Instituto Infnet

ESTI - ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO

Curso de Graduação em Engenharia de Software

Nome: Kaio Henrique Silva da Cunha

Estruturas de Dados e Algoritmos Avançados

TP1

Fortaleza - Ceará

KAIO HENRIQUE SILVA DA CUNHA

TP1

TP1 da disciplina de Estruturas de Dados e Algoritmos Avançados do curso de Graduação em Engenharia de Software.

Source code: [kaiohenricunha/kaio_cunha_DR3_TP1]

Parte 1 - Implementação e Aplicações da Estrutura de Dados Heap Binária

Exercício 1: Motivação para usar Heap

Explique a principal motivação para o uso da estrutura de dados Heap. Em quais tipos de problemas a utilização dessa estrutura é mais vantajosa em relação a outras, como filas e listas ordenadas?

R: O heap permite inserir e remover elementos rapidamente, mantendo sempre o maior (ou menor) valor na raiz. Isso o torna ideal para problemas que envolvem prioridades, como escalonamento de tarefas e algoritmos de busca, onde ele supera filas e listas ordenadas em eficiência.

Exercício 2: Implementação da estrutura de dados MinHeap

Enunciado: Implemente uma MinHeap em Python sem utilizar bibliotecas auxiliares, garantindo que os elementos sempre sejam organizados corretamente após inserções e remoções.

R: kaio_cunha_DR3_TP1/min_heap.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Exercício 3: Implementação da estrutura de dados MaxHeap

Enunciado: Considere a seguinte MaxHeap armazenada em um array:

[50, 30, 40, 10, 20, 35]

Após a inserção do elemento 45, qual será a nova estrutura do Heap? Justifique sua resposta.

R: A nova estrutura será: [50, 30, 45, 10, 20, 35, 40].

Justificativa: O 45 é inserido no final, na posição 6. Comparado ao seu pai (40), como 45 > 40, ocorre a troca, resultando em [50, 30, 45, 10, 20, 35, 40]. Como agora 45 não é maior que o pai (50), o heap já está correto.

Exercício 4: Remoção em Heap

Enunciado: Explique o processo de remoção do elemento raiz em uma MinHeap e implemente uma função pop() que execute essa operação corretamente.

R: Ao remover a raiz de uma MinHeap, o último elemento do array substitui a raiz e é removido da última posição. Em seguida, usamos o procedimento de heapify-down para

restaurar a propriedade do heap, comparando o novo elemento com seus filhos e trocando-o com o menor deles, se necessário.

Segue a implementação da função pop():

kaio_cunha_DR3_TP1/min_heap_pop.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Parte 2 - Manipulação e Uso de Arrays como Heaps em Estruturas de Dados Avançadas

Exercício 5: Heaps e Arrays

Enunciado: Em uma implementação de Heap usando um array, como podemos calcular:

- **O índice do pai de um nó no índice i?**
- **O índice do filho esquerdo de um nó no índice i?**
- **O índice do filho direito de um nó no índice i?**

R: Em um heap armazenado em array (com índice iniciando em 0):

Pai: (i - 1) // 2
Filho esquerdo: 2 * i + 1
Filho direito: 2 * i + 2
Essas fórmulas garantem acesso rápido aos nós e mantêm a estrutura do heap.

Exercício 6: Aplicação de Heap em um cenário real

Enunciado: Explique como uma Heap pode ser utilizada para implementar uma Fila de Prioridade. Em seguida, implemente um sistema de agendamento de tarefas onde cada tarefa possui uma prioridade, utilizando uma MinHeap.

R: Uma Heap organiza os elementos de forma que o de menor valor – no caso, a tarefa com maior prioridade – fique sempre na raiz, facilitando a remoção do item mais urgente. Assim, uma fila de prioridade baseada em MinHeap garante que a cada extração você esteja obtendo a tarefa de maior prioridade.

Segue um exemplo de um sistema de agendamento de tarefas usando uma MinHeap:

kaio_cunha_DR3_TP1/task_scheduler.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Exercício 7: Implementação de Trie

Enunciado: Implemente uma Trie em Python para armazenar um conjunto de palavras. A implementação deve conter métodos para inserir e buscar palavras.

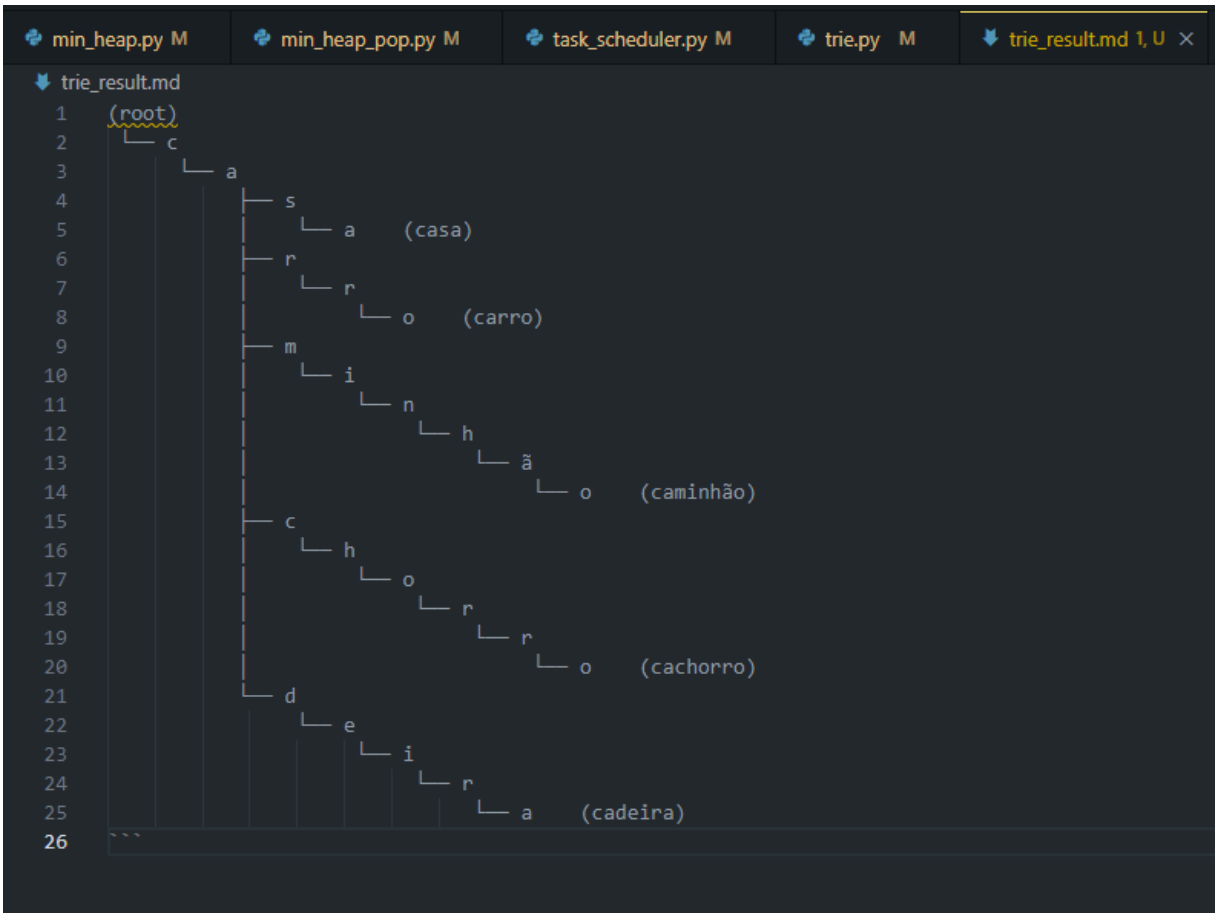
R: Segue uma implementação simples de uma Trie em Python, com os métodos para inserir e buscar palavras:

kaio_cunha_DR3_TP1/trie.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Exercício 8: Operações em Trie

Enunciado: Após inserir as palavras "casa", "carro", "caminhão", "cachorro" e "cadeira" em uma Trie, represente sua estrutura hierárquica.

R: Após inserir as palavras, a estrutura hierárquica da Trie fica assim:



kaio_cunha_DR3_TP1/trie_c_words.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Parte 3 - Implementação de Estruturas de Dados Trie

Exercício 9: Inserção e Busca em Trie

Enunciado: Dado o seguinte código para inserir palavras em uma Trie, complete a função search(word) para verificar se uma palavra está presente na estrutura.

```python
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False
class Trie:
    def __init__(self):
        self.root = TrieNode()
    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
    def search(self, word):
        # Implementar aqui a função de busca
```

R: Segue o código completo:

kaio_cunha_DR3_TP1/trie_search.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Exercício 10: Aplicação de Trie

Enunciado: Explique como a estrutura Trie pode ser usada para desenvolver um sistema de autocomplete. Implemente uma função que, dado um prefixo, retorne todas as palavras armazenadas na Trie que começam com esse prefixo.

R: A Trie é ideal para sistemas de autocomplete porque organiza as palavras de forma hierárquica, permitindo que, ao percorrer os nós correspondentes ao prefixo digitado, possamos explorar rapidamente a subárvore em busca de todas as palavras que iniciam com aquele prefixo.

Segue o código com a função autocomplete implementada:

kaio_cunha_DR3_TP1/trie_autocomplete.py at main · kaiohenricunha/kaio_cunha_DR3_TP1

Exercício 11: Comparação entre as estruturas Heap e Trie

Enunciado: Compare as estruturas Heap e Trie. Em quais situações uma Trie seria mais eficiente do que uma Heap e vice-versa?

R: A Heap é ótima para gerenciar prioridades – ela garante acesso rápido ao maior ou menor elemento, o que é ideal em filas de prioridade, escalonamento de tarefas e algoritmos de ordenação parcial. Já a Trie é especializada em operações com strings, permitindo buscas e inserções rápidas baseadas no comprimento da palavra, não no total de itens. Assim, em sistemas de autocomplete ou dicionários, a Trie é mais eficiente, enquanto a Heap brilha em problemas que exigem gerenciamento dinâmico de prioridades.

Exercício 12: Casos de Uso da estrutura de Dados Trie

Enunciado: Cite pelo menos três aplicações reais onde a estrutura Trie pode ser utilizada. Escolha uma delas e implemente um exemplo funcional em Python.

R: Três aplicações reais de uma Trie são:

Sistema de Autocomplete: Sugere palavras à medida que o usuário digita (usado em buscadores e editores de texto).
Verificação Ortográfica: Permite armazenar dicionários para conferir se uma palavra existe e sugerir correções.
Roteamento de Pacotes de Rede: Facilita a busca por prefixos de endereços IP para encaminhamento eficiente.