Exercícios para o TP2:

1. Implemente um procedimento eficiente para a multiplicação de matrizes usando OpenMP.

2. Escreva uma rotina para calcular o produto escalar de dois vetores utilizando rotinas do OpenMP. Considere cada vetor com 1000 posições. Utilize a cláusula de redução.

3. Construa um servidor simples que atenda múltiplas requisições de forma assíncrona.

4. Refaça a questão 2 usando Python asyncio.

5. Apresente uma implementação do algoritmo QuickSort.

6. Apresente uma implementação do algoritmo QuickSelect.

7. Faça uma análise da complexidade do algoritmo QuickSort.

8. Faça uma análise da complexidade do algoritmo QuickSelect.

9. Usando o arquivo de teste construído no TP1, demonstre o funcionamento dos algoritmos implementados na questão 5 e 6.

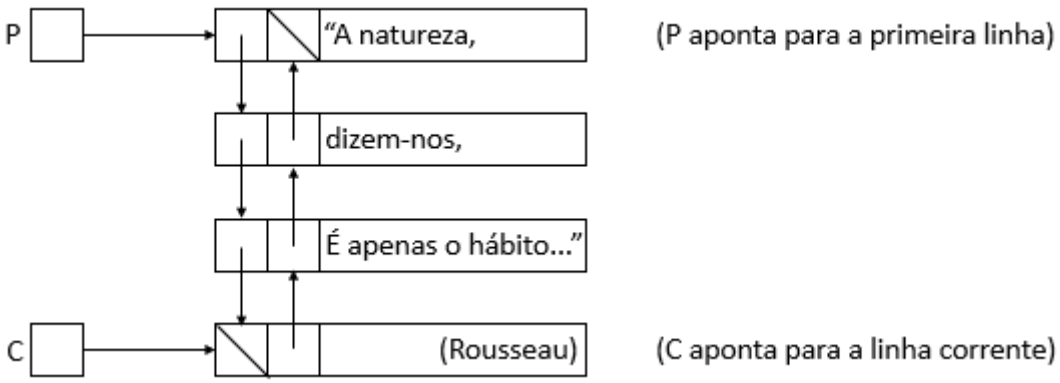
10. Implemente uma classe ListaEncadeada com métodos para manutenção da lista, tais como inserir item, remover item, buscar item, imprimir lista e ordenar lista.

11. Implemente uma fila circular usando encadeamento. Seu programa deve fornecer métodos para manutenção da fila.

12. Editores de linha podem ser eficientemente implementados através do uso de listas duplamente encadeadas. Esta estrutura permite que linhas de texto sejam facilmente inseridas, excluídas, movidas ou copiadas. Por exemplo, considere o texto a seguir:

    "*A natureza*,
    dizem-nos,
    é apenas o hábito..."
    (Rousseau)

    Usando uma lista duplamente encadeada, podemos representa-lo da seguinte maneira:



Desenvolva um programa que use listas duplamente encadeadas para fazer representações de texto como ilustrado acima.

13. Baseando-se na estrutura sugerida na Questão 12, desenvolva um editor de textos, que ofereça ao usuário os seguintes comandos:

I <n>:
Entra no modo de inserção, incluindo as novas linhas digitadas após a linha <n>. Caso o valor <n> não seja fornecido, a inserção ocorrerá após a linha corrente.

E <i>,<f>:
Exclui as linhas da posição <i> até a posição <f>. Caso os parâmetros não sejam fornecidos, apenas a linha corrente deverá ser excluída do texto.

D <i>,<f>,<p>:
Duplica o bloco de linhas <i> até <f> imediatamente após a linha <p>.

L <i>,<f>:
Lista o bloco de linhas <i> até <f>. Caso nenhum parâmetro seja fornecido, todo o texto é listado.

C <arq>,<n>:
Carrega as linhas do arquivo <arq> para a posição imediatamente após a linha <n>. Caso <n> não seja fornecida, ela é assumida como sendo a linha corrente.

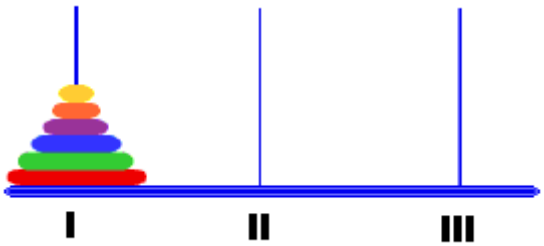
S <arq>,<i>,<f>:
Salva no arquivo <arq> o conjunto de linhas <i> até <f>. Se <i> e <f> não são fornecidos, todo o texto é salvo.

A <n>:
Permite alterar o texto na linha <n>.

F:
Finaliza a execução

14. O Problema das Torres de Hanoi é um problema clássico da computação. Que consiste em movimentar todos os discos de um poste **I** para outro poste (o poste **III**, por exemplo) sem que, em momento algum, um disco maior fique sobre um

disco menor (vide a figura abaixo). Apresente uma solução recursiva para o Problema da Torre de Hanoi.



15. Implemente as versões recursivas para os procedimentos para atravessamento em árvores binárias em pré-ordem, pós-ordem e em ordem simétrica.

16. Implemente o algoritmo de ordenação MergeSort usando recursividade.

17. Escreva uma função recursiva que imprima uma régua de ordem n no intervalo $[0 .. 2^n]$. O traço no ponto médio da régua deve ter comprimento n; os traços nos pontos médios dos subintervalos superior e inferior devem ter comprimento $n-1$; e assim por diante. A figura mostra uma régua de ordem 4.

```
.
. -
. --
. -
. ---
. -
. --
. -
. ----
. -
. --
. -
. ---
. -
. --
. -
.
```

18. Escreva um procedimento para busca em árvore binária usando OpenMP.

19. O Algoritmo de Dijkstra é um algoritmo clássico para a determinação de caminhos em grafos. Apresente uma versão deste algoritmo usando OpenMP.

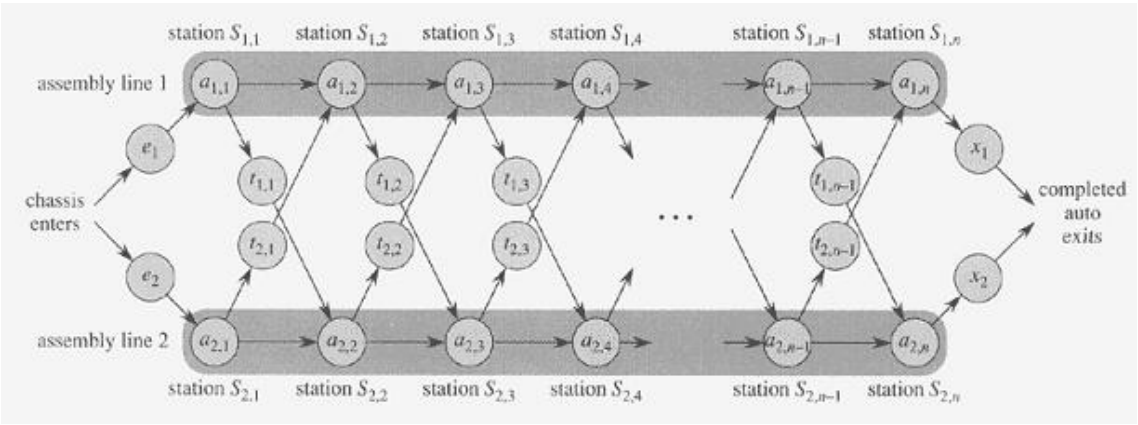
20. Escreva uma rotina para a geração do Triângulo de Pascal usando OpenMP.

21. Escreva uma rotina para a geração da Sequência de Fibonacci usando OpenMP.

22. Considere a versão clássica do Problema da Linha de Montagem:

- Há duas linhas de montagem, cada uma com n estações
- A *j*-ésima estação na linha *i* é denotada por $S_{i,j}$ e o tempo de montagem nessa estação é $a_{i,j}$.
- Um chassi entra na fábrica e vai para a linha i (onde i = 1 ou 2), demorando o tempo e_i.
- Depois de passar pela *j*-ésima estação em uma linha, o chassi vai para a (*j* + 1)-ésima estação de uma das linhas.
- Não há custo de transferência se ele ficar na mesma linha, mas demora o tempo $t_{i,j}$ para transferir a peça para a outra linha após a estação $S_{i,j}$.
- Depois de sair da *n*-ésima estação em uma linha, decorre o tempo x_i para a peça estar pronta.

O problema consiste em determinar que estações escolher na linha 1 e quais escolher na linha 2 de forma a minimizar o tempo total de passagem de uma peça pela fábrica. A figura abaixo ilustra o problema.



Para a elaboração de uma solução com programação dinâmica, é necessário buscar pela solução recursiva. Apresente a solução recursiva do problema clássico da linha de montagem.

23. Desenvolva a solução em Programação Dinâmica para o problema clássico da Linha de Montagem.

24. Para uma versão do problema da linha de montagem com **três** linhas de montagem, como ficaria a solução recursiva do problema?

25. Desenvolva a solução em Programação Dinâmica para a versão apresentada na questão 24.