UNIP
UNIVERSIDADE PAULISTA

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 1

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 1

Material instrucional específico, cujo conteúdo integral ou parcial não pode ser reproduzido ou utilizado sem autorização expressa, por escrito, da CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP – UNIVERSIDADE PAULISTA.

Questão 1

Questão 1.[1]

Com relação às diferentes tecnologias de armazenamento de dados, julgue os itens a seguir.

I. Quando a tensão de alimentação de uma memória ROM é desligada, os dados dessa memória são apagados. Por isso, esse tipo de memória é denominado volátil.

II. O tempo de acesso à memória RAM é maior que o tempo de acesso a um registrador da unidade central de processamento (UCP).

III. O tempo de acesso à memória cache da UCP é menor que o tempo de acesso a um disco magnético.

IV. O tempo de acesso à memória cache da UCP é maior que o tempo de acesso à memória RAM.

São corretos apenas os itens

A. I e II.

B. I e III.

C. II e III.

D. II e IV.

E. III e IV.

1. Introdução teórica

1.1. Memória

Memória é o termo genérico utilizado para designar os dispositivos de um computador que possibilitam o armazenamento de dados ou de programas. Sem os dispositivos de memória, a UCP (Unidade Central de Processamento), também conhecida como CPU (*Central Processing Unit*), não poderia ler e escrever informações.

Os sistemas de armazenamento são organizados em hierarquia, segundo critérios que envolvem a velocidade, o custo e a volatilidade de cada dispositivo (figura 1).

[1]Questão 11 - Enade 2008.

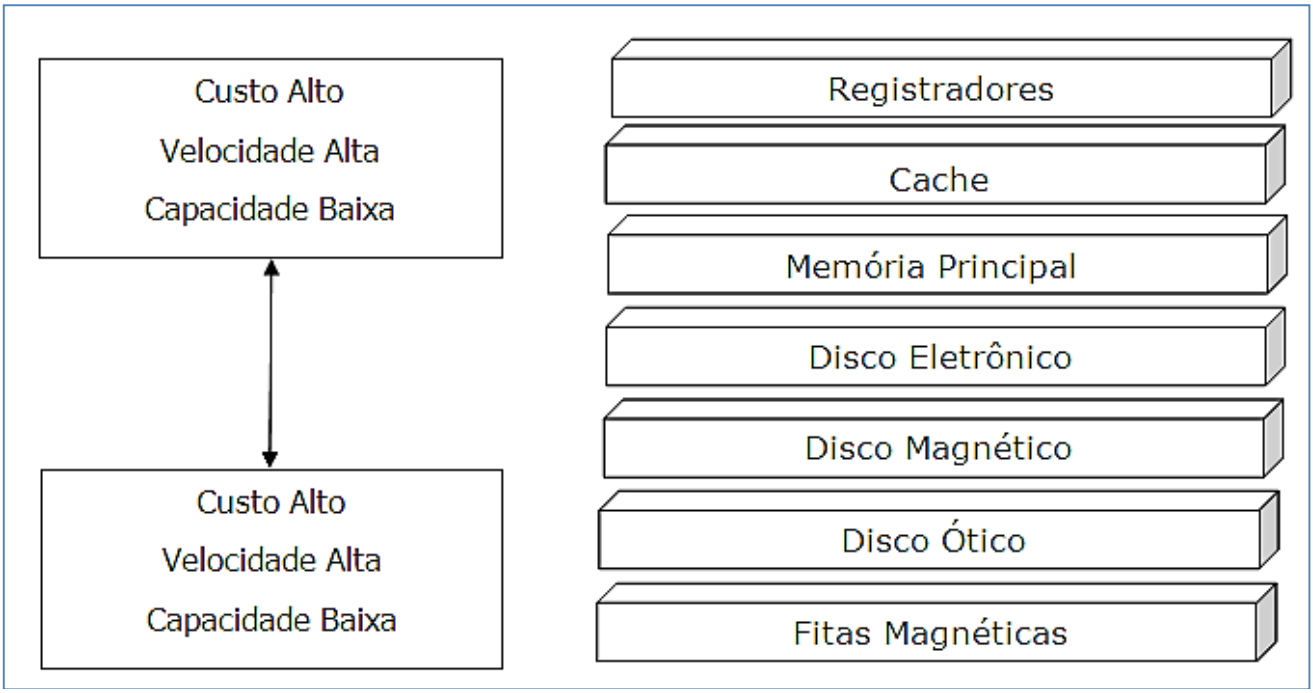


Figura 1. Hierarquia de dispositivos de armazenamento.
Fonte: Silberschatz, Galvin e Gagne (2008).

A memória do computador pode ser dividida em duas categorias, conforme segue.

- **Principal**. A memória principal é de acesso rápido e de capacidade restrita. Armazena informações temporariamente durante um processamento realizado pela UCP.
- **Secundária**: A memória secundária é de acesso lento e de capacidade elevada. Armazena conjunto de dados que a memória principal não suporta.

1.2. Tipos de memórias

As memórias podem ser classificadas segundo a volatilidade. Quando o computador é desligado, os dados armazenados na memória continuarão armazenados (memória não-volátil) ou serão perdidos (memória volátil). De acordo com esse critério e outras características, podemos ter os tipos de memórias descritos a seguir.

- **Memória RAM:** é a memória principal do computador, usada para guardar dados e instruções de um programa. Apresenta volatilidade, pois o seu conteúdo é perdido quando o computador é desligado, e permite o acesso aleatório aos dados para leitura e gravação.
- **Vídeo RAM:** é uma área especializada da memória RAM na qual a CPU compõe, detalhadamente, a imagem mostrada no monitor.
- **Memória ROM:** é um tipo de memória que contém instruções de acesso aleatório. Não apresenta volatilidade, ou seja, os dados não são perdidos com a ausência de energia.

Nela, estão localizadas rotinas que inicializam o computador quando ele é ligado. Alguns dos tipos de memória ROM são EPROM e EEPROM.

- **Memória cache:** é uma memória de alta velocidade, constituindo a interface entre o processador e a memória primária (memória RAM).
- **Memórias secundárias:** são memórias não voláteis, com elevadas capacidades de armazenamento e baixos custos. Podem ser removíveis ou não. Alguns exemplos são o disco rígido (HD), os CDs, os DVDs, o *pen drive* etc.

2. Indicações bibliográficas

- TANENBAUM, A. S. *Organização estruturada de computadores*. 5. ed. São Paulo: Prentice Hall, 2007.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Sistemas operacionais com Java.* Rio de Janeiro: Elsevier, 2008.

3. Análise das afirmativas

I – Afirmativa incorreta.
JUSTIFICATIVA. Os dados da memória ROM não são apagados com a perda da alimentação. Trata-se de uma memória não-volátil.

II – Afirmativa correta.
JUSTIFICATIVA. Segundo a hierarquia de desempenho das memórias, o registrador é o tipo de memória mais rápida disponível nos computadores atuais. Portanto, o acesso à memória RAM (memória principal) é mais lento do que o acesso aos registradores (figura 1).

III – Afirmativa correta.
JUSTIFICATIVA. Segundo a hierarquia de desempenho das memórias, a memória cache é o segundo tipo de memória mais rápida disponível nos computadores atuais. Portanto, o acesso à memória representada pelos discos magnéticos é mais lento do que o acesso à memória cache (figura 1).

IV – Afirmativa incorreta.

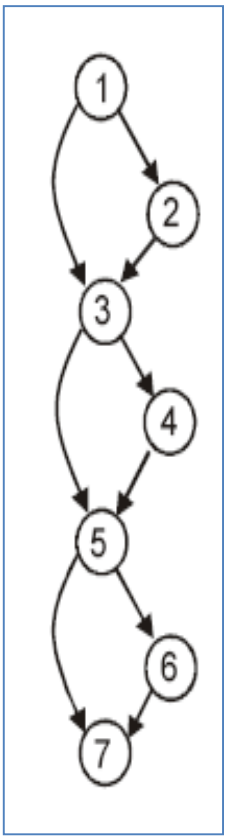
JUSTIFICATIVA. Segundo a hierarquia de desempenho das memórias, a memória cache só perde em tempo de acesso para os registradores. Portanto, o acesso à memória RAM é mais lento do que o acesso à memória cache (figura 1).

Alternativa correta: C.

Questão 2

Questão 2.[2]

Ao longo de todo o desenvolvimento do software, devem ser aplicadas atividades de garantia e qualidade de software (GQS), entre as quais se encontra a atividade de teste. Um dos critérios de teste utilizados para gerar casos de teste é o denominado critério dos caminhos básicos, cujo número de caminhos pode ser determinado com base na complexidade ciclomática. Considerando-se o grafo de fluxo de controle apresentado na figura ao lado, no qual os nós representam os blocos de comandos e as arestas representam a transferência de controle, qual a quantidade de caminhos básicos que devem ser testados no programa associado a esse grafo de fluxo de controle, sabendo-se que essa quantidade é igual à complexidade ciclomática mais um?



A. 1 B. 3 C. 4 D. 7 E. 8

1. Introdução teórica

Testes de *software*

O desenvolvimento de sistemas de software envolve uma série de atividades de produção, que apresentam diversas oportunidades para a ocorrência de erros. Os erros podem acontecer desde o início do processo, quando os objetivos que o software deverá obedecer podem ser especificados de maneira equivocada. Cada erro não identificado em alguma fase do ciclo de desenvolvimento de um software pode multiplicar-se várias vezes.

O teste de software é um elemento crítico da garantia de qualidade de software e está presente em todas as fases do ciclo de desenvolvido.

O teste de caminho básico corresponde a uma técnica de teste de software proposta por Tom McCabe, em 1976, que permite ao projetista de casos de teste originar uma medida

[2]Questão 12 - Enade 2008.

da complexidade lógica de um projeto procedimental e usar essa medida como guia para definir um conjunto básico de caminhos de execução. Esse teste mede a quantidade de lógica de decisão usada em um módulo de software, ou seja, o número de caminhos linearmente independentes através do código-fonte de um programa.

O grafo de fluxo representa o fluxo de controle (teste de caminho básico), sendo que cada construção estruturada apresenta um símbolo correspondente (figura 1).

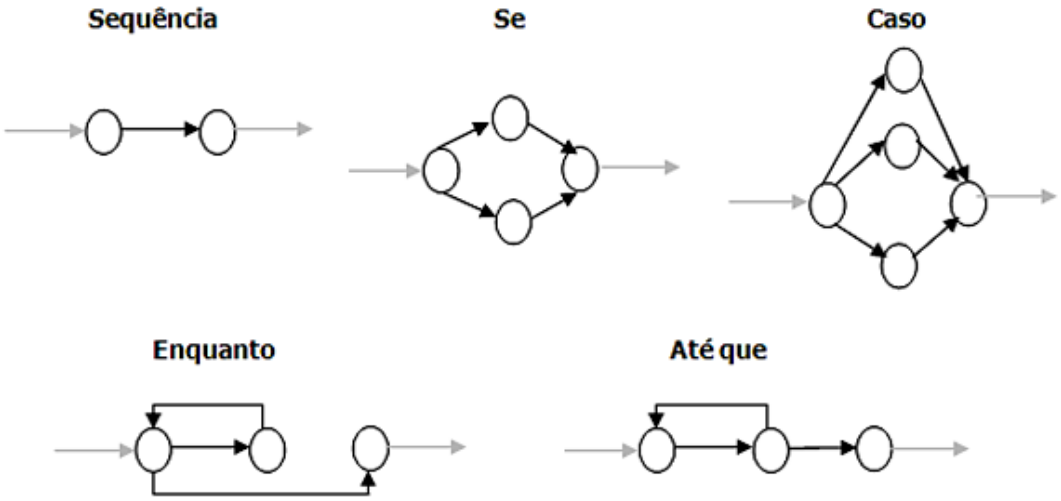


Figura 1. Notação de um grafo de fluxo.
Fonte: Pressman, R. S. (2002).

A figura 2 exemplifica uma comparação entre uma técnica tradicional de representação da sequência de um algoritmo (fluxograma) e a técnica denominada grafo de fluxo.

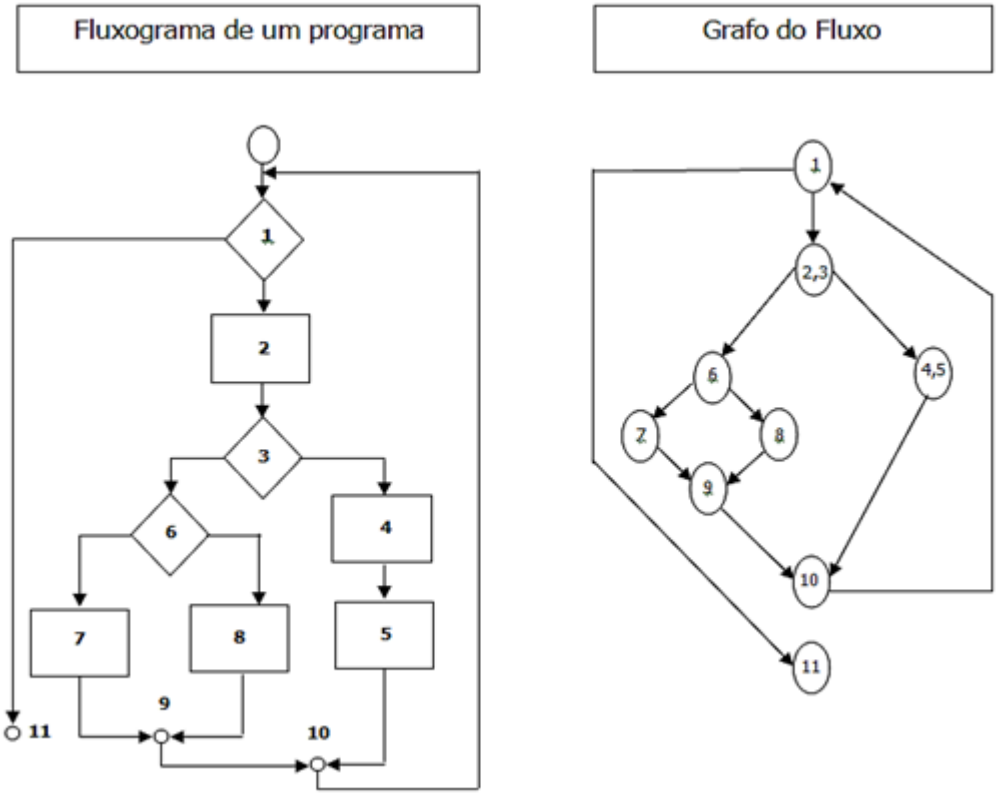


Figura 2. Mapeamento de um fluxograma em um grafo de fluxo.
Fonte: Pressman, R. S. (2002).

Em um grafo de fluxo, os círculos representam nodos e as linhas representam arestas. Esse conceito será necessário para o cálculo ciclomático.

A métrica de software que fornece uma medida quantitativa da complexidade lógica de um programa é denominada complexidade ciclomática. Quando essa métrica é utilizada no contexto do método de teste de caminho básico, o valor calculado para a complexidade ciclomática define o número de caminhos independentes no conjunto base de um programa, fornecendo um limite superior para a quantidade de testes que deve ser conduzida a fim de garantir que todos os comandos tenham sido executados pelo menos uma vez.

Para a figura 2, o grafo de fluxo apresenta as situações abaixo.

- Caminho 1: 1 – 11.
- Caminho 2: 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11.
- Caminho 3: 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11.
- Caminho 4: 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 - 11.

Quanto maior for a complexidade, maior será o número de casos de testes necessários para verificar um módulo.

Sendo E o número de arestas do grafo, N o número de nodos do grafo e P o número de componentes do grafo G, o número ciclomático de um grafo V(G) é igual a:

$$V(G) = E – N + P$$

2. Indicação bibliográfica

- PRESSMAN, R. S. *Engenharia de software*. 5. ed. Rio de Janeiro: MacGraw-Hill, 2002.

3. Resolução da questão

Vimos que o número ciclomático V(G) de um grafo é dado por:

$$V(G) = E – N + P$$

Um componente de um grafo é um conjunto de nodos conectados. Nesse caso, todo grafo de programa terá número de componentes P igual a 1 (P = 1), pois os nodos não conectados correspondem a comandos que nunca serão executados.

Da figura do enunciado, o número de arestas E vale 9 e o número N de nodos vale 7. Logo: V(G) = 9 – 7 + 1 = 3

O método dos caminhos básicos de McCabe acrescenta que, quando os grafos de um programa não forem fortemente conexos, é necessário adicionar uma aresta conectando o último nodo ao primeiro.

Ou seja, para se obter o resultado correto da questão deve-se somar 1 ao cálculo V(G). Como V(G) = 3, o resultado da será 3 + 1 = 4.

Alternativa correta: C.

Questão 3

Questão 3.[3]

Considerando o conjunto A = {1, 2, 3, 4, 5, 6}, qual opção corresponde a uma partição desse conjunto?

A. {{1}, {2}, {3}, {4}, {5}, {6}}

B. {{1}, {1,2}, {3,4}, {5, 6}}

C. {{ }, {1, 2, 3}, {4, 5, 6}}

D. {{1, 2, 3}, {5, 6}}

E. {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}}

1. Introdução teórica

Partição de um conjunto

Uma partição de um conjunto A é uma decomposição de A em subconjuntos não vazios tais que todo o elemento de A pertence a somente um desses subconjuntos. A cada um desses subconjuntos, chamamos elementos da partição.

As condições de partição estão listadas a seguir.

- A união de todos os subconjuntos é igual a A.
- Os subconjuntos são disjuntos, isto é, não há elementos comuns a dois ou mais subconjuntos (ou seja, a intersecção de cada par de subconjuntos é vazia).
- Não existe a possibilidade de um subconjunto vazio.

Representa-se uma partição de um conjunto por PA={A1, A2,..., An}, para n > 0.

Vejamos um exemplo: se A={1, 2, 3, 4, 5}, então temos o que segue abaixo.

- α={A1, A2}, com A1={1, 2} e A2={3, 4, 5}, é uma partição do conjunto A.
- β={{1}, {2, 3}, {4, 5}} é uma partição do conjunto A.
- δ={{1}, {1, 2, 3}, {4, 5}} não é uma partição do conjunto A.
- γ={$\varnothing$, {1, 2, 3}, {4, 5}} não é uma partição do conjunto A.

[3]Questão 13 - Enade 2008.

2. Indicações bibliográficas

- GERSTING, J. L. *Fundamentos matemáticos para a Ciência de Computação.* Rio de Janeiro: LTC, 2004.
- HACK, N. F. R. *Álgebra, uma introdução*. Porto Alegre: EDIPUCRS, 2009.

3. Análise das alternativas

A – Alternativa correta.

JUSTIFICATIVA. A união dos conjuntos {1}, {2}, {3}, {4}, {5}, {6} é equivalente ao conjunto A = {1, 2, 3, 4, 5, 6}. Os conjuntos {1}, {2}, {3}, {4}, {5}, {6} cumprem os requisitos básicos de uma partição de um conjunto, ou seja, não há elemento repetido nos conjuntos nem há elemento vazio.

B – Alternativa incorreta.

JUSTIFICATIVA. A segunda condição da partição de um conjunto não é respeitada. Os conjuntos não são disjuntos, pois o elemento 1 é comum a mais de um subconjunto.

C – Alternativa incorreta.

JUSTIFICATIVA. A terceira condição da partição de um conjunto não é respeitada. A alternativa apresenta um subconjunto vazio.

D – Alternativa incorreta.

JUSTIFICATIVA. A primeira condição da partição de um conjunto não é respeitada. A alternativa não inclui todos os elementos do conjunto A, pois está faltando o elemento 4.

E – Alternativa incorreta.

JUSTIFICATIVA. A segunda condição da partição de um conjunto não é respeitada. Os conjuntos não são disjuntos, pois os elementos 2, 3, 4 e 5 são comuns a mais de um subconjunto.

Questão 4

Questão 4.[4]

Um programador propôs um algoritmo não recursivo para o percurso em pré ordem de uma árvore binária com as seguintes características.

- Cada nó da árvore binária é representado por um registro com três campos: chave, que armazena seu identificador; esq e dir, ponteiros para os filhos esquerdo e direito, respectivamente.
- O algoritmo deve ser invocado inicialmente tomando o ponteiro para o nó raiz da árvore binária como argumento.
- O algoritmo utiliza push() e pop() como funções auxiliares de empilhamento e desempilhamento de ponteiros para nós de árvore binária, respectivamente.

A seguir, está apresentado o algoritmo proposto, em que λ representa o ponteiro nulo.

```
Procedimento preordem (ptraiz : PtrNoArvBin)
Var ptr : PtrNoArvBin;
  ptr := ptraiz;
  Enquanto (ptr ≠ λ) Faça
    escreva (ptr^.chave);
    Se (ptr^.dir ≠ λ) Então
         push(ptr^.dir);
    Se (ptr .esq ≠ λ) Então
        push(ptr^.esq);
     ptr := pop();
  Fim_Enquanto
Fim_Procedimento
```

Com base nessas informações e supondo que a raiz de uma árvore binária com n nós seja passada ao procedimento preordem(), julgue os itens seguintes.

I. O algoritmo visita cada nó da árvore binária exatamente uma vez ao longo do percurso.

II. O algoritmo só funcionará corretamente se o procedimento pop() for projetado de forma a retornar λ caso a pilha esteja vazia.

III. Empilhar e desempilhar ponteiros para nós da árvore são operações que podem ser implementadas com custo constante.

[4]Questão 14 - Enade 2008.

IV. A complexidade do pior caso para o procedimento preordem() é O(n).

Assinale a opção correta.

A. Apenas um item está certo.
B. Apenas os itens I e IV estão certos.
C. Apenas os itens I, II e III estão certos.
D. Apenas os itens II, III e IV estão certos.
E. Todos os itens estão certos.

1. Introdução teórica

Inicialmente, deve-se ressaltar que a questão em análise trata de um processo de simulação de recursividade utilizando pilha.

1.1. Árvores binárias

As árvores binárias são estruturas de dados representadas por um conjunto finito de elementos. Quando o número de nós n for igual a zero, a árvore é nula. Quando n for maior do que zero, a árvore pode formar três conjuntos disjuntos. O primeiro subconjunto contém um único elemento, chamado raiz da árvore. Os outros dois subconjuntos são árvores binárias, chamadas subárvores esquerda e direita da árvore original.

Um conjunto de dados possui a estrutura de uma árvore binária quando se encontra distribuído (figura 1). Os círculos indicam as posições dos elementos, chamadas nós ou vértices. O segmento que une dois vértices denomina-se aresta. O nó mais alto chama-se raiz. Nível de um nó é a distância do nó até a raiz, ou seja, é o número de arestas entre ele e a raiz. Cada nó possui, no nível seguinte, no máximo dois nós vizinhos chamados filhos. Folha ou nó terminal é o nó sem filho. A altura K de uma árvore é a altura do nó mais distante da raiz.

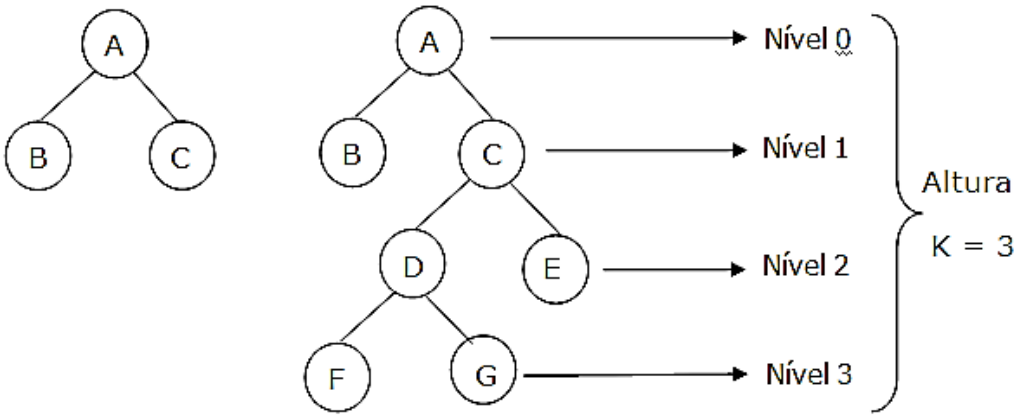


Figura 1. Representação de uma árvore binária.
Fonte: Salvetti, D. D. (1993).

Uma árvore binária é completa se todas as folhas estão no mesmo nível e todos os nós que não possuem folhas têm exatamente dois filhos (figura 2).

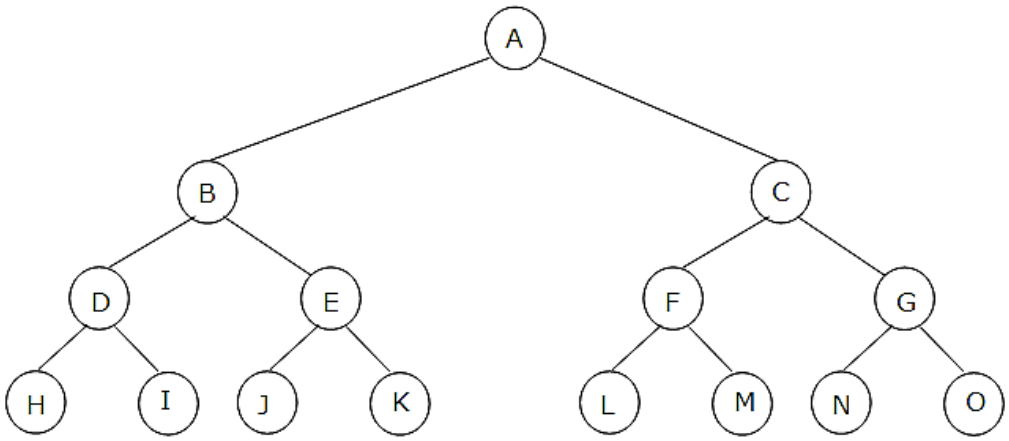


Figura 2. Árvore binária completa.
Fonte: Salvetti, D. D. (1993).

Não há uma ordem natural para se percorrerem os nós das árvores binárias. São utilizados ordenamentos diferentes de percurso em casos distintos. Os métodos disponíveis utilizam recursividade, de modo que percorrer uma árvore binária envolve visitar a raiz e percorrer suas subárvores esquerda e direita. A única diferença entre os métodos é a ordem na qual essas três operações são efetuadas.

No **percurso em pré-ordem**, ou em profundidade, são realizadas as seguintes operações, na ordem abaixo.

1. Visitar a raiz.
2. Percorrer a subárvore esquerda em ordem prévia.
3. Percorrer a subárvore direita em ordem prévia.

No **percurso em ordem**, são realizadas as seguintes operações, na ordem abaixo.

1. Percorrer a subárvore esquerda em ordem simétrica.
2. Visitar a raiz.
3. Percorrer a subárvore direita em ordem simétrica.

No **percurso em pós-ordem**, são realizadas as seguintes operações, na ordem abaixo.

1. Percorrer a subárvore esquerda em ordem posterior.
2. Percorrer a subárvore direita em ordem posterior.
3. Visitar a raiz.

Na figura 3, estão ilustrados os tipos de percursos citados anteriormente.

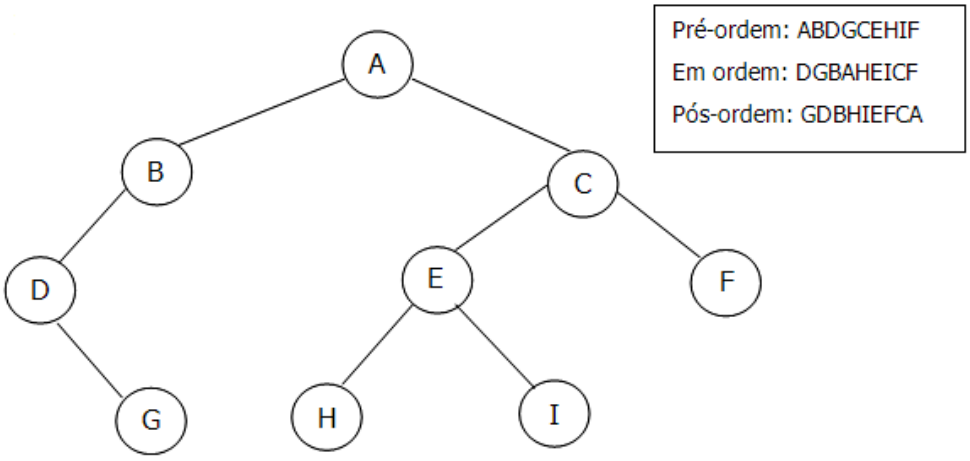


Figura 3. Percorrendo uma árvore em pré-ordem, em ordem e pós-ordem.
Fonte: Tenenbaum, A. M. (1995).

1.2. Complexidade de algoritmos

A função mais primitiva da computação consiste na execução de um aplicativo que represente determinado problema que se pretende resolver. Depois de o problema ser analisado e as decisões serem finalizadas, o algoritmo tem que ser implementado. Nessa fase, o analista deve estudar as várias opções de algoritmos que podem ser utilizados, sendo que aspectos como o tempo de execução e o espaço ocupado em memória são relevantes.

Na área de análise de algoritmos, existem dois tipos de problemas distintos, apontados por Knuth (1971) e apresentados a seguir.

I. Análise de um algoritmo particular.

Qual é o custo de usar dado algoritmo para resolver um problema específico? Realiza-se uma análise do número de vezes que cada parte do algoritmo deve ser executada, seguida do estudo da quantidade de memória necessária.

II. Análise de uma classe de algoritmo.

Qual é o algoritmo de menor custo possível para resolver um problema particular? Investiga-se toda uma família de algoritmos, procurando-se identificar o mais adequado ao problema que se pretende resolver.

Para avaliar o custo de execução de um algoritmo, define-se a função de custo ou função de complexidade f, sendo f(n) a medida de tempo necessário para a execução de um algoritmo relativo a um problema de tamanho n. A não ser que haja uma referência explícita, f denota uma função de complexidade de tempo. A complexidade de tempo, na realidade, não representa diretamente o tempo, mas o número de vezes que determinada operação é executada.

A complexidade de um algoritmo consiste na quantidade de trabalho necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo, e em função do volume de dados.

Existem três escalas de complexidade: melhor caso, caso médio e pior caso. Nas três escalas, a função f(n) retorna a complexidade de um algoritmo com entrada de n elementos.

No caso do algoritmo baseado no pior caso, leva-se em consideração o maior tempo de execução sobre todas as entradas de tamanho n. Vejamos os casos a seguir.

- f(n) = O(1). São algoritmos de complexidade constante, independentes do tamanho de n. As instruções do algoritmo são executadas um número fixo de vezes.
- f(n) = O(log n). São algoritmos de complexidade logarítmica, nos quais o tempo de execução ocorre em algoritmos que resolvem um problema transformando-o em problemas menores.
- f(n) = O(n). São algoritmos de complexidade linear, nos quais, em geral, os elementos de entrada recebem algum tipo de tratamento. Essa é a melhor situação possível para um algoritmo que deve processar n elementos de entrada ou produzir n elementos de saída.
- f(n) = O(n log n). São algoritmos que dividem o problema em problemas menores, juntando, posteriormente, a solução dos problemas menores.

- f(n) = O(n²). São algoritmos de complexidade quadrática, que ocorrem quando os itens de dados são processados aos pares, geralmente um *loop* dentro do outro.

2. Indicações bibliográficas

- TENENBAUM, A. M.; LANGSAM; Y.; AUGENSTEIN M. J. *Estruturas de dados usando C*. São Paulo: Makron Books, 1995.
- ZIVIANI, N. *Projeto de algoritmos: com implementações em Java e C++*. São Paulo: Thomson, 2007.

3. Análise das afirmativas

Os nós da árvore binária utilizada como modelo pela questão apresentam-se subdivididos por um registro com três campos: chave (que representa o valor armazenado), esquerdo e direito (ponteiros para os filhos esquerdo e direito), conforme mostrado na figura 4.

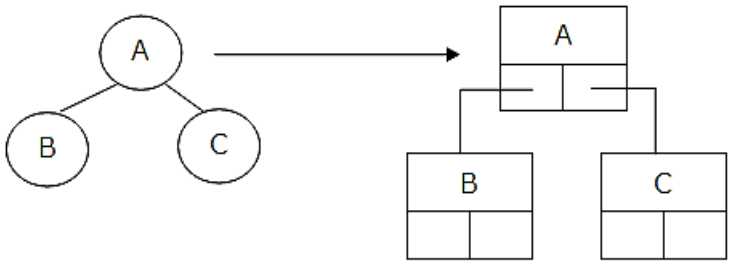


Figura 4. Árvore binária na qual seus registros estão subdivididos em três elementos (chave, esquerdo e direito).

I – Afirmativa correta.

JUSTIFICATIVA. O método de pré-ordem utilizado para percorrer a árvore (figura 3) mostra que cada nó é visitado apenas uma vez.

II – Afirmativa correta.

JUSTIFICATIVA. O algoritmo proposto usa, em paralelo ao algoritmo de árvore, o algoritmo de uma pilha, pois está utilizando as funções push() e pop() das pilhas. A função do algoritmo da pilha é manter armazenados os nós que ainda não foram visitados. O algoritmo proposto executa uma operação pop() que remove da pilha o próximo nó, que deverá ser visitado em pré-ordem. O elemento removido participa da checagem da condição de término do laço de

repetição e, caso o elemento nunca seja nulo, o algoritmo irá entrar em *loop* infinito. A função pop() é a responsável por retornar o valor nulo, caso não exista mais nenhum nó a ser visitado.

III – Afirmativa correta.
JUSTIFICATIVA. As operações de push() e pop() apresentam custos constantes para as operações de empilhamento e desempilhamento. De modo geral, o tempo necessário para recuperar e armazenar um dado na memória costuma ser constante.

IV – Afirmativa correta.
JUSTIFICATIVA. Para percorrer integralmente a árvore, o algoritmo precisa visitar todos os n nodos. A cada passagem pelo laço, um nodo é visitado. Como, no total, são n passos com um número constante de operações em cada passo, o esforço será medido por O(n).

Alternativa correta: E.

Questão 5

Questão 5.[5]

O gerenciamento de configuração de software (GCS) é uma atividade que deve ser realizada para identificar, controlar, auditar e relatar as modificações que ocorrem durante todo o desenvolvimento ou mesmo durante a fase de manutenção, depois que o *software* for entregue ao cliente. O GCS é embasado nos chamados itens de configuração, que são produzidos como resultado das atividades de engenharia de software e que ficam armazenados em um repositório.

Com relação ao GCS, avalie as asserções e a relação proposta entre elas.

I. No GCS, o processo de controle das modificações obedece ao seguinte fluxo: começa com um pedido de modificação de um item de configuração, que leva à aceitação ou não desse pedido e termina com a atualização controlada desse item no repositório

PORQUE

II. O controle das modificações dos itens de configuração baseia-se nos processos de check in e check out que fazem, respectivamente, a inserção de um item de configuração no repositório e a retirada de itens de configuração do repositório para efeito de realização das modificações.

Acerca dessas asserções, assinale a opção correta.

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

[6]Questão 16 - Enade 2008.

1. Introdução teórica

Gestão de configuração de *software*

A gestão de configuração de software é uma atividade guarda-chuva aplicada ao longo de todo o processo. Algumas atividades relacionadas à gestão de configuração de software são:

- identificação e controle de modificações;
- garantia de que as modificações sejam adequadamente implementadas;
- relato das modificações a quem possa interessar.

A modificação é um fato corriqueiro no desenvolvimento de *software*. Clientes modificam requisitos. Desenvolvedores modificam a abordagem técnica. Gerentes modificam a estratégia do projeto. Isso por que, com o passar do tempo, as partes envolvidas ficam mais interadas sobre o que realmente necessitam.

Uma linha-base é um conceito de gestão de configuração de software que auxilia no controle das modificações. No contexto da engenharia de software, uma linha-base é um marco no desenvolvimento de software, constituído pela entrega de um ou mais itens de configuração de software e pela aprovação desse SCI (Item de Configuração de Software), obtida por meio de uma revisão formal.

2. Indicações bibliográficas

- PRESSMAN, R. S. *Engenharia de software*. 5. ed. Rio de Janeiro: MacGraw-Hill, 2002.
- FILHO, W. P. P. *Engenharia de software: fundamentos, métodos e padrões.* 2. ed. Rio de Janeiro: LTC, 2003.

3. Análise das asserções

Análise da primeira asserção.

A primeira asserção é correta, pois o primeiro passo para realizar qualquer modificação de um item de configuração é realizar uma solicitação (figura 1). Se a resposta for positiva, então será possível modificar o item. Ao término do processo de modificação, a base de dados do projeto será atualizada.

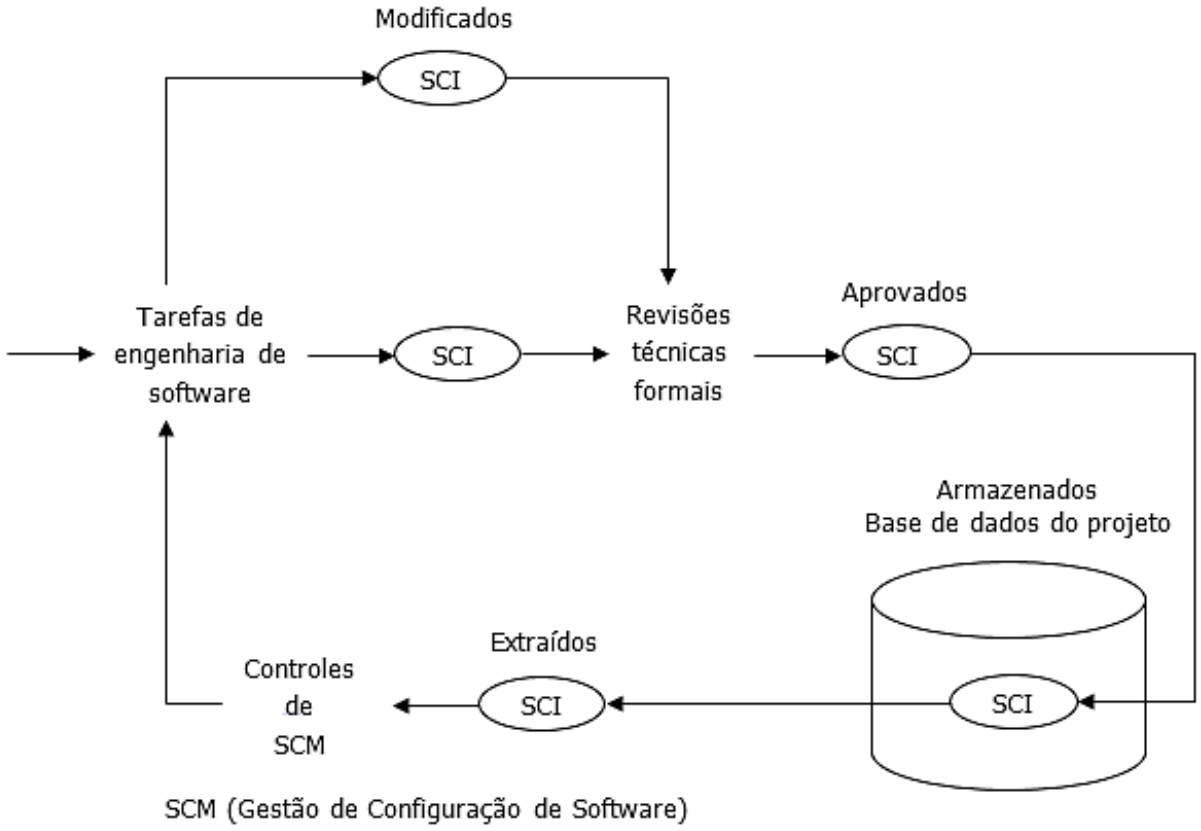


Figura 1. SCI transformados em Linha-Base
Fonte: Pressman, R. S. (2002)

Análise da segunda asserção.

A segunda asserção é correta, pois o controle de modificações dos itens de configuração utiliza os processos de check-in e check-out para a inserção e a retirada de um item de configuração da base de dados.

Alternativa correta: A.

Questão 6

Questão 6.[6]

Uma fórmula bem formada da lógica de predicados é válida se ela é verdadeira para todas as interpretações possíveis. Considerando essa informação, avalie as asserções e a relação proposta entre elas.

I. A fórmula bem formada $(\exists x)\, P(x) \Rightarrow (\forall x)\, P(x)$ é válida.

PORQUE

II. Em qualquer interpretação de uma fórmula da lógica de predicados, se todo elemento do conjunto universo tem a propriedade P, então existe um elemento do conjunto que tem essa propriedade.

Assinale a opção correta com relação a essas asserções.

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

1. Introdução teórica

Lógica e conectivos

Os conectivos são palavras que "ligam" proposições, ou seja, são palavras usadas na formação de outras sentenças. Os principais conectivos são as partículas "e", "ou", "não", "se e somente se" e "se... então". Em resumo:

- Conjunção - conectivo E ($\wedge$)
- Disjunção – conectivo OU ($\vee$)
- Negação – conectivo NÃO ($\neg$)
- Implicação - condicional SE... ENTÃO ($\rightarrow$)
- Equivalência - condicional SE E SOMENTE SE ($\leftrightarrow$)

A lógica de orações explica como funcionam conectivos como "e" (AND), "ou" (OR), "não" (NOT), "se então", "se e somente se" e "nem-ou". Frege incluiu palavras como "todos",

[7]Questão 17 - Enade 2008.

"alguns", e "nenhum" e mostrou como podemos introduzir variáveis e quantificadores para reorganizar orações. Vejamos alguns exemplos.

- "Todos os humanos são mortais" torna-se "Para todo x, se x é humano, então x é mortal", o que pode ser escrito simbolicamente como:
  $\forall x(H(x) \rightarrow M(x))$
- "Alguns humanos são vegetarianos" torna-se "Existe algum (ao menos um) x tal que x é humano e x é vegetariano", o que pode ser escrito simbolicamente como:
  $\exists x(H(x) \wedge V(x))$.

2. Indicação bibliográfica

- GERSTING, J. L. *Fundamentos matemáticos para a Ciência de Computação*. Rio de Janeiro: LTC, 2004.

3. Análise das asserções

É verdade que, se qualquer elemento x (∀x) de um conjunto A tem a propriedade P(x), então existe elemento de A que tem a propriedade P(x).
Trata-se de uma relação de implicação do tipo: (∀x)P(x)→(∃x)P(x).
No entanto, a implicação não prevê que A→B seja idêntico a B→A.
Logo, se (∀x)P(x)→(∃x)P(x), **não** podemos dizer que (∃x)P(x)→(∀x)P(x).
Em outras palavras, se existe elemento x do conjunto A que tem a propriedade P(x), isso não significa que **todo** elemento x do conjunto A tem a propriedade P(x).

Alternativa correta: D.

Questão 7

Questão 7.[7]

Os números de Fibonacci constituem uma sequência de números na qual os dois primeiros elementos são 0 e 1 e os demais, a soma dos dois elementos imediatamente anteriores na sequência. Como exemplo, a sequência formada pelos 10 primeiros números de Fibonacci é: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Mais precisamente, é possível definir os números de Fibonacci pela seguinte relação de recorrência:

fib (n) = 0, se n = 0

fib (n) = 1, se n = 1

fib (n) = fib (n - 1) + fib (n - 2), se n > 1

Abaixo, apresenta-se uma implementação em linguagem funcional para essa relação de recorrência:

```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Considerando que o programa acima não reutilize resultados previamente computados, quantas chamadas são feitas à função fib para computar fib 5?

A. 11 B. 12 C. 15 D. 24 E. 25

1. Introdução teórica

1.1. Recursividade

Uma função ou método que chama a si mesmo, direta ou indiretamente, é dito recursivo. O uso da recursividade geralmente permite uma descrição mais clara e concisa dos algoritmos, em especial quando o problema a ser resolvido é recursivo por natureza ou utiliza estruturas recursivas, tais como as árvores.

[8]Questão 18 - Enade 2008.

1.2. Sequência de Fibonacci

Os números de Fibonacci foram introduzidos, em 1202, pelo matemático italiano Leonardo Fibonacci, que relacionou a sequência de números produzidos pela recorrência abaixo:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Essa sequência consiste em uma sucessão tal que, definidos os dois primeiros números da sequência como 0 e 1, os números seguintes são obtidos por meio da soma dos seus dois antecessores.

Logo, os primeiros números são: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,...

Segue, abaixo, um exemplo de programa em Linguagem C que calcula termos da sucessão:

```
int fib (int n)
{
        if (n <= 1)
                return n;
        return fib (n – 1) + fib (n – 2);
}
```

2. Indicações bibliográficas

- TENENBAUM A. M.; LANGSAM y.; AUGENSTEIN M. J. *Estruturas de dados usando C*. São Paulo: Makron Books, 1995.
- ZIVIANI, N. *Projeto de Algoritmos: com implementações em Java e C++*. São Paulo: Thomson, 2007.

3. Análise da questão

Para calcular fib (5), são necessários os valores de fib (4) e de fib (3). Para calcular fib (4), são necessários os valores de fib (3) e de fib (2). Para calcular fib (3), são necessários os valores de fib (2) e de fib (1). Para calcular fib (2), são necessários os valores de fib (1) e de fib (0). Isso está mostrado na figura 1.

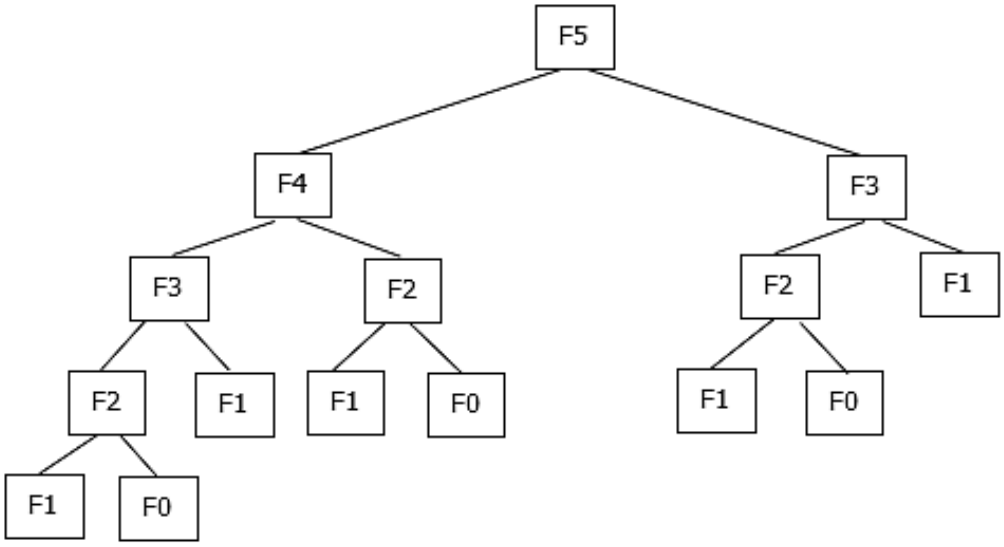


Figura 1. Ocorrências da função fib(n).

Pela figura 1, verificamos que os somatórios dos números de ocorrências da função fib (n), para cada elemento, são:

- fib (0): 3 ocorrências
- fib (1): 5 ocorrências
- fib (2): 3 ocorrências
- fib (3): 2 ocorrências
- fib (4): 1 ocorrência
- fib (5): 1 ocorrência
- Total = 15 ocorrências

Alternativa correta: C.

Questão 8

Questão 8.[8]

Uma alternativa para o aumento de desempenho de sistemas computacionais é o uso de processadores com múltiplos núcleos, chamados multicores. Nesses sistemas, cada núcleo, normalmente, tem as funcionalidades completas de um processador, já sendo comuns, atualmente, configurações com 2, 4 ou mais núcleos.

Com relação ao uso de processadores multicores, e sabendo que threads são estruturas de execução associadas a um processo, que compartilham suas áreas de código e dados, mas mantêm contextos independentes, avalie as asserções e a relação proposta entre elas.

I. Ao dividirem suas atividades em múltiplas threads que podem ser executadas paralelamente, aplicações podem se beneficiar mais efetivamente dos diversos núcleos dos processadores multicores

PORQUE

II. O sistema operacional nos processadores multicores pode alocar os núcleos existentes para executar simultaneamente diversas sequências de código, sobrepondo suas execuções e, normalmente, reduzindo o tempo de resposta das aplicações às quais estão associadas.

Acerca dessas asserções, assinale a opção correta.

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

1. Introdução teórica

1.1. Sistemas com múltiplos processadores

Segundo Tanenbaum (2003), desde o seu início, a indústria de computadores tem buscado cada vez mais poder computacional. Hoje, há máquinas um milhão de vezes mais rápidas do que o primeiro computador e a exigência por maior poder computacional continua aumentando.

[9]Questão 19 - Enade 2008.

No passado, a solução foi aumentar o clock ou deixar o relógio executar mais rapidamente. No entanto, estamos começando a atingir alguns limites fundamentais na velocidade do relógio. De acordo com a teoria da relatividade de Einstein, nenhum sinal elétrico pode propagar-se mais rápido do que a velocidade da luz. Isso significa que, em um computador com um relógio de 10 GHz, os sinais não podem trafegar mais do que 2 cm no total. Para um computador de 100 GHz, o caminho máximo é de 2 mm. Para um computador de 1 THz (1000 GHz), o caminho terá de ser menor do que 100 mícrons, simplesmente para permitir que o sinal trafegue de uma extremidade à outra e retorne dentro de um único ciclo de relógio.

Não é impossível construir computadores tão pequenos e, além disso, outro problema deve ser considerado: como dissipar o calor gerado. Quanto mais rápido for o computador, mais calor ele produz.

Outro meio de se aumentar o poder computacional é utilizar computadores paralelos. Essas máquinas são constituídas de muitas CPUs, sendo que cada uma trabalha com sua velocidade normal, mas coletivamente oferecem muito mais poder computacional do que uma única CPU.

1.2. Gerência de processos

Os sistemas operacionais executam diversos programas que, por sua vez, representam apenas um conjunto de bytes enquanto se encontrarem estáticos em qualquer tipo de memória não volátil. Para que um programa tenha significado do ponto de vista computacional, ele deve ser capaz de ser executado pela CPU. Para que tal fato ocorra, o programa precisa passar por uma série de checagens e mudanças de estado. Quando o programa tiver condições de ser executado pela CPU, receberá a denominação de processo.

Cada processo possui um Bloco de Controle de Processos (BCP), no qual se armazenam as seguintes informações:

a) estado do processo;
b) contador de programa;
c) registradores da CPU;
d) escalonamento de CPU;
e) gerenciamento de memória;
f) dados contábeis;
g) status de E/S (entrada e saída).

Durante a troca de um processo que está sendo executado pela CPU por outro que se encontra na fila de processos prontos (figura 1), uma série de atividades é realizada:

a) interrupção do processo que está sendo executado pela CPU;

b) transferência do controle para o sistema operacional;

c) salvamento das informações do processo em seu bloco de controle;

d) solicitação de um novo processo pelo despachante ao escalonador de CPU que, por sua vez, deverá selecionar o próximo processo a ser executado pela CPU na fila de processos prontos, enviando-o ao despachante, para que ele possa ser levado à CPU para execução.

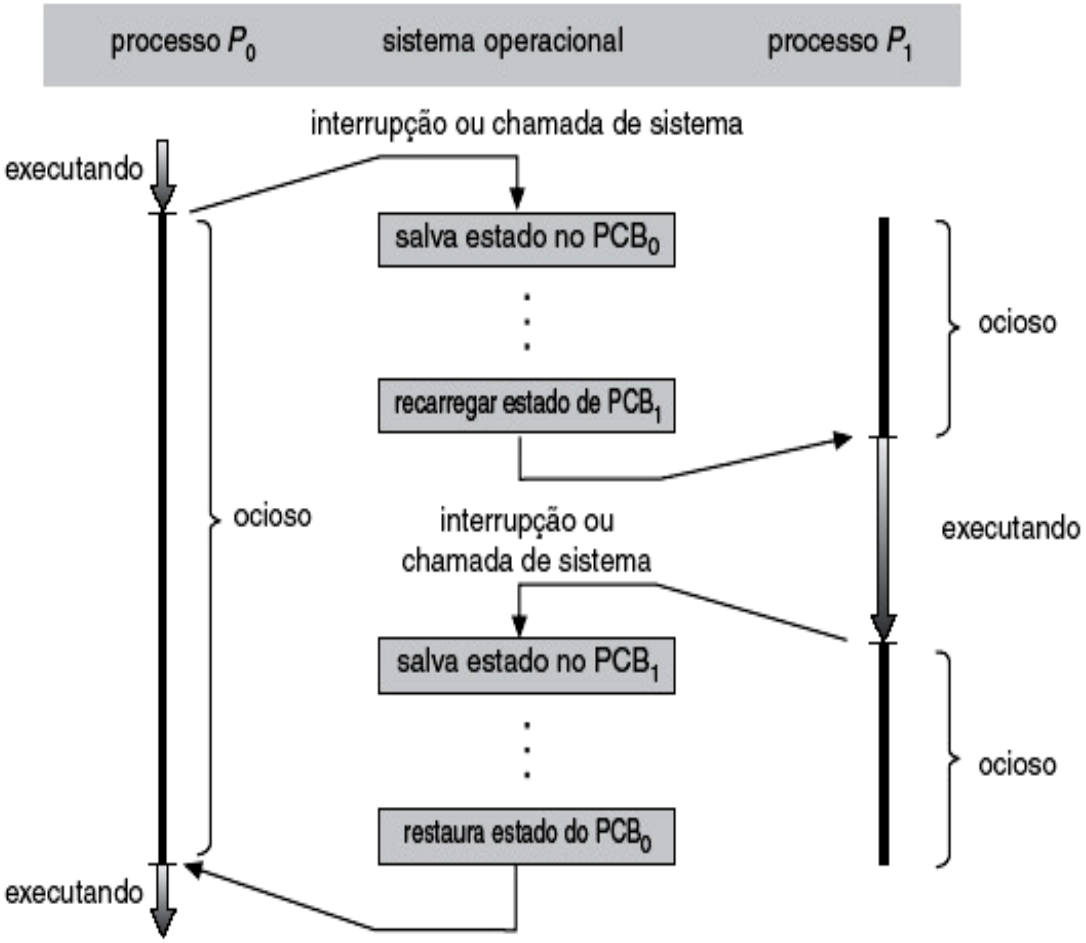


Figura 1. Troca de um processo que está sendo executado pela CPU por outro.
Fonte: Silberschatz, Galvin e Gagne (2008).

A realização de todas as tarefas envolvidas na troca de um processo que se encontra em execução por outro que se encontra na fila de processos prontos acarreta ociosidade por parte da CPU. Portanto, há perda de desempenho do computador. Para minimizar esse problema, criou-se a possibilidade de processos gerarem subprocessos que, por sua vez, poderiam gerar outros, formando uma árvore de processos. Evita-se, assim, a troca de processos entre a CPU e a fila de processos prontos. Apesar de existirem vários processos na

forma de subprocessos, cada um pode estar realizando uma tarefa independentemente de outro subprocesso ou, ainda, pode haver compartilhamento de funções entre os subprocessos, mas, para a CPU, existe apenas um processo sendo executado. Os subprocessos são chamados de threads.

Conforme mostrado na figura 2, cada thread contém:

a) um código;

b) um conjunto de registradores;

c) pilhas do usuário e do kernel separadas;

d) uma área de armazenamento privada.

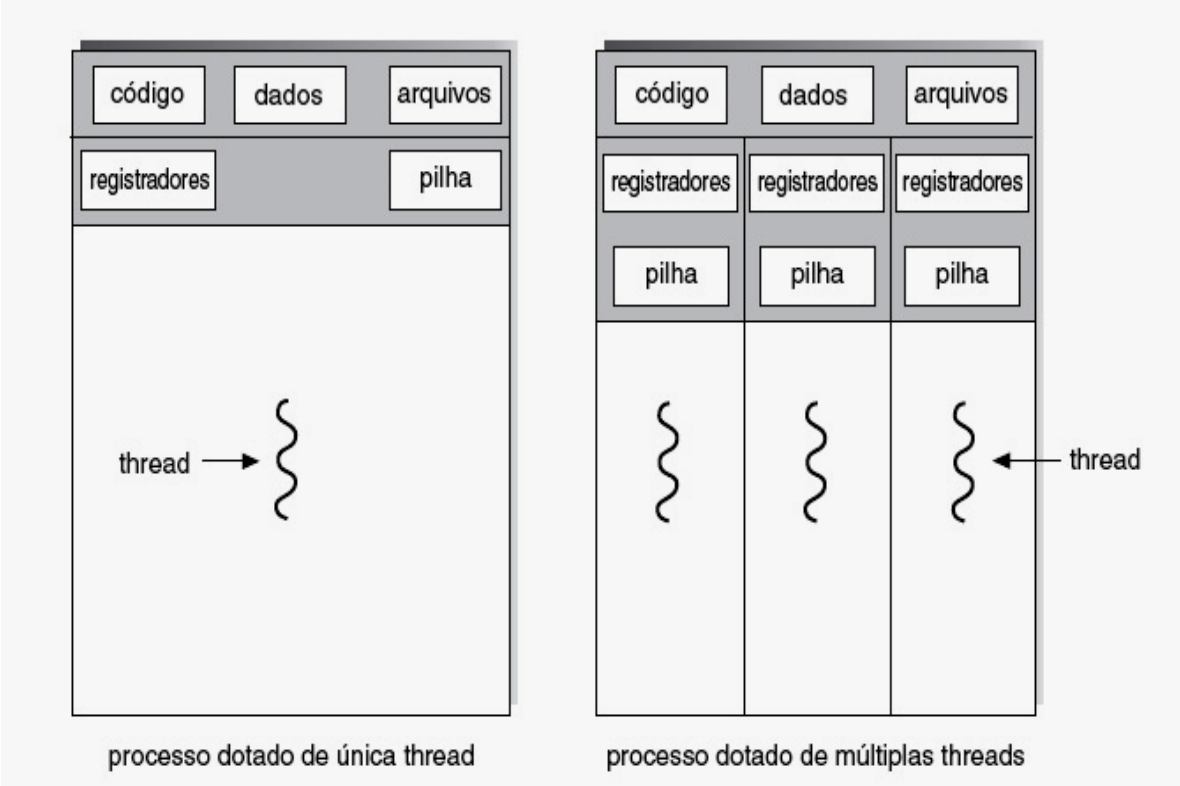


Figura 2. Detalhamento de uma thread.
Fonte: Silberschatz, Galvin e Gagne (2008).

As possibilidades de compartilhamento de recursos em threads são as seguintes: pai e filhos compartilham todos os recursos; filhos compartilham um subconjunto dos recursos do pai; pai e filhos não compartilham recurso algum.

As possibilidades de execução das threads são as seguintes: pai e filhos são executados concorrentemente; pai espera até que os filhos terminem; um processo independente não pode afetar ou ser afetado pela execução de outro processo; um processo cooperante pode afetar ou ser afetado pela execução de outro processo.

A interferência de uma thread em outra pode causar o travamento do computador, pois seções compartilhadas podem afetar a execução de outro subprocesso. A esse travamento

dá-se o nome de deadlock. O sincronismo entre as regiões compartilhadas pelos subprocessos é essencial.

Pode haver deadlock se quatro condições ocorrerem simultaneamente, conforme indicado a seguir.

- **Exclusão mútua**: somente um processo de cada vez pode usar um recurso.
- **Manter e esperar**: um processo retendo pelo menos um recurso está esperando para adquirir recursos adicionais mantidos por outros processos.
- **Sem preempção**: um recurso pode ser liberado apenas voluntariamente pelo processo que o mantém, após esse processo ter completado sua tarefa.
- **Espera circular**: existe um conjunto {P0, P1,…, Pn-1, Pn, P0} de processos esperando, de modo que P0 está esperando um recurso que é mantido por P1, P1 está esperando um recurso que é mantido por P2,…, Pn–1 está esperando um recurso que é mantido por Pn, e Pn está esperando um recurso que é mantido por P0.

2. Indicações bibliográficas

- BUYYA, R. *High performance cluster computing*: architectures and systems. New Jersey: Prentice Hall, 1999. v. 1
- HENNESSY, J.; PATTERSON, D. *Arquitetura de computadores*: uma abordagem quantitativa. 3. ed. Rio de Janeiro: Campus, 2003.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Sistemas operacionais com Java*. Rio de Janeiro: Elsevier, 2008.
- TANENBAUM, A. S. *Sistemas operacionais modernos*. 2. ed. São Paulo: Prentice Hall, 2003.

3. Análise da questão e das asserções

As threads podem ser utilizadas em sistemas computacionais que dipõem de um processador ou de vários processadores. No caso de sistemas com um processador, as threads permitem diminuição da ociosidade da CPU durante a troca de processos. No caso de vários processadores, as threads permitem a migração de processos entre eles, reduzindo ainda mais o tempo de ociosidade das CPUs.

O problema está em manter o sincronismo entre os processos durante o compartilhamento de recursos. A concorrência por uma região da memória pode alterar dados que serão utilizados por outros processos. Caso isso ocorra, haverá divergência entre o dado

que o processo busca na memória e o que realmente se encontra naquela posição de memória, corrompendo o processo.

Análise da primeira asserção.

A primeira asserção faz afirmações corretas sobre as arquiteturas multicores, pois o aumento da capacidade computacional pode ser obtido com o aumento do número de núcleos dos processadores. O autor da questão tomou o cuidado de afirmar que o acréscimo de núcleos não é a única maneira de se elevar o desempenho computacional. Uma alternativa seria a construção de clusters, formados pela união de vários computadores em rede, formando um sistema de imagem única. Os conceitos sobre threads também estão pertinentes.

Análise da segunda asserção.

A segunda asserção é verdadeira e justifica a primeira, pois o sistema operacional nos processadores multicores pode executar simultaneamente várias sequências de código distribuídas pelos diversos processadores.

Alternativa correta: A.

Questão 9

Questão 9.[9]

Tabelas de dispersão (tabelas hash) armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. As funções de dispersão transformam chaves em endereços base da tabela, ao passo que o tratamento de colisões resolve conflitos em casos em que mais de uma chave é mapeada para um mesmo endereço-base da tabela.

Suponha que uma aplicação utilize uma tabela de dispersão com 23 endereços-base (índices de 0 a 22) e empregue h(x) = x mod 23 como função de dispersão, em que x representa a chave do elemento cujo endereço-base deseja-se computar.

Inicialmente, essa tabela de dispersão encontra-se vazia. Em seguida, a aplicação solicita uma sequência de inserções de elementos cujas chaves aparecem na seguinte ordem: 44, 46, 49, 70, 27, 71, 90, 97, 95.

Com relação à aplicação descrita, faça o que se pede a seguir.

A. Escreva, no espaço reservado, o conjunto das chaves envolvidas em colisões.
B. Assuma que a tabela de dispersão trate colisões por meio de encadeamento exterior. Esboce a tabela de dispersão para mostrar seu conteúdo após a seqüência de inserções referida.

1. Introdução teórica

Tabela hash

A tabela hash, conhecida como tabela de espalhamento ou dispersão, é uma estrutura de dados que armazena valores com base no valor absoluto de suas chaves e nas técnicas de tratamento de colisões. A coluna do quadro 1 denominada posição representa os índices da tabela hash e a coluna denominada valor representa valores/itens armazenados na tabela.

[9] Questão 20 – Discursiva - Enade 2008.

Quadro 1. Representação de uma tabela Hash.

Posição	Valor
0	100
1	
2	240
...	
N	23

Existe uma organização ao se associar um índice a um valor. Para armazenar um valor na tabela, será necessário utilizar uma função que determina a posição correspondente ao valor que se pretende armazenar.

A fórmula do cálculo da função hash é h(x) = x mod m. Nessa fórmula, x representa o valor que se pretende armazenar, m representa o tamanho da tabela e x mod m representa o resto da divisão de x por m.

Segue um exemplo da aplicação da fórmula:

Lista de valores: {236, 35, 57, 33, 466, 876, 26, 18, 49}

Para uma tabela com tamanho 15, temos os cálculos abaixo.

236 mod 15 → Valor que se pretende armazenar (x)

```
236 | 15  → Tamanho da tabela Hash ( m )
15    15
 86
 75
 11   → Resto da divisão
```

h (236)	= 236 mod 15	índice/posição = 11
h (35)	= 35 mod 15	índice/posição = 5
h (57)	= 57 mod 15	índice/posição = 12
h (33)	= 33 mod 15	índice/posição = 3
h (466)	= 466 mod 15	índice/posição = 1
h (876)	= 236 mod 15	índice/posição = 6
h (26)	= 236 mod 15	índice/posição = 11
h (18)	= 18 mod 15	índice/posição = 3
h (49)	= 49 mod 15	índice/posição = 4

O resto da divisão de 236 por 15 será igual a 11, portanto o índice utilizado para inclusão do valor 236 na tabela *hash* será o correspondente à posição 11.

O quadro 2 apresenta o problema das colisões entre valores que disputam a mesma posição da tabela. São os casos: 33 com 18 e 236 com 26.

Quadro 2. Tabela *hash* - problema das colisões.

Posição	Valor
0	
1	466
2	
3	33 → 18
4	49
5	35
6	876
7	
8	
9	
10	
11	236 → 26
12	57
13	
14	
15	

Colisão entre os valores 33 e 18.

Colisão entre os valores 236 e 26.

Uma vez que as colisões são teoricamente impossíveis de serem evitadas, devemos reduzi-las ou tratá-las. Para reduzir as colisões, deve-se melhorar a eficiência da função hash, sendo que uma boa função hash é aquela que garanta o máximo espalhamento da tabela.

Para tratar as colisões, podem-se utilizar as técnicas de encadeamento, sondagem ou rehashing.

No encadeamento, utilizam-se listas para os valores com mesmo valor hash. Ao se fazer a inserção de um valor em uma posição que se encontre ocupada, cria-se uma lista com os valores correspondentes àquela posição.

Na sondagem linear, em caso de colisão, o novo valor irá ocupar a próxima posição livre da tabela. Na sondagem quadrática, em caso de colisão, o novo valor terá sua posição determinada por uma relação quadrática.

No hash duplo, em caso de colisão, utiliza-se uma segunda função hash.

No rehashing, quando a tabela apresenta um número excessivo de valores, ela será aumentada para um valor próximo ao dobro de sua capacidade atual e a função hash é ajustada ao novo limite. Os valores são novamente realocados na tabela.

2. Indicações bibliográficas

- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistemas de banco de dados*. São Paulo: Makron Books, 2003.
- ZIVIANI, N. *Projeto de algoritmos*: com implementação em Pascal e C. 5. ed. São Paulo: Pioneira, 1999.

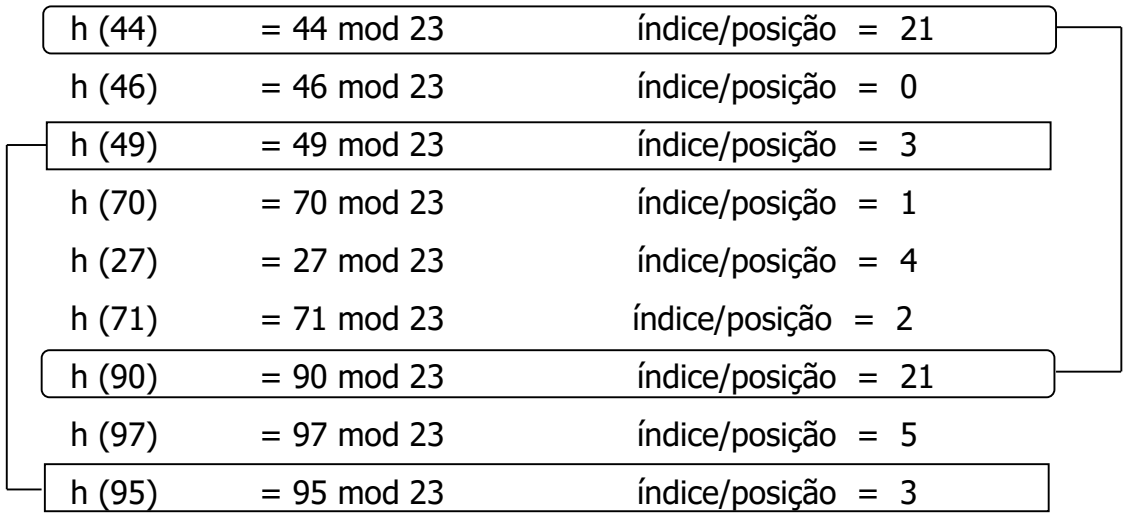
3. Resolução dos itens A e B.

A. A fórmula do cálculo da função hash é h(x) = x mod 23.

Lista de valores: {44, 46, 49, 70, 27, 71, 90, 97, 95}

44 mod 23 → Valor que se pretende armazenar (x)

```
44  | 23  → Tamanho da tabela Hash ( m )
23    1
21  → Resto da divisão
```

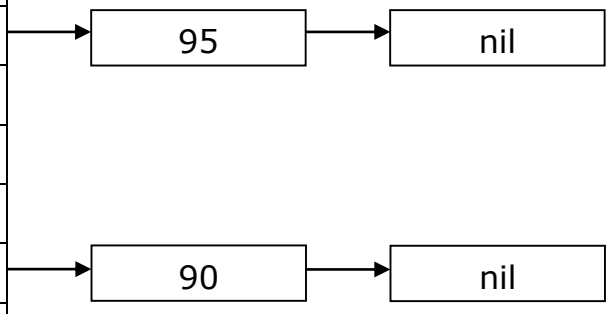


O resto da divisão de 44 por 23 será igual a 21, portanto, o índice utilizado para inclusão do valor 44 na tabela hash será o correspondente à posição 21. As chaves que apresentam colisão são: 44 com 90 e 49 com 95.

B. Um valor colidir com outro implica que o próximo valor a chegar a uma posição ocupada terá de encontrar outro lugar para ser alocado. No caso do tratamento de colisões por encadeamento exterior, os valores que concorrem por uma posição ocupada são alocados em uma lista, ou seja, a tabela hash passa a se comportar como uma lista de listas (quadro 3).

Quadro 3. Hash com colisões por meio de encadeamento exterior

Posição	Valor
0	46
1	70
2	71
3	49
4	27
5	97
...	...
21	44
22	



ÍNDICE REMISSIVO