

# Semana 6: Conceitos básicos sobre *Hashing* e Tratamento de Colisões por Encadeamento Direto

Prof. Dr. Juliano Henrique Foleis

Estude com atenção os vídeos e as leituras sugeridas abaixo. Os exercícios servem para ajudar na fixação do conteúdo e foram escolhidos para complementar o material básico apresentado nos vídeos e nas leituras. Quando o exercício pede que crie ou modifique algum algoritmo, sugiro que implemente-o em linguagem C para ver funcionando na prática. O único exercício que é necessário entregar está descrito na Seção “Atividade Para Entregar”.

## Vídeos

[Conceitos Básicos sobre Hashing](#)

[Tratamento de Colisões por Encadeamento Direto](#)

## Leitura Sugerida

FEOFILOFF, Paulo. Estruturas de Dados. *Hashing* - **Seções:** Idéias Preliminares, Funções de Hashing, Função de Hashing Modular, O que se espera de uma função de hashing ideal, Implementação 1: hashing com encadeamento. ([Link](#))

## Exercícios

### Exercícios dos materiais de leitura sugerida

Exercícios 1.1, 1.2, 2.1, 3.4, 4.1, 4.2, 4.6, 4.7 da página do Prof. Feofiloff (Hashing): ([Link](#))

### Exercícios Complementares

**1. *Hashing de Strings*** – Uma função de hashing popularmente utilizada em strings baseia-se em converter a representação em ASCII da string em um inteiro em uma base  $K$  qualquer, seguido por módulo  $M$ . Por exemplo, a string “ABC” (65, 66, 67) pode ser representada na base  $K=31$  por:  $65 \times 31^2 + 66 \times 31^1 + 67 \times 31^0 = 65 \times 961 + 66 \times 31 + 67 \times 1 = 64578$ . Com  $M=37$ ,  $H(ABC; 37) = 64578 \% 37 = 13$ . Lembrando que  $K$  e  $M$  são parâmetros e podem ser escolhidos de acordo com a aplicação.

**a.** Implemente a função de hashing sugerida acima. Use o protótipo: `unsigned int string_hash(char* string, int M)`. Considere que a string é terminada em ‘\0’ (*null-terminated*).

**b.** Uma forma eficiente de implementar a função de hashing sugerida acima é utilizando o Método de Horner. Um exemplo do método:

$$2x^3 + 3x^2 + 4x + 5 = ((2x + 3)x + 4)x + 5$$

Implemente a função `unsigned int string_hash(char* string, int M)` usando o método de horner.

c. Seja  $n$  o comprimento da string. Encontre duas funções (matemáticas)  $f_a(n)$  e  $f_b(n)$  que calcule o número de somas e multiplicações realizadas pelos métodos implementados nos exercícios **a** e **b**. Compare os  $f_a(n)$  e  $f_b(n)$  para  $n = 5, 10, 20, 100$ .

2. Clone (ou atualize!) o repositório da disciplina no [github](#). A implementação da tabela hashing com encadeamento direto está nos arquivos *hashing/hashtable\_ed.c* e *hashing/hashtable\_ed.h*.

a. Implemente as funções em branco conforme mostrado no [vídeo](#).

b. Implemente as funções a seguir no arquivo *hashing/hashtable\_ed.c*. Estas funções retornam as **chaves** mínima e máxima que estão armazenadas na tabela hash. Não esqueça de acrescentar os protótipos em *hashing/hashtable\_ed.h*!

```
i. int min(HTED* HT);  
ii. int max(HTED* HT);
```

c. Modifique a estrutura da tabela hash em *hashing/hashtable\_ed.h* para que as funções *min* e *max* sejam executadas em  $O(1)$ , ou seja, de forma que não seja necessário varrer toda a estrutura para encontrar os respectivos valores. Modifique todas as funções que sejam necessárias.

**DICA:** Basta atualizar variáveis na estrutura que indicam o valor mínimo ou máximo quando houver alguma mudança na estrutura, seja essa inserção, remoção ou atualização.

## Atividade para Entregar

A atividade a seguir é para ser feita individualmente e entregue via Moodle no tópico da Semana 6. A data-limite para entrega é dia 16/9/2020 às 23:55. Em caso de cópia as atividades dos participantes serão desconsideradas.

### Descrição da Atividade

Nesta atividade vamos avaliar a diferença entre usar números primos e números compostos como  $M$  na função de hashing modular. Esta avaliação é apenas um teste empírico bem simples, então não é possível tirar conclusões muito abrangentes. No entanto, é um bom exercício para ter uma idéia de uma situação prática.

a. Faça uma função *THED\* inserir\_n\_aleatorios(int n, int m, int seed)* que crie uma tabela hash com encadeamento direto com  $m$  posições e insira  $n$  elementos aleatórios com a semente *seed*. Os números aleatórios devem estar no intervalo  $[0 \dots 10n]$ . Sua função deve retornar a tabela preenchida. A implementação *ilist* não permite chaves repetidas na lista, portanto não se preocupe com esse caso.

b. Implemente as funções a seguir que retornam o comprimento da lista mais curta, o comprimento da lista mais longa e o comprimento médio das listas em uma tabela hash *TH*:

```
i. int THED_MaisLonga(THED* TH);  
ii. int THED_MaisCurta(THED* TH);  
iii. float THED_TamMedio(THED* TH);
```

c. Preencha as Tabelas 1, 2 e 3 a seguir usando as funções implementadas nos itens **a** e **b**. Use *seed* = 42.

d. Preencha a Tabela 4 com o valor teórico do comprimento das listas ( $N/M$ ) em um caso de hashing uniforme.

e. Nos casos avaliados houve alguma diferença significativa no comprimentos das listas mais longas e mais curtas entre  $M$  primo e  $M$  composto? Se sim, em qual(is) caso(s)?

f. Nos casos avaliados houve alguma diferença significativa no comprimento médio das listas entre  $M$  primo e  $M$  composto? Se sim, em qual(is) caso(s)?

g. O comprimento médio das listas (Tabela 3) é comparável ao comprimento esperado diante da hipótese de hashing uniforme (Tabela 4)? O que isso quer dizer?

		n = 1000	n = 10000	n = 50000	n = 100000
M Composto	m = 10				
	m = 100	X			
	m = 500	X	X		
	m = 1000	X	X	X	
M Primo	m = 7				
	m = 97	X			
	m = 499	X	X		
	m = 997	X	X	X	

Figure 1: Comprimento da Lista Mais Longa

		n = 1000	n = 10000	n = 50000	n = 100000
M Composto	m = 10				
	m = 100	X			
	m = 500	X	X		
	m = 1000	X	X	X	
M Primo	m = 7				
	m = 97	X			
	m = 499	X	X		
	m = 997	X	X	X	

Figure 2: Comprimento da Lista Mais Curta

		n = 1000	n = 10000	n = 50000	n = 100000
M Composto	m = 10				
	m = 100	X			
	m = 500	X	X		
	m = 1000	X	X	X	
M Primo	m = 7				
	m = 97	X			
	m = 499	X	X		
	m = 997	X	X	X	

Figure 3: Comprimento Médio das Listas

		n = 1000	n = 10000	n = 50000	n = 100000
M Composto	m = 10				
	m = 100	X			
	m = 500	X	X		
	m = 1000	X	X	X	
M Primo	m = 7				
	m = 97	X			
	m = 499	X	X		
	m = 997	X	X	X	

Figure 4: Comprimento Esperado Diante da Hipótese de Hashing Uniforme

## Você deve Entregar

Entregue em formato .zip os arquivos a seguir:

- Os arquivos-fonte desenvolvidos nos itens **a–b**, bem como os arquivos-fonte criados para realizar os testes. Faça um *Makefile* para compilar o seu programa. Modularize conforme julgar necessário.
- As Tabelas preenchidas nos itens **c e d** e as respostas das perguntas dos itens **e–g** em um *pdf*.

**Por favor entregue como especificado acima!**

**A data-limite para entrega é dia 13/4/2021 às 23:55.**

**BONS ESTUDOS!**