Universidade Federal de Minas Gerais
Programação e Desenvolvimento de Software 2

Máquina de Busca

Integrantes do Grupo:
André Lara Pinto
Bruno Nicolau Machado
Hermon Dória Vasconcelos Barros
Leticia Maia Silva Araújo

Introdução:

O Trabalho Prático teve como objetivo programar uma Máquina de Busca, na qual seria implementada uma estrutura de dados capaz de ler múltiplos arquivos, tratá-los da maneira desejada e armazenar todas as informações necessárias. Além disso, tais informações tiveram que ser manipuladas e interpretadas, a fim de obtermos os resultados esperados.

Por ser um programa que trabalha com uma quantidade enorme de dados, informações, funcionalidades e problemas, foi essencial buscar a melhor forma de lidar com todas essas questões. Seria impossível realizar com sucesso uma Máquina de Busca sem otimizar o acesso, tratamento e armazenamento dos dados, a resolução dos problemas e a implementação de todo o código.

Tivemos que usar muito do nosso conhecimento adquirido durante esse semestre, como programação orientada a objeto com utilização de classes, tipos abstratos de dados, leitura de arquivo, modularização e controle de versionamento de código, testes, entre outros.

Implementação:

Divisão de arquivos:

Fizemos o arquivo de cabeçalho (header) baseDados.h, que contém a interface de toda a classe BaseDados, inclusive sua definição e os cabeçalhos de todos os seus métodos.

Para a implementação da classe BaseDados, com todos os seus métodos, fizemos o arquivo baseDados.cpp.

A fim de testarmos o funcionamento do projeto, incluímos o arquivo doctest.h na solução e fizemos o arquivo teste.cpp. Nele, implementamos a classe Teste, cujo método acessa a variável privada da classe BaseDados, para que pudéssemos testá-la propriamente. Além disso, no arquivo teste.cpp, estão os testes de cada método da classe, que levam em conta vários casos possíveis na execução das funções.

Implementação do Índice Invertido:

Implementamos o índice invertido na forma de uma variável do tipo map. As chaves do mapa são representadas por cada palavra lida de cada arquivo, ou seja, são do tipo string. Os valores do mapa são os conjuntos de arquivos onde a palavra-chave aparece, isto é, são do tipo set<string>. Dessa forma, o índice invertido é do tipo: map<string, set<string>>.

Para implementarmos esse índice, fizemos três métodos dentro da classe BaseDados:

void ler_inserir_Arquivos( ): lê a coleção de arquivos e trata as palavras contidas neles para que todos os seus caracteres sejam letras minúsculas ou dígitos (elimina os símbolos). Depois de tratadas, todas as palavras e seus respectivos arquivos são inseridas no mapa do índice invertido.

void inserir(string palavra, string documento): insere no índice invertido uma palavra e um arquivo específicos, que são passados como parâmetros.

void imprimirIndice( ): imprime todas as chaves e seus respectivos valores do índice invertido.

Além disso, criamos métodos da classe BaseDados para adquirirmos algumas das informações essenciais para definirmos as coordenadas dos documentos e as operações necessárias para a implementação do ranking cosseno.

São eles:

int ocorrenciasArquivo(string palavra, string documento): lê o documento cujo nome é passado como parâmetro e retorna o número de vezes que a palavra recebida ocorre nele.

bool pertence(string palavra): retorna uma variável booleana, que diz se a palavra recebida está presente em qualquer um dos documentos da coleção.

map<string, int> frequencia(string palavra): retorna um mapa cuja cada chave é o nome de cada arquivo onde ela está presente. Seus valores são o número de vezes que a palavra ocorre no respectivo arquivo da chave. O mapa retornado será acessado para obtermos o *term frequency (*tf).

int numeroArquivos(string palavra): retorna o número total de arquivos em que a palavra recebida como parâmetro ocorre em toda a coleção de documentos.

bool existe(string palavra, string documento): teste se a palavra passada como parâmetro está presente no documento passado como parâmetro.

double calculaIdf(string palavra): retorna o logaritmo natural do número total de documentos, dividido pelo número de documentos em que a palavra aparece.

```cpp
int BaseDados::ocorrenciasArquivo(string palavra, string documento) {
    ifstream inFile;
    string palavra_no_arq;
    int contador = 0;

    inFile.open("C:\\Users\\André\\Documents\\UFMG\\PDS II\\" + documento + ".txt");
    if (!inFile) {
        cerr << "O arquivo" + documento + "nao foi aberto";
    }
    else {
        while (inFile >> palavra_no_arq) {
            for (int i = 0; palavra_no_arq[i] != '\0'; i++) { // Loop para eliminar simbolos e letras maiusculas das palavras do arquivo
                palavra_no_arq[i] = tolower(palavra_no_arq[i]);
                palavra_no_arq.erase(remove_if(palavra_no_arq.begin(), palavra_no_arq.end(), [](char c) { return !isalpha(c) && !isdigit(c); }), palavra_no_arq.end());
            }
            if (palavra == palavra_no_arq) {
                contador++;
            }
        }
    }
    inFile.close();
    return contador;
}

bool BaseDados::pertence(string palavra) {
    return (mapa.find(palavra) != mapa.end());
}

map<string, int> BaseDados::frequencia(string palavra) {
    set<string> arquivos;
    map<string, int> tf;
    set<string>::iterator it_arqs;
    map<string, int>::iterator it_tf;
    if (this->pertence(palavra)) {
        arquivos = mapa[palavra];
    }

    for (it_arqs = arquivos.begin(); it_arqs != arquivos.end(); it_arqs++) {
        tf[*it_arqs] = this->ocorrenciasArquivo(palavra, *it_arqs);
    }

    for (it_tf = tf.begin(); it_tf != tf.end(); it_tf++) {
        cout << "\nChave: " << it_tf->first;
        cout << "\nValor: " << it_tf->second << endl;
    }

    return tf;
}

int BaseDados::numeroArquivos(string palavra) {
    set<string> arquivos;

    if (this->pertence(palavra)) {
        arquivos = mapa[palavra];
    }
    //cout << "Aparece em: " << arquivos.size() << endl;
    return arquivos.size();
}
```

Variáveis privadas:

Colocamos o mapa que representava o índice invertido como variável privada da classe, pois ela precisa ser acessada por diversas funções públicas.
Além disso, tem o nomeDocumentos, que retorna um vector com o nome de todos os documentos da coleção.
Para a parte do ranking cosseno precisamos usar a variável map coordenadas, que assume todas as coordenadas de todas as palavras em relação a todos os documentos e assume também as coordenadas das palavras da busca feita pelo leitor em relação a todas as palavras.

Testes:

No arquivo que fizemos para os testes, testes.cpp, implementamos a classe Teste. Ela possui métodos que recebem uma variável do tipo BaseDados e retornam a variável privada da classe, para podermos acessá-las para testarmos.

Os métodos são:

```
class Teste {
public:
    static map<string, set<string>> mapa_(const BaseDados& c) {
        return c.mapa;
    }
    static vector<string> nomeDocumentos_(const BaseDados& c) {
        return c.nomeDocumentos;
    }
    static map<int, double> coordenadas_(const BaseDados & c){
        return c.coordenadas;
    }
};
```

Também programamos testes que avaliam o funcionamento das funções da classe BaseDados nos mais diversos casos que podem ocorrer.

Tabela Hash e Ranking Cosseno:

Foi criada um Método Hash (executado pelo construtor)para criar e armazenar o sistemas de coordenadas. Nele cada palavra se comporta como uma coordenada para algum vetor Dj (certo documento). O sistema de coordenadas é armazenado em um Map(Coordenadas) declarado como um atributo privado , em que cada chave é a multiplicação do total de palavras (Numeropalavras) pelo número da posição de certa palavra (x), somada ao número do documento (y) em questão.
Chave = x*(Numeropalavras) + y.

	P1	P2	P3...
D1	IdfxTf	IdfxTf	IdfxTf
D2	IdfxTf	IdfxTf	IdfxTf
D3...	IdfxTf	IdfxTf	IdfxTf

Cada célula possui uma chave única.

Assim uma coordenada de algum documento no eixo de alguma palavra possuía um valor que nunca se repete. Já o valor para esta chave era a multiplicação de Tf com o Idf. A função hash possui um laço for de repetição que passa por todas as palavras (elas foram pegadas com um iterator no índice invertido inicial) na qual, para cada palavra ele preenche o valor para cada documento e gera a respectiva chave (um for() implementado dentro do primeiro para passar do primeiro documento até o último).

Dentro do segundo for() é chamado a função frequência(alguma palavra) que retorna um Map que cada chave é um o número do documento em que ela aparece e o valor é quantas vezes ela aparece. Deste modo, com um if e um else ele olha se o primeiro documento tá no map da frequência, se ele tiver quer dizer que o tf não é zero e assim ele calcula o valor IdfxTf , se ele nao tiver quer dizer que nesse eixo dessa palavra o documento não existe, assim o valor vira zero automaticamente.

Esse if/ else pode ser visto abaixo:

```
map<int, double> BaseDados::hash() {
    int numeroPalavra=50000;
    vector<string> = nomeDocumentos;
    map<int, double>aux;
    auto iAux = aux.begin();
    vector<string>::iterator i;//iterator referente ao vector com nome dos documentos

    for (auto it = mapa.begin(), int x ; it != mapa.end();it++,x++){ // Avanca o map<string,set<string>
        string palavra = it->first;
        map<string, int> resultado = frequencia(it->first);
        auto docFrequencia = resultado.begin(); //Mapa (doc,frequencia)
        for(i = nomeDocumentos.begin(); != nomeDocumentos.end();i++){
            int valorVetor = atoi(i->c_str());
            int valorMap = atoi(x->first->c_str());
        if((valorVetor == valorMap ) {
            iAux->fist = x*numeroPalavra + valorVetor;
            int tf = ocorrenciasArquivo(palavra,*i);
            iAux->second = importancia(palavra)*tf;//pegar valor frequencia e botar idf
        }
        else {
            iAux->fist = x*numeroPalavra + valorVetor;
                iAux->second = 0;
        }
    }


    return aux;
}
```

Logo após sobrecarregamos o método hash a fim de calcular o vetor da string de entrada fornecida pelo usuário, portanto o map que inter-relaciona os vet foi finalmente construído.

O método rankingCoss( ) foi implementado com o intuito de fazer o cosseno(distância) entre todos os vetores documentos (Dj) com o vetor relacionado com a pesquisa Q. Sendo assim, foi criado somente um laço repetitivo for() que passa por todas as palavras existentes e gera os 3 Somatórios presentes na seguinte fórmula para cálculo da similaridade(cosseno):

$$sim(d_j, q) = \cos(\theta) = \frac{\sum_{i=1}^{t}(W(d_j, P_i) \times W(q, P_i))}{\sqrt{\sum_i W(d_j, P_i)^2} \times \sqrt{\sum_i W(q, P_i)^2}}$$

Depois disso as raízes são calculadas e a divisão também, gerando um valor de similaridade SimDQ para o documento em questão. Como existe um for() por cima que passa por todos os documentos, todos os valores SimDQ são gerados e armazenados em um novo Map SIM que possui a chave igual a similaridade e o valor igual ao número do documento analisado. Este Map foi feito deste jeito para que as chaves fiquem em ordem automaticamente e o ranking seja facilmente impresso do documento mais similar para o menos similar.

```
map<int, double> BaseDados::rankingCoss(string busca) {
    map<int, double> aux;
    int numero_palavras = mapa.size();
    double Somatorio1, Somatorio2, Somatorio3 = 0;
    vector<string>::iterator i;
    map<int, double>::iterator j;
    for (i = nomeDocumentos.begin(); i != nomeDocumentos.end(); i++) {
        for (int x = 1; x <= numero_palavras; x++) {
            j = coordenadas.find(x * numero_palavras + atoi(i->c_str()));
            double A = j->second;
            j = coordenadas.find(x * numero_palavras + 1);
            double B = j->second;
            Somatorio1 += (A * B);
            Somatorio2 += (A * A);
            Somatorio3 += (B * B);
        }
        double SimDQ = Somatorio1 / (sqrt(Somatorio2) * sqrt(Somatorio3));
        aux.insert(pair<>)
            Joga no map de similiaridade com valor = atoi(i->c_str()  e chave = SIMDQ
    }  //somatorio3 inferior 3

    joga similiaridade em um set que possuia a similiaridade de cada doc
}
}
```

Já o método ImprimirRanking( ) citado acima pega o Map com a similaridade e Imprimi na tela o documento com a maior similaridade para o de menor. Para isto foi usado a função rbegin() que pega o iterator do último elemento e considera ele como o primeiro, assim, o último documento Map (O mais similar à pesquisa Q) será o primeiro a ser impresso.

Conclusão:

Por ser um trabalho prático de alta complexidade e extensão, Máquina de Busca exigiu um planejamento estratégico por parte da equipe para que conseguíssemos implementá-la. A execução do trabalho se torna muito mais simples e organizada quando pré-definimos as tarefas de cada integrante do grupo e usamos uma plataforma de controle de versionamento de código (GitHub), mais especificamente a aplicação GitKraken .

Foi necessário planejar qual estrutura de dados seria melhor para cada situação dentro projeto, para que o armazenamento, manipulação, organização e acesso dos dados fossem otimizados.

A programação orientada a objeto foi essencial para a implementação das funcionalidades do projeto, com utilização de tipos abstratos de dados e a biblioteca padrão de C++. O código ficou mais claro e organizado, fator de grande importância quando se trata de um programa extenso e complexo.

Além disso, por ser um trabalho com tantas funcionalidades, aprender a realizar testes de unidade foi muito relevante. Tanto escrever testes ao longo do código para verificar a compilação e os eventuais erros, quanto escrever testes para o funcionamento de métodos com vários casos possíveis (doctest).

Sendo assim, aprendemos muito sobre estruturas de dados, que poderemos utilizar em linguagens que vão muito além do C++. O contato com novas técnicas e estratégias de programação, além do GitHub, também foi muito agregadora.