Tabela Hash

Prof. Marcelo Rosa

Algoritmos e Estrutura de Dados 2 (AE43CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

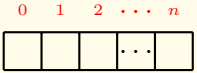
UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

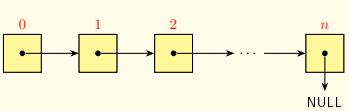
Sumário

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

A Abordagem Ingênua: Usando uma Lista ou Vetor





- Busca: $O(n)$

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

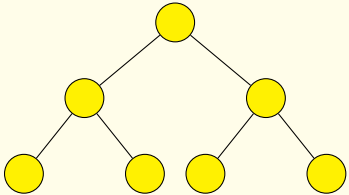
Vetor ordenado

- Utilizar um vetor ordenado
- Busca: $O(\log n)$
- Custo da inserção e remoção $O(n)$

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

Árvore binária



- Todas as operações têm tempo $O(\log n)$

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

Endereçamento direto

- Criar um vetor suficientemente grande V para que ele seja indexado pelos CPFs ($V[cpf]$)
- Todas as operações têm tempo $\Theta(1)$
- Porém, temos 10^{11} possíveis número possíveis

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

- É possível manter as operações com tempo constante ($\Theta(1)$) e ao mesmo tempo reduzir a complexidade de espaço?

Sistema de gerenciamento de funcionários

- Suponha que queremos projetar um sistema que armazena os dados de n funcionários usando como chaves seus CPFs (11 dígitos)
- Esse sistema deve dar suporte as operações: inserção, remoção e **busca**

- É possível manter as operações com tempo constante ($\Theta(1)$) e ao mesmo tempo reduzir a complexidade de espaço?
- **Tabelas hash** (*hash tables*).

Sumário

Tabela *Hash*

- As tabelas *hash* (tabelas de espalhamento, tabelas de dispersão) são uma solução para o problema mencionado

Tabela hash

Uma tabela *hash* associa chaves e valores:

- Chave: uma parte da informação que compõe o item a ser inserido ou buscado
- Valor: posição onde o item deve estar no vetor

Tabela *Hash*

- As tabelas *hash* (tabelas de espalhamento, tabelas de dispersão) são uma solução para o problema mencionado

Tabela hash

Uma tabela *hash* associa chaves e valores:

- Chave: uma parte da informação que compõe o item a ser inserido ou buscado
- Valor: posição onde o item deve estar no vetor

Exemplo

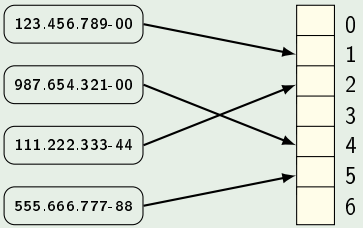


Tabela *Hash*

- As tabelas *hash* (tabelas de espalhamento, tabelas de dispersão) são uma solução para o problema mencionado

Tabela hash

Uma tabela *hash* associa chaves e valores:

- Chave: uma parte da informação que compõe o item a ser inserido ou buscado
- Valor: posição onde o item deve estar no vetor

Aplicações da tabela *hash*

banco de dados, tabela de símbolos, redes de computadores, criptografia, etc.

Sumário

Função *hash*

A tabela *hash* usa uma função[a] *hash* $h : U \mapsto \{0, 1, \ldots, m-1\}$, onde:

- A entrada é uma chave k pertencente ao conjunto de chaves U
- A saída é a posição (endereço) onde o elemento associado a chave k deve ser inserido
- $|U| \gg m$, ou seja, o número de chaves em U é muito maior do que m.

[a] *Hashing* é o processo de mapear uma chave para um índice.

Tabela *Hash*

Função *hash*

A tabela *hash* usa uma função[a] *hash* $h : U \mapsto \{0, 1, \ldots, m-1\}$, onde:

- A entrada é uma chave k pertencente ao conjunto de chaves U
- A saída é a posição (endereço) onde o elemento associado a chave k deve ser inserido
- $|U| \gg m$, ou seja, o número de chaves em U é muito maior do que m.

[a] *Hashing* é o processo de mapear uma chave para um índice.

Observações

- Um elemento x com uma chave k é **mapeado para a posição** $h(k)$, ou seja, $\mathbf{v}[h(k)] = x$ com $\mathbf{v}$ sendo um vetor de m posições.
- $h(k)$ é **valor hash** da chave k
- Com essa função, os dados podem não ser inseridos de forma ordenada
  - Por isso, o processo de aplicação de *hashing* é conhecido "como espalhamento"

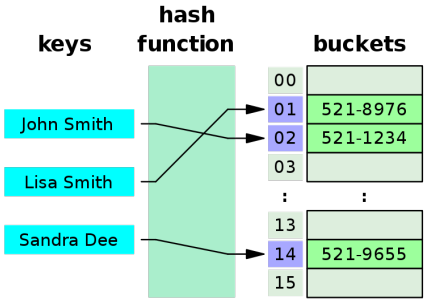
Ideia geral da função *hash*

Particionar um conjunto de elementos (possivelmente infinito) em um número finito de classes:

- m classes (endereços), de 0 a $m - 1$
- Essas classes são chamadas de `buckets`

Tabela *Hash*

- Conceitos relacionados:
  - $h(k)$ (função *hash*) retorna o valor *hash* de k
  - k pertence ao *bucket* $h(k)$

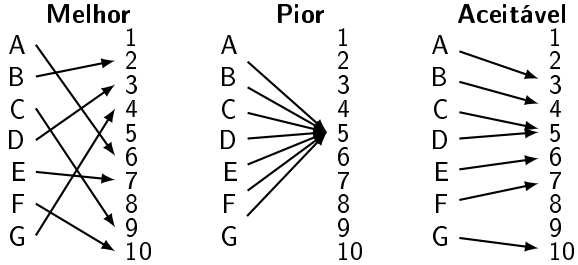


Fonte da figura: `https://commons.wikimedia.org/w/index.php?curid=6471238`

Sumário

Colisão

- Uma situação comum em tabelas *hash* é a **colisão**, que ocorre quando duas chaves diferentes são mapeadas para a mesma posição.
- Evitar colisões é impossível, pois $|U| > m$ implica que pelo menos duas chaves devem ter o mesmo valor *hash*.
- Uma função *hash* bem projetada, que pareça "aleatória", pode minimizar o número de colisões, mas ainda é necessário um método para resolvê-las



Sumário

Funções *hash*: Projeto

- Uma boa função *hash* se aproxima da suposição de **hashing uniforme simples**, onde cada chave tem igual probabilidade de ser mapeada para qualquer uma das m posições, independentemente das outras chaves

Funções *hash*: Projeto

- Uma boa função *hash* se aproxima da suposição de **hashing uniforme simples**, onde cada chave tem igual probabilidade de ser mapeada para qualquer uma das m posições, independentemente das outras chaves

Observações

- Distribuição uniforme é muito difícil, pois depende de cálculos matemáticos e estatísticos complexos
- Existe chance de alguns endereços serem gerados mais de uma vez e de outros nunca serem gerados

"Segredos" para um bom *hashing*

- Escolher uma boa função *hash* (em função dos dados)
  - Distribui uniformemente os dados, na medida do possível
  - Evita colisões
  - Fácil implementação
  - Rápida ao ser computada
- Estabelecer uma boa estratégia para tratamento de colisões

Sumário

Sumário

Método da divisão (resto)

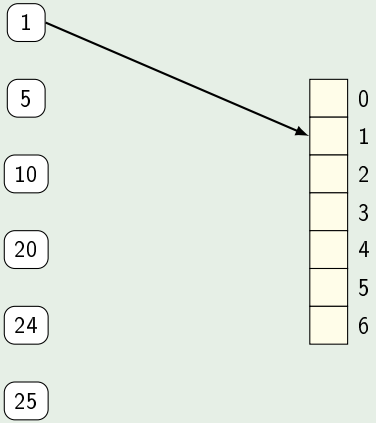
- A posição da chave k na tabela é dada pelo **resto** da divisão entre k pelo tamanho da tabela.
- Formalmente, a chave k é mapeada para uma das m posições pegando o resto de k dividido por m:

$$h(k) = k \mod m$$

- Simples e comumente utilizada, pois geralmente produz bons resultados

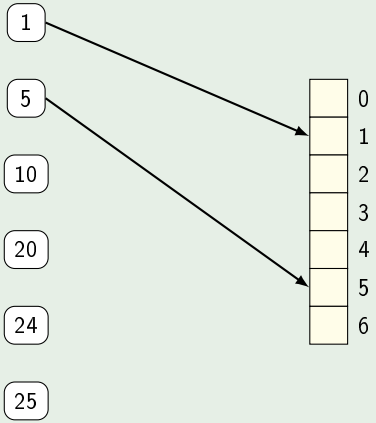
Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



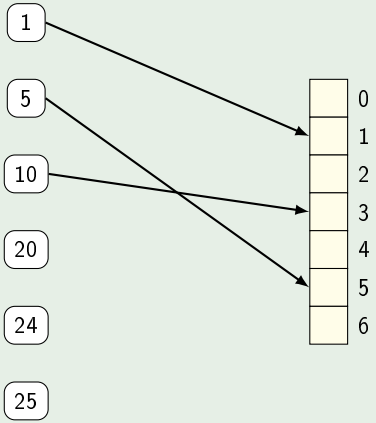
Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



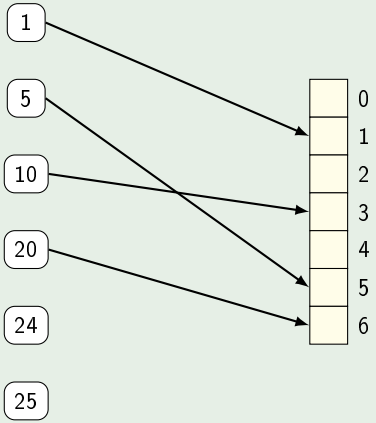
Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



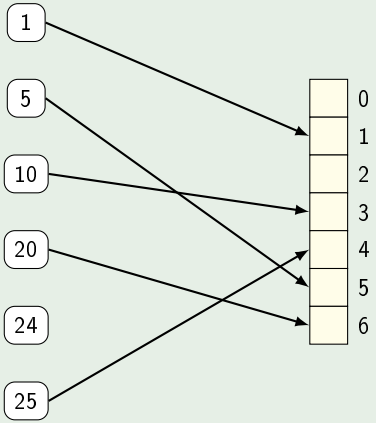
Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



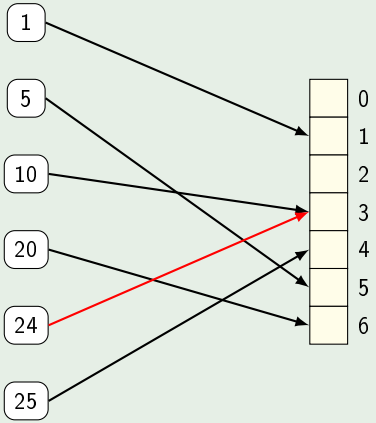
Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



Exemplo: Método da divisão - $h(k) = k \mod m$

- Seja T uma tabela (vetor) de 7 elementos e deseja-se a inserção das chaves 1, 5, 10, 20, 25, 24



Método da divisão: sugestões

- Geralmente, m não deve ser uma potência de 2, pois $h(k)$ seria apenas os p bits de ordem mais baixa de k se $m = 2^p$
  - Por exemplo, para $k = 45$ e $m = 8 = 2^3$ temos que 45 em binário 101101, resulta em $45 \mod 8 = 5$ ou 101 em binário
- Um número primo não muito próximo de uma potência exata de 2 é uma boa escolha para m
- Para chaves do tipo *string*, tratar cada caractere como um valor inteiro (ASCII), somá-los e obter o resto da divisão por m

Sumário

Método da Multiplicação

- Pode ser descritos conforme as seguintes etapas:
  1. A chave k é multiplicada por uma constate A, com $0 < A < 1$
  2. A parte fracionária de kA é selecionada
  3. A parte fracionária de kA é multiplicada por m
  4. Então a parte inteira dessa última multiplicação é usada como posição.
- Formalmente, esse processo de *hashing* pode ser descrito pela função

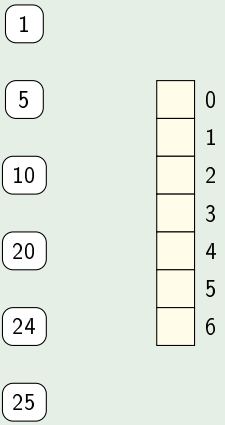
$$h(k) = \lfloor m\,(kA \mod 1) \rfloor$$

  com $(kA \mod 1)$ correspondendo a parte fracionária de kA, ou seja, $kA = \lfloor kA \rfloor$
- [1] sugere que $A \approx (\sqrt{5} - 1)/2 = 0,6180339887\ldots$

Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

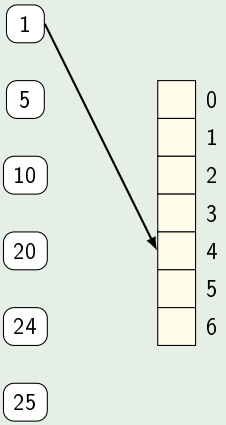
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

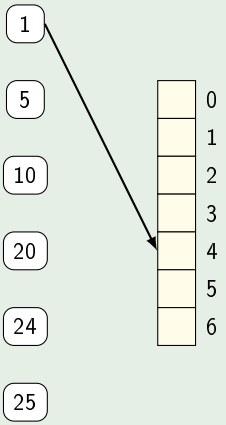
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

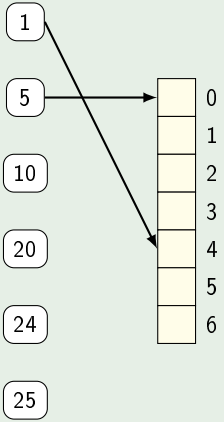
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

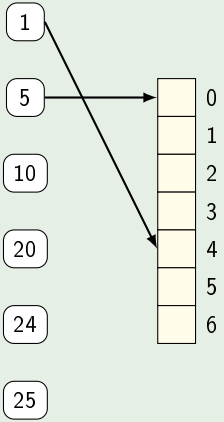
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

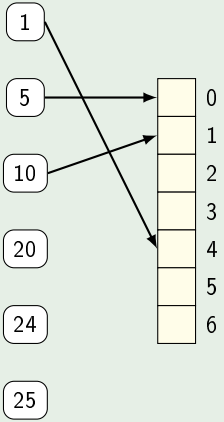
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

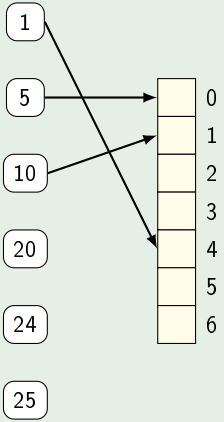
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

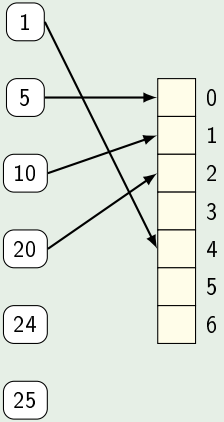
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

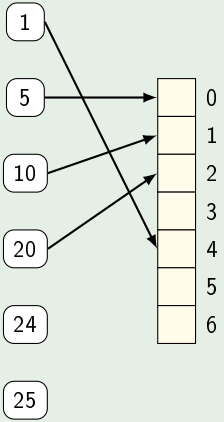
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

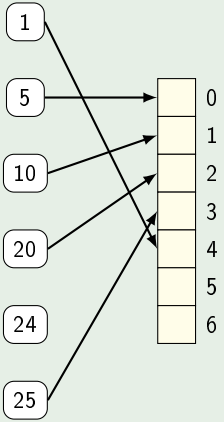
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$
- $k = 25$: $25 \cdot 0,62 = 15,50 \Rightarrow h(25) = \lfloor 7 \cdot 0,50 \rfloor = \lfloor 3,50 \rfloor = 3$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

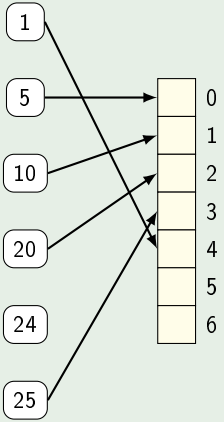
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$
- $k = 25$: $25 \cdot 0,62 = 15,50 \Rightarrow h(25) = \lfloor 7 \cdot 0,50 \rfloor = \lfloor 3,50 \rfloor = 3$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

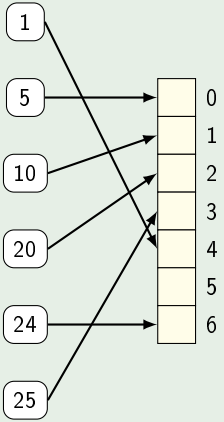
- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$
- $k = 25$: $25 \cdot 0,62 = 15,50 \Rightarrow h(25) = \lfloor 7 \cdot 0,50 \rfloor = \lfloor 3,50 \rfloor = 3$
- $k = 24$: $24 \cdot 0,62 = 14,88 \Rightarrow h(24) = \lfloor 7 \cdot 0,88 \rfloor = \lfloor 6,16 \rfloor = 6$



Exemplo: Método da multiplicação - $h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor$, com $A = 0,62$ e $m = 7$

- Seja T uma tabela com $m = 7$ posições (índices de 0 a 6), e deseje-se inserir as chaves: $1,\ 5,\ 10,\ 20,\ 25,\ 24$.

- $k = 1$: $1 \cdot 0,62 = 0,62 \Rightarrow h(1) = \lfloor 7 \cdot 0,62 \rfloor = \lfloor 4,34 \rfloor = 4$
- $k = 5$: $5 \cdot 0,62 = 3,10 \Rightarrow h(5) = \lfloor 7 \cdot 0,10 \rfloor = \lfloor 0,70 \rfloor = 0$
- $k = 10$: $10 \cdot 0,62 = 6,20 \Rightarrow h(10) = \lfloor 7 \cdot 0,20 \rfloor = \lfloor 1,40 \rfloor = 1$
- $k = 20$: $20 \cdot 0,62 = 12,40 \Rightarrow h(20) = \lfloor 7 \cdot 0,40 \rfloor = \lfloor 2,80 \rfloor = 2$
- $k = 25$: $25 \cdot 0,62 = 15,50 \Rightarrow h(25) = \lfloor 7 \cdot 0,50 \rfloor = \lfloor 3,50 \rfloor = 3$
- $k = 24$: $24 \cdot 0,62 = 14,88 \Rightarrow h(24) = \lfloor 7 \cdot 0,88 \rfloor = \lfloor 6,16 \rfloor = 6$



Sumário

Implementação

- Na implementação a ser realizada nessa aula não tratará colisões
- O código será aprimorada na próxima aula para o tratamento de colisões
- Estrutura de dados simples para tabela *hash*

```
typedef struct{
   int tamanho;
   int *buckets;
}HashTable;
```

Implementação

- TAD simples para a tabela *hash*:

```
HashTable* gerarHT(unsigned int tam);

int procurar_chave(unsigned int chave, HashTable* t);

int inserir_chave(unsigned int chave, HashTable* t);

int remover_chave(unsigned int kchave, HashTable* t);

void imprimir_tabela(HashTable* t);

int liberar_tabela(HashTable* t);
```

Implementação

- Código para arquivo .c

```
static int hash_code(int chave, int tam_tab){
   return chave % tam_tab;
}

HashTable* gerarHT(unsigned int tam){
   HashTable* t = malloc(sizeof(HashTable));
   int i;

   t->tamanho = tam;
   t->buckets = malloc(tam * sizeof(int));

   for (i = 0; i < tam; i++)
    t->buckets[i] = -1;

   return t;
}
```

Implementação

- Código para arquivo .c

```
int procurar_chave(unsigned int chave, HashTable* t){
   int hc;

   if (t != NULL){
    hc = hash_code(chave, t->tamanho);

    if (t->buckets[hc] == chave)
      return hc;
   }

   return -1;
}
```

Implementação

- Código para arquivo .c

```
int inserir_chave(unsigned int chave, HashTable* t){
   int hc;

   if (t != NULL){
    hc = hash_code(chave, t->tamanho);

    if (t->buckets[hc] < 0){
      t->buckets[hc] = chave;

      return 1;
    }

    printf("Houve colisao ao tentar inserir a chave %d!\n", chave);
   }

   return 0;
}
```

Implementação

- Código para arquivo .c

```
int remover_chave(unsigned int chave, HashTable* t){
   int pos = procurar_chave(chave, t);

   if (pos >= 0){
    t->buckets[pos] = -1;

    return 1;
   }else
    return 0;
}

void imprimir_tabela(HashTable* t){
   int i;

   if (t != NULL)
    for (i = 0; i < t->tamanho; i++)
      if (t->buckets[i] > -1)
        printf("%d:  %d\n", i, t->buckets[i]);
}
```

Implementação

- Código para arquivo .c

```
int liberar_tabela(HashTable* t){
   int i;

   if (t != NULL){
    free(t->buckets);
    free(t);

    return 1;
   }else
    return 0;
}
```

Referências I


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.


Oliva, J. T.
Tabela Hash. AE22CP – Algoritmos e Estrutura de Dados II.
Notas de Aula. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2024.