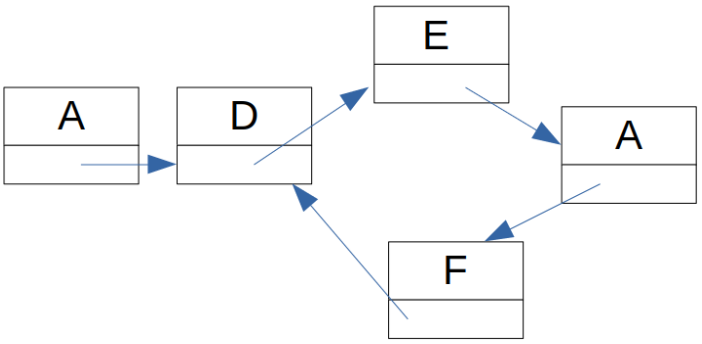
Universidade Federal da Bahia - UFBA
Instituto de Computação - IC
Departamento de Ciência da Computação - DCC
MATA40 – Estrutura de Dados e Aplicações
Prof. Ricardo Rios

- A prova vale 10,0 pontos e todas as questões têm o mesmo peso.
- Interpretação faz parte da avaliação. Para responder uma questão, elabore suas hipóteses e desenvolva a solução.
- Conversas entre alunos não são permitidas. Caso ocorra, as duas provas serão anuladas e os envolvidos ficarão com nota 0,0.

1 Implemente uma função que detecta ciclos em uma lista encadeada. Exemplo:



Neste exemplo, o resultado seria "Ciclo Detectado". Sugestão: Utilize dois ponteiros, com diferentes velocidades de movimentação.

2 Utilizando operações primitivas, calcule a complexidade do seu algoritmo.

3 Discuta a complexidade calculada anteriormente em termos de O (.) [funções como limitantes superiores] e Ω (.) [funções como limitantes inferiores].

4 Suponha que você precise implementar um compilador. Utilizando uma pilha, verifique se expressões são válidas considerando "(", ")", "[", "]", "{" e "}". Exemplo:
   a (a + b) + (c + d): Válida
   b ((a + b) + (c + d): Inválida
   c ((a + b) + (c + d)]: inválida
   d [{a + b} + (c + d)]: Válida

5 Você deve implementar o algoritmo Round Robin (RR) de escalonamento em um Sistema Operacional (SO). Esse algoritmo executa da seguinte maneira:
   a Cada processo no sistema tem um custo de execução representado por Unidades de Processamento (UP);

b Cada novo processo é enviado para uma fila de "pronto" que contém processos que estão aptos para serem executados;

c A CPU libera um limite de tempo (quantum) para cada processo. Se um processo está executando e precisa de mais UP que o quantum liberado pela CPU, ele executa até o limite e depois volta para o final da fila de pronto com as UPs necessárias para concluir sua tarefa.

Seu código deve ler a seguinte entrada:

```
3
5 15
1 2 3
10 1 20
```

A primeira linha ("3") informa quantas linhas em sequência devem ser lidas. A segunda linha contém duas informações importantes: "5" - o quantum da CPU e "15" o máximo de UP que seu código vai processar. A próxima linha contém os IDs dos processos. A última linha contém todas as UPs necessárias por processo.

Nesse caso, o processo "1", que chegou primeiro, começa a executar. Esse processo ocupa a CPU por um quantum de "5" UP. Depois, ele volta para o final da fila com "5" UP.

O processo "2" precisou apenas de um "1" UP. Em seguida, ele sai da CPU e não volta para a fila porque não tem UP restante.

Ao final do máximo de UP, seu código deve imprimir a fila atual e as respectivas UPs restantes:

```
3 1
15 1
```

Importante:

1 - Não é permitido usar funções existentes do Python para gerenciamento de listas (ex.: lista = [], lista.append, ...), pilhas ou qualquer outra estrutura. Respostas que as utilizarem serão zeradas!

2 - Utilize sua implementação do TAD Fila.

6 Considere uma implementação de lista encadeada com cabeça e sem cauda que armazena valores reais. Implemente a função **conc**, que concatena duas listas L1 e L2 passadas como parâmetro, intercalando seus elementos. Ao final, imprima o resultado na tela.

Exemplo:

L1: [1.2 3.5 9.78]
L2: [5 2.9]
Saída: [1.2 5 3.5 2.9 9.78]

7 Implemente uma lista duplamente encadeada com elementos que armazenem valores inteiros. Além disso, implemente uma função que recebe um vetor de inteiros e os insere na lista de maneira ordenada. Observação: seu algoritmo não pode recomeçar a partir da cabeça a busca pela posição correta após cada inserção na lista, exceto na primeira inserção. Ex.:

Lista atual: [0 1 3 5 6]
Inserir vetor: [4 2 7]

Para inserir o elemento 4, a busca deve começar a partir do elemento 0. Ao inserir o elemento 4 após o 3 [0 1 3 4 5 6], o algoritmo não deve voltar para o primeiro elemento 0 para buscar a posição do valor 2 que será inserido na sequência. O algoritmo deve fazer essa inserção usando o encadeamento para trás.

8 A turma de ED estava “batendo um baba” na praia quando um novo estudante chegou e disse “15 minutos, 2 gols!”. Após 15 minutos, o time que perdeu de 1x0 precisou escolher alguém para sair. Para decidir qual estudante deveria dar o lugar, todos do time perdedor fizeram um círculo e cada um disse um número aleatório de 0-5. Somaram, então, os números dos estudantes e, em seguida, escolheram alguém do círculo para começar a contagem. A partir desse estudante, no sentido horário, começaram a contar até chegar na soma total. A pessoa, na qual a contagem finalizou, deveria dar o lugar ao estudante que chegou.

Exemplo de entrada:
[5 0 2 0 4 3] *//números aleatórios escolhidos pelos estudantes time perdedor*
3 *//posição que deve iniciar a contagem. Posições: [0 1 2 3 4 5]*

Execução
[0 1 2 3 4 5]
[1 2 3]
[4 5 6 7 8 9]
[10 11 12 13 14]

Saída:
4 *// a pessoa que estava na posição 4 deve dar lugar ao novo estudante*

9 Implemente uma lista duplamente encadeada circular e duas funções que insere (operação 1) e remove (operação 0) elementos em uma determinada posição específica.

Exemplo:
// [operação] [valor, se operação 1; posição, se operação 0] [posição, se operação 1, vazio se operação 0]
1 1 0
1 2 1
1 5 2
1 3 3
0 2
1 7 2

Lista antes da remoção:
[1⇋ 2 ⇋ 5 ⇋3]
↑_____|

Lista após a remoção:

[1⇋ 2 ⇋ 3]
↑_____|

Lista após a nova inserção:

[1⇋ 2 ⇋ 7 ⇋3]
↑_____|

10 Implemente uma pilha e suas funções de empilhar e desempilhar. Em seguida, implemente uma função que retorne uma a pilha cujo topo contém o elemento que estava na base. Utilize uma pilha auxiliar para resolver o problema. Exemplo:

Entrada	*Saida*
[4]	[23]
[17]	[4]
[9]	[17]
[23]	[9]

11 Implemente uma heap que receba entradas no seguinte formato:

```
11
1 10
1 5
1 9
1 8
1 1
1 3
1 2
0 1
1 4
0 2
1 7
```

A primeira linha informa quantas operações serão realizadas pela sua função. As linhas seguintes são compostas de 2 números: 0 indicando a remoção do elemento com a chave apresentada na sequência (e.g. “0 1” remove o elemento com chave “1”) e 1 a inserção de um elemento com chave indicada na sequência (e.g. “1 10” insere o elemento com chave “10”). Por fim, seu código deve imprimir a heap usando a função “em-ordem”.

12 Usando o algoritmo da questão 01, desenvolva uma função que receba uma entrada no seguinte formato:

```
12
1 10
1 5
1 9
1 8
1 1
1 3
1 2
0 1
1 4
0 2
1 7
```

2 9

A primeira linha informa quantas operações serão realizadas pela sua função. As linhas seguintes são compostas de 2 números: 0 indicando a remoção do elemento com a chave apresentada na sequência (e.g. “0 1” remove o elemento com chave “1”), 1 a inserção de um elemento com chave indicada na sequência (e.g. “1 10” insere o elemento com chave “10”) e 2 a busca de um elemento com chave indicada na sequência (e.g. “2 9” busca o elemento com chave “9”). Utilize a busca em largura e no final apresente o caminho realizado pela busca, i.e., as chaves dos nós visitados.

13 Repita o exercício anterior, mas implementando a busca em profundidade.

14 A primeira linha informa quantas operações serão realizadas pela sua função. As linhas seguintes são compostas de 2 números: 0 indicando a remoção do elemento com a chave apresentada na sequência (e.g. “0 1” remove o elemento com chave “1”), 1 a inserção de um elemento com chave indicada na sequência (e.g. “1 10” insere o elemento com chave “10”) e 2 a busca de um elemento com chave indicada na sequência (e.g. “2 9” busca o elemento com chave “9”). Utilize a busca em largura e no final apresente o caminho realizado pela busca, i.e., as chaves dos nós visitados.

15 Implemente a estrutura de dados lista encadeada simples e execute as seguintes tarefas:

- a Ler dois vetores (entrada) e armazená-los em listas encadeadas separadas;
- b Implementar uma função que insere a segunda lista em um determinado ponto da primeira lista;
- c Inserir a segunda lista na primeira e imprimir o resultado;
- d Implementar uma função que remove os valores repetidos de uma lista, mantendo apenas a primeira ocorrência;
- e Remover os valores repetidos da junção das listas e imprimir o resultado.

Seu código deve ler o seguinte modelo de entrada:

```
3           // Quantidade de linhas a seguir.
0 1 2 3 4 5
0 9 8 0 0
4           // Posição para inserção.
```

Seu código deve ter o seguinte modelo de saída:

```
0 1 2 3 0 9 8 0 0 4 5   // Print após inserção
0 1 2 3 9 8 4 5         // Print após remoção de valores repetidos
```

16 Implemente as estruturas de dados lista encadeada circular e Pilha, em seguida execute as seguintes tarefas:

- a Ler um vetor e armazená-lo em uma lista encadeada circular;
- b Implementar uma função que procura o próximo maior elemento da lista, remove-o e o adiciona a uma pilha que deve ser retornada;
    - i A primeira pesquisa deve começar no primeiro valor da lista (o primeiro valor do vetor recebido);

ii Ao chegar ao final da lista, você deve continuar até encontrar o valor novamente e removê-lo;
iii A partir desse momento, as novas pesquisas não necessariamente iniciarão no primeiro valor, mas sim de onde você removeu o último;
iv Você deve contar a quantidade de voltas feitas na lista e imprimir esse valor quando a lista estiver vazia;
v Você deve retornar a pilha resultante (que estará ordenada com os menores valores no topo e maiores na base).

c Você deve imprimir a pilha resultante.

Seu código deve ler o seguinte modelo de entrada:

```
1           // Quantidade de linhas a seguir.
5 0 7 2 4
```

Seu código deve ter o seguinte modelo de saída:

```
5           // Quantidade de voltas realizadas
0 2 4 5 7   // Valores nas ordem armazenada na pilha.
```

17 Implemente a estrutura de dados Pilha e execute as seguintes tarefas:

a Ler dois vetores e armazená-los em pilhas separadas;
i Os vetores deverão ser armazenados em suas respectivas pilhas com "Push(...)";
ii As pilhas resultantes resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo).

b Implementar (usando apenas pilhas) uma função que empilhe a segunda pilha na primeira sem alterar a ordem das mesmas;

c Você deve imprimir a pilha resultante.

Seu código deve ler o seguinte modelo de entrada:

```
2           // Quantidade de linhas a seguir.
1 2 3 4 5
6 7 8 9 0
```

Seu código deve ter o seguinte modelo de saída:

```
0
9
8
7
6
5
4
3
2
1
```

18 Implemente a estrutura de dados Pilha e execute as seguintes tarefas:

a Ler um vetor e armazená-lo em uma pilha;

b Implementar uma função que utiliza 3 pilhas e transfira os elementos da primeira pilha para a terceira utilizando a segunda como auxiliar;
i O vetor deve ser armazenado com "Push(...)";

ii A pilha resultante deve estar ordenada da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo).

c Você deve imprimir quantos passos foram necessários para mover a primeira pilha para a terceira.
   i Utilize apenas pilhas;
   ii A primeira pilha sempre estará ordenada (maior valor na base, menor no topo);
   iii Você nunca deve empilhar um valor maior sobre um valor menor.

Seu código deve ler o seguinte modelo de entrada:

```
1       // Quantidade de linhas a seguir
2 1
```

O que seu código deve simular:

```
1   |   |
2   |   |

1°
|   |   |
2   1   |

2°
|   |   |
|   1   2

3°
|   |   1
|   |   2
```

Seu código deve ter o seguinte modelo de saída:

```
3
```

19 Implemente as estruturas de dados Pilha e Fila; em seguida, execute as seguintes tarefas:

a Ler dois vetores e armazená-los em pilhas separadas;
   i Os vetores deverão ser armazenados em suas respectivas pilhas com "Push(...)";
   ii As pilhas resultantes resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na base, último valor no topo);
   iii Imprima as pilhas de acordo a representação abaixo (base na esquerda, topo na direita).

b Transferir os elementos da pilha para filas separadas;
   i Os vetores deverão ser armazenados em suas respectivas filas com "Append(...)";
   ii As filas resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final);
   iii Utilize pilhas para alcançar isso (dica: simule uma fila utilizando pilhas);
   iv Imprima as filas.

c Implementar uma função que recebe 2 filas e retorne uma terceira fila que intercala os valores das duas primeiras;
   i Utilize apenas filas;
   ii Valores da primeira fila primeiro;
   iii A fila maior terá seus últimos elementos no final;

iv Imprima a fila resultante.

Seu código deve ler o seguinte modelo de entrada:

```
2           // Quantidade de linhas a seguir.
0 2 4 6 8 10 12
1 3 5 7 9
```

Seu código deve ter o seguinte modelo de saída:

```
0 2 4 6 8 10 12
1 3 5 7 9
0 2 4 6 8 10 12
1 3 5 7 9
0 1 2 3 4 5 6 7 8 9 10 12
```

20 Na última semana, a fila de entrada de um certo evento ficou extraordinariamente grande. Os organizadores perceberam que algumas pessoas necessárias para o evento acontecer estavam presas nesta fila. Além disso, perceberam também uma certa quantidade de idosos e grávidas ao longo da mesma. Para evitar que essa situação se repita, a equipe que organiza eventos como esse decidiu pedir à turma de estrutura de dados da UFBA para escrever um programa para auxiliá-los. Eles preferiram que esse programa fosse acionado manualmente toda vez que eles julgassem que a fila precisasse ser organizada. O programa deve dividir a fila em duas:

a Sem prioridade;

b Com prioridade: a partir de 60 anos, até 5 anos, grávidas e palestrantes - Grávidas e palestrantes serão especificados pela posição na fila.

Seu código deve:

a Implementar a estrutura de dados fila;

b Ler um vetor e armazená-lo em uma fila;

i O vetor deve ser armazenado na fila com "Append(...)";

ii A fila resultante deve estar ordenada da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final).

c Transferir os elementos dentro das especificações para uma fila separada;

i Todas as operações de transferência de fila deverão ser realizadas utilizando filas: Pop(...)" para remover da fila inicial e "Append(...)" para adicionar na fila auxiliar/final;

ii As filas resultantes devem estar ordenadas da mesma forma que o vetor de entrada (primeiro valor na frente, último valor no final);

d Imprima as filas.

Seu código deve ler o seguinte modelo de entrada:

```
2           // Quantidade de linhas a seguir.
50 65 30 33 11 3 25 39 17 22 27 26 44 28 53 28 20 70 58 27 33 36 49 17  // Vetor de idades
7 8 14      // Vetor com a posição das grávidas e palestrantes na fila
```

Seu código deve ter o seguinte modelo de saída:

```
50 30 33 11 17 22 27 26 44 53 28 20 58 27 33 36 49 17 // Fila original sem os preferenciais
65 3 25 39 28 70            // Fila preferencial
```

21 Implemente as estruturas de dados pilha, fila e árvore; em seguida, execute:

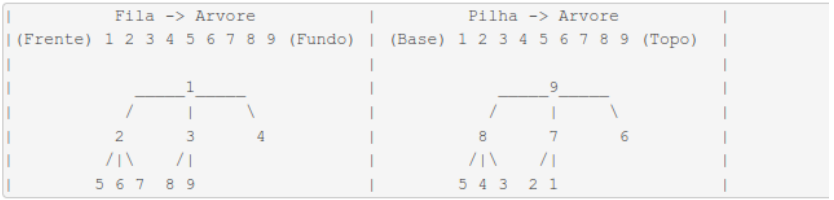
a Ler um vetor e armazenar uma cópia do mesmo em uma pilha e uma cópia em uma fila;
   i Os vetores deverão ser armazenados em suas respectivas estruturas com seu método de adicionar associado;
   ii As estruturas devem estar ordenadas da mesma forma que os vetores de entrada (primeiro/base/frente -> último/topo/final).

b Implementar uma função que recebe uma fila, uma pilha e cria/manipula duas árvores de grau 3 da seguinte maneira:
   i As estruturas devem estar ordenadas da mesma forma que os vetores de entrada (primeiro/base/frente -> último/topo/final).
   ii A primeira deve ser preenchida com a fila, a segunda com a pilha;
   iii Limpe as árvores da esquerda para a direita, priorizando sempre remover a folha mais profunda à esquerda (RemoveLeftmostLeaf);
   iv Imprima o conteúdo à medida que o remove (DisplayAndClearTree + RemoveLeftmostLeaf);

Seu código deve ler o seguinte modelo de entrada:

```
1           // Quantidade de linhas a seguir.
1 2 3 4 5 6 7 8 9  // Vetor que deve ser transformado em uma fila e em uma pilha
```

Processo:



Seu código deve ter o seguinte modelo de saída:

```
5 6 7 2 8 9 3 4 1    // Saída obtida ao esvaziar a árvore criada com a fila.
5 4 3 8 2 1 7 6 9    // Saída obtida ao esvaziar a árvore criada com a pilha.
```

```
^              ^
|              |_ Último elemento removido é a raiz.
|_ Primeiro elemento removido é o mais à esquerda.
```

22 Implemente uma estrutura de dados árvore binária completa e execute as seguintes tarefas:

a Ler um vetor e armazená-lo em uma árvore binária;
   i Todos os nós de um dado nível devem ser preenchidos (da esquerda para a direita) antes de preencher o nível seguinte;
   ii Os elementos devem ser preenchidos na mesma ordem do vetor de entrada.

b Implementar uma função que recebe uma árvore e dois números para pesquisa;
   i Considere que não haverá valores repetidos;
   ii Caso ambos elementos sejam encontrados, imprima a diferença de nível deles;
   iii Caso apenas um seja encontrado, imprima a altura do mesmo;
   iv Caso nenhum seja encontrado, imprima a profundidade da árvore.

c Seu código deve receber uma árvore e "N = linhas - 1" pares de valores para pesquisa.

Seu código deve ler o seguinte modelo de entrada:

```
5                          // Quantidade de linhas
1 2 3 4 5 6 7 8 9 10       // Vetor de entrada para a árvore
1 2                        // Pares de valores para pesquisa
1 11
3 10
0 15
```

```
Árvore Binária Completa:
|                ____1____             |
|               /         \            |
|              2           3           |
|           /     \     /     \        |
|          4       5   6       7       |
|        /   \    /                    |
|       8     9  10                    |
```

Seu código deve ter o seguinte modelo de saída:

```
1
3
2
3
```

23 Implemente a estrutura de dados lista duplamente encadeada não-circular e execute as seguintes tarefas:

a Ler dois vetores (entrada) e armazená-los em listas separadas;

b Implementar uma função que procura os elementos da segunda lista na primeira e retorna uma terceira lista composta pelo caminho percorrido nessa procura;

i Assuma que a primeira lista sempre estará ordenada;

ii Seu algoritmo deve procurar através do melhor caminho sempre;

iii Você deve armazenar os valores toda vez que houver uma comparação.

c Você deve imprimir a lista resultante com o caminho percorrido.

Seu código deve ler o seguinte modelo de entrada:

```
2           // Quantidade de linhas a seguir.
3 4 5 7 8
1 10 6 4
```

Seu código deve ter o seguinte modelo de saída:

```
3 3 4 5 7 8 8 7 5 5 4
```