UNIVESP
Universidade Virtual do Estado de São Paulo

GABARITO

DISCIPLINA	APLICAÇÃO	CÓDIGO DA PROVA
COM160 - Estruturas de Dados	22/04/2021	P001/P002

QUESTÕES OBJETIVAS

Questão 1.1

Qual das alternativas a seguir apresenta apenas estruturas lineares?

a) Árvores AVL e grafos.
b) Filas, pilhas e grafos.
c) Filas, pilhas e árvores.
d) Filas, pilhas e listas encadeadas.
e) Vetores, listas encadeadas e árvores.

RESOLUÇÃO

A resposta correta é: Filas, pilhas e listas encadeadas.

Justificativa

Filas, pilhas e listas encadeadas são estruturas lineares, mas árvores e grafos não são, pois para ser uma estrutura linear é preciso que cada elemento tenha no máximo um sucessor e um predecessor. Observe que árvores possuem no máximo um predecessor, mas podem ter zero ou mais sucessores. Um grafo generaliza o conceito de árvore um pouco mais, dado que um grafo pode ter mais de um sucessor e mais de um predecessor.

Questão 1.2

Sobre as estruturas de dados pilha e fila, assinale a alternativa correta.

a) O comportamento de uma pilha é semelhante ao comportamento do botão "desfazer" de editores de texto, em que, ao acionar o botão, o último caractere inserido é o primeiro a ser removido. Uma fila, pelo contrário, tem comportamento semelhante ao botão "refazer" no mesmo editor de texto, em que supondo que clicamos no botão "desfazer" três vezes para apagar três caracteres (A, B e C, nessa ordem, por exemplo), ao clicar no "refazer", o primeiro elemento "desfeito" será o primeiro refeito (o caractere A será inserido novamente).
b) Na implementação de uma pilha, precisamos de uma variável para indicar a cabeça (topo) da pilha, local onde ocorrerão as inserções e remoções. Na implementação de uma fila, pelo contrário, precisamos de duas variáveis para indicar a posição da frente e de trás da fila, dado que na frente da fila removemos elementos e atrás da fila inserimos elementos.
c) Com uma pilha, implementamos situações em que precisamos garantir acesso justo a um recurso compartilhado, uma pilha de impressão, por exemplo, em que documentos de vários usuários são enviados para a impressora.
d) Com uma fila, podemos implementar situações em que queremos garantir que certas estruturas estão balanceadas, o que é muito comum em linguagens de programação (escopo, chamada de funções, ...). Um exemplo minimalista é a verificação de parênteses, colchetes e chaves aninhados em uma string, tarefa tipicamente solucionada com o uso de filas.

e) Uma pilha é uma estrutura linear, dado que inserções e remoções e acessos ocorrem sempre no topo da pilha. Por outro lado, a fila não é uma estrutura linear, dado que remoções ocorrem no final da fila e inserções ocorrem no início.

RESOLUÇÃO
A resposta correta é: Na implementação de uma pilha, precisamos de uma variável para indicar a cabeça (topo) da pilha, local onde ocorrerão as inserções e remoções. Na implementação de uma fila, pelo contrário, precisamos de duas variáveis para indicar a posição da frente e de trás da fila, dado que na frente da fila removemos elementos e atrás da fila inserimos elementos.

Justificativa
A) O comportamento de uma pilha é semelhante ao comportamento do botão "desfazer" de editores de texto, em que, ao acionar o botão, o último caractere inserido é o primeiro a ser removido. Uma fila, pelo contrário, tem comportamento semelhante ao botão "refazer" no mesmo editor de texto, em que supondo que clicamos no botão "desfazer" três vezes para apagar três caracteres (A, B e C, nessa ordem, por exemplo), ao clicar no "refazer", o primeiro elemento "desfeito" será o primeiro refeito (o caractere A será inserido novamente).

--- Incorreto, dado que o botão refazer mencionado na alternativa também deveria utilizar uma estrutura de pilha, e não de fila. O primeiro elemento desfeito em uma sequência deve ser o último a ser "refeito". A alternativa afirma o contrário disso. Por exemplo, se desfazemos A, B e C, nessa ordem, o primeiro a ser refeito deve ser o caractere C, e não o A, como afirma a alternativa.

B) Na implementação de uma pilha, precisamos de uma variável para indicar a cabeça (topo) da pilha, local onde ocorrerão as inserções e remoções. Na implementação de uma fila, pelo contrário, precisamos de duas variáveis para indicar a posição da frente e de trás da fila, dado que na frente da fila removemos elementos e atrás da fila inserimos elementos.

--- Correto.

C) Com uma pilha, implementamos situações em que precisamos garantir acesso justo a um recurso compartilhado, uma pilha de impressão, por exemplo, em que documentos de vários usuários são enviados para a impressora.

--- Incorreto, a fila é a estrutura ideal para acesso justo a um recurso compartilhado, e não a pilha. No exemplo da impressora, se usamos uma pilha, o último elemento enviado será o primeiro a ser impresso. Nesse caso, pode um usuário enviar um documento e este nunca ser atendido, devido às contínuas demandas de usuários.

D) Com uma fila, implementamos situações em que queremos garantir que estruturas estão balanceadas, o que é muito comum em linguagens de programação. Um exemplo minimalista é a verificação de parênteses, colchetes e chaves aninhados em uma string.

--- Incorreto, a pilha é a estrutura ideal para garantir que estruturas estão balanceadas, como o caso da verificação de parênteses, colchetes e chaves aninhados. A verificação ocorreria inserindo na pilha

as estruturas de abertura e, no caso de encontrar uma estrutura de fechamento, verificamos se o elemento no topo da pilha pode ser pareado com a estrutura de fechamento, consumindo o topo da pilha se isso ocorrer.

E) Uma pilha é uma estrutura linear, dado que inserções e remoções e acessos ocorrem sempre no topo da pilha. Por outro lado, a fila não é uma estrutura linear, dado que remoções ocorrem no final da fila e inserções ocorrem no início.

--- Incorreto, a alternativa interpreta incorretamente o conceito de estrutura linear, sendo que tanto pilha quanto fila são estruturas lineares. Note que uma estrutura linear é aquela em que um elemento tem no máximo um predecessor e um sucessor. Uma fila possui essa estrutura, dado que cada elemento possui no máximo um elemento antes dele na fila, e um elemento depois dele.

Questão 1.3

Sobre tabela hash, assinale a alternativa correta.

a) Em uma tabela hash, temos uma função de espalhamento que mapeia uma determinada chave para um endereço de memória. Do ponto de vista teórico, isso permitiria obter o conteúdo desejado acessando este endereço de memória. Entretanto, do ponto de vista prático, não é possível converter chaves em endereço de memória, o que impede a implementação de tabelas hash.
b) Em um cenário ideal sem colisões, implementar uma tabela se resumiria a criar uma função de espalhamento para indicar em qual posição de memória um elemento seria alocado. Em um cenário prático, a quantidade de colisões normalmente força o desenvolvedor a assumir que todos os elementos serão mapeados para uma mesma posição de memória. Por isso, costuma-se dizer que buscas em tabela hash não são melhores que buscas sequenciais.
c) Ao implementar uma tabela hash internamente como um vetor e assumindo um cenário com colisões, o tratamento dessas colisões pode usar um espaço de memória adicional ou pode usar o espaço de memória do próprio vetor. Um exemplo de tratamento de colisões que usa um espaço de memória adicional é o encadeamento separado, e um exemplo de tratamento de colisões que usa o espaço do próprio vetor é o teste linear.
d) Para tratar colisões, podemos usar uma estrutura interna como um vetor com tamanho muito maior do que o número de elementos que queremos adicionar. Como o vetor é muito grande, colisões não existirão e será possível gerar algoritmos mais simplificados.
e) As tabelas hash são estruturas muito úteis para fazer inserção, remoção e busca de maneira eficiente. No caso da implementação interna como um vetor, mesmo com o tratamento usando teste linear que pode formar clusters de números um do lado do outro, é possível fazer uma busca eficiente usando o algoritmo de busca binária.

RESOLUÇÃO

A resposta correta é: Ao implementar uma tabela hash internamente como um vetor e assumindo um cenário com colisões, o tratamento dessas colisões pode usar um espaço de memória adicional ou pode usar o próprio espaço de memória de vetor. Um exemplo de tratamento de colisões que usa um espaço de memória adicional é o encadeamento separado, e um exemplo de tratamento de colisões que usa o espaço do próprio vetor é o teste linear.

Justificativa

A) Em uma tabela hash, temos uma função de espalhamento que mapeia uma determinada chave para um endereço de memória. Do ponto de vista teórico, isso permitiria obter o conteúdo desejado acessando este endereço de memória. Entretanto, do ponto de vista prático, não é possível converter chaves em endereço de memória, impedindo a implementação de tabelas hash.

--- Incorreto, as tabelas hash não são apenas curiosidade teórica, mas uma técnica que já vem por padrão implementada em várias linguagens de programação. Dito isso, é possível sim converter chaves em endereço de memória, sendo que algumas funções que fazem isso foram mostradas em aula.

B) Em um cenário ideal sem colisões, implementar uma tabela se resumiria a criar uma função de espalhamento para indicar em qual posição de memória um elemento seria alocado. Em um cenário prático, a quantidade de colisões normalmente força o desenvolvedor a assumir que todos os elementos serão mapeados para uma mesma posição de memória, por isso costuma-se dizer que buscas em tabela hash não são melhores que buscas sequenciais.

--- Incorreto, o desenvolvedor irá fazer um estudo para gerar funções de espalhamento que minimizem o número de colisões. Além disso, irá lançar mão de algumas técnicas que irão diminuir a ocorrência de colisões. Feito o estudo, o número de colisões deve ser baixo e limitado por uma constante. Em nenhuma hipótese o desenvolvedor assumirá que criou uma função que mapeará todos os elementos para a mesma posição de memória.

C) Ao implementar uma tabela hash internamente como um vetor e assumindo um cenário com colisões, o tratamento dessas colisões pode usar um espaço de memória adicional ou pode usar o próprio espaço de memória de vetor. Um exemplo de tratamento de colisões que usa um espaço de memória adicional é o encadeamento separado, e um exemplo de tratamento de colisões que usa o espaço do próprio vetor é o teste linear.

--- Correto.

D) Para tratar colisões, podemos usar uma estrutura interna como um vetor com tamanho muito maior do que o número de elementos que queremos adicionar. Como o vetor é muito grande, colisões não existirão e será possível gerar algoritmos mais simplificados.

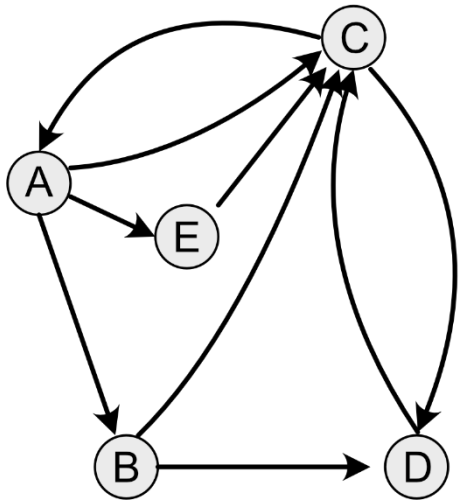
--- Incorreto, por mais que usemos um vetor muito grande, não podemos assumir de maneira categórica que "colisões não existirão", como afirma a questão. Note que evitar as colisões não é o mesmo que tratar as colisões. Para o algoritmo ser correto, é preciso que as colisões sejam tratadas.

E) As tabelas hash são estruturas muito úteis para fazer inserção, remoção e busca de maneira eficiente. No caso da implementação interna como um vetor, mesmo com o tratamento usando teste linear que pode formar clusters de números um do lado do outro, é possível fazer uma busca eficiente usando o algoritmo de busca binária.

--- Incorreto, a alternativa menciona uma busca binária que não se aplica ao contexto. Em nenhum momento ordenamos o vetor interno que representa a hash de modo a ser minimamente possível imaginar uma busca binária.

Questão 1.4

Sobre o grafo apresentado a seguir, assinale a alternativa verdadeira.



a) Uma possível busca em largura, iniciando no vértice B, poderia visitar os vértices na seguinte ordem: B, C, D, A, E.
b) Uma possível busca em profundidade, iniciando no vértice C, poderia visitar os vértices na seguinte ordem: C, A, E, D, B.
c) Uma busca em largura neste grafo utilizando o algoritmo visto em aula, iria utilizar uma pilha para gerenciar os backtrackings.
d) Uma possível busca em largura, iniciando no vértice E, poderia visitar os vértices na seguinte ordem: E, C, A, B, D.
e) Uma possível busca em profundidade, iniciando no vértice D, poderia visitar os vértices na seguinte ordem: D, C, E, A, B.

RESOLUÇÃO

A resposta correta é: Uma possível busca em largura, iniciando no vértice B, poderia visitar os vértices na seguinte ordem: B, C, D, A, E.

Justificativa

A) Uma possível busca em largura, iniciando no vértice B, poderia visitar os vértices na seguinte ordem: B, C, D, A, E.

* Correto, na busca em largura, os vértices são visitados camada por camada, ao contrário da busca em profundidade, em que a estratégia consiste em se aprofundar no grafo sempre que possível. A alternativa correta mostra a estratégia em largura sendo usada. Iniciamos com o nó B (source), seguido pelos seus filhos C e D, o que conclui a primeira camada. A camada seguinte consiste no filho

de C, que é o nó A. Como D não possui filho ainda por visitar, seguimos para a visita do filho de A, que no caso é E.

B) Uma possível busca em profundidade, iniciando no vértice C, poderia visitar os vértices na seguinte ordem: C, A, E, D, B.

* Incorreto, a estratégia consiste em se aprofundar no grafo sempre que possível. Note que iniciamos no nó C e visitamos os nós A e E, em que o aprofundamento fica evidente. Entretanto, a partir de E não temos quem visitar, o que força um backtracking. O ponto mais próximo de backtracking é retornar para o vértice A, que permitirá a visita de B. Nesse caso, não poderemos visitar D antes de visitar B, como mostrado na alternativa.

C) Uma busca em largura neste grafo utilizando o algoritmo visto em aula, iria utilizar uma pilha para gerenciar os backtrackings.

* Incorreto, uma busca em largura utilizando o algoritmo visto em aula usaria uma fila para organizar a ordem de visitação e não uma pilha, dado que na busca em largura não usamos realmente o conceito de backtracking.

D) Uma possível busca em largura, iniciando no vértice E, poderia visitar os vértices na seguinte ordem: E, C, A, B, D.

* Uma busca em largura iniciando em E visitará C imediatamente, dado que C é o único sucessor de E. Como C tem dois sucessores, A e D, esses sucessores seriam visitados em sequência. Entretanto, a alternativa indica que B será visitado em algum momento entre A e D, o que viola a ordem de visitação da busca em largura.

E) Uma possível busca em profundidade, iniciando no vértice D, poderia visitar os vértices na seguinte ordem: D, C, E, A, B.

* Incorreto, a busca em profundidade poderia realmente percorrer o caminho de D até C. Entretanto, não existe caminho entre C e E, sendo que a busca em profundidade deveria percorrer o caminho entre C e A que existe.

QUESTÕES DISSERTATIVAS

Questão 2

Dado o trecho de código mostrado a seguir:

```
1  #include <cstddef>
2  #include <iostream>
3
4  using namespace std;
5
6  void funcao1(int*& p){
7    p = new int;
8    *p = 8;
9  }
10 void funcao2(int* p) {
11   p = new int;
12   *p = 8;
13 }
14 int main(){
15   int* a = new int;
16   int* b = new int;
17
18   *a = 2;
19   *b = 3;
20
21   funcao1(a);
22   funcao2(b);
23
24   cout << *a << endl;
25   cout << *b << endl;
26 }
```

Apresente qual a saída do programa fornecida nas linhas 24 e 25. Dada essa saída, explique as movimentações que ocorrem na memória nas linhas 6-9 e 10-13 que explicam o comportamento na saída.

RESOLUÇÃO

Na saída padrão, aparecem os números:

8
3

Isso significa que *funcao1* está alterando o valor do ponteiro a, mas *funcao2* não está alterando o valor do ponteiro b.

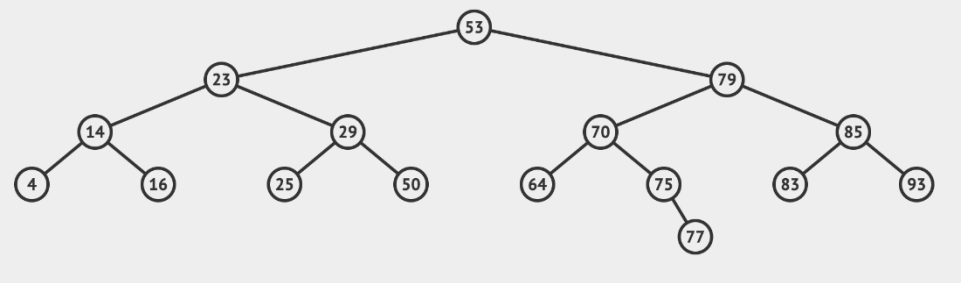
Observando *funcao1*, notamos que o ponteiro foi passado por referência. Nesse caso, a linha 7 faz com que o ponteiro aponte para uma segunda posição de memória, sendo que essa mudança de endereço surte efeito também fora da função. Isso quer dizer que a linha 8 atualizará o valor da variável local "p" dentro da função, e da variável "a" fora da função.

Observando *funcao2*, notamos que o ponteiro foi passado por valor. Nesse caso, a linha 11 faz com que a variável local aponte para uma nova região. Entretanto, fora da função, o ponteiro "b" continuará apontando para a região de memória original. Dito isso, o que fizermos dentro da *funcao2* com a variável local "p" não surtirá efeito fora da função.

Esses dois fatos explicam o motivo de "a" ter mudado de valor, mas "b" ter permanecido com o valor original.

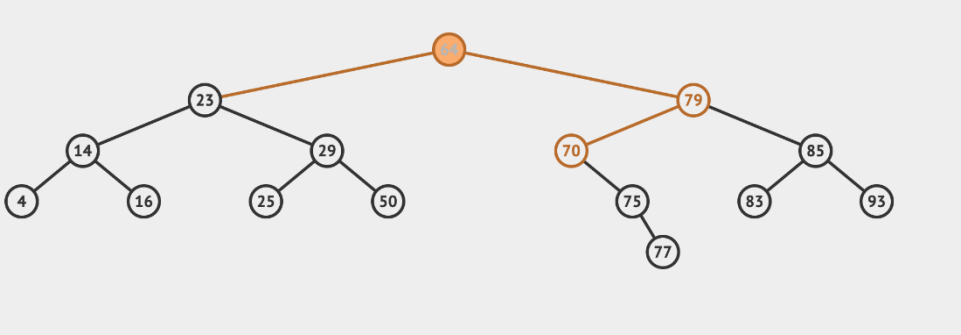
Rubricas | critérios de correção
Fornecer 40% da questão para quem acertou a saída. Além disso, 30% para quem explicou corretamente *funcao1* e 30% para quem explicou corretamente *funcao2*.

Questão 3
Dada a árvore AVL a seguir:

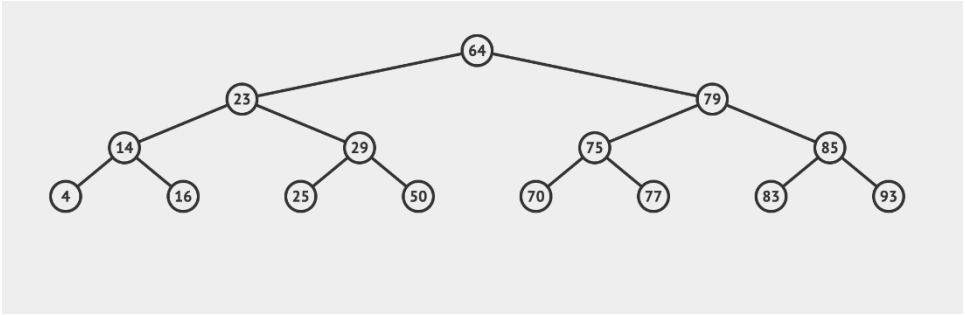


Faça a remoção do elemento 53. Na remoção, use o algoritmo que utiliza o sucessor lógico quando um determinado nó possui dois filhos. Nesta questão, mostre o resultado da remoção antes e depois da rotação necessária para fazer o balanceamento da árvore AVL.

RESOLUÇÃO
Como o elemento 53 possui dois filhos, então não podemos simplesmente removê-lo, precisaremos substituir pelo sucessor lógico, como pedido pela questão. O sucessor lógico é o elemento 64. A busca pelo sucessor lógico começa na subárvore da direita, passando pelos nós 79 e 70, como mostra a figura a seguir. Ao achar o sucessor lógico (elemento mais à esquerda da subárvore da direita), colocamos no lugar do 53.



Entretanto, o nó 70 ficou desbalanceado, exigindo uma rotação simples à esquerda para tornar a árvore novamente uma árvore AVL.



Rubricas | critérios de correção

Sugiro fornecer 50% da nota para quem acertar a árvore antes da rotação e 50% da nota para quem acertar a árvore após a rotação.