

---

## Trabalho Prático 4 – TAD HASH

Duração: até 2 semanas – 10 Pontos

---

### 1 Objetivos

O objetivo desse trabalho é rever conceitos básicos de programação bem como explorar os conceitos de Tipos Abstratos de Dados (TADs), em especial, tabelas Hash.

#### 1.1 Funções Hash

Reveja o conteúdo sobre tabelas hash em: Sedgewick. Cap. 3.4 e Cormen et. al. Cap. 11.  
Breve resumo:

- Funções hash transformam chaves (*keys*) em índices de um arranjo (*Hash table*).
- São propriedades desejáveis:
  - Consistência: chaves iguais devem produzir o mesmo valor de *hash*;
  - Eficiência: computar a função *hash* não deve ser custoso;
  - **Distribuição uniforme:** cada chave deve ser/ter independente e igualmente provável de ser alocada em qualquer um dos  $m$  *slot* da tabela *hash*.

### 2 Introdução

Você irá implementar e analisar uma tabela *hash* com sonda linear (*linear probing*). Nesta implementação, a ideia é armazenar  $N$  pares de chave-valor em uma tabela *hash* de tamanho  $M > N$ . Com isso, as entradas vazias da tabela serão usadas para ajudar na resolução de conflitos. Tal método é conhecido como *open-addressing hashing*.

O método mais simples *open-addressing*<sup>1</sup> é o chamado *linear probing*: quando existe uma colisão (quando duas ou mais chaves são atribuídas ao mesmo índice pela `hash()`), então procuramos pela próxima entrada livre na tabela (fazemos isso ao incrementar o índice resultante da *hash*). São três os possíveis resultados desse método:

- 1.) A chave (*key*) é igual ao valor buscado: *search hit*;
- 2.) Posição vazia (um valor `null` é encontrado no índice): *search miss*;
- 3.) A chave (*key*) não é igual a chave retornada na posição indicada pela *hash*: tentamos a próxima entrada.

---

<sup>1</sup>Passagens do livro do Sedgewick. Cap. 3.4

key	hash	value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	6	0						S										
E	10	1						S					E					
A	4	2					A	S					E					
R	14	3					A	S					E				R	
C	5	4					A	C	S				E				R	
H	4	5					A	C	S	H			E				R	
E	10	6					A	C	S	H			E				R	
X	15	7					A	C	S	H			E				R	X
A	4	8					A	C	S	H			E				R	X
M	1	9		M			A	C	S	H			E				R	X
P	14	10	P	M			A	C	S	H			E				R	X
L	6	11	P	M			A	C	S	H	L		E				R	X
E	10	12	P	M			A	C	S	H	L		E				R	X

Figura 1: Exemplo de uma *hash* com *linear probing*.

## 2.1 Clustering

A Fig. 1 apresenta um tabela *hash* com *linear probing*. Perceba a as chaves (*keys*), índices resultantes da `hash()`, e valores associados à chave (*values*) no lado esquerdo da figura. Ao usar o método *linear probing* verifica-se que existem agrupamentos (*clustering*) de chaves inseridas na *hash* (ex: a inserção da chave C resulta em um agrupamento de comprimento 3 na Fig.1). Isso significa que 4 sondagens (*probes*) são necessárias para inserir H, pois a função `hash(H)` indica a primeira posição deste agrupamento para inserir H.

Esses agrupamentos estão diretamente relacionados ao custo médio da busca por uma chave. Grupos curtos são certamente um requisito para melhor desempenho. Este requisito pode se tornar problemático ao passo que a tabela é preenchida, pois grupos longos são mais comuns. Além disso, todas as entradas da tabela são igualmente prováveis de serem escolhidas (caso a 3ª propriedade desejável de func. *hash* seja atendida). Também grupos mais longos são mais prováveis de crescer do que grupos pequenos, pois uma nova chave atribuída a um grupo causa a esse grupo o acréscimo de 1 em seu comprimento. A Fig. 2 apresenta visualmente os custers.

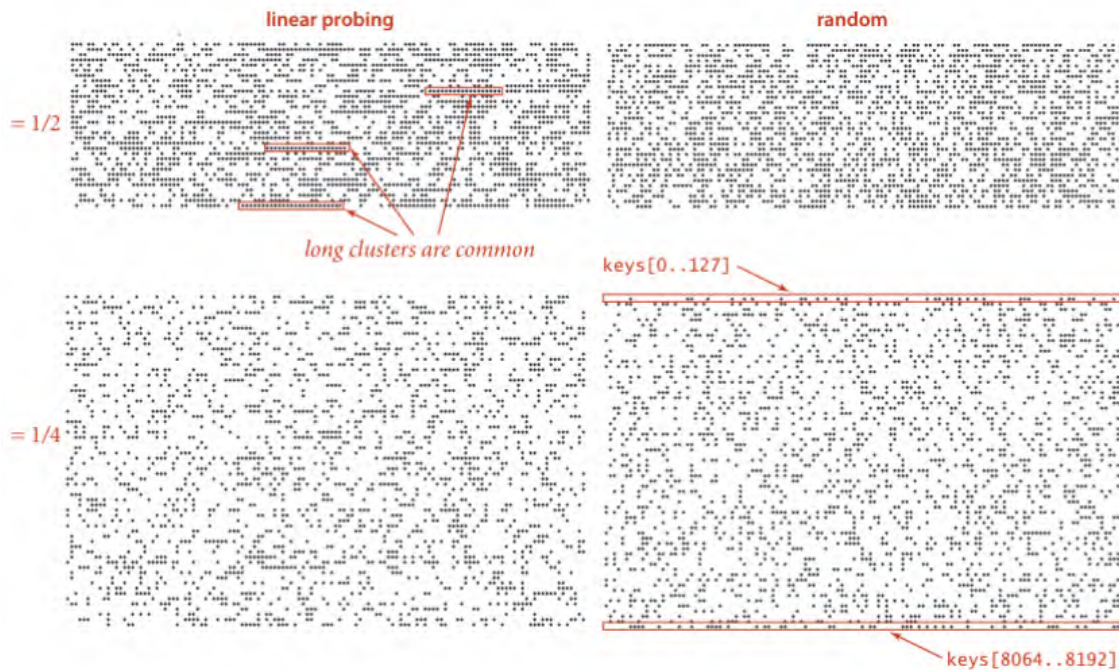


Figura 2: Agrupamentos ao usar *linear probing*

## 2.2 Análise do *linear probing*

Uma análise precisa do *linear probing* é uma tarefa bastante desafiadora. Abaixo são apresentadas duas fórmulas que foram derivadas por Knuth em 1962 ao usar técnicas de análise de algoritmos, matemática e estatística.

Em uma *hash table* com *linear probing* de tamanho  $M$  e  $N = \alpha \cdot M$  chaves, o número médio<sup>2</sup> de sondas (probings) requeridas são:

$$\sim \frac{1}{2} \left( 1 + \frac{1}{1 - \alpha} \right) \quad (1)$$

$$\sim \frac{1}{2} \left( 1 + \frac{1}{(1 - \alpha)^2} \right) \quad (2)$$

para *search hits* e *search misses/inserts* respectivamente.

## 3 Descrição

Seu trabalho está dividido em duas partes:

1.) Analisar *search misses*.

2.) Implementar a função `delete(key)` para uma *hash table* com *linear probing*;

<sup>2</sup>Assumindo que as chaves são distribuídas uniformemente

### 3.1 Custo médio: *search misses*

Você terá que escrever um programa que insere  $\frac{N}{2}$  chaves aleatórias em uma tabela *hash* de tamanho  $N$  usando *linear probing*<sup>3</sup>. Então deverá computar o custo médio de uma *search miss* (isto é, avaliar o comprimento médio dos grupos). Faça isso para  $N = 10^3, 10^4, 10^5, 10^6$ . Discuta até que ponto seus resultados validam as equações 1 e 2.

- No seu relatório, você deve justificar suas decisões, discutir os resultados encontrados e apresentar conclusões.
- Você poderá fornecer plotar gráficos e intervalos de confiança para as estimativas.

#### 3.1.1 Também responda:

- Se  $M$  (tamanho da tabela *hash*) é muito grande. O que acontece?
- Se  $M$  (tamanho da tabela *hash*) é muito pequena. O que acontece?
- Qual é o  $\alpha = \frac{N}{M} = ?$  típico? Por quê?

### 3.2 Deletar(key)

Como nós deletamos um par chave-valor de uma *hash table with linear probing*?

Você deverá implementar essa funcionalidade. Note que não basta adicionar um valor NULL na tabela. Suponha que tentarmos deletar a chave **C** da tabela apresentada na Fig. 1 usando essa estratégia. Logo após, tentamos buscar por **H**. A função `hash(H) = 4`, mas **H** está no final do grupo. Ao atribuir NULL na pos. 5 (ao deletar **C**) então não conseguiríamos encontrar **H**.

## 4 O que deve ser entregue?

- Código fonte do programa em C (bem indentada e comentada).
- Documentação do trabalho seguindo o padrão da SBC ([Template SBC](#)). **LIMITE SUA DOCUMENTAÇÃO A NO MÁXIMO 3 PÁGINAS**. Entre outras coisas, a documentação poderá conter:
  - 1.) **Introdução:** descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  - 2.) **Implementação:** descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  - 3.) **Estudo de complexidade:** estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação  $O$ ).

<sup>3</sup>Sim. O  $N$  aqui está sendo usado como tamanho do tabela *hash*.

- 4.) **Listagem de testes executados:** os testes executados devem ser simplesmente apresentados.
- 5.) **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
- 6.) **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
- 7.) **Formato:** obrigatoriamente em **PDF**

## 5 Dicas

- Consulte as dicas do Prof. Nívio Ziviani de como deve ser feita uma boa implementação e documentação de um trabalho prático ([LINK](#)).
- [Clique aqui e veja uma ótima referência para estudos de C e TAD.](#)
- [Linear probing.](#)
- [Hash with linear probing.](#)
- [Números aleatórios em C.](#)
- [Intervalo de confiança.](#)

## 6 Como deve ser feita a entrega

A entrega DEVE ser feita pelo Moodle na forma de um único arquivo zipado, contendo o código, os arquivos e a documentação. Também deve ser entregue a documentação impressa (em frente/verso) na próxima aula após a data de entrega do trabalho.

## 7 Comentários gerais:

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- A qualidade da solução/código também faz parte da avaliação;
- Clareza, indentação e comentários no programa também vão valer pontos;
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados (documentação e/ou código fonte) terão nota zero;
- Trabalhos entregue em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Códigos que não compilares serão penalizados.
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.

## 8 Código de suporte

O professor disponibilizará um código de suporte. O código apresenta comentários.