Universidade Federal de Uberlândia - Campus Monte Carmelo
Faculdade de Computação - Sistemas de Informação
GSI521 - Organização e Recuperação da Informação

Prof. Dr. Murillo G. Carneiro
Phelipe Rodovalho Santos

Trabalho 1: Índice invertido e modelo booleano

Visão Geral

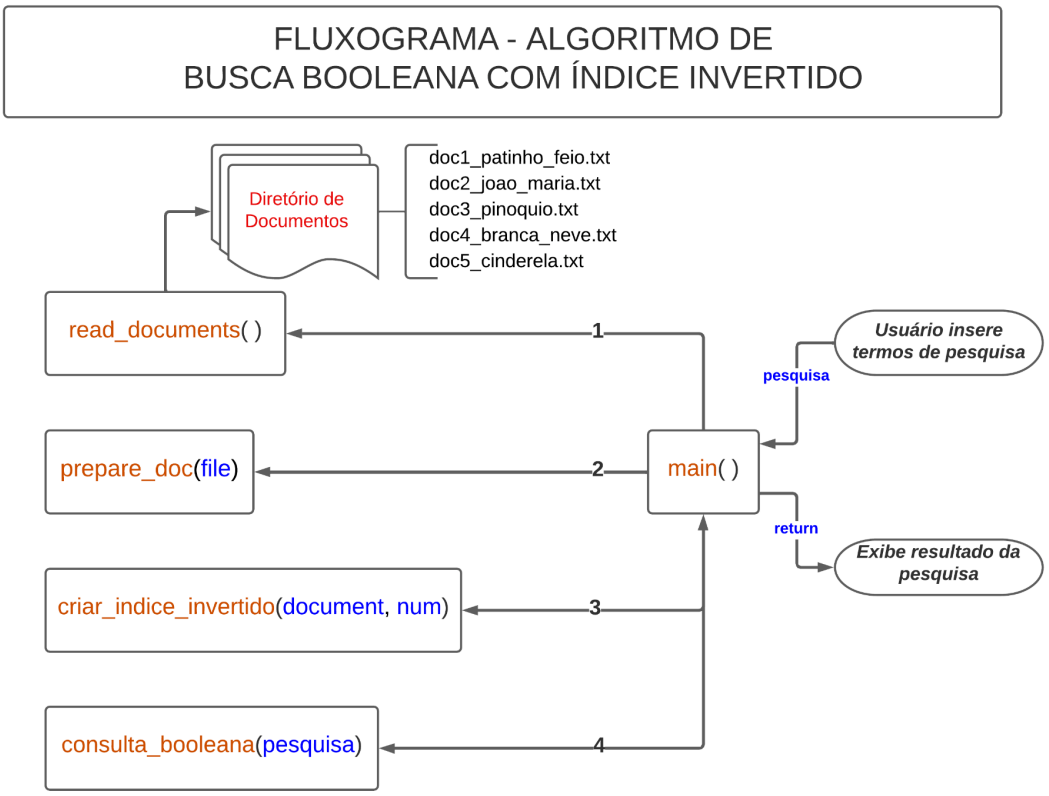
A eficiência de um sistema de recuperação da informação é diretamente associada ao modelo que se utiliza, apesar dos primeiros modelos terem sido idealizados e criados nos anos 60, seus principais mecanismos ainda estão presentes em grande parte dos sistemas de recuperação atuais. O modelo booleano se baseia na teoria dos conjuntos e álgebra booleana, permitindo buscas formuladas por meio de expressão booleana que compõe-se por operadores lógicos como AND, OR e NOT. Tais operadores podem ser combinados para atingir buscas mais amplas ou restritas, o produto da busca booleana é um conjunto de documentos resultado da expressão sobre os termos buscados. Para que a expressão booleana sobre os termos ocorra, é necessário que um documento seja representado por um conjunto de termos de indexação.
No contexto deste trabalho será implementado para indexação dos termos de um conjunto de documentos, o índice invertido, que pode ser definido como uma estrutura de dados que armazena um mapeamento de conteúdo, como palavras em um documento, nesse cenário os termos de um documento e a localização de cada termo no conjunto de documentos.

Algoritmo

O algoritmo implementado possui 4 funções definidas e o método principal, será explicado detalhadamente cada uma delas.

1. def main()
2. def read_documents()
3. def prepare_doc(file)
4. def criar_indice_invertido(document, num)
5. def consulta_booleana(pesquisa)

Este fluxograma tem como objetivo simplificar o entendimento do algoritmo, iniciando pelo método main() no qual o usuário insere os termos que serão pesquisados nos documentos e depois visualiza o resultado da pesquisa, que corresponde aos índice dos documentos que possuem os termos pesquisados em comum.



Entendendo melhor cada umas das funções:

def main():

É o método principal, responsável por administrar o fluxo do código, seguindo uma sequência definida de passos:

1. Chama a função *read_documents()* responsável por ler os documentos do "diretório de documentos" representado no fluxograma, lê também os documentos de **stopwords** e **pontuações,** depois cria listas globais com o conteúdo de todos estes documentos, esta função será melhor explicada posteriormente;
2. Chama a função *prepare_doc(nome do arquivo),* passando o nome do arquivo, por fins didáticos optei por separar o conteúdo de cada documento em uma lista, mas em um contexto real essa não seria a melhor abordagem, esta função também será melhor explicada adiante.
3. Ordena o conteúdo de cada lista referente aos respectivos documentos utilizando a função *sort()*.
4. Chama a função *criar_indice_invertido(documento, num_documento),* neste momento é criado o índice invertido de cada um dos documentos e salvo os termos em um dicionário global, por isso esta função não tem retorno.
5. Lê os termos de pesquisa inseridos pelo usuário e ordena colocando tudo em minúsculo.
6. Chama a função *consulta_booleana(pesquisa)*, passando a lista com os termos a serem pesquisados.

```python
def main():
    # função que lê e preenche as variaveis globais que serão usadas por todo o código
    read_documents()

    # preparando os documentos retirando pontuações e stopwords
    document01 = prepare_doc('doc1_patinho_feio.txt')
    document02 = prepare_doc('doc2_joao_maria.txt')
    document03 = prepare_doc('doc3_pinoquio.txt')
    document04 = prepare_doc('doc4_branca_neve.txt')
    document05 = prepare_doc('doc5_cinderela.txt')

    # ordenando os documentos por ordem alfabetica
    document01.sort()
    document02.sort()
    document03.sort()
    document04.sort()
    document05.sort()

    # criando indice invertido de cada um dos documentos e salvando no dicionario global dict_terms
    criar_indice_invertido(document01, '1')
    criar_indice_invertido(document02, '2')
    criar_indice_invertido(document03, '3')
    criar_indice_invertido(document04, '4')
    criar_indice_invertido(document05, '5')

    pesquisa = str(input("Informe os termos da pesquisa:"))
    # convertendo para minusculo e separando os termos
    pesquisa = pesquisa.lower().split()

    consulta_booleana(pesquisa)
```

def read_documents( ):

Função responsável por ler os documentos no diretório especificado na variável *path* , lê também os arquivos de pontuação e stopwords inserindo o conteúdo em variáveis globais do tipo lista. Acredito que os comentários no código abaixo complementam a explicação.

Por fim, após a finalização dessa função o resultado será 3 listas globais contendo:

1. *“punctuation” :* lista que posteriormente será usada para retirar as pontuações das listas de documentos.
2. *“stopwords” :* lista que posteriormente será usada para retirar as stopwords das listas de documentos.
3. *“docs”* : lista com o conteúdo dos documentos no diretório definido na variável *path.*

```python
def read_documents():
    # definindo variaveis globais
    global punctuation
    global stopwords
    global dict_terms
    global docs

    path = "collection_docs"  # caminho do diretório que contem a coleção de documentos
    docs = {}  # criando dicionário vazio que irá armazenar o conjunto de documentos
    # criando dicionario vazio que irá armazenas os termos do conjunto de documentos
    dict_terms = {}

    # lendo o arquivo de pontuação
    with open('punctuation.txt', 'r') as f:
        punctuation = f.read()
    # separando todas as pontuações e inserindo em formato de lista em punctuation
    punctuation = punctuation.split()

    # lendo o arquivo de stopwords
    with open('stopwords_ptbr.txt', 'r', encoding="utf8") as f:
        stopwords = f.read()
    # separando todas as stopwords e inserindo em formato de lista em stopwords
    stopwords = stopwords.split("\n")

    # lendo todos os arquivos dentro do diretorio no caminho PATH
    for filename in os.listdir(path):
        # abrindo arquivo com encoding utf8 para receber acentos e caracteres especiais adequad
        with open(os.path.join(path, filename), 'r', encoding="utf8") as f:
            # salvando conteudo dos arquivos na lista de documentos
            docs[filename] = f.readlines()
```

def prepare_doc(file):

Função responsável por separar o conteúdo de cada documento em uma lista realizando operações no conteúdo:

1. Retirando quebra de linha “\n” e convertendo todos os caracteres para minúsculos.
2. Percorrendo o vetor de pontuação e retirando a pontuação da lista *“document”*.
3. Inserindo na lista final somente os termos do documento que não constam na lista de stopwords, ou seja retirando as stopwords dos termos do documento.
4. Retorna o resultado final da lista “*document”* com os termos do documento em minúsculo, sem pontuação e sem stopwords.

```
def prepare_doc(file):
    # passando o conteudo de cada arquivo para uma lista de string
    document = str(docs.get(file))
    # convertendo tudo para minusculo e retirando o \n
    document = document.lower().replace("\\n", "")

    for p in punctuation:  # percorrendo o vetor de pontuação
        document = document.replace(p, "")  # retirando a pontuação da string

    # separando no espaço todas as palavras e inserindo em uma lista
    document = document.split()
    # inserindo na lista somente as palavras que NAO constam na lista de stopwords
    document = [d for d in document if not d in stopwords]

    return document
```

def criar_indice_invertido(document, num):

Função responsável por implementar o mapeamento dos termos do documento preenchendo um dicionário global de termos, recebe o nome do documento e o número do mesmo para ser usado como índice.

1. Percorre o documento recebido *"document"* comparando se cada termo deste documento existe no dicionário de termos *"dict_terms",* se não existir então insere o termo no *dicionário {key : value}* sendo:
    a. key = termo;
    b. value = índice referente ao número do documento;
2. Se o termo já existir no dicionário de termos *"dict_terms",* e o índice referente ao número do documento não existir no value do termo, então é feito um update no valor do value do termo adicionando o índice do documento atual.

```
def criar_indice_invertido(document, num):  # criando indice invertido
    for d in document:  # percorrendo o documento
        if d not in dict_terms:
            dict_terms[d] = num
        # se o termo não esta no dict e não é um termo repetido
        if d in dict_terms and num not in dict_terms[d]:
            x = str(dict_terms[d])+" "
            # atualiza a lista de documentos do termo
            dict_terms.update({d: x+num})
```

def consulta_booleana(pesquisa):

Função responsável por implementar a pesquisa booleana, recebe os termos a serem pesquisados.

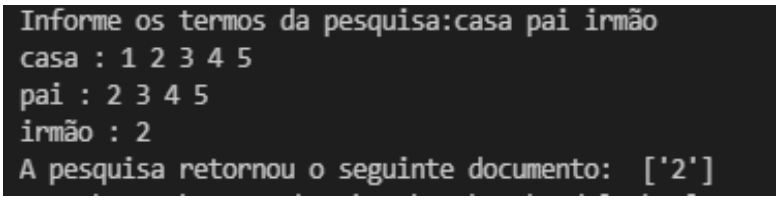
1. Percorre a lista com os termos da pesquisa comparando se o termo existe no dicionário de termos, se o termo existir, insere o *"value"* do dicionário, ou seja, o valor correspondente ao índice do documento que contém o termo pesquisado na lista *"docs_pesquisa"*.
2. Verifica se a lista *"docs_pesquisa"* não possui conteúdo, caso a verificação for positiva, então o programa se encerra informando o usuário que "nenhum termo de pesquisa válido foi encontrado".
3. Convertendo a primeira posição da lista para o tipo set, depois percorre o tamanho da lista *"docs_pesquisa"* transformando as posições para o tipo set e realizando a intersecção dos termos, que nesse ponto é o índice dos documentos, a fim de obter apenas os documentos que satisfazem todos os termos da chave.
4. Por último, compara-se o tamanho da lista de resultado *"result_pesquisa",* para exibir uma mensagem ao usuário com o índice referente aos documentos em comum que satisfazem todos os termos de pesquisa do operador lógico AND.

```
def consulta_booleana(pesquisa):
    docs_pesquisa = []
    for p in pesquisa:  # percorrendo os termos de pesquisa fornecidos pelo usuário
        if dict_terms.get(p) is not None:  # se o termo existir
            print(p + " : " + dict_terms.get(p))
            aux = dict_terms.get(p)
            aux = aux.split()
            # inserindo o numero dos documentos de cada termo na lista
            docs_pesquisa.append(aux)

    if len(docs_pesquisa) == 0:  # caso nenhum termo for encontrado nos termos mapeados, encerra
        print("Nenhum termo de pesquisa valido foi encontrado")
        exit()

    # convertendo a primeira posição para set
    result_pesquisa = set(docs_pesquisa[0])
    for dc in range(len(docs_pesquisa)):  # percorrendo o tamanho da lista
        # convertendo a lista em set
        docs_pesquisa[dc] = set(docs_pesquisa[dc])
        result_pesquisa = result_pesquisa.intersection(
            docs_pesquisa[dc])  # função de intersecção

    result_pesquisa = sorted(result_pesquisa)
    if(len(result_pesquisa) > 1):
        print("A pesquisa retornou os seguintes documentos: ",
              result_pesquisa)
    elif(len(result_pesquisa) == 1):
        print("A pesquisa retornou o seguinte documento: ",
              result_pesquisa)
    else:
        print("A pesquisa não retornou nenhum documento")
```

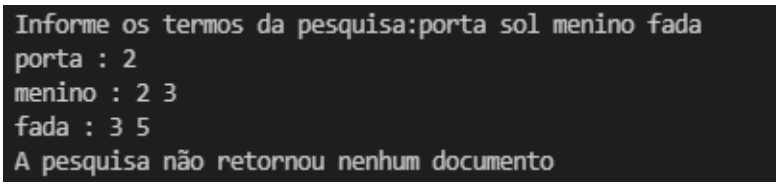
Conclusões e Resultados obtidos

Os documentos escolhidos para a realização deste trabalho possuem histórias infantis e possibilitaram resultados de pesquisa interessantes, como os exemplos abaixo:

Nesta consulta a pesquisa retornou um documento em comum com os três termos, então pode-se dizer que o objetivo inicial deste trabalho foi alcançado.



Nesta consulta, a pesquisa não retornou nenhum documento em comum para os três termos, mas isso chama atenção pois é evidente que os termos se encontram em alguns documentos e nenhum deles foi apresentado ao usuário, logo existe espaço para melhorias no algoritmo, por exemplo para o algoritmo implementar outros operadores lógicos como OR e NOT na pesquisa booleana, pois neste trabalho foi implementado somente o operador AND.



Em termos gerais, o objetivo traçado foi concluído implementando com sucesso o modelo de busca booleana com índice invertido, todavia entende-se que existem possibilidades de melhoria no algoritmo de busca.