2o trabalho de Estrutura de Dados II
Busca em Memória Principal

Este trabalho trata de ***Busca em Memória Principal*** e deverá ser realizado em dupla.
O objetivo 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.

A avaliação ocorrerá em três etapas sequenciais:

1) Entrega do código fonte conforme a especificação, compilando e gerando o resultado esperado para os casos de teste fornecidos;
    a) O código deverá aplicar boas práticas de programação, como comentários, modularização e reuso de código. Por exemplo, o mesmo código poderia ser facilmente modificado para outros tipos de dados, ou outros métodos poderiam ser facilmente acrescentados.
2) Relatório comparando e explicando os resultados obtidos, em particular em relação ao desempenho dos métodos de ordenação utilizados.
3) Entrevista com os componentes do grupo sobre o trabalho executado.

A falha em uma etapa implica que as seguintes são automaticamente zeradas. Falha em responder as questões da entrevista, demonstrando que o entrevistado não compreende o próprio trabalho, implica em perda de pontos para a dupla. Portanto, se o seu companheiro do trabalho não estiver interessado na compreensão do mesmo, comunique ao professor para entregar o trabalho individualmente.

Atenção: plágio não será tolerado! Os grupos envolvidos receberão nota zero. Se algum colega pedir ajuda, não forneça o seu código como exemplo, mas trabalhem juntos sobre o código dele/dela.

Bônus: a (correta) inclusão/análise de outros métodos além dos exigidos ou alguma melhoria no método de busca proposto permitirá à dupla recuperar pontos (eventualmente) perdidos na avaliação (deixe claro no relatório se houver tais inclusões)

Entrega: o aluno deverá realizar via Google Classroom o upload de dois arquivos: (i) um arquivo .zip contendo os arquivos fontes e makefile, e (ii) um arquivo .pdf para o relatório.

Especificação:

Dado um texto com N caracteres, encontrar todas as ocorrências de uma string Q a ser consultada (query string) juntamente com o contexto C. O contexto é o número de caracteres adicionais de cada lado que devem ser exibidos quando o termo da busca for encontrado. Deve-se fazer um pré processamento para permitir uma busca rápida de substrings usando array de sufixos.

Exemplo: Dado o texto N= "in/tale.txt", a query Q = "better thing" e o contexto C = 15, o resultado esperado é:

```
$ ./a.out -c in/abra.txt 15 "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
```

Passos para desenvolvimento do trabalho

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

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 fornecido nos arquivos str.{h, c}.

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 -a 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.

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 -o in/abra.txt
!
A!
ABRA!
ABRACADABRA!
ACADABRA!
ADABRA!
BRA!
BRACADABRA!
CADABRA!
DABRA!
RA!
RACADABRA!
```

Para fazer a ordenação do array de sufixos, duas opções devem ser disponibilizadas:

1) A função de sistema qsort;
2) Algum outro método de sua escolha.

Passo 4: Realizar uma consulta
Dada uma string de consulta *query*, 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 em que *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 em que o termo se encontra no texto. Assim, basta exibir os caracteres no intervalo (index - context, index + size_query + context). A figura abaixo ilustra como fazer a busca.


```
$ ./a.out -s 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
```

Execução: ./a.out [-aorcs] caminho_arquivo [contexto] [query]

O programa deverá considerar os seguintes parâmetros:

(a) Processa o texto e imprime o array de sufixos
(o) Processa o texto e imprime o array de sufixos ordenado
(r) Processa o texto, produz o array de sufixo, ordena usando dois (ou mais) métodos e imprime o tempo de ordenação para cada método.
(c) Dado um contexto e uma query, imprime as ocorrências encontradas
(s) Dado um contexto, lê queries do teclado e imprime as ocorrências encontradas, até que uma string vazia seja informada.

Arquivos de entrada

A primeira linha do arquivo texto 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.

Três arquivos de entrada são fornecidos:

- `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 trabalho 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.