Documentação para execução e uso do arquivo da prova

Desenvolvedor: Edson Coelho dos Santos
Data: 01/12/2019
Sistema: Contagem de incidência de palavras em documentos

Introdução

Este documento serve como manual de execução e documentação explicativa de como foram implementadas as soluções para o problema proposto no arquivo fornecido para o exame admissional para a empresa Cinnecta. A seguir serão apresentadas as estruturas utilizadas, assim como metodologia empregada e no final um breve resumo de por que foram empregadas as soluções e ideias de como aplicar melhorias para ganhos tanto com tempo de execução quanto com estrutura.

Implantação do Software(Deploy)

Todos os arquivos de execução estão disponíveis no repositório do github com privacidade pública e podem ser baixado com o comando:

```
#git clone https://github.com/Edson-Fullstack/cinnecta.git
```

Após clonar o repositório é necessário instalar uma lista de dependencias listadas no arquivo “install.sh”, este foi construido em formato de arquivo bash e pode ser executado em sistemas linux(lembrando que este script foi feita para executar em sistemas debian like e foi testado em no debian 9 e no ubuntu 18.04).

```
#./install.sh
```

Arquitetura

O sistema foi desenvolvido para rodar em plataformas independentes, onde a aplicação roda em um servidor, os arquivos podem rodar em uma rede de CDN(Content Delivery Networking) e a base de dados pode ou não rodar em outro servidor independente.

Colocar os serviços em em servidores independentes seria o ideal para melhor aproveitamento de desacoplamento e obter o melhor desempenho dos servidores em seus respectivos serviços.

Atualmente por restrições de tempo e valores, as ferramentas utilizadas foram o google drive(google) para armazenar arquivos de texto e o módulo Atlas fornecido pela mongodb(mongodb) no tier gratuito para hospedagem de base de dados NoSQL.

Solução

Formatação e Código

O código foi formatado utilizado o padrão PEP8 e a organização de pastas aconselhada pelo guia do framework Flask. De forma geral foram utilizados apenas as estruturas fornecidas pela linguagem Python e HTML.

Frameworks

Foi utilizado o frameworks Flask para implementação da API REST em Python, com auxílio de algumas bibliotecas como pymongo para acesso a base de dados NoSQL. As outras importações são referentes a tratamento de strings e conversoẽs para adaptar o formato ao modelo BsonDocument adotado pelo NoSQL.

Para uma apresentação mais amigável foi utilizado o framework front end bootstrap e a importação de seus arquivos se encontra no formato bootstrap CDN nos arquivos “header.html” e “footer.html” localizados dentro da do endereço /templates/includes a partir do ponto de montagem da raiz.

Controle

Durante o desenvolvimento para procurar erros e acompanhar o desenvolvimento da solução a função “tests(controle,mensagem)” foi utilizada e esta utiliza de uma variável de controle chamada TEST_CONTROL. Esta variável é responsável por imprimir o passo a passo da solução e controlar a base de dados nos ambientes de produção.

A retirada das chamadas desta função poderia reduzir potencialmente a o número de linhas, porém para rastrear erros tanto no ambiente de teste quanto no de desenvolvimento, quanto controlar o uso ou não da base de dados tornam esta essencial e não aconselho a retirada.

Resumo: em posição 0 desativa todos os prints de saídas no servidor e ativa o banco de dados. Em posição 1 desativa a base de dados e ativa o mínimo de saídas a cada passo. Em posição 2 ativa os prints no servidor para acompanhar o passo a passo da execução e desativa o acesso a base de dados.

Testes

O arquivo “tests.py” que acompompanha o programa executa uma bateria de testes listado, sendo testadas as respostas das portas em todas as urls disponibilizadas para o acesso além de conferir algumas respostas básicas fornecidas pelo programa.

Para executar a bateria de testes, ao se localizar no arquivo raiz do programa execute o seguinte comando:

```
#python3 tests.py
```

Observação: Não foram executados testes na base de dados ou nos arquivos disponibilizados pois os serviços utilizados provavelmente irão mudar.
Observação: O saída dos testes difere a partir do valor utilizado na variável de controle TEST_CONTROL, porém o resultado obtido após a execução é sempre o mesmo e pode ser acertado para testes para aproximar a resposta do valor ideal.

Execução

Para se executar o programa aconselha-se executar com a chamada python3 e idealmente ter a versão 3.7 do python, deve-se aplicar o comando a seguir na pasta raiz do documento, levando em consideração que se caso o programa for executado na porta 80 não pode ter outro programa sendo executado nesta mesma porta. O modo ideal seria configurar um servidor Web como Apache ou nginx para lidar com o gerenciamento de chaves e certificados SSL e para entregar páginas estáticas e redirecionar os outros servicos para o python.

```
#python3 app.py
```

Rotas

Foram apresentadas as seguintes rotas de navegação no site, as quais serão listadas abaixo e explicado o uso e aplicação de cada uma.

```
@myapp.route("/")
def home():
@myapp.route("/prova")
def prova():
@myapp.route("/manual")
def manual():
@myapp.route("/db")
def db():
@myapp.route("/gram1", defaults)
@myapp.route("/gram1<enter>")
```

```
def gramatica1(enter, methods=['GET', 'POST']):
@myapp.route("/gram2", defaults)
@myapp.route("/gram2<entrada>")
def gramatica2(entrada):
```

Home(/): define a página principal a qual será apresentada ao se entrar no site. Esta apresenta um formulário e dois botões que executam o processamento solicitado das palavras no campo de “text area” e redirecionando para a devida subrota(gram1(gramatica1) ou gram2(gramatica2)).Após o processamento o template “lista.html” será renderizado para mostrar a solução.
Observação: para facilitar teste a entrada vazia possui um valor default contendo os valores apresentados na prova.
Prova(prova): a rota prova foi utilizada para renderizar o documento da prova e facilitar o acesso a informação durante a produção.
Manual(manual): a rota manual é utilizada para renderizar este documento, o qual demonstra o uso da aplicação e a arquitetura/lógica utilizadas para o seu desenvolvimento.
db(db):gera um output básico contendo todos os objetos inserido na base de dados até o momento. Como futuro trabalho seria ideal organizar os dados para serem melhor visualizados e utilizar o campo de pesquisa para filtrar tais dados.
gram1(gramatica1) e gram2(gramatica2):as rotas gram1 e gram2 executam o processamento lógico e independente do tipo de execução solicitado. Para facilitar testes o acesso a rota com a string “enter” vazia gera uma entrada default com os valores apresentados na prova.

Desenvolvimento

Abaixo serão listados os métodos utilizados e explicadas as suas aplicações na solução:

```
def tests(controle, mensagem):
def is_int(valor):
def remover_caracteres(old_string, to_remove):
def substituir_caracteres(old_string, to_remove, to_replace):
def contar_incidencia(vetor, vocabulario):
def contar_incidencia2(vetores, vocabulario):
def contar(vetores, vocabulario, gramatica):
def gerar_listas_simples(textos, stop_words, gramatica):
def processar_texto(textos, stop_words, gramatica):
def formatar_saida(vocabulario,vector,dicionario,gramatica):
def conectar_bd():
def inserir_bd(conteudo):
def show_db():
```

Principais:
processar_texto(textos,stop_words,gramatica): função principal que executa de forma simplificada a ordem [gerar lista simples,contar incidências,formatar]. aqui também é feito o acesso ao banco de dados para inserção das execuções feitas.
gerar_lista_simples(textos,stop_words,gramatica): passa por todo o texto removendo os caracteres indevidos e formando o vetor de vocabulário e o vetor de palavras que define as palavras que ocorrem no vetor. Esta função acumula os 2 procedimentos e é alterada a partir da flag de controle “gramatica” que modifica o cálculo dos vetores.
Observação: com vetores muito pequenos 1 ou mesmo nulos de entrada, esta função apresenta erro na saída por não entrar nos referidos loops para formação dos vetores ou não execução por limites adicionados para não serem acessados itens foram da contagem do vetor.
contar(vetores,vocabulario,gramatica):utiliza a flag “gramatica” passada ao acessar o url para controlar o tipo de gramatica que sera aplicada.
contar_incidencia(vetor,vocabulario) e contar_incidencia2(vetores,vocabulario): conta o número de incidências do vocabulário no vetor, porém como as contagens se diferem, são separadas em vetores distintos.
formatar_saida(vocabulário,vector,dicionário,gramatica): formata a saída para ser denotada como solicitada no documento de prova.

Suportes
tests():controla as mensagens que serão impressas,dado um valor de controle.
is_int(): checa se um valor passado é inteiro
substituir_caracteres(): remove a sequência de caracteres fornecida no parâmetro “to_remove”.
conectar_bd():gera uma conexão com a base de dados para efetuar as operações de inserção de documento e mostrar documentos.
inserir_db(conteúdo):inserir o conteúdo passado como parametro na base de dados. Pode-se ser melhorada utilizando a criação do ObjectId utilizando alguma lógica para evitar documentos repetidos. Atualmente gera o _id randomicamente.

show_db():imprime os valores armazenados na base de dados como resultado. Não utiliza nenhum filtro ou metodologia.

Tipos de dados e Representação

Internamente a modelagem dos dados for feita diferente utilizando um vetor para armazenar o Vocabulário e um dicionário para armazenar as incidências ao qual é indexado com o valor do vetor e o vocabulário ao qual a incidência ocorre. Foi escolhida essa aproximação pois a base de dados para representação é NoSQL e permite uma melhor maleabilidade das informação.

Como trabalho futuro o ideal seria eliminar a armazenagem dos elementos que não aparecem no vetor tirando todos os elementos 0 do dicionário de incidências e ganhando assim um menor vetor de armazenamento.

A seguir é apresentado um output de um teste executado demonstrando as saídas de processamento para os 2 processo perpendiculares seguindo o exemplo dado na prova:

textos:['Falar é fácil. Mostre-me o código.', 'É fácil escrever código. Difícil é escrever código que funcione.']

Vocabulario-gram1[11]:{0: 'falar', 1: 'é', 2: 'fácil', 3: 'mostre', 4: 'me', 5: 'o', 6: 'código', 7: 'escrever', 8: 'difícil', 9: 'que', 10: 'funcione'}

vocabulario:{0: 'falar', 1: 'é', 2: 'fácil', 3: 'mostre', 4: 'me', 5: 'o', 6: 'código', 7: 'escrever', 8: 'difícil', 9: 'que', 10: 'funcione'}

Incidencia-[22]:{'1-falar': 1, '1-é': 1, '1-fácil': 1, '1-mostre': 1, '1-me': 1, '1-o': 1, '1-código': 1, '1-escrever': 0, '1-difícil': 0, '1-que': 0, '1-funcione': 0, '2-falar': 0, '2-é': 2, '2-fácil': 1, '2-mostre': 0, '2-me': 0, '2-o': 0, '2-código': 2, '2-escrever': 2, '2-difícil': 1, '2-que': 1, '2-funcione': 1}

vetores:{0: ['falar', 'é', 'fácil', 'mostre', 'me', 'o', 'código'], 1: ['é', 'fácil', 'escrever', 'código', 'difícil', 'é', 'escrever', 'código', 'que', 'funcione']}

Final:{'vocabulario': 'Vocabulario :[1.falar 2.é 3.fácil 4.mostre 5.me 6.o 7.código 8.escrever 9.difícil 10.que 11.funcione ', 'vetor': 'texto 1:[1,1,1,1,1,1,1,0,0,0,0]texto 2:[0,2,1,0,0,0,2,2,1,1,1]'}

textos:['Falar é fácil. Mostre-me o código.', 'É fácil escrever código. Difícil é escrever código que funcione.']

Vocabulario-gram2[13]:{0: 'falar é', 1: 'é fácil', 2: 'fácil mostre', 3: 'mostre me', 4: 'me o', 5: 'o código', 6: 'fácil escrever', 7: 'escrever código', 8: 'código difícil', 9: 'difícil é', 10: 'é escrever', 11: 'código que', 12: 'que funcione'}

vocabulario:{0: 'falar é', 1: 'é fácil', 2: 'fácil mostre', 3: 'mostre me', 4: 'me o', 5: 'o código', 6: 'fácil escrever', 7: 'escrever código', 8: 'código difícil', 9: 'difícil é', 10: 'é escrever', 11: 'código que', 12: 'que funcione'}

Incidencia-[26]:{'1-falar é': 1, '1-é fácil': 1, '1-fácil mostre': 1, '1-mostre me': 1, '1-me o': 1, '1-o código': 1, '2-é fácil': 1, '2-fácil escrever': 1, '2-escrever código': '2', '2-código difícil': 1, '2-difícil é': 1, '2-é escrever': 1, '2-código que': 1, '2-que funcione': 1, '2-falar é': 0, '2-fácil mostre': 0, '2-mostre me': 0, '2-me o': 0, '2-o código': 0, '1-fácil escrever': 0, '1-escrever código': 0, '1-código difícil': 0, '1-difícil é': 0, '1-é escrever': 0, '1-código que': 0, '1-que funcione': 0}

vetores:{0: ['falar', 'é', 'fácil', 'mostre', 'me', 'o', 'código'], 1: ['é', 'fácil', 'escrever', 'código', 'difícil', 'é', 'escrever', 'código', 'que', 'funcione']}

Final:{'vocabulario': 'Vocabulario :[1.falar é 2.é fácil 3.fácil mostre 4.mostre me 5.me o 6.o código 7.fácil escrever 8.escrever código 9.código difícil 10.difícil é 11.é escrever 12.código que 13.que funcione]', 'vetor': 'texto 1:[1,1,1,1,1,1,0,0,0,0,0,0,0]texto 2:[0,1,0,0,0,0,1,2,1,1,1,1,1]'}

Conclusão

O desenvolvimento foi feito idealizando o resultado desejado, e caso existisse alguma maleabilidade para dispor as informações seria muito mais simples, Como por exemplo aplicar a regra de criação de conjunto set() ao conjunto de strings de texto para eliminar todas as strings repetidas e ficar apenas com o conjunto de Vocabulário, porém esta solução não retorna os elementos sempre na mesma ordem e mesmo aplicando um sorted() a ordem não seria a apresentada no resultado exigido pela prova.

A aplicação consegue realizar a execução dos processos solicitados porém possui apenas algumas pequenas discrepâncias de resultado normalmente relacionadas a entrada de vetores muito pequenos para o processamento, Logo o ideal é a entrada de vetor com tamanho de palavras 2 ou superiores. Outra delimitação da aplicação é quanto ao uso da base de dados, em testes o processamento leva um tempo ínfimo, porém ao ativar a base de dados por si só o processamento ganha 1 s devido ao uso de base de dados em nível gratuito.