Grupo 1: Heap Binária (Leitura, Busca, Inserção, Deleção)

Exercício 1.1 – Implementar criação e leitura de heap

- Crie uma função que receba uma lista de inteiros e a transforme em uma heap binária.
- Em seguida, faça uma função que exiba a heap em um formato de lista.

Exercício 1.2 – Inserir elemento na heap

- Dada uma heap já existente, implemente uma função para inserir um novo elemento e manter a estrutura de heap válida.
- Mostre o estado da heap antes e depois da inserção.

Exercício 1.3 – Buscar elemento na heap

- Implemente uma função que busque um determinado valor na heap e retorne True ou False conforme a existência do valor.
- A função deve percorrer a estrutura da heap de forma eficiente.

Exercício 1.4 – Remover o menor elemento (ou maior, se for max-heap)

- Implemente uma função que remova o elemento raiz (menor em min-heap ou maior em max-heap) e reestruture a heap, mantendo a propriedade de heap.
- Mostre o estado da heap antes e depois da remoção.

Exemplo:

Considere uma lista [5, 2, 3, 7, 1].

1. Ao transformá-la em uma min-heap, a raiz deve ser o menor elemento (1).
2. Se inserirmos o valor 0, ele deve se tornar a nova raiz.
3. Ao buscar o elemento 7, a função retorna True se ele ainda estiver na heap.
4. Ao remover o menor elemento, 0, a raiz passa a ser o próximo menor, reordenando a heap.

Grupo 2 – Autocomplete com Trie

Exercício 2.1 – Criar um Trie e inserir palavras

- Crie uma classe Trie que armazene palavras de forma hierárquica (caractere a caractere).
- Insira manualmente uma lista de palavras no Trie.

Exercício 2.2 – Busca simples em Trie

- Implemente um método que, dada uma palavra, verifique se ela foi inserida no Trie.

Exercício 2.3 – Autocomplete básico

- Implemente um método que, dada um prefixo, retorne todas as palavras do Trie que começam com esse prefixo.

Exercício 2.4 – Remoção de uma palavra do Trie

- Implemente um método que remova uma palavra específica do Trie, ajustando os nós conforme necessário (removendo nós que se tornem desnecessários).

Exemplo

- Se o Trie contiver as palavras: ["casa", "casamento", "casulo", "cachorro"]
  - Busca de "casa" deve retornar True, pois a palavra existe.
  - Autocomplete de "cas" deve retornar ["casa", "casamento", "casulo"].
  - Ao remover "casa", o Trie deve continuar a reconhecer "casamento" e "casulo", mas não "casa".

Grupo 3 – Busca em Grafos (DFS e BFS)

Exercício 3.1 – Representação de grafo (lista de adjacência)

- Crie uma função que receba uma lista de arestas (ex.: (A, B)) e construa um grafo orientado ou não-orientado (à sua escolha) em forma de lista de adjacência.

Exercício 3.2 – DFS em grafo

- Implemente uma função que receba o grafo e um nó inicial e retorne a ordem de visita dos nós usando Depth-First Search.

Exercício 3.3 – BFS em grafo

- Implemente uma função que receba o grafo e um nó inicial e retorne a ordem de visita dos nós usando Breadth-First Search.

Exercício 3.4 – Busca de caminho entre dois nós

- **Usando DFS ou BFS, encontre se existe um caminho entre dois nós fornecidos.**
- **Se existir, retorne o caminho percorrido.**

Exemplo

Considere o grafo não-orientado com arestas:

A-B, A-C, B-D, C-D, D-E

- A DFS partindo de A pode gerar a ordem de visita A -> B -> D -> E -> C.
- A BFS partindo de A pode gerar a ordem A -> B -> C -> D -> E.
- Para verificar se existe um caminho de A até E, basta usar BFS (ou DFS) e ver que sim: A -> B -> D -> E (um dos caminhos possíveis).

Grupo 4 – Cliente e Servidor (Sockets TCP e UDP)

Exercício 4.1 – Servidor TCP simples

- Crie um servidor TCP que escute em uma porta específica.
- Quando receber uma conexão, retorne uma mensagem de boas-vindas ao cliente.

Exercício 4.2 – Cliente TCP simples

- Implemente um cliente que se conecte ao servidor do exercício anterior.
- O cliente envia uma mensagem e exibe a resposta recebida do servidor.

Exercício 4.3 – Servidor UDP simples

- Crie um servidor UDP que escute em uma porta específica.
- Quando receber um datagrama, responda com um “ack”.

Exercício 4.4 – Cliente UDP simples

- Implemente um cliente UDP que envie uma mensagem ao servidor do exercício anterior e exiba a resposta “ack”.

Observações Importantes

- Para testar localmente, você pode usar localhost (ou endereço IP 127.0.0.1) e portas diferentes (por exemplo, 5000, 5001 etc.).
- Execute primeiro o servidor (TCP ou UDP) e depois o cliente correspondente.

Grupo 5: Comunicação de Rede (Telnet e cURL)

Exercício 5.1 - Testar servidor TCP via Telnet

- Inicie o servidor TCP do Grupo 4 (exercício 1).
- Via terminal, use o comando telnet 127.0.0.1 5000 (ou a porta utilizada).
- Observe a mensagem do servidor.
- Explique o que acontece se você enviar dados manualmente no terminal telnet.

Exercício 5.2 - Testar servidor TCP via cURL

- Ainda com o servidor TCP em execução, tente usar curl 127.0.0.1:5000.
- Descreva o que acontece. (Observação: O cURL normalmente espera comunicação HTTP, então o comportamento pode ser diferente.)

Exercício 5.3 - Simular requisição HTTP com cURL

- Inicie um servidor HTTP simples (por exemplo, usando python -m http.server 8000).
- Use o comando curl 127.0.0.1:8000 para obter a página padrão.
- Descreva o que você recebe como resposta.

Exercício 5.4 - Enviar dados via cURL (POST)

- Use o comando curl -X POST -d "nome=Joao" http://127.0.0.1:8000 (ou serviço HTTP configurado).
- Explique a diferença em relação a um GET simples.

Exemplos e Explicações

- Telnet é um protocolo que permite estabelecer comunicação textual com um servidor em uma porta específica. Ao conectar em um servidor TCP simples, você consegue “conversar” enviando strings diretamente.

- cURL é uma ferramenta de linha de comando para transferências de dados. Se o servidor não for HTTP, o cURL pode não compreender o que exibir, mas ainda assim é possível observar a troca de dados bruta.
- Quando usamos curl para fazer um POST, estamos enviando dados no corpo da requisição, diferente de um GET, que envia parâmetros na URL ou não envia dados adicionais.

Observações de implementação

- Para esses exercícios, não há um “código-fonte” Python específico para telnet e cURL, pois tratam-se de testes de linha de comando e de um servidor que pode ser baseado no Grupo 4. A ideia é entender e praticar a comunicação com servidores através dessas ferramentas.