

# Projeto individual com revisão por pares - Lista Invertida

Enviar tarefa

**Vencimento** Quarta-feira por 23:59 **Pontos** 7

**Enviando** uma caixa de entrada de texto, uma gravação de mídia, ou um upload de arquivo

**Tipos de arquivo** zip, rar, e pdf **Disponível** até 3 jun em 23:59

Neste projeto, vocês deverão implementar uma lista invertida. Essa lista deverá retornar, a partir de um conjunto de nomes (possivelmente armazenados em um arquivo), quais são aqueles que possuem um determinado conjunto de palavras, independentemente da sua posição ou da sua ordem.

A lista deve ser armazenada em arquivo, de tal forma que possa ser vinculada a projetos como o do Amigo Oculto. Você deverá implementar dois métodos:

- `create( id, nome )` - Esse método será o responsável pela inclusão de um novo nome na lista. Na realidade, ele deverá extrair cada palavra do nome e inseri-la separadamente na lista. A vinculação de cada palavra a um nome será feita por meio do seu ID.
- `int[] ids = read( [lista de palavras] )` - A consulta será feita por meio desse método, que receberá uma lista de palavras como parâmetro. Ela deve retornar todos os IDs cujos nomes atendam ao critério selecionado.

## FUNCIONAMENTO DA LISTA INVERTIDA

Da mesma forma que na tabela *hash* dinâmica, você precisará de dois arquivos. O primeiro arquivo será uma estrutura mais simples: uma tabela composta por registros do tipo [*termo*, *endereço*]. O *endereço* apontará para a lista encadeada desse termo (ou palavra) no segundo arquivo, que será uma lista de IDs (dos nomes que contém esse termo).

Veja essa lista de nomes a ser armazenada e os exemplos dos arquivos:

Lista de Nomes

ID	Nome
1	Marcos Antônio de Oliveira
2	José Marcos Resende
3	Paula Oliveira
4	Carlos José Antônio Souza
5	José Carlos de Paula

Para ela teríamos a lista de termos abaixo. Note que os termos foram transformados para letras minúsculas e os acentos foram removidos. Também retiramos as stop words do processo (ex.: "de"). O endereço apresentado representa o endereço da lista do termo no segundo arquivo

Termos  
(primeiro arquivo)

Termo	Endereço
marcos	50
antonio	100
oliveira	150
jose	200
resende	250
paula	300
carlos	350
souza	400

Abaixo, está o segundo arquivo contendo a lista de IDs de cada termo:

Listas de ID (segundo arquivo)

Endereço	N	IDs (até 10)	Próximo
50	2	1, 2	-1
100	2	1, 4	-1
150	2	1, 3	-1
200	3	2, 4, 5	-1
250	1	2	-1
300	2	3, 5	-1
350	2	4, 5	-1
400	1	4	-1

Aqui, N indica a quantidade de IDs já presentes no bloco, que deve conter até 10 IDs. Se esse bloco estourar, então um novo bloco deve ser criado no fim do arquivo e o encadeamento deverá ser feito por meio do ponteiro Próximo.

Ao consultarmos essa tabela, teremos uma lista de termos. Por exemplo, suponha que a string de busca seja "José de Paula" e, assim, teremos o conjunto [ "jose", "paula"], já transformado e sem as *stop words*.

O primeiro termo será usado para a construção do conjunto resposta e seu valor será:

*resposta* = [2, 4, 5]

A partir daí, tomamos a lista de cada outro termo do conjunto de consulta (ex: [3, 5] do termo "paula") e verificamos quais elementos do conjunto resposta aparecem nessa outra lista. Aqueles que não aparecerem, são removidos. O nosso novo conjunto resposta fica assim:

*resposta* = [5]

Quando terminarmos de processar a listas dos outros termos, basta devolvermos o conjunto resposta.

## ESTRUTURA DA LISTA INVERTIDA

A melhor estrutura para a tabela do primeiro arquivo é uma tabela *hash*, mas, considerando o curto espaço de tempo disponível, você poderá fazer um arquivo com registros sequenciais desordenados (e usar sempre a busca sequencial). Assim, toda nova inclusão poderá ser feita no fim do arquivo.

Essa estrutura deve permitir duas operações:

- Inserção do registro [*termo*, *endereço*], sendo que o endereço deverá apontar para o fim do segundo arquivo (pois lá, as inclusões também serão feitas no fim do arquivo). Assim, a interface do método deverá ser:

```
int id = create(termo, endereço);
```

- Busca de um determinado termo. Quando fizermos a consulta, usaremos um método que, dado um termo, retornará o seu endereço. Esse método deverá retornar -1 se o termo não existir. Sua interface deve ser:

```
long endereço = read(termo);
```

No segundo arquivo, teremos as listas invertidas de cada termo. Basicamente, estamos falando de uma estrutura que conterá listas de IDs. Aqui vale a pena você usar uma estrutura de blocos, para evitar leituras lentas por causa de deslocamentos no arquivo.

Crie algo como um bloco de uns 10 IDs (ou qualquer outra quantidade). Esse bloco deve ter, no fim, um ponteiro para o próximo bloco, caso você tenha mais de 10 IDs vinculados àquele termo). Use também um contador de IDs para saber quantos IDs estão na lista e facilitar as operações.

Nesse segundo arquivo, você também deve ter as operações de inclusão e leitura também. Na leitura, a partir do endereço retornado pela consulta ao primeiro arquivo, você irá até esse endereço e recuperará o bloco de IDs. Você deve acumular todos esses IDs em um vetor e, se o ponteiro do bloco não estiver vazios, avance para o próximo bloco e continue lendo, até não existirem mais blocos de IDs para aquele termo.

Na inclusão, você deve seguir até o seu endereço e ver se o bloco de IDs comporta já não tem esse ID. Se tiver, ignore a inclusão. Se não tiver, você precisa testar se cabe mais um ID no bloco. Se couber, insira o ID no bloco e atualize o registro. Se não couber, atualize o ponteiro do registro atual para o fim do arquivo e insira um novo registro nesse fim de arquivo, com um bloco que conterá apenas esse novo ID.

## ORIENTAÇÕES GERAIS

- O código fonte deve ser comentado, organizado e anexado à atividade, de tal forma que permita a sua execução com facilidade. É esperado que um pequeno documento nos formatos texto ou pdf seja anexado com uma descrição sucinta o projeto (facilitando a observação dos pares).

- Você deve implementar as operações necessárias para fazermos a inclusão de pares de nomes e IDs, bem como a consulta de termos (que retorna IDs). Se possível, liste também os nomes das pessoas associadas a esses IDs (você pode usar um CRUD para isso), mas isso é opcional.
- O trabalho é individual. Cada aluno deve enviar o seu próprio projeto.
  - A avaliação será por pares. Cada aluno observará o trabalho de 3 colegas e registrará o que foi observado em uma tabela de critérios. Essa avaliação não será usada para atribuir nota aos colegas, mas servirá como um apoio para a correção do professor.