam um rodízio entre as tarefas usando um tempo fixo de execução para cada uma delas. Não são apropriados para sistemas de tempo real justamente porque esses algoritmos não assumem prioridades na execução – fato necessário no escalonamento de sistemas de tempo real.

E – Alternativa correta.
JUSTIFICATIVA. Um algoritmo preemptivo que tem por base prioridades é aquele que executa as tarefas prioritárias primeiro. A forma como acontece a atribuição de prioridades (estática ou dinâmica) é importante nesses algoritmos. Um sistema interativo que espera ações externas pode ter vantagens ao utilizar esse tipo de algoritmo, pois as tarefas mais importantes (prioritárias) seriam executadas primeiro e dependeriam apenas da ação externa para a continuidade.

Questão 2

I – Asserção verdadeira.
JUSTIFICATIVA. Isso ocorre por meio de algumas opções de exclusão mútua que estão listadas na tabela 1. A desabilitação realizada pelo usuário ou por algum processo não é recomendada, já que existe a possibilidade de o processo em execução monopolizar o uso do microprocessador.

II – Asserção falsa.
JUSTIFICATIVA. Realmente, a desabilitação da interrupção evita a interrupção de uma tarefa para a execução de outro processo. Contudo, a afirmativa é equivocada, uma vez que a técnica é mais eficaz em sistemas com um único processador. Quando o sistema é multiprocessado, tarefas concorrentes podem ser executadas simultaneamente nos diferentes processadores, possibilitando o acesso à zona crítica, o que invalida a solução.

Alternativa correta: C.

3. Indicações bibliográficas

- MAZIERO, C. A. *Sistemas operacionais: conceitos e mecanismos*. Curitiba: DINF-UFPR, 2019. Disponível em <https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf>. Acesso em 25 dez. 2023.
- OSHANA, R. *DSP Software Development Techniques for Embedded and Real-Time Systems*. Burlington: Newnes, 2006, p. 261-320.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de sistemas operacionais*. Rio de Janeiro: LTC, 2013.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 5. ed. São Paulo: Pearson Brasil, 2024.

Questão 3

Questão 3.[3]

A biblioteca de coleções da linguagem Java disponibiliza implementações de propósito geral para estruturas de dados elementares, como listas, filas e pilhas. Considere as seguintes definições de classes que representam implementações de estruturas de dados disponíveis na biblioteca da linguagem.

- **Classe A.** Os objetos são organizados em uma ordem linear e podem ser inseridos somente no início ou no final dessa sequência.
- **Classe B.** Os objetos são organizados em uma ordem linear determinada por uma referência ao próximo objeto.
- **Classe C.** Os objetos são removidos na ordem oposta em que foram inseridos.
- **Classe D**. Os objetos são inseridos e removidos respeitando a seguinte regra: o elemento a ser removido é sempre aquele que foi inserido primeiro.

Nesse contexto, assinale a alternativa que representa, respectivamente, as estruturas de dados implementadas pelas classes A, B, C e D.

A. Lista circular, lista simplesmente ligada, pilha e fila.
B. Deque, lista simplesmente ligada, pilha e fila.
C. Lista duplamente ligada, lista simplesmente ligada, fila e pilha.
D. Pilha, fila, deque e lista simplesmente encadeada.
E. Deque, pilha, lista ligada e fila.

1. Introdução teórica

1.1. Linguagem Java

Java é uma linguagem de programação de propósito geral, orientada a objetos, desenvolvida pela empresa Sun Microsystems na década de 1990, tendo como principal característica a portabilidade, permitindo que um mesmo código possa ser executado em diferentes plataformas. Isso é possível devido à presença da máquina virtual Java - *Java Virtual Machine* (JVM).

O código-fonte Java é compilado para um código intermediário chamado bytecode, que é então executado pela JVM. Isso permite que o mesmo código Java seja executado em

[3]Questão 10 – Enade 2021

diferentes plataformas sem modificação, desde que elas tenham uma implementação da máquina virtual Java.

A execução de código Java ocorre em um ambiente controlado pela JVM, o que impede certos tipos de atividades maliciosas e contribui muito para a segurança. Além disso, a linguagem Java tem recursos de gerenciamento de memória automáticos, ajudando a evitar muitos tipos de erros de programação que podem levar a vulnerabilidades.

A linguagem Java também conta com um ecossistema robusto de bibliotecas e frameworks que facilitam o desenvolvimento de aplicativos em diferentes domínios, como, por exemplo, desenvolvimento *web* (*Spring framework*), banco de dados (*Hibernate*) e interfaces gráficas (JavaFX).

1.2. Estruturas de dados

As estruturas de dados são formas de organizar, distribuir e relacionar os dados disponíveis de modo a tornar mais eficientes os algoritmos que manipulam esses dados. Há grande variedade de estruturas de dados que podemos utilizar na programação, como, por exemplo: listas, filas, pilhas, deques, listas circulares, árvores (de diversos tipos, como árvores binárias, binárias de busca e não binárias), grafos e tabelas *hash*.

Existem cada vez mais aplicações para resolvermos problemas envolvendo grande quantidade de dados. É imprescindível, quando se projeta um sistema computacional, que a escolha dos recursos de programação que irão manipular esses dados seja o mais eficiente possível.

A linguagem Java, por exemplo, apresenta uma coleção completa de estruturas de dados (*Collections Framework*) que representa e trata grupos de dados como uma unidade única a ser utilizada. Mas é importante entender como cada implementação funciona internamente. Só assim é possível decidir qual é a melhor opção para o programa a ser desenvolvido.

As diversas formas de construir e conectar estruturas de dados são muito similares. A diferença está na forma como manipulamos cada estrutura, ou ainda, nas operações que realizamos em cada uma delas.

Uma pilha é uma coleção dinâmica de dados que estabelece uma política de entrada e saída conhecida como *Last In - First Out* (LIFO), ou seja, o último elemento a entrar é o primeiro a sair da pilha. Os elementos são empilhados um a um e toda inserção e toda remoção são feitas no topo. A disciplina de acesso da pilha deve garantir que apenas o último

elemento empilhado possa ser removido da pilha. As pilhas são muito usadas no processamento de linguagens, por exemplo, em compiladores.

A fila (*queue*) é uma estrutura de dados dinâmica que admite remoção de elementos e inserção de novos objetos. Ela estabelece uma política de entrada e saída do tipo *First In - First Out* (FIFO), ou seja, o primeiro elemento a entrar na lista é o primeiro a sair. Os elementos são sempre inseridos no início da lista e removidos do final da lista. Uma variação ocorre em situações nas quais é preciso priorizar determinado elemento para que ele "fure a fila". Para isso, existe a fila de prioridades (*priority queue*).

Outra estrutura de dados similar é chamada de "deques" (do inglês, *Double Ended Queue*), que são filas duplamente ligadas, isto é, filas com algum tipo de prioridade. Esse tipo de estrutura de dados generaliza a ideia de uma fila, incluindo a possibilidade de adicionar ou de remover elementos no seu topo (*head*) ou na sua base *(tail*). Os deques são usados, por exemplo, em sistemas distribuídos que, muitas vezes, priorizam os processos mais rápidos enquanto armazenam os mais lentos em uma fila de espera. O deque pode ser implementado seguindo a abordagem circular, pois isso confere eficiência à estrutura ao mesmo tempo em que evita o desperdício de memória.

Uma lista encadeada é uma estrutura mais livre, sem muitas regras. Uma inserção pode ser feita no início, no meio, no final, em qualquer parte da lista de forma crescente ou decrescente. Os elementos que compõem as listas são chamados nós. Cada nó tem ao menos um ponteiro utilizado para "apontar para outro nó".

Dessa forma, uma lista é uma sequência de nós, em que cada nó contém uma informação de algum tipo de dado e o endereço do nó seguinte. Uma lista simplesmente encadeada ou ligada é uma sequência finita de elementos ligados entre si com nós que apontam para os sucessores. As listas simplesmente encadeadas são úteis para representar conjuntos dinâmicos de dados (útil quando não sabemos ou não queremos definir um número máximo de elementos). Na lista simplesmente encadeada, utilizamos o nó inicial para acessar e percorrer a lista, do início ao fim, em uma mesma direção – cada nó aponta para o seu sucessor direto. Mas há também variações desse tipo de estrutura, como a lista duplamente encadeada. Nesse caso, cada nó tem dois ponteiros, um para o nó seguinte e outro para o nó anterior. Assim, é possível percorrer a lista nos dois sentidos, do início para o fim e vice-versa.

2. Análise das alternativas

As classes descritas no enunciado podem ser classificadas da forma a seguir.

- Classe A: deque – elementos podem ser inseridos apenas no topo e na base.
- Classe B: lista simplesmente ligada – os nós seguem uma sequência linear.
- Classe C: pilha – os primeiros a entrar são os últimos a sair, enquanto os últimos a entrar são os primeiros a sair.
- Classe D: fila – os primeiros elementos (nós) a entrar são também os primeiros a sair.

A – Alternativa incorreta.
JUSTIFICATIVA. A primeira opção "lista circular" não corresponde à classe A. Em uma lista circular, o último nó da lista aponta novamente para o primeiro nó da lista. A classe correta da classe A é deque.

B – Alternativa correta.
JUSTIFICATIVA. As opções listadas correspondem exatamente às classes A, B, C e D.

C – Alternativa incorreta.
JUSTIFICATIVA. A única opção corretamente classificada é a atribuída à classe B (lista simplesmente ligada). A classe A é um deque, a classe C é uma pilha e a classe D é uma fila.

D – Alternativa incorreta.
JUSTIFICATIVA. Nenhuma das opções corresponde às classes listadas. Analisando a questão, note que as classes estão fora de ordem. A classe A é um deque, a classe B é uma lista simplesmente ligada, a classe C é uma pilha e a classe D é uma fila.

E – Alternativa incorreta.
JUSTIFICATIVA. A classe B é uma lista simplesmente ligada, não é uma pilha. Já a Classe C é uma pilha e não uma lista ligada.

3. Indicações bibliográficas

- AGUILAR, J. L. *Fundamentos de programação: algoritmos, estrutura de dados e objetos*. Porto Alegre: AMGH, 2011.
- ASCENCIO, A. F. G.; de ARAÚJO, G. S. *Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++*. São Paulo: Pearson Prentice Hall, 2010.

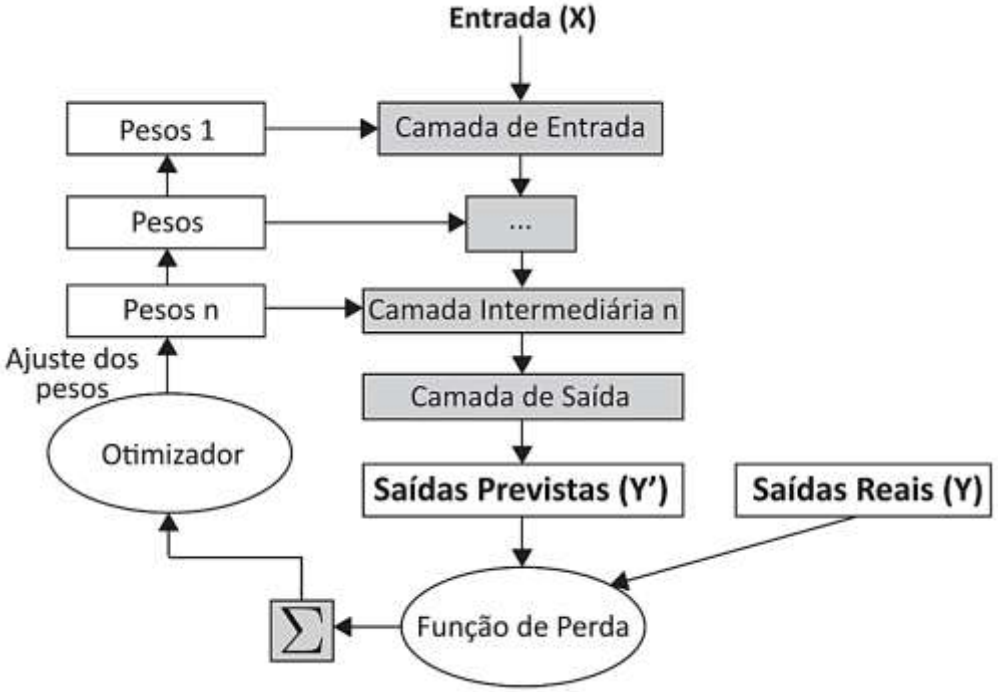
- CORMEN, T. H. et al. *Introduction to algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. *Algoritmos*. Porto Alegre: AMGH, 2010.
- KNUTH, D. E. *The art of computer programming: fundamental algorithms*. 3. ed., v. 1. Boston: Addison-Wesley, 1997.
- LINTZMAYER, C. N.; MOTA, G. O. *Análise de algoritmos e estruturas de dados*. Disponível em <https://www.ime.usp.br/~mota/livros/livro_AAED.pdf>. Acesso em 19 set. 2023.
- MANZANO, J. A. N. G.; COSTA JUNIOR, R. A. *Programação de computadores com Java*. São Paulo: Érica, 2015.
- VETORAZZO, A. S. et al. *Estrutura de dados.* Porto Alegre: Grupo A, 2018.

Questões 4 e 5

Questão 4.[4]

Uma equipe de cientistas da computação de determinada empresa de animação foi designada para desenvolver um sistema capaz de varrer a web no intuito de detectar sites que possam estar usando imagens de seus personagens de animação sem o devido consentimento. Portanto, o sistema deverá receber imagens como entrada, classificá-las entre imagens da empresa e imagens não produzidas pela empresa.

A figura abaixo esboça uma arquitetura de rede neural profunda e o processo de treinamento que os cientistas pretendem usar.



CHOLLET, F. *Deep Learning with Python*. New York: Manning Publications, 2017 (com adaptações).

Após uma tentativa, notaram-se duas dificuldades: 1) o tempo de treinamento da rede estava muito longo e 2) a acurácia da rede treinada não estava no patamar aceito pela empresa.

Diante deste contexto, avalie as afirmativas.

I. Aumentar o número de camadas é uma alternativa que pode levar a uma melhora na acurácia, além de diminuir o tempo de treinamento da rede.

II. Fazer uso de redes convolucionais é uma alternativa que pode levar a uma melhora na acurácia, no entanto, pode exigir uso de máquinas com maior poder de processamento.

III. Aumentar o número de unidades de processamento (neurônios) nas camadas pode levar a uma piora na acurácia, além de diminuir o tempo de treinamento da rede.

[4]Questão 11 – Enade 2021

IV. Aumentar o número de amostras de treinamento é uma alternativa que pode levar a uma melhora na acurácia, apesar de aumentar o tempo de treinamento da rede.

V. Fazer uso de redes recorrentes é uma alternativa que pode levar a uma melhora na acurácia, no entanto, pode exigir uso de máquinas com maior poder de processamento.

É correto apenas o que se afirma em

A. I e IV. B. I e V. C. II e III. D. II e IV. E. III e V.

Questão 5.[5]

Leia o texto a seguir.

As técnicas de aprendizado de máquinas empregam um princípio de inferência denominado indução, no qual é possível obter conclusões genéricas a partir de um conjunto particular de exemplos. Estas técnicas de aprendizados indutivos podem ser divididas em dois principais tipos: os supervisionados e os não supervisionados. No aprendizado supervisionado é fornecida uma referência do objetivo a ser alcançado, isto é, um treinamento com o conhecimento do ambiente. Diferentemente do aprendizado supervisionado, o não supervisionado não utiliza referências, ou seja, não ocorre um treinamento com o conhecimento do ambiente.

PELLUCCI P. R. S. *et al*. Utilização de técnicas de aprendizado de máquina no reconhecimento de entidades nomeadas no português. Belo Horizonte: *E-xacta*, v. 4, n. 1, p. 73-81, 2011 (com adaptações).

Considerando as informações do texto, avalie as afirmativas.

I. A regressão linear é um exemplo de modelo baseado no aprendizado supervisionado.

II. A diferença entre a saída desejada e a saída gerada é o valor do erro de um aprendizado não supervisionado.

III. O aprendizado não supervisionado é mais utilizado quando o entendimento dos dados é feito por meio de reconhecimento de padrões.

IV. O aprendizado supervisionado é capaz de tomar decisões precisas ao receber novos dados a partir de um treinamento com dados conhecidos.

É correto apenas o que se afirma em

A. I e III.

B. II e III.

C. II e IV.

D. I, II e IV.

E. I, III e IV.

[5]Questão 18 – Enade 2021

1. Introdução teórica

1.1. Inteligência Artificial (IA)

Definir o termo "Inteligência Artificial" (IA) é algo complexo e alvo de uma discussão entre especialistas na área desde os seus primórdios. Contudo, uma definição bastante comum é a de que Inteligência Artificial é o estudo de como fazer computadores efetuarem tarefas nas quais seres humanos são atualmente melhores (RICH, 1983). Essas tarefas incluem aprendizado, raciocínio, reconhecimento de padrões, resolução de problemas, compreensão da linguagem natural e interação social. O objetivo da IA é criar sistemas capazes de realizar atividades complexas de forma autônoma, sem intervenção humana direta.

Muitos especialistas costumam dividir a IA em dois tipos principais, citados a seguir.

- IA fraca: refere-se a sistemas projetados para realizar tarefas específicas, como reconhecer voz, jogar xadrez, recomendar produtos ou diagnósticos médicos. Esses sistemas são altamente especializados e não têm a capacidade de realizar tarefas fora de sua área designada.
- IA forte: este é um nível mais avançado de IA, no qual as máquinas têm a capacidade de realizar qualquer tarefa cognitiva humana. Esse tipo de IA ainda não foi totalmente alcançado, e a maioria das aplicações de IA existentes é considerada IA fraca.

As abordagens para implementar sistemas com IA podem ser divididas em duas categorias principais, mencionadas a seguir.

- IA baseada em regras (ou simbólica): envolve a programação de sistemas com regras específicas que orientam o comportamento do sistema. Esse método funciona bem em ambientes controlados, mas pode ser limitado quando se trata de lidar com a complexidade e a imprevisibilidade do mundo real.
- IA baseada em aprendizado de máquina: esta abordagem envolve a criação de algoritmos que permitem que um sistema aprenda padrões e tome decisões com base em dados.

1.2. Aprendizado de máquina

O aprendizado de máquina é a ciência (e a arte) da programação de computadores com o intuito de que eles possam aprender com os dados. Dito isso, há diversas técnicas que

possibilitam que uma máquina possa aprender a partir da análise dos dados aos quais ela é submetida. O aprendizado pode ser dividido em **supervisionado, não supervisionado e por reforço**, sendo que cada um deles adota abordagens específicas para a resolução de problemas.

No **aprendizado supervisionado,** um algoritmo é treinado utilizando um conjunto de dados rotulados, em que as entradas (ou os dados de entrada) estão associadas a saídas conhecidas (utilizando os rótulos). O objetivo é que o algoritmo aprenda a mapear corretamente as entradas para as saídas, de modo que, quando apresentado a novos dados, ele seja capaz de fazer previsões ou classificações com base no que aprendeu durante o treinamento. Entre as características da técnica supervisionada, destacam-se a classificação dos resultados, os previsores que representam particularidades do alvo e a avaliação da correlação entre as variáveis fornecidas, conhecida por regressão. A utilização da técnica supervisionada é bastante comum em sistemas utilizados para a identificação de *spam* (mensagens de e-mail não solicitadas), na definição do valor de automóveis usados, ou quando desejamos obter algum tipo de resposta a partir de padrões já identificados e estabelecidos anteriormente.

Ao contrário do aprendizado supervisionado, o **aprendizado não supervisionado** envolve a utilização de conjuntos de dados não rotulados, a partir dos quais o algoritmo busca encontrar padrões ou estruturas por conta própria. A principal ideia é explorar a estrutura subjacente nos dados, identificar padrões, agrupar dados semelhantes ou reduzir a dimensionalidade do conjunto de dados. Quando há o aprendizado não supervisionado, não há classificação ou categorização dos resultados. A máquina (algoritmo) é responsável por encontrar as correlações e realizar a rotulação ou o agrupamento desses dados. Há diversos algoritmos com essa finalidade, entre os quais podemos destacar os de *clustering* (agrupamento), de visualização e de redução da dimensionalidade, além dos algoritmos de aprendizado da regra da associação. O uso desse tipo de solução é comum, por exemplo, em ferramentas que analisam o tipo e o perfil do público de sites ou de redes sociais ou quando é necessária a identificação de padrões em um grupo extenso.

No **aprendizado por reforço,** um agente interage com um ambiente e toma decisões para alcançar um objetivo específico. O agente recebe recompensas ou penalidades com base em suas ações. Ou seja, é um tipo de aprendizado no qual a máquina (algoritmo) é recompensada quando acerta e é penalizada quando erra. Essa forma de treinamento permite que, a partir de uma técnica de tentativa e erro, o algoritmo vá se tornando cada vez mais especializado e apresente resultados mais precisos. O que se busca é que a máquina aprenda

a tomar ações sequenciais de modo a maximizar a recompensa cumulativa ao longo do tempo. O agente deve explorar diferentes ações e aprender a associar ações a resultados positivos. Entre os exemplos, temos o treinamento de um algoritmo para jogar um jogo, como Xadrez ou Go, em que o agente aprende a tomar decisões para ganhar o jogo ao longo do tempo. Assim, a base de conhecimento interna utilizada pelo algoritmo deve guiar a tomada de decisões ou a geração de informações a ser utilizada na resolução de algum tipo de problema.

1.3. Redes neurais artificiais

As **redes neurais artificiais** são um subconjunto do aprendizado de máquina, cuja estrutura é vagamente inspirada no cérebro humano, imitando a maneira como os neurônios biológicos enviam sinais uns para os outros e como adquirem conhecimento com a experiência.

A rede neural é uma estrutura composta por células computacionais simples, chamadas neurônios, massivamente interconectadas em uma estrutura paralelamente distribuída capaz de armazenar e processar informações para exercer uma tarefa.

Os neurônios nas redes neurais são normalmente organizados em grupos de unidades de processamento chamados de camadas, que, por sua vez, são organizadas em uma cadeia. As diversas camadas de uma rede neural artificial podem ser classificadas da forma mencionada a seguir.

- Camada de entrada: primeira camada, na qual os padrões são apresentados à rede.
- Camadas ocultas: camadas que realizam a maior parte do processamento, por meio das conexões ponderadas e da extração de características.
- Camada de saída: última camada, responsável por apresentar o resultado do processamento.

Cada nó, ou neurônio artificial, pode se conectar a outros nós e tem um peso e um limite associados. Se a saída de qualquer nó individual estiver acima do valor do limite especificado, esse nó será ativado, enviando dados para a próxima camada da rede. Caso contrário, nenhum dado será transmitido para a próxima camada da rede.

As redes neurais contam com dados de treinamento para aprender e melhorar a precisão e o desempenho ao longo do tempo, fato que as transforma em ferramentas poderosas para aplicações como agrupamento e/ou classificação de dados em altas velocidades. Por exemplo, as tarefas de reconhecimento de fala ou de reconhecimento de

imagem podem levar minutos em vez de horas, quando comparadas à identificação manual feita por especialistas humanos.

Na IA, o Aprendizado Profundo (*Deep Learning*) é uma subcategoria dos algoritmos de *Machine Learning* (ML) que utiliza redes neurais artificiais com muitas camadas (por isso, o nome "rede neural profunda") para aprender e realizar tarefas complexas, como reconhecimento de imagens, processamento de linguagem natural e uso de jogos estratégicos.

O aprendizado profundo engloba estruturas diversas, que oferecem múltiplas camadas de processamento, mas as redes neurais concentram a maior parte dos esforços de pesquisa dos algoritmos utilizados.

No aprendizado profundo, o aumento no número de camadas ocultas permite que cada uma delas se especialize em identificar um conjunto de características em particular. As camadas mais próximas da camada de entrada são responsáveis por identificar as características mais primitivas, como arestas em uma imagem, enquanto as seguintes combinam essas informações para identificar padrões mais complexos, como animais ou semáforos na mesma imagem.

As redes neurais podem ser classificadas em diferentes tipos e podem ser usadas para diferentes propósitos. Embora esta não seja uma lista abrangente, podemos citar os exemplos a seguir.

- *Perceptron*: é a rede neural mais antiga, criada por Frank Rosenblatt em 1958. Tem um único neurônio e é a forma mais simples de uma rede neural.
- *Feedforward* ou *perceptrons* multicamadas: são compostas por uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Essas redes neurais passam por treinamento e são a base para a visão computacional e o processamento de linguagem natural.
- Redes neurais convolucionais: são semelhantes às redes *feedforward*, mas geralmente são utilizadas para reconhecimento de imagem, reconhecimento de padrões e/ou visão computacional. Essas redes aproveitam os princípios de álgebra linear, particularmente a multiplicação de matrizes, para identificar padrões em uma imagem.
- Redes neurais recorrentes: são identificadas por seus *loops* de *feedbac*k – a saída desse tipo de rede é realimentada à entrada. Devido à realimentação, as redes neurais recorrentes consideram tempo e sequência. Por isso, elas têm capacidade de memória de curto prazo, sendo indicadas para reconhecer padrões em

sequências de dados, como textos, genomas, caligrafia, palavras faladas (áudio), ou dados de séries numéricas que emanam de sensores, bolsas de valores e agências governamentais.

2. Análise das afirmativas

Questão 4

I – Afirmativa incorreta.
JUSTIFICATIVA. O aumento do número de camadas aumenta o processamento e consequentemente a precisão (acurácia). Contudo, esse aumento no número de camadas também costuma aumentar o tempo de treinamento, em vez de diminuí-lo.

II – Afirmativa correta.
JUSTIFICATIVA. Como são voltadas para visão computacional, com tarefas de reconhecimento de padrões e de imagens, as operações realizadas pelas redes neurais convolucionais são bastante complexas (por exemplo, tratamento de matrizes e operações algébricas). Por isso, apesar da acurácia resultante, máquinas com poder de processamento maior são necessárias.

III – Afirmativa incorreta.
JUSTIFICATIVA. Na verdade, o aumento do número de neurônios (unidades de processamento) nas camadas aumenta a acurácia. Por sua vez, o tempo de treinamento tende a aumentar.

IV – Afirmativa correta.
JUSTIFICATIVA. Aumentar o número de amostras de treinamento é uma alternativa que pode levar à melhora na acurácia, apesar de aumentar o tempo de treinamento da rede. Mais amostras no treinamento levam a uma representação mais fiel da característica analisada pela rede neural. Apesar de haver um limite para o aumento de amostras em relação à melhora da acurácia, o aumento no número de amostras analisadas tende a aumentar a precisão.

V – Afirmativa incorreta.
JUSTIFICATIVA. As redes neurais recorrentes realimentam as entradas com as próprias saídas. Isso significa que os níveis de ativação da rede formam um sistema dinâmico que pode atingir

um estado estável, apresentar oscilações ou ter um comportamento caótico. Além disso, a resposta da rede para determinada entrada depende do seu estado inicial, que pode depender de entradas anteriores. Portanto, devido à capacidade de memória recorrente de curto prazo que possuem, as redes neurais recorrentes não são recomendadas para aplicações de longo prazo devido à dificuldade de processamento de estados distantes, pois pode haver queda significativa da acurácia.

Alternativa correta: D.

Questão 5

I – Afirmativa correta.
JUSTIFICATIVA. A regressão linear é uma técnica supervisionada de indução que busca explicar ou prever, a partir de algumas variáveis explicativas com comportamento linear, os resultados de determinado evento.

II – Afirmativa incorreta.
JUSTIFICATIVA. Na verdade, a afirmativa não se refere ao aprendizado não supervisionado, cujo cálculo do erro se dá pela diferença entre a saída real e a saída prevista.

III – Afirmativa correta.
JUSTIFICATIVA. Em determinados casos, a velocidade de análise proporcionada pelas máquinas permite que os padrões sejam reconhecidos quando há uma quantidade considerável de dados a serem analisados. Assim, o aprendizado não supervisionado é o método mais recomendado para a identificação de padrões.

IV – Afirmativa correta.
JUSTIFICATIVA. A inserção de novos dados pode melhorar a precisão e tornar as decisões ainda mais confiáveis. Em alguns casos, o aprendizado supervisionado pode se tornar contínuo, observando novas tendências e se adaptando a diferentes cenários e situações.

Alternativa correta: E.

3. Indicações bibliográficas

- BOCHIE, K. et al. Aprendizado profundo em redes desafiadoras: conceitos e aplicações. *In: Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores. Sociedade Brasileira de Computação (SBC)*. Sistemas Distribuídos. SBC. 2020, p. 140-189. Disponível em <https://sol.sbc.org.br/livros/index.php/sbc/catalog/book/50>. Acesso em 10 out. 2023.
- ERTEL, W. *Introduction to Artificial Intelligence*. 2. ed. London: Springer, 2017.
- GÉRON, A. *Mãos à obra: aprendizado de máquina com Scikit-Learn & TensorFlow*. Rio de Janeiro: Alta Books, 2019.
- HAYKIN, S. *Redes neurais: princípios e prática*. 2. ed. Porto Alegre: Bookman, 2001.
- IBM. *Redes Neurais*. Disponível em <https://www.ibm.com/br-pt/cloud/learn/neural-networks>. Acesso em 26 dez. 2023.
- MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *In:* REZENDE, S.O. (Org.). *Sistemas inteligentes: Fundamentos e Aplicações*. São Paulo: Manole, 2003.
- RICH, E. *Artificial Intelligence*. New York: McGraw-Hill, 1983.
- RUSSEL, S. J.; NORVIG, P. *Inteligência Artificial*. 4. ed. Rio de Janeiro: Elsevier, 2022.

Questão 6

Questão 6.[6]

A computação em nuvem (*cloud computing*) pode ser definida como a infraestrutura de comunicação representada por vários servidores web, responsáveis por armazenar dados e aplicações, em que cada parte desta infraestrutura é provida como um serviço e estes são normalmente alocados em centros de dados, utilizando hardware compartilhado para computação e armazenamento. Segundo o Instituto Nacional de Padrões e Tecnologia (NIST), um modelo de Computação em Nuvem deve apresentar 5 características essenciais, 3 modelos de serviço e 4 modelos de implantação. As características essenciais são: *self-service* sob demanda, acesso à rede ampla, *pooling* de recursos, elasticidade rápida e serviço medido. Os modelos de serviços são: Software como um Serviço (SaaS), Plataforma como um Serviço (PaaS) e Infraestrutura como um Serviço (IaaS) e os modelos de implantação são: Nuvem Privada, Nuvem Pública, Nuvem Comunidade e Nuvem Híbrida.

Considerando as informações apresentadas, avalie as afirmativas.

I. No modelo SaaS, o usuário não precisa adquirir ou realizar upgrade de hardware para rodar as aplicações, não administra ou controla a infraestrutura subjacente e as atualizações de software são de responsabilidade do provedor do serviço em nuvem.

II. A elasticidade é a capacidade de aumentar ou diminuir de forma automática o tempo de disponibilidade dos recursos computacionais que foram provisionados contratualmente para cada usuário.

III. A Nuvem Comunidade tem como objetivo gerenciar os recursos computacionais pertencentes a cada uma das organizações participantes de uma comunidade de organizações para compartilhar a infraestrutura de software e hardware entre todos.

IV. No modelo IaaS, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados.

É correto apenas o que se afirma em

A. I e II.
B. I e IV.
C. II e III.
D. I, III e IV.
E. II, III e IV.

[6]Questão 25 – Enade 2021.

1. Introdução teórica

Computação em nuvem

A partir dos anos 1990, houve uma evolução bastante acentuada das tecnologias computacionais. As redes de computadores, sobretudo a Internet e as redes sem fio, tiveram grandes ganhos de desempenho e receberam implementações de segurança, consolidando-se como elementos comuns no cotidiano das pessoas.

O custo do armazenamento (discos rígidos e memórias) passou por quedas sucessivas, fato que facilitou a retenção de grandes volumes de dados a um baixo custo e o uso de servidores com alta capacidade de memória.

O poder de processamento das máquinas (tanto servidores quanto clientes) aumentou significativamente, garantindo poder de processamento de dados a custos muito mais baixos que os praticados anteriormente. Esse aumento, por sua vez, incentivou a virtualização de máquinas, redes e dispositivos tecnológicos nos *datacenters* e empresas, diminuindo custos e gerando facilidades como backups e migrações mais eficientes.

Essa evolução conjunta gerou os conceitos de **nuvem** computacional e de tecnologia oferecida como **serviço** – no contexto de tecnologia computacional são conceitos indissociáveis. A partir de certo momento, os recursos e os elementos de tecnologia se transformaram em **serviços** e passaram a ser oferecidos de forma mais ampla àqueles que deles necessitavam. Vamos agora explorar algumas definições essenciais para a área de computação em nuvem.

A nuvem é um modelo computacional que permite o acesso a inúmeros recursos tecnológicos (serviços), como servidores, armazenamento, bancos de dados, aplicativos, segurança cibernética e transmissão de vídeos, entre outros, pela Internet. Em vez de manter recursos e infraestrutura de tecnologia próprios (*on-premise*), as empresas podem acessar esses serviços a partir de provedores de nuvem, pagando apenas pelo que usam.

De acordo com o NIST, a computação em nuvem é um modelo onipresente, compartilhado, que funciona sob demanda e que permite acesso a recursos configuráveis oferecidos em rede, como servidores, armazenamento, recursos compartilháveis, aplicativos e serviços. Refere-se a um ambiente de tecnologia projetado com a finalidade de provisionamento de recursos de Tecnologia da Informação (TI) dimensionáveis e medidos. É usado como uma arquitetura orientada a serviços que reduz as informações de sobrecarga do usuário final.

A definição do NIST é composta por três elementos: as características essenciais associadas à computação em nuvem, os tipos de modelos de serviços oferecidos e os modelos de implantação associados. Vamos analisar brevemente cada um desses elementos.

As cinco características essenciais associadas ao conceito de nuvem computacional são:

- autoatendimento sob demanda;
- acesso amplo pela rede;
- agrupamento de recursos;
- elasticidade rápida;
- serviço mensurável (ou medido).

A primeira característica, autoatendimento sob demanda, significa que o cliente deve ser capaz de solicitar recursos computacionais de forma independente e autônoma, sem a necessidade de uma interação humana. Por exemplo, se o cliente quiser aumentar a quantidade de espaço de armazenamento, ele deve poder fazer isso por meio de uma interface computacional (por exemplo, uma interface *web*, como um site para gerenciamento de recursos da rede), sem a necessidade de falar com um atendente.

A segunda característica, acesso amplo pela rede, significa que os recursos providos pela nuvem devem ser facilmente acessados por uma rede de computadores convencional, podendo ser acessados por diferentes tipos de equipamentos, desde simples telefones celulares até computadores de elevado desempenho.

A terceira característica, agrupamento de recursos, é a que confere o ganho de escala às nuvens computacionais. Utilizando tecnologias como a virtualização, os provedores de serviços de nuvem computacional podem, por exemplo, agrupar vários clientes em uma mesma máquina física. No entanto, se um cliente requisitar uma quantidade significativa de recursos, é possível fazer o inverso e associar recursos computacionais disponíveis em várias máquinas físicas. O uso de virtualização também permite que máquinas virtuais sejam facilmente movidas de um servidor físico para outro. Isso significa que o cliente pode não saber exatamente a localização física dos recursos utilizados, exceto de uma maneira ampla: por exemplo, o cliente pode especificar a região geográfica do recurso (o país) ou indicar qual *datacenter* deve ser utilizado, mas não será capaz de saber exatamente em qual (ou quais) máquina(s) física(s) os serviços contratados estão sendo executados.

Essa flexibilidade nos leva até a quarta característica, a elasticidade rápida. O uso de técnicas de virtualização e o agrupamento de recursos dão aos provedores de serviços de nuvens computacionais a possibilidade de contratação de recursos computacionais e de rede de forma simples e automática.

Finalmente, a quinta característica, o serviço mensurável, associa uma capacidade de medir os diversos serviços prestados, o que adiciona transparência tanto para o cliente quanto para o fornecedor.

As cinco características previamente listadas são os componentes fundamentais da "infraestrutura da nuvem". Obviamente, essa infraestrutura é composta tanto por componentes físicos, como computadores e rede, quanto por componentes "abstratos", que são os diversos programas que possibilitam a operação dos componentes físicos.

Nesse contexto, existem três modelos de serviços que podem ser fornecidos:

- o software como um serviço;
- a plataforma como um serviço;
- a infraestrutura como um serviço.

No caso do software como um serviço, o que é fornecido é uma aplicação executando em uma infraestrutura de nuvem. O cliente pode acessar a aplicação via um *website* ou um aplicativo de celular, por exemplo. Muitos serviços de e-mail populares costumam estar nessa categoria. Observe que, no caso do software como um serviço, o cliente tipicamente tem visibilidade apenas da aplicação que está sendo contratada, tendo pouco ou nenhum conhecimento sobre a plataforma na qual ela está sendo executada, sejam informações sobre o sistema operacional, sejam informações sobre o tipo de servidor etc. Além disso, na maioria dos casos, o cliente compra um serviço pronto e quer apenas pagar pelo uso desse serviço, sem a necessidade de desenvolver algum aspecto do software contratado.

Contudo, há circunstâncias nas quais o cliente tem interesse em desenvolver uma aplicação, mas quer utilizar uma infraestrutura de nuvem para executá-la. Esse pode ser tanto o caso de uma empresa que desenvolve uma aplicação que será comercializada como "software como um serviço", ou uma empresa que, por algum motivo, desenvolve uma aplicação em particular, mas quer utilizar uma infraestrutura de nuvem. Esses tipos de necessidades deram origem aos outros dois modelos de serviços listados anteriormente: plataforma como um serviço e infraestrutura como um serviço.

No caso da plataforma como um serviço, o fornecedor deve prover diversos componentes de software como linguagens, bibliotecas, ferramentas e outros serviços, ou seja, toda plataforma de desenvolvimento que será utilizada para escrever uma aplicação, seja pelo próprio cliente, seja por um terceiro.

Contudo, se o cliente quiser ainda mais liberdade na escolha do software e de alguns aspectos do hardware, há o modelo de infraestrutura como um serviço. Esse terceiro modelo é o mais flexível de todos, dando ao contratante a liberdade na escolha do sistema operacional,

das ferramentas e dos softwares a serem utilizados para desenvolver e executar uma aplicação. Apenas a infraestrutura básica de nuvem é fornecida para o cliente. Contudo, o hardware físico associado a essa infraestrutura continua sendo gerenciada pelo provedor.

Por fim, esses três modelos de serviços dão origem a quatro modelos de implantação, ou quatro "tipos de nuvens computacionais": nuvens privadas, nuvens comunitárias, nuvens públicas e nuvens híbridas. Esses quatro tipos estão intimamente relacionados ao tipo de uso e ao público-alvo dos serviços prestados pela nuvem.

Uma nuvem privada é o caso mais "fechado" de todos, sendo tipicamente utilizada por uma empresa internamente. Por exemplo, uma empresa pode ter um sistema de controle de estoque que é executado em uma nuvem privada e que deve ser acessado apenas por pessoas autorizadas da organização (ainda que, geograficamente, esses acessos possam ser diversificados, especialmente no caso de uma grande empresa com muitas filiais em diversos locais diferentes, por exemplo). Dessa forma, essa empresa utilizaria uma nuvem privada para prover esse tipo de serviço.

Uma segunda possibilidade, mais aberta do que a anterior, é a nuvem comunitária. Nesse caso, o acesso à nuvem é um pouco mais amplo, já que pode envolver pessoas de mais de uma organização. Por exemplo, um governo pode construir uma nuvem com serviços fornecidos para diversas organizações de pesquisa, como universidades, centros de pesquisa e desenvolvimento e empresas ligadas a determinado setor. O número de pessoas com acesso a essa nuvem pode ser limitado (por exemplo, apenas pesquisadores de determinado setor), mas deve envolver diversas organizações diferentes. Nesse caso, a ideia é que a nuvem esteja focada em uma "comunidade" específica, como pesquisadores da área de sustentabilidade e meio-ambiente.

A terceira possibilidade, nuvem pública, ocorre quando a infraestrutura da nuvem está aberta ao público em geral. Esse caso pode envolver diversas situações, já que esse tipo de nuvem pode ser operado por diversos tipos de organizações, como instituições acadêmicas, órgãos públicos ou empresas privadas. O tipo de organização depende do uso desejado da nuvem (por exemplo, se a nuvem vai ser utilizada com finalidades comerciais ou se vai ser utilizada com algum outro tipo de fim não lucrativo).

Finalmente, existe a possibilidade das nuvens híbridas. Nesses casos, ocorre uma composição de infraestruturas de nuvem, ou seja, temos uma nuvem que pode ser composta da interconexão de duas ou mais infraestruturas diferentes. Isso pode ser feito caso exista a possibilidade de portabilidade de dados e de aplicações, permitindo que mais de um modelo de infraestrutura de nuvem seja utilizado.

2. Análise das afirmativas

I - Afirmativa correta.
JUSTIFICATIVA. No modelo SaaS, o provedor fornece a aplicação, que executa sobre uma infraestrutura de serviços em nuvem. Por ter foco na aplicação fornecida, no modelo SaaS, o cliente (ou usuário) não precisa se preocupar em gerenciar ou controlar essa infraestrutura. Além disso, o provedor também deve fornecer as atualizações de software e upgrades de hardware necessários para o funcionamento da infraestrutura.

II - Afirmativa incorreta.
JUSTIFICATIVA. A elasticidade envolve a capacidade de alterar as capacidades fornecidas de forma rápida e dinâmica, tanto no sentido de ampliação quanto no sentido de redução. Isso envolve diversos elementos da infraestrutura de nuvem, não se limitando apenas a um desses aspectos.

III - Afirmativa incorreta.
JUSTIFICATIVA. A ideia de "nuvem comunidade" (ou nuvem comunitária) é de que ela forneça seus recursos para usuários de diversas organizações, ao contrário da nuvem privada, que é muito mais fechada. Essa nuvem pode ser gerenciada por uma das organizações-membro, ou por uma outra organização terceira especializada. Os recursos de software e hardware necessários ao seu funcionamento são normalmente gerenciados pelo fornecedor da nuvem. Contudo, não devemos afirmar que o objetivo desse tipo de nuvem é gerenciar todos os recursos de software ou hardware dos participantes. Os recursos gerenciados pelo fornecedor da nuvem são apenas aqueles ligados ao funcionamento da nuvem comunitária em si, e não a outros serviços de software ou hardware que não fazem parte dessa nuvem.

IV - Afirmativa correta.
JUSTIFICATIVA. No modelo IaaS, o provedor fornece os recursos computacionais fundamentais para serem gerenciados de acordo com a necessidade do cliente, como capacidade de processamento, armazenamento e rede. Entretanto, a infraestrutura subjacente não é gerenciada pelo cliente.

Alternativa correta: B.

3. Indicações bibliográficas

- BUYYA, R.; VECCHIOLA, C.; SELVI, S. T. *Mastering cloud computing: foundations and applications programming*. Waltham: Morgan Kaufmann, 2013.
- KOLBE JR., A. *Computação em nuvem*. Curitiba: Contentus, 2020.
- LORENZI, U. M.; GREIN, W. B.; CORCINI, L. F. *Computação em nuvem: conceitos, aplicações e novas tecnologias*. Faculdades Santa Cruz, [s. l], v. 13, n. 1, p. 1-20, 01 jan. 2022. Disponível em <http://unisantacruz.edu.br/revistas/index.php/revusc/article/view/8/8>. Acesso em 3 mar. 2024.
- MARTINEZ, M. *Computação em nuvem*. InfoEscola, 2023. Disponível em <https://www.infoescola.com/informatica/computacao-em-nuvem/>. Acesso em 13 mar. 2023.
- MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg: National Institute of Standards and Technology, 2011. Disponível em <https://doi.org/10.6028/NIST.SP.800-145>. Acesso em 26 dez. 2023.
- MICROSOFT. *O que é o SaaS? Software como serviço*. Disponível em <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-saas/>. Acesso em 25 set. 2023.
- SANTOS, T. *Fundamentos da computação em nuvem*. São Paulo: Senac, 2018.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 5. ed. São Paulo: Pearson Brasil, 2024.

Questão 7

Questão 7.[7]

O uso da estrutura de dados tipo Árvore Binária de Busca é uma técnica fundamental de programação. Uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos, a saber: 1) raiz da árvore - elemento inicial (único), 2) subárvore da esquerda - se vista isoladamente, compõe outra árvore e 3) subárvore da direita - se vista isoladamente, compõe outra árvore. A árvore pode não ter qualquer elemento (árvore vazia). A definição de árvore é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. Sendo "A" a raiz de uma árvore binária e "B" a raiz de sua subárvore esquerda ou direita, é dito que "A" é pai de "B" e que "B" é filho de "A". Um elemento sem filhos é chamado de folha. A altura da árvore é o número de elementos encontrados no caminho descendente mais longo que liga a sua raiz até uma folha. Uma Árvore de Busca Binária é uma árvore binária especializada, na qual a informação que o elemento filho esquerdo possui é numericamente menor do que a informação do elemento pai. De forma análoga, a informação que o elemento filho direito possui é numericamente maior ou igual à informação do elemento pai. O objetivo de organizar dados em Árvores Binárias de Busca é facilitar a tarefa de encontrar determinado elemento. O percurso completo de uma árvore binária consiste em visitar todos os elementos dessa árvore, segundo algum critério, a fim de processá-los. Três formas são bem conhecidas para a realização desse percurso: 1) pré-ordem, 2) em-ordem e 3) pós-ordem. A figura a seguir mostra um exemplo de árvore binária.

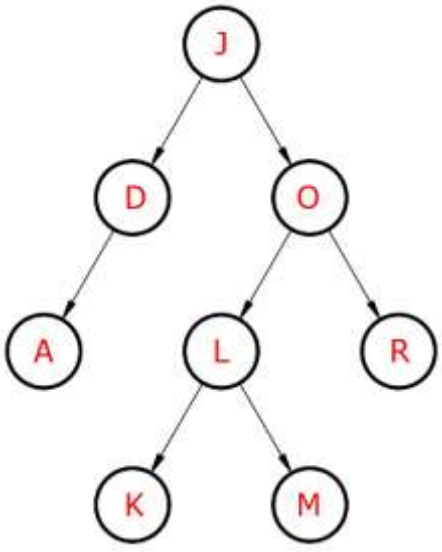


Figura. Exemplo de árvore binária.
LAUREANO, M. A. P. *Estrutura de Dados com Algoritmos*. São Paulo: Brasport, 2008. p. 126, 129, 136 (com adaptações).

[7]Questão 23 – Enade 2021.

Considerando o texto e a figura apresentados e que a seguinte lista de elementos numéricos: (27, 34, 40, 18, 23, 5, 25, 36, 10, 7, -2) seja totalmente transferida para uma estrutura de Árvore Binária de Busca, inicialmente vazia, elemento a elemento, da esquerda para a direita, assinale a alternativa correta.

A. A árvore resultante terá 5 níveis de altura, com 6 elementos à esquerda da raiz principal (inicial) e 4 elementos à direita.
B. O percurso da árvore em Pré-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 7, 10, 5, 25, 23, 18, 36, 40, 34, 27.
C. O percurso da árvore em Em-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 5, 7, 10, 18, 23, 25, 27, 34, 36, 40.
D. O percurso da árvore em Pós-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36.
E. O número máximo de elementos que essa árvore poderá ter com 10 níveis será de 1024 elementos.

1. Introdução teórica

1.1. Estruturas de dados

As estruturas de dados representam maneiras de organizar, distribuir e estabelecer relações entre os dados disponíveis, com o objetivo de otimizar o desempenho dos algoritmos que operam sobre esses dados. Na programação, há uma diversidade considerável de estruturas de dados disponíveis, como listas (incluindo filas, pilhas, deques e listas circulares), árvores (como as binárias, as binárias de busca e as não binárias), grafos e tabelas *hash*.

À medida que enfrentamos desafios relacionados a conjuntos volumosos de dados, surgem cada vez mais aplicações para lidar com esses cenários. Ao projetar um sistema, é essencial selecionar recursos de programação que possam manipular os dados de forma eficiente. Embora algumas linguagens de programação ofereçam bibliotecas e *frameworks* dedicados ao tratamento de dados, compreender a lógica interna de cada implementação é fundamental para tomarmos decisões corretas sobre qual abordagem é mais adequada para o desenvolvimento do programa em questão.

Os métodos de construção e de conexão das estruturas de dados são geralmente similares. A distinção reside na maneira como cada estrutura é manipulada, assim como nas operações específicas realizadas em cada uma delas.

1.2. Árvores binárias

Uma árvore binária é uma estrutura de dados hierárquica composta por nós, e cada nó tem, no máximo, dois filhos: um filho à esquerda e um filho à direita. Cada filho de um nó é também a raiz de uma subárvore binária. No caso das árvores binárias de busca, os dados contidos nos diversos nós estão organizados de forma que o valor do nó filho esquerdo seja menor que o valor do nó pai, enquanto os valores contidos no nó filho direito são maiores do que ou iguais ao valor do nó pai.

As árvores binárias podem ser classificadas nos tipos a seguir.

- Árvores binárias de busca - *Binary Search Trees* (BST). Em uma BST, a propriedade de ordenação é mantida, de forma que, para cada nó pai, todos os nós filhos na subárvore à esquerda têm valores menores, e todos os nós filhos na subárvore à direita têm valores maiores. Isso facilita a busca eficiente de elementos na árvore.
- Árvores AVL (Adelson, Velsky e Landis). Uma árvore AVL é uma árvore binária de busca equilibrada, em que a altura das subárvores à esquerda e à direita de cada nó difere de, no máximo, uma unidade. Esse equilíbrio garante tempos de busca, inserção e exclusão mais eficientes.
- Árvores rubro-negras. Similares às árvores AVL, as árvores rubro-negras também são árvores de busca balanceadas. Elas utilizam uma técnica de coloração de nós para garantir o balanceamento da árvore, proporcionando desempenho eficiente nas operações.
- Árvores *Splay*. As árvores *Splay* são projetadas para otimizar o acesso aos elementos frequentemente acessados, movendo o nó acessado recentemente para a raiz da árvore. Isso pode melhorar o desempenho em cenários nos quais certos elementos são mais propensos a ser acessados repetidamente.
- Árvores de Huffman. Utilizadas em compressão de dados, as árvores de Huffman são árvores binárias usadas para criar códigos de compressão eficientes, atribuindo códigos de comprimento variável a diferentes símbolos com base em sua frequência de ocorrência.

Cada tipo de árvore binária tem características distintas, sendo escolhido de acordo com os requisitos específicos de aplicação e com as operações que precisam ser realizadas com eficiência.

1.3. Manipulação de árvores binárias

As estruturas de dados do tipo árvore são não lineares, o que significa que os elementos não estão armazenados de forma sequencial, sendo que nem todos estão encadeados. Assim, o procedimento de inserção em árvores binárias inicia com uma busca pelo valor a ser inserido na árvore. Caso o elemento não exista, é alcançado um nó folha e, então, o valor é inserido nessa posição.

A cada inserção, a raiz é examinada e é introduzido um novo nó na subárvore da esquerda; caso o valor novo seja menor do que a raiz, ou à direita, caso o valor novo seja maior ou igual à raiz.

1.4. Percorrendo árvores binárias

Uma vez que uma árvore binária tenha sido montada, ela pode ser utilizada em um programa para o acesso das informações nela contidas. Uma das operações mais comuns em programas que utilizam árvores binárias é chamada de percurso. Para explicar esse conceito, considere o exemplo de árvore binária na figura 1.

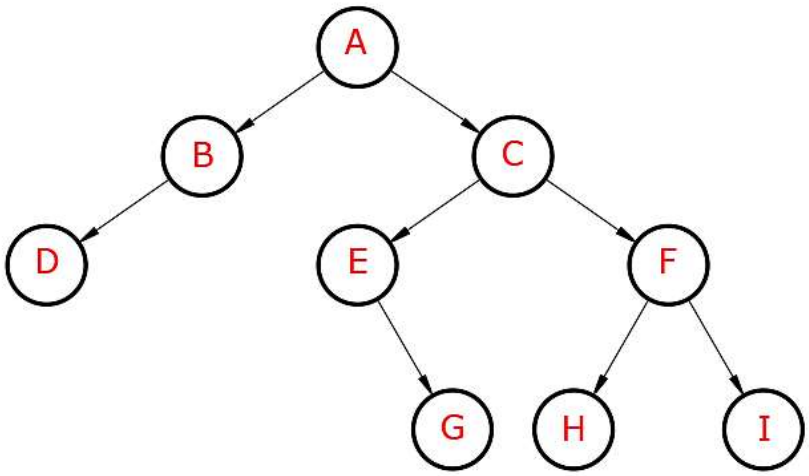


Figura 1. Exemplo de árvore binária.

Percorrer uma árvore binária consiste em visitar todos os nós existentes a partir de um critério. Há três modos: pré-ordem, pós-ordem e em-ordem. Vamos analisar brevemente cada um desses modos.

1. Percurso em pré-ordem: visita-se a raiz, percorre-se a subárvore esquerda em pré-ordem e, depois, percorre-se a subárvore da direita (também em pré-ordem). O percurso ocorre visitando os nós filhos da esquerda para a direita, indo para níveis cada

vez mais profundos da árvore. Para a árvore listada na figura 1, o percurso em pré-ordem resulta em A, B, D, C, E, G, F, H, I.

2. Percurso em pós-ordem: em primeiro lugar, percorre-se a subárvore esquerda em pós-ordem, depois percorre-se a subárvore direita em pós-ordem e, por último, visita-se o nó pai. Observe que, nesse percurso, os filhos são processados antes do nó raiz (nó pai). Para a árvore listada na figura 1, o percurso em pós-ordem resulta em D, B, G, E, H, I, F, C, A.

3. Percurso em-ordem: em primeiro lugar, percorre-se a subárvore esquerda em-ordem, posteriormente se visita o nó pai e, finalmente, percorre-se a subárvore direita em-ordem. Desse modo, o percurso se dá de forma que inicialmente se processa o filho à esquerda, chega-se até a raiz e, por fim, percorre-se o filho à direita, não levando em consideração os nós pais que já foram percorridos. Para a árvore listada na figura 1, o percurso em-ordem resulta em D, B, A, E, G, C, H, F e I.

2. Solução da questão e análise das alternativas

Para resolver a questão, devemos levar em consideração três aspectos, citados no enunciado:

- os filhos do lado esquerdo são sempre menores do que o nó pai, enquanto os filhos do lado direito são maiores do que o nó pai;
- os elementos são inseridos da esquerda para a direita;
- a árvore está inicialmente vazia.

Dessa forma, a inserção da lista de números (27, 34, 40, 18, 23, 5, 25, 36, 10, 7, -2) será feita nos passos descritos a seguir.

O primeiro número a ser inserido (27) é o nó raiz, como ilustrado na figura 2.

Figura 2. Inserção do primeiro elemento na árvore binária.

O segundo elemento a ser inserido (34) é maior do que a raiz (27). Portanto, deve ser inserido como o nó filho direito do primeiro nó, como ilustrado na figura 3.

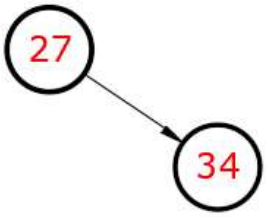


Figura 3. Inserção do segundo elemento na árvore binária.

O terceiro elemento a ser inserido (40) é maior do que o nó raiz (27), mas, como esse nó já tem um nó filho do lado direito, é necessário descermos um nível na árvore, para fazermos a inserção considerando o nó seguinte (34). Como 40 também é maior do que 34, esse nó vai ser inserido como o filho do lado direito do nó 34, e obtemos a árvore da figura 4.

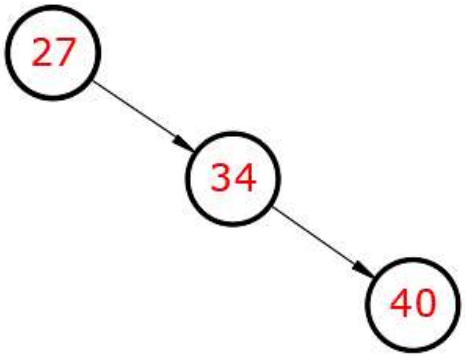


Figura 4. Inserção do terceiro elemento na árvore binária.

Agora, devemos inserir o número 18, que é menor do que o nó raiz. Como o nó raiz não tem, no momento, nenhum nó filho do lado esquerdo, devemos inserir o nó com o número 18, como sendo o seu nó filho esquerdo, e obtemos a árvore da figura 5.

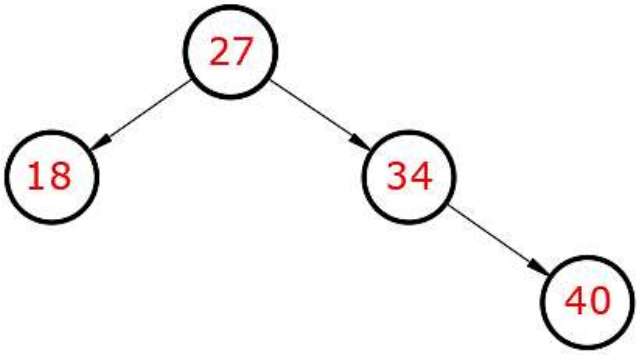


Figura 5. Inserção do quarto elemento na árvore binária.

O quinto elemento a ser inserido (23) é menor do que o nó raiz, mas, como o nó raiz já tem nós filhos de ambos os lados e estamos construindo uma árvore binária, devemos tentar a fazer a inserção abaixo do filho esquerdo do nó raiz, no caso, o número 18. Como 23 é maior do que 18, esse nó vai ser inserido como o nó filho direito do nó 18. A árvore resultado após a inserção é mostrada na figura 6.

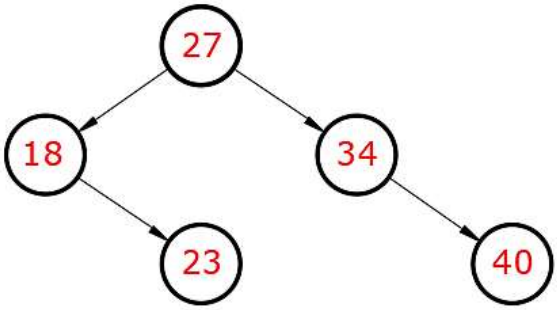


Figura 6. Inserção do quinto elemento na árvore binária.

O sexto elemento a ser inserido é o número 5. Devemos seguir um procedimento similar aos anteriores, verificando que 5 é menor do que 27 e, posteriormente, que 5 é menor do que 18. Logo, o nó com o número 5 deve ser inserido como o filho esquerdo do nó com o número 18, e obtemos a árvore da figura 7.

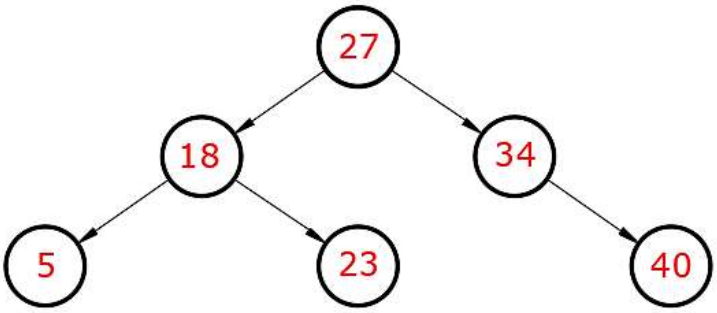


Figura 7. Inserção do sexto elemento na árvore binária.

Agora, devemos inserir o sétimo elemento, que é o número 25. Esse número é menor do que o nó raiz (27), o que nos leva a percorrer o lado esquerdo da árvore, em direção ao nó com o número 18. Como 25 é maior do que 18, devemos agora ir para o lado direito da subárvore, encontrando o nó com o número 23. Como 25 é maior do que 23 e, nesse momento, o nó com o número 23 não tem nenhum nó filho, devemos fazer a inserção do nó com o número 25 como sendo o nó filho direito do nó com o número 23. Obtemos, assim, a árvore ilustrada na figura 8.

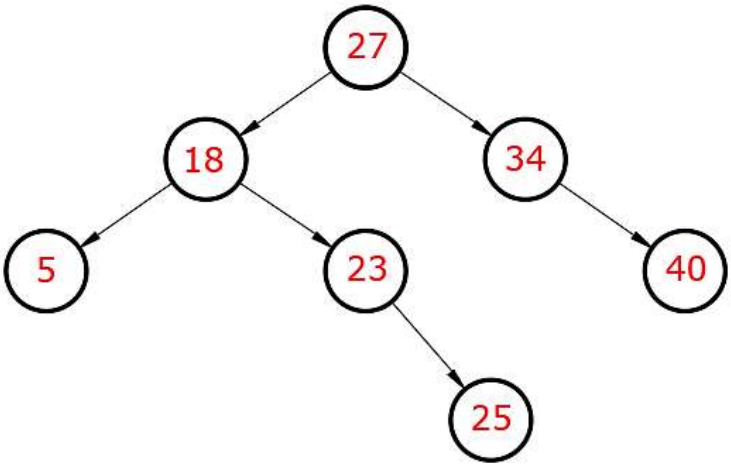


Figura 8. Inserção do sétimo elemento na árvore binária.

Para inserirmos o nó com o número 36, primeiramente devemos observar que 36 é maior do que 27. Logo, devemos percorrer o lado direito da árvore. Nesse lado, temos o nó com o número 34, que é menor do que 36. Percorremos o lado direito da subárvore, encontrando o nó com o número 40. Esse nó não tem nós filhos e, como 36 é menor do que 40, devemos inserir o nó com o número 36 como o nó filho esquerdo do nó com o número 40, e obtemos a árvore ilustrada na figura 9.

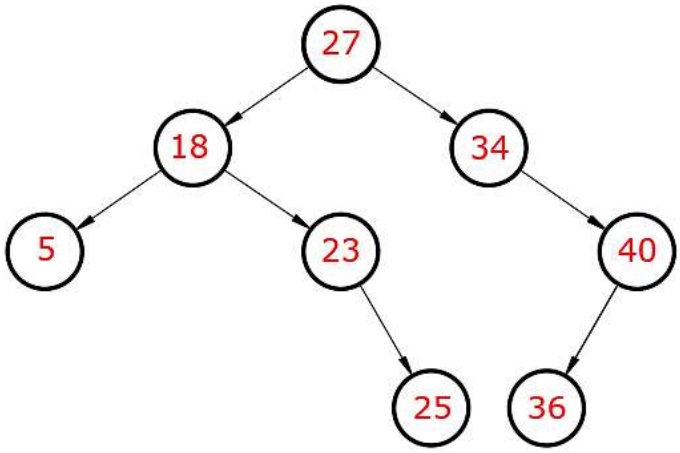


Figura 9. Inserção do oitavo elemento na árvore binária.

O nono elemento a ser inserido é o nó com o número 10. Observe que 10 é menor do que 27 e menor do que 18, mas maior do que 5. Logo, devemos inserir o nó com o número 10 como sendo o nó filho direito do nó com o número 5. Obtemos, dessa forma, a árvore ilustrada na figura 10.

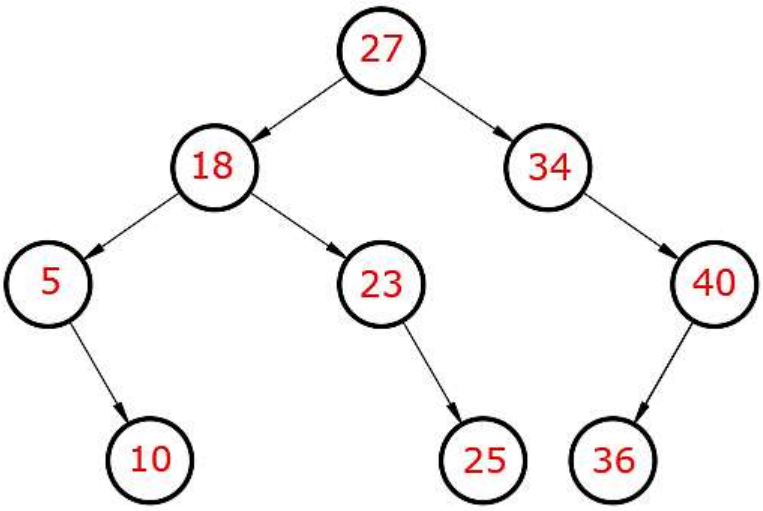


Figura 10. Inserção do nono elemento na árvore binária.

O décimo elemento a ser inserido é o número 7. Observe que ele é menor do que 27, menor do que 18, maior do que 5 e menor do que 10. Logo, devemos inserir esse nó como o nó filho esquerdo do nó com o número 10. Dessa forma, a árvore da figura 11.

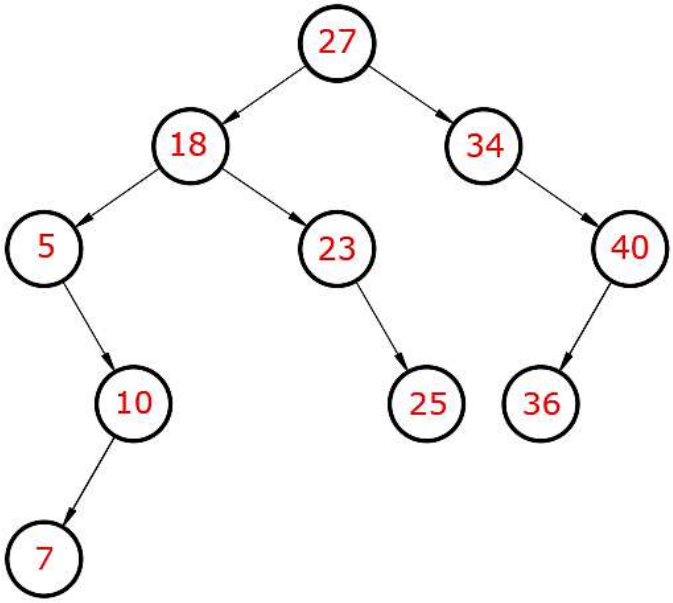


Figura 11. Inserção do décimo elemento na árvore binária.

Finalmente, devemos inserir o número -2. Ele é menor do que o número do nó raiz (27), menor do que o número do filho esquerdo do nó raiz (18) e menor do que o número do filho esquerdo deste último nó (5), que não possui nenhum nó filho do lado esquerdo. Logo, o número -2 deve ser inserido como o nó filho do lado esquerdo do nó com o número 5. Como esse é também o último elemento da lista de números a serem inseridos, chegamos à árvore binária final mostrada na figura 12.

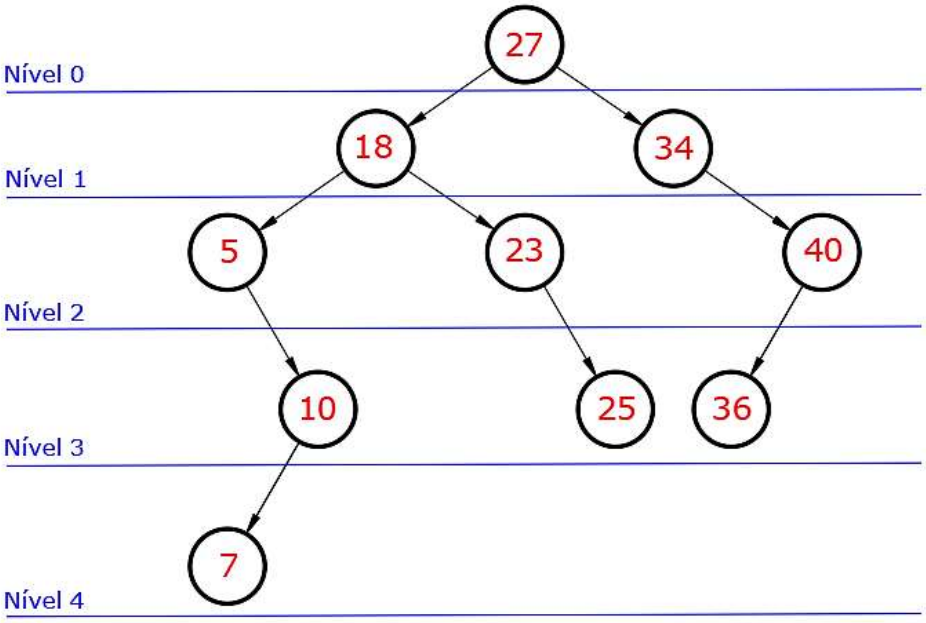


Figura 12. Árvore binária após a inserção do décimo primeiro elemento.

Neste ponto, é interessante fazermos algumas observações teóricas sobre o assunto. A árvore obtida (figura 12), ainda que contenha todos os elementos da lista de números do enunciado, não é considerada uma árvore binária completa. Para entendermos o motivo, devemos nos lembrar de alguns conceitos importantes da teoria de estruturas de dados.

Primeiramente, lembremos que o grau de um nó em uma árvore corresponde ao número de nós filhos desse nó. Observe também que o nó pai não conta no cálculo do grau. Por exemplo, na figura 12, o nó com o número 7 tem grau zero, já que não tem nenhum nó filho. O nó com o número 23 tem grau 1, já que tem apenas um único nó filho.

Outros dois conceitos fundamentais na área são o de altura e o de profundidade. A profundidade de determinado nó é o comprimento do caminho desde o nó raiz até esse nó. Na figura 12, o nó com o número 5 apresenta profundidade (ou nível) 2. Já a altura de um nó qualquer é definida como o número de arestas no caminho descendente simples mais longo desde o nó até uma folha. Com base nessa definição de altura para um nó genérico, podemos definir a altura de uma árvore como a altura do seu nó raiz.

Se observarmos os nós na parte mais inferior da figura 12, notamos que eles não têm filhos. Nós que não têm filhos são chamados de nós folhas. Adicionalmente, um nó que não é um nó folha é chamado de nó interno.

Finalmente, podemos definir uma árvore binária completa como uma árvore na qual todos os nós folhas têm a mesma profundidade e todos os nós internos têm grau dois. É por isso que não podemos dizer que a árvore obtida na figura 12 é uma árvore binária completa:

observe que alguns dos seus nós folhas estão no nível três, mas um dos nós folhas (com o número sete) está no nível 4. Logo, essa árvore binária não pode ser considerada completa.

Não devemos confundir o conceito de árvore binária completa com o conceito de árvore binária cheia. Para uma árvore binária ser considerada cheia, basta que cada nó seja considerado um nó folha ou tenha grau 2. Contudo, em uma árvore binária cheia, os nós folhas não precisam estar todos no mesmo nível, como é o caso da árvore binária completa.

Podemos agora considerar a situação na qual queremos calcular o número máximo de nós de uma árvore binária. Vamos analisar cada um dos níveis dessa árvore e verificar quantos nós conseguimos colocar para cada um dos níveis. Considere, a título de exemplo, a árvore binária mostrada na figura 13. Note que essa é uma árvore binária hipotética, com o máximo número de nós em cada um dos níveis.

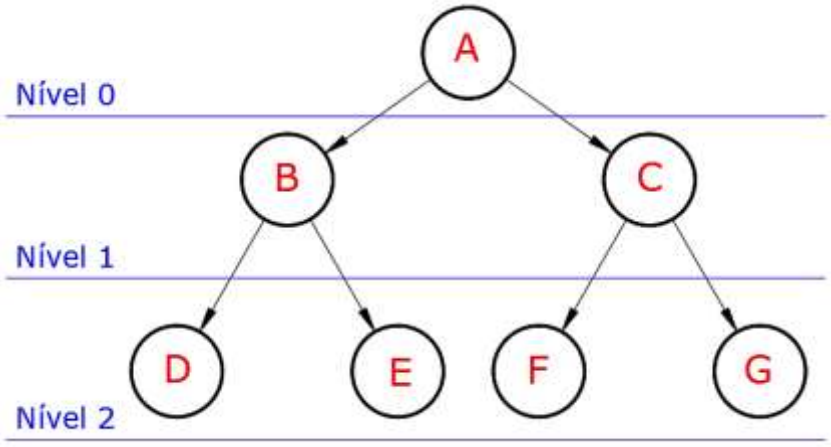


Figura 13. Árvore binária com o máximo número de elementos em cada nível.

- No nível 0, o número máximo de elementos é $2^0=1$ (apenas o nó A).
- No nível 1, o número máximo de elementos é $2^1=2$ (nós B e C).
- No nível 2, o número máximo de elementos é $2^2=4$ (nós D, E, F e G).

Observe que, para cada nível n, o número máximo de nós é dado por 2^n. Por exemplo, no nível 2, temos um total de elementos igual a $2^2=4$. Note que devemos tomar um pouco de cuidado com a nomenclatura: chamamos o primeiro nível de "nível zero" (para facilitar o cálculo do número de elementos por nível).

Para calcularmos o número total de nós na árvore binária, é necessário somar o total de nós de todos os níveis. Para o exemplo da figura 13, temos:

$$2^2+2^1+2^0=4+2+1=7$$

Também podemos calcular o número de elementos de uma árvore binária perfeita (com o máximo de elementos possíveis) sabendo a altura da árvore (normalmente indicada pela

letra h). Lembremos que a altura de uma árvore é definida como a altura do seu nó raiz, que, por sua vez, é definida como a distância entre o nó raiz e o seu descendente mais afastado. Para uma árvore binária perfeita de altura h, o número total de nós é dado por:

$$2^{h+1} -1$$

Para a árvore da figura 13, que tem altura *h=2*, podemos calcular:

$$2^{h+1} -1=2^{2+1} -1=2^3 -1=8-1=7$$

A – Alternativa incorreta.
JUSTIFICATIVA. Observe que, na árvore resultante (figura 12), há 7 elementos à esquerda da raiz principal (18, 5, 23, -2, 10, 25 e 7), e não 6, como diz a afirmativa. E, à direita da raiz principal, há apenas 3 elementos (34, 40 e 36), e não 4.

B – Alternativa incorreta.
JUSTIFICATIVA. O percurso correto em pré-ordem é: 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36.

C – Alternativa correta.
JUSTIFICATIVA. Iniciando pelos últimos nós mais à esquerda, o percurso em ordem visita a raiz entre a visita do filho esquerdo e do filho direito. O resultado é: -2, 5, 7, 10, 18, 23, 25, 27, 34, 36, 40.

D – Alternativa incorreta.
JUSTIFICATIVA. O percurso correto em pós-ordem é: -2, 7, 10, 5, 25, 23, 18, 36, 40, 34 e 27.

E – Alternativa incorreta.
JUSTIFICATIVA. Primeiramente, é importante ter em mente que a afirmação é sobre o número máximo de elementos de uma árvore binária considerando todos os seus níveis, e não sobre o número máximo de elementos de apenas em um único nível. Dessa forma, o número máximo de elementos de uma árvore binária perfeita (e que não esteja vazia) nunca poderá ser um número par: lembre-se de que o nível zero tem apenas um único nó, e todos os níveis restantes terão um total de nós que será uma potência de 2 (portanto, um número par, já que estamos nos referindo a um número do tipo 2^k, com k maior ou igual a 1). Dessa forma, o número total de nós será a soma de um número par mais um, o que é, obrigatoriamente, um número ímpar. Como 1024 é um número par, ele não pode ser igual ao número máximo

de nós de uma árvore com 10 níveis. Contudo, o número máximo de nós do nível 10 de uma árvore binária, especificamente, é 2^{10}=1024, mas esse é o total de apenas um único nível, e não de toda a árvore.

3. Indicações bibliográficas

- ASCENCIO, A. F. G.; de ARAÚJO, G. S. *Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++*. São Paulo: Pearson Prentice Hall, 2010.
- CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 4. ed. Rio de Janeiro: GEN, 2024.
- FEOFILOFF, P. *Árvores binárias*. 2018. Disponível em <https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html>. Acesso em 23 jan. 2024.
- KNUTH, D. E. *The art of computer programming: fundamental algorithms*. 3. ed. v. 1. Boston: Addison-Wesley, 1997.
- LINTZMAYER, C. N.; MOTA, G. O. *Análise de algoritmos e estruturas de dados*. Disponível em <https://www.ime.usp.br/~mota/livros/livro_AAED.pdf>. Acesso em 19 set. 2022.
- VETORAZZO, A. S. et al. *Estrutura de dados*. Porto Alegre: Grupo A, 2018.

Questão 8

Questão 8.[8]

Leia o texto a seguir.

A criptografia de ponta a ponta do WhatsApp garante que somente você e a pessoa com quem você está se comunicando possam ler o que é enviado. Ninguém mais terá acesso a elas, nem mesmo o WhatsApp. As suas mensagens estão seguras com cadeados e somente você e a pessoa que as recebe possuem as chaves especiais necessárias para abri-los e ler as mensagens. E, para uma proteção ainda maior, cada mensagem que você envia tem um cadeado e uma chave únicos.

Disponível em <https://faq.whatsapp.com/pt_br/general/28030015>. Acesso em 05 mai. 2020.

Com base no texto acima e considerando os conceitos de segurança e criptografia, avalie as afirmativas.

I. Se um par de chaves é gerado durante a instalação do aplicativo e a chave pública do usuário é armazenada no servidor, é possível verificar a autenticidade de uma mensagem recebida usando a chave pública do remetente obtida do servidor.

II. A estratégia de utilizar um vetor de inicialização (IV) variável para compor chaves criptográficas diferentes para cada mensagem enviada oculta padrões de dados, além de dificultar os chamados ataques de reprodução.

III. O uso do algoritmo AES nas comunicações entre dois usuários indica o emprego de criptografia simétrica, isto é, aquela que utiliza um par de chaves, uma usada pelo remetente, para encriptar a mensagem, e outra para o destinatário decriptá-la.

IV. A presença do algoritmo SHA-256, no protocolo de comunicação entre cliente e servidor, sugere a verificação de integridade das mensagens, visto que é possível detectar se ocorreu alguma modificação comparando-se os valores de hash da mensagem enviada e recebida.

É correto apenas o que se afirma em

A. I e IV.

B. II e III.

C. III e IV.

D. I, II e III.

E. I, II e IV.

[8]Questão 24 – 2021.

1. Introdução teórica

1.1. Criptografia

A criptografia é um processo matemático para a conversão de uma mensagem aberta em uma mensagem cifrada. Alguns exemplos são os processos de permutação (troca de posição dos caracteres de um texto) e de substituição (troca dos caracteres originais por outros) que tornam a mensagem ininteligível para um interceptor que não possua os métodos e, principalmente, as chaves utilizadas na criptografia.

Em geral, com a criptografia, é possível obter confidencialidade e/ou autenticação. A confidencialidade garante que ninguém, exceto quem possui a chave, poderá ler uma mensagem cifrada. A autenticação garante a procedência de uma mensagem, atestando a identidade de um emissor ou de um destinatário, de um parceiro de negócios, ou mesmo a autoria de uma ação.

A criptografia (cifração ou encriptação) é composta por quatro elementos fundamentais: (1) um texto não cifrado, chamado de mensagem aberta; (2) um algoritmo criptográfico; (3) uma chave; e (4) um texto cifrado. O processo inverso é a decifração (descriptografia ou desencriptação) e utiliza os mesmos elementos (figura 1).

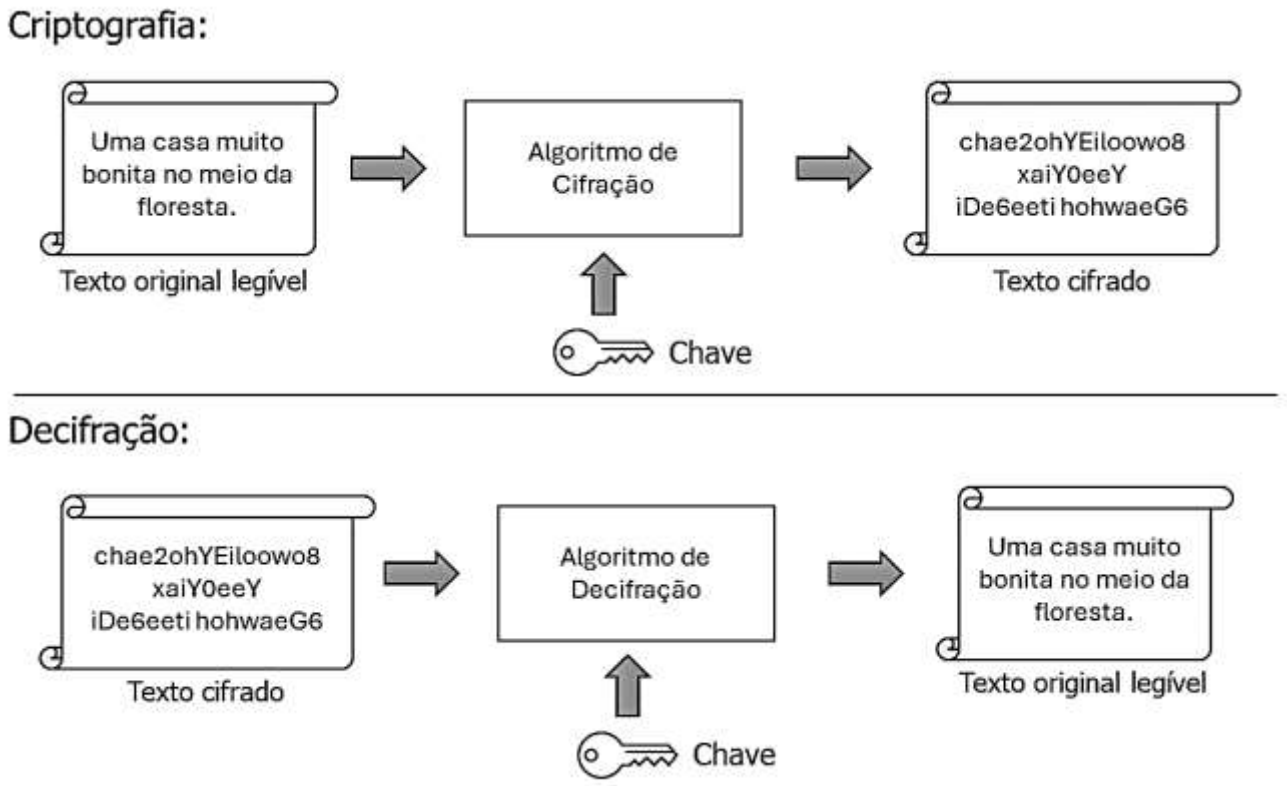


Figura 1. Elementos da criptografia.

A mensagem aberta é o texto que vai passar por criptografia. É a partir desse arquivo que será gerada a mensagem cifrada (criptografada). É importante ressaltar isso porque, na maioria dos casos, a mensagem aberta é mantida, não sendo substituída pela mensagem cifrada.

O algoritmo é uma função matemática responsável pela permutação e pela substituição do conteúdo da mensagem aberta. Em geral, algoritmos de código aberto são mais confiáveis, já que o entendimento da matemática do processo não deveria ser suficiente para se quebrar a criptografia de uma mensagem. Em relação ao processamento, existem algoritmos de bloco, que quebram a mensagem aberta em pedaços de tamanho pré-definido e cifram separadamente cada um desses blocos. Também há algoritmos de fluxo (*stream*), que agem diretamente nos bits dos arquivos a serem encriptados pela aplicação de processos de XOR entre a mensagem aberta e a chave. A criptografia por *stream* costuma ser mais rápida, mas a maioria dos algoritmos usa a cifragem por bloco, devido à flexibilidade que o método oferece.

Um exemplo de algoritmo de criptografia por bloco é o *Advanced Encryption Standard* (AES), ou, em português, Padrão de Criptografia Avançada. Esse algoritmo de criptografia simétrica é considerado padrão e amplamente adotado para proteger dados sensíveis. O AES opera em blocos de dados e suporta chaves de 128, 192 ou 256 bits. Sua segurança é baseada na dificuldade computacional de realizar ataques de força bruta para decifrar a mensagem sem a chave correspondente. O AES é considerado altamente seguro e é utilizado em uma variedade de aplicações, desde comunicações seguras na internet até criptografia de arquivos e dispositivos.

A chave, também conhecida como senha, é o elemento central de segurança da criptografia. Ou seja, a segurança de todo o processo criptográfico está na ocultação da senha. Na medida em que a senha se mantiver oculta, maiores serão as chances de todo o processo ser bem-sucedido em relação à confidencialidade e à autenticação.

É importante pensar na criptografia tendo por base a comunicação. Mensagens, arquivos, textos ou dados encriptados são trocados constantemente entre as pessoas e, nesse contexto, o uso de processos criptográficos é muito importante. Em relação à troca de chaves, existem a criptografia simétrica e a criptografia de chave pública/privada (assimétrica).

Na criptografia simétrica (convencional), apenas uma chave é utilizada. A mesma senha usada na encriptação do arquivo é utilizada na decifração do texto cifrado. Questões de desempenho são importantes na criptografia. Mensagens maiores são computacionalmente mais custosas para serem cifradas do que mensagens menores. Apesar de garantir a confidencialidade e de ser mais rápido do que a criptografia assimétrica, o método simétrico

não garante a autenticação - já que é necessário que uma chave esteja em mais de um lugar se dois parceiros de comunicação precisarem trocar mensagens encriptadas. A criptografia simétrica também pode representar problemas quando desejamos transferir a chave, já que é difícil encontrar formas de se passar uma senha de forma totalmente segura para um destinatário.

A criptografia de chave pública/privada utiliza duas chaves no processo criptográfico, uma pública, que deve ser divulgada, e uma privada, que deve ser mantida em segredo. Ambas são geradas em conjunto e de forma matemática. Tudo o que for encriptado com a chave pública só pode ser decifrado com a respectiva chave privada. Do mesmo modo, tudo o que for cifrado com a chave privada, apenas a respectiva chave pública é capaz de decifrar. A grande vantagem do esquema de chave pública/privada é que, além da confidencialidade, é possível ter a autenticação em uma troca de mensagens.

Para obter a confidencialidade, o remetente cifra a mensagem utilizando a chave pública do destinatário. Ao receber o arquivo cifrado, o destinatário decifra a mensagem utilizando a própria chave privada. Como somente o destinatário conhece a própria chave privada, consegue-se obter o nível de sigilo desejado. Para a obtenção da autenticação, o remetente cifra a mensagem com a própria chave privada. Sabemos que a decifração da mensagem pode ser realizada por qualquer outra entidade utilizando a chave pública do remetente, por isso esse processo prova apenas que o remetente poderia ter realizado a encriptação da mensagem aberta original, visto que apenas ele possui a chave privada necessária para isso.

Processos que usam confidencialidade em conjunto com autenticação também são possíveis com a junção dos dois métodos. Entretanto, a criptografia assimétrica é mais lenta do que a simétrica. Como nos processos computacionais o desempenho é um item fundamental, a adoção irrestrita da criptografia de chave pública/privada para a troca de dados torna-se inviável e, por isso, ela é utilizada apenas na troca de chaves de sessão. Uma chave de sessão é uma chave simétrica aleatória temporária utilizada durante curto período. O remetente envia uma chave de sessão com confidencialidade e autenticação a um destinatário. Essa chave de sessão é então utilizada como senha simétrica para a troca de dados na comunicação e depois apagada. Assim, a troca de chaves também fica resolvida ao mesmo tempo em que o desempenho continua garantido.

1.2. Obtendo integridade

A obtenção de integridade das mensagens pode ser feita com o uso da técnica de *hash* criptográfico (também chamada simplesmente de *hashes*).

Um *hash* criptográfico é uma *string* (cadeia de texto) de tamanho fixo resultante de uma função irreversível aplicada em um texto. Essa técnica tem por base dois fatos: um *hash* se mantém igual quando aplicado a um texto de mesmo conteúdo e muda quando o conteúdo do texto de entrada é alterado (figura 2).

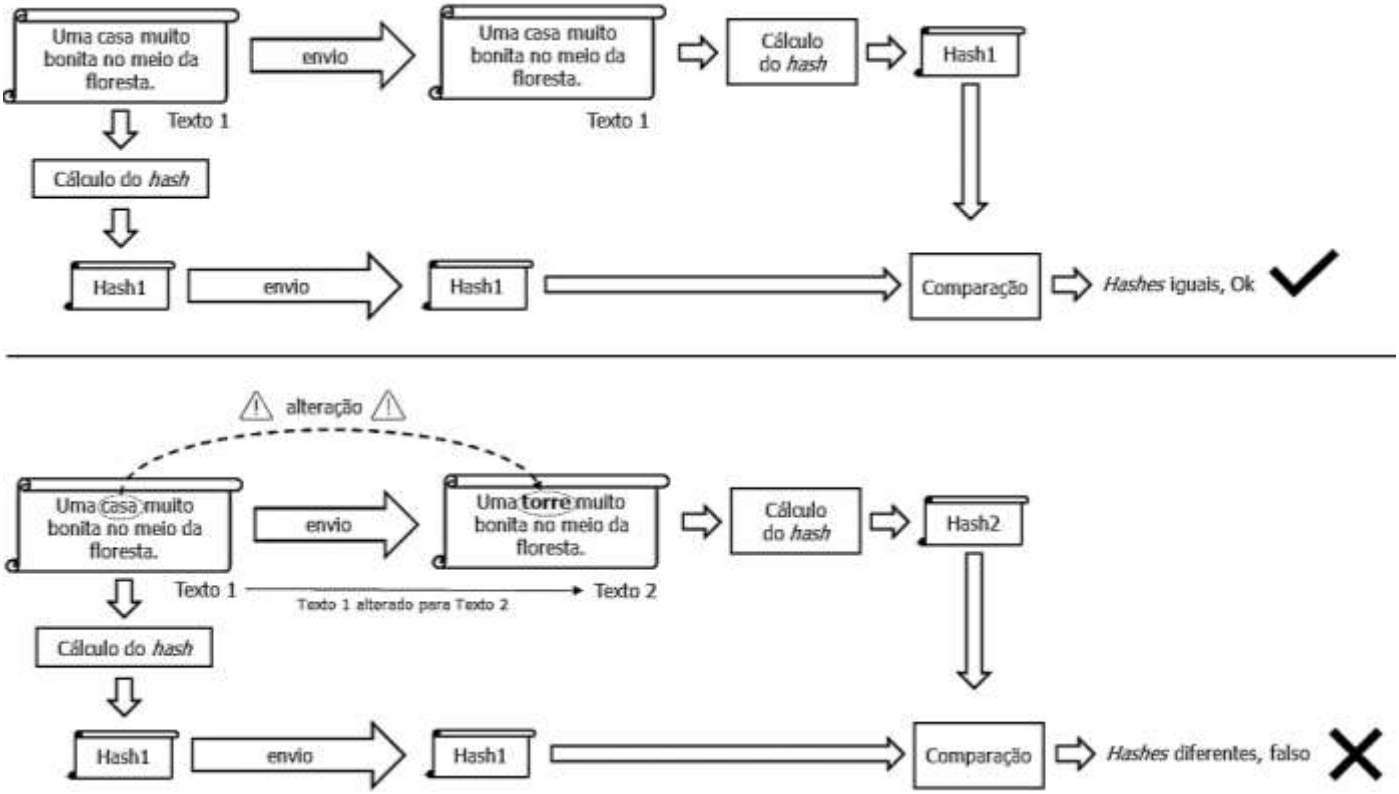


Figura 2. Processo de geração de *hash* para integridade.

Assim, o *hash* pode ser utilizado para garantir a integridade de um texto. Por exemplo, um *hash* calculado a partir de uma mensagem aberta pode ser enviado para um destinatário com a respectiva mensagem. O destinatário, ao receber a mensagem, pode calcular o *hash* da mensagem que chegou. Se o *hash* calculado for idêntico ao *hash* enviado com a mensagem original, o destinatário saberá que o conteúdo da mensagem original não foi alterado (como ilustrado na parte superior da figura 2). Se o valor do *hash* for diferente, houve alterações na mensagem original e quebra de integridade (ilustrado na parte inferior da figura 2).

O algoritmo *Secure Hash Algorithm* (SHA), ou, em português, Algoritmo de Hash Seguro, é uma família de funções criptográficas de *hash* desenvolvidas para produzir um valor

de *hash* fixo e único para dados de entrada de qualquer tamanho. A série SHA é amplamente utilizada para garantir a integridade dos dados e é fundamental em várias aplicações de segurança, como verificação de integridade de arquivos, senhas seguras e assinaturas digitais.

Existem diferentes versões do SHA, como SHA-1, SHA-256, SHA-384 e SHA-512, que variam em tamanho de saída do *hash* e na força criptográfica. O SHA-256 e SHA-384, por exemplo, são comumente empregados para garantir a segurança em várias implementações. Essas funções de *hash* são projetadas para serem computacionalmente difíceis de inverter, garantindo que seja praticamente impossível gerar os dados originais a partir do valor de *hash*.

1.3. Assinatura digital

Para evitar que mensagens abertas sejam alteradas e *hashes* falsos sejam gerados, a aplicação de autenticação nos *hashes* é fundamental. Isso é possível por meio de assinaturas digitais.

A assinatura digital é um método de criptografia que permite ao destinatário verificar a integridade, a autenticidade e a irretratabilidade de uma informação, observando se a mensagem realmente provém do remetente esperado e se não foi alterada por nenhum invasor.

A assinatura digital usa a técnica de cifragem assimétrica, utilizando a chave privada do remetente para a encriptação do *hash* da mensagem original, que só pode ser aberto com a chave pública do remetente. A aplicação de autenticação ocorre sobre o *hash* porque ele tem um tamanho fixo e, em geral, muito menor do que o tamanho da mensagem.

1.4. Criptografia de ponta a ponta

A criptografia de ponta a ponta é um método de comunicação segura que realiza a troca de informações encriptadas entre o remetente e o destinatário durante toda a cadeia de transferência. Somente o remetente e o destinatário têm a chave para decriptação da informação encriptada, não sendo possível a recuperação da informação por nenhuma outra entidade além deles.

Na área de criptografia, muitas vezes são necessários métodos adicionais para garantir a segurança. O Vetor de Inicialização (IV) é um valor adicional introduzido com a chave durante o processo de cifragem, especialmente em algoritmos de cifra de bloco, como o AES.

O principal propósito do IV é garantir que textos simples idênticos não produzam cifras idênticas, o que ajuda a fortalecer a segurança do sistema.

O IV é utilizado para inicializar o estado interno do algoritmo de cifra antes da cifragem propriamente dita. Ele é essencial para garantir que padrões repetitivos nos dados de entrada não resultem em padrões repetitivos nos dados cifrados, o que poderia ser explorado por atacantes para inferir informações sobre os dados originais.

É importante observar que o IV não precisa ser secreto e geralmente é transmitido com a mensagem cifrada. No entanto, ele deve ser único para cada mensagem cifrada com determinada chave. Em alguns casos, o IV é gerado aleatoriamente; em outros casos, pode ser derivado de informações específicas da mensagem ou contexto. O uso apropriado do IV contribui para a segurança e a robustez dos esquemas de criptografia.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Há dois pontos a serem observados em relação ao entendimento de como é realizada uma assinatura digital:

- a chave pública do remetente tem relação matemática com a chave privada dele (apesar de terem sido geradas juntas, alguém de posse da chave pública não consegue determinar qual a chave privada);
- a chave privada é utilizada para criptografar o *hash* de uma mensagem, sendo que apenas o remetente tem essa chave secreta.

Portanto, a abertura da assinatura digital com a chave pública (apesar de estar disponível a qualquer um) garante que o *hash* foi feito pelo remetente usando a chave privada (chave secreta que só ele tem).

II – Afirmativa correta.

JUSTIFICATIVA. O vetor de inicialização (IV) é um bloco aleatório de dados usado para iniciar a criptografia de vários blocos de texto simples, quando uma técnica de criptografia em cadeia de blocos é usada. Isso resulta em diferentes mensagens criptografadas, ainda que o conteúdo das mensagens seja o mesmo. Tal fato dificulta o possível reconhecimento de padrões, ainda que um terceiro seja capaz de interceptar as mensagens transmitidas.

III – Afirmativa incorreta.

JUSTIFICATIVA. O padrão de criptografia *Advanced Encryption Standard* (AES), publicada pelo NIST em 2001, é um padrão de criptografia simétrica, que tem por característica a utilização de uma única chave privada para a encriptação e a decriptação, e não um par de chaves distintas.

IV – Afirmativa correta.

JUSTIFICATIVA. O *Secure Hash Algorithm* (SHA) é um conjunto de funções *hash* desenvolvido pelo NIST em 1993. É utilizado para a criação de *hashes* que permitem a verificação da integridade de um texto. Caso o *hash* de um texto gerado por um remetente seja igual ao *hash* gerado pelo destinatário, isso significa que o texto enviado pelo remetente não passou por alterações e está íntegro. Caso os *hashes* estejam diferentes, isso indica que o texto enviado pelo remetente perdeu a integridade, tendo sido alterado ou corrompido.

Alternativa correta: E.

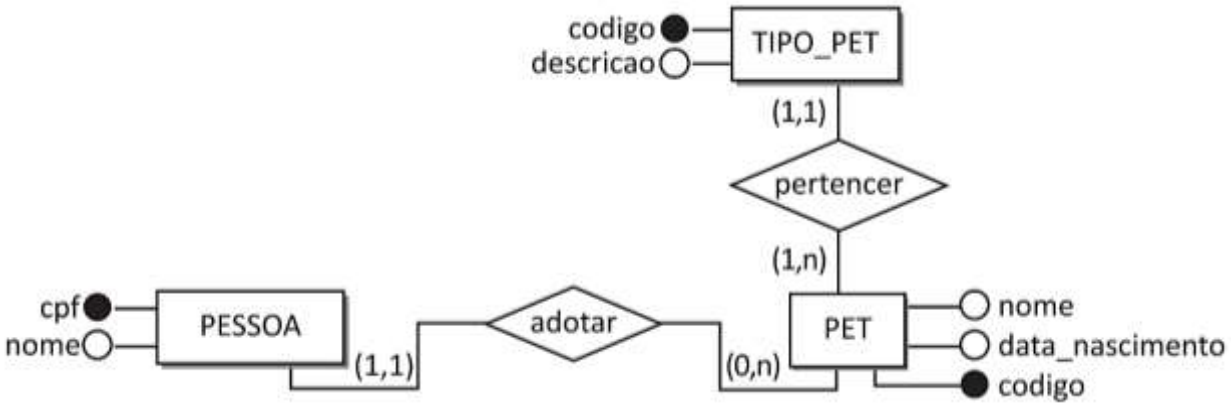
3. Indicações bibliográficas

- KATZ, J.; LINDELL, Y. *Introduction to modern cryptography*. Boca Raton: CRC Press, 2007.
- LUTKEVICH, B.; BACON, M. *End-to-end encryption (E2EE)*. Disponível em <https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE>. Acesso em 28 dez. 2023.
- STALLINGS, W. *Criptografia e segurança de redes: princípios e práticas*. 6. ed. São Paulo: Pearson Education do Brasil, 2015.

Questão 9

Questão 9.[9]

Uma organização não governamental (ONG), relacionada à causa animal, registra os pets (animais de estimação) amparados por ela, de acordo com o seguinte Diagrama Entidade Relacionamento (DER).



A partir das regras de mapeamento do Modelo Conceitual para o Modelo Lógico Relacional, assinale o Esquema Relacional mais adequado a ser gerado. Considere que as chaves primárias estão sublinhadas.

A. PESSOA(cpf: texto, nome: texto)
TIPO_PET(codigo: inteiro, descricao: texto)
PET(codigo: inteiro, nome: texto, data_nascimento: data, codigo_tipo_pet: inteiro, adotante: texto)
codigo_tipo_pet referencia TIPO_PET(codigo)
adotante referencia PESSOA(cpf)

B. PET(codigo: inteiro, nome: texto, data_nascimento: data)
PESSOA(cpf: texto, nome: texto, codigo_pet: inteiro)
codigo_pet referencia PET(codigo)
TIPO_PET(codigo: inteiro, descricao: texto, codigo_pet: inteiro)
codigo_pet referencia PET(codigo)

C. TIPO_PET(codigo: inteiro, descricao: texto)
PET(codigo: inteiro, nome: texto, data_nascimento: data, codigo_tipo_pet: inteiro)
codigo_tipo_pet referencia TIPO_PET(codigo)
PESSOA(cpf: texto, nome: texto, codigo_pet: inteiro)
codigo_pet referencia PET(codigo)

D. PET_PESSOA(codigo_pet: inteiro, nome_pet: texto, data_nascimento: data, cpf: texto, nome_pessoa: texto, codigo_tipo_pet: inteiro, descricao_tipo_pet: texto)

E. PESSOA(cpf: texto, nome: texto)
PET(codigo: inteiro, nome: texto, data_nascimento: data, codigo_tipo_pet: inteiro, descricao_tipo_pet, adotante: texto)
adotante referencia PESSOA(cpf)

[9]Questão 22 – 2021.

1. Introdução teórica

1.1. Banco de dados

Dados são fatos que podem ser registrados e têm um significado. Eles são a matéria-prima da informação, ou seja, a informação não tratada. A informação, por sua vez, é o conjunto desses dados organizados.

Um banco de dados é um conjunto de dados que apresentam alguma relação, como, por exemplo, nomes, endereços e telefones de pessoas conhecidas. Como se trata de uma definição bastante genérica, as seguintes características podem ajudar a melhorar o significado de um banco de dados:

- um banco de dados representa aspectos do mundo real;
- os dados têm coerência (não são simplesmente arbitrários ou aleatórios) e podem ter tamanho e complexidade variáveis;
- existe uma finalidade específica para os dados.

Um Sistema Gerenciador do Banco de Dados (SGBD) é uma aplicação que permite a criação, a manutenção e o compartilhamento de um banco de dados e, também, o acesso a ele.

As tabelas são componentes típicos de boa parte dos bancos de dados comumente utilizados na computação. As tabelas são compostas por diversos campos (colunas), e cada linha da tabela representa um registro "inserido" na tabela em questão. Um exemplo desse tipo de cenário pode ser visto na tabela 1, que apresenta dados sobre funcionários de uma empresa fictícia.

Tabela 1. Exemplo de tabela de um banco de dados, com os tipos e tamanhos associados

Campos → **Tipo →** **Tamanho →**	**Código** **(Caractere)** **(4)**	**Nome** **(Caractere)** **(60)**	**Cidade** **(Caractere)** **(20)**	**Refeição** **(Lógico)** **(1)**	**Nascimento** **(Data)** **(10)**	**Salário** **(Numérico)** **(10,2)**
Registro 1	5021	José Silva	São Paulo	S	03/07/1985	3500,00
Registro 2	5023	Maria Ana	Osasco	S	05/11/1955	8200,00
Registro 3	7022	João Alves	Ibiúna	N	25/03/1977	5200,00
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Como podemos ver na tabela 1, os campos podem ter um tipo e um tamanho associados. O tipo determina o formato dos dados que um campo recebe e serve para garantir a entrada de dados de forma correta no campo. O tamanho define a quantidade de caracteres ou números que determinado campo suporta, por exemplo.

Entre outras coisas, um SGBD também garante a integridade das tabelas de um banco de dados. Um dos principais mecanismos utilizados pelos SGBDs para garantir a integridade dos dados consiste no uso e na verificação das chamadas *restrições de integridade*. Existem vários tipos diferentes de integridade, como a integridade de domínio, a integridade da entidade e a integridade referencial.

A integridade de domínio verifica atributos e regras de validação para garantir que valores corretos sejam inseridos nos campos (por exemplo, um campo numérico não deve armazenar letras). A integridade da entidade deve garantir a unicidade dos valores da chave primária e alternativa.

Um exemplo de cenário no qual o SGBD deve verificar as restrições da entidade está em uma tentativa de inserção de um novo registro em uma tabela. A cada tentativa de inserção de um novo registro, o SGBD deve checar se a chave primária continua sendo única, bloqueando a inserção de registros que violarem essa regra, evitando a repetição da chave primária.

Por exemplo, suponha um SGBD de uma empresa com uma tabela chamada de "Funcionário", com a chave primária "código", um campo numérico que identifica cada funcionário da empresa. Se em algum momento um usuário tentar inserir um novo registro que tenha um código idêntico ao de outro registro (funcionário) já armazenado na tabela, o SGBD deve bloquear a tentativa, em função da verificação da restrição da entidade. Dessa forma, o SGBD está evitando uma situação na qual dois funcionários diferentes teriam o mesmo código associado.

A integridade referencial verifica se as operações no banco de dados atendem às regras de relacionamento definidas para o sistema. Por exemplo, em um sistema de gerenciamento de estoque, determinado produto deve estar associado a uma localização física (um local de armazenagem). Nesse exemplo, a remoção de um produto deve se refletir tanto na informação da sua própria disponibilidade quanto na informação da sua localização.

1.2. Modelo Entidade Relacionamento (MER)

A modelagem de dados relaciona-se à ideia de como organizar os dados que serão armazenados no banco de dados. Essa representação deve ressaltar os aspectos relevantes para criação do banco de dados. Em alguns casos, essa representação também envolve a maneira como serão construídas e organizadas as estruturas de dados e feitos os relacionamentos necessários.

Para a construção de um modelo desse tipo, é necessário o uso de uma linguagem de modelagem de dados. Podem ser utilizadas linguagens textuais ou gráficas para a descrição em diferentes níveis de abstração. Uma das etapas da criação de um banco de dados envolve a elaboração de um modelo lógico, que compreende uma descrição dos dados que serão armazenadas no banco. O resultado do modelo lógico é uma representação gráfica dos dados que nomeia cada um dos componentes e as ações que eles exercem uns sobre os outros.

O Modelo Entidade-Relacionamento é um modelo conceitual que identifica como as entidades (pessoas, objetos ou conceitos), com suas propriedades e suas características (atributos), relacionam-se em um sistema. O Diagrama Entidade Relacionamento (DER) é a representação gráfica deste modelo e contém os elementos a seguir.

- Retângulo: traz o nome de uma entidade, por exemplo uma pessoa ou um *pet*. Existem dois tipos de entidades: fortes e fracas.
    - Entidade forte: sua existência independe da existência de outras entidades.
    - Entidade fraca: sua existência depende de outras entidades. Uma entidade fraca e o seu relacionamento são representados por traços duplos.
- Losango: indica um relacionamento (conexão lógica entre duas entidades), por exemplo, "adotar" ou "pertencer". Existem três tipos básicos de relacionamentos, indicados a seguir.
    - Relacionamentos um para um (1:1): relacionamentos entre duas entidades, e cada uma delas faz referência obrigatoriamente a uma única unidade da outra.
    - Relacionamentos um para muitos (1:N): uma das entidades envolvidas pode fazer referência a várias unidades da outra entidade, entretanto o inverso não é verdadeiro.
    - Relacionamentos muitos para muitos (N:N): cada entidade pode fazer referência a múltiplas unidades da outra.
- Elipse: indica um atributo (cada entidade tem os seus próprios atributos). Podem ser dos tipos a seguir.
    - Atributo simples ou atômico: é aquele que não pode ser dividido, por exemplo, a idade de uma pessoa ou o seu salário.

- Atributo composto: é o resultado da composição de vários atributos simples, por exemplo, o atributo endereço pode ser decomposto por vários valores como rua, número do prédio, cidade, estado e CEP.

Veja a figura 1.

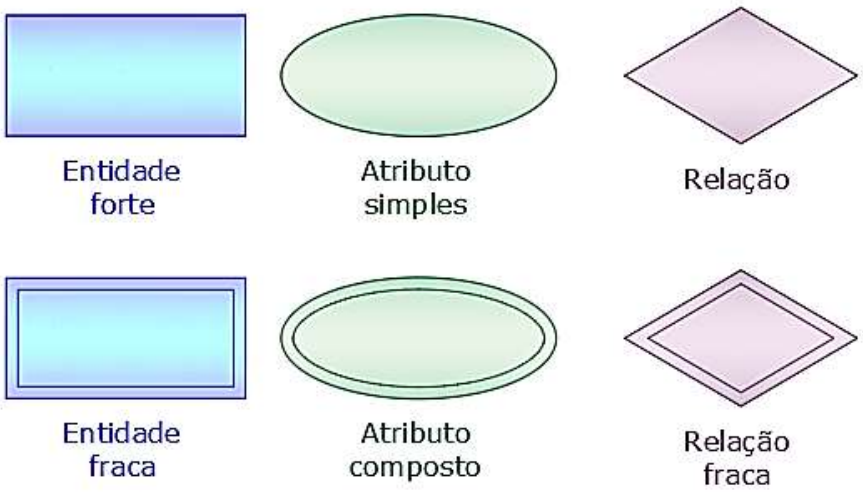


Figura 1. Símbolos.
Disponível em <https://www.edrawsoft.com/pt/entity-relationship-diagrams.html>. Acesso em 18 mar. 2024 (com adaptações).

Um atributo que identifica unicamente um registro em uma tabela é chamado de chave primária. Obviamente, ele não pode se repetir nessa tabela (ou não seria mais uma chave primária) nem pode ser nulo. O atributo chave primária é representado pelo nome do campo grifado ou é destacado por um círculo preto, no caso de um diagrama de entidade relacionamento.

Em geral, alguns formatos (ou, mais precisamente, tipos) de campos são adotados de acordo com a situação, como vemos a seguir.

- Texto: é um formato alfanumérico, em geral, usado para campos que contêm descritivos, mas também pode ser utilizado em campos de código cujo conteúdo seja formado por letras, números e caracteres especiais.
- Numérico: contém apenas números, que podem ser inteiros, decimais, negativos ou positivos.
- Data: usado quando o conteúdo do campo é uma data. Nesse caso, a maioria dos SGBD verifica em tempo real se as datas inseridas nas tabelas são válidas.
- Lógico: formato usado quando o conteúdo do campo é binário e contém uma indicação de verdadeiro ou falso.

2. Solução da questão e análise das alternativas

A partir dos formatos discutidos, as tabelas listadas no enunciado se adequam aos tipos de campos a seguir.

- PESSOA: tabela que traz informações sobre o cliente.
  - CPF: contém um campo chave/único com o número do CPF do cliente no formato 999.999.999-99. Por isso, um campo formado por caracteres alfanuméricos é o ideal.
  - NOME: contém o nome do cliente, que costuma ser representado por um formato alfanumérico.
- PET: tabela com dados sobre o animal de estimação.
  - CÓDIGO: é um campo chave usado como código único para cada animal. Como se trata de um código numérico, o formato de número inteiro é o melhor a ser adotado.
  - NOME: como se trata do nome do animal de estimação, com caracteres dos mais diversos tipos, um campo alfanumérico é o melhor formato para essa representação.
  - DATA_NASCIMENTO: o formato data (99/99/9999) é o ideal para este campo que marca a data em que o animal nasceu.
- TIPO_PET: tabela que armazena a descrição dos tipos de animais de estimação.
  - CÓDIGO: campo de tipo numérico.
  - DESCRIÇÃO: por ser um conteúdo descritivo do tipo do animal, o ideal é utilizar um campo alfanumérico.
- Esquema relacional: na tabela TIPO_PET, o campo código é a chave que representa os diferentes tipos de animais. Um PET deve ser identificado pelo campo código (sua chave primária). O campo CPF da tabela PESSOA é a chave que diferencia cada adotante de um animal. Essa construção viabiliza o MER, pois cada PET deve pertencer somente a uma PESSOA.

A – Alternativa correta.

JUSTIFICATIVA. A alternativa contempla todas as entidades contidas no diagrama de entidade relacionamento do enunciado, bem como os seus relacionamentos, conforme descrito anteriormente. Além disso, devemos observar que os diversos campos e os tipos associados a esses campos também são corretos.

B e C – Alternativas incorretas.

JUSTIFICATIVA. Os modelos propostos nas alternativas colocam uma chave estrangeira para PET na entidade PESSOA, o que significa que uma pessoa poderia adotar apenas um único animal (PET). Contudo, o diagrama de entidade relacionamento do enunciado sugere um relacionamento de um-para-muitos (na realidade, também existe a possibilidade de um animal não ser adotado por ninguém), o que significa que uma pessoa deve ser capaz de adotar mais de um animal (PET).

D – Alternativa incorreta.

JUSTIFICATIVA. Em termos de relacionamento, é desaconselhável a criação de uma tabela única que congregue os dados das entidades PESSOA, PET e TIPO_PET. A melhor solução é mantê-las separadas e ligadas por campos chave únicos, que garantam a integridade referencial proposta no modelo conceitual. Por exemplo, essas restrições devem garantir que cada pet seja adotado por apenas uma única pessoa e que esse mesmo pet seja de um único tipo. Dessa forma, podemos dizer que a solução proposta na alternativa é bastante inadequada, não normalizada, o que iria corresponder à criação de uma enorme tabela, com campos de várias entidades diferentes misturadas.

E – Alternativa incorreta.

JUSTIFICATIVA. A solução proposta na alternativa também contém sérios problemas de normalização. Por exemplo, registros da entidade PET que não fossem adotados por nenhuma pessoa teriam o campo "adotante" nulo, já que não foram adotados por ninguém. Por sua vez, o campo "adotante" é uma chave estrangeira para a tabela PESSOA, o que caracteriza um modelo ruim. Além disso, o campo "descricao_tipo_pet" da entidade PET (que, na descrição da alternativa, não apresenta nenhum tipo associado, o que também é um erro) seria repetido várias vezes na tabela PET para animais do mesmo tipo, o que também representa um problema de normalização.

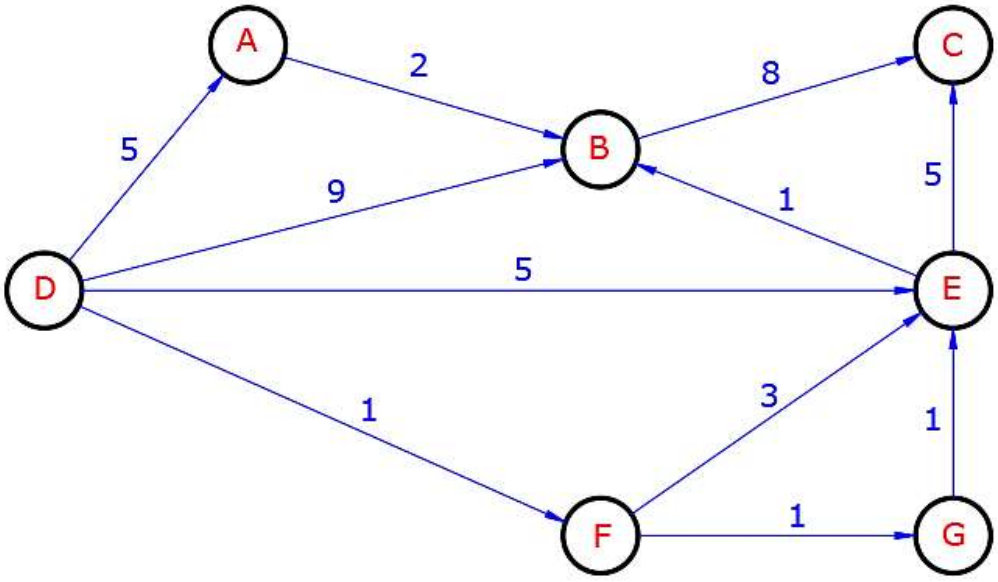
3. Indicações bibliográficas

- DATE, C. J. *Introdução a sistemas de banco de dados.* Rio de Janeiro: Elsevier, 2003.
- ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 7. ed. São Paulo: Pearson Education do Brasil, 2019.
- HEUSER, C. A. *Projeto de banco de dados*. 7. ed. Joinville: Clube dos Autores, 2024.

Questão 10

Questão 10.[10]

O algoritmo de Dijkstra para o problema do caminho mínimo em dígrafos com pesos utiliza uma fila de prioridades de vértices, na qual as prioridades são uma estimativa do custo final. A cada iteração, um vértice é retirado da fila, e os arcos que começam nesse vértice são analisados. Considere o seguinte grafo, no qual se deseja conhecer o custo de um caminho mínimo para cada vértice, a partir do vértice D. Considere que -1 representa um custo "infinito", ou seja, nenhum caminho até o vértice foi até o momento descoberto.



Com base nas informações e no grafo apresentados, assinale a alternativa que representa a estimativa de custo após duas iterações do algoritmo.

A. A: 5 B: 6 C: 10 D: 0 E: 4 F: 1 G: -1
B. A: 5 B: 9 C: -1 D: 0 E: 5 F: 1 G: -1
C. A: 5 B: 9 C: -1 D: 0 E: 4 F: 1 G: 2
D. A: 5 B: 7 C: 8 D: 0 E: 4 F: 1 G: 2
E. A: 5 B: 6 C: 8 D: 0 E: 3 F: 1 G: 2

[10]Questão 34 – Enade 2021.

1. Introdução Teórica

1.1. Teoria dos Grafos

A Teoria dos Grafos é um campo da Matemática muito utilizado em diversas áreas do conhecimento, especialmente na Computação. Um grafo é composto por dois conjuntos principais de elementos: vértices (também chamados de nós) e arestas (também chamadas de arcos). As arestas conectam pares de vértices, representando relações ou interações entre eles.

Os vértices (nós) representam entidades discretas ou pontos no grafo, enquanto as arestas (arcos) representam conexões ou relações entre pares de vértices. As conexões podem ser direcionadas (com uma orientação específica, normalmente representadas por uma seta) ou não direcionadas.

Em grafos direcionados, as arestas têm uma orientação específica, indicando uma relação unidirecional. Em grafos não direcionados, as arestas representam relações bidirecionais. Além dessas características, há uma série de propriedades e definições sobre os grafos que precisam ser discutidas para a melhor compreensão do assunto. Vamos relembrar, brevemente, alguns desses conceitos fundamentais.

- A atribuição de pesos às arestas é a forma utilizada para representar quantidades ou custos associados às relações. Isso é utilizado nos grafos ponderados.
- Um ciclo é uma sequência de arestas que forma um loop fechado.
- Uma árvore é um grafo sem ciclos e conectado;
- A conectividade refere-se à capacidade de alcançar todos os vértices a partir de um vértice específico.
- A planaridade é a propriedade que um grafo tem de ser representado em um plano sem que suas arestas se cruzem.
- A coloração de grafos é uma forma de organização, baseada na atribuição de cores aos vértices ou às arestas, de modo que vértices adjacentes (ou arestas conectadas) não compartilhem a mesma cor.

A Teoria dos Grafos é extensivamente aplicada em Ciência da Computação, redes de computadores, logística, biologia molecular, análise de redes sociais, otimização, transporte e planejamento urbano, entre outras áreas. Sua capacidade de modelar e analisar estruturas complexas de relacionamento faz com que ela seja uma ferramenta poderosa em diversos contextos.

1.2. Algoritmo de Dijkstra

O algoritmo de Dijkstra é utilizado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado, no qual as arestas têm pesos representando distâncias, custos, tempos ou qualquer outra métrica associada. O algoritmo foi proposto pelo cientista da computação Edsger Dijkstra em 1956.

A principal aplicação do algoritmo de Dijkstra é em redes de computadores, roteamento de pacotes, sistemas de transporte, logística e outras situações em que a minimização de custos em caminhos é fundamental. O algoritmo funciona apenas em grafos com arestas não negativas e pode ser implementado de maneira eficiente utilizando uma fila de prioridade para selecionar o próximo vértice a ser processado.

O funcionamento do algoritmo segue as etapas a seguir.

1) Inicialização: atribuir uma distância inicial para o vértice de origem (zero) e infinito para todos os outros vértices. Criar um conjunto que contenha todos os vértices ainda não processados.

2) Iteração: escolher o vértice com a menor distância no conjunto de vértices não processados. Atualizar as distâncias dos vértices vizinhos, considerando o peso das arestas e a distância acumulada até o momento.

3) Remoção e atualização: remover o vértice processado do conjunto e repetir a iteração até que todos os vértices sejam processados ou até que o vértice de destino seja atingido.

4) Resultado: após o término do algoritmo, as distâncias mínimas de origem para todos os outros vértices terão sido calculadas, bem como os caminhos mais curtos.

2. Solução da questão

Uma rede com vários nós (máquinas, equipamentos e dispositivos, entre outros) pode ser representada por um grafo, como no caso ilustrado pela figura do enunciado. No grafo, os nós correspondem aos círculos nomeados por letras (A, B, C, D, E e F) e são chamados de vértices. As linhas numeradas com setas são chamadas de arestas ou arcos, servem como ligação entre os elementos da rede e correspondem ao custo do caminho entre os nós. Os valores das arestas podem indicar diversos valores, como, por exemplo, o tempo de viagem, a distância percorrida ou a quantidade de roteadores em uma rota de rede.

Há um conjunto de iterações (passos) que definem o algoritmo de Dijkstra. A cada iteração, a soma dos custos é armazenada. Na sequência, os custos dos caminhos são calculados a partir do vértice D com a ajuda do algoritmo de Dijkstra.

2.1. Estimativa de custo da primeira iteração

Partindo-se do vértice D e analisando-se os valores das arestas, como indica o enunciado, existem as seguintes possibilidades de caminhos na primeira iteração mostradas na figura 1.

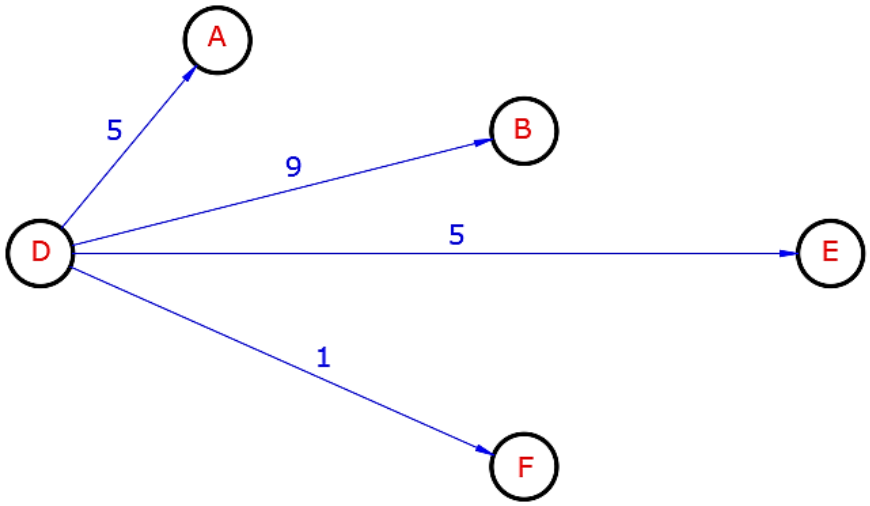


Figura 1. Caminhos possíveis partindo-se do vértice D.

Temos o que segue.

- Caminho de D para A = 5. Logo, (A: 5).
- Caminho de D para B = 9. Logo, (B: 9).
- Caminho de D para C = -1, pois não é possível na primeira iteração. Logo, (C: -1).
- Caminho de D para D = 0, pois esse é o ponto de partida. Logo, (D: 0).
- Caminho de D para E = 5. Logo, (E: 5).
- Caminho de D para F = 1. Logo, (F: 1).
- Caminho de D para G = -1, pois não é possível na primeira iteração. Logo: (G: -1).

Portanto, o resultado da estimativa de custo da primeira iteração é:

A: 5 B: 9 C: -1 D: 0 E: 5 F: 1 G: -1

Percebemos que o menor caminho em relação a D é 1 (de D para F). Isso significa que o vértice inicial considerado na iteração 2 será o vértice F, pois é o que tem a menor distância em relação a D.

2.2. Estimativa de custo da segunda iteração

A figura 2 utiliza o caminho de D para F (menor custo da iteração 1). A partir dele, há apenas dois caminhos possíveis na segunda iteração, de F para E e de F para G.

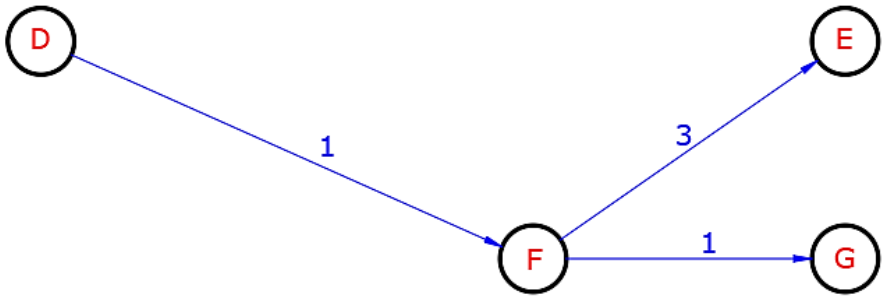


Figura 2. Caminhos possíveis partindo-se do vértice F.

Observando a figura 2, podemos perceber que a soma dos caminhos resulta nas considerações a seguir.

- Para calcularmos o custo do caminho do nó D para o nó G, somamos o custo do caminho do nó D para o nó F (igual a 1) e o valor do caminho do nó F para o nó G (igual 1), obtendo 2 como resultado. Logo, G: 2.
- Para calcularmos o custo do caminho do nó D para o nó E, somamos o custo do caminho do nó D para o nó F (igual a 1) e o valor do caminho do nó F para o nó E (igual 3), obtendo 4 como resultado. Logo, E: 4.

Mantendo os demais casos com os mesmos valores da iteração anterior e atualizando os valores para os nós G e E, obtemos a seguinte estimativa de custo na segunda iteração:

A: 5 B: 9 C: -1 D: 0 E: 4 F: 1 G: 2

3. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. A:5 B:6 C:10 D:0 E:4 F:1 G:-1. Os valores B:6, C:10 e G:-1 estão errados, pois deveriam ser B:9, C:-1 e G:2.

B – Alternativa incorreta.

JUSTIFICATIVA. A:5 B:9 C:-1 D:0 E:5 F:1 G:-1. Os valores E:5 e G:-1 estão errados, pois deveriam ser E:4 e G:2.

C – Alternativa correta.

JUSTIFICATIVA. Todos os valores de custo estão calculados corretamente para as duas iterações iniciadas no vértice D.

D – Alternativa incorreta.

JUSTIFICATIVA. A:5 B:7 C:8 D:0 E:4 F:1 G:2. Os valores B:7 e C:8 estão errados, pois deveriam ser B:9 e C:-1.

E – Alternativa incorreta.

JUSTIFICATIVA. A: 5 B:6 C:8 D: 0 E:3 F: 1 G: 2. Os valores B:6, C:8 e E:3 estão errados, pois deveriam ser B:9, C:-1 e E:4.

4. Indicações bibliográficas

- CORMEN, T. H. et al. *Introduction to algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- JOHNSONBAUGH, R. *Discrete Mathematics*. 4. ed. London: Prentice Hall International, 1997.
- LINTZMAYER, C. N.; MOTA, G. O. *Análise de algoritmos e estruturas de dados*. Disponível em <https://www.ime.usp.br/~mota/livros/livro_AAED.pdf>. Acesso em 19 set. 2023.
- NETTO, P. O. B. *Grafos – Teoria, modelos, algoritmos*. 5. ed. São Paulo: Blücher, 2012.

ÍNDICE REMISSIVO

Questão 1	Sistemas operacionais. Multiprogramação. Escalonamento de processos.
Questão 2	Sistemas operacionais. Escalonamento de processos. Região crítica. Exclusão mútua.
Questão 3	Java. Estrutura de dados. Listas. Filas. Pilhas. Deques.
Questão 4	Inteligência Artificial. Rede neural profunda. Redes convolucionais. Treinamento e aprendizado de máquinas.
Questão 5	Inteligência Artificial. Aprendizado de máquina. Regressão linear. Aprendizado supervisionado. Aprendizado não supervisionado.
Questão 6	Computação em nuvem. Serviço sob demanda. Software as a Service (SaaS). Plataforma como um Serviço (PaaS). Infraestrutura como um Serviço (IaaS).
Questão 7	Árvore binária de busca. Busca em pré-ordem. Busca em-ordem. Busca em pós-ordem.
Questão 8	Criptografia. Criptografia simétrica. Chaves pública e privada. Algoritmo de Hash.
Questão 9	Banco de dados. Diagrama Entidade Relacionamento (DER). Modelo conceitual. Modelo lógico-relacional.
Questão 10	Grafos. Algoritmo de Dijkstra. Caminho mínimo.