Universidade Federal do Espirito Santo - UFES

TRABALHO 1
ESTRUTURA DE DADOS 1
WikED

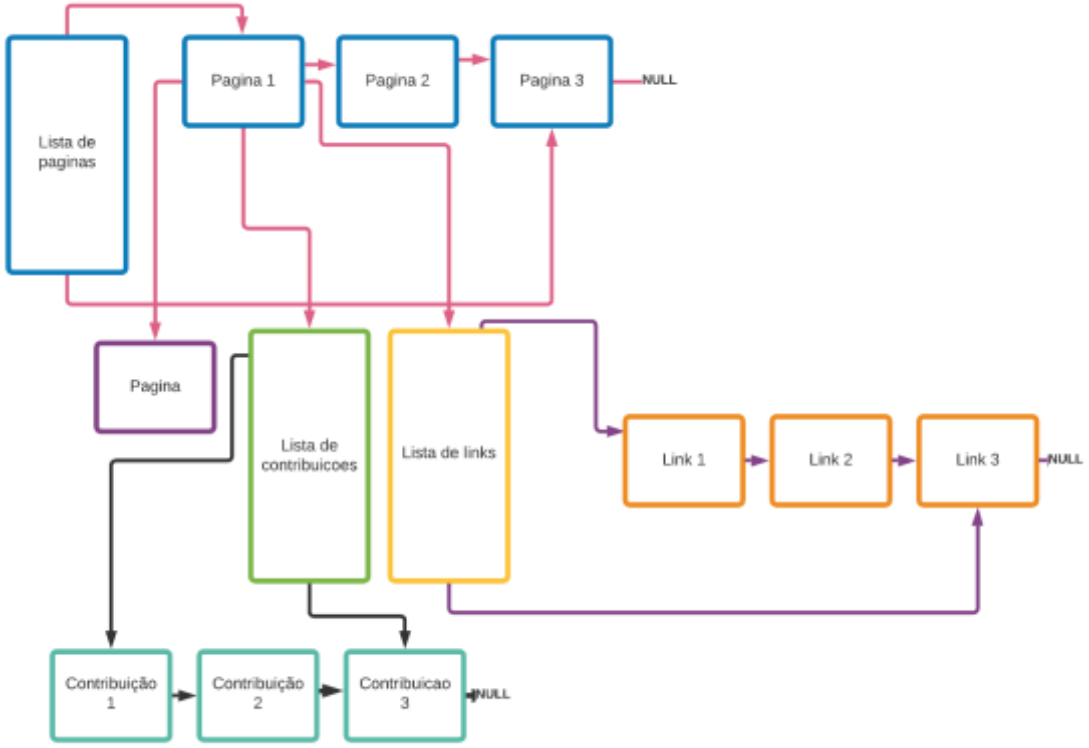
Autoria: Maria Luiza Armini Correa

Data: 23/11/2020

Introdução

O trabalho consistia na implementação de um sistema colaborativo chamado WIKED, nessa plataforma poderíamos inserir páginas, editores, contribuições, fazer links de uma página a outra, de forma que membros de uma comunidade pudessem contribuir com textos e informações e assim desenvolver um artefato útil para a sociedade.

Para iniciar o projeto comecei fazendo uma análise geral de quais seriam, e como seria as estruturas dos TADS principais do sistema.

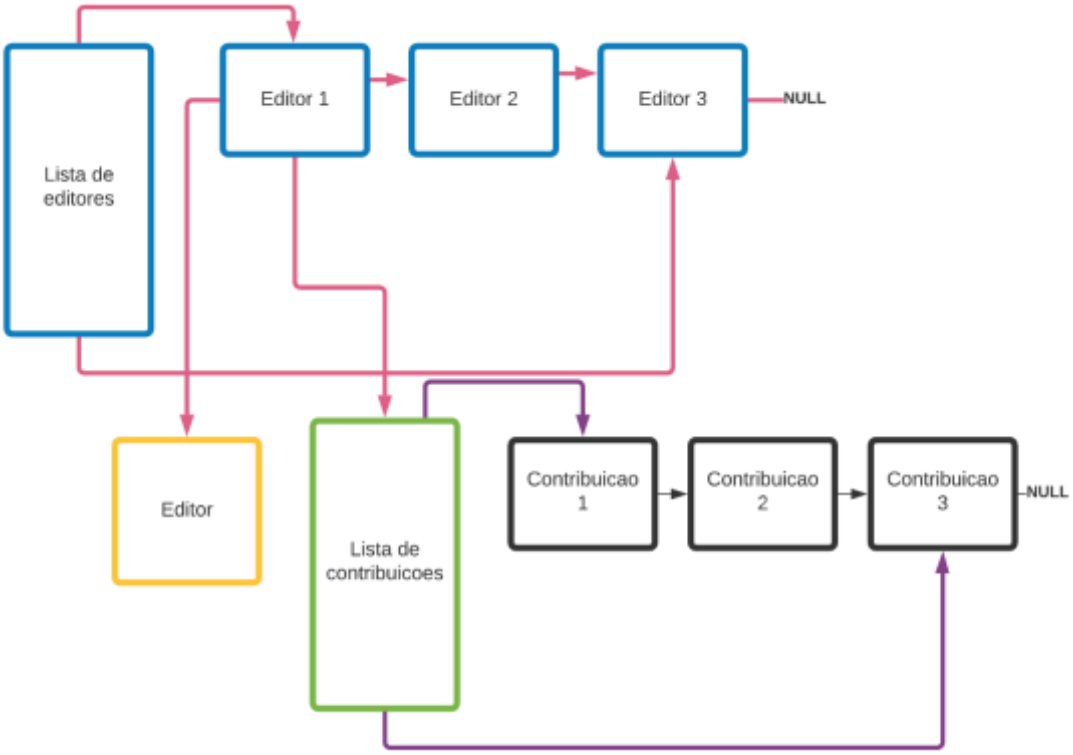


Então, desenvolvi esse fluxograma, onde a ideia principal é: Uma lista de páginas, sendo que, cada célula dessa lista é composta por uma página (conteúdo), um ponteiro apontando para a próxima célula (responsável por fazer o encadeamento da lista), por uma lista de contribuições dessa página e também uma lista de links.

O conceito usado foi o de lista com sentinela, cada bloco maior desse, mostrado no fluxograma acima, representa uma lista com sentinela e sendo assim, possuem um ponteiro apontando para a primeira célula e um ponteiro apontando para última célula.

Além disso, estruturei também uma lista de editores, usando o mesmo conceito de lista com sentinela e cada célula dessa lista é composta por

um editor, uma lista de contribuições feitas por esse editor e um ponteiro apontando para a próxima célula da lista, para mantê-la encadeada.



Depois de ter elaborado a estrutura geral do trabalho, comecei a implementar os TADS, sendo eles:

- **Pagina**
- **Editor**
- **Contribuição**
- **Lista de Pagina**
- **Lista de Editor**
- **Lista de Contribuição**
- **Lista de Link**

Cada um desses, foi implementado o .h com as definições e o .c.

Depois da criação e testes dos TADS, iniciei a leitura e impressão de arquivos.

O arquivo entrada.txt é aberto, lido e fechado na main do programa e o arquivo log, também é aberto e fechado na main, porem ele é enviado como argumento de algumas funções para que ocorra os fprints dos erros.

A estratégia usada para a leitura do entrada.txt, foi ler uma string usando o operador “%s”, pois ele faz a leitura até encontrar o primeiro espaço, e com isso eu consigo separar a função, e os argumentos da mesma.

Metodologia

Como citado acima, ao todo implementei 7 TADS.

No geral todos os TADS possuem as funções para as operações básicas dele, e à medida que foi havendo necessidade foi sendo implementando funcionalidades extras para cada um.

As funções gerais **básicas** são:

- Inicia: Essa função é responsável por inicializar a estrutura, alocando espaço para todos os elementos.

- Retorna: Cada TAD possui informações que em algum momento serão necessárias acessar, para isso as funções ‘Retorna’ existem, elas devolvem ao cliente algo que foi requisitado, como um nome, uma lista e etc.

- Imprime: No código existem dois tipos da função Imprime, uma que é usada para printar no terminal do cliente, e uma outra responsável por fazer os prints nos arquivos .txt. Com o decorrer da implementação do programa, as funções de impressão no terminal, acabaram ganhando uma funcionalidade de debuger, já que os prints do programa passaram a ser feitos apenas nos arquivos, porém mantive a implementação das mesmas para caso o cliente necessitasse de um print no próprio terminal.

- Destroi: Função responsável por liberar toda a memória alocada.

Pagina

Esse TAD possui uma estrutura de mesmo nome, Pagina, composta por um char * nomePagina e um char * nomeArquivo.

Não houve necessidade de implementar nenhuma função extra para o ele, então nele temos:

- IniciaPagina;
- RetornaNomeDaPagina;
- RetornaNomeArquivo;

- ImprimePagina;
  Recebe uma página e imprime ela no terminal.

- DestroiPagina;

Editor

Esse TAD possui uma estrutura de mesmo nome, Editor, composta por um char *nomeEditor.

Não houve necessidade de implementar nenhuma função extra para ele, então nele temos:

- IniciaEditor;

- RetornaNomeEditor;

- ImprimeEditor;
  Recebe uma página e imprime ela no terminal.

- DestroiEditor;

Contribuição

Esse TAD possui uma estrutura de mesmo nome, Contribuição, composta por um char *nomeEditor, char *nomeArquivo, int retirada (Para saber o status da contribuição, se já foi retirada ou não).

Nesse TAD houve a necessidade de implementar mais 2 funções extras para manipular o elemento ‘retirada’.

- IniciaContribuicao;

- RetornaNomeEditorContribuicao;

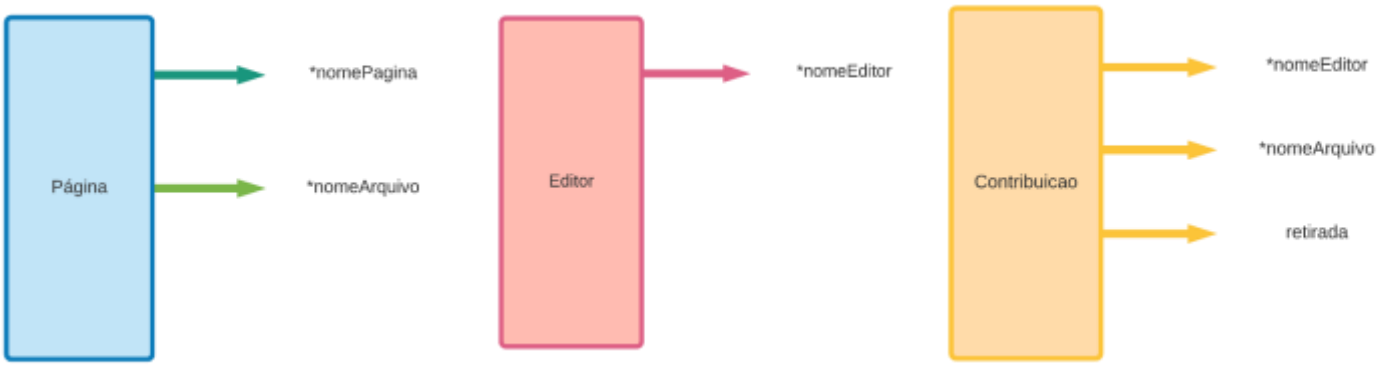
- RetornaNomeArquivo;

- ImprimeContribuicao;
  Recebe uma página e imprime ela no terminal.

- DestroiContribuicao;

- ContribuicaoRetirada;
  Recebe uma contribuição e altera o valor do inteiro ‘retirada’ para 1.

- RetornaStatusContribuicao;
  Recebe uma contribuição e retorna o status ‘retirada’ da contribuição.



Os TADS de listas foram implementados usando o conceito de lista encadeada com sentinela, sendo assim, a sentinela possui um ponteiro que aponta para o primeiro elemento da lista e um ponteiro que aponta para o último, ambos do tipo CelulaDaLista, que contém o conteúdo da lista e um ponteiro que apontará para a próxima célula.

Esses TADS possuem algumas funcionalidades **básicas** a mais:

- IniciaPagina: Função responsável por inicializar a estrutura, alocando espaço para todos os elementos.

- InsereLista: Função responsável por inserir um elemento no final da lista. Recebe uma lista e o elemento a ser inserido na mesma. Nessa função sempre é necessário conferir se estou inserindo um elemento em uma lista vazia, para não ter nenhum problema como NULL-> algumaCoisa, que me resultaria em uma falha na segmentação.

Por fim eu encadeio a minha nova célula no final da lista, fazendo:

lista->ultimo->próximo = nova;

lista->ultima=nova;

nova->prox= NULL

- Retira Lista: Recebe uma lista e um parâmetro chave de busca e então essa função é usada para fazer a retirada de um elemento de uma lista, usando o parâmetro passado para ela no argumento. Nela são tratados todos os possíveis casos, retirar primeiro elemento, último elemento, caso comum e único elemento.

- DestroiLista: Recebe uma lista e libera toda a memória alocada para a mesma.

ListaPagina

Esse TAD é responsável por armazenar as páginas do WIKED, nele existe uma estrutura chamada CelulaPagina, que possui:

- Pagina *pagina;
- ListaContribuicao *contribuicoes;
- ListaLink *links;
- CelulaPagina *prox;

E uma estrutura da lista com sentinela, que possui:

- CelulaPagina *prim;
- CelulaPagina *ult;

Além das funções:

- iniciaListaPagina;

- InsereListaPagina;
  Insere uma página no final da lista.

- RetiraListaPagina;
  Retira uma página da lista.

- imprimeListaPagina;
  Imprime uma lista de páginas no terminal.

- DestroiListaPagina;
  Libera toda a memória alocada para a lista, inclusive a alocada para a lista de links, lista de contribuições e as páginas.

- RetornaPagina;

Recebe uma lista de páginas, uma chave de busca e percorre a lista até encontrar a página e retorna-la ou ate chegar no final da lista.

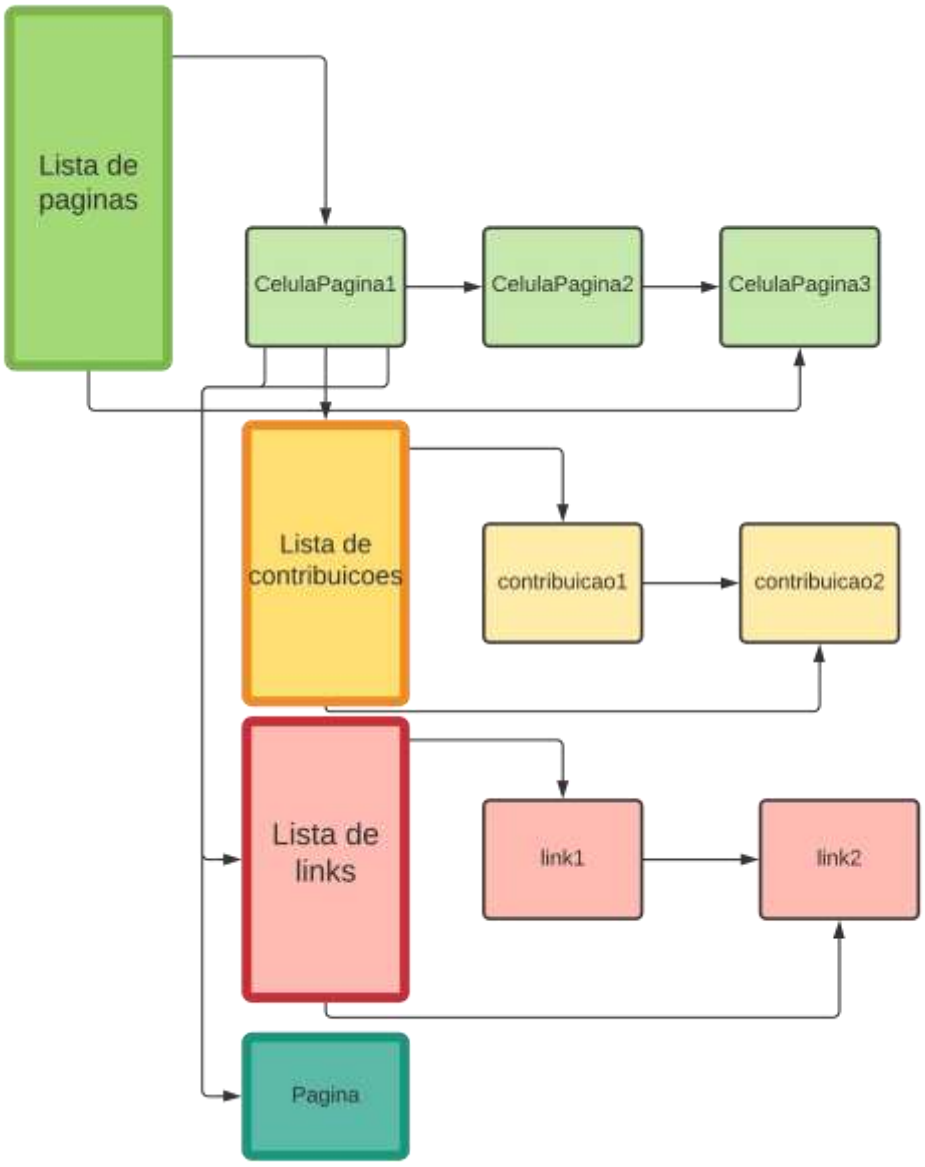
- imprimePaginaCompleta;
  Recebe uma lista de páginas e uma chave de busca para assim imprimir ela no terminal.

- RetornaListaContribuicaoPagina;
  Recebe uma lista de páginas e uma chave de busca para conseguir encontrar a página e retornar a lista de contribuição dela, para por exemplo inserir uma nova contribuição.

- RetornaListaLinkPagina;
  Recebe uma lista de páginas e uma chave de busca para conseguir encontrar a página e retorna a lista de links dela.

- ConfereExistenciaPagina;
  Essa função recebe uma lista de páginas e uma chave de busca para verificar se uma página existe dentro da lista, fazendo uma busca pelo nome recebido no argumento.

- ImprimeWikEd;
  Recebe uma lista de páginas. Essa função é usada para fazer a impressão das páginas nos arquivos .txt, ela é chamada no IMPRIMEWIKED.



ListaEditor

Esse TAD é responsável por armazenar os editores do WIKED, nele existe uma estrutura chamada CelulaEditor, que possui:

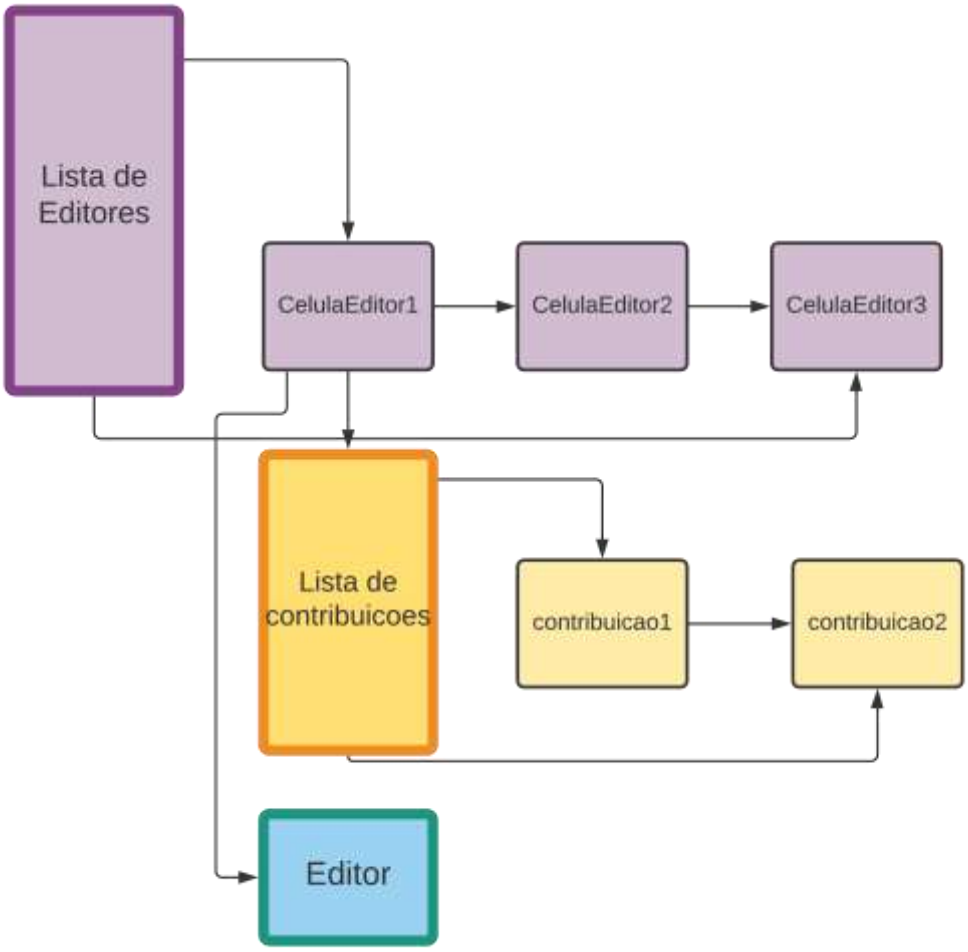
- Editor *ed;
- ListaContribuicao *contribuicoes;
- CelulaEditor *prox;

E uma estrutura da lista com sentinela:

- CelulaEditor *prim;
- CelulaEditor *ult;

Além das funções:

- IniciaListaEditor;
- InsereListaEditor;
- RemoveEditor
- ImprimeListaEditor;
- DestroiListaEditor;
- ConfereExistenciaEditor;
  Essa função recebe uma lista de editores e uma chave de busca para então percorrer a lista de editores e me retornar 1 caso encontre o editor, indicando que ele existe ou 0 caso não o encontre.
- RetornaListaContribuicaoEditor;
  Recebe uma lista de editor e uma chave de busca e então, retorna a lista de contribuição de um editor específico, para por exemplo inserir uma contribuição nela, ou fazer alguma conferencia antes da retirada de uma contribuição.



ListaContribuicao

Esse TAD é responsável por armazenar as contribuições de um editor ou pagina, nele existe uma estrutura chamada CelulaContribuicao, que possui:

- Contribuicao *c;
- CelulaContribuicao *prox;

E uma estrutura da lista com sentinela:

- CelulaContribuicao *prim;
- CelulaContribuicao *ult;

Além das funções:

- IniciaListaContribuicao;

- InsereListaContribuicao;

- RetiraListaContribuicao;
  Função para retirar uma contribuição do jeito tradicional, apagando-a da lista e encadeando os ponteiros, entretanto, como eu preciso do histórico de retiradas ela não é usada, mas mantive a implementação.

- RetiraListaContribuicaoParaHistorico;
  Função que atualiza o status de uma contribuição, retirada = 1, para indicar que essa contribuição já foi retirada.

- ImprimeListaContribuicao
  Imprime uma lista de contribuição no terminal.

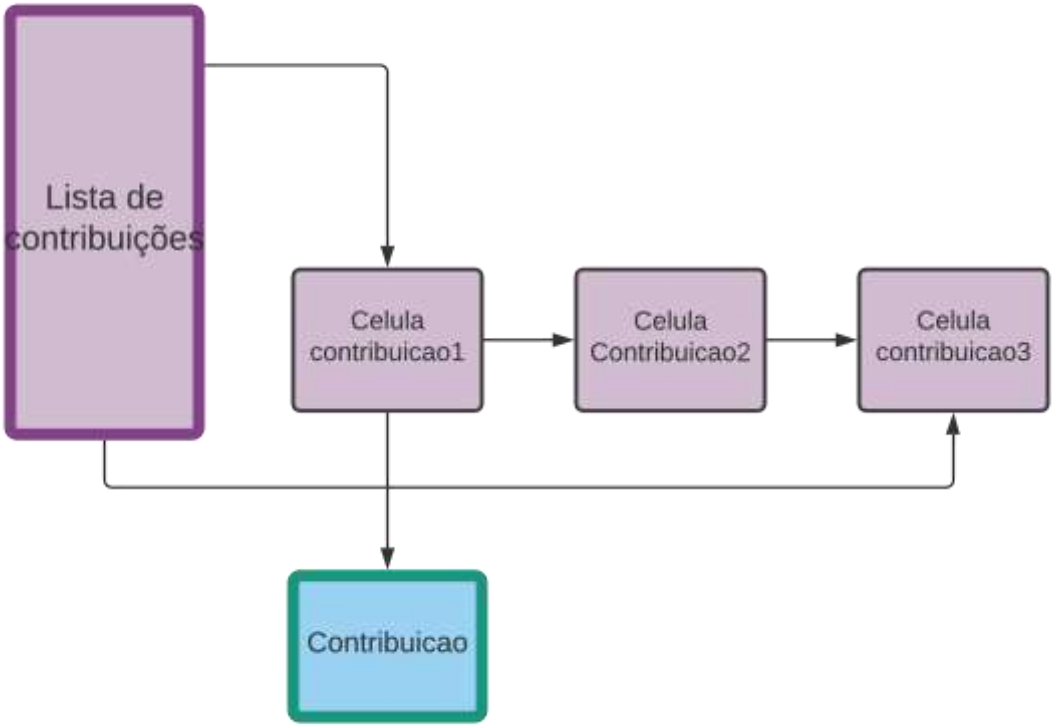
- ImprimeListaContribuicoesParaHistorico;
  Imprime uma lista de contribuição no arquivo .txt.

- imprime;
  Essa função copia um arquivo para outro, por exemplo de um arquivo c1.txt para a página UFES.txt.

- ImprimeListaContribuicaoTextos;
  Nessa função eu percorro toda a lista de contribuição, analisando o status das contribuições e fazendo a chamada da função “imprime”, caso o status da contribuição esteja em retirada = 0.

- ConfereContribuicao;
  Confere se existe essa contribuição na lista de contribuições.

- DestroiListaContribuicao;



ListaLink

Esse TAD é responsável por armazenar os links de uma página, nele existe uma estrutura chamada CelulaLink, que possui:

- Pagina *link;
- CelulaLink *prox;

E uma estrutura da lista com sentinela:

- CelulaLink *prim;
- CelulaLink *ult;

Além das funções:

- IniciaListaLink;
- InsereListaLink;
- RetiraListaLink;
- ImprimeListaLink;
  Impressão da lista de links no terminal.

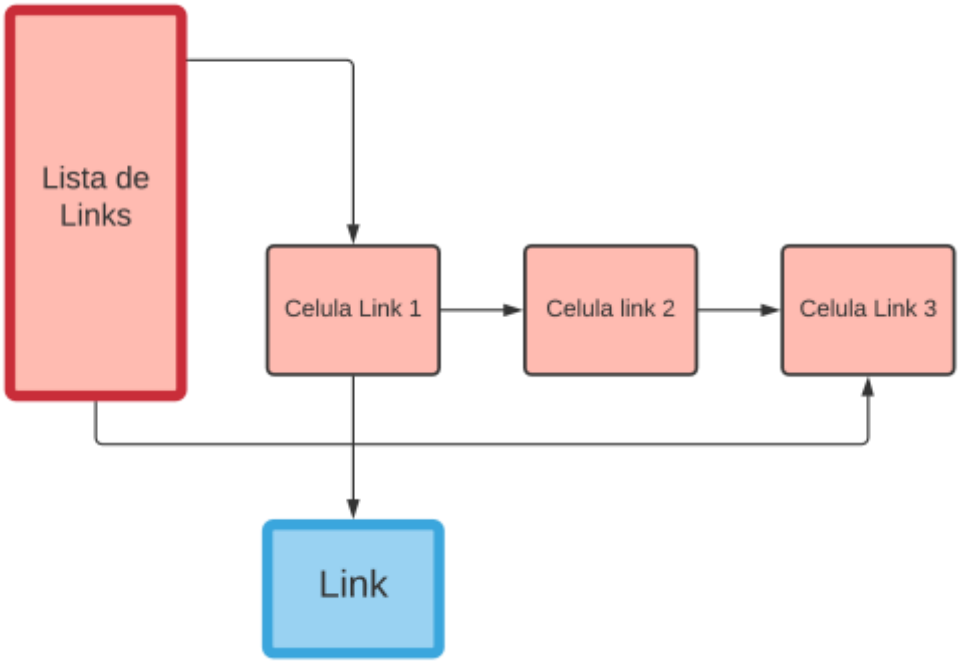
- ImprimeListaLinksArquivo;
  Impressão da lista de links no arquivo .txt.

- DestroiListaLink;

Abaixo as funções que se encontram no TAD e foram implementadas para a funcionalidade **CAMINHO.**

- RetornaProximoLink
  Recebe duas listas de links, a lista de links que eu vou analisar e a lista de links das páginas percorridas até o momento, eu caminho na lista de link e vou comparando se as páginas dessa lista se encontram na lista de links percorridos, se a página não estiver na lista dos percorridos, significa que eu ainda não passei nela e então ela é retornada.

- RetornaLinkLista;
  Recebe uma lista de links e uma chave de busca e partir disso percorre essa lista verificando se encontra um link com o nome passado no argumento da função



Implementação da função CAMINHO

Para implementar a funcionalidade caminho, foi necessário pesquisar o um pouco melhor sobre o algoritmo de grafos e recursão de funções.

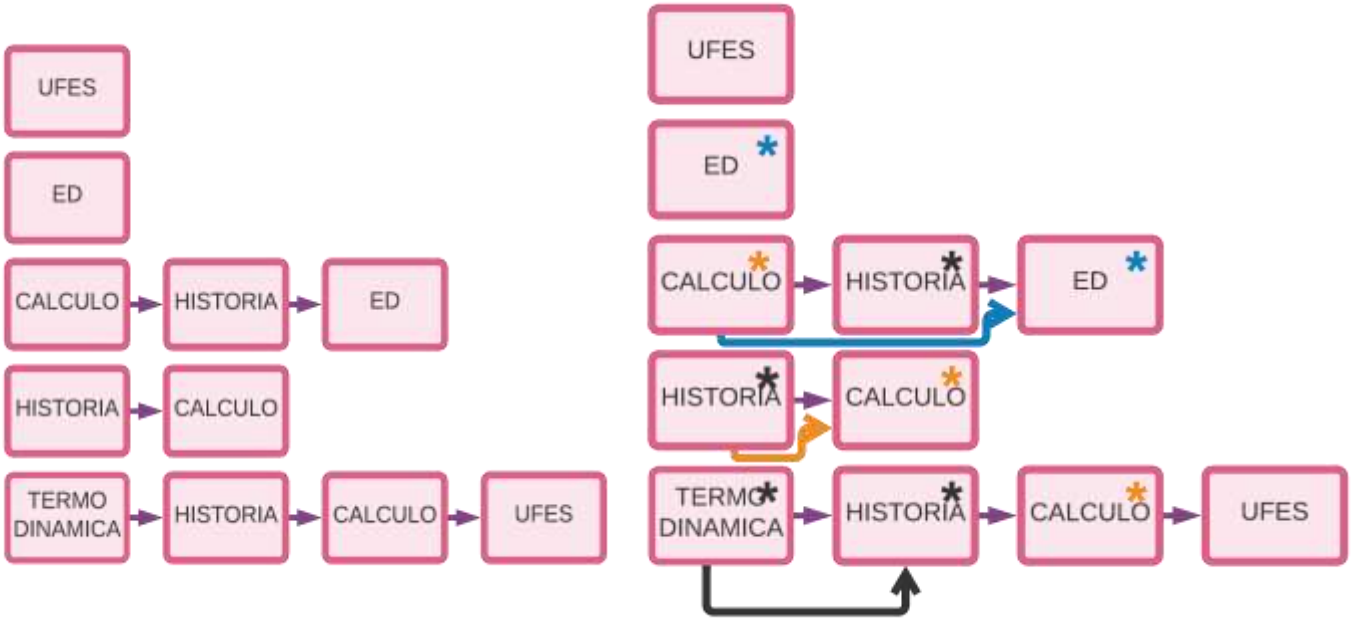
A ideia que eu tive e usei no código foi de, criar uma lista de links auxiliar, para ir armazenando as paginas que eu fosse percorrendo, ou seja, entrei em uma página, armazeno ela na lista e na próxima chamada da função eu analiso se essa pagina já se encontra na minha lista de percorridas, se sim, significa que eu já passei por essa pagina, então eu não posso entrar nela de novo, tenho que passar para a seguinte.

Ao final, eu vou caminhar na lista de links “percorridas” verificando se eu encontro a página que eu quero saber se possui um link, se encontrar significa que em algum momento, partindo da lista de link da página 1, eu consegui chegar nela, e com isso estabelecer um caminho.

Porém, existia um caso que acabava quebrando a lógica do meu programa, que era quando eu ia percorrer uma lista de links que inicialmente já estava vazia ou que já havia sido percorrido todos os links dela, ela retornava NULL e acabava desencadeando sem terminar de percorrer todas as outras listas, então implementei uma função chamada ‘**ConfereLista’,** responsável por fazer essa analise antes de continuar o prosseguimento da função**,** ela faz a verificação e se por acaso se encontrar nessa situação, de lista vazia ou com todos os links já percorridos, ela retorna a lista de links inicial.

Exemplo de funcionamento: CAMINHO TERMODINÂMICA ED

Na primeira coluna temos as paginas da lista de paginas e as que vem ao lado delas, simbolizam os links que essas páginas possuem.



À medida que eu vou percorrendo as listas de links eu vou colocando um asterisco no fluxograma, que seria equivalente a estar adicionando-as na minha lista de links percorridos, e dessa forma eu consigo controlar e saber em quais páginas eu já passei.

Partindo da página TERMODINAMICA,(setas pretas e asteriscos pretos), eu sempre coloco um asterisco na pagina atual, nesse caso TERMODINAMICA, logo depois vamos para HISTORIA e colocamos asterisco em todas as páginas de nome HISTORIA, em seguida, começaremos a percorrer a lista de links da página HISTORIA(seta laranja e asteriscos laranjas), seguindo o mesmo processo, colocando asteriscos em todas as páginas de nome CALCULO e percorrendo a lista de links de CALCULO (seta azul e asteriscos azuis), encontramos HISTORIA, porem ela já tem um asterisco, que equivale a eu já ter percorrido a mesma, então eu pulo ela e encontro ED.

Conclusão

De maneira geral o trabalho serviu para por em pratica o conteúdo visto no semestre sobre as listas encadeadas e tipos abstratos de dados, foi necessário estruturar ele para que as listas estivessem todas ligadas e trabalhando em conjunto. Como maior dificuldade, destaco a implementação da função CAMINHO, que foi necessário um pouco mais de tempo para ser concluída, e se tratava do uso de um algoritmo completamente novo.

Outro ponto que acabou sendo um pouco mais complicado foi a forma que eu iria armazenar os arquivos de contribuições, de inicio eu tinha pensado em armazena-los dentro do código e isso acabou me dando problema então mudei a estratégia para simplesmente armazenar o nome do arquivo, e quando necessário fazer uma cópia de um arquivo para outro.

No mais, as outras funções já haviam sidos trabalhadas em “sala de aula” e não deram trabalhos maiores.

Instruções Makefile

- make compile: compilar o programa
- make clean: limpar os executáveis
- make run: executar o programa já verificando o uso de memória com o VALGRIND.

Referências

Copie um arquivo para outro arquivo em C

https://forgetcode.com/c/577-copy-one-file-to-another-file

Manipulação de arquivos em C:

http://wiki.icmc.usp.br/images/8/82/Manipulacao_arquivos.pdf

https://www.cprogressivo.net/2013/11/Como-ler-arquivos-em-C-As-funcoes-fgetc-fscanf-fgets.html

Algoritmo de Grafos em C:

https://programacaodescomplicada.wordpress.com/tag/grafos/

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/graphdatastructs.html

http://professor.ufabc.edu.br/~leticia.bueno/classes/teoriagrafos/materiais/dfs.pdf