Centro Federal de Educação Tecnológica de Minas Gerais
Trabalho Prático 4 – **Disciplina**: Laboratório de Algoritmos e Estrutura de Dados I
Valor: 25 pontos – **Data de Entrega**: 27/06/2017 – 23:00h.
Professor: André Rodrigues da Cruz

Instruções

(1) Realiza-se o Trabalho Prático 4 (TP4) **individualmente**.

(2) O TP4 consiste em implementar e testar as estruturas de dados para pesquisa em tabela hash, e resolver dois problemas do URI Online Judge.

(3) Todo aluno deve ter uma conta do URI e fornecer ao professor o acesso as soluções elaboradas.

(4) O exercício proposto pelo professor totaliza 15 pontos. Para este, deve-se criar um único fonte cujo arquivo deve se chamar *nome-matricula.cpp*, um *makefile* e os arquivos gerados conforme explicação abaixo. Ao final, compacte tais arquivos no formato **zip**, cujo padrão de nome é *nome-matricula.zip*. Tal arquivo deverá ser submetido ao sistema ava.cefetmg.br até a data limite.

(5) O exercício do URI vale 10 pontos. São disponíveis 2 opções, cada um valendo 5 pontos, para o qual o aluno deve fazer todos.

(6) A pontuação de cada problema será dado pela concordância do que foi solicitado e implementado, e pela taxa de acerto obtida no mesmo.

(7) As soluções devem ser autênticas. Logo, não é permitido cópia parcial ou total de soluções entre alunos ou da Internet. Trabalhos copiados serão anulados[1].

(8) Se houver algum trecho de ocorrência de plágio em pelo menos um exercício, a nota total do TP4 será 0.

(9) No cabeçalho de cada código-fonte deve haver um conjunto de comentários com:

(a) O número de matrícula e o nome do aluno.

(b) Número e nome do problema URI solucionado pelo código-fonte.

Exemplo:

```
1 /* ALUNO: 201340506070 - Kurt Sloane
2  * PROBLEMA: 1055 - Soma Permutada Elegante
3  */
```

(10) Não se aceitará trabalhos entregues após a data especificada.

[1]Será aplicado o sistema antiplágio MOSS (Measure Of Software Similarity), desenvolvido por Alex Aiken (Stanford).

Problema Proposto pelo Professor

Deve-se implementar três versões da estrutura de dados tabela hash cuja chave é uma string[2], experimentá-las e analisar o número de colisões e tempo de pesquisa em *nanossegundos*. Para determinar a transformação da chave, deve-se utilizar a função hash[3] da STL e depois aplicar a congruência módulo M (dimensão da tabela). Serão realizados experimentos com cada par M, N (dimensão da tabela, número de entradas) tal que $M \in \{500, 1.000, 5.000\}$ e $N \in \{2.000, 7.000, 10.000\}$.

Para isto, as seguintes tarefas devem ser executadas:

a) (6 pontos) Implemente as tabelas hash com endereçamento aberto, com lista encadeada, e com árvore binária, com as operações de inserção, remoção e pesquisa.

b) (5 pontos) Implemente uma função que dado um par M, N:

   1) Insira, se possível, as N primeiras chaves contidas no arquivo *chaves-inserir.txt* na tabela de tamanho M, contabilize o número de colisões (quando não encontra um local adequado para a inserção), e determine a média amostral de tempo das inserções.
   2) Pesquise as primeiras 2.000 chaves contidas no arquivo *chaves-pesquisar.txt* na tabela de tamanho M, e determine a média amostral de tempo das pesquisa com sucesso.
   3) Apague, se existir, as primeiras 1.000 chaves contidas no arquivo *chaves-apagar.txt* na tabela de tamanho M, e determine a média amostral de tempo para deleção com sucesso.

c) (4 pontos) Para cada par M, N deve-se escrever na saída padrão o resultado do número de colisões e tempo médio solicitados conforme formatação abaixo:

```
Tamanho da tabela: M
Tamanho da entrada: N
Numero de colisoes na insercao: NCOL
Tempo medio para inserir um elemento: TINS (nanossegundos)
Tempo medio para consultar um elemento: TCON (nanossegundos)
Tempo medio para apagar um elemento da tabela: TDEL (nanossegundos)
```

Observações

(I) Para a computação do tempo é obrigatório o uso da biblioteca *chrono*.

(II) Quando não for mais possível inserir na tabela, devido a alguma razão de natureza da estrutura de dados, envie uma mensagem de erro para o usuário.

(III) Para gerar os arquivos *chaves-inserir.txt*, *chaves-pesquisar.txt*, e *chaves-apagar.txt*, utilize o código-fonte abaixo:

```
#include <fstream>
#include <vector>
#include <string>
#include <random>
#include <chrono>

using namespace std;

int main()
{
   const int num_keys = 10000;
   ofstream fins, fsch, fdel;
   int keysize, i, j;
```

[2] http://pt.cppreference.com/w/cpp/string/basic_string
[3] http://pt.cppreference.com/w/cpp/utility/hash

```
    string key;
    mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());
    uniform_int_distribution<int> unif_keysize(10, 30);
    uniform_int_distribution<int> unif_char(65, 90);
    uniform_real_distribution<double> unif_dbl(0, 1);

    fins.open("chaves-inserir.txt");
    fsch.open("chaves-pesquisar.txt");
    fdel.open("chaves-apagar.txt");

    for(i=0; i<num_keys; i++)
    {
       keysize = unif_keysize(rng);
       key = "";
       for(j=0; j<keysize; j++)
       key += static_cast<char>(unif_char(rng));
       fins << key << endl;

       if(unif_dbl(rng) < 0.3) fsch << key << endl;
       if(unif_dbl(rng) < 0.3) fdel << key << endl;
    }

    fins.close();
    fsch.close();
    fdel.close();

    return 0;
}
```