UNIP
EDUCAÇÃO, POR SUA ESCOLHA

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 17

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 17

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Estatística Aplicada e Ciência de Dados, 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.

Tarcísio de Souza Peres

Mestre em Ciências, Especialista em Gestão de Projetos e Engenheiro da Computação, com MBA em Gestão de TI. Cursou Gestão Estratégica e Competitividade. Professor adjunto da Universidade Paulista.

Questão 1

Questão 1.[1]

Leia o texto a seguir.

O protocolo de roteamento interno OSPF (open shortest path first) representa um sistema autônomo (SA) como um grafo ponderado, em que roteadores são os vértices, conexões entre os roteadores são as arestas e atrasos nas conexões são os pesos. No OSPF, a identificação de cada conexão e seu respectivo atraso são passados de roteador a roteador até que todos os roteadores formem uma base de dados com o grafo que descreve o SA. O OSPF utiliza uma versão distribuída do algoritmo de caminhos mínimos de Dijkstra para computar as melhores rotas para todos os possíveis destinos e para produzir as tabelas de rotas para cada roteador. Cada rota computada é a que apresenta o menor valor para a soma dos atrasos nas conexões usadas na rota entre a rede de origem e a rede de destino.

Disponível em <https://memoria.rnp.br/newsgen/9705/n1-1.html>. Acesso em 1 mar. 2023 (com adaptações).

Acerca do protocolo OSPF e com base nas informações apresentadas no texto, avalie as afirmativas.

I. Quando há diferentes caminhos possíveis entre uma origem e um destino, a rota selecionada é a que apresenta o menor número de conexões.

II. Há uma instância da base de dados relativa a conexões e a atrasos, formando o grafo que descreve o SA em cada roteador que compõe o SA.

III. O algoritmo de Dijkstra é executado por um único roteador dentro do SA e a tabela de rotas resultante é passada para todos os roteadores no SA.

IV. Para todos os roteadores dentro de um SA, há a necessidade de tráfego de informações acerca de atrasos e de conexões entre roteadores.

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.

1. Introdução teórica

1.1. Sistemas autônomos e redes

Em computação, um sistema autônomo é um sistema computacional capaz de operar com um grau relevante de independência operacional, isto é, de perceber os estados do

[1]Questão 11 – Enade 2023.

ambiente (ou do próprio sistema), de processar informações, de tomar decisões segundo algumas regras e determinados modelos e de executar certas ações sem a intervenção humana contínua, mantendo seu comportamento de acordo com as restrições e os objetivos definidos.

Essa definição é dependente do domínio e, por isso, em redes de computadores, o termo apresenta um significado técnico específico e consagrado (que veremos em breve). Em robótica e em IA (Inteligência Artificial), designa agentes ou plataformas que percebem, planejam e atuam de forma adaptativa. Em teoria de sistemas, pode referir-se a modelos cuja dinâmica não depende explicitamente do tempo, de modo que "autônomo" descreve a forma de controle e de decisão do sistema, e não a ausência de projeto, de supervisão ou de limites impostos por humanos.

No contexto do roteamento em redes IP, um sistema autônomo é um conjunto de redes e de roteadores sob uma mesma administração técnica, que adota as políticas de roteamento coerentes e se apresenta de forma unificada para o restante da Internet. Muitos protocolos de roteamento foram concebidos para operar nesses domínios administrativos, nos quais é razoável supor alguns objetivos comuns, alguns critérios estáveis de engenharia de tráfego e alguns mecanismos uniformes de configuração. Nesse domínio, o problema essencial é decidir, para cada roteador, quais caminhos devem ser usados para encaminhar seus pacotes até as diferentes redes de destino, de modo a satisfazer um critério de "melhor caminho" definido por uma métrica. Para descrever e resolver esse problema com precisão, recorre-se à teoria dos grafos.

1.2. Grafos

Um grafo é uma estrutura matemática composta por vértices e por arestas, utilizada para representar as relações entre elementos. Quando se associa um valor numérico às arestas, por exemplo, para representar o custo, o atraso ou outro parâmetro, obtém-se um grafo ponderado. Esse formalismo é especialmente adequado para as redes de computadores, pois permite transformar uma topologia física e lógica (os roteadores interconectados por enlaces) em um objeto matemático sobre o qual se podem aplicar os algoritmos bem estabelecidos. A passagem do "mundo físico" para o "modelo" é direta: cada roteador é representado como um vértice e cada enlace que conecta dois roteadores é representado como uma aresta; o peso dessa aresta expressa a métrica escolhida para comparar alternativas de encaminhamento.

Ao modelar a rede como um grafo ponderado, o roteamento interno pode ser entendido como um problema de caminhos mínimos. Dado um roteador de origem, deseja-se encontrar, para cada destino, um caminho cujo custo total - entendido como a soma dos pesos das arestas ao longo do trajeto - seja mínimo. Essa formulação é importante porque deixa claro que "melhor caminho" não significa necessariamente "caminho com menos saltos". Uma rota com mais enlaces pode ser preferível se os pesos associados a esses enlaces resultarem em menor custo total. Para ilustrar, considere o grafo ponderado a seguir (a rota de menor custo está destacada em verde).

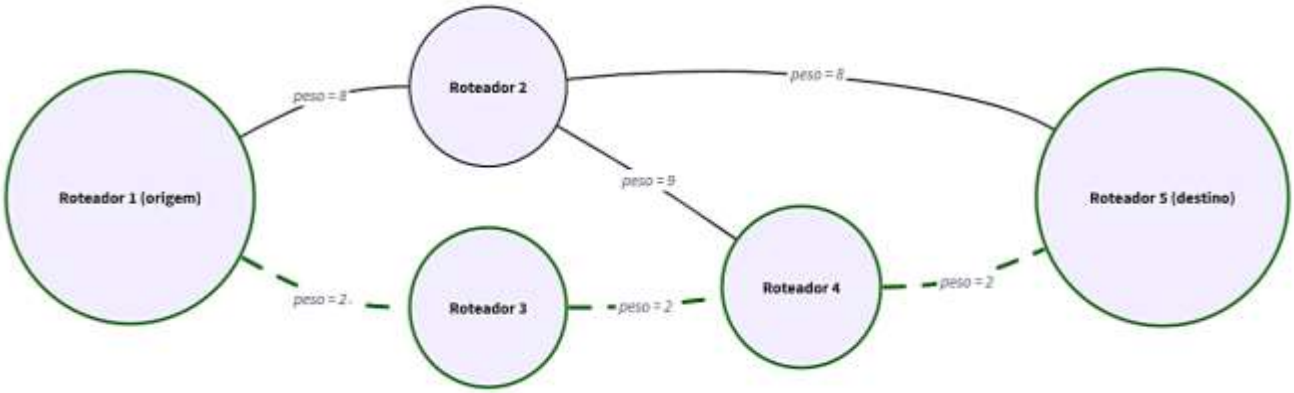


Figura 1. Rota de menor custo (em verde) em um grafo.

Assim, a qualidade de uma rota depende do modo como a métrica é definida e atribuída aos enlaces, e não apenas da contagem de conexões intermediárias.

1.3. Algoritmos de caminho mínimo

Entre os algoritmos clássicos para resolver o problema de caminho mínimo em grafos com pesos não negativos, destaca-se o algoritmo de Dijkstra. Sua ideia central é construir gradualmente um conjunto de vértices para os quais o menor custo a partir da origem já é conhecido de forma definitiva. O processo começa fixando o custo zero para a origem e os custos inicialmente "infinitos" para os demais vértices. Em seguida, escolhe-se repetidamente o vértice ainda não fixado com o menor custo provisório, fixa-se esse custo como o menor possível e relaxam-se as arestas que partem desse vértice, isto é, tenta-se melhorar (reduzir) os custos provisórios dos vizinhos ao considerar caminhos que passam por ele. Esse método é correto sob a condição de que os pesos sejam não negativos, exatamente o cenário típico quando pesos representam os custos de enlace. Ao final, obtém-se uma árvore de caminhos mínimos enraizada na origem, que informa tanto o custo mínimo até cada destino quanto o próximo salto a ser usado para alcançá-lo.

1.4. OSPF

É justamente essa combinação - a modelagem por grafos ponderados e o uso de caminhos mínimos - que sustenta o funcionamento do OSPF. O OSPF é um protocolo de roteamento interno do tipo estado de enlace (link-state) projetado para operar em um sistema autônomo. Em vez de cada roteador anunciar apenas quais destinos ele conhece e a que distância eles estão (como ocorre nos protocolos do tipo vetor de distância), no OSPF cada roteador anuncia descrições do estado de seus enlaces: quem são seus vizinhos e quais são os custos associados às conexões. Essas descrições são disseminadas de forma controlada para que todos os roteadores de uma mesma área mantenham uma base de dados coerente da topologia, conhecida como base de dados de estado de enlace.

A partir dessa visão comum do grafo, cada roteador executa localmente o algoritmo de Dijkstra com ele próprio como origem, produzindo sua árvore de caminhos mínimos e, a partir dela, sua tabela de roteamento. Desse modo, a convergência do OSPF depende da disseminação consistente da informação topológica e do recálculo do caminho mínimo quando há mudanças relevantes (por exemplo, uma falha de enlace ou uma alteração de custo), preservando a propriedade de que a rota escolhida é aquela que minimiza a soma dos pesos ao longo do caminho, conforme a métrica configurada.

2. Análise das afirmativas

I – Afirmativa incorreta.

JUSTIFICATIVA. O enunciado afirma que o OSPF representa o sistema autônomo como um grafo ponderado e que as melhores rotas são obtidas pela aplicação de um algoritmo de caminhos mínimos, no qual o critério de escolha é o menor valor para a soma dos pesos (no texto, "atrasos") ao longo do caminho. Isso significa que, havendo caminhos alternativos entre uma origem e um destino, a rota escolhida é aquela cujo custo total é mínimo, e não necessariamente a que tem menos conexões (menos enlaces). Em um grafo ponderado, um caminho com mais arestas pode ter custo menor do que outro com menos arestas, se os pesos forem menores; portanto, "menor número de conexões" não é o critério descrito.

II – Afirmativa correta.

JUSTIFICATIVA. O texto descreve que, no OSPF, a identificação das conexões e dos respectivos "atrasos" é difundida entre os roteadores para que "todos os roteadores formem

uma base de dados com o grafo que descreve o SA". Se todos precisam formar essa base de dados do grafo, então cada roteador que compõe o sistema autônomo mantém uma instância local dessa base (isto é, uma cópia/representação do mesmo grafo, dentro do escopo do SA considerado). Essa interpretação é exatamente a lógica de protocolos link-state: cada nó mantém uma base topológica e, a partir dela, calcula suas rotas.

III – Afirmativa incorreta.
JUSTIFICATIVA. O enunciado indica explicitamente que o OSPF utiliza "uma versão distribuída do algoritmo de caminhos mínimos de Dijkstra" para "produzir as tabelas de rotas para cada roteador". A noção de versão distribuída, em consonância com "para cada roteador", significa que o cálculo não fica concentrado em um único roteador; ao contrário, cada roteador, de posse da base com o grafo, executa o procedimento de cálculo para obter sua própria tabela. Se apenas um roteador calculasse e repassasse a tabela pronta, o processo não seria distribuído no sentido indicado no texto e não haveria necessidade de cada roteador produzir a sua tabela a partir do grafo.

IV – Afirmativa correta.
JUSTIFICATIVA. O texto afirma que a identificação de cada conexão e do respectivo "atraso" é passada de roteador em roteador, até que todos formem a base de dados do grafo do sistema autônomo. Isso implica, necessariamente, o tráfego de informações sobre a conectividade (quais enlaces existem, entre quais roteadores) e sobre os pesos associados a esses enlaces (no texto, os "atrasos"). Sem essa circulação de informação, não seria possível que todos os roteadores reconstruíssem o mesmo grafo e, consequentemente, calculassem as rotas consistentes com o estado da rede.

Alternativa correta: C.

3. Indicações bibliográficas

- CARROLL, J. D.; DOYLE, J*. Routing TCP/IP, Volume II: CCIE Professional Development*. 2. ed. Indianapolis/IN: Cisco Press, 2016.
- CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 4. ed. Rio de Janeiro: GEN LTC, 2024.
- DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. *Algoritmos*. Tradução de Guilherme Albuquerque Pinto. AMGH, 2009.

- DOYLE, J. *OSPF and IS-IS: Choosing an IGP for Large-Scale Networks*. Upper Saddle River/NJ: Addison-Wesley Professional, 2006.
- MOY, J. T. *OSPF: Anatomy of an Internet Routing Protocol*. Reading, Mass: Addison-Wesley, 1998.
- TANENBAUM, A.; FEAMSTER, N.; WETHERALL, D. *Redes de computadores (coedição Bookman e Pearson)*. 6. ed. São Paulo: Bookman, 2021.

Questão 2

Questão 2.[2]

Leia o texto a seguir.

*Uma relação **R** em um conjunto **S** é uma relação de equivalência se ela satisfizer todas as propriedades a seguir.*
*1. Reflexividade. Para todo a ∈ **S**, a **R** a;*

*2. Simetria. Para todos a, b ∈ **S**, a **R** b ⇔b **R** a;*

*3. Transitividade. Para todos a, b, c ∈ **S**, se a **R** b e b **R** c, então, a **R** c.*
*Uma partição de um conjunto **S** é uma coleção de subconjuntos disjuntos não vazios, cuja união é igual a **S**. Se **R** é uma relação de equivalência em um conjunto **S** e se x ∈ **S**, denota-se por [x] o conjunto de todos os elementos relacionados a x em **S** e chama-se esse conjunto de classe de equivalência de x.*

GERSTING, J. L. *Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta*. 5. ed. Rio de Janeiro: LTC, 2008 (com adaptações).

A partir dessas informações, considere que, em um cluster computacional, haja 10 computadores com configurações de hardware diferentes. Os administradores desse cluster pretendem desenvolver um algoritmo de escalonamento de tarefas que mantenha processos de uma mesma aplicação sendo executados em máquinas semelhantes, mesmo com estruturas arquiteturais distintas. Assim, os administradores fizeram uma tabela relacionando os computadores. Essa tabela foi montada considerando pares de computadores. Dessa forma, dois computadores do cluster fazem parte de uma linha na tabela se possuem alguma característica semelhante, ou seja, se apresentam algum tipo de relacionamento (por exemplo, quantidade de memória ou de núcleos de processamento semelhantes). Obviamente, apesar de não estar evidente na tabela, um computador também tem relação consigo mesmo. Os computadores estão numerados de 1 até 10 e a tabela resultante pode ser vista a seguir

Computadores com características semelhantes	
1	5
2	8
5	7
7	10
2	9
3	4
10	1
5	10
7	1
9	8

[2]Questão 12 – Enade 2023.

Com base nesse cenário e no conceito de relações de equivalência, assinale a opção correta.

A. A relação descrita pela tabela é uma relação de equivalência.
B. A relação descrita pela tabela apresenta a propriedade de simetria, mas não a de transitividade.
C. A relação descrita pela tabela apresenta a propriedade de transitividade, mas não a de simetria.
D. O subconjunto de computadores representados pelos números 1, 3, 5, 7 e 10 forma uma classe de equivalência.
E. O subconjunto de computadores representados pelos números 2, 3, 4, 8 e 9 forma uma classe de equivalência.

1. Introdução teórica

Relação de equivalência

Uma relação binária R em um conjunto S é um subconjunto de $S \times S$. Quando $(a, b) \in R$, escreve-se $a\,R\,b$. Uma relação R em S é uma relação de equivalência quando satisfaz, simultaneamente, as propriedades a seguir.

- Reflexividade: para todo $a \in S$, $a\,R\,a$.
- Simetria: para todos $a, b \in S$, $a\,R\,b \Rightarrow b\,R\,a$.
- Transitividade: para todos $a, b, c \in S$, $(a\,R\,b \text{ e } b\,R\,c) \Rightarrow a\,R\,c$.

A relação R é de equivalência: ela separa o conjunto S em grupos de elementos indistinguíveis segundo o critério definido por R. Esses grupos são as classes de equivalência. Para $x \in S$, a classe de equivalência de x é

$$[x] = \{y \in S : y\,R\,x\}$$

Há uma ligação fundamental entre os dois conceitos, como explicamos a seguir.

- Cada relação de equivalência em S determina uma partição de S em classes de equivalência (subconjuntos não vazios, dois a dois disjuntos, cuja união é S).
- Cada partição de S determina uma relação de equivalência: $a\,R\,b$ se, e somente se, a e b pertencem ao mesmo bloco da partição.

2. Análise das alternativas

A - Alternativa correta.

JUSTIFICATIVA. Uma relação de equivalência deve satisfazer, ao mesmo tempo, a reflexividade, a simetria e a transitividade. No enunciado, a reflexividade é assumida ("um computador também tem relação consigo mesmo"). A simetria decorre do sentido de "par de computadores semelhantes": se o computador "a" é semelhante ao computador "b", então "b" é semelhante ao "a". A transitividade é verificada porque os conjuntos sugeridos pela tabela formam grupos fechados: {1,5,7,10}, {2,8,9}, {3,4}, e {6}. Dentro de cada grupo, sempre que a R b e b R c, vale a R c (por exemplo, 1 R 5 e 5 R 7 implicam 1 R 7, o que também aparece na tabela como o par (7,1)).

B - Alternativa incorreta.

JUSTIFICATIVA. A alternativa afirma que existe simetria, mas não existe transitividade. Entretanto, a transitividade é satisfeita nos agrupamentos exibidos. Exemplos diretos: 1 R 5 e 5 R 10 implicam 1 R 10, que aparece como o par (10,1). Além disso, 2 R 8 e 8 R 9 implicam 2 R 9, que aparece como o par (2,9). Esse comportamento é o esperado quando os dados descrevem classes de equivalência, pois as classes correspondem a blocos de uma partição.

C - Alternativa incorreta.

JUSTIFICATIVA. A alternativa afirma que existe transitividade, mas não existe simetria. No contexto de "pares de computadores semelhantes", o par (a,b) representa semelhança mútua. Logo, se a R b, então b R a. Assim, a simetria está presente no critério usado para montar a tabela. Formalmente, isso equivale a interpretar cada linha como um par não ordenado que gera os dois pares ordenados (a,b) e (b,a), conforme a ideia de simetria em relações de equivalência.

D - Alternativa incorreta.

JUSTIFICATIVA. O conjunto {1,3,5,7,10} não pode ser uma classe de equivalência porque o computador 3 aparece relacionado somente com o computador 4 (par (3,4)), formando a classe {3,4}. Como 3 não está relacionado com 1, 5, 7 e 10, ele não pertence à mesma classe desses elementos. Em uma classe de equivalência, todos os elementos devem ser equivalentes entre si.

E - Alternativa incorreta.

JUSTIFICATIVA. O conjunto {2,3,4,8,9} reúne elementos de duas classes distintas: {2,8,9} e {3,4}. Não há, na tabela, relação entre 2 e 3 (nem entre 2 e 4), então não existe uma única classe que contenha simultaneamente esses cinco elementos. Uma classe de equivalência não pode "fundir" dois blocos diferentes de uma partição.

Alternativa correta: A.

3. Indicações bibliográficas

- EPP, S. S. *Discrete Mathematics with Applications*. 5. ed. Boston: Cengage, 2020.
- GERSTING, J. L. *Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta*. 5. ed. Rio de Janeiro: LTC, 2008.
- GRIMALDI, R. P. *Discrete and Combinatorial Mathematics: An Applied Introduction*. 5. ed. Boston: Pearson Addison Wesley, 2004.
- ROSEN, K. H. *Discrete Mathematics and Its Applications*. 8. ed. New York: McGraw-Hill Education, 2019.

Questão 3

Questão 3.[3]

Uma lista pode ser dividida em duas partes: o primeiro elemento (a cabeça da lista) e os demais elementos (sua cauda). Por exemplo, em uma lista de inteiros `[1, 2, 3, 4]`, a cabeça dessa lista é o valor inteiro `1`, enquanto sua cauda é a lista de inteiros `[2, 3, 4]`. Uma lista vazia é representada por `[]`.

O código a seguir define duas funções descritas em uma linguagem de programação funcional que manipulam listas de inteiros. A função `enade` recebe uma lista de inteiros e produz uma nova lista de inteiros. A função `auxiliar` é chamada pela função `enade` e possui dois parâmetros: um número inteiro e uma lista de inteiros. Essa função produz uma lista de inteiros.

```
enade :: [Int] -> [Int]
enade [] = []
enade (cabeca:cauda) = auxiliar cabeca (enade cauda)
auxiliar :: Int -> [Int] -> [Int]
auxiliar x [] = [x]
auxiliar x (cabeca:cauda)
  | (x `mod` 2 == 0) = x:cabeca:cauda
  | otherwise = cabeca:auxiliar x cauda
```

Considerando o código apresentado, é correto afirmar que se a função `enade` for executada recebendo como parâmetro de entrada a lista `[1, 2, 3, 4, 5, 6, 7, 8]`, o resultado será

A. `[].`

B. `[2, 4, 6, 8].`

C. `[1, 2, 3, 4, 5, 6, 7, 8].`

D. `[2, 4, 6, 8, 1, 3, 5, 7].`

E. `[2, 4, 6, 8, 7, 5, 3, 1].`

1. Introdução teórica

Linguagens funcionais costumam favorecer uma forma de programação em que a estrutura dos dados orienta a estrutura das funções. As listas, por sua definição em termos de cabeça e de cauda, oferecem um exemplo clássico dessa ideia. A recursão fornece o

[3]Questão 14 – Enade 2023.

mecanismo de processamento, e o pattern matching oferece uma forma expressiva e rigorosa de descrever os casos possíveis. Juntos, esses elementos compõem uma base conceitual importante para o estudo e a prática da programação funcional.

1.1. Programação funcional

Nas linguagens de programação funcionais, o foco do raciocínio do programador costuma recair sobre as funções e sobre a transformação de valores, em vez da sequência de comandos que modificam um estado de memória ao longo do tempo. Em termos conceituais, um programa é descrito como um conjunto de definições que relacionam as entradas e as saídas.

Em muitas linguagens desse paradigma (como Haskell, OCaml, F# e, em parte, Elixir), aparecem com frequência ideias como a imutabilidade, a composição de funções e as definições guiadas pela estrutura dos dados.

1.2. Listas

Uma lista é, em geral, uma estrutura de dados sequencial definida de forma indutiva. Isso significa que ela pode ser descrita por dois casos fundamentais: (1) a lista vazia e (2) uma lista formada por um elemento inicial, chamado de cabeça (head), seguido do restante da lista, chamado de cauda (tail).

Em notação típica de linguagens funcionais, a lista [1,2,3] pode ser vista como 1 : (2 : (3 : [])).Essa leitura é importante porque ela revela a "forma interna" da lista: há sempre um caso-base (a lista vazia) e um caso recursivo (cabeça + cauda).

1.3. Recursão

A recursão é a técnica em que uma definição faz referência a si mesma, de maneira controlada, para resolver um problema. Em programação funcional, a recursão costuma substituir os laços imperativos (for, while) em muitos contextos. Uma definição recursiva correta precisa de dois componentes: um caso-base, que interrompe a chamada recursiva, e um passo recursivo, que reduz o problema a uma versão menor dele mesmo. Em listas, isso se encaixa de forma natural: a lista vazia encerra o processo, e a lista com cabeça e com cauda permite continuar o processamento sobre a cauda.

Os algoritmos recursivos em listas exploram exatamente essa estrutura. Para calcular a soma dos elementos de uma lista, por exemplo, define-se que a soma da lista vazia é 0 (caso-base) e que a soma de uma lista não vazia é a cabeça somada à soma da cauda (passo recursivo). O mesmo esquema aparece em funções como o comprimento da lista, a busca de um elemento, a transformação de elementos (map), a filtragem (filter) e a redução (fold). O ponto importante é que a forma do algoritmo acompanha a forma do dado: se a lista é "vazia" ou "cabeça + cauda", a função também terá casos que refletem essa decomposição.

1.4. Pattern Matching

É nesse contexto que o pattern matching (casamento de padrões) se torna especialmente útil. Trata-se de um mecanismo de definição e de seleção de casos em que a própria forma do dado é usada para decidir qual regra aplicar. Em vez de escrever testes manuais para verificar se a lista está vazia ou para acessar seus elementos por índices, o programador declara padrões como [] (lista vazia) e x:xs (uma lista com cabeça x e cauda xs). Quando o valor recebido corresponde a um desses padrões, a linguagem associa automaticamente nomes às partes relevantes e executa a definição correspondente. Isso torna o código mais legível e mais próximo da estrutura conceitual do problema.

2. Análise da questão

O enunciado não nomeia explicitamente a linguagem, mas, pela notação usada, trata-se de uma escrita típica de Haskell (ou de um pseudocódigo fortemente inspirado em Haskell), que é uma linguagem funcional.

A ideia central da questão é perceber que a função foi construída com pattern matching e recursão sobre listas. Primeiro, ela distingue dois casos estruturais: o caso da lista vazia e o caso da lista não vazia, que é decomposta em cabeça (primeiro elemento) e cauda (restante da lista). Esse reconhecimento da forma da lista é o pattern matching. A partir dessa decomposição, a função principal trata a lista de modo recursivo: ela processa, em primeiro lugar, a cauda e só depois reinsere a cabeça no resultado já obtido. É isso que permite dizer, de forma intuitiva, que a lista é tratada de trás para frente, pois os últimos elementos ficam prontos antes dos primeiros.

A função auxiliar também usa pattern matching para separar o caso em que a lista está vazia do caso em que ela tem cabeça e cauda. Depois que o padrão da lista é reconhecido, entra a

regra condicional (as guardas): se o número a ser inserido for par, ele é colocado no início da lista; se for ímpar, ele é deslocado para mais adiante, até o final. Assim, ao longo do processo, os números pares vão sendo colocados na frente, enquanto os ímpares acabam ficando no fim.

Vamos aplicar essa lógica à entrada [1, 2, 3, 4, 5, 6, 7, 8]. O último elemento é o 8, que é par. Como ele é o primeiro a ser processado, a nova lista começa com [8]. Em seguida, entra o 7, que é ímpar. Então, ele vai para o final da lista já formada, ficando [8, 7]. Depois, entra o 6, que é par. Então, ela vai para o começo, produzindo [6, 8, 7]. Depois, entra o 5, que é ímpar. Então, ele vai para o final, resultando em [6, 8, 7, 5].

Continuando o mesmo raciocínio, entra o 4 (par), que vai para o começo, formando [4, 6, 8, 7, 5]. Depois entra o 3 (ímpar), que vai para o final, ficando [4, 6, 8, 7, 5, 3]. Em seguida, entra o 2 (par), que vai para o começo, gerando [2, 4, 6, 8, 7, 5, 3]. Por fim, entra o 1 (ímpar), que vai para o final, e o resultado final passa a ser [2, 4, 6, 8, 7, 5, 3, 1].

Portanto, o resultado da execução da função com a lista [1, 2, 3, 4, 5, 6, 7, 8] é [2, 4, 6, 8, 7, 5, 3, 1].

Alternativa correta: E.

3. Indicações bibliográficas

- BIRD, R. *Introduction to Functional Programming using Haskell*. 2. ed. Prentice Hall, 1998.
- CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 4. ed. Rio de Janeiro: GEN LTC, 2024.
- DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. *Algoritmos*. AMGH, 2009.
- HUTTON, G. *Programming in Haskell*. 2. ed. Cambridge University Press, 2016.
- O'SULLIVAN, B.; GOERZEN, J.; STEWART, D. B. *Real World Haskell*. O'Reilly Media, 2008.
- THOMPSON, S. *Haskell: The Craft of Functional Programming*. 3. ed. Pearson Education, 2015.

Questão 4

Questão 4.[4]

Memory leak, ou vazamento de memória, é um problema que ocorre em sistemas computacionais quando uma parte da memória, alocada para uma determinada operação, não é liberada quando se torna desnecessária. Na linguagem C, esse tipo de problema é quase sempre relacionado ao uso incorreto das funções `malloc( )` e `free( )`. Esse erro de programação pode levar a falhas no sistema se a memória for completamente consumida.

A partir dessas informações, assinale a opção que apresenta um trecho com memory leak.

A.
```
void f( ){
void *s;
s = malloc(50);
free(s);
}
```

B.
```
int f( ){
 float *a;
 return 0;
}
```

C.
```
int f(char *data){
void *s;
s = malloc(50);
int size = strlen(data);
if (size > 50)
 return(-1);
free(s);
return 0;
}
```

D.
```
int *f(int n){
int *num = malloc(sizeof(int)*n);
return num;
}
int main(void){
  int *num;
  num = f(10);
  free(num);
  return 0;
}
```

[4]Questão 15 – Enade 2023.

E.
```
void f(int n){
  char *m = malloc(10);
  char *n = malloc(10);
  free(m);
  m = n;
  free(m);
  free(n);
  }
```

1. Introdução teórica

Alocação dinâmica de memória em linguagem C

Em C, a alocação dinâmica de memória é feita, em geral, com a `malloc` (e funções relacionadas, como a `calloc` e a `realloc`), que reservam um bloco no heap e devolvem um ponteiro para o início desse bloco. A liberação desse recurso é feita com a função `free`. Todo bloco alocado dinamicamente deve ter uma liberação correspondente, exatamente uma vez, em algum ponto apropriado do fluxo do programa.

Um vazamento de memória (memory leak) ocorre quando o programa perde a capacidade de acessar (isto é, não há mais ponteiros válidos que levem até) um bloco previamente alocado sem tê-lo liberado. O bloco continua ocupado no heap, mas ficou "órfão". Em programas de longa duração (como os serviços, os daemons, as aplicações que ficam abertas por horas), esse acúmulo pode levar a aumento de consumo de memória e degradação do funcionamento. As causas típicas de memory leak em C incluem as mencionadas a seguir.

- Saídas antecipadas (com `return`, `break`, `goto`) em caminhos de erro antes de executar a função `free`.
- Reatribuição de ponteiros (por exemplo, `p = q;`) antes de liberar o bloco originalmente apontado por `p` (o ponteiro "perde" a referência).
- Protocolos de "dono" do recurso (ownership) mal definidos, em que nenhuma parte do código assume a responsabilidade de liberar.
- Múltiplos pontos de retorno, sem uma seção de limpeza única (cleanup).

É útil também separar memory leak de outros erros comuns com ponteiros, listados a seguir.

- Dupla liberação (double free): chamar `free` duas vezes para o mesmo bloco.
- Uso após liberação (use-after-free): usar um ponteiro depois de liberar o bloco.

- Ponteiro não inicializado: declarar `T *p;` e usar `p` sem atribuir um endereço válido.

Esses erros não são memory leak (embora possam coexistir), mas também causam comportamento indefinido e falhas.

2. Análise das alternativas

A - Alternativa incorreta.

JUSTIFICATIVA. No trecho, há alocação dinâmica com `malloc(50)` e, em seguida, a liberação correspondente com `free(s)`. Assim, o bloco alocado no heap é devidamente liberado antes do término da função. Portanto, não há vazamento de memória (memory leak).

B - Alternativa incorreta.

JUSTIFICATIVA. O código apenas declara um ponteiro (`float *a;`) e retorna `0`. Não ocorre chamada a `malloc`, `calloc` ou `realloc`, isto é, não há alocação dinâmica de memória no trecho. Sem alocação dinâmica, não há como caracterizar memory leak.

C - Alternativa correta.

JUSTIFICATIVA. O ponteiro s recebe memória alocada por `malloc(50)`. Em seguida, se `size > 50`, a função executa `return(-1)`; antes de chegar a `free(s)`. Nesse caminho de execução, o bloco alocado não é liberado, e a referência a ele se perde ao sair da função. Isso caracteriza vazamento de memória (memory leak) por saída antecipada sem limpeza do recurso.

D - Alternativa incorreta.

JUSTIFICATIVA. A função `f` aloca memória e retorna o ponteiro para o chamador. No `main`, o ponteiro retornado é armazenado em `num` e depois liberado com `free(num)`. O ciclo de alocação e liberação está completo no trecho apresentado. Portanto, não há memory leak.

E - Alternativa incorreta.

JUSTIFICATIVA. O problema principal do trecho não é vazamento de memória, e sim liberação incorreta. Após `m = n;`, os dois ponteiros passam a apontar para o mesmo bloco (o segundo bloco alocado). Com isso, `free(m);` libera esse bloco, e `free(n);` tenta liberá-lo novamente, o que configura dupla liberação (double free), um erro grave e distinto de memory

leak. Além disso, o trecho reaproveita o identificador `n` como parâmetro e variável local, o que torna o código inválido em C padrão.

Alternativa correta: C.

3. Indicações bibliográficas

- HARBISON, S. P. III; STEELE JR., G. L. *C: manual de referência*. São Paulo: Ciência Moderna, 2002.
- KERNIGHAN, B.; RITCHIE, D. C. *A linguagem de programação padrão ANSI*. Rio de Janeiro: Campus, 1990.
- KERRISK, M. *The Linux Programming Interface: a Linux and UNIX System Programming Handbook*. San Francisco: No Starch Press, 2010.
- RYANT, R. E.; O'HALLARON, D. R. *Computer Systems: a Programmer's Perspective*. 3. ed. Boston: Pearson, 2015.
- SEACORD, R. C. *Effective C: An Introduction to Professional C Programming*. 2. ed. San Francisco: No Starch Press, 2024.

Questão 5

Questão 5.[5]

Leia o texto a seguir.

Na programação de sistemas embarcados, algumas posições de memória servem para diferentes propósitos, não apenas para armazenar valores. Em algumas dessas memórias, cada um dos bits possui um significado diferente, sendo necessário manipulá-los individualmente ou em pequenos grupos. Por isso, o conhecimento da álgebra booleana, bem como dos operadores utilizados para realizar operações binárias nas linguagens de programação, é essencial para o desenvolvimento desse tipo de sistema.

ALMEIDA, R. M.; MORAES, C. H. V.; SERAPHIM, T. F. P. *Programação de Sistemas Embarcados: desenvolvendo software para microcontroladores em linguagem C*. Rio de Janeiro: Elsevier, 2016 (com adaptações).

A partir dessas informações, observe o código apresentado a seguir, escrito na linguagem C, que faz uso de operações binárias sobre variáveis inteiras.

```
#include <stdio.h>
int main()
{
  int a, b;
  int x, y, z;
  scanf("%d %d", &a, &b);
  x = a; y = b; z = a + b;
  while (a) {
    x = x | b;
    y = y ^ a;
    z = z & (a+b);
    a = a >> 1;
    b = b << 1;
  }
  printf ("%d %d %d\n", x, y, z);
  return 0;
}
```

Após a chamada desse programa, caso o usuário entre com os valores 10 e 1, nessa ordem, qual será, exatamente, o valor da saída do programa?

A. 10 1 0

B. 10 1 11

C. 11 11 11

D. 15 12 2

E. 15 13 0

[5]Questão 16 – Enade 2023.

1. Introdução teórica

Memória ou registradores em sistemas embarcados

Em sistemas embarcados, é comum que áreas de memória ou registradores representem campos de bits (flags, estados, modos de operação). Nesses casos, o programador precisa ler, testar, ligar, desligar ou alternar bits específicos, em vez de tratar o dado apenas como um número convencional. Em C, isso é feito principalmente com operações bit a bit e deslocamentos.

1.1. Operadores bit a bit (em inteiros)

As operações bit a bit constituem um mecanismo fundamental para a manipulação direta da representação binária de valores inteiros em C. Diferentemente das operações aritméticas usuais, que tratam os operandos como magnitudes numéricas, os operadores bit a bit atuam de forma independente sobre cada posição binária dos operandos. Assim, ao considerar dois inteiros `p` e `q`, operadores como OR (`p | q`), AND (`p & q`) e XOR (`p ^ q`) produzem um resultado cuja formação depende da comparação entre os bits correspondentes de `p` e `q`, posição por posição. O quadro 1, apresentado a seguir, sintetiza essas regras de funcionamento e serve como referência para a leitura operacional dessas expressões.

Quadro 1. Operações bit a bit.

Operação	Símbolo	Regra em cada bit	Uso típico
OR	p | q	1 se pelo menos um dos bits for 1.	Ligar bits usando uma máscara.
AND	p & q	1 se ambos os bits forem 1.	Filtrar bits (manter só os bits de interesse) ou zerar bits via máscara.
XOR	p ^ q	1 se os bits forem diferentes.	Alternar (toggle) bits via máscara.

No contexto de sistemas embarcados, essas operações são particularmente relevantes porque os registradores de hardware costumam armazenar as informações em campos de bits, nos quais cada posição pode representar um estado, uma configuração ou uma sinalização específica. Nessa situação, o operador OR é empregado, em geral, para ativar bits por meio de máscaras, preservando os demais bits do registrador. O operador AND, por sua vez, é amplamente utilizado para filtrar os bits de interesse (isto é, manter apenas

determinadas posições) e, também, para zerar alguns bits específicos, quando combinado com as máscaras apropriadas. Já o operador XOR é útil para alternar o estado lógico de bits selecionados, operação conhecida como toggle, também realizada por meio de máscaras.

Desse modo, o domínio desses operadores é indispensável para a programação de baixo nível, na qual a precisão sobre cada bit tem impacto direto no comportamento do sistema.

1.2. Deslocamentos (shifts)

Os operadores de deslocamento (shifts) complementam as operações bit a bit ao permitirem o reposicionamento sistemático dos bits de um valor inteiro. Em C, o deslocamento à esquerda (`p << k`) move os bits de `p` em `k` posições para a esquerda, enquanto o deslocamento à direita (`p >> k`) move esses bits em `k` posições para a direita. Sob condições específicas - especialmente para os valores não negativos e sem ocorrência de estouro - o deslocamento à esquerda pode ser interpretado como uma multiplicação por 2^k, ao passo que o deslocamento à direita, para valores não negativos, corresponde a uma divisão inteira por 2^k. Essa interpretação é útil para a análise conceitual e para o raciocínio sobre eficiência, embora a operação, no nível da linguagem, continue sendo definida em termos de manipulação de bits.

Em sistemas embarcados, os deslocamentos têm papel central na montagem e na extração de campos dentro de registradores. É comum, por exemplo, deslocar um valor para posicioná-lo em um intervalo específico de bits (como o campo correspondente aos bits 4 a 7) antes de combiná-lo com o conteúdo de um registrador por meio de máscaras e operadores bit a bit. Da mesma forma, deslocamentos sucessivos são empregados em rotinas de varredura de bits, nas quais cada posição é examinada iterativamente.

Em termos de boas práticas, convém destacar que, em manipulação de registradores, costuma-se preferir tipos inteiros sem sinal, pois isso reduz ambiguidades de interpretação associadas ao deslocamento de valores negativos e torna o comportamento mais previsível no nível binário.

2. Análise da questão

A análise da execução pode ser feita por simulação direta do programa para a entrada 10 1. Inicialmente, o código lê os valores de `a` e `b` e, em seguida, atribui `x = a, y = b e z = a + b`. Portanto, no estado inicial, tem-se `a = 10` (isto é, `1010` em binário) e `b = 1` (`0001` em binário). Consequentemente, `x = 10, y = 1 e z = 11`.

A partir desse ponto, o laço `while (a)` permanece em execução enquanto `a` for diferente de zero. Em cada iteração, o programa atualiza `x` por meio da operação OR com `b`, atualiza `y` por meio da operação XOR com `a`, atualiza `z` por meio da operação AND com a soma corrente `a + b`, e então desloca `a` uma posição à direita e `b` uma posição à esquerda.

Na primeira iteração, com `a = 10` e `b = 1`, calcula-se `x = x | b = 10 | 1`. Em binário, isso corresponde a `1010 | 0001 = 1011`, resultando em `x = 11`. Em seguida, `y = y ^ a = 1 ^ 10`, isto é, `0001 ^ 1010 = 1011`, de modo que `y = 11`. Para `z`, o cálculo é `z = z & (a + b) = 11 & (10 + 1) = 11 & 11`, permanecendo `z = 11`. Ao final da iteração, o programa realiza os deslocamentos: `a = 10 >> 1 = 5` e `b = 1 << 1 = 2`.

Na segunda iteração, agora com `a = 5` e `b = 2`, obtém-se `x = 11 | 2`. Em binário, `1011 | 0010 = 1011`, portanto `x` permanece igual a `11`. Em seguida, `y = 11 ^ 5`, isto é, `1011 ^ 0101 = 1110`, resultando em `y = 14`. Para `z`, tem-se `z = 11 & (5 + 2) = 11 & 7`; em binário, `1011 & 0111 = 0011`, logo `z = 3`. Após isso, os deslocamentos produzem `a = 5 >> 1 = 2` e `b = 2 << 1 = 4`.

Na terceira iteração, com `a = 2` e `b = 4`, o programa calcula `x = 11 | 4`. Em binário, `1011 | 0100 = 1111`, o que leva a `x = 15`. Em seguida, `y = 14 ^ 2`, isto é, `1110 ^ 0010 = 1100`, de modo que `y = 12`. Para `z`, obtém-se `z = 3 & (2 + 4) = 3 & 6`; em binário, `0011 & 0110 = 0010`, logo `z = 2`. Ao término dessa etapa, os deslocamentos atualizam as variáveis para `a = 2 >> 1 = 1` e `b = 4 << 1 = 8`.

Na quarta iteração, com `a = 1` e `b = 8`, tem-se `x = 15 | 8`. Em binário, `1111 | 1000 = 1111`, de forma que `x` permanece `15`. Depois, `y = 12 ^ 1`, isto é, `1100 ^ 0001 = 1101`, resultando em `y = 13`. Para `z`, o cálculo é `z = 2 & (1 + 8) = 2 & 9`; em binário, `0010 & 1001 = 0000`, o que produz `z = 0`. Por fim, os deslocamentos dão `a = 1 >> 1 = 0` e `b = 8 << 1 = 16`.

Como `a` passa a ser zero ao final da quarta iteração, a condição do `while` deixa de ser satisfeita e o laço é encerrado. Assim, no momento do `printf`, os valores finais são `x = 15`, `y = 13` e `z = 0`. A saída exata do programa é, portanto, `15 13 0`.

Alternativa correta: E.

3. Indicações bibliográficas

- CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 4. ed. Rio de Janeiro: GEN LTC, 2024.
- DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. *Algoritmos.* Tradução de Guilherme Albuquerque Pinto. AMGH, 2009.
- HARBISON, S. P. III; STEELE JR., G. L. *C: manual de referência*. 1. ed. [S.l.]. Rio de Janeiro: Ciência Moderna, 2002.
- KERNIGHAN, B.; RITCHIE, D. C. *A linguagem de programação padrão ANSI*. Rio de Janeiro: Campus, 1990.
- SEACORD, R. C. *Effective C: an Introduction to Professional C Programming*. 2. ed. San Francisco: No Starch Press, 2024.

Questão 6

Questão 6.[6]

Computação em nuvem representa um conceito relativo ao compartilhamento de recursos, tais como capacidade de processamento, armazenamento, comunicação de dados e pessoal qualificado para manter sistemas computacionais disponíveis na internet. Quando esse compartilhamento constitui um serviço disponível para qualquer pessoa, o serviço é conhecido como nuvem pública. Quando as mesmas tecnologias são empregadas para uma única empresa, não permitindo que terceiros utilizem parte dos recursos, temos uma nuvem privada. Considerando as vantagens que o gerente de uma empresa espera obter na contratação de um serviço de nuvem pública, avalie as afirmativas.

I. Um serviço de nuvem pública proporciona a redução de custos operacionais, pois é possível dimensionar a necessidade desses recursos com base na demanda, visto que há momentos em que a empresa precisa de mais recursos e há momentos em que ela precisa de menos recursos.

II. Um serviço de nuvem pública gera aumento da velocidade de execução do software e de acesso ao software a partir de qualquer localidade, pois a nuvem pública garante acesso rápido em qualquer parte do mundo, o que contrasta com um servidor localizado na cidade da empresa.

III. Um serviço de nuvem pública viabiliza a redução do investimento inicial com equipamentos, com infraestrutura e com pessoal para iniciar a operação, visto que torna possível adiar a instalação desses recursos na empresa até que a operação se demonstre economicamente viável.

IV. Um serviço de nuvem pública propicia aumento de segurança da informação, pois as rotinas de segurança são responsabilidade do administrador da nuvem pública, não do contratante, o que contrasta com um servidor localizado dentro da empresa.

É correto apenas o que se afirma em

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

[6]Questão 20 – Enade 2023.

1. Introdução teórica

Computação em nuvem

A computação em nuvem organiza os recursos de TI (a saber, o processamento, o armazenamento, as redes e os softwares) como serviços acessados por rede, com o provisionamento sob demanda e o crescimento ou a redução de capacidade conforme a necessidade. Uma formulação recorrente na literatura descreve a nuvem por atributos como a elasticidade, a cobrança pelo uso (pay as you go) e o autoatendimento para provisionar os recursos.

Na nuvem pública, a infraestrutura pertence ao provedor e é compartilhada entre múltiplos clientes (com isolamento lógico). O valor gerencial costuma aparecer em três frentes abaixo descritas.

- Economia e investimento: substitui-se parte do investimento inicial (CAPEX) por despesa operacional (OPEX), com o pagamento conforme o consumo e sem a exigência da compra imediata de servidores para iniciar o serviço.
- Escalabilidade e elasticidade: a capacidade pode aumentar ou diminuir conforme o regime de picos e de vales de demanda, evitando o superdimensionamento permanente.
- Acesso e desempenho percebido: a nuvem amplia o acesso remoto, mas o desempenho não é necessariamente mais rápido porque depende de rede, de latência e de outros fatores. Aplicações web, por exemplo, podem ser mais lentas do que alternativas locais em certos cenários.

Em segurança, um ponto técnico central é que, em nuvem, a proteção não migra inteira para o provedor: há divisão de responsabilidades entre o provedor e o cliente, que varia de acordo com o modelo adotado - IaaS, PaaS ou SaaS - e as configurações do contratante).

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. A afirmativa descreve a elasticidade da computação em nuvem pública: a empresa pode aumentar ou reduzir recursos conforme a demanda. Isso tende a reduzir os desperdícios e os custos operacionais em comparação com a manutenção de infraestrutura superdimensionada o tempo todo.

II – Afirmativa incorreta.
JUSTIFICATIVA. A nuvem pública amplia o acesso remoto ao software, porém não garante, por si só, o aumento da velocidade de execução nem o acesso rápido em qualquer localidade. O desempenho depende de fatores como a latência de rede, a qualidade da conexão, a distância até a região do provedor e a arquitetura da aplicação.

III – Afirmativa correta.
JUSTIFICATIVA. A afirmativa está de acordo com uma vantagem clássica da nuvem pública: a redução do investimento inicial em equipamentos e na infraestrutura própria. Em vez de comprar os servidores e instalar toda a estrutura antes de operar, a empresa pode contratar os recursos sob demanda e expandi-los conforme sua necessidade.

IV – Afirmativa incorreta.
JUSTIFICATIVA. A segurança em nuvem pública não é de responsabilidade exclusiva do provedor. Há responsabilidade compartilhada entre o provedor e o contratante (cliente), que varia conforme o modelo de serviço contratado (IaaS, PaaS ou SaaS). Portanto, a justificativa apresentada na afirmativa está incorreta.

Alternativa correta: B.

3. Indicações bibliográficas

- BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. *Cloud Computing: principles and Paradigms*. Hoboken: Wiley, 2011.
- DOTSON, C. *Practical Cloud Security: a Guide for Secure Design and Deployment*. 2. ed. Sebastopol: O'Reilly Media, 2023.
- ERL, T.; MONROY, E. B. *Computação em nuvem: conceitos, tecnologia, segurança e arquitetura*. 2. ed. Porto Alegre: Bookman, 2024.
- MARINESCU, D. C. *Cloud Computing: Theory and Practice*. 3. ed. Amsterdam: Morgan Kaufmann, 2022.
- VELTE, A. T.; VELTE, T. J.; ELSENPETER, R. C. *Cloud computing, computação em nuvem: uma abordagem prática*. Rio de Janeiro: Alta Books, 2012.

Questão 7

Questão 7.[7]

A fim de melhorar a qualidade de vida de seus cidadãos e de criar eficiência nos serviços e nas operações urbanas, um grupo de vereadores de uma pequena cidade decidiu fazer algumas propostas de lei que, se aprovadas e devidamente implementadas, tendem a aproximar a cidade do conceito de "cidade inteligente" por meio da implementação de novos sistemas de software. O grupo de vereadores, preocupado com acessibilidade, consultou especialistas da área de interação humano-computador (IHC) e levantou informações a respeito de fundamentos de acessibilidade em IHC. Entre eles, estão os mencionados a seguir.

Fundamento 1. Um produto ou serviço é equitativo quando é projetado de modo que possa atender a todos os usuários, independentemente de suas habilidades físicas, sensoriais ou cognitivas.

Fundamento 2. Um produto ou serviço deve ter informações perceptíveis, o que pode envolver o uso de recursos como, por exemplo, texto alternativo para imagens ou contrastes de cores suficientes para usuários com deficiências visuais ou com daltonismo.

Considerando esse contexto, avalie as afirmativas.

I. Uma lei que prevê que semáforos devem exibir ícones universais associados a cada uma de suas cores está relacionada ao Fundamento 1.

II. Ainda que esteja relacionado à acessibilidade em IHC, o Fundamento 1 deixa de tratar da inclusão de pessoas cegas.

III. Dado que a cidade possui uma página na web para a obtenção de informações, uma lei que preveja a existência de telas digitais públicas que permitam o acesso às informações disponibilizadas é suficiente para caracterizar a aplicação do Fundamento 2.

É correto o que se afirma em

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

[7]Questão 21 – Enade 2023.

1. Introdução teórica

Interação humano-computador

Na interação humano-computador (IHC), a acessibilidade é um tema central no projeto de sistemas digitais, de interfaces públicas e de serviços mediados por tecnologia. Em termos conceituais, ela diz respeito às condições que permitem que pessoas com diferentes características sensoriais, físicas e cognitivas percebam, compreendam e utilizem um artefato digital com autonomia e com segurança. Esse campo não se limita a adaptações para grupos específicos; ele envolve uma lógica de projeto que busca reduzir as barreiras desde a concepção da interface, aproximando-se das ideias de desenho universal e de design inclusivo. Nessa perspectiva, a qualidade de um sistema é avaliada também por sua capacidade de atender à diversidade humana real, e não por um usuário médio abstrato.

Em IHC, é importante distinguir a existência de um recurso tecnológico da efetiva acessibilidade desse recurso. A presença de uma tela, de um site, de um painel eletrônico ou de um aplicativo pode ampliar o alcance de informações, mas isso não significa, por si, que o conteúdo esteja acessível. A análise depende de como a informação é apresentada, de quais canais perceptivos são exigidos e de quais obstáculos foram removidos ou mantidos. Uma interface pode estar disponível e, ainda assim, excluir as pessoas por depender somente de uma certa cor, por ter o contraste insuficiente, por utilizar uma tipografia pouco legível, por não oferecer as alternativas textuais ou as sonoras, ou por exigir algumas interações incompatíveis com as tecnologias assistivas.

Dois conhecimentos são especialmente relevantes para esse tipo de avaliação. O primeiro é o princípio da equidade, ligado ao desenho de produtos e de serviços que possam ser usados por pessoas com perfis diversos, sem segregação e sem perda de funcionalidade para determinados grupos. A equidade, nesse contexto, implica projetar para a pluralidade de usuários, incluindo as pessoas cegas, as pessoas com baixa visão, as pessoas com daltonismo, as pessoas com limitações motoras e as pessoas com diferentes perfis cognitivos. O foco está em evitar que uma decisão de projeto transforme uma característica humana em barreira de acesso.

O segundo conhecimento é o da perceptibilidade da informação. Uma mensagem deve ser apresentada de modo perceptível para diferentes usuários, o que costuma exigir a redundância de sinais e o cuidado com a forma de representação. Em vez de depender de um único código perceptivo, como a cor, o projeto acessível combina os recursos complementares,

como o texto, os ícones, os símbolos, o contraste adequado, os padrões visuais, o feedback sonoro e outros mecanismos compatíveis com o contexto de uso. Essa redundância tem uma função cognitiva e sensorial: ela melhora a compreensão geral e reduz a exclusão de quem não percebe um dos canais empregados.

Também é necessário compreender que a acessibilidade e a usabilidade são conceitos relacionados, mas distintos. Um sistema pode ser relativamente fácil de usar para quem consegue acessá-lo e, ainda assim, ser inacessível para outros usuários. A avaliação consistente exige observar se a interface permite o acesso inicial à informação, se a informação pode ser percebida com clareza e se as ações exigidas podem ser realizadas por diferentes pessoas. Em ambientes urbanos e em serviços públicos digitais, essa análise ganha relevância adicional, porque o uso é heterogêneo, porque ocorre em condições variadas (a presença de ruído, a iluminação diversa, a pressa na interação, a circulação intensa) e envolve direitos de acesso à informação e aos serviços.

2. Análise das afirmativas

I – Afirmativa correta.
JUSTIFICATIVA. A associação de ícones universais às cores dos semáforos amplia o uso por pessoas com diferentes condições sensoriais, especialmente por pessoas com daltonismo, e se relaciona ao fundamento da equidade, pois favorece o atendimento de um conjunto mais amplo de usuários.

II – Afirmativa incorreta.
JUSTIFICATIVA. O fundamento da equidade abrange usuários com diferentes habilidades sejam físicas, sensoriais e até cognitivas. Portanto, ele inclui, sim, as pessoas cegas; a afirmativa exclui indevidamente esse grupo.

III – Afirmativa incorreta.
JUSTIFICATIVA. A mera existência de telas digitais públicas para acesso às informações não é suficiente para caracterizar a aplicação do fundamento das informações perceptíveis. Para isso, é necessário que a apresentação da informação seja acessível (por exemplo, com o contraste adequado, a legibilidade, a redundância de sinais e as alternativas para pessoas com deficiência visual).

Alternativa correta: A.

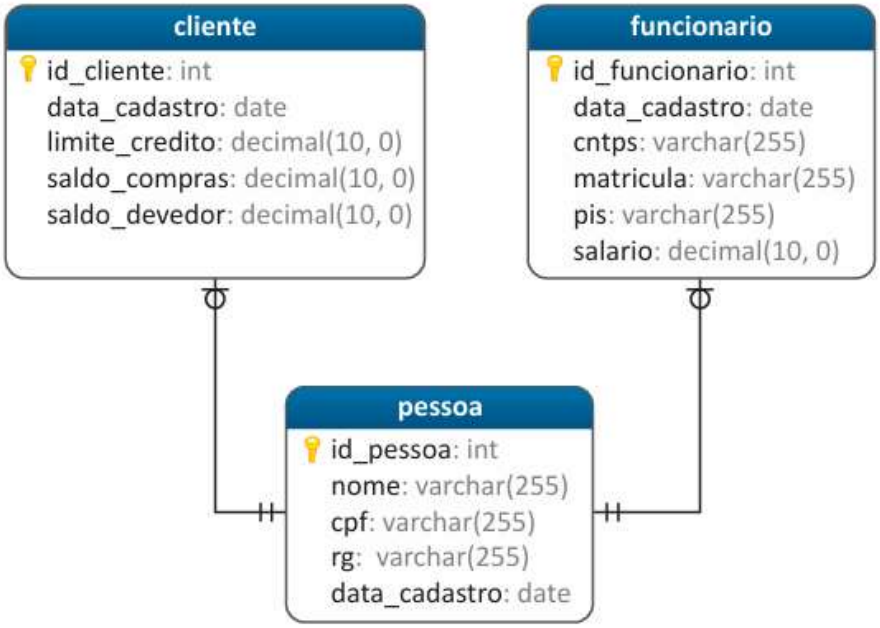
3. Indicações bibliográficas

- BARBOSA, S. D. J.; SILVA, B. S. *Interação humano-computador*. Rio de Janeiro: Elsevier, 2010.
- NIELSEN, J. *Usability Engineering*. Morgan Kaufmann, 1994.
- PREISER, W. F. E.; SMITH, K. H. *Universal Design Handbook*. 2. ed. McGraw-Hill, 2010.
- ROGERS, Y.; SHARP, H.; PREECE, J. *Design de interação: além da interação humano-computador*. 3. ed. Porto Alegre: Bookman, 2013.

Questão 8

Questão 8.[8]

Considere um banco de dados relacional formado por três tabelas, conforme é apresentado na figura a seguir. As chaves primárias das tabelas `cliente` e `funcionario` são chaves estrangeiras da tabela `pessoa`.



A partir dessas informações, considere que se queira realizar uma consulta que liste o nome e o saldo devedor de um subconjunto dos clientes. Essa consulta tem por objetivo encontrar clientes que são funcionários e que possuem saldo devedor maior do que seu salário.

Com base nessas informações, assinale a opção que apresenta corretamente a consulta SQL, em ordem crescente por saldo devedor.

A. `SELECT * FROM cliente as c INNER JOIN pessoa as p, funcionario as f WHERE c.saldo_devedor > f.salario AND c.id_cliente=p.id_pessoa AND f.id_funcionario=p.id_pessoa ORDER BY c.saldo_devedor ASC`

B. `SELECT p.nome, c.saldo_devedor FROM cliente as c, pessoa as p WHERE c.saldo_devedor > f.salario AND c.id_cliente=p.id_pessoa AND f.id_funcionario=p.id_pessoa ORDER BY c.saldo_devedor DESC`

C. `SELECT p.nome, c.saldo_devedor FROM cliente as c, pessoa as p, funcionario as f WHERE c.saldo_devedor < f.salario AND c.id_cliente=p.id_pessoa AND f.id_funcionario=p.id_pessoa ORDER BY c.saldo_devedor ASC`

[8]Questão 22 – Enade 2023.

D. `SELECT p.nome, c.saldo_devedor FROM cliente as c LEFT OUTER JOIN pessoa as p on c.id_cliente=p.id_pessoa LEFT OUTER JOIN funcionario as f on p.id_pessoa=f.id_funcionario WHERE c.saldo_devedor > f.salario ORDER BY f.salario, c.saldo_devedor ASC`

E. `SELECT p.nome, c.saldo_devedor FROM cliente as c RIGHT OUTER JOIN pessoa as p ON c.id_cliente=p.id_pessoa RIGHT OUTER JOIN funcionario as f on p.id_pessoa=f.id_funcionario WHERE c.saldo_devedor > f.salario ORDER BY c.saldo_devedor ASC`

1. Introdução teórica

Linguagem SQL

O modelo relacional organiza os dados em relações (ou tabelas), nas quais cada linha representa uma ocorrência de um fato e cada coluna representa um atributo desse fato. A utilidade desse modelo, do ponto de vista de consulta, está na separação entre a estrutura lógica e a operação de recuperação: os dados podem ser distribuídos em tabelas distintas por razões de integridade, redução de redundância e clareza semântica, e depois podem ser recompostos por consultas. Essa recomposição é realizada por operações de junção, fundamentadas em correspondências entre os atributos que representam a mesma identidade lógica em tabelas diferentes. Por isso, a compreensão de chaves primárias e de chaves estrangeiras é central: a chave primária identifica univocamente uma linha, enquanto a chave estrangeira preserva os vínculos entre as relações e permite reconstruir, por consulta, uma visão integrada dos dados. Considere essa distinção na ilustração a seguir.

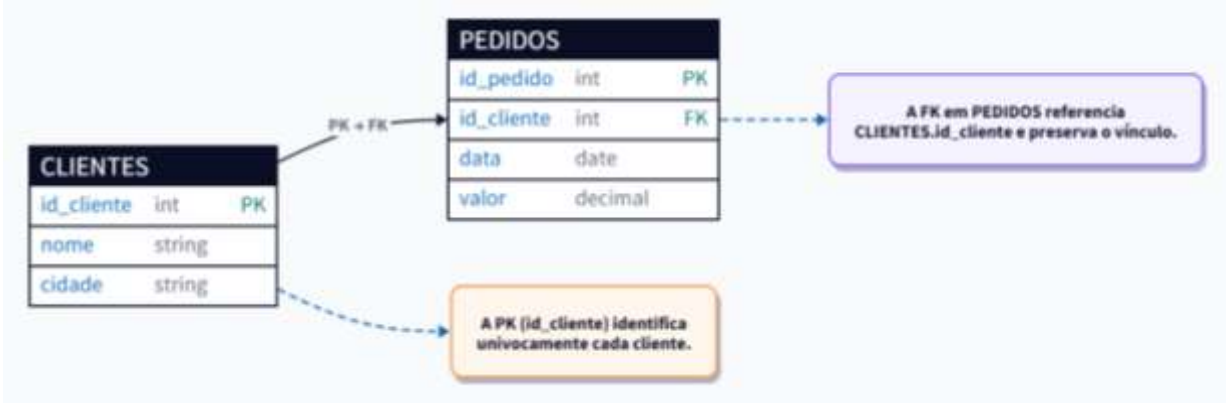


Figura 1. Chaves primárias e chaves secundárias.

Em muitos esquemas relacionais, uma entidade genérica é especializada em papéis diferentes, cada qual armazenado em tabela própria. Nesse tipo de modelagem, uma mesma

identidade pode aparecer em mais de uma tabela especializada, e o vínculo entre essas tabelas costuma ocorrer por meio de uma chave que referencia a tabela-base de identidade. Isso exige atenção ao interpretar o significado da consulta: às vezes, a tarefa não é comparar registros de pessoas distintas, e sim verificar se a mesma pessoa acumula papéis diferentes no sistema. A consulta correta, nesse contexto, depende de reconhecer que o predicado de junção não serve apenas para ligar tabelas, mas para expressar uma condição semântica precisa sobre identidade.

A linguagem SQL (Structured Query Language) materializa essa lógica declarativa por meio da cláusula `SELECT` (projeção dos atributos desejados), da cláusula `FROM` (fontes de dados e junções), da cláusula `WHERE` (restrições e filtros) e da cláusula `ORDER BY` (ordenação da saída). Embora existam formas diferentes de escrever junções - sintaxe explícita com `JOIN ... ON` e sintaxe antiga por lista de tabelas com condições no `WHERE` -, o ponto decisivo é preservar a correção semântica e evitar ambiguidades. O uso de apelidos (aliases) é especialmente importante quando há múltiplas tabelas na consulta, pois torna os predicados legíveis e impede confusão entre colunas homônimas ou entre papéis distintos na mesma expressão.

A junção interna (`INNER JOIN`) retorna somente as combinações com correspondência entre as tabelas envolvidas. Já as junções externas (`LEFT`, `RIGHT`, `FULL`) preservam linhas de um dos lados mesmo quando não há correspondência, introduzindo os valores nulos nas colunas ausentes. Essa diferença tem efeito direto no resultado quando se aplica um filtro comparativo no `WHERE`, porque as comparações com `NULL` não produzem "verdadeiro" no sentido clássico e podem eliminar linhas que a junção externa havia preservado. Em termos práticos, isso significa que uma junção externa pode se comportar como uma junção interna depois de certo filtro, mas sem que isso implique equivalência completa de intenção ou de escrita. A figura a seguir ilustra o conceito de `JOIN`.

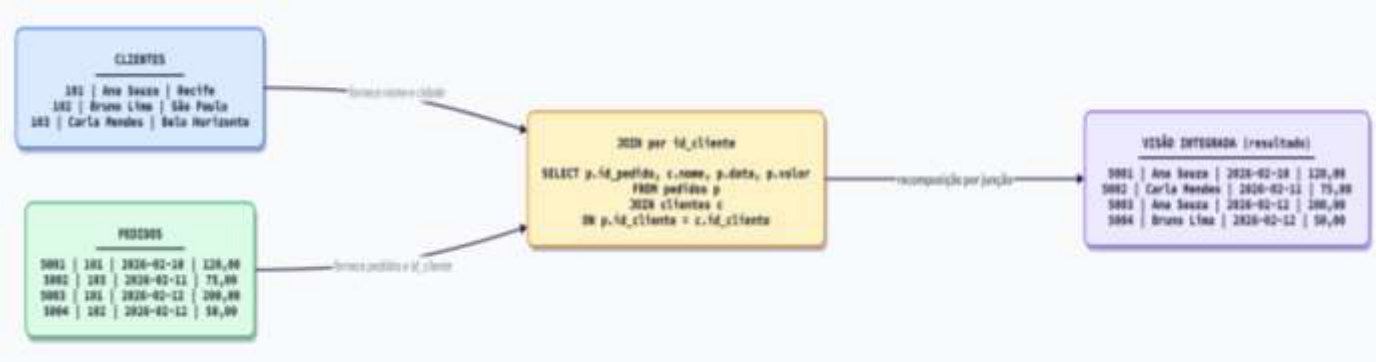


Figura 2. Conceito de `JOIN`.

Em SQL, a condição relacional expressa no `WHERE` define quais tuplas satisfazem o predicado; trocar um operador como `>` por `<` altera completamente o conjunto retornado, ainda que o restante da consulta permaneça formalmente válido. Do mesmo modo, a ordenação precisa refletir com exatidão o critério solicitado: uma ordenação crescente por uma coluna não é equivalente a uma ordenação decrescente, nem a uma ordenação composta em que outra coluna venha antes, pois isso muda a prioridade do arranjo dos resultados.

2. Análise das alternativas

A - Alternativa incorreta.

JUSTIFICATIVA. A alternativa usa `INNER JOIN` sem a cláusula `ON` (ou `USING`) logo após a junção com `pessoa`, o que torna a instrução sintaticamente inadequada no padrão SQL. Além disso, a projeção pedida é o `nome` e o `saldo devedor`, e a consulta usa `SELECT *`, retornando todas as colunas das tabelas envolvidas. Mesmo que a lógica do filtro e da ordenação esteja próxima do objetivo, a forma escrita da consulta não atende corretamente ao que foi solicitado.

B – Alternativa incorreta.

JUSTIFICATIVA. A alternativa referencia `f.salario` e `f.id_funcionario` no `WHERE`, mas a tabela `funcionario` (alias `f`) não foi incluída na cláusula `FROM`. Isso torna a consulta inválida, pois o alias `f` não existe no escopo da instrução. Além disso, a ordenação está em ordem decrescente (`DESC`), enquanto o critério pedido é crescente por saldo devedor.

C – Alternativa incorreta.

JUSTIFICATIVA. A estrutura geral da consulta é válida e inclui as três tabelas com os aliases necessários, porém o predicado de comparação está invertido: usa `c.saldo_devedor < f.salario`, quando o requisito é selecionar os casos em que o saldo devedor é maior do que o salário. Assim, a consulta retorna o conjunto oposto ao desejado para essa comparação.

D – Alternativa incorreta.

JUSTIFICATIVA. A alternativa consegue relacionar as tabelas e aplica a comparação `c.saldo_devedor > f.salario`, mas a ordenação não corresponde ao enunciado. A cláusula `ORDER BY f.salario, c.saldo_devedor ASC` ordena primeiro pelo salário e

só depois pelo `saldo devedor`. O pedido é ordenar em ordem crescente por saldo devedor, isto é, com prioridade de ordenação em `c.saldo_devedor`. Esse detalhe altera a sequência dos resultados e invalida a alternativa.

E – Alternativa correta.

JUSTIFICATIVA. A alternativa projeta exatamente os atributos pedidos (`p.nome` e `c.saldo_devedor`), relaciona as tabelas `cliente`, `pessoa` e `funcionario`, aplica corretamente o filtro `c.saldo_devedor > f.salario` e finaliza com `ORDER BY c.saldo_devedor ASC`, que corresponde à ordenação crescente por saldo devedor. Embora utilize `RIGHT OUTER JOIN` (uma escrita menos usual para esse caso), o filtro no `WHERE` elimina linhas sem correspondência em cliente, preservando o conjunto que atende ao objetivo da consulta.

3. Indicações bibliográficas

- CONNOLLY, T.; BEGG, C. *Database Systems: a Practical Approach to Design, Implementation, and Management*. 6. ed. Pearson, 2021.
- DATE, C. J. *SQL e teoria relacional: como escrever códigos SQL precisos*. São Paulo: Novatec, 2015.
- ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 7. ed. São Paulo: Pearson, 2018.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de banco de dados.* 7. ed. Rio de Janeiro: GEN LTC, 2020.

Questão 9

Questão 9.[9]

Considere um cenário em que um computador seja organizado com múltiplos processadores, os quais compartilham a mesma memória RAM. Cada processador possui múltiplos núcleos. Nesse arranjo, o sistema operacional permite múltiplas threads, as quais podem ser dinamicamente alocadas para execução em diferentes núcleos e processadores. A partir das informações apresentadas nessa situação, assinale a opção correta.

A. Sistemas com múltiplos processadores devem alocar a mesma quantidade de memória RAM para cada processador do arranjo.
B. Como há múltiplos processadores, são desnecessários os semáforos, uma vez que não há acessos concorrentes a recursos compartilhados.
C. Como a exclusão mútua não é possível em arquitetura de múltiplos processadores, apenas uma aplicação pode ser executada de cada vez, mas com múltiplas threads.
D. Os processos que possuem múltiplas threads em execução são mantidos por meio de funções da biblioteca no código da aplicação e dispensam serviços do sistema operacional.
E. Dados trocados durante a comunicação entre processos podem ser armazenados nas áreas de memória compartilhada, mas o acesso a essas áreas é intermediado pelo sistema operacional.

1. Introdução teórica

Desempenho em sistemas computacionais

Em sistemas computacionais contemporâneos, o desempenho não depende apenas da frequência de um processador, mas da capacidade de executar múltiplas atividades de forma concorrente em arquiteturas com vários núcleos e, em certos arranjos, com múltiplos processadores físicos. Nesses ambientes, o sistema operacional organiza o uso da CPU, da memória e dos dispositivos de entrada e saída, distribuindo o trabalho entre unidades de execução e controlando a competição por recursos compartilhados. Para compreender esse funcionamento, é indispensável distinguir os conceitos de processo e de thread, entender os mecanismos de escalonamento, conhecer os problemas de sincronização e reconhecer o papel do sistema operacional na comunicação entre processos.

[9]Questão 26 – Enade 2023.

1.1. Threads

Um processo é a instância de um programa em execução, com espaço de endereçamento próprio, com recursos associados e com contexto de execução. As threads são fluxos de execução dentro de um processo; elas compartilham o mesmo espaço de memória do processo e vários de seus recursos, mas têm um contexto de execução próprio, como o contador de programa e seus registradores. Essa distinção é central para a análise de concorrência, porque os processos e as threads podem executar simultaneamente em núcleos diferentes, produzindo ganhos de desempenho em tarefas paralelizáveis, ao mesmo tempo em que introduzem os riscos de inconsistência quando acessam os dados compartilhados sem a coordenação adequada.

O escalonamento dessas unidades de execução é uma função clássica do sistema operacional. Em arquiteturas com múltiplos núcleos, o kernel decide quais threads ou quais processos serão executados em cada núcleo, levando em conta as políticas de justiça, de prioridade, de responsividade e do uso eficiente da CPU. A mobilidade de threads entre os núcleos, comum em sistemas modernos, exige que a análise de concorrência não seja feita como se houvesse uma correspondência fixa entre uma thread e um processador. A execução é dinâmica, e a simultaneidade efetiva de acessos à memória compartilhada é uma consequência direta dessa organização.

Quando duas ou mais threads acessam a mesma estrutura de dados, e ao menos uma delas realiza a escrita, surge a necessidade de sincronização. Sem a sincronização, ocorre a condição de corrida, em que o resultado final passa a depender da ordem temporal das operações, produzindo comportamentos erráticos e difíceis de reproduzir. A exclusão mútua é o princípio que permite garantir que apenas uma thread por vez execute uma região crítica, preservando a consistência dos dados. Esse princípio é implementado por mecanismos como os mutexes, os semáforos, os monitores, as variáveis de condição e as operações atômicas. Em arquiteturas multiprocessadas, esses mecanismos permanecem essenciais, porque a execução paralela real amplia a chance de concorrência sobre os mesmos recursos.

Os semáforos podem ser usados tanto para a exclusão mútua quanto para a sincronização entre atividades, coordenando as ordens de execução e a disponibilidade de recursos. Seu uso correto exige disciplina de programação, pois falhas na aquisição e na liberação podem causar os deadlocks, a starvation ou a perda de desempenho por contenção excessiva. A presença de múltiplos processadores não elimina essas necessidades; ao contrário, torna mais evidente a importância de protocolos de sincronização bem definidos.

1.2. Comunicação entre processos

Embora processos tenham, em regra, espaços de endereçamento separados, o sistema operacional fornece os mecanismos para troca de dados, como os pipes, as filas de mensagens, os sockets, os sinais e a memória compartilhada.

A memória compartilhada é um dos mecanismos mais eficientes para a IPC (Interprocess Communication) local porque evita cópias repetidas de dados entre os espaços de usuário e de kernel após o estabelecimento da região compartilhada. O papel do sistema operacional é de criar, de mapear e de proteger essa região, além de controlar as permissões. Depois de mapeada, os processos acessam a área compartilhada diretamente, e a sincronização entre eles continua necessária para impedir as leituras e as escritas inconsistentes. A eficiência e a segurança caminham juntas: o ganho de desempenho da memória compartilhada exige mais cuidado com a coordenação de acessos.

Bibliotecas de threads fornecem as APIs (Interfaces de Programação de Aplicações) para a criação, a sincronização e o gerenciamento de threads, simplificando a programação concorrente. Mesmo assim, elas não tornam dispensáveis os serviços do sistema operacional. O kernel continua responsável por aspectos fundamentais, como o escalonamento, o gerenciamento de memória, a proteção, o tratamento de interrupções e o suporte às primitivas de sincronização e da IPC.

Em muitos sistemas, as threads utilizadas pelas aplicações são threads em nível de kernel, diretamente visíveis ao escalonador do sistema operacional. Em outros modelos, podem existir threads em nível de usuário, mas, ainda assim, a execução concreta depende da infraestrutura oferecida pelo sistema.

2. Análise das alternativas

A - Alternativa incorreta.

JUSTIFICATIVA. Em arquiteturas com os múltiplos processadores e com a memória compartilhada, não existe a exigência de que cada processador tenha "a mesma quantidade de memória RAM alocada". A memória principal pode ser compartilhada por todos os processadores (como em arranjos SMP/UMA, no tratamento didático clássico), e a distribuição lógica de uso da RAM não é feita por uma regra fixa de igualdade por processador. A afirmação descreve uma obrigação inexistente na teoria de sistemas operacionais.

B – Alternativa incorreta.

JUSTIFICATIVA. A existência de múltiplos processadores ou de múltiplos núcleos não elimina os acessos concorrentes a recursos compartilhados; ela aumenta a possibilidade de acessos simultâneos. Sempre que os processos ou as threads compartilham dados, pode ocorrer uma condição de corrida se não houver a coordenação adequada. Os semáforos e outros mecanismos de sincronização continuam necessários para proteger as regiões críticas e para ordenar o acesso aos recursos compartilhados.

C – Alternativa incorreta.

JUSTIFICATIVA. A exclusão mútua é possível em arquiteturas multiprocessadas e constitui um dos fundamentos da programação concorrente. Ela é implementada por mecanismos de sincronização, como os mutexes, os semáforos e as operações atômicas, com suporte do sistema operacional e do hardware. Também é incorreta a afirmação de que apenas uma aplicação pode ser executada por vez porque os sistemas multiprocessados são projetados justamente para permitir execução concorrente de múltiplos processos e threads.

D – Alternativa incorreta.

JUSTIFICATIVA. As bibliotecas de threads fornecem funções para a criação e o controle de threads no código da aplicação, mas isso não dispensa os serviços do sistema operacional. O sistema operacional permanece responsável pelo escalonamento, pela gerência de memória, pela proteção entre processos, pelas chamadas de sistema e pelo suporte aos mecanismos de sincronização e de comunicação entre processos. A alternativa generaliza de modo incorreto o papel das bibliotecas e exclui indevidamente a atuação do kernel.

E – Alternativa correta.

JUSTIFICATIVA. Na comunicação entre processos por memória compartilhada, os dados podem ser colocados em áreas de memória compartilhada entre os processos. O sistema operacional intermedeia a criação, o mapeamento e a proteção dessas áreas, controlando as permissões e o acesso autorizado. Após o mapeamento, os processos passam a acessar a região compartilhada para leitura e escrita, mantendo a necessidade de sincronização para garantir a consistência dos dados.

3. Indicações bibliográficas

- MACHADO, F. B.; MAIA, L. P. *Fundamentos de sistemas operacionais*. Rio de Janeiro: LTC, 2011.
- MACHADO, F. B.; MAIA, L. P. *Arquitetura de sistemas operacionais.* 5. ed. Rio de Janeiro: LTC, 2013.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de sistemas operacionais*. 9. ed. Rio de Janeiro: LTC, 2015.
- STALLINGS, W. *Arquitetura e organização de computadores*. 8. ed. São Paulo: Pearson Prentice Hall, 2010.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson, 2016.

Questão 10

Questão 10.[10]

Leia o texto a seguir.

Alguns sistemas com memória virtual utilizam uma técnica chamada de paginação. Nesses sistemas, existe um conjunto de endereços de memória, denominados endereços virtuais, que são gerados durante a execução dos programas, com o uso de indexação, de registradores-base, de registradores-segmento ou de outras técnicas. Um endereço virtual é dividido em número de página virtual e deslocamento. O número de página virtual é usado como índice dentro da tabela de páginas para encontrar o quadro correspondente. O endereço físico de memória é a concatenação entre o endereço do quadro com o deslocamento do endereço virtual. Um mecanismo denominado TLB (do inglês, translation lookaside buffer), tipicamente implementado em hardware, fornece auxílio durante a atividade de mapeamento de endereços virtuais para endereços físicos sem passar pela tabela de página. A função do TLB é agilizar o processo de tradução de endereços lógicos para físicos.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 3. ed. São Paulo: Pearson Prentice Hall, 2009 (com adaptações).

Com relação à memória paginada, avalie as asserções e a relação proposta entre elas.

I. Quando um processo é escalonado para execução, tanto a MMU (Memory Management Unit) quanto o TLB são reconfigurados para o novo processo.

PORQUE

II. Para livrar-se de resíduos do processo executado anteriormente, a tabela de páginas do novo processo deve tornar-se a tabela atual, o que, em geral, é feito por meio da cópia da tabela ou de um ponteiro para ela em registradores em hardware.

A respeito dessas asserções, assinale a opção correta.

A. As asserções I e II são proposições verdadeiras, e a II é uma justificativa da I.
B. As asserções I e II são proposições verdadeiras, e a II não é uma justificativa da 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.

[10]Questão 28 – Enade 2023.

1. Introdução teórica

1.1. Memória virtual paginada

A memória virtual paginada é uma técnica de gerência de memória em que o espaço de endereçamento de cada processo é organizado em páginas de tamanho fixo, enquanto a memória principal (RAM) é dividida em quadros (frames) do mesmo tamanho; com isso, os endereços gerados pelo programa (chamados de endereços virtuais) são traduzidos pelo hardware e pelo sistema operacional para os endereços físicos por meio de estruturas como a tabela de páginas. Esse mecanismo permite que um processo utilize um espaço de memória maior do que a memória RAM disponível, pois somente as páginas necessárias em dado momento precisam estar carregadas na memória principal, enquanto as outras permanecem em armazenamento secundário (como no SSD ou em disco) e são trazidas quando ocorre uma falta de página (page fault).

O modelo paginado melhora o aproveitamento da memória, reduz a fragmentação externa e favorece o isolamento e a proteção entre processos, embora introduza os custos de tradução e os de acesso quando há muitas faltas de página.

A tradução de endereços em sistemas com uma memória virtual paginada é um ponto central da gerência de memória em sistemas operacionais. Nesse modelo, o programa trabalha com endereços virtuais (ou lógicos), enquanto a memória principal é acessada por meio de endereços físicos. Para tornar essa abstração viável, o endereço virtual é dividido em duas partes: o número da página virtual e o deslocamento (offset). O número da página é usado para consultar a tabela de páginas do processo, e essa consulta informa qual quadro (frame) da memória física contém a página correspondente. Em seguida, o endereço físico é formado pela combinação do número do quadro com o mesmo deslocamento.

1.2. MMU

A MMU (Memory Management Unit) é o componente de hardware que realiza essa tradução durante a execução das instruções e dos acessos à memória. Como essa tradução ocorre com enorme frequência, fazer uma consulta completa à tabela de páginas a cada acesso seria custoso. Para reduzir esse custo, os processadores usam a TLB (Translation Lookaside Buffer), uma memória associativa pequena e muito rápida, que mantém em cache traduções recentes de páginas virtuais para quadros físicos. Quando a tradução desejada está

na TLB, o acesso é acelerado; quando não está, ocorre um "miss", e o hardware (com apoio do sistema operacional, dependendo da arquitetura) precisa consultar a tabela de páginas.

Esse mecanismo conecta-se diretamente à troca de contexto entre processos. Cada processo tem seu próprio espaço de endereçamento virtual e, em geral, sua própria tabela de páginas. Quando o escalonador troca o processo em execução, o sistema operacional precisa fazer com que a MMU passe a usar a tabela de páginas do novo processo, normalmente carregando em registrador(es) de hardware um ponteiro/base para essa tabela (por exemplo, o registrador base da tabela de páginas). Além disso, como a TLB pode conter traduções do processo anterior, suas entradas podem precisar ser invalidadas (flush) ou diferenciadas por identificadores de espaço de endereçamento (ASID/PCID, conforme a arquitetura). Sem esse cuidado, traduções antigas poderiam ser usadas de forma indevida, produzindo acessos incorretos à memória.

2. Análise das asserções

I – Asserção verdadeira.

JUSTIFICATIVA. Quando há troca de processo, a MMU precisa passar a traduzir endereços usando a tabela de páginas do novo processo. Isso exige a atualização de informações de hardware (por exemplo, o registrador base da tabela de páginas). Em relação à TLB, como ela armazena as traduções do processo que estava executando antes, também é necessário tratá-la: em muitas arquiteturas, isso ocorre por invalidação das entradas; em outras, por troca/uso de identificadores de contexto. Em ambos os casos, há uma adaptação do estado da tradução para o novo processo, o que sustenta a ideia de "reconfiguração".

II – Asserção verdadeira.

JUSTIFICATIVA. O novo processo precisa ter sua tabela de páginas como referência atual da tradução. Em arquiteturas modernas, isso é feito tipicamente por meio de um ponteiro/base para a tabela em registrador(es) de hardware. A redação menciona "cópia da tabela ou de um ponteiro", o que é aceitável em termos didáticos/históricos: há modelos de hardware mais simples em que estruturas de tradução eram mantidas diretamente em registradores, enquanto, na prática mais comum, usa-se um ponteiro/base para a tabela em memória. A ideia essencial da asserção está correta: a base da tradução precisa ser trocada para o novo processo, e isso é feito com suporte de hardware.

Relação entre asserções. A asserção II justifica a I. Se a tabela de páginas do novo processo precisa tornar-se a tabela atual para que a tradução passe a refletir o novo espaço de endereçamento, então é necessário atualizar o estado da MMU (por exemplo, o registrador base da tabela de páginas). Além disso, para evitar o uso de traduções antigas (resíduos), é preciso tratar a TLB (invalidação ou mecanismo equivalente de distinção por contexto). Portanto, a razão apresentada na II explica por que, ao escalonar um novo processo, MMU e TLB precisam ser ajustados ao novo contexto de execução.

Alternativa correta: A.

3. Indicações bibliográficas

- MACHADO, F. B.; MAIA, L. P. *Fundamentos de sistemas operacionais*. Rio de Janeiro: LTC, 2011.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de sistemas operacionais*. 9. ed. Rio de Janeiro: LTC, 2015.
- STALLINGS, W. *Operating Systems: Internals and Design Principles*. 9. ed. Pearson, 2021.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson Education do Brasil, 2015.

ÍNDICE REMISSIVO