

# Trabalho Prático de PDS2: Máquina de Busca

Pedro Dalla-Lana 2020420613

Daniel Santos 2022037620

Repositório:

<https://github.com/gourox/Trabalho-Pr-tico-PDS2---Daniel-Santos---Pedro-Dalla.git>

## Documentação:

### 1. Introdução

O Trabalho Prático de PDS2 tem como objetivo principal implementar os subsistemas de Indexação e Recuperação de uma máquina de busca (usualmente subdividido em 3 etapas: Coleta, Indexação e Recuperação). O segmento de coleta já foi realizado previamente.

O desenvolvimento conjunto pelo github teve o auxílio da ferramenta Github Desktop para a sincronização, pull e push do repositório.

---

### 2. Desenvolvimento

De início, tivemos a intenção de implementar duas classes distintas, uma de normalização e outra de pesquisa, originalmente classe `normaliza_palavra` operava da seguinte maneira:

Recolhe a string de dados requeridas e realiza o tratamento indicado no enunciado. Houveram problemas para normalizar todos os caracteres, pois o tipo `char` reconhece apenas valores ascii até 127 (128-1). O que dificulta o tratamento de caracteres como vogais acentuadas e cedilha. Realizamos o tratamento em duas funções, uma para excluir os caracteres especiais e numéricos, mais outra para transformar todos os caracteres em minúsculo usando a biblioteca `<cctype>`.

Transferimos-a para dentro da própria classe `pesquisa`, tendo em vista que economiza em linhas de código.

A compilação inicialmente foi realizada através de um `task.json`.

```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL
o Executing task: g++ -std=c++17 main.cpp pesquisa.cpp doctest.h -lstdc++ && ./a.out
Insira a palavra a ser pesquisada
```

```
pesquisa_teste.cpp tasks.json doctest.h
projeto > .vscode > {} tasks.json > ...
1 {
2     // See https://go.microsoft.com/fwlink/?LinkId=733558
3     // for the documentation about the tasks.json format
4     "version": "2.0.0",
5     "tasks": [
6         {
7             "label": "Compilar e Rodar",
8             "type": "shell",
9             "command": "g++ -std=c++17 main.cpp pesquisa.cpp -lstdc++ && ./a.out && rm a.out",
10            "problemMatcher": [],
11            "group": {
12                "kind": "build",
13                "isDefault": true
14            }
15        },
16
17        {
18            "label": "Compilar e Testar",
19            "type": "shell",
20            "command": "g++ -std=c++17 pesquisa_teste.cpp pesquisa.cpp doctest.h -lstdc++ && ./a.out",
21            "problemMatcher": [],
22            "group": {
23                "kind": "build",
24                "isDefault": false
25            }
26        }
27    ]
28 }
```

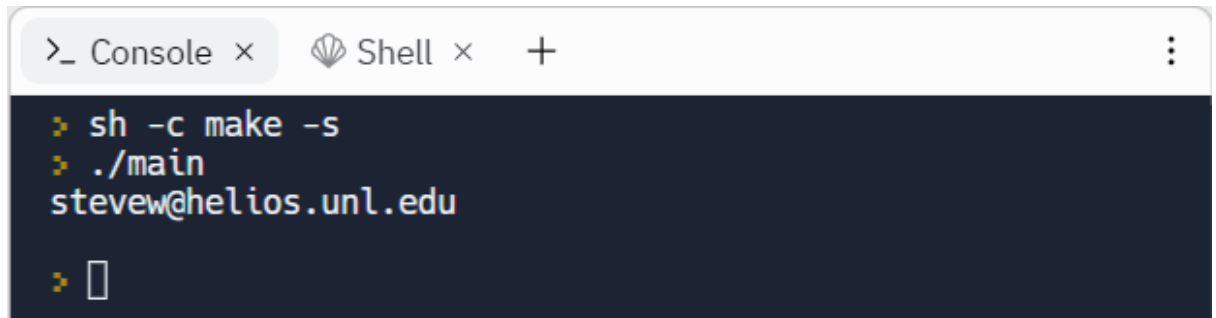
### 3. Implementação

**pesquisa:** Cria um índice invertido do tipo `map<string, set<string>`, ou seja, um mapa que liga uma determinada palavra chave a uma lista de documentos da qual ela pertence. O mapa é criado através do uso das bibliotecas “filesystem” e “fstream” para iterar sobre todos os arquivos da pasta /documentos. Além disso, o método interno `pesquisar()` envia como output na entrada padrão todos os documentos relevantes para determinada string passada como parâmetro.

**ordenar:** recebe como parâmetro uma lista de string e retorna um vetor lexicograficamente ordenado. Utiliza a função `sort()` do STL.

**pesquisar:** recebe como parâmetro uma string e utiliza a função `find()` da biblioteca `map` para procurar a lista correspondente a string normalizada (utilizando a função `normalizar()`) e então retorna um vetor ordenado lexicograficamente utilizando-se na função `ordenar()` da classe.

normaliza:

A terminal window with a dark background. The title bar shows 'Console' and 'Shell' tabs. The prompt is '>'. The user enters 'sh -c make -s', followed by './main'. The output is 'steve@helios.unl.edu'. The prompt returns to '>'.

Recolhe a string de dados requeridas e realiza o tratamento, como mostra o exemplo acima.

Casos de exceção: foram utilizados blocos try-catch na função main com os structs que definiam possíveis erros, dentre eles:

**CaminhoInexistente:** chamado quando o caminho dado na função pesquisa() não existe;

**PastaVazia:** chamado quando o caminho dado na função pesquisa() não possui arquivos;

**PalavraInexistente:** chamado quando a palavra dada na função pesquisar() não existe em nenhum documento do diretório.

Testes: Foi utilizado o framework doctest para a realização dos testes de unidade da classe pesquisa.

---

## 4. Conclusão

O grupo em seu Trabalho Prático obteve o ranqueamento com sucesso, e apesar de algumas falhas percebidas em casos-teste, o programa realiza a máquina de busca.

Houve um certo atraso na implementação da função de normalização, tendo em vista que a exibição de caracteres especiais varia de sistema operacional para sistema operacional, sendo necessário utilizar a biblioteca <locale> setlocale(LC\_ALL, "pt\_BR\_utf8") para exibir caracteres como “ç” (c cedilha).

Além disso, também houveram dificuldades na criação do makefile, tendo em vista que a biblioteca *filesystem* requer um tipo especial de compilação.

Além disso, também houveram dificuldades na criação do makefile, tendo em vista