Busca_de_indice_e_Consulta

June 11, 2024

1 Busca de Índice e Consulta de documento

Autor: Davi J. Leite Santos
Versão: 0.0.3
Data: 25 de Abril de 2024
Localização: Ribeirão das Neves, Minas Gerais - Brasil

1.1 Contato

- **Endereço:** Ribeirão das Neves, Minas Gerais - Brasil
- **Email:** davi.jls@outlook.com
- **LinkedIn:** davi-j-leite-santos
- **Website:** davijls.com.br

1.2 Principais Competências

- **Cibersegurança**
- **Segurança da Informação**
- **Operações de TI**

2 Sobre o codigo

2.1 Cabeçalho e Metadados

O cabeçalho inicial identifica o autor, a versão, a data e o local de criação do código, além de detalhes de contato e competências principais, que incluem Cibersegurança, Segurança da Informação e Operações de TI.

2.2 Dicionários de Vocabulário

Dois dicionários, VOCAB e VOCAB2, são definidos para associar palavras-chave a índices numéricos. Esses vocabulários são essenciais para mapear termos a seus respectivos índices em operações de indexação e busca.

2.3 Estruturas de Dados Iniciais

O código define array como um dicionário que organiza índices de documentos e as posições de palavras dentro deles. Este dicionário é então transformado em uma lista de tuplas chamada ii2, que facilita o manuseio dos dados.

2.4 Funções de Reconstrução de Documentos

Várias funções (reconstruir_documento, reconstruir_documento2, reconstruir_documento3) são implementadas para converter os índices armazenados de volta em formato legível, demonstrando como as palavras estão organizadas nos documentos.

2.5 Indexação de Novos Documentos

O código segue com uma simulação de indexação de novos documentos através de um processo que envolve a divisão de textos em palavras, limpeza de pontuações e conversão para minúsculas, além de atualizar o vocabulário e índices invertidos com novas palavras.

2.6 Consulta Usando Árvore Sintática

Uma funcionalidade de consulta utilizando uma árvore de operadores lógicos (AND, OR) é desenvolvida, permitindo a realização de buscas complexas por meio de combinações de termos.

2.7 Compressão de Texto

A parte final do código aborda duas técnicas de compressão de texto: a compressão usando o método de Huffman, que cria uma árvore de frequência das palavras para gerar códigos de compressão eficientes, e a compressão baseada em dicionário, que substitui palavras por tokens definidos manualmente para reduzir o tamanho do texto.

Cada parte do código é minuciosamente explicada por meio de comentários e divisões lógicas, facilitando a compreensão das operações realizadas e de como cada componente contribui para o processamento de texto e recuperação de informações. O uso de tabelas e formatos de impressão amigáveis (como prettytable) melhora a visualização dos resultados, tornando as saídas mais legíveis para análise ou demonstração.

3 ————————————————————————————————————-

4 Processo de carregar os arquivos e guarda-los

Essa parte server para acessar cada documento e fazer o indice invertido de cada um, justamente para armazena-los de alguma forma.

[ ]:
```
VOCAB = {
    "boa": (0),
    "noite": (1),
    "pessoal": (2),
    "ja": (3),
    "comecem": (4),
    "0": (5),
    "projeto": (6),
    "alguma": (7),
    "duvida": (8),
    "isso": (9),
    "e": (10),
```

```
    "tudo": (11),
}


VOCAB2 = {
    "cada": 0,
    "qual": 1,
    "sabe": 2,
    "amar": 3,
    "a": 4,
    "seu": 5,
    "modo": 6,
    "o": 7,
    "pouco": 8,
    "importa": 9,
    "essencial": 10,
    "e": 11,
    "que": 12,
    "saiba": 13,
    "boa": 14,
    "prova": 15,
    "todos": 16,
    "nao": 17,
    "quero": 18,
    "ter": 19,
    "terrivel": 20,
    "limitacao": 21,
    "de": 22,
    "quem": 23,
    "vive": 24,
    "apenas": 25,
    "do": 26,
    "passivel": 27,
    "fazer": 28,
    "sentido": 29,
}
```

[ ]:
```
# Dicionário original
array = {
    0: {0: [0]},
    1: {0: [1]},
    2: {0: [2]},
    3: {0: [3, 16]},
    4: {0: [4], 1: [2], 2: [3]},
    5: {0: [5]},
    6: {0: [6, 8]},
    7: {0: [7, 11]},
```

```
    8: {0: [9]},
    9: {0: [10]},
    10: {0: [12]},
    11: {0: [13], 2: [12]},
    12: {0: [14], 2: [11]},
    13: {0: [15]},
    14: {1: [0]},
    15: {1: [1]},
    16: {1: [3]},
    17: {2: [0]},
    18: {2: [1]},
    19: {2: [2]},
    20: {2: [4]},
    21: {2: [5]},
    22: {2: [6, 14]},
    23: {2: [7]},
    24: {2: [8]},
    25: {2: [9]},
    26: {2: [10]},
    27: {2: [13]},
    28: {2: [15]},
    29: {2: [16]},
}

# Transformando o dicionário em uma lista de tuplas
ii2 = [(k1, k2, v2) for k1, v1 in array.items() for k2, v2 in v1.items()]

# Imprimindo a lista de tuplas
for item in ii2:
    print(item)
```

```
(0, 0, [0])
(1, 0, [1])
(2, 0, [2])
(3, 0, [3, 16])
(4, 0, [4])
(4, 1, [2])
(4, 2, [3])
(5, 0, [5])
(6, 0, [6, 8])
(7, 0, [7, 11])
(8, 0, [9])
(9, 0, [10])
(10, 0, [12])
(11, 0, [13])
(11, 2, [12])
(12, 0, [14])
(12, 2, [11])
```

```
(13, 0, [15])
(14, 1, [0])
(15, 1, [1])
(16, 1, [3])
(17, 2, [0])
(18, 2, [1])
(19, 2, [2])
(20, 2, [4])
(21, 2, [5])
(22, 2, [6, 14])
(23, 2, [7])
(24, 2, [8])
(25, 2, [9])
(26, 2, [10])
(27, 2, [13])
(28, 2, [15])
(29, 2, [16])
```

[ ]:
```python
ii = [
    (0, 0, [1]),
    (1, 0, [1]),
    (2, 0, [2]),
    (2, 1, [2]),
    (2, 2, [2]),
    (2, 3, [3]),
    (3, 1, [1]),
    (4, 1, [1]),
    (5, 1, [2]),
    (6, 1, [0]),
    (7, 2, [0]),
    (8, 2, [1]),
    (9, 3, [1]),
    (10, 3, [1]),
    (11, 3, [2]),
]
```

[ ]:
```python
# Apresentando de forma mais bonita:
from prettytable import PrettyTable


def reconstruir_documento2(ii, vocab):
    # Inicializando o documento como uma lista vazia
    documento = []

    # Iterando sobre cada entrada em 'ii'
    for entry in ii:
```

```
        # Desempacotando a entrada em índice de termo, índice de documento e posições
        term_index, doc_index, positions = entry

        # Encontrando o termo correspondente ao índice de termo no vocabulário
        term = list(vocab.keys())[list(vocab.values()).index(term_index)]

        # Adicionando a tupla (termo, índice de documento, posições) ao documento
        documento.append((term, doc_index, positions))

    # Criando uma tabela bonita
    table = PrettyTable()

    # Definindo os cabeçalhos da tabela
    table.field_names = ["Termo", "Índice do Documento", "Posições"]

    # Adicionando cada entrada do documento à tabela
    for term, doc_index, positions in documento:
        table.add_row([term, doc_index, positions])

    # Retornando a tabela como uma string
    return str(table)


def reconstruir_documento3(ii, vocab):
    # Inicializando o documento como um dicionário padrão
    documento = defaultdict(list)

    # Iterando sobre cada entrada em 'ii'
    for entry in ii:
        # Desempacotando a entrada em índice de termo, índice de documento e posições
        term_index, doc_index, positions = entry

        # Encontrando o termo correspondente ao índice de termo no vocabulário
        term = list(vocab.keys())[list(vocab.values()).index(term_index)]

        # Adicionando a tupla (termo, posições) ao documento correspondente
        documento[doc_index].append((term, positions))

    # Imprimindo cada entrada do documento
    for doc_index, entries in documento.items():
        print(f"Documento {doc_index}:")
        for term, positions in entries:
            print(f"  Termo: {term}, Posições: {positions}")
```

[ ]:
```
def reconstruir_documento(ii, vocab):
    documento = []
    for entry in ii:
        term_index, doc_index, positions = entry
        term = list(vocab.keys())[list(vocab.values()).index(term_index)]
        documento.append((term, doc_index, positions))
    return documento
```

[ ]:
```
documento_reconstruido = reconstruir_documento(ii, VOCAB)
print(documento_reconstruido)
```

```
[('boa', 0, [1]), ('noite', 0, [1]), ('pessoal', 0, [2]), ('pessoal', 1, [2]),
('pessoal', 2, [2]), ('pessoal', 3, [3]), ('ja', 1, [1]), ('comecem', 1, [1]),
('0', 1, [2]), ('projeto', 1, [0]), ('alguma', 2, [0]), ('duvida', 2, [1]),
('isso', 3, [1]), ('e', 3, [1]), ('tudo', 3, [2])]
```

[ ]:
```
documento_reconstruido = reconstruir_documento(ii2, VOCAB2)
print(documento_reconstruido)
```

```
[('cada', 0, [0]), ('qual', 0, [1]), ('sabe', 0, [2]), ('amar', 0, [3, 16]),
('a', 0, [4]), ('a', 1, [2]), ('a', 2, [3]), ('seu', 0, [5]), ('modo', 0, [6,
8]), ('o', 0, [7, 11]), ('pouco', 0, [9]), ('importa', 0, [10]), ('essencial',
0, [12]), ('e', 0, [13]), ('e', 2, [12]), ('que', 0, [14]), ('que', 2, [11]),
('saiba', 0, [15]), ('boa', 1, [0]), ('prova', 1, [1]), ('todos', 1, [3]),
('nao', 2, [0]), ('quero', 2, [1]), ('ter', 2, [2]), ('terrivel', 2, [4]),
('limitacao', 2, [5]), ('de', 2, [6, 14]), ('quem', 2, [7]), ('vive', 2, [8]),
('apenas', 2, [9]), ('do', 2, [10]), ('passivel', 2, [13]), ('fazer', 2, [15]),
('sentido', 2, [16])]
```

[ ]:
```
documento_reconstruido = reconstruir_documento2(ii2, VOCAB2)
print(documento_reconstruido)
```

```
+-----------+---------------------+----------+
|   Termo   | Índice do Documento | Posições |
+-----------+---------------------+----------+
|    cada   |          0          |   [0]    |
|    qual   |          0          |   [1]    |
|    sabe   |          0          |   [2]    |
|    amar   |          0          | [3, 16]  |
|     a     |          0          |   [4]    |
|     a     |          1          |   [2]    |
|     a     |          2          |   [3]    |
|    seu    |          0          |   [5]    |
|    modo   |          0          |  [6, 8]  |
|     o     |          0          | [7, 11]  |
|   pouco   |          0          |   [9]    |
|  importa  |          0          |   [10]   |
| essencial |          0          |   [12]   |
|     e     |          0          |   [13]   |
```

```
|     e     |          2          |   [12]   |
|    que    |          0          |   [14]   |
|    que    |          2          |   [11]   |
|   saiba   |          0          |   [15]   |
|    boa    |          1          |   [0]    |
|   prova   |          1          |   [1]    |
|   todos   |          1          |   [3]    |
|    nao    |          2          |   [0]    |
|   quero   |          2          |   [1]    |
|    ter    |          2          |   [2]    |
|  terrivel |          2          |   [4]    |
| limitacao |          2          |   [5]    |
|     de    |          2          | [6, 14]  |
|    quem   |          2          |   [7]    |
|    vive   |          2          |   [8]    |
|   apenas  |          2          |   [9]    |
|     do    |          2          |   [10]   |
|  passivel |          2          |   [13]   |
|   fazer   |          2          |   [15]   |
|  sentido  |          2          |   [16]   |
+-----------+---------------------+----------+
```

Irei indexar os documentos fornecidos e atualizar o vocabulário existente.

[ ]:
```python
# Documentos a serem indexados
documentos = ["Ser ou não ser, eis a questão.", "Até tu, Brutus, filho meu?"]

# Inicializando o índice invertido como um dicionário padrão
II2 = defaultdict(list)

# Iterando sobre cada documento
for doc_index, doc in enumerate(documentos):
    # Dividindo o documento em palavras
    palavras = doc.split()

    # Iterando sobre cada palavra no documento
    for pos, palavra in enumerate(palavras):
        # Removendo a pontuação e convertendo para minúsculas
        palavra = palavra.strip(",.!?").lower()

        # Se a palavra não está no vocabulário, adicione-a
        if palavra not in VOCAB2:
            VOCAB2[palavra] = len(VOCAB2)

        # Adicionando a posição da palavra ao índice invertido
        II2[VOCAB2[palavra]].append((doc_index, pos))

# Imprimindo VOCAB2 e II2
```

```
print("VOCAB2 = ", VOCAB2)
print("II2 = ", dict(II2))
```

```
VOCAB2 =  {'cada': 0, 'qual': 1, 'sabe': 2, 'amar': 3, 'a': 4, 'seu': 5, 'modo':
6, 'o': 7, 'pouco': 8, 'importa': 9, 'essencial': 10, 'e': 11, 'que': 12,
'saiba': 13, 'boa': 14, 'prova': 15, 'todos': 16, 'nao': 17, 'quero': 18, 'ter':
19, 'terrivel': 20, 'limitacao': 21, 'de': 22, 'quem': 23, 'vive': 24, 'apenas':
25, 'do': 26, 'passivel': 27, 'fazer': 28, 'sentido': 29, 'ser': 30, 'ou': 31,
'não': 32, 'eis': 33, 'questão': 34, 'até': 35, 'tu': 36, 'brutus': 37, 'filho':
38, 'meu': 39}
II2 =  {30: [(0, 0), (0, 3)], 31: [(0, 1)], 32: [(0, 2)], 33: [(0, 4)], 4: [(0,
5)], 34: [(0, 6)], 35: [(1, 0)], 36: [(1, 1)], 37: [(1, 2)], 38: [(1, 3)], 39:
[(1, 4)]}
```

Irei criar uma representação da árvore de sintaxe e atribuir índices a cada nó para a consulta fornecida. Aqui está o código para construir essa árvore e os índices, onde o objetivo é construir a árvore de sintaxe e os índices para a consulta pessoal AND (boa OR tudo).

[ ]:
```
class Nodo:
    def __init__(self, valor=None, esquerda=None, direita=None):
        self.valor = valor
        self.esquerda = esquerda
        self.direita = direita
```

[ ]:
```
def construir_arvore_consulta():
    # Criar os nodos para os termos da consulta
    nodo_pessoal = Nodo(valor="pessoal")
    nodo_boa = Nodo(valor="boa")
    nodo_tudo = Nodo(valor="tudo")

    # Nodos para os operadores lógicos
    nodo_or = Nodo(valor="OR", esquerda=nodo_boa, direita=nodo_tudo)
    nodo_and = Nodo(valor="AND", esquerda=nodo_pessoal, direita=nodo_or)

    return nodo_and
```

[ ]:
```
def atribuir_indices_arvore(nodo, vocab):
    indices = {}

    # Percorrer a árvore em pré-ordem para atribuir índices aos termos
    def percorrer_arvore(nodo):
        nonlocal indices
        if nodo is not None:
            if nodo.valor in vocab:
                indices[nodo.valor] = vocab[nodo.valor]
            percorrer_arvore(nodo.esquerda)
            percorrer_arvore(nodo.direita)
```

```
    percorrer_arvore(nodo)
    return indices
```

[ ]:
```
# Construir a árvore para a consulta pessoal AND (boa OR tudo)
arvore_consulta = construir_arvore_consulta()

# Atribuir índices aos termos na árvore
indices_arvore = atribuir_indices_arvore(arvore_consulta, VOCAB)
```

[ ]:
```
print("\nÍndices atribuídos aos termos na árvore:")
print(indices_arvore)
```

```
Índices atribuídos aos termos na árvore:
{'pessoal': 2, 'boa': 0, 'tudo': 11}
```

4.1 Indexar o documento “Boatarde galera”

Para indexar o documento “Boatarde galera”:

[ ]:
```
def indexar_documento(documento, vocabulario):
    palavras = documento.lower().split()
    indice = []

    for palavra in palavras:
        if palavra in vocabulario:
            indice.append(vocabulario[palavra])

    return indice
```

[ ]:
```
# Documento a ser indexado
documento = "Boa tarde galera"

# Indexar o documento usando o vocabulário
indice_1 = indexar_documento(documento, VOCAB)

print("Índice do documento 'Boatarde galera':", indice_1)
```

```
Índice do documento 'Boatarde galera': [0]
```

4.2 Recuperação de Documentos Relevantes

Recuperar e Reconstruir documento(s) relevantes para a consulta “Boa AND noite”

[ ]:
```
def recuperar_documentos_relevantes(consulta, ii):
    relevantes = []
    for term_index in consulta:
        for entry in ii:
            if entry[0] == term_index:
```

```
                relevantes.append(entry)
    return relevantes
```

[ ]:
```
consulta = [(7), (8)]  # Boa AND noite
documentos_relevantes = recuperar_documentos_relevantes(consulta, ii)
documentos_relevantes_reconstruidos = reconstruir_documento(
    documentos_relevantes, VOCAB
)
print(documentos_relevantes_reconstruidos)
```

```
[('alguma', 2, [0]), ('duvida', 2, [1])]
```

4.3 Compressão Estática com Código de Huffman

Para realizar a compressão estática com o código de Huffman:

[ ]:
```
class NodoHuffman:
    def __init__(self, palavra=None, frequencia=0):
        self.palavra = palavra
        self.frequencia = frequencia
        self.esquerda = None
        self.direita = None
```

[ ]:
```
def calcular_frequencias(texto):
    frequencias = {}
    palavras = texto.split()

    for palavra in palavras:
        if palavra in frequencias:
            frequencias[palavra] += 1
        else:
            frequencias[palavra] = 1

    return frequencias
```

[ ]:
```
def construir_arvore_huffman(frequencias):
    fila = [
        NodoHuffman(palavra=palavra, frequencia=freq)
        for palavra, freq in frequencias.items()
    ]

    while len(fila) > 1:
        fila = sorted(fila, key=lambda x: x.frequencia)

        esquerda = fila.pop(0)
        direita = fila.pop(0)

        pai = NodoHuffman(frequencia=esquerda.frequencia + direita.frequencia)
```

```
        pai.esquerda = esquerda
        pai.direita = direita

        fila.append(pai)

    return fila[0]
```

[ ]:
```
def construir_tabela_codigos(nodo, codigo="", tabela={}):
    if nodo is not None:
        if nodo.palavra is not None:
            tabela[nodo.palavra] = codigo
        construir_tabela_codigos(nodo.esquerda, codigo + "0", tabela)
        construir_tabela_codigos(nodo.direita, codigo + "1", tabela)
```

[ ]:
```
def codificar_texto(texto, tabela_codigos):
    palavras = texto.split()
    texto_codificado = ""

    for palavra in palavras:
        if palavra in tabela_codigos:
            texto_codificado += (
                tabela_codigos[palavra] + " "
            )  # Adiciona o código da palavra e um espaço
        else:
            texto_codificado += (
                palavra + " "
            )  # Adiciona a palavra diretamente com um espaço

    return texto_codificado.strip()  # Remove o espaço extra no final
```

[ ]:
```
def decodificar_texto(texto_codificado, tabela_codigos):
    texto_decodificado = ""
    codigos = texto_codificado.split()

    for codigo in codigos:
        palavra_decodificada = next(
            (
                palavra
                for palavra, codigo_tabela in tabela_codigos.items()
                if codigo_tabela == codigo
            ),
            None,
        )
        if palavra_decodificada is not None:
            texto_decodificado += (
                palavra_decodificada + " "
            )  # Adiciona a palavra decodificada ao texto
```

```
    return texto_decodificado.strip()  # Remove o espaço extra no final, se houver
```

[ ]:
```
# Função principal para compressão de texto usando Huffman com palavras
def compressao_huffman(texto):
    frequencias = calcular_frequencias(texto)
    arvore_huffman = construir_arvore_huffman(frequencias)

    tabela_codigos = {}
    construir_tabela_codigos(arvore_huffman, "", tabela_codigos)

    texto_codificado = codificar_texto(texto, tabela_codigos)

    return texto_codificado, tabela_codigos
```

4.4 Método baseado em Dicionário

[ ]:
```
# Exemplo de uso da compressão de Huffman com palavras
texto_original = "O tempo respondeu pro tempo que não tem tempo pro tempo"

# Realiza a compressão usando Huffman com palavras
texto_codificado, tabela_codigos = compressao_huffman(texto_original)
```

[ ]:
```
print("Tabela de códigos Huffman:")
for palavra, codigo in tabela_codigos.items():
    print(f"{palavra}: {codigo}")
print("Texto comprimido (em binário):", texto_codificado)

# Decodifica o texto comprimido usando Huffman com palavras
texto_decodificado = decodificar_texto(texto_codificado, tabela_codigos)
print("Texto decodificado:", texto_decodificado)
print("Texto original    :", texto_original)
```

```
Tabela de códigos Huffman:
O: 000
respondeu: 001
que: 010
não: 011
tem: 100
pro: 101
tempo: 11
Texto comprimido (em binário): 000 11 001 101 11 010 011 100 11 101 11
Texto decodificado: O tempo respondeu pro tempo que não tem tempo pro tempo
Texto original    : O tempo respondeu pro tempo que não tem tempo pro tempo
```

Nesta implementação:

- A função calcular_frequencias calcula as frequências de cada caractere no texto.

- A função construir_arvore_huffman constrói a árvore de Huffman com base nas frequências calculadas.
- A função codificar_texto utiliza uma tabela de códigos Huffman para codificar o texto original.
- A função decodificar_texto decodifica o texto comprimido usando a árvore de Huffman.

4.5 Compressão do Texto: "Esperando a prova, sigo estudando para a prova"

Para aplicar a compressão baseada em dicionário:

[ ]:
```python
def compressao_baseada_em_dicionario(texto, dicionario):
    palavras = texto.lower().split()
    texto_comprimido = []

    for palavra in palavras:
        if palavra in dicionario:
            texto_comprimido.append(dicionario[palavra])
        else:
            texto_comprimido.append(
                palavra
            )  # Mantém a palavra se não estiver no dicionário

    texto_comprimido = " ".join(texto_comprimido)
    return texto_comprimido
```

[ ]:
```python
# Dicionário de substituição para compressão
dicionario_compressao = {
    "esperando": "Esp",
    "a": "a",
    "prova": "P",
    "sigo": "S",
    "estudando": "E",
    "para": "p",
    "a": "a",
}
```

[ ]:
```python
# Exemplo de texto para compressão
texto_original = "Esperando a prova, sigo estudando para a prova"

# Realiza a compressão baseada em dicionário
texto_comprimido = compressao_baseada_em_dicionario(
    texto_original, dicionario_compressao
)
```

[ ]:
```python
print("Texto original:", texto_original)
print("Texto comprimido:", texto_comprimido)
```

```
Texto original: Esperando a prova, sigo estudando para a prova
Texto comprimido: Esp a prova, S E p a P
```

[ ]:

[ ]:

[ ]: