INSTITUTO FEDERAL DA PARAÍBA

SERGIO HENRIQUE DA SILVA

VALMIR JUVINO DA SILVA

VICTOR HENRIQUE SANTOS FERREIRA

WILIAM TERROSO DE SOUSA MELO

IMPLEMENTAÇÃO DE UM SIMULADOR DE BROWSER COM ÁRVORE AVL

João Pessoa

2024

Introdução

O presente relatório descreve as classes AVLNode e AVLTree, que juntas implementam uma árvore AVL. Uma árvore AVL é uma árvore binária de busca auto-balanceada que mantém sua altura minimizada, garantindo a eficiência nas operações de busca, inserção e remoção. Esse balanceamento automático é obtido através de rotações que garantem que, para qualquer nó, a diferença de altura entre suas subárvores esquerda e direita não seja maior que 1.

A seguir, detalhamos a função de cada classe e seus métodos, explicando o papel de cada componente na manutenção e manipulação da árvore AVL.

Classe AVLNode

A classe AVLNode define a estrutura básica de um nó na árvore AVL. Cada nó contém um valor (chave), ponteiros para os filhos à esquerda e à direita, e a altura do nó, usada para manter o balanceamento da árvore.

Atributos

key: A chave ou valor armazenado no nó. Esse valor é utilizado para determinar a posição do nó dentro da árvore binária de busca.

left: Ponteiro para o filho à esquerda, que contém valores menores que a chave do nó atual.

right: Ponteiro para o filho à direita, que contém valores maiores que a chave do nó atual.

height: A altura do nó na árvore, que é utilizada para calcular o balanceamento e realizar rotações conforme necessário.

Método __init__

```
def __init__(self, key):
    self.key = key
    self.left = None
    self.right = None
    self.height = 1
```

O método __init__ inicializa um nó AVL com uma chave fornecida, sem filhos (filhos definidos como None), e com a altura inicial definida como 1 (já que um nó sozinho tem altura 1).

Classe AVLTree

A classe AVLTree é responsável pela lógica da árvore AVL como um todo. Ela mantém a raiz da árvore e oferece métodos para inserir novos nós, buscar valores, e garantir o balanceamento da árvore por meio de rotações.

Atributo root

root: Este atributo armazena a raiz da árvore AVL. Inicialmente, a árvore é vazia, e root é definido como None.

Método get_height

```
def get_height(self, node):
    return node.height if node else 0
```

Esse método retorna a altura de um nó, ou 0 se o nó for None. É utilizado para calcular o fator de balanceamento e atualizar as alturas após a inserção ou remoção de nós.

Método get_balance

```
def get_balance(self, node):
    return self.get_height(node.left) - self.get_height(node.right)
```

Calcula o fator de balanceamento de um nó, subtraindo a altura da subárvore direita da subárvore esquerda. Esse valor é usado para determinar se o nó está balanceado e, se não estiver, qual tipo de rotação é necessária.

Método right_rotate

```
def right_rotate(self, y):
    x = y.left
    T2 = x.right
    x.right = y
    y.left = T2
    y.height = 1 + max(self.get_height(y.left), self.get_height(y.right))
    x.height = 1 + max(self.get_height(x.left), self.get_height(x.right))
    return x
```

Realiza uma rotação à direita para reequilibrar a árvore quando ela está desbalanceada para a esquerda. Esse processo move o nó esquerdo (x) para cima e o nó desbalanceado (y) para a direita de x, ajustando os ponteiros adequadamente.

Método left_rotate

```
def left_rotate(self, x):
    y = x.right
    T2 = y.left
    y.left = x
    x.right = T2
    x.height = 1 + max(self.get_height(x.left), self.get_height(x.right))
    y.height = 1 + max(self.get_height(y.left), self.get_height(y.right))
    return y
```

Realiza uma rotação à esquerda quando a árvore está desbalanceada para a direita, movendo o nó direito (y) para cima e o nó desbalanceado (x) para a esquerda de y.

Método insert

```
def insert(self, node, key):
    if not node:
        return AVLNode(key)
    elif key < node.key:
        node.left = self.insert(node.left, key)
    else:
        node.right = self.insert(node.right, key)

    node.height = 1 + max(self.get_height(node.left), self.get_height(node.right))
    balance = self.get_balance(node)
```

Balanceamento

```
    if balance > 1:
        if key < node.left.key:
            return self.right_rotate(node)
        else:
            node.left = self.left_rotate(node.left)
            return self.right_rotate(node)

    if balance < -1:
        if key > node.right.key:
            return self.left_rotate(node)
        else:
            node.right = self.right_rotate(node.right)
            return self.left_rotate(node)

    return node
```

O método de inserção segue as regras de uma árvore binária de busca, adicionando o novo nó à esquerda ou à direita do nó atual, dependendo do valor da chave. Após a inserção, a altura do nó é atualizada e o fator de balanceamento é calculado. Se o nó estiver desbalanceado, rotações são realizadas para restaurar o balanceamento da árvore:

Desbalanceado à esquerda (fator de balanceamento > 1): uma rotação à direita é realizada, ou uma rotação dupla (esquerda-direita) se necessário.

Desbalanceado à direita (fator de balanceamento < -1): uma rotação à esquerda é realizada, ou uma rotação dupla (direita-esquerda) se necessário.

Método search

```
def search(self, node, key):
    if not node or node.key == key:
        return node
    elif key < node.key:
        return self.search(node.left, key)
    else:
        return self.search(node.right, key)
```

Realiza a busca de uma chave na árvore AVL, retornando o nó que contém a chave, ou None se a chave não for encontrada.

Método in_order_traversal

```
def in_order_traversal(self, node):
    res = []
    if node:
        res = self.in_order_traversal(node.left)
        res.append(node.key)
        res = res + self.in_order_traversal(node.right)
    return res
```

Realiza um percurso em ordem (in-order traversal) da árvore, retornando uma lista de chaves em ordem crescente.

Classe Browser:

O código é estruturado dentro da classe Browser possui diversas funcionalidades, como manter um histórico de navegação, validar URLs, e organizar URLs em uma árvore AVL (uma árvore binária balanceada). Abaixo está um resumo das principais partes do código:

1. Atributos da Classe Browser:

self.sitemap: Armazena as URLs em uma árvore AVL, permitindo a organização e busca eficiente.

self.visit_history: Lista que guarda o histórico de páginas visitadas.

self.current_page: Armazena a página atual em que o usuário está navegando.

2. Métodos Principais:

__init__: Inicializa os atributos da classe.

load_urls: Carrega URLs de um arquivo de texto (urls.txt) e as adiciona ao sitemap após verificar se são válidas.

is_valid_url: Verifica se uma URL segue o formato correto utilizando uma expressão regular (regex).

add_url: Adiciona uma URL ao sitemap após validar sua formatação. Se a URL já existir no sitemap, o código evita duplicatas.

show_history: Exibe o histórico de páginas visitadas, se houver.

navigate: Navega para uma página a partir de uma URL, adicionando a página atual ao histórico antes de mudar para a nova. Se o caminho da URL for relativo (iniciando com "/"), ele o completa com a URL atual.

back: Retorna à última página visitada no histórico, ou exibe uma mensagem se o histórico estiver vazio.

show_sitemap: Exibe todas as URLs armazenadas no sitemap por meio de uma travessia in-order da árvore AVL.

help_command: Exibe uma lista de comandos disponíveis para o usuário.

run: O loop principal do navegador, que fica rodando continuamente até que o usuário insira o comando para sair. O loop aceita comandos de navegação e manipulação de URLs.

Interatividade:

O navegador permite as seguintes operações:

Navegar por URLs.

Adicionar novas URLs ao sitemap.

Ver o histórico de navegação.

Retornar à página anterior.

Visualizar o sitemap completo.

Exibir comandos de ajuda.

Comandos Suportados:

#add <url>: Adiciona uma URL ao sitemap.

#back: Retorna à última página visitada.

#showhist: Exibe o histórico de navegação.

#showsitemap: Exibe o sitemap.

#help: Exibe a ajuda com a lista de comandos.

#sair: Encerra a execução do navegador.

Método is_valid_url

A expressão regular usada no método is_valid_url tem o objetivo de validar URLs, permitindo diferentes formatos como URLs com prefixos comuns (http, https, ftp), domínios padrão, endereços de IP e nomes de host locais como "localhost". Aqui está uma explicação detalhada de cada parte da expressão:

Descrição por partes:

^: Indica o início da string (começo da URL).

(?:http|ftp)s?://:

(?: ...): Grupo sem captura para combinar uma das opções dentro dos parênteses.

http|ftp: Combina os protocolos "http" ou "ftp".

s?: O "s" é opcional, permitindo "http" ou "https" (ou "ftp" ou "ftps").

://: Literalmente "://", que deve aparecer após o protocolo.

|(?: ...):

Define um conjunto alternativo, ou seja, se a URL não começar com um protocolo "http" ou "ftp", pode seguir um domínio, IP, ou host local. Esta parte divide a validação para domínios, endereços IP, ou "localhost".

(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+:

[A-Z0-9]: Um caractere alfanumérico (A-Z ou 0-9) no início do domínio.

(?:[A-Z0-9-]{0,61}[A-Z0-9])?: Um grupo sem captura que permite domínios com hífen (mas não no início ou no final). Limita o comprimento da sequência de caracteres a um máximo de 63 caracteres, como é típico em nomes de domínio.

\.: Um ponto literal que separa as partes do domínio.

+: O domínio pode ter várias partes separadas por pontos (ex.: sub.example.com).

(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?):

[A-Z]{2,6}\.?: Domínios de topo (TLDs) como ".com", ".org", ou códigos de país com 2 a 6 letras (ex.: ".br", ".museum").

[A-Z0-9-]{2,}\.?: Também permite TLDs mais genéricos, que podem incluir números ou hífens (ex.: ".co-uk").

localhost: Permite a string literal "localhost", que é usada para referenciar a máquina local.

\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}:

Valida endereços IPv4. Cada grupo \d{1,3} deve ter entre 1 e 3 dígitos, e são separados por pontos (.). Isso cobre endereços IP como "192.168.0.1".

\[?[A-F0-9]*:[A-F0-9:]+\]?:

Valida endereços IPv6.

\[?\]: O endereço IPv6 pode estar entre colchetes (necessário ao especificar a porta em uma URL IPv6).

[A-F0-9]*:[A-F0-9:]+: Sequência de caracteres hexadecimais e dois-pontos, como "2001:0db8:85a3:0000:0000:8a2e:0370:7334".

(?::\d+)?:

Porta opcional na URL. O número após os dois-pontos (:) representa a porta (ex.: :8080), e \d+ significa um ou mais dígitos. O grupo é opcional.

(?:/?|[/?]\S+)?$:

/?: Permite que a URL termine com uma barra opcional.

[/?]\S+: Permite caminhos ou parâmetros na URL, após a barra. \S+ representa um ou mais caracteres que não sejam espaços.

$: Indica o fim da string (fim da URL).

Resumo:

Esta expressão regular permite validar URLs nos seguintes formatos:

URLs com prefixo "http", "https", "ftp", ou "ftps".

Domínios válidos (alfanuméricos, hífenes e pontos).

Endereços IPv4 e IPv6.

URLs locais como "localhost".

URLs com ou sem portas e caminhos opcionais.

Ela é uma expressão bastante abrangente e flexível, garantindo que URLs nos principais formatos sejam aceitas.

Conclusão

As classes AVLNode e AVLTree implementam de forma completa uma árvore AVL, uma estrutura de dados auto-balanceada que otimiza as operações de busca, inserção e remoção. A árvore AVL garante que sua altura permaneça logarítmica, o que mantém o tempo de complexidade dessas operações em O(log n). Através de rotações à esquerda e à direita, a árvore se mantém balanceada automaticamente após cada inserção.

Essas classes são extremamente úteis em sistemas que requerem operações rápidas de busca e inserção em grandes volumes de dados, como bancos de dados, compiladores e sistemas de arquivos.

Em resumo, este código cria um navegador básico de terminal que armazena URLs em uma estrutura eficiente de árvore AVL, permite navegação e manipulação de URLs, e oferece uma interface simples de linha de comando para o usuário.