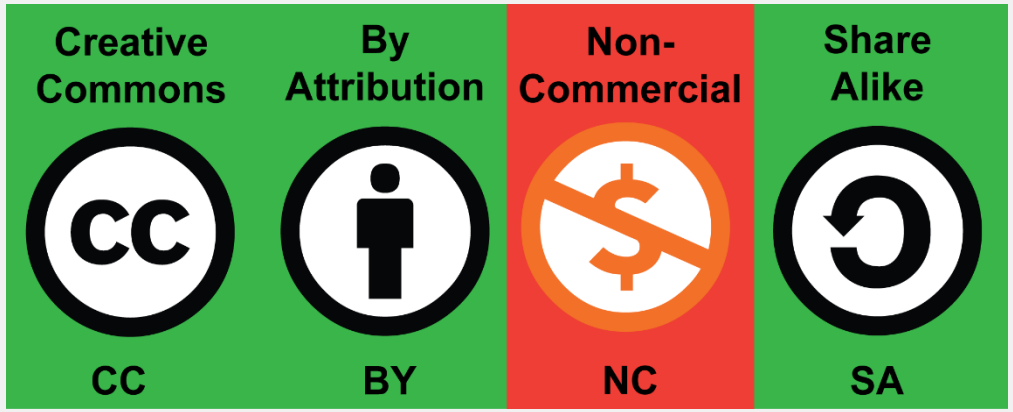
Tabela Hash

Prof. MSc. Jackson Antonio do Prado Lima
jacksonpradolima at gmail.com

Departamento de Sistemas de Informação – DSI

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Licença



UDESC

Histórico de Modificação

- Esta apresentação possui contribuição dos seguintes professores:
  - Alex Luiz de Souza (UDESC)
  - Anderson Fabiano Dums (UDESC)
  - Fernando José Muchalski (UDESC)
  - Silvio Luiz Bragatto Boss (UTFPR)
  - Jackson Antonio do Prado Lima (UDESC)

UDESC

Agenda

- Contextualização
- Introdução
  - Aplicações
  - Vantagens
  - Desvantagens
  - Limitações
  - Conceitos
  - Função Hashing
- Funções de Hashing
  - Divisão
  - Meio Quadrado
  - Folding
- Colisões
- Tratamento de Colisões
  - Hashing Aberto
  - Hashing Fechado
  - Hashing Múltiplo
  - Hashing Dinâmico

UDESC

CONTEXTUALIZAÇÃO

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Contextualização

- Dado um conjunto de pares (chave, valor)
  - Determinar se uma chave está no conjunto e o seu valor associado
  - Inserir um novo par no conjunto
  - Remover um par do conjunto
- Estruturas que podem ser usadas
  - Tabelas simples (vetores ou listas)
  - Árvores de busca
  - Tabelas de espelhamento (hash)

Contextualização

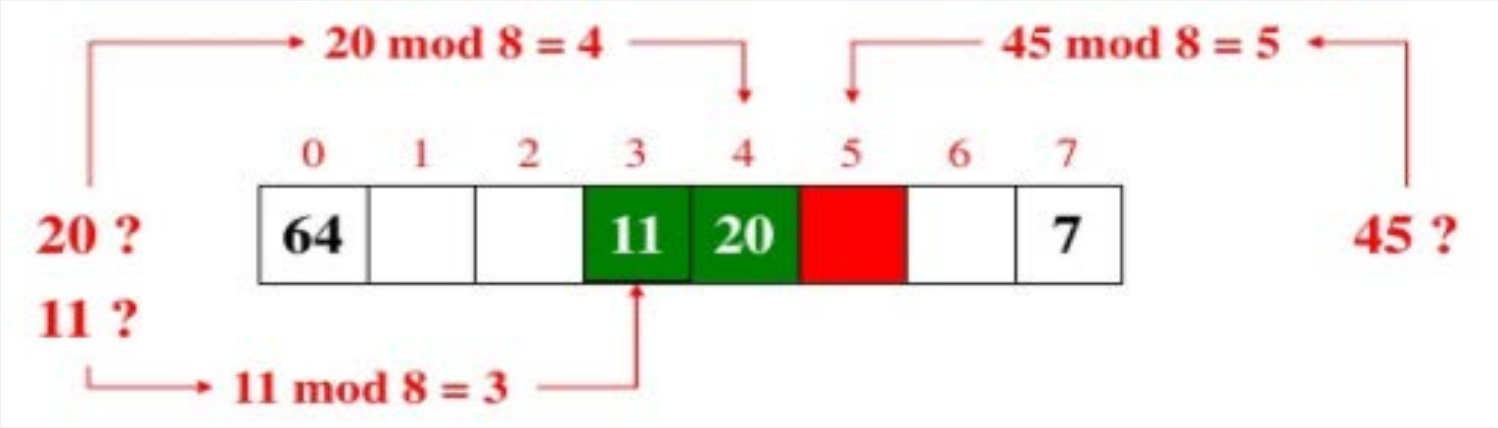
- Os métodos de pesquisa conhecidos buscam informações armazenadas com base na comparação de suas chaves
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito desta ordenação

Contextualização

- Conclusão:
  - Os algoritmos mais eficientes de busca conhecidos demandam esforço computacional **O(n)**, quando usamos uma tabela hash, esta pode realizar tais operações em tempo **O(1)**

Contextualização

- Em algumas aplicações, é necessário obter valor com poucas comparações, logo, é preciso saber a posição em que o elemento se encontra, sem precisar varrer todas as chaves
- A estrutura com esta propriedade é chamada de tabela hash



UDESC

INTRODUÇÃO

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Introdução

- É uma estrutura de dados que realiza o **mapeamento** entre **chaves (hash)** e **valores (dados)**
- Também conhecido como **tabela de espelhamento** ou **tabela de dispersão**
- Tem o objetivo de, a partir de uma chave **simples**, fazer uma busca **rápida** para obter um valor desejado
- Oferece **inserção** e busca muito **rápidas**, independente do **número** de itens

Introdução

- A ideia central consiste na divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.
- **Hashing** é uma técnica que utiliza uma função **h** para transformar uma chave **k** em um endereço
  - O endereço é usado para armazenar e recuperar registros

Introdução

- Tabelas hash são tipicamente utilizadas para implementar, por exemplo, vetores associativos, sets e realizar a indexação de grandes volumes de informação (como bases de dados)

Aplicações das Tabelas Hash

- As aplicações de tabelas de dispersão (***hash***) **incluem**, por exemplo:
  - **Banco de dados**;
  - Implementações das **tabelas de símbolos** dos **compiladores**;
  - Na **implementação** de **dicionários**;
  - Na **programação** de **jogos** para **acessar rapidamente** uma **posição** para onde o **personagem/avatar** irá **se mover**

Vantagens

- Para um usuário humano o acesso é instantâneo.
- A tabela pode receber mais itens mesmo quando um índice já foi ocupado
- Tabelas hash são muito mais rápidas que árvores e listas e são relativamente fáceis de se programar.

Desvantagens

- São baseadas em vetores, e vetores são difíceis de expandir depois de criados;
- Para alguns tipos de tabelas hash, o desempenho pode diminuir quando uma tabela torna-se muito cheia;
- Não há uma maneira conveniente de visitar os itens em uma tabela hash em qualquer tipo de ordem.
- Exige maior espaço para armazenamento das listas
- Listas longas implicam em muito tempo gato na busca
  - Se as listas estiverem ordenada, reduz-se o tempo de busca

Limitações

- As tabelas hash são **estrutura de dados** do **tipo** dicionário (ou conjunto), ou seja, **estruturas** que:
  - **Não permitem** armazenar elementos repetidos; ou
  - **Não permitem** recuperar elementos sequencialmente (ordenação); ou que
  - **Não permitem** recuperar o **elemento** antecessor e sucessor de outro **elemento**

Limitações

- Para **otimizar** a função de dispersão é **necessário** conhecer a **natureza** da chave a ser **utilizada**
  - No pior caso a **complexidade das operações** pode ser **O(N)**, ou seja, **caso** em que **todos** os elementos **inseridos** colidem
  - Hash com encadeamento interno (ou seja, com tratamento de colisões na própria tabela) pode **necessitar** de **redimensionamento** (a medida que a tabela aumenta)

Introdução

- São constituídas por dois conceitos fundamentais
  1. **Tabela de hash**: estrutura de dados que permite o acesso aos valores
  2. **Função de hashing**: função que realiza um mapeamento entre valores de chaves e entradas na tabela

Conceito

- O conceito de **hashing** envolve:
  - Guardar **valores** em uma tabela (vetor) indexado por inteiros
  - Mapear **chaves** em posições da tabela

- Ex1: Mapeamento Disciplina x Professor

Disciplina (chave)	Professor (valor)
DAD2	Jackson
PROG2	Tamanini
BAN	Vivian

Disciplina (chave)	Posição
DAD2	4
PROG2	0
BAN	1

	Professor
0	Tamanini
1	Vivian
2	
3	
4	Jackson

Conceito

- A questão é: “**Como mapear chaves em valores?**”
- A solução é usar uma **função** que calcule um valor **inteiro** a partir da chave

Função Hashing

- A função **h** é chamada de função hash **h(k)** retorna o valor hash de **k**
- Usado como endereço para armazenar a informação cuja chave é **k**
- **K** pertence a chave **h(k)**

Função Hashing

- A função hash é utilizada para inserir, remover ou buscar um determinado elemento
- A função hash deve ser determinística, ou seja, resultar sempre no mesmo valor para uma determinada chave

Função de Hashing

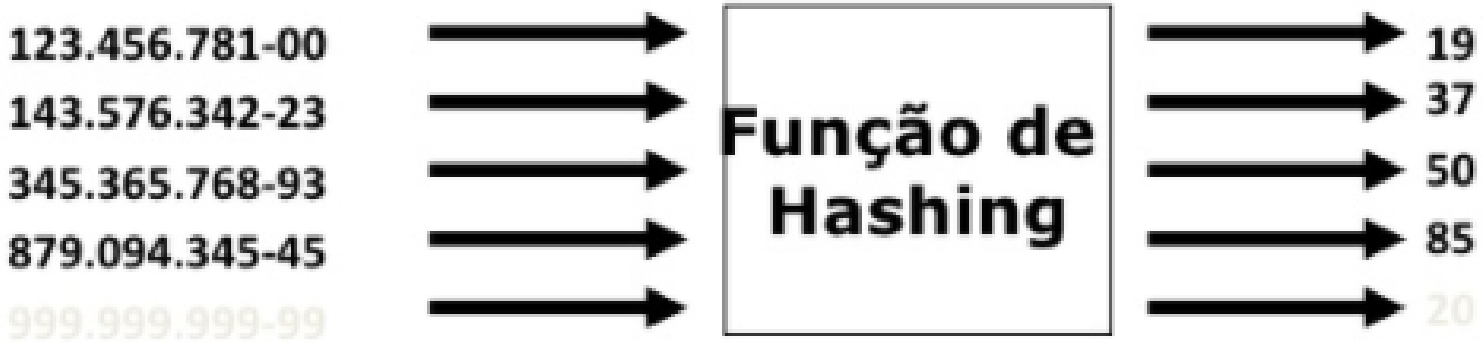
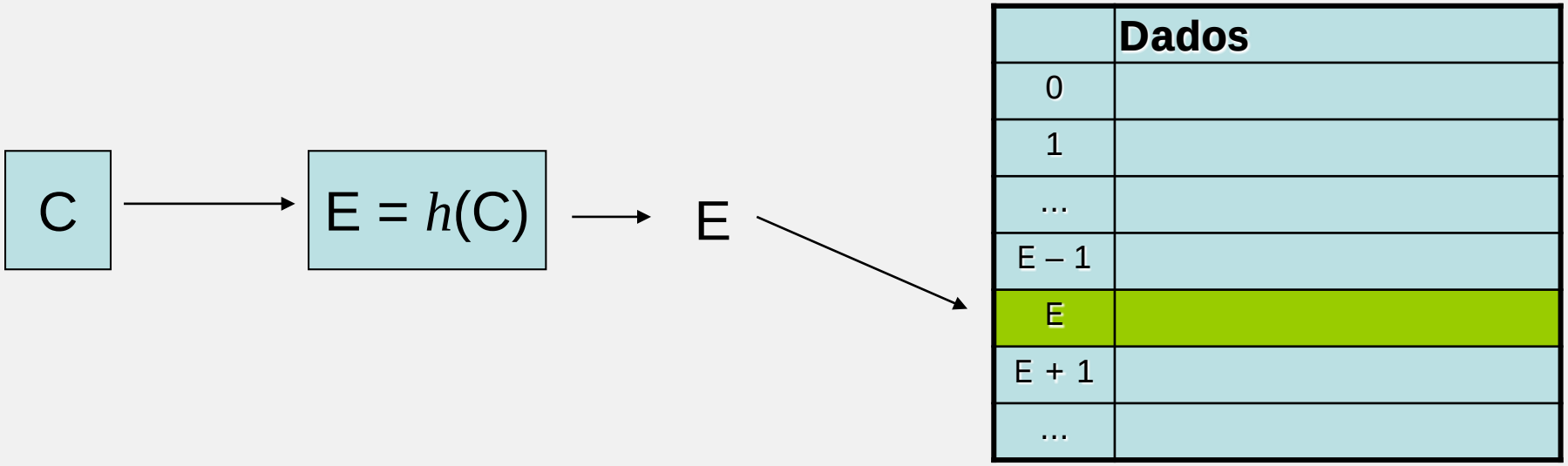


Tabela Hash

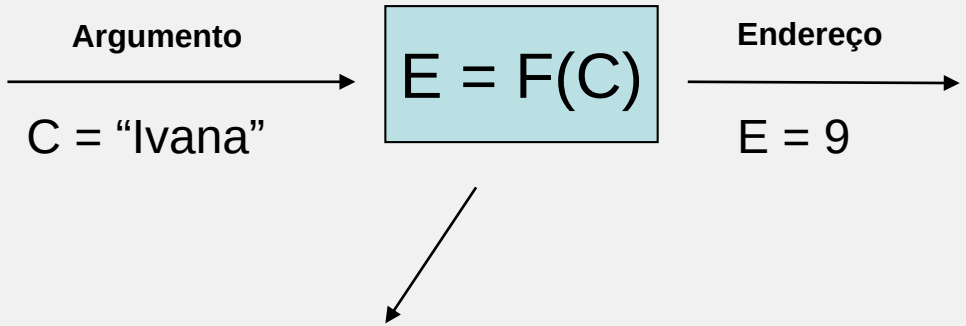
19	123.456.781-00; Fausto Silva; Av. Canal. Nº 45.
20	
...	
37	143.576.342-23; Carla Perez; Rua Celso Oliva. Nº 27.
...	
50	345.365.768-93; Gugu Liberato; Av. Atlântica. S/N.
...	
85	879.094.345-45 ; Hebe Camargo; Rua B. Nº 100.
...	

Função de Hashing

- Os registros armazenados em uma tabela (vetor) são **diretamente** endereçados a partir de uma **transformação aritmética** sobre a chave de pesquisa.



Exemplo



Função que calcula um índice a partir do valor da primeira letra do nome

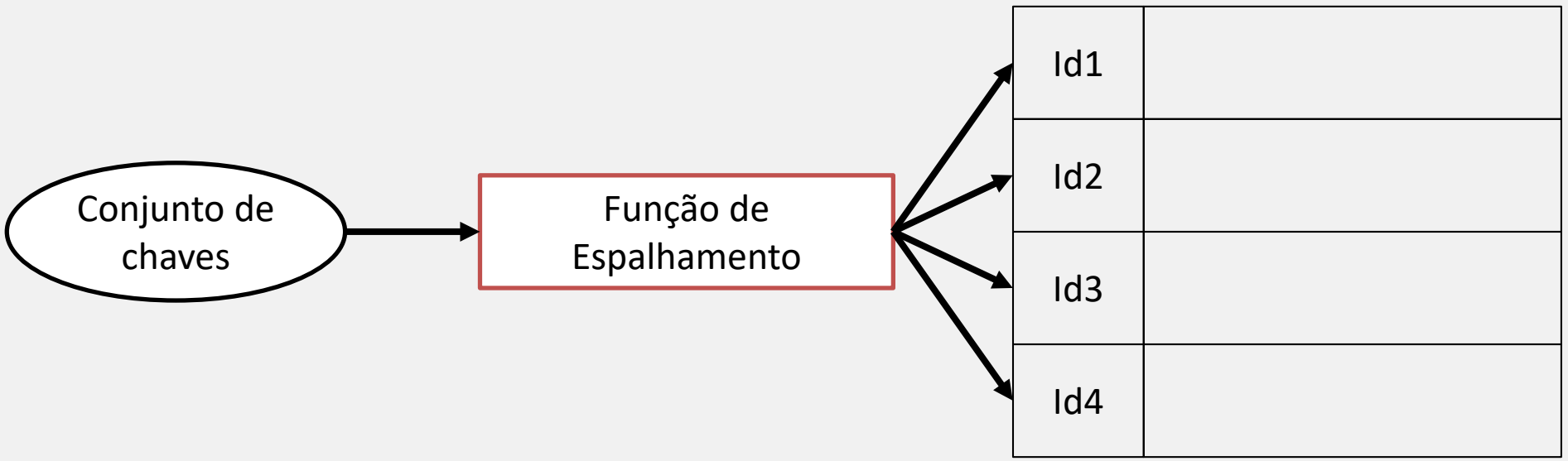
1	1100	Antonio
2		
3	1400	Carlos
4		
5	1800	Eduarda
6	1950	Flavio
7		
8		
9	3150	Ivana
...		
m		

Função de Espalhamento

- A **função de espalhamento** ou **função hash** é responsável por gerar um índice a partir de determinada chave
- O ideal para a função hash é que sejam fornecidos **índices únicos** para as chaves de entrada
- Caso a função seja mal escolhida/implementada, toda a tabela poderá ter um mau desempenho

Função de Espalhamento

- Exemplo:



Função de Espalhamento

- A função deve **mapear** chaves em inteiros dentro do intervalo [0..M-1], onde **M** é o tamanho da tabela
- A função ideal é aquela que:
  - Seja simples de ser computada
  - Para cada chave de entrada, qualquer uma das saídas possíveis é **igualmente provável** de ocorrer.

Funções de Espalhamento

- As funções de hashing necessitam sempre de uma chave, normalmente números inteiros
- Caso a chave seja um outro tipo de dado, uma string por exemplo, é necessário uma função que transforme a chave em um valor inteiro

Funções de Espalhamento

- **Principais funções de hashing:**
  - Divisão
  - Meio do quadrado
  - *Folding* ou desdobramento
  - Análise de dígitos

FUNÇÕES DE HASHING

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Funções de Hashing

- Divisão
- Meio do quadrado
- *Folding* ou desdobramento

Funções de Hashing

- **Divisão**
- Meio do quadrado
- *Folding* ou desdobramento

Método da Divisão

- Técnica simples e muito utilizada que produz bons resultados
- Nessa função, a chave é interpretada como um valor numérico que é dividido por outro valor
- O endereço de um elemento na tabela é dado simplesmente pelo resto da divisão por sua chave por **M**
  - **F(x) = x mod M ou x%M**
    - x = valor da chave (inteiro)
    - M = tamanho do espeço de endereçamento (tamanho da tabela)

Método da Divisão

- **Resto da divisão** (%) do valor da chave (x) pela dimensão do vetor (M):

$$f(x) = x \% M$$

- Quando **M** não for um número primo, uma função que apresenta um melhor comportamento é:

$$f(x) = (x \% p) \% M$$

onde **p** é um número primo maior que **M**

Método da Divisão

- **Exemplo**: calcular a posição da chave x = 1376 para uma tabela com M = 50 posições:

$$f(1376) = 1376 \% 50 = 26$$

- Utilizando a fórmula com um número primo maior que **M**, nesse caso p = 53.

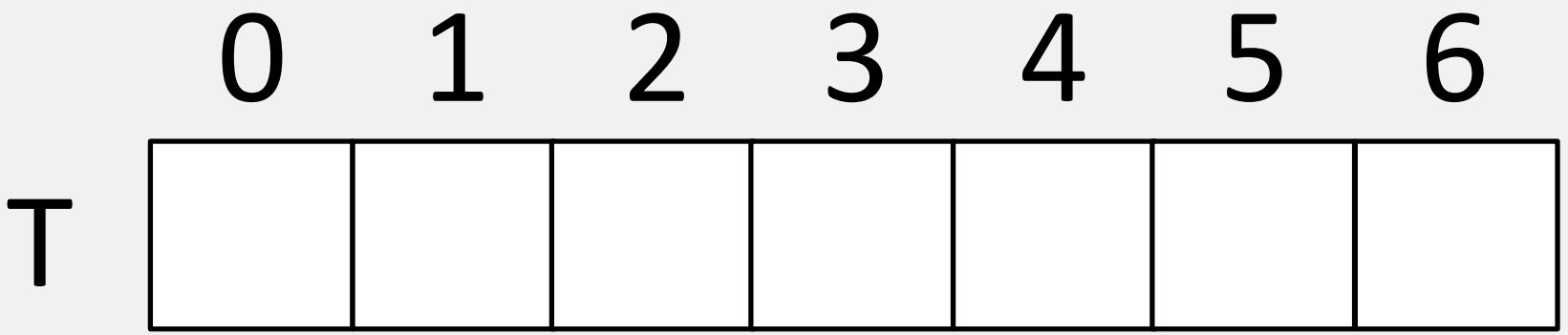
$$f(1376) = (1376 \% 53) \% 50 = 1$$

Método da Divisão

- Para chaves do tipo *string*, tratar cada caractere como um valor inteiro (ASCII), somá-los e pegar o resto da divisão por M
- **Desvantagens**
  - Função extremamente dependente do valor de M escolhido
  - M deve ser preferencialmente um número primo (reduz endereços duplicados)
  - Valores recomendáveis de M devem ser > 20

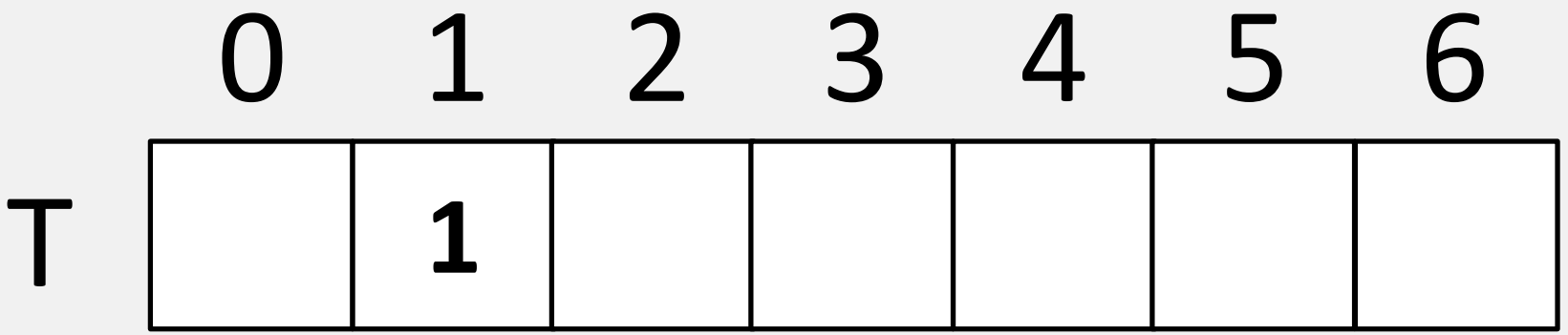
Método da Divisão
Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24



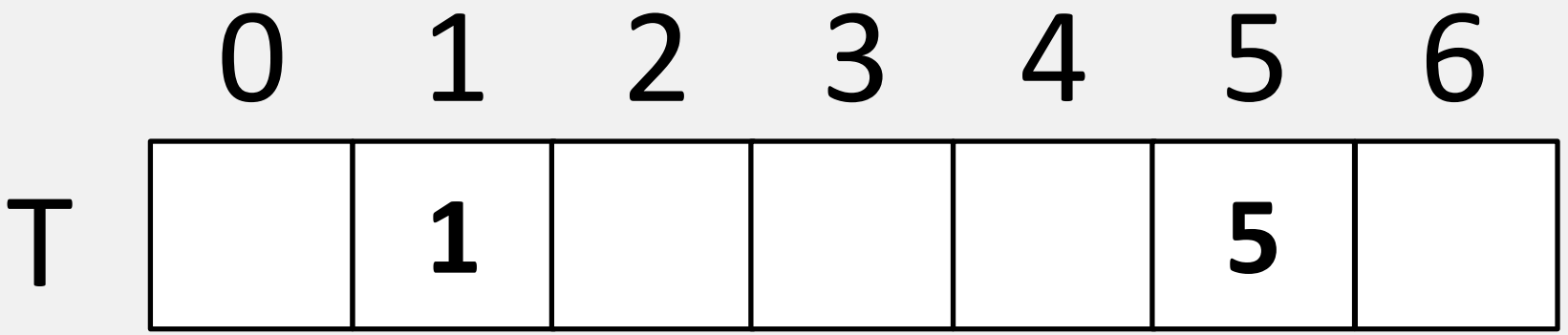
Método da Divisão
Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 1%7 = 1



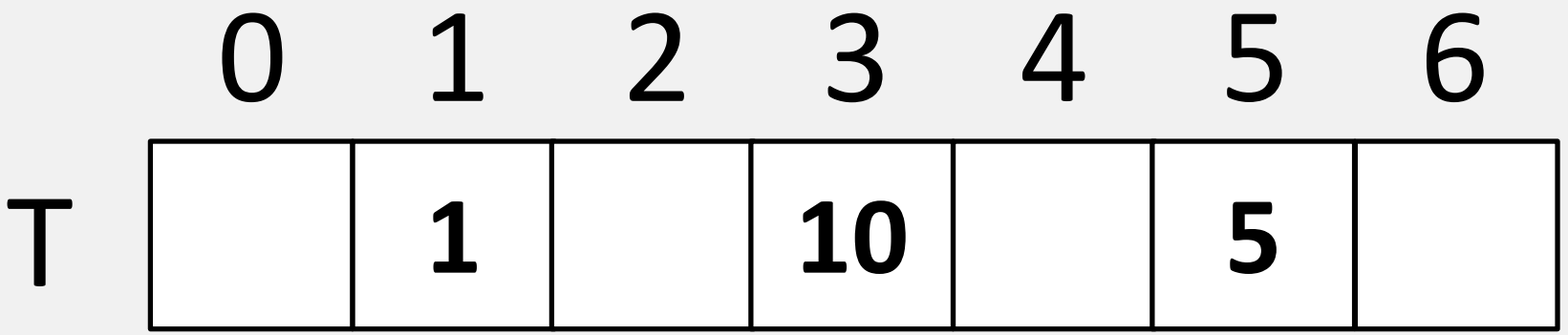
Método da Divisão
Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 5%7 = 5



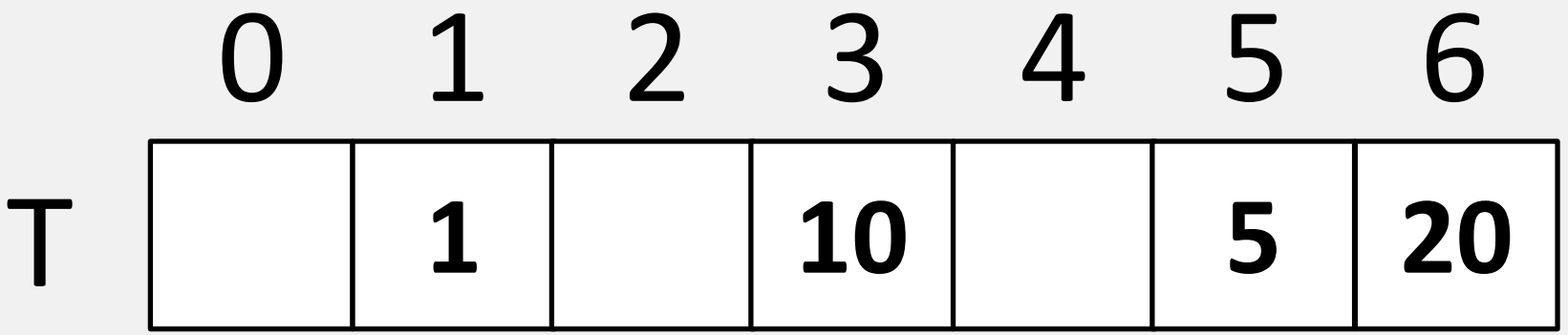
Método da Divisão
Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 10%7 = 3



Método da Divisão
Exemplo

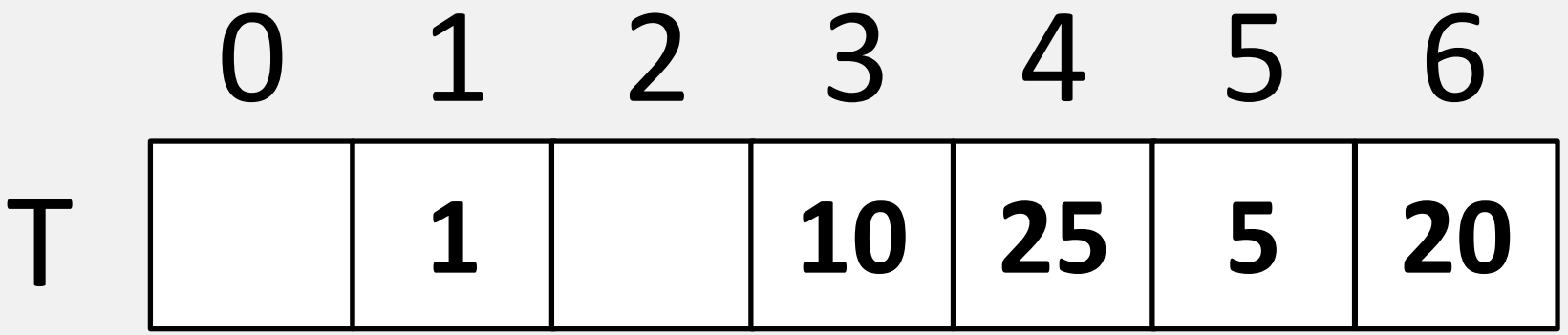
- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 20%7 = 6



Método da Divisão

Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 25%7 = 4



Método da Divisão
Exemplo

- Seja
  - B um vetor de 7 elementos
  - Inserção dos números 1, 5, 10, 20, 25, 24
  - 24%7 = 3
  - **COLISÃO**

	0	1	2	3	4	5	6
T		1		10 24	25	5	20

UDESC

Funções de Hashing

- Divisão
- **Meio do quadrado**
- *Folding* ou desdobramento

Método Meio do Quadrado

- Calculada em 2 passos:
  - Eleva-se a chave ao quadrado
  - Utiliza-se de um determinado número de dígitos (ou bits) do meio do resultado.
- **Exemplo**: calcular o valor para a chave 1376
  - $1376^2 = 1893376$
  - Usando o meio do resultado 1893376 geramos o endereço 933

Funções de Hashing

- Divisão
- Meio do quadrado
- ***Folding* ou desdobramento**

Folding ou Desdobramento

- Usada para cadeia de caracteres
- Baseia-se em uma representação numérica de uma cadeia de caracteres
- Dois tipos:
  - *Shift Folding*
  - *Limit Folding*

Shift Folding

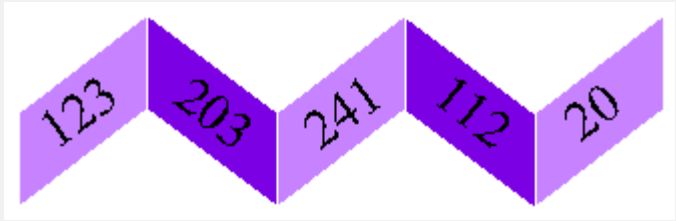
- Divide-se uma *string* em pedaços, onde cada pedaço é representado numericamente, e soma-se as partes.
  - Exemplo mais simples: somar o valor ASCII de todos os caracteres.
- O resultado pode ser usado diretamente ou como uma chave para uma $f'(x)$

Shift Folding
Exemplo

- Calcular a função para a chave “DAD2”
  - Calculando o valor ASCII de cada caractere temos
    - D=68, A=65, D=68, 2=50
  - O *folding* será 68+65+68+50 = **251**

Limit Folding

- Uso da ideia de uma tira de papel como sanfona:



- Soma-se o valor ASCII de todos os caracteres, invertendo os dígitos a cada segundo caractere

Limit Folding
Exemplo

- Calcular a função para a chave “DAD2”
  - Sabendo os valores ASCII:
    - D=68, A=65, D=68, 2=50
  - O folding será 68+56+68+05 = **197**

COLISÕES

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Colisões

- Ocorre quando a função hash produz o mesmo endereço para chaves diferentes
- Funções Hashing perfeitas ou quase perfeitas são as que não aconteceria colisões
- Na prática, funções hash **perfeitas** ou **quase perfeitas** são **encontradas apenas** onde a colisão é intolerável
  - Por exemplo, nas **funções hash** de criptografia, **ou quando conhecemos previamente** o conteúdo **armazenado** na tabela

Colisões

- Qualquer que seja a função de transformação, algumas **colisões** irão ocorrer fatalmente, e tais colisões tem de ser resolvidas de alguma forma
- Mesmo que se obtenha uma função de transformação que distribua os registros de forma uniforme entre as entradas da tabela, existe uma **alta probabilidade** de haver colisões
- Nas tabelas *hash* comuns a **colisão** é apenas indesejável, **diminuindo** o desempenho do sistema (**desempenho gasto** com a **busca** de novos **índices únicos** - processamento)

Colisões

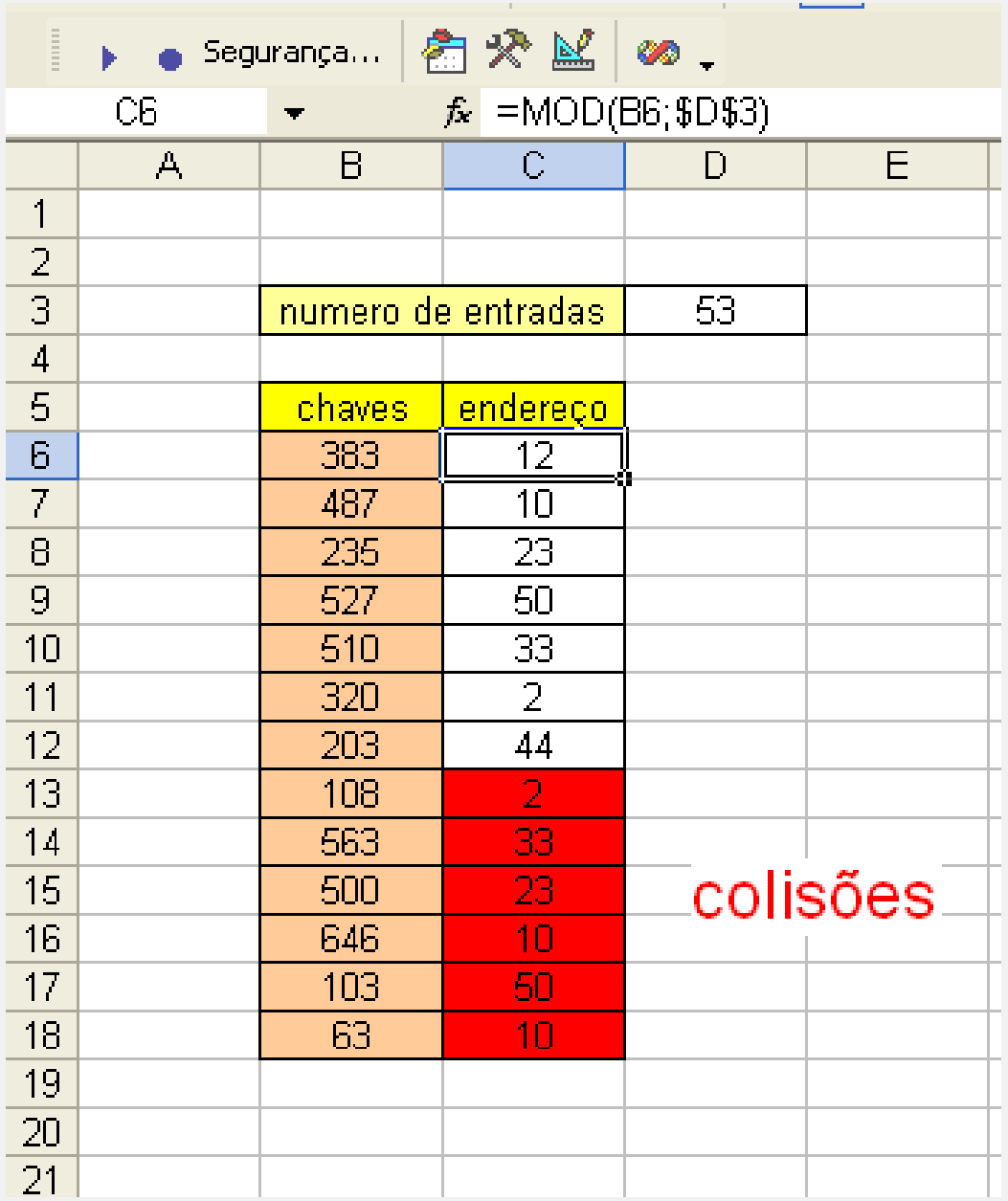
- Muitos **programas funcionam** sem a **suspeita** de que a função hash seja **ruim** e **atrapalhe** o desempenho
- **Por causa** das **colisões**, **muitas** tabelas hash são **aliadas** com **alguma** outra estrutura de dados
  - **Por exemplo**, com listas encadeadas **ou** até **mesmo** com árvores
- Em **outras oportunidades** a colisão é **solucionada dentro** da própria tabela
  - **Por exemplo**, **com** uma estrutura de dados que verifique e corrija **índices duplicados** na tabela

Para um bom Hashing

- Escolher uma boa função de hash – em função dos dados
  - Distribui uniformemente os dados, na medida do possível
  - Evita colisões
- Estabelecer uma boa estratégia para tratamento de colisões

Exemplo de Colisões

- Temos 53 entradas
- Função de Espalhamento:
  - E(chave) = (chave % 53)



UDESC

Paradoxo do Aniversário

- O **paradoxo do aniversário** (Ziviani, 2004), diz que em um grupo de 23 ou mais pessoas, juntas ao acaso, existe uma chance maior do que 50% de que 2 pessoas comemorem aniversário no mesmo dia.
- Assim, se for utilizada uma função de transformação uniforme que enderece 23 chaves randômicas em uma tabela de tamanho 365, a probabilidade de que haja **colisões** é maior do que 50%.

Como tratar colisões?

- Como é quase impossível evitar a colisão é preciso então um meio para tratá-las e assim minimizar seu efeito
- Existem três técnicas básicas

Como tratar colisões?

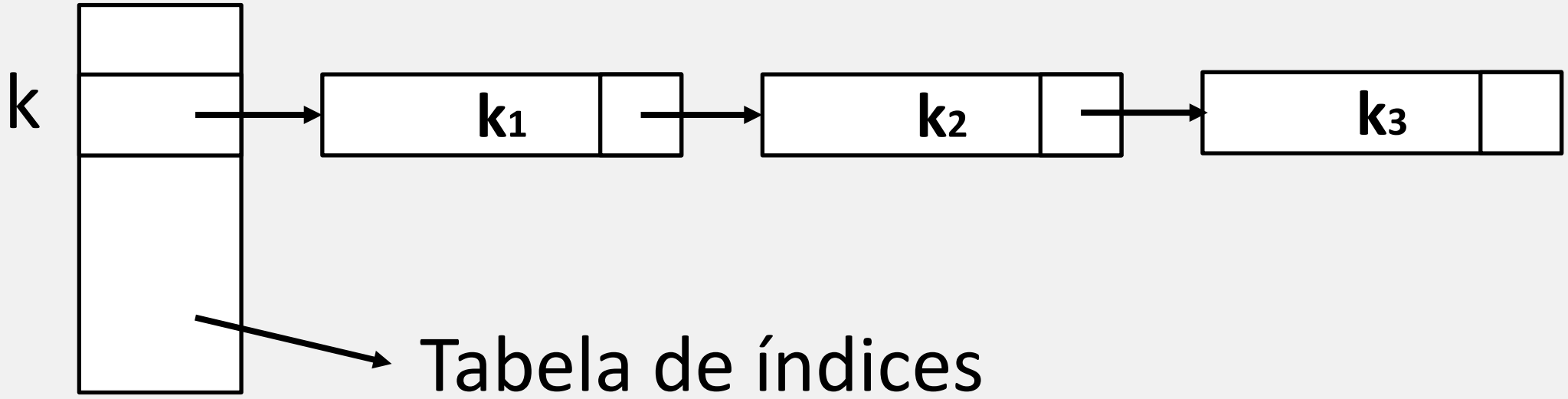
- **Hashing Fechado / Endereçamento Aberto**:
  - Técnicas de rehash para tratamento de colisões
  - Permite armazenar um conjunto de informações de tamanho limitado
- **Hashing Aberto / Encadeamento Externo**:
  - Encadeamento de elementos para tratamento de colisões
  - Permite armazenar um conjunto de informações de tamanho potencialmente ilimitado
- **Hashing Multiplo / Rehash**

Hashing Aberto

- Uma das formas de resolver as colisões é construir uma **lista linear encadeada** para cada endereço da tabela. Assim todas as chaves com o mesmo em endereço serão encadeadas em uma lista

Hashing Aberto

- A tabela de índices, contém apenas os ponteiros para uma lista de elementos
- Quando há colisão, o elemento é inserido no índice como um novo nó da lista



UDESC

Hashing Aberto

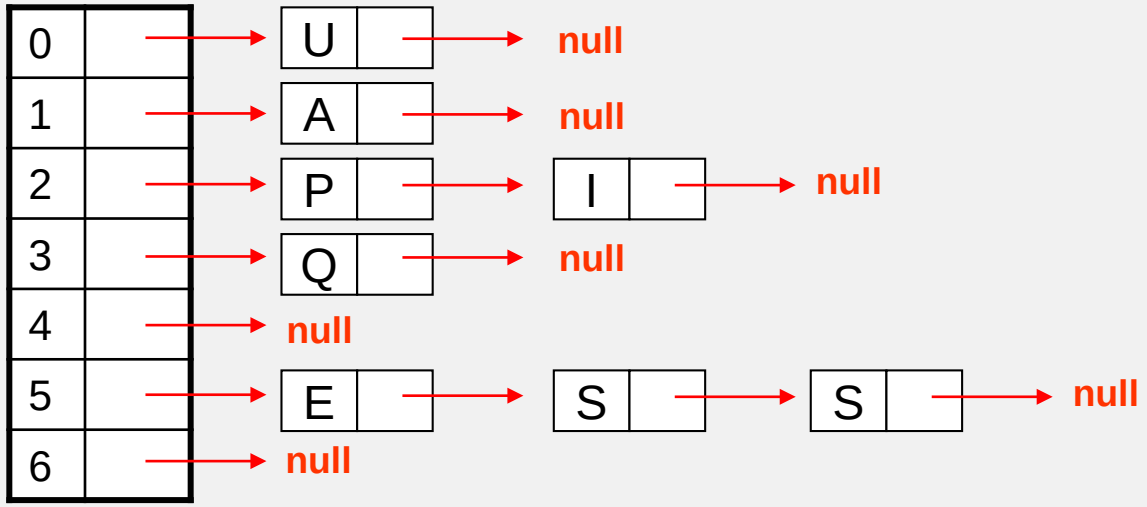
Exemplo

- Se a i-ésima letra do alfabeto é representada pelo número *i* e a função de transformação h(Chave) = Chave % M é utilizada para M = 7, o resultado da inserção das chaves P, E, S, Q, U, I, S, A na tabela é o seguinte:

Hashing Aberto

Exemplo

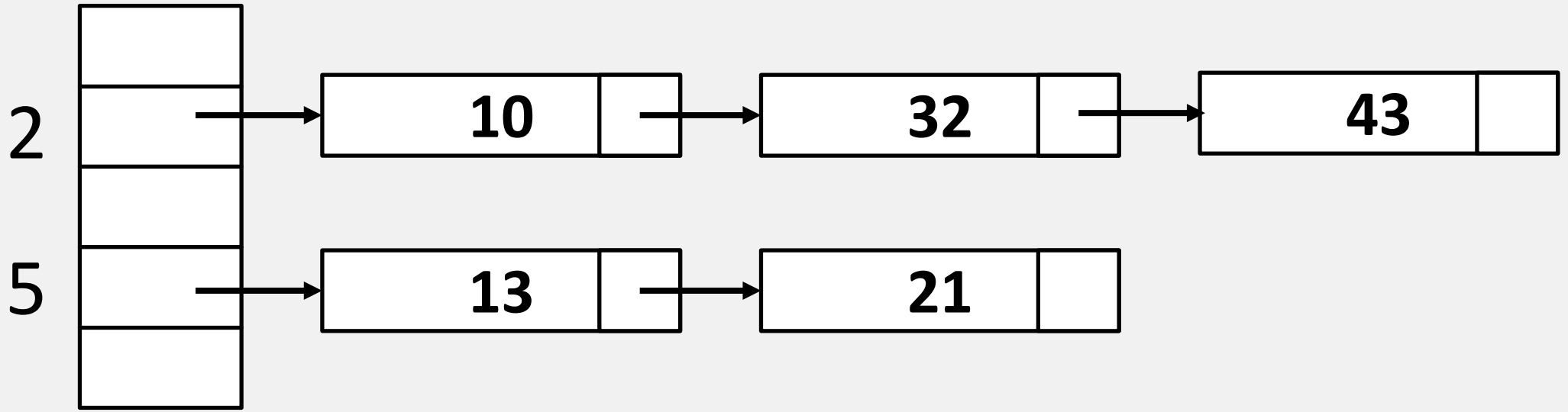
- f(A) = f(1) = 1 % 7 = 1
- f(E) = f(5) = 5 % 7 = 5
- f(S) = f(19) = 19 % 7 = 5...



UDESC

Hashing Aberto

- Se as listas estiverem ordenadas, reduz-se o tempo de busca



Hashing Fechado

- Quando o **número de registros** a serem armazenados na tabela puder ser **previamente estimado**, então não haverá necessidade de usar apontadores para armazenar os registros
- Existem vários métodos para armazenar **N** registros em uma tabela de tamanho **M > N**, os quais utilizam **lugares vazios** na própria tabela para resolver as colisões

Hashing Fechado

- O método utilizado pelo endereçamento aberto consiste em, a partir da posição já ocupada, prosseguir a verificação, pela ordem, das posições subsequentes, até que seja encontrada uma posição não utilizada (vazia)

Hashing Fechado

- Uma tabela de **itens** é utilizada para armazenar informações
  - Os elementos são armazenados na própria tabela
- Colisões
  - Aplicar técnica de rehash

Hashing Fechado

Exemplo

- Se a i-ésima letra do alfabeto é representada pelo número i e a função de transformação h(Chave) = Chave % M é utilizada para M = 7, então o resultado da inserção das chaves L U N E S na tabela, usando *endereçamento aberto* para resolver colisões é mostrado a seguir:

f(L) = h(12) = 5

f(U) = h(21) = 0

f(N) = h(14) = 0

f(E) = h(5) = 5

f(S) = h(19) = 5

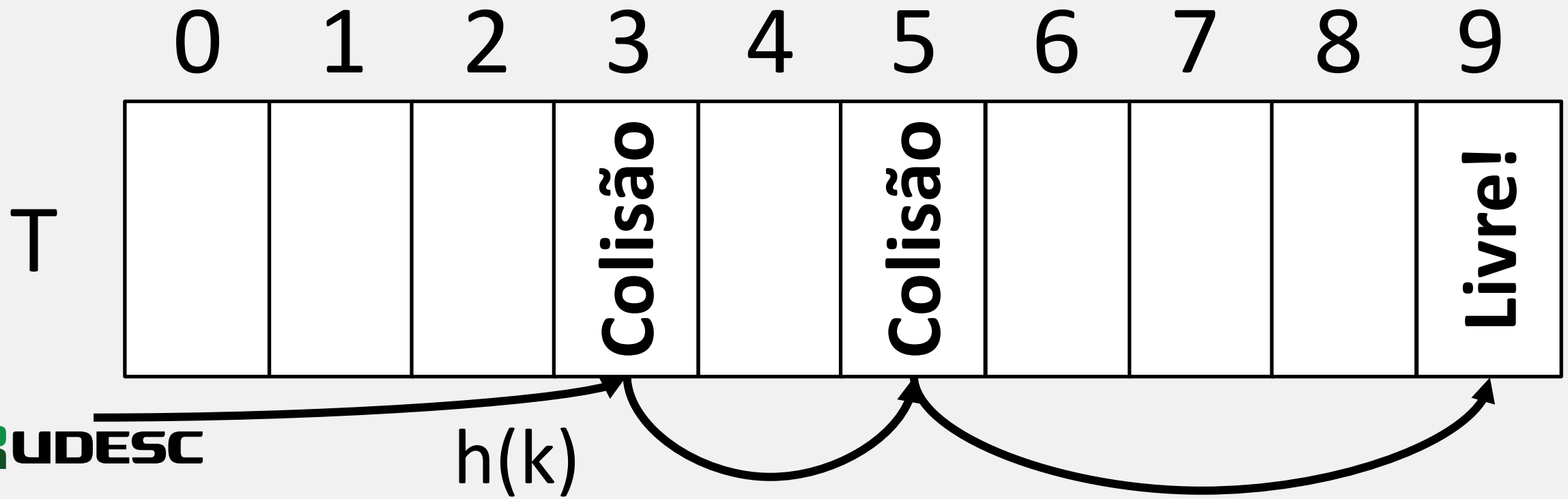
0	U
1	N
2	S
3	
4	
5	L
6	E

Hashing Múltiplo

- Outra forma de resolver uma colisão é utilizar o chamado **hashing múltiplo**:
  - Caso a primeira função hash tenha gerado uma colisão, aplica-se uma segunda função de hash.
  - Se novamente ocorrer uma colisão, o programa usará o endereçamento aberto ou uma terceira função hash.

Hashing Múltiplo

- Se posição **h(k)** está ocupada (lembre-se de que h(k) é um índice da tabela), aplicar função de **rehash** sobre **h(k)**, que deve retornar o próximo índice livre



UDESC

Hashing Dinâmico

- No hashing estático o espaço de armazenamento hash é fixo, os esquemas de hashing dinâmico tentam reparar esta situação com a criação de novas estruturas de acordo com o crescimento do arquivo dinâmico

Resumo

- Uma **tabela hash** é baseada em um **vetor;**
- A faixa de valores-chaves é geralmente maior que o tamanho do vetor;
- Um **valor-chave** é **convertido** em um **índice** de vetor por uma **função de hash**;
- A conversão de uma chave para uma célula do vetor já preenchida é chamada de **colisão**;
- Colisões podem ser lidadas de duas maneiras principais: encadeamento separado ou endereçamento aberto:
  - Com **encadeamento separado**, cada elemento do vetor consiste de uma lista encadeada. Todos os itens de dados que se convertem para um certo índice do vetor são inseridos nessa lista;
  - Com **endereçamento aberto**, itens de dados que se convertam para uma célula cheia do vetor são colocados em outra célula livre no vetor.

IMPLEMENTAÇÃO EM JAVA

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Implementação em Java

Pacote java.util

- A linguagem JAVA utiliza o **conceito** de “Maps” (**mapas**) para **implementar** Tabelas Hash:
  - Um Map é uma **coleção** de **pares** chave/valor
- O **pacote** java.util (import java.util.*;) **possui** as seguintes **classes** que **implementam** os **conceitos** de tabelas hash **para** a manipulação de mapas:
  - **Classe** HashMap (**import** java.util.HashMap;)
  - **Classe** Hashtable (**import** java.util.Hashtable;)
- **Cada** classe tem um **conjunto** de métodos **próprios** para a manipulação dos mapas

Implementação em Java
Classe HashMap

- HashMap é uma **estrutura** para o **rápido** armazenamento e recuperação de **objetos**, aceita **objetos nulos**, e não garante **fornecer** uma **lista** em ordem cronológica de inserção do **conteúdo**. Um **exemplo** de utilização:

```
import java.util.HashMap;
public class aula8exemplo1
{
        public static void main (String args[])
        {
                HashMap telefones = new HashMap();
                telefones.put("Pizza", "34354455");
                String numero=(String)telefones.get("Pizza");
                System.out.println(numero);
        }
}
```

chave-valor

Obs.: permite predefinir o tipo <K,V>, evitando a necessidade de conversão *cast*

UDESC

Implementação em Java

Classe HashMap – Principais Métodos

- put(Object key, Object value): associa (**adiciona**) um valor a uma **determinada** chave
- remove(Object key): remove uma **entrada especificada** pela chave
- clear(): remove **todas** as **entradas** do HashMap
- size(): **retorna** o número de chaves-valor do **mapa**
- containsValue(Object value): **retorna** *true* **se** o mapa contém **uma ou mais** chaves **associadas** com um valor **específico**

Implementação em Java

Classe HashMap – Principais Métodos

- isEmpty(): **retorna** *true* **se** o mapa **não contiver** nenhum **par** chave-valor
- values(): **retorna uma** visão dos valores **contidos** no mapa
- get(Object key): **retorna** o valor **associado** à chave ou *null*
- keySet(): **retorna** uma visão das chaves **contidas** no **mapa**

Implementação em Java

Classe Hashtable

- Hashtable é uma **estrutura** que **implementa** tabelas Hash em **JAVA**, **mapeando** chaves para valores
  - **Qualquer** objeto **não nulo** pode **ser usado como** uma chave **ou** como um valor

```
Hashtable numeros = new Hashtable();
numeros.put("um", 1);
numeros.put("dois", 2);
numeros.put("tres", 3);
Integer n = (Integer)numeros.get("dois");
if (n != null) {
    System.out.println("dois = " + n);
}
```

Implementação em Java

Classe Hashtable – Principais Métodos

- put(Object key, Object value): mapeia (**associa**) um valor a uma **determinada** chave
- get(Object key): **retorna** o valor **associado** à chave
- clear(): remove **todas** as **entradas** da HashTable
- isEmpty(): **retorna** *true* **se** a tabela **não contiver** nenhum **par** chave-valor
- size(): **retorna** o número de chaves da **tabela**

Implementação em Java

Classe Hashtable – Principais Métodos

- toString(): **retorna** uma string do **objeto** HashTable, **com** as **entradas separadas** por vírgula (",") **e entre** chaves (“{ }”)
- remove(Object key): remove **uma** chave **e** seu valor **correspondente**

Implementação em Java

Classe HashMap x Classe Hashtable

- HashMap:
  - **Não é** *syncronized*, ou seja, **não garante** o sincronismo dos **métodos** quando forem **executados** por *threads* (o que pode modificar a estrutura do mapa, adicionando/removendo novos itens – exceto alteração do valor)
  - **Aceita** *null* tanto **no** valor quanto **na** chave
- Hashtable:
  - **É** *syncronized*
  - **Não aceita** *null*

Implementação em Java

Classe HashMap x Classe Hashtable

- Uma **equivalente** HashMap **sincronizada** pode ser **obtida** utilizando-se:
  - Collections.synchronizedMap(minhaMap)

Implementação em Java

Meios de Acesso

- **Toda** coleção **possui** uma **representação interna** para o armazenamento e a organização de seus **elementos**
- Por outro lado, essa coleção deve **permitir** que seus elementos **sejam** acessados **sem** que **sua** estrutura interna seja **exposta**
- A linguagem JAVA **provê** uma forma de sequencialmente **acessar** os elementos de **uma** coleção sem **expor** sua **representação interna**

Implementação em Java
Meios de Acesso

- De uma maneira geral, pode-se **desejar que** os elementos **sejam** percorridos de **várias maneiras**, **sem** no entanto ter que **modificar** a interface da coleção em **função** do tipo de varredura **desejado**
  - **Por exemplo**: percorrer de trás para frente ou vice-versa

Implementação em Java
Interface Iterator

- A interface Iterator (**import** java.util.Iterator;) **permite** implementar uma **forma** de **percorrer** os elementos de **uma** coleção **sem violar** o encapsulamento da **mesma**
  - **Objetivo**: iterar **sobre** (percorrer sequencialmente) **uma** coleção de objetos **sem expor** sua **representação**

Implementação em Java
Interface Iterator - Métodos

- A **interface** Iterator **possui** os **seguintes** métodos:
  - hasNext(): **retorna** *true* **se** a iteração **possui** mais elementos
  - next(): **retorna** o próximo elemento da **iteração**
  - remove(): **remove** da **iteração** o último elemento **retornado** da coleção (é um método opcional)

Implementação em Java

Interface Iterator – Requisitos para Utilizaçõ

- Como um **modo** de localizar um elemento específico da **coleção**, tal **como** o **primeiro elemento**
- Um **modo** de **obter** acesso ao elemento atual
- Um **modo** de **obter** o próximo elemento
- Um **modo** de indicar que **não há mais** elementos a **percorrer**

```
Iterator k = map.keySet().iterator();
while (k.hasNext()) {
        String key = (String) k.next();
        System.out.println("Chave: " + key + "; Valor: " + (String) map.get(key));
}
```

UDESC

EXERCÍCIOS

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Exercício

1. Dadas as chaves 1030, 10054, 839, 2030, 17 e 136 calcule as respectivas posições para uma tabela com 100 elementos (M=100), usando o método da divisão para a função de hashing.
2. Agora recalcule as posições adaptando a função hash com o número primo p = 101.

Exercícios

3. Crie um programa em JAVA que armazene em uma tabela com HashMap o **nome dos estados** na seguinte ordem:
   - **Santa Catarina; Rio de Janeiro; São Paulo; Parana; e Rio Grande do Sul**

   a) Utilize como chave a **sigla do estado (SC, RJ, SP, PR e RS)**
   b) Mostre na tela o conteúdo da tabela utilizando os métodos get() e keySet()
   c) Verifique se a **ordem dos dados** mostrados é a mesma da sequencia em que foram inseridos
   d) Insira um par chave-valor primeiro com o **valor nulo**, depois tente com a **chave nula**
   e) Verifique se esta **inserção é aceita** e se todos os pares são apresentados com o **método** values()

Exercícios

4. Crie um outro programa em JAVA agora utilizando Hashtable, que armazene o **nome dos estados** utilizando como chave sua respectiva **sigla**
   a) Execute a mesma **sequência de testes** feita para o primeiro exercício
   b) Tente inserir um par chave-valor duplicado, p. ex.: Santa Catarina, e veja o resultado
   c) Inclua sua **análise comparativa**, do **Hashtable** com o **HashMap**, no corpo do programa como **comentário** (/* */)

Exercícios

5. Teste um dos exercícios com a **interface** Iterator

Exercícios

6. Se a i-ésima letra do alfabeto é representada pelo número i e a função de transformação h(Chave) = Chave % M é utilizada para M = 7. Dadas as chaves U, D, E, S, C, quais os resultados da inserções na tabela utilizando:
   a) Hashing aberto / encadeamento externo
   b) Hashing fechado / endereçamento aberto

Exercícios

7. Dadas as chaves 44, 13, 88, 94, 11, 39, 20 e 5 e usando a função de hash $h(x) = x \% 11$, desenhe a tabela de dispersão considerando o tratamento de colisões por:
   a) Hashing aberto / encadeamento externo
   b) Hashing fechado / endereçamento aberto

EXERCÍCIOS EM DUPLA OU TRIO

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Exercícios em dupla ou trio

1. Crie uma tabela Hash para armazenar as disciplinas de um curso. A chave da tabela será a sigla da disciplina que deverá ter no máximo 5 caracteres. A função de transformação representará a i-ésima letra do alfabeto pelo número i enquanto os dígitos numéricos manterão o seu valor. Individualmente, o valor de cada caractere será multiplicado pelo respectivo peso (potencia de 10) de sua posição, gerando um valor que será aplicado na função f(val_chave) % M.

(continua no próximo slide)

Exercícios em dupla ou trio

- M é o valor da tabela que deverá ser definido na inicialização do programa.
- Utilize a técnica de endereçamento aberto para o tratamento de colisões.

Ex: chave = DAD2, considerando uma tabela com M = 7

$D = 4 \rightarrow 4 * 10^3 = 4000$
$A = 1 \rightarrow 1 * 10^2 = 100$
$D = 4 \rightarrow 4 * 10^1 = 40$
$2 = 2 \rightarrow 2 * 10^0 = 2$

h(4142) → 4142 % 7 = 5

UDESC

Exercícios em dupla ou trio

2. Com base no exercício anterior faça o tratamento de colisões por lista encadeada.

Exercícios em dupla ou trio

3. Crie um programa que permita inserir e consultar elementos em uma tabela Hash. Na tabela serão armazenados os dados das disciplinas de um curso, a chave será a sigla da disciplina. A função de transformação representará a i-ésima letra do alfabeto pelo número i enquanto os dígitos numéricos manterão o seu valor. Individualmente, o valor de cada caractere será multiplicado pelo respectivo peso (potencia de 10) de sua posição, gerando um valor que será aplicado na função h(val_chave) % M

- M é o valor da tabela que deverá ser definido na inicialização do programa (preferencialmente um número primo)
- Utilizar a técnica de lista encadeada para o tratamento de colisões

Obrigado

jacksonpradolima.github.io
github.com/ceplan

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA