INF 213 - Roteiro da Aula Pratica 11

Objetivo: praticar hashing.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/open?id=1UcmSOc6ssjpfje71S_-Ws3qAEWglHgib

Etapa 1

Hashing possui inumeras aplicacoes. Exemplos:

- Estrutura de dados tabela hash.
- Armazenamento seguro de senhas em banco de dados.
- Calculo de verificadores (exemplo: digito verificador de CPF, md5sum de arquivos).

No Brasil, o CPF e' um numero composto por 9 digitos + 2 digitos verificadores. Assim como no caso do md5sum , o objetivo desses digitos verificadores e' reduzir a chance de aceitacao de valores errados (quando um usuario fornece um numero de CPF, por exemplo, podemos calcular os digitos verificadores dos 9 primeiros numeros e, assim, verificar se houve algum erro de digitacao).

Antes de prosseguir, pense nas seguintes perguntas (e discuta com um colega/professor):
1) se os digitos verificadores não corresponderem aos fornecidos pelo usuario, isso significa que houve algum erro de digitacao (supondo, claro, que o usuario tentou digitar seu CPF corretamente)?
2) se os digitos verificadores corresponderem aos fornecidos pelo usuario, isso significa que não houve erros de digitacao?

O "hash" do CPF e' calculado da seguinte forma: multiplica-se cada digito do CPF (da esquerda para a direita) por um dos numeros 10, 9, 8, … 2 e, entao, calcula-se a soma *S* desses produtos.

A seguir, calcula-se o resto da divisao entre 10*S* e 11.

Exemplo, se os 9 primeiros digitos de um CPF for: 123555876 (123555876-20, com os dois digitos verificadores), entao S = 1*10 + 2 *9 + 3*8 + 5*7 + 5*6 + 5*5 + 8*4 + 7*3 + 6*2 = 207

O primeiro digito sera: (10*207) % 11 = 2 (se o resto for 10 → o primeiro digito sera 0)

O segundo digito e' calculado de forma similar, mas agora comecando do 11 e considerando os 10 primeiros numeros (9 originais + o primeiro digito verificador calculado acima). Assim, o segundo digito verificador do CPF 123555876 sera:

S = 1*11 + 2 *10 + 3*9 + 5*8 + 5*7 + 5*6 + 8*5 + 7*4 + 6*3 + 2*2 = 253

O segundo digito sera: (253*10)%11 = 0

Crie um programa (com nome **geraDigitosCPF.cpp**) que le um numero N e, entao, N strings (cada uma representando um CPF composto de 9 digitos). Seu programa devera, entao, imprimir N linhas contendo, em cada linha, os dois digitos verificadores do CPF correspondente.

Entrada	Saida esperada
4 123555876 123456789 111111111 078123587	20 09 11 12

Etapa 2

Considere a classe MyHashMap (similar a vista em sala).

Termine a implementacao do método set e teste sua implementacao usando o programa testaMyHashMapString.cpp.

Etapa 3

Considere o programa testaMyHashMapTamanhoBaldes.cpp.

Compile-o e execute-o redirecionando o arquivo Frankenstein.txt para sua entrada padrao (i.e., vamos guardar as palavras do livro Frankenstein em uma tabela hash e ver a uniformidade com que nossa funcao hash distribui as palavras na tabela).

Para executar, digite “time ./a.out 1 < Frankenstein.txt” (o numero 1 indica que usaremos a primeira implementacao da funcao hash ; o comando time sera utilizado para medir o tempo de execucao do programa)

A seguir, implemente pelo menos mais 4 funcoes hash distintas (use um numero distinto para identificar cada uma) e veja como elas se comportam (avalie o tempo de execucao do programa usando o programa time e verifique tambem a distribuicao das chaves nos baldes). Tente criar tanto funcoes hash boas quanto ruins.

Finalmente, mude o tamanho padrao da tabela hash para 1000 e meca os tempos (para cada funcao hash) novamente.

Anote os tempos e suas conclusoes em uma folha de papel (ou crie um arquivo para isso).

Discuta suas conclusoes e as vantagens/desvantagens de cada funcao hash com o professor. (esta etapa não sera avaliada de forma automatica pelo Submitty)

Etapa 4

Uma importante operacao em tabelas hash e' a operacao de redimensionar a tabela. A ideia e' que se os baldes ficarem muito cheios a tabela podera ser recriada com um tamanho (numero de baldes) maior (o objetivo e' que, com uma tabela maior, espera-se que cada balde tenha menos elementos).

A operacao de rehashing pode ser executada de forma automatica (ou seja, a tabela hash chama a operacao de rehashing quando detectar, por exemplo, que os baldes estao "cheios demais") ou manual (nesse caso o usuario devera escolher manualmente o novo tamanho da tabela).

Implemente uma funcao "reHash(int newSize)" que, dado um novo tamanho para a tabela hash, a redimensiona para o novo tamanho (esse tamanho podera ser menor, maior ou igual ao atual!).

Note que, ao redimensionar uma tabela, todas as chaves devem ser reinseridas nos novos baldes corretos.

Por exemplo, suponha que usamos uma funcao hash para strings que retorna 0 para strings que comecam com a letra a, 1 para strings que comecam com b, 2 para strings que comecam com c e assim por diante. Em uma tabela hash de tamanho 4 as palavras "abc" e "elefante" seriam colocadas no balde 0. Porem, se a tabela fosse redimensionada para tamanho 10, entao "abc" iria para o balde 0 e "elefante" iria para o balde 4.

Submissao da aula pratica:

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Envie todos os arquivos fonte (tanto os arquivos .h e .cpp fornecidos neste laboratorio quanto os que você implementou). Atualmente a submissao so pode ser realizada dentro da rede da UFV.