UNIVERSIDADE FEDERAL DE MINAS GERAIS

Faculdade de Ciência da Computação

Trabalho Prático de Programação e Desenvolvimento de Software 2

Júlio Assis

Thales Fernandes

Máquina de Busca

Belo Horizonte

2022

Git-hub:

https://github.com/JulioAssisSouzaAmorim/TP-Maquina-de-Busca

Introdução:

O objetivo desse projeto é fazer uma máquina de busca através da implementação de um código que atua como um subsistema de indexação e recuperação de dados. Esse trabalho foi apresentado após a primeira prova de programação e desenvolvimento de software 2, o que significa que já tínhamos uma noção sobre algumas práticas e algumas estruturas que usaríamos para implementar o código. Essas práticas incluem os bons hábitos de desenvolvimento como a depuração e a modularização, assim como o versionamento ao qual utilizamos o git-hub e a utilização de arquivos de texto na entrada e na saída do nosso programa. As estruturas citadas anteriormente abrangem os diversos tipos abstratos de dados (TADs) como set, string e map que podem ser utilizados através da inclusão de bibliotecas específicas, e também incluem os TADs que nós podemos criar através do nosso próprio código. Outra estrutura importantíssima que já tínhamos contato eram os iteradores que são usados para percorrer e manipular os diferentes tipos abstratos de dados e, consequentemente, são essenciais em praticamente todos os programas que nós produzimos.

Logo após a introdução do trabalho, tivemos aulas de programação defensiva que abrangem conceitos de segurança como o tratamento de exceções e os testes de unidade. Essas duas práticas são valiosas para reduzir as incertezas do código e, segundo o enunciado do trabalho, deverão ser implementadas. A seguir, descreveremos o desenvolvimento do código do projeto.

Implementação:

A nossa máquina de busca será realizada através de dois arquivos de código chamados main.cpp e maquinadebusca.hpp. Esses códigos foram desenvolvidos e executados no ambiente de desenvolvimento Replit e, portanto, o makefile e os dois arquivos do nosso programa são baseados no compilador padrão desse site que é chamado Clang. A main possui uma estrutura set<string> chamada query que passará as palavras que queremos consultar nos documentos para as funções que farão essa busca e também possui uma string k que é iniciada vazia e que vai receber

o valor de cada palavra que desejamos buscar na entrada do programa. Também criamos uma estrutura de repetição while que só interrompe os seus loops quando (cin >> k) é falso, ou seja, quando não há mais palavras de busca na entrada do código. Dentro desse while, nós normalizamos as palavras da entrada através da função normalizar que será explicada posteriormente e utilizamos a função insert que vai inserir o valor da string k no set query. Após realizarmos a inserção de todas as palavras desejadas, o código sai do while e ativa a função buscar(query) que está inclusa através do arquivo maquinadebusca.hpp. Assim, o arquivo main chega ao fim e começaremos a explicar o nosso arquivo header no próximo parágrafo.

O arquivo maquinadebusca.hpp inclui 7 bibliotecas: iostream; set; map; fstream; dirent.h; string; algorithm. As bibliotecas fstream e dirent.h serão detalhadas na descrição das funções que as utilizam. Esse arquivo também possui 12 funções, sendo que as chamadas “normalizar”, “guardar”, “mapear”, “recuperar”, “imprimir” e “buscar” serão responsáveis por atuar como um subsistema de indexação e recuperação de dados e as outras funções serão responsáveis por realizar os testes de unidade do código. A seguir detalharemos o funcionamento das 12 funções.

A função “void normalizar” realiza a normalização de uma string s que é passada como referência. No início da função criamos as variáveis char c e string s2 que serão usadas como auxiliares para manipular e padronizar a string s. Em seguida, a função tem uma estrutura de repetição for que é iniciada com a variável int i com o valor 0, incrementando 1 unidade dessa variável a cada loop, sendo que a condição de parada é (i < s.length( )), ou seja, o nosso for será finalizado quando o valor de i for igual ao número de caracteres da string s. Esse for será usado para remover os caracteres que não sejam letras do alfabeto através de duas etapas. A primeira é igualar a variável c à variável s[i] que é um único caractere da string s. A segunda etapa é realizada através de um condicional if, sendo que a condição para ativar esse if é que a variável c esteja entre os intervalos inteiros de 65 a 90 ou de 97 122, que representa os intervalos do alfabeto minúsculo e maiúsculo da tabela ASCII. Caso esse if seja ativado, significa que a variável c é uma letra do alfabeto e iremos igualar a string s2 a (s2 + c), de forma a guardar apenas as letras de s. Dessa maneira, a primeira parte da normalização está concluída e iremos iniciar a segunda parte que é converter todas as letras da string para minúsculas através de outra estrutura de repetição for. Esse outro for é iniciado com a variável int i com o valor 0, incrementando 1 unidade dessa

variável a cada loop, sendo que a condição de parada é (i < s2.length( )), ou seja, o nosso for será finalizado quando o valor de i for igual ao número de caracteres da string s2. Dentro dessa estrutura, nós igualamos a variável c à s2[i] e temos um condicional if, sendo que a condição para ativar esse if é que a variável c esteja entre os intervalos inteiros de 65 a 90, que representa os intervalos do alfabeto maiúsculo da tabela ASCII. Caso esse if seja ativado, iremos incrementar a variável c com 32 unidades, tornando essa letra maiúscula em minúscula, e iremos igualar s2[i] à variável c. Após esse condicional if, o for será repetido para todos os caracteres de s2 e será finalizado. Depois desse processo, iremos igualar a string s à string s2 e como estamos passando s por referência, ela será alterada em qualquer parte do código que a chamarmos, concluindo o processo de normalização.

A função “bool teste_de_minusculizacao” testa a funcionalidade da função normalizar de converter letras maiúsculas para minúsculas. Para isso, criamos uma string que possui uma letra maiúscula e utilizamos a função normalizar nessa string. Caso todas as letras da string forem minúsculas retornamos true, caso contrário retornamos false.

A função “bool teste_de_exclusividade” testa a funcionalidade da função normalizar de remover caracteres que não sejam letras do alfabeto. Para isso, criamos uma string que possui letras, números, letras acentuadas e símbolos diversos e, em seguida, utilizamos a função normalizar nessa string. Caso a string resultante contenha apenas letras não acentuadas retornamos true, caso contrário retornamos false.

A função “set<string> guardar” irá salvar em uma estrutura set<string> chamada nomes, criada no início da função, todos os nomes dos arquivos de texto que estão dentro da pasta documentos. Isso será muito útil para podermos manusear os documentos individualmente através de iteradores em outras funções, sendo extremamente importante tanto no subsistema de indexação quanto no subsistema de recuperação. Para realizar esse processo, inicializaremos uma estrutura que representa o fluxo do nosso diretório através do comando (DIR *dir) em que DIR é a função incluída através da biblioteca dirent.h que abre e retorna um ponteiro para esse fluxo e *dir é o ponteiro pelo qual será possível abrir a nossa pasta documentos. Em seguida, também inicializaremos o comando (struct dirent *ent) que cria um ponteiro de uma estrutura que irá abrir e ler os arquivos do diretório criado anteriormente, sendo que essa estrutura possui um atributo do tipo char chamado d_nome que será inicializado com o nome dos documentos do diretório. Posteriormente, nós temos um

condicional if, sendo que a condição para ativar esse if é que o ponteiro dir abra a pasta documentos através da função opendir. Dentro desse condicional, há uma estrutura de repetição while em que a condição de loop é que o ponteiro ent esteja lendo os arquivos de dir através da função readdir. Dentro do while, há um if que é ativado caso a função teste_de_anomalia for verdadeira e um else caso contrário. Se o else for ativado, o nome dos arquivos contidos na pasta documentos será inserido no objeto nomes através da função insert. Ao final do else, saímos da estrutura while e voltamos para o primeiro condicional if e, então, fechamos o diretório através do comando closedir. Finalmente, usamos o comando return nomes.

A função “bool teste_de_anomalia” recebe, como parâmetro, a string d_name enviada dentro da função guardar, explicada anteriormente. A questão é nas duas primeiras tentativas de acesso do atributo d_name do ponteiro ent, esse atributo é igualado aos caracteres ‘.’ e ‘. .’. Apenas na terceira tentativa de acesso que d_name recebe o nome do arquivo. Dessa forma, essa função irá retornar true se a string d_name for ‘.’ e ‘. .’ e irá retornar false caso contrário. Isso fará com que esses dois caracteres não sejam inseridos dentro do objeto nomes da função guardar.

A função “map<string, set<string>> mapear” guarda todas as palavras de cada documento listado na estrutura set<string> nomes, recebida como parâmetro, dentro da estrutura map<string,set<string>> indice, recebida como parâmetro, para que possamos saber se as palavras buscadas na entrada do programa estão presentes em determinados arquivos ou não, sendo que o key value desse map é o nome do documento de texto e o mapped value são as palavras contidas nesse documento. Dessa forma, iniciaremos a função criando um objeto fs da classe ifstream que vai realizar operações de entrada e saída de dados a partir da abertura dos nossos arquivos. Em seguida, criamos uma variável chamada palavras do tipo set<string> que irá guardar todas as palavras de cada documento e depois será inserida dentro da estrutura indice. Depois, a função tem uma estrutura de repetição for que é iniciada com a estrutura auto it que recebe o valor de nomes.begin( ), incrementando 1 unidade dessa estrutura a cada loop, sendo que a condição de parada é (it!=nomes.end( )), ou seja, o nosso for será finalizado quando o iterador tiver percorrido todos os arquivos de texto. Dentro desse for, nós igualamos a string n à string “./documentos/” somada com *it que será o nome do arquivo de texto. Depois abrimos esse arquivo através do objeto fs com a função fs.open(n) e criamos uma estrutura de repetição while em que

a condição de loop é que (fs >> l), ou seja, iremos atribuir uma palavra do documento aberto à string l até que tenhamos percorrido todo o arquivo. Dentro desse while, iremos ativar a função normalizar para padronizar a string l e iremos inserir essa string normalizada na estrutura palavras. Depois desse while iremos fechar a stream com a função fs.close( ) e iremos inserir o par {*it, palavras} dentro de indice. Finalmente, finalizamos a estrutura for e retornamos a estrutura indice.

A função “bool teste_de_mapeamento_vazio” testa se a função mapear retorna um mapa vazio caso o “set<string> nomes” seja vazio. Para isso, criamos um auxiliar “map<string,set<string>> index” e um auxiliar “set<string> names” que são os parâmetros da função detalhada anteriormente, sendo que não inserimos nada dentro deles. Em seguida, atribuímos a ativação da “mapear” ao parâmetro index. Finalmente, retornamos false caso o tamanho de index for maior que 0 e retornamos true caso contrário.

A função “set<string> recuperar” recebe os objetos “set<string> query” e “map<string,set<string>>indice” e confere em quais arquivos as palavras pesquisadas contidas em query se encontram, inserindo o nome desses arquivos dentro do objeto “set<string> dados”, criado no início da função. Para fazer isso, nós criamos uma variável inteira o e uma estrutura de repetição for que é iniciada com a estrutura auto it. Esse iterador recebe o valor de índice.begin( ), incrementando 1 unidade desse it a cada loop, sendo que a condição de parada é (it!=índice.end( )), ou seja, o nosso for será finalizado quando o iterador tiver percorrido todos os pares do nosso map. Dentro desse for, temos uma estrutura condicional if que é ativada caso o número de palavras contidas no mapped value relacionada ao arquivo do key value for menor que o número de palavras contidas no query. Caso esse if seja ativado, a função não realiza nada, já que esse arquivo não pode conter todas as palavras de query e, consequentemente, não satisfaz as condições para ser um documento relevante. Porém, se esse if não for ativado, iremos cair em um condicional else que iguala a variável inteira o ao valor 1 e possui um outro condicional for. Esse segundo for dentro do else é iniciado com a estrutura auto itr recebendo o valor de query.begin( ), incrementando 1 unidade desse itr a cada loop, sendo que a condição de parada é (itr!=query.end( )), ou seja, o nosso for será finalizado quando tivermos percorrido todas as palavras da busca. Dentro desse segundo for, nós teremos dois condicionais if, sendo que o primeiro é ativado caso as strings contidas em query não estiverem

contidas no índice, o que significa que o documento mapeado pelo índice não é um documento relevante. No caso desse primeiro if ser ativado, a variável o é igualada a 0, o que faz com que o segundo if não seja ativado. Caso contrário, esse segundo if insere o nome do arquivo contido no key value de índice dentro do objeto dados. Finalmente, após todos os loops serem finalizados retornaremos o objeto dados.

A função “bool teste_de_recuperacao” testa se a função “recuperar” detalhada anteriormente retorna um objeto vazio caso os termos pesquisados só possuam caracteres especiais. Para isso, criamos estruturas auxiliares chamadas “set<string>pesquisa”, “map<string,set<string>> index”, “set<string> names” e “set<string> data”. Em seguida, inserimos strings que contém apenas caracteres especiais dentro do set pesquisa. Depois, atribuímos a função “guardar” ao set names e atribuímos a função mapear(index, names) ao map index. Seguidamente, atribuímos a função recuperar(pesquisa, index) ao set data. Finalmente, se o tamanho de data for maior que 0 retornamos false, caso contrário retornamos true.

A função “void imprimir” realiza a impressão do nome de todos os documentos relevantes contidos em “set<string> dados”, que é passado como parâmetro. Essa impressão é realizada através de uma estrutura de repetição for que percorre todos os elementos do set dados e imprime os valores dos elementos a cada interação.

A função “resultados testes” cria e retorna um struct do tipo resultados chamada r, determinando se o código passa ou não nos testes de unidade, sendo que o teste de anomalia já é realizado pela função guardar. Esse struct possui 5 atributos bool chamados t1, t2, t3, t4 e t0. Na função, nós atribuímos o resultado das funções teste_de_exclusividade, teste_de_minusculizacao, teste_de_mapeamento_vazio e teste_de_recuperacao, respectivamente, aos atributos t1, t2, t3 e t4 da struct r. Seguidamente, se todos esses atributos forem iguais a true, a função iguala o atributo t0 de r a true e, caso contrário, iguala o atributo t0 de r a false. Finalmente, retornamos r.

A função “void buscar” recebe “set<string> query” como parâmetro e chama todas as funções responsáveis por atuar como um subsistema de indexação e recuperação de dados ou informa quais testes de unidade falharam. Para isso, atribuímos a função testes, detalhada anteriormente, a um struct resultados chamado r. Caso o atributo t0 de r for false o código não realiza a busca e imprime quais dos testes de unidade não

tiveram sucesso. Caso o atributo t0 for true nós criamos as estruturas “set<string> nomes”, “map<string,set<string>> índice” e “set<string> dados”. Em seguida, nós atribuímos o retorno da função guardar ao set nomes, atribuímos o retorno da função mapear(índice, nomes) ao map índice e atribuímos o retorno da função recuperar(query, índice) ao set dados. Finalmente, chamamos a função imprimir(dados) e concluímos o processo da máquina de busca.

Conclusão:

Esse trabalho abordou uma ferramenta extremamente importante para o funcionamento de todas as áreas da nossa sociedade. A capacidade de buscar informações transformou diversos aspectos da vida de cidadãos com acesso à internet como compras e vendas on-line, como a grande facilidade de obter informações no geral, entre outras inúmeras utilidades. Dessa maneira, o presente trabalho foi uma experiência muito interessante para o nosso grupo, já que, além de pôr em prática diversos conceitos de programação abordados em sala de aula, nos introduziu ao ramo de análise de dados que é muito importante para o nosso curso. Muitas partes da implementação do projeto serviram para a fixação de conteúdo que já havíamos usado antes em VPLs, como o uso de tipos abstratos de dados e o uso de programação defensiva através de testes de unidade. Outras partes da implementação serviram para incorporarmos uma parte do conteúdo que não havíamos praticado muito nos VPLs, como o controle de versão através do git-hub e como o manuseamento de arquivos e diretórios. No geral, o TP foi uma experiência de aprendizado desafiante e positiva.