Técnicas de Busca e Ordenação
Roteiro de Laboratório – *Arrays* de Sufixos

1 Objetivo

O objetivo deste laboratório é implementar um método de busca em textos, chamado de *keyword-in-context (KWIC) search* (busca de palavras chave em contexto). Um método como esse é muito utilizado em variadas aplicações, tais como linguística, banco de dados, busca na *web*, processamento de texto, etc.

2 *Keyword-in-context search*

A definição do problema é como a seguir: dado um texto com N caracteres, fazer um pré-processamento para permitir uma busca rápida de *substrings*, encontrando todas as ocorrências, juntamente com o contexto, de uma *strings* a ser consultada (*query string*).

2.1 Arquivos de entrada

Foram fornecidos três arquivos de entrada:

- `abra.txt` é uma entrada de teste pequena para depuração.
- `tale.txt` é o texto do livro *A Tale of Two Cities* de *Charles Dickens*.
- `moby.txt` é o texto do livro *Moby Dick* de *Herman Melville*.

Obs.: Toda a análise deste laboratório também poderia ser realizada para textos em português (ou qualquer outra língua). O único motivo para usarmos um texto em inglês é evitar complicações na implementação devido à codificação de caracteres especiais como 'ç', 'é', etc.

Podemos ver o conteúdo inicial de um dos arquivos.

```
$ less tale.txt
726570
it was the best of times it was the worst of times
it was the age of wisdom it was the age of foolishness
it was the epoch of belief it was the epoch of incredulity
it was the season of light it was the season of darkness
it was the spring of hope it was the winter of despair
```

A primeira linha é o **número de caracteres** que o arquivo contém *a partir da segunda linha*. Essa informação não é essencial mas pode ser útil na hora de fazer a leitura do arquivo.

2.2 Busca com contexto

O resultado final que se deseja obter pode ser facilmente explicado com um exemplo. No comando abaixo, o número `15` é o *contexto*, isto é, o número de caracteres adicionais de cada lado que devem ser exibidos quando o termo da busca for encontrado.

```
$ ./a.out in/tale.txt 15
search
o st giless to search for contraband
her unavailing search for your fathe
le and gone in search of her husband
t provinces in search of impoverishe
 dispersing in search of other carri
n that bed and search the straw hold

better thing
t is a far far better thing that i do than
 some sense of better things else forgotte
was capable of better things mr carton ent

majesty
 most gracious majesty king george th
rnkeys and the majesty of the law fir
on against the majesty of the people
se them to his majestys chief secreta
h lists of his majestys forces and of

the worst
w the best and the worst are known to y
f them give me the worst first there th
for in case of the worst is a friend in
e roomdoor and the worst is over then a
pect mr darnay the worst its the wisest
is his brother the worst of a bad race
ss in them for the worst of health for
 you have seen the worst of her agitati
cumwented into the worst of luck buuust
n your brother the worst of the bad rac
 full share in the worst of the day pla
mes to himself the worst of the strife
f times it was the worst of times it wa
ould hope that the worst was over well
urage business the worst will be over i
clesiastics of the worst world worldly
```

O funcionamento resumido do programa é descrito pelos seguintes passos:

1. Pré-processar o arquivo de entrada. (Detalhes adiante.)
2. Ficar em *loop*, lendo uma *string* de consulta do terminal e exibir o resultado.
3. Terminar quando a *string* de consulta for vazia.

3 Passos para desenvolvimento do laboratório

Siga os passos a seguir, *na ordem apresentada*, para desenvolver o seu programa.

3.1 Passo 1: Ler e limpar a entrada

Você deve abrir o arquivo de entrada informado e "limpar" os caracteres extras de espaço em branco e quebra de linha (enter). No final, o resultado da leitura do arquivo deve ser uma grande *string* aonde cada palavra é separada por *apenas um* espaço, e não há mais quebra de linha. Por exemplo, a entrada

```
of comparison only

there     were a    king with
```

deve gerar como resultado

```
of comparison only there were a king with
```

Utilize o tipo de dado `String` apresentado em sala para criar a sua *string* final. (Veja os `str.{h, c}` relativos a aula de *Radix Sort* (18) no Classroom.)

3.2 Passo 2: Construir um *array* de sufixos

Um sufixo de uma *string* s é uma *substring* de s começando a partir de um i-ésimo caractere. Por exemplo, se `s = "abcd"`, então `suf(s, 2) = "cd"`, contando os caracteres a partir de $i = 0$. Assim, para determinar um sufixo, precisamos de duas informações: a *string* s e o índice i. Podemos então criar uma estrutura como abaixo.

```
typedef struct {
    String *s;
    int index;
} Suffix;
```

Criada essa estrutura, você deve construir um *array* de `Suffix*`, criando todos os sufixos para o texto de entrada (*string* gerada no passo 1), variando o índice de 0 até $N - 1$, aonde N é o tamanho do texto. Um exemplo do resultado esperado para esse passo:

```
$ ./a.out in/abra.txt
ABRACADABRA!
BRACADABRA!
RACADABRA!
ACADABRA!
CADABRA!
ADABRA!
DABRA!
ABRA!
BRA!
RA!
A!
!
```

Obviamente, o *array* de sufixos fica bem grande quando o texto cresce. Mas como não há cópia de *strings*, não há um consumo excessivo de memória.

3.3 Passo 3: Ordenar *array* de sufixos

O próximo passo é ordenar o *array* de sufixos. Isso é simples uma vez que cada sufixo é uma *substring* do texto de entrada. Assim, basta criar uma função de comparação para sufixos similar à função `compare_from` para *strings* (no arquivo `str.c`). A regra de comparação de sufixos é a mesma para *strings* de tamanho variável. Um exemplo do resultado esperado para esse passo:

```
$ ./a.out in/abra.txt
!
A!
ABRA!
ABRACADABRA!
ACADABRA!
ADABRA!
BRA!
BRACADABRA!
CADABRA!
DABRA!
RA!
RACADABRA!
```

Para fazer a ordenação do *array* de sufixos, utilize a função de sistema `qsort`. Depois vamos compará-la com um *MSD radix sort*.

3.4 Passo 4: Realizar uma consulta

Leia a *string* de consulta `query` do terminal e realize a busca no *array* de sufixos. Note que como agora o *array* está ordenado, você pode fazer uma *busca binária* no *array*, procurando a primeira posição aonde `query` aparece no começo do sufixo, e varrendo todas as posições do *array* sequencialmente até `query` não aparecer mais. O valor `index` no sufixo indica a posição aonde o termo se encontra no texto. Assim, basta exibir os caracteres no intervalo `(index - context, index + context)`. A figura abaixo ilustra como fazer a busca.

4 *MSD radix sort*

A ordenação do *array* de sufixos é um bom caso de teste para a aplicação de um método de *radix sort*, pois todos os sufixos têm muitas partes em comum por terem sido gerados a partir do mesmo texto.

Baseando-se no código da aula de *Radix Sort* (18), implemente um *MSD radix sort* para ordenar o *array* de sufixos, e compare o seu desempenho contra a função de ordenação do sistema usada anteriormente.