BOLIVAR AUGUSTO DIAS JUNIOR

WIKI EDI

Barbacena – MG

2023

Introdução

Neste texto, abordaremos a criação de um programa chamado "wikiedi" que tem como objetivo gerenciar uma enciclopédia colaborativa, semelhante a uma plataforma Wiki. O programa permite a criação e edição de páginas, bem como a criação e gerenciamento de links entre essas páginas. Além disso, ele também rastreia informações sobre os editores envolvidos no processo de edição.

O programa é desenvolvido em linguagem C e faz uso de várias estruturas de dados e algoritmos para atingir seus objetivos. Ao longo deste texto, discutiremos as principais estruturas de dados e algoritmos utilizados, bem como suas complexidades e características.

Visão Geral do Funcionamento do Programa

O programa "wikiedi" é uma aplicação que simula uma enciclopédia colaborativa, onde os usuários podem criar, editar e interconectar páginas de informações. Vamos fornecer uma visão geral das principais estruturas de dados e algoritmos utilizados no programa:

Estruturas de Dados para Páginas e Conteúdo:

TInfoPage: Esta estrutura armazena informações sobre uma página, incluindo seu nome, nome do arquivo associado e os links para outras páginas.

TNodoPage: Representa um nó em uma lista encadeada de páginas, permitindo a navegação entre elas.

TPagina: Esta é a estrutura principal que mantém o controle sobre as páginas, rastreia a página atual e fornece métodos para adicionar, remover e acessar páginas.

Estruturas de Dados para Editores:

TConteudoEdit: Armazena informações sobre a edição de uma página, incluindo o nome da página, o nome do editor e o conteúdo colaborativo.

TNodoEditor: Representa um nó em uma lista encadeada de editores.

TEditores: Mantém uma lista de editores, permitindo adicionar, remover e pesquisar informações sobre os editores.

Estruturas de Dados para Links:

TLinks: Mantém informações sobre os links entre as páginas, incluindo a página de origem e a página de destino.

TListaLinks: Permite criar e gerenciar links entre as páginas.

Algoritmos Auxiliares:

O programa utiliza diversos algoritmos auxiliares, como funções para pesquisa, inserção e remoção de elementos nas estruturas de dados. Além disso, são implementadas funções para verificar a existência de páginas ou editores, bem como o controle de espaços disponíveis.

Execução do Programa:

A execução do programa ocorre através da interpretação de comandos fornecidos pelos usuários. O programa lê comandos de um arquivo de teste (por exemplo, "TestOENGHUS.txt") e executa as operações correspondentes.

Nas próximas páginas, discutiremos em detalhes cada uma das estruturas de dados e algoritmos mencionados acima, suas complexidades e como eles funcionam em conjunto para permitir o funcionamento eficiente do programa "wikiedi". Além disso, abordaremos cenários de uso típicos e exemplos de comandos que podem ser executados com este programa.

Implementação

Função bornPage()

Propósito: Esta função é responsável por criar uma nova página.

Entrada: Nenhum argumento é passado para a função.

Saída: Retorna um ponteiro para uma nova página, do tipo TPagina.

A função bornPage aloca memória para uma nova página (TPagina) e inicializa os campos dessa estrutura. Ela define um cursor, um início e um fim da página como nulos e define o tamanho e a posição corrente como 0. Essa função é essencial para a criação de uma nova página vazia.

Função firstPage()

Propósito: Adicionar a primeira página à lista de páginas.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages e uma estrutura TInfoPage chamada infoEnter.

Saída: Retorna 1 em caso de sucesso, 0 em caso de erro.

Essa função adiciona a primeira página à lista de páginas (wikiPages). Ela verifica se a página está cheia usando a função fullPage. Se a página não estiver cheia, aloca memória para uma nova página (TNodoPage) e inicializa seus campos. Essa nova página é então atribuída como início, fim e cursor da lista, e o tamanho da página é atualizado. Se ocorrer algum erro, a função retorna 0.

Função lastPage()

Propósito: Adicionar uma página no final da lista de páginas.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages e uma estrutura TInfoPage chamada infoEnter.

Saída: Retorna 1 em caso de sucesso, 0 em caso de erro.

Essa função adiciona uma página ao final da lista de páginas (wikiPages). Assim como em firstPage, ela verifica se a página está cheia usando a função fullPage. Se a página não estiver cheia, aloca memória para uma nova página (TNodoPage) e inicializa seus campos. Se a lista estiver vazia, ela chama

firstPage para adicionar a primeira página. Caso contrário, a nova página é vinculada ao último elemento da lista, atualizando o fim, e o tamanho da página é incrementado. Se ocorrer algum erro, a função retorna 0.

Função removePage()

Propósito: Remover uma página da lista de páginas com base no nome da página.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages, uma string nomePage e um ponteiro para uma estrutura TInfoPage chamada infoEnter.

Saída: Retorna 1 em caso de sucesso (página removida) ou 0 em caso de falha.

Essa função permite a remoção de uma página da lista de páginas (wikiPages) com base no nome da página fornecido. Ela percorre a lista, comparando o nome da página com a string nomePage. Se encontrar uma correspondência, a página é removida da lista, ajustando os ponteiros inicio e fim conforme necessário. A função retorna 1 se a página for removida com sucesso, caso contrário, retorna 0.

Função finderPage()

Propósito: Encontrar uma página com base no nome da página e fornecer um ponteiro para a estrutura TInfoPage.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages e uma string nomePage. Também recebe um ponteiro para um ponteiro de estrutura TInfoPage chamado infoEnter.

Saída: Retorna 1 em caso de sucesso (página encontrada), 0 em caso de falha.

Essa função permite encontrar uma página com base no nome da página fornecido. Ela percorre a lista de páginas, comparando o nome da página com a string nomePage. Se encontrar uma correspondência, ela atualiza o ponteiro infoEnter para apontar para a estrutura de informações (TInfoPage) da página encontrada e atualiza o cursor da página para essa página. A função retorna 1 se a página for encontrada com sucesso, caso contrário, retorna 0.

Função emptyPage()

Propósito: Verificar se a lista de páginas está vazia.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages.

Saída: Retorna 1 se a lista estiver vazia, 0 caso contrário.

Essa função simplesmente verifica se a lista de páginas está vazia, retornando 1 se estiver ou 0 caso contrário.

Função fullPage()

Propósito: Verificar se a lista de páginas está cheia.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages.

Saída: Retorna -1 se não puder alocar memória (indicando que a lista está cheia), 0 caso contrário.

Essa função é usada para verificar se a lista de páginas está cheia. Ela tenta alocar memória para uma nova página, e se essa alocação falhar, a função retorna -1, indicando que a lista está cheia. Caso contrário, retorna 0.

Função quantityPages()

Propósito: Obter a quantidade de páginas na lista.

Entrada: Recebe um ponteiro para uma estrutura TPagina chamada wikiPages.

Saída: Retorna o número de páginas na lista.

Essa função simplesmente retorna o tamanho da lista de páginas, que é mantido atualizado à medida que as páginas são adicionadas ou removidas.

Essas funções juntas compõem o sistema de gerenciamento de páginas, permitindo criar, adicionar, remover, encontrar, verificar a lista vazia, verificar a lista cheia e obter a quantidade de páginas em uma estrutura de dados. Tenha em mente que a explicação acima é baseada no código fornecido e pode haver detalhes de implementação adicionais que não são evidentes a partir do código.

Função retiraEnter(char string)*

Propósito: Esta função remove o caractere de nova linha (enter) de uma string.

Entrada: Recebe um ponteiro para uma string string.

Saída: Não possui valor de retorno.

A função retiraEnter verifica se o último caractere na string é uma nova linha (\n) e, se for, substitui esse caractere por um caractere nulo (\0). Isso é útil para processar strings lidas de entradas, garantindo que elas não contenham caracteres de nova linha.

Função separarComandoEArquivo(char entrada, char* comando, char* nomeArquivo)*

Propósito: Essa função separa um comando e um nome de arquivo de uma entrada.

Entrada: Recebe uma string entrada, um ponteiro para uma string comando e um ponteiro para uma string nomeArquivo.

Saída: Não possui valor de retorno.

A função separarComandoEArquivo analisa a string entrada em busca do primeiro espaço em branco. O texto anterior a esse espaço é copiado para a string comando, e o texto após o espaço (se houver) é copiado para a string nomeArquivo. Isso é usado para separar um comando de um nome de arquivo em uma entrada.

Função separarComandoE4Palavras(const char entrada, char** comandosLinha)*

Propósito: Essa função divide uma entrada em até quatro palavras separadas.

Entrada: Recebe uma string entrada e um vetor de ponteiros para strings comandosLinha.

Saída: Não possui valor de retorno, mas preenche o vetor comandosLinha.

A função separarComandoE4Palavras divide a string entrada em palavras com base em espaços em branco. Ela percorre a entrada, identificando as palavras

e armazenando-as nos elementos do vetor comandosLinha. O vetor comandosLinha é um vetor de ponteiros de strings, onde cada elemento contém uma das palavras encontradas na entrada. Ela divide a entrada em até quatro palavras.

Função *openFileTester*(char* nomeArquivo)

Propósito: Esta função tenta abrir um arquivo com o nome especificado.

Entrada: Recebe o nome do arquivo como uma string nomeArquivo.

Saída: Retorna 0 se o arquivo puder ser aberto com sucesso, -1 caso contrário.

A função openFileTester tenta abrir um arquivo especificado pelo nome fornecido. Ela usa a função fopen_s para tentar abrir o arquivo no modo de leitura ("r"). Se o arquivo for aberto com sucesso, a função retorna 0. Caso contrário, retorna -1 para indicar uma falha na abertura do arquivo.

Função pesquisaFuncion(char comando)*

Propósito: Esta função pesquisa se um comando existe em um vetor de comandos pré-definidos.

Entrada: Recebe o nome do comando como uma string comando.

Saída: Retorna um valor inteiro correspondente à posição do comando no vetor pré-definido, ou -1 se o comando não for encontrado.

A função pesquisaFuncion verifica se o comando fornecido está presente em um vetor chamado vetorCommand, que contém comandos pré-definidos. Ela compara o comando fornecido com os comandos no vetor e, se encontrar uma correspondência, retorna a posição desse comando no vetor mais 1 (a posição é incrementada em 1 para que um valor de retorno 0 represente "comando não encontrado"). Caso contrário, retorna -1 para indicar que o comando não foi encontrado no vetor.

Essas funções desempenham papéis específicos no processamento de entradas, separação de comandos e manipulação de arquivos no programa. Elas são úteis para garantir que as entradas sejam tratadas corretamente e que os

comandos sejam reconhecidos, facilitando a execução das operações desejadas no programa.

Estruturas de Dados Principais:

A função executer é parte de um programa que gerencia uma espécie de sistema de edição colaborativa semelhante a uma wiki. ***Essa função é o centro de todo o programa.***

TPagina: Representa uma página na wiki. Cada página possui informações como nome, conteúdo e links para outras páginas.

TInfoPage: Armazena informações sobre uma página, como o nome da página e o nome do arquivo correspondente.

TEditors: Mantém uma lista de editores, armazenando informações sobre cada editor, como nome.

TConteudoEdit: Armazena informações sobre contribuições dos editores, incluindo o nome da página, nome do editor e conteúdo da contribuição.

TListaLinks: Mantém uma lista de links entre páginas.

TLinksLista: Armazena informações sobre um link, incluindo a página de origem e a página de destino.

Abertura de Arquivos:

A função executer começa abrindo dois arquivos: um arquivo de teste (arqOpen) e um arquivo de log (arqLog). A abertura de arquivos é crucial para a leitura dos comandos e o registro de eventos no programa.

Loop de Leitura de Comandos:

A função entra em um loop while que lê cada linha do arquivo de teste (arqOpen). Cada linha contém um comando que deve ser executado no sistema de edição colaborativa.

Análise de Comandos

A função analisa os comandos da seguinte maneira:

Inicializa um vetor de palavras para armazenar as partes do comando.

Copia a linha lida do arquivo de teste para a variável entrada.

Divide a linha em palavras usando a função separarComandoE4Palavras, armazenando essas palavras no vetor de comandos comandosLinha.

Determina o tipo de comando usando a função pesquisaFuncion, que compara a primeira palavra do comando com uma lista de comandos conhecidos.

Tratamento de Comandos:

Cada comando é tratado em um bloco switch, onde diferentes ações são executadas com base no comando identificado. Aqui estão algumas das ações comuns para comandos específicos:

Case 1: INSEREPAGINA

Neste caso, o programa lida com a inserção de uma nova página na wiki. Ele começa copiando o nome da página e o nome do arquivo a partir da entrada. Em seguida, verifica se a página já existe na wiki e, se existir, registra um erro. Caso contrário, a página é criada e o evento é registrado no arquivo de log.

Case 2: RETIRAPAGINA

Neste caso, o programa trata da remoção de uma página da wiki. Ele extrai o nome da página da entrada e verifica se a página existe. Se a página for encontrada, ela é removida, caso contrário, um erro é registrado.

Case 3: INSEREEDITOR

Este case lida com a inserção de um novo editor. Ele verifica se o editor já está cadastrado e, se não estiver, adiciona o novo editor à lista de editores.

Case 4: RETIRAEDITOR

Aqui, o programa trata da remoção de um editor. Ele verifica se o editor existe e, se sim, remove o editor e todas as suas colaborações. Caso contrário, registra um erro.

Case 5: INSERECONTRIBUICAO

Neste caso, o programa cuida da inserção de uma contribuição. Ele verifica se a página de destino e o editor existem, e se sim, registra a contribuição.

Case 6: RETIRACONTRIBUICAO

Aqui, o programa trata da remoção de uma contribuição. Ele verifica a existência da página, do editor e da colaboração e, se todas as condições forem atendidas, remove a colaboração.

Case 7: INSERELINK

Neste case, o programa lida com a inserção de um link entre duas páginas. Ele verifica a existência das páginas de origem e destino, insere o link e registra a ação no arquivo de log.

Case 8: RETIRALINK

Aqui, o programa trata da remoção de um link entre duas páginas. Ele verifica a existência das páginas de origem e destino, remove o link e registra a ação no arquivo de log.

Case 9: CAMINHO

Neste caso, o programa verifica se há um caminho entre duas páginas usando as listas de links. Se um caminho existir, ele registra no arquivo de log que há um caminho entre as páginas, caso contrário, registra que não há caminho.

Case 10: IMPRIMEPAGINA

Aqui, o programa gera um arquivo e imprime as informações da página especificada. Ele verifica se a página existe e, se existir, gera um arquivo com informações sobre a página.

Case 11: IMPRIMEWIKED

Este case gera arquivos e imprime informações de todas as páginas da wiki, conforme especificado.

Case 12: FIM

Neste case, o programa determina a finalização do programa. Ele fecha os arquivos que ainda estão abertos e desaloca a memória alocada durante a execução.

Finalização e Desalocação:

Após processar todos os comandos, a função fecha os arquivos (arqOpen e arqLog) e realiza a desalocação de memória, se necessário.

Analise de Complexidade

Para realizar uma análise da notação "O" (complexidade de tempo e espaço) do código apresentado acima, é necessário observar cada função e entender como elas contribuem para a complexidade geral do programa. Vou considerar tanto apenas o tempo em minha análise[1].

INSEREPAGINA (Case 1):

Complexidade de Tempo (O): Inserir uma página envolve principalmente a busca na estrutura de dados da wiki (geralmente implementada como uma lista encadeada ou árvore) para verificar se a página já existe e, em seguida, adicionar a nova página. A complexidade da busca é O(n) no pior caso, onde n é o número de páginas na wiki. Adicionar a página pode ser feito em O(1) se a estrutura de dados for bem organizada. Portanto, a complexidade total é O(n).

RETIRAPAGINA (Case 2):

Complexidade de Tempo (O): Remover uma página envolve uma busca (O(n)) para encontrar a página a ser removida e, em seguida, remover a página. A remoção em uma estrutura de dados bem organizada geralmente é O(1). Portanto, a complexidade total é O(n).

INSEREEDITOR (Case 3):

Complexidade de Tempo (O): Inserir um editor envolve verificar se o editor já existe na lista de editores (O(n)), onde n é o número de editores. A inserção em uma estrutura de dados bem organizada é O(1). Portanto, a complexidade total é O(n).

[1] Nota Importante: A análise de complexidade "O" geralmente se concentra no pior caso, mas pode haver cenários onde a complexidade média é mais relevante.

RETIRAEDITOR (Case 4):

Complexidade de Tempo (O): Remover um editor envolve verificar se o editor existe na lista de editores (O(n)), onde n é o número de editores. A remoção dos colaboradores associados é O(m), onde m é o número total de colaborações feitas por esse editor. A remoção em uma estrutura de dados bem organizada é O(1). Portanto, a complexidade total é O(n + m).

INSERECONTRIBUICAO (Case 5):

Complexidade de Tempo (O): Inserir uma contribuição envolve verificar se a página e o editor existem (O(n) no pior caso) e, em seguida, inserir a contribuição (O(1) em uma estrutura de dados bem organizada). Portanto, a complexidade total é O(n).

RETIRACONTRIBUICAO (Case 6):

Complexidade de Tempo (O): Remover uma contribuição envolve verificar se a página, o editor e a contribuição existem (O(n) no pior caso). A remoção em uma estrutura de dados bem organizada é O(1). Portanto, a complexidade total é O(n).

INSERELINK (Case 7):

Complexidade de Tempo (O): Inserir um link envolve verificar se a página de origem e a página de destino existem (O(n) no pior caso) e, em seguida, inserir o link (O(1) em uma estrutura de dados bem organizada). Portanto, a complexidade total é O(n).

RETIRALINK (Case 8):

Complexidade de Tempo (O): Remover um link envolve verificar se a página de origem e a página de destino existem (O(n) no pior caso). A remoção em uma

estrutura de dados bem organizada é O(1). Portanto, a complexidade total é O(n).

CAMINHO (Case 9):

Complexidade de Tempo (O): Encontrar um caminho entre duas páginas envolve verificar se existem links que conectam as páginas de origem e destino. A complexidade depende da implementação da estrutura de dados que armazena os links. No pior caso, a complexidade é O(n^2), onde n é o número de páginas.

IMPRIMEPAGINA (Case 10):

Complexidade de Tempo (O): Imprimir uma página envolve verificar se a página existe (O(n) no pior caso) e, em seguida, gerar o arquivo de impressão. A complexidade da geração do arquivo depende do tamanho da página e do número de colaborações. A complexidade total pode ser alta, mas depende do conteúdo da página.

IMPRIMEWIKED (Case 11):

Complexidade de Tempo (O): Imprimir todas as páginas da wiki envolve percorrer todas as páginas (O(n)) e gerar um arquivo para cada uma. A complexidade da geração de cada arquivo depende do tamanho da página. Portanto, a complexidade total é O(n).

FIM (Case 12):

Complexidade de Tempo (O): Fechar os arquivos e desalocar a memória geralmente é uma operação rápida, considerada O(1).

Complexidade Geral do Programa:

A complexidade de tempo geral do programa é dominada pelas operações que envolvem busca e verificação de existência, resultando em uma complexidade de tempo total de O(n) no pior caso, onde n é o número de páginas, editores ou colaborações, dependendo da operação.

Conclusão

As principais dificuldades encontradas ao realizar este trabalho incluíram:

Complexidade do Código: O código fornecido era extenso e complexo, com muitas funções e estruturas de dados interconectadas, o que tornou a análise detalhada um desafio.

Quantidade de Informações: A descrição de cada caso e função exigia muitos detalhes, o que aumentou o esforço para elaborar uma análise completa.

Dependência da Implementação: A complexidade e a eficiência de muitas operações dependem da implementação específica das estruturas de dados e algoritmos, o que torna difícil fornecer análises precisas sem detalhes específicos.

Para futuros trabalhos, sugiro que:

Forneça um resumo conciso: O trabalho poderia ser dividido em seções mais curtas e resumos mais concisos para facilitar a leitura e a compreensão.

Melhorias de Eficiência: Identifique possíveis melhorias de eficiência ou otimizações no código, se aplicável.

Bibliografia

GeeksforGeeks. Disponível em: https://www.geeksforgeeks.org/. Acesso em:

Wikipedia. Disponível em: https://en.wikipedia.org/. Acesso em: [16/10/2023].

Coursera. Disponível em: https://www.coursera.org/. Acesso em: Acesso em: [16/10/2023].

TopCoder. Disponível em: https://www.topcoder.com Acesso em: [16/10/2023]..

Khan Academy. Disponível em: https://www.khanacademy.org/. Acesso em: [16/10/2023].

Stack Overflow. Disponível em: https://stackoverflow.com/. Acesso em: Acesso em: [16/10/2023].

GitHub. Disponível em: https://github.com/. Acesso em: [16/10/2023].

SEDGEWICK, Robert; WAYNE, Kevin. Algorithms. Disponível em: https://algs4.cs.princeton.edu/home/. Acesso em: [16/10/2023].

MORIN, Pat. Open Data Structures. Disponível em: https://opendatastructures.org/. Acesso em: [16/10/2023].