

Busca_de_indice_e_Consulta

May 2, 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

1.2.1 Reconstruir o documento

Para reconstruir o documento usando o índice invertido.

```
[ ]: 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)  
}
```

```
[ ]: 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]),
]
```

```
[ ]: 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])]
```

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}

1.3 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]

1.4 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])]

1.5 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

```

```

[ ]: # Exemplo de uso da compressão de Huffman com palavras
texto_original = "for my rose, a rose is a rose, e moises jorge legal"

# 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:

e: 000

moises: 001

jorge: 010

legal: 011

rose,: 100

a: 101

for: 1100

my: 1101

rose: 1110

is: 1111

Texto comprimido (em binário): 1100 1101 100 101 1110 1111 101 100 000 001 010 011

Texto decodificado: for my rose, a rose is a rose, e moises jorge legal

Texto original : for my rose, a rose is a rose, e moises jorge legal

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.

1.6 Compressão do Texto: “Esperando a prova, sigo estudando para a prova”

Para aplicar a compressão baseada em dicionário:

```
[ ]: 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
```

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

```
[ ]: # 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)
```

```
[ ]: 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

[]: