{TP1}
Malditos Macacos!!

Algoritmos e Estruturas de Dados III

17 de Abril de 2018

1 Introdução

Mathias, como todo estudante, é facinado pela vida e por sua aleatoriedade. Porém, ele é um estudante esforçado e, por mais improvável e louca a ideia, ele não mede esforços para realizar seus desejos. Certa vez, ouviu falar do "Infinite Monkey Theorem". Ninguém duvidava de Mathias. Muito menos depois de conseguir alguns macacos para lhe ajudar (nenhum animal foi machucado para o enunciado deste trabalho).

Embora tenha conseguido os macacos, que digitam linhas de tamanho fixo (sim, eles conseguem fazer isso sem problemas), Mathias está com um problema! Ele quer ordenar essas linhas digitadas em um computador ancião que ele tem em casa. O problema é que de vez em quando, um pente de memória RAM falha, fazendo com que Mathias não tenha mais espaço o suficiente para armazenar todas as linhas nela.

Por isso, ele quer sua ajuda! Você é um amigo de Mathias meio preguiçoso (e que, por questões óbvias, tenta não se envolver muito com Mathias). Mathias tem quase todo o código pronto, ele só precisa de uma parte em específico: ordenar as linhas em uma quantidade pequena de memória. Você, apesar de não ser o único capaz de ajudar Mathias, foi a única pessoa que ele conseguiu encontrar (você não viu ele se aproximando...).

2 O que fazer?

Felizmente, Mathias já tem quase tudo pronto. Ele até criou um sistema para verificar quanta memória virtual (é só com isso que precisamos nos preocupar). Então, você deve fazer o *download* do código dele para se livrar logo desta tarefa.

Após o download, o único arquivo a ser editado é: `sort.c`. Nele, você encontrará duas funções com uma breve descrição delas:

1. `a_menor_que_b`: você deve implementar esta função de forma que ela retorne `1` se a cadeia de caracteres `a` é menor que a cadeia de caracteres `b` (as duas com tamanho `len`);

```
1        int a_menor_que_b(char* a, char* b, int len);
```

2. `external_sort`: essa é a função principal! Você deve ler o arquivo `input_file`, ordená-lo e escrever as linhas ordenadas no arquivo `output_file`, usando apenas `memory` KB de memória virtual.

```
1        void external_sort(const char* input_file, const char*
             output_file, unsigned int memory);
```

Os formatos dos arquivos (de entrada e saída) estão especificados na próxima seção!

[1]IMPORTANTE: toda e qualquer variável do tipo *array* deverá ser alocado dinamicamente!

[2]IMPORTANTE: você não deve usar a função `malloc` e `free`, mas sim as versões de Mathias `mathias_malloc` e `mathias_free`.

3 O arquivo de entrada

O arquivo de entrada é bem simples! Além disso, Mathias aproveitou e colocou, na primeira linha, o tamanho das linhas que os macacos digitaram. Porém, ele esqueceu de colocar a quantidade de linhas após a primeira:

```
10
AXwW7"FN>Z
2G$y<#3.W!
:CtH[<?sYA
RiL!udRyK"
3C8GgoN@);
G3q?M:nbq$
```

Note que, por se tratar de vários macacos digitando rapidamente em paralelo, o número de linhas, assim como o tamanho das linhas pode ser bem grande! No entanto, fique tranquilo! Mathias te assegurou que você sempre conseguirá armazenar, pelo menos 2 linhas na memória.

4 O arquivo de saída

No arquivo de saída, você deverá imprimir algo parecido como o arquivo de entrada: na primeira linha deve haver um número com o tamanho dos registros; e nas linhas subsequentes deverão estar as linhas ordenadas em ordem lexicográfica.

```
10
2G$y<#3.W!
3C8GgoN@);
:CtH[<?sYA
AXwW7"FN>Z
G3q?M:nbq$
RiL!udRyK"
```

5 Compilando

ESTE TRABALHO DEVE SER FEITO EM UMA MÁQUINA COM LINUX!!

(Existem algumas bibliotecas e funções que só existem no Linux)

Mesmo que Mathias não entenda muito de ordenação de arquivos muito grande em computadores com recursos reduzidos, ele foi evangelizado como *Linux User*. Por isso, dentre os arquivos que ele te entregou estão:

- **Arquivos ".c" e ".h":** você não deve se preocupar com eles (a não ser o `sort.c`, que você deve implementar);
- **Makefile:** prontinho (você não precisa nem mecher nele!). Você só irá precisar de dois comandos:

```
        Compila todos os arquivos, criando um executável "tp1".
        # make
        Remove o executável, junto com todos da pasta "resultados/".
        # make clean
```

- **"run_tests.sh":** eles executam para todos os arquivos dentro da pasta `testes/`. Note que, por ser um arquivo ".sh", você deve executá-lo em uma distribuição Linux, com o seguinte comando:

```
        # ./run_tests.sh <memoria-em-KB>
```

6 Resultados

Assumindo que sua implementação no arquivo `sort.c` está correta, abaixo se encontra um exemplo de execução do programa "tp1":

```
# ./tp1 testes/test_10.5_2.txt resultados/test_10.5_2.txt.out 1
=------------- Informacoes -------------=
Entrada: testes/test_10.5_2.txt
Saida: resultados/test_10.5_2.txt.out
Memoria: 1
=---------------------------------------=
=------------- Resultados -------------=
Ordenado: SIM
Memoria: OK (0.1035 kb)
Tempo: 0.2110 milisegundos
=---------------------------------------=
```

Para que a resposta seja aceita (considerada como correta), no mínimo, ela deve estar ordenada e gastar menos memória do que o requisitado. Note que ao executar o programa `run_tests.sh`, ele executará "tp1" para todos os casos de testes presentes na pasta `testes`. Logo, várias dessas informações serão escritas na tela e, no final, uma lista com todos os casos de testes que não satisfizeram essas duas restrições, como podemos observar a seguir:

```
=-------------------------------------------------------------------=
=-------------------------- RESULTADOS -----------------------------=
=-------------------------------------------------------------------=
Falharam:
  testes/test_1000.500_10.txt
  testes/test_1000.500_1.txt
  testes/test_1000.500_2.txt
  testes/test_1000.500_3.txt
  testes/test_1000.500_4.txt
  testes/test_1000.500_5.txt
  testes/test_1000.500_6.txt
  testes/test_1000.500_7.txt
  testes/test_1000.500_8.txt
  testes/test_1000.500_9.txt
=-------------------------------------------------------------------=
```

7 O que entregar

Você deve submeter um arquivo compactado `seu_nome_sua_matricula.tar.gz` contendo[1]:

[1] Para gerar isso no linux usa-se o comando `tar`, no windows, use o 7-Zip (https://www.7-zip.org/)

1. SOMENTE O ARQUIVO `sort.c`;
2. sua documentação.

Sua documentação deve ter até 10 páginas contendo:

1. uma breve descrição do problema,
2. explicações das estruturas de dados e dos algoritmos utilizados para resolver o problema,
3. análise da complexidade da solução proposta (espaço e tempo – número de vezes de escrita e leitura em arquivos).
4. um experimento com o tempo gasto por execução para os diferentes arquivos teste (pode ser os que eu dei para vocês).

O seu TP deverá ser entregue de acordo com a data especificada no moodle. A penalidade em porcentagem para os TPs atrasados é dada pela fórmula $2^{(d-1)}/0.16$.