Estudo de Caso - Frequência de Palavras

1. Descrição do problema

Deve-se criar um programa em Javascript, respeitando os aspectos de programação funcional vistos, para fazer a leitura do conteúdo de dois (ou mais) arquivos de texto, contabilizar o número total de aparições de cada palavra* existente, o quanto representa percentualmente esse total de aparições (em relação ao número total) e gerar um novo arquivo contendo esse resultado.

Naturalmente, o processamento a ser realizado deverá ser dividido entre várias funções, cada uma responsável por uma tarefa ou sub-tarefa. Uma função principal, então, chamará todas as outras seguindo uma sequência (ou composição) lógica para que o problema seja resolvido.

* *Apenas palavras devem ser contabilizadas (ex. substantivos, adjetivos, pronomes, verbos, advérbios, conjunções, preposições, etc.). Ou seja, números, datas, sinais de pontuação, e vários outros caracteres especiais devem ser sumariamente excluídos da contagem.*

2. Como apresentar a solução?

1. Arquivo `solucao.js`

Contém todo o código Javascript para resolver o problema. Cada função de apoio criada deve vir acompanhada de um comentário explicando seu funcionamento.

Deve existir uma função principal chamada `gerarResultado(dados)` que é a responsável por chamar todas as outras na sequência adequada, de acordo com o algoritmo mental que foi elaborado para resolver o problema.

2. Arquivos `dados1.txt` e `dados2.txt`

Cada arquivo contém um texto copiado de portais/blogs da Internet de escolha livre (mas devem ser em português do Brasil). É importante que os conteúdos contenham boa variedade de termos como palavras, números, sinais de pontuação, caracteres especiais, etc, de modo que as tarefas de processamento realizadas mostrem todo seu potencial. A primeira linha de cada arquivo deve conter a url completa da fonte usada (essas urls também serão, portanto, alvos de processamento).

Exemplo de arquivo `dados1.txt` :

https://saense.com.br/2019/07/conhecimento-desconhecido/
Há quase 4 anos, em meu primeiro artigo aqui nesse portal "Máquinas que compreendem a linguagem humana", escrevi sobre o estrelado sistema Watson/IBM e seu grande feito no mundo das disputas de conhecimento televisas no melhor estilo "Quem que ser um milionário?" lá no início dessa década; não apenas isso, mas principalmente sobre sua enorme capacidade de analisar a linguagem...
...

Exemplo de arquivo `dados2.txt` :

https://saense.com.br/2019/06/tragam-nos-de-volta-os-genios-e-os-genios-foram-trazidos/
Se existe uma obra artística impossível de tirar da cabeça é a da pintura surrealista "La persistència de la memòria" (1931) do mestre catalão Salvador Domingo Felipe Jacinto Dali i Domènech. Sabe aquela com os relógios derretidos e uma paisagem com mar no horizonte? Tenho ela na mente desde garoto. Um dia fui procurar pela explicação da obra nas palavras do próprio Dalí: "Toda a minha ambição no campo pictórico é materializar as imagens da irracionalidade concreta com a mais imperialista fúria da precisão". Ah, tá! 😕
...

3. Arquivo `resultado.txt`

Contém o resultado do processamento realizado: uma lista de registros (objetos) das palavras existentes nos textos. Cada registro deve conter a palavra em si, a quantidade de ocorrências desta em todo o texto e o percentual em relação ao total.

Supondo que os textos tenham um total de 2700 palavras (incluindo repetições), o arquivo `resultado.txt` iria conter uma lista semelhante à do exemplo abaixo:

```
[
  {"palavra": "que", "qtde": 135, "%": 5},
  {"palavra": "está", "qtde": 82}, "%": 3},
  ...
  {"palavra": "Eustáquio", "qtde": 1, "%": 0.03},
  {"palavra": "paralelepípedo", "qtde": 1, "%": 0.03}
]
```

O exemplo mostra que a palavra "que" apareceu 135 vezes, o que representaria 5 do total de palavras do texto ($135/2700 = 0.05 = 5$), enquanto que a palavra "paralelepípedo" apareceu apenas uma vez, representando, portanto, apenas 0.03 do total.

Observe ainda que a lista deve estar **ordenada de forma descrescente** pela quantidade de aparições (`qtde`).

3. Algumas dicas e sugestões

1. Pode-se elaborar um modelo mental de solução do problema e um rabisco detalhado no papel de todo o processamento que será necessário realizar nos dois arquivos de texto para compor o arquivo final contendo o resultado.
2. Observe que a solução pode ser composta por várias tarefas. Cada tarefa pode ser representada por uma função ou mais de uma. Listo abaixo algumas macro-tarefas gerais que podem ser interessantes/úteis:

- Ler os arquivos de dados e gerar uma única string representativa dos dois conteúdos;
- Eliminar elementos indesejados
  - `'.', '?', '"','♪','_','[','(','%', etc...` (dica: é possível criar uma constante lista de caracteres indesejados)
  - datas e horas: `dd/mm/aaaa`, `HH:MM:SS`, `HHhMM` (ex. 13h50), etc..
  - valores numéricos de uma maneira geral: `5`, `3.14`, `911`, `-2,3`, etc...
  - dica: pode-se fazer uso de expressões regulares (*regex*) em Javascript para identificar variados tipos de padrões indesejados de forma mais elegante e robusta (existem vários tutoriais a exemplo de https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_Expressions)

3. Pode-se fazer uso de algumas funções pre-definidas de JS para auxiliar na solução de algumas sub-tarefas. Exemplos:

- Para ordenar as palavras pela frequência (ao final do processamento), pode-se usar a função pré-definida `sort(...)`;
- Para eliminar espaços em brancos de strings, ou testar se uma string possui um dado elemento ou se termina com algum padrão textual, entre outros, pode-se usar: `trim()`, `includes(...)`, `endsWith(...)`, etc;

4. Pode-se fazer uso de *composição de funções* para organizar ainda mais o processamento. Ou seja, a sua função principal, que deve se chamar `gerarResultado`, pode ser uma composição de todas as outras que foram definidas na sequência de processamento correta.

```
const gerarResultado = composicao(
    funcao1,
    funcao2,
    ...,
    funcaoN
)

const dados = ['dados1.txt', 'dados2.txt']
gerarResultado(dados)
```

4. Uma solução disponível

O arquivo disponível para download a seguir é uma solução criada por um aluno de uma turma da disciplina de Programação Funcional do DCOMP/UFS. O arquivo compactado contém os arquivos de dados .txt, o arquivo resultado.txt e o arquivo solucao.js.

Recomenda-se tentar resolver o problema antes de visitar essa solução. Ainda que tome-se conhecimento da solução disponível, recomenda-se realizar modificações e testes nas variadas funções para experimentar e treinar alternativas interessantes. Recomenda-se ainda alterar os textos dos arquivos de dados de entrada e observar o efeito no novo arquivo de resultado a ser gerado.

SOLUÇÃO