

ACH2024 - Algoritmos e Estrutura de Dados II

Exercício Programa 2 - Busca em textos

Prazo de entrega: 02/07/2017

Versão preliminar (faltam apenas alguns poucos trechos sinalizados como “*under construction...*”)

1. Descrição

Nesse exercício-programa trataremos as abordagens de busca em textos utilizando os algoritmos KMP e TRIE.

Este EP foi dividido em 3 sub-problemas, os EPs 2.1, 2.2 e o 2.3. Os dois primeiros deverão ser submetidos diretamente na plataforma URI Online Judge e o 2.3 deverá ser submetido pelo TIDIA.

Os EPs 2.1 e 2.2 deverão ser implementados utilizando exclusivamente TRIES. Já no EP 2.3, você deverá implementar os problemas 2.1 e 2.2 com o KMP e comparar o desempenho computacional dos dois algoritmos.

2. Sobre o URI Online Judge

O URI Online Judge é uma plataforma WEB desenvolvida pelo Departamento de Ciência da Computação da Universidade Regional Integrada (URI) do Rio Grande do Sul. O principal objetivo é promover a prática de programação e o compartilhamento de conhecimento.

Nele será possível identificar, com maior rapidez, a corretude da implementação do problema. O portal possui um juiz virtual, que de forma automática, verifica se seu código está correto.

Para conseguir submeter o seu código, que contém a solução aos problemas, siga os seguintes passos:

2.1: Cadastre-se na plataforma acessando o endereço:
<https://www.urionlinejudge.com.br/judge/pt/login>

2.2: Procure seu código de identificação acessando PERFIL > CONFIGURAÇÕES.

Hi, Z Student

HOME PERFIL NEWS ACADEMIC CONTESTS PROBLEMAS SUBMISSÕES RANKS SAIR

URI ONLINE JUDGE PROBLEMS & CONTESTS

CONFIGURAÇÕES

ADMINISTRE SUAS PREFERÊNCIAS DE CONTA.

INFORMAÇÃO PESSOAL

NOME	IDENTIFICAÇÃO
Victor Jatobá	07707
EMAIL	ATUALIZAR MINHAS CONFIGURAÇÕES DE SEGURANÇA (SENHA).
ÍCONE DE PERFIL	GÊNERO
Gravatar	Masculino

INFORMAÇÃO EDUCACIONAL

UNIVERSIDADE

2.3: Se você pertence à turma ACH2024_Turma94_1Sem_2017_ET, cadastre seu código de identificação na planilha online:

https://docs.google.com/spreadsheets/d/1acB4RD7JQtUh0c-_mpaN0nN42PB1vBdIYAmv30vNvzM/edit?usp=sharing

Se você pertence à turma ACH2024_Turma04_1Sem_2017_ET, cadastre seu código de identificação na planilha online:

https://docs.google.com/spreadsheets/d/1jw0oCV6YDJuF9I7f1of__8A4eAa0ZdLKAU0LCM68bE/edit?usp=sharing

Obs.: para verificar a qual turma você pertence, basta verificar o nome da turma no Tidia ou acessar as duas planilhas e verificar onde o seu nome está cadastrado.

Aguarde até que o monitor libere o seu acesso aos problemas. Qualquer dúvida basta enviar um email para victorjatoba@usp.br.

3. Exercício Programa 2.1

Você deverá implementar um programa que recebe um texto e um conjunto de palavras (strings de busca) e deve retornar a primeira posição de cada palavra que está presente no texto.

Constraints (podem assumir essas coisas, não precisam verificar):

- Palavras: pode conter apenas letras minúsculas do alfabeto ('a'-'z').
- Texto: pode conter apenas letras minúsculas do alfabeto ('a'-'z') e **espaços em branco**. Texto não contém quebra de linha ('\n') e nem caracteres especiais.
- O tamanho das palavras nunca será maior que o tamanho do texto.
- Posição: você deve considerar que a primeira posição do texto deve ser a posição 0 (zero). Assim, em um texto com 50 caracteres, a última posição será a 49.
- Você deve procurar apenas as palavras exatas, **NÃO VALE SUBPALAVRA**. Por exemplo, a palavra AMAR é diferente de PROGRAMAR.

Input:

O arquivo de entrada contém exatamente:

- Na primeira linha: O texto, que contém no mínimo 50 e no máximo 1600 caracteres.
- Na segunda linha: A quantidade de palavras a serem lidas. Com no mínimo 1 e no máximo 128.
- Na terceira linha: As palavras separadas por espaço. Cada palavra terá no mínimo 2 e no máximo 50 caracteres.

Output

Para cada palavra de busca deve ser impressa uma linha com a(s) posição(s) da primeira letra onde a palavra está localizada no texto. Lembre-se que a mesma palavra pode estar repetida no texto. Caso isso ocorra, as posições devem ser impressas uma na frente da outra, separadas por espaços em branco. Caso a palavra não esteja contida no texto, retorne -1 (número um negativo).

Exemplos:

#0

Entrada:

see a bear sell stock see a bull buy stock bid stock bid stock hear the bell
stop
3
bear bid hear

Saída esperada:

6
43 53
63

#1

Entrada:

lorem ipsum dolor sit amet meis illum nec at summo cetero et usu adhuc justo
tacimates cum et sint pericula mei eu pri ipsum eruditi periculis an no usu
graecis explicari has animal sententiae in ut oportere suscipiantur mea ex est
ullum quaestio in sit eius tibi que no dolore numquam qui sed malorum persius
utroque te ei sed omittam dissentias quaerendum ipsum altera vocent at cum
facilisis iracundia sea ea mel tollit eripuit ex ne mei discere albucius sit
tation convenire interesset at est te modus augue ei tempor assueverit eam ius
causae definiebas at te wisi vituperata eos quem feugait vulputate mel et eum
ut dicat ornatus pro cu prima deleniti patrioque ex mel ridens doctus mel
consul volumus noluisse te mel oblique noluisse an te vis errem consulatu
theophrastus est ne atomorum intellegam et mei scripta admodum has cu tollit
primis essent exerci equidem vix te his ut sonet elaboraret qui at dicam
epicurei et vel saepe instructior in soluta percipitur est quo reque voluptatum
utfacilis tibi que sapientem qui ut quo scripta voluptaria ad mea at possit
nusquam mandamus dui facer legimus te sea id sale meis atqui nec scripta
antiopam qui te nominavi mnesarchum incorrupte ut his qui ei putant impedit
facilis partem nullam elaboraret vix id id probatus omittantur pro eum in
ornatus repudiandae id qui alterum honestatis disputando errem graeco audiam
vim ne
1
ipsum

Saída esperada:

6 119 357

4. Exercício Programa 2.2

Você deverá implementar um programa que recebe uma lista de pequenos textos seguidos de palavras (strings de busca) e deve retornar a primeira posição dessas palavras em cada texto.

Constraints:

- Palavras: pode conter apenas letras minúsculas do alfabeto ('a'-'z').
- Texto: pode conter apenas letras minúsculas do alfabeto ('a'-'z') e **espaços em branco**. Texto não contém quebra de linha ('\n') e nem caracteres especiais.
- O tamanho das palavras nunca será maior que o tamanho do texto.
- Posição: você deve considerar que a primeira posição do texto deve ser a posição 0 (zero). Assim, em um texto com 50 caracteres, a última posição será a 49.
- Você deve procurar apenas as palavras exatas, **NÃO VALE SUBPALAVRA**. Por exemplo, a palavra AMAR é diferente de PROGRAMAR.

Input:

O arquivo de entrada contém exatamente, em cada linha:

- Na primeira linha: A quantidade de textos ($2 \leq q \leq 100000$) a serem lidos, que também será a mesma quantidade de palavras a serem buscadas em cada texto. Em outras palavras, será fornecido uma palavra para cada texto.
- Nas linhas seguintes, para cada entrada, o programa irá receber, consecutivamente:
 - O texto de no mínimo 10 e no máximo 128 caracteres;
 - A palavra a ser buscada. Cada palavra terá no mínimo 2 e no máximo 128 caracteres. Uma palavra nunca será maior que o texto.

Output:

Para cada par (texto/palavra de busca), deve ser impressa uma linha com a lista das posições da primeira letra onde a palavra está localizada no respectivo texto. Lembre-se que a mesma palavra pode estar repetida no texto. Caso isto ocorra, as posições devem ser impressas uma na frente da outra, separadas por espaços. Caso a

palavra não esteja contida no texto, retorne -1 (número um negativo).

Exemplos:

#0

Entrada:

```
3
see a bear sell stock see a bull buy stock bid stock bid stock hear the bell
bear
see a bear sell stock see a bull buy stock bid stock bid stock hear the bell
bid
see a bear sell stock see a bull buy stock bid stock bid stock hear the bell
hear
```

Saída esperada:

```
6
43 53
63
```

#1

Entrada:

```
3
lorem ipsum dolor sit amet
ipsum
meis illum nec at summo cetero et usu adhuc justo tacimates cum et sint
pericula mei eu pri ipsum eruditi periculis an no
meis illum nec at summo cetero et usu adhuc justo tacimates cum et sint
pericula mei eu pri ipsum eruditi periculis an no
aaaa
aa
```

Saída esperada:

```
6
0
-1
```

5. Exercício Programa 2.3

Você deve realizar duas implementações para os problemas ep2.1 e o ep2.2. Uma utilizando tries e outra utilizando o algoritmo KMP. Após a implementação, você deve informar qual foi o tempo total (em

milisegundos, com no máximo 4 casas decimais) de execução.

Você também deve responder às seguintes perguntas:

- 1.) Qual a melhor opção para o problema ep2.1? TRIE ou KMP?
- 2.) Qual a melhor opção para o problema ep2.2? TRIE ou KMP?

Input:

Para este problema, você deve contabilizar cada tempo de execução considerando, EXATAMENTE, as entradas abaixo:

Entrada para o ep2.1:

#Entrada 1

(Under construction...)

#Entrada 2

(Under construction...)

Entrada para o ep2.2:

#Entrada 1

(Under construction...)

#Entrada 2

(Under construction...)

Output:

Você deve enviar no tidia, os códigos-fonte dos algoritmos do KMP e do TRIE que você implementou. O nome dos arquivos devem conter seu número usp seguido do seu último nome e depois pela palavra kmp (caso esse seja o arquivo da implementação do KMP) ou trie, separados pelo caractere underscore ('_') com a extensão .c no final.

Exemplos de arquivos .c esperados:

#0

Nome dos arquivos:

7752093_jatoba_kmp.c

7752093_jatoba_trie.c

Além disso, você deve enviar um arquivo .txt, também para o tidia, contendo exatamente o padrão descrito abaixo. Arquivos fora do padrão serão automaticamente rejeitados.

Para cada entrada informada na subseção "Input", você deve imprimir o tempo de execução do algoritmo TRIE seguido do tempo do KMP. Nesse caso, as duas primeiras linhas representam os tempos de execução para o ep2.1 considerando as duas entradas fornecidas na subseção input. A terceira linha deve estar em branco (quebra de linha, '\n'). As linhas 4 e 5 devem conter os tempos para o ep2.2 considerando as duas entradas fornecidas na subseção input.

A sexta linha você deve deixar em branco (quebra de linha '\n').

No final você deve colocar a sua resposta para cada questão, separados por espaço. Para cada questão (1 e 2) sua resposta deve ser, EXCLUSIVAMENTE ou TRIE ou KMP (em maiúsculo).

O nome do arquivo deve conter seu número usp seguido do seu último nome, separados pelo caractere underscore ('_') com a extensão .txt no final.

Exemplos de arquivos .txt esperados:

#0

Nome do arquivo:

7752093_jatoba.txt

Conteúdo do arquivo:

0.2431 2.4215

0.2431 2.4215

0.2431 2.4215

0.2431 2.4215

KMP TRIE

#1

Nome do arquivo:

000000_norris.txt

Conteúdo do arquivo:

0.0001 0.0002
0.0000 0.0000

0.0001 0.0000
0.0000 0.0000

KMP KMP

DICAS DE COMO CALCULAR O TEMPO DE EXECUÇÃO: (*Under construction...*)
Mas adiantando, para quem usa Linux há o comando **time**