Estrutura de Dados – 1º semestre de 2021

Professor Mestre Fabio Pereira da Silva

Tabela de Espalhamento

- Índices em vetores ou listas sequenciais são utilizados para acessar informações
- No entanto, se quisermos acessar uma informação de um determinado conteúdo (e não posição)?

Família

1	2	3	4	5	6
José Maria	Leila	Artur	Jolinda	Gisela	Alciene

```
Família[1] = "José Maria"
Família[3] = "Artur"
Família[2] = "Leila"
```

Em qual posição está "Alciene" ?

Tabela de Espalhamento

- **Hash** é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo dicionário
- Dicionários são estruturas especializadas em prover as operações de inserir, pesquisar e remover.
- A ideia central do **Hash** é utilizar uma função, aplicada sobre parte da informação (**chave**), para retornar o índice onde a informação deve ou deveria estar armazenada.

Tabela de Espalhamento

- Esta função que mapeia a chave para um índice de um arranjo é chamada de **Função de Hashing**
- A estrutura de dados Hash é comumente chamada de **Tabela Hash.**

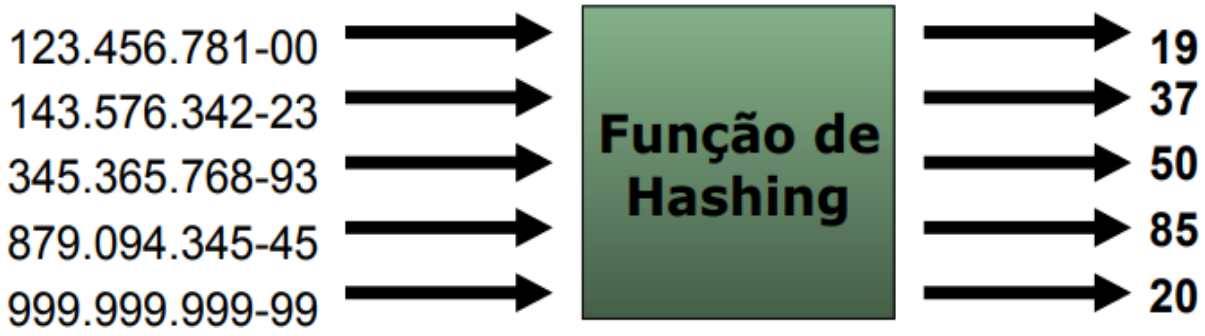
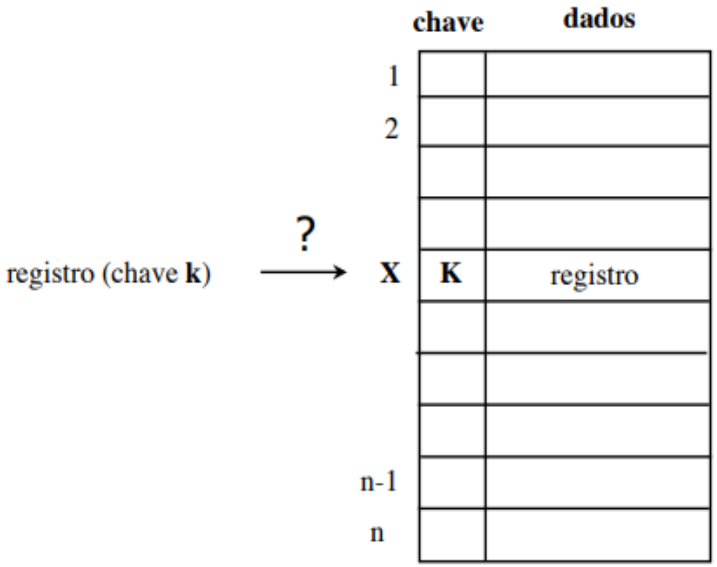


Tabela Hash

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.**
...	

Tabela de Espalhamento

- É uma estrutura de dados especial
- Armazena as informações desejadas associando chaves
- Objetivo
  - A partir de uma chave, fazer uma busca rápida e obter o valor desejado.



Como o registro (com chave **K**) foi armazenado na posição **X** na Tabela Hash ao lado?

Resp: Através de uma **Função de Hashing**.

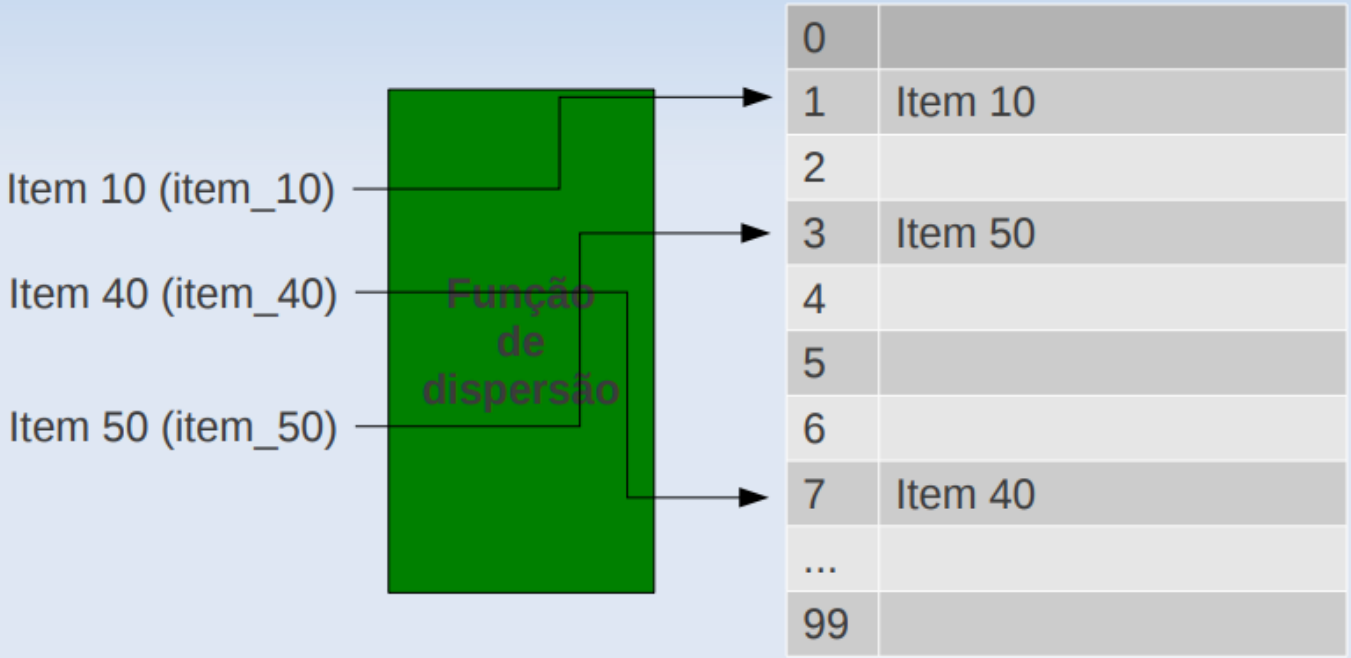
Tabela de Espalhamento

- Tabela de dispersão: permite realizar buscas eficientes em um conjunto de elementos armazenados em uma tabela (um vetor) **de maneira não ordenada**
- Mecanismos de **inserção e recuperação do elemento**
- Dado um par {chave, elemento}, onde chave é a chave de busca e elemento é o elemento/dado a ser armazenado:
- Inserção: Utilizando a chave, calcula-se o índice da posição na tabela onde o elemento deve ser armazenado
- Inserir o elemento na posição calculada
- Recuperação (busca/remoção)
- Utilizando a chave, calcula o índice onde o elemento está armazenado
- Pega o elemento na posição do índice calculado

Tabela de Espalhamento

- Dado um conjunto de pares (chave,valor)
- Determinar se uma chave está no conjunto e o valor associado (busca)
- Inserir um novo par no conjunto
- Remover um par do conjunto
- Uma tabela de dispersão (espalhamento) é uma **estrutura de dados eficiente**
- A busca por um elemento, no pior caso, pode levar O(n)
- O tempo médio esperado de busca é na ordem de O(1)

- Exemplo: inventário de 100 itens



Vantagens e Desvantagens

- Vantagens:
  - Algoritmos simples e eficientes para inserção, busca e remoção
- Desvantagens:
  - Nenhuma garantia de balanceamento (depende da função de dispersão)
  - Espaço subutilizado nas tabelas (depende da função de dispersão)
  - O grau de espalhamento é sensível à função de dispersão utilizada e ao tipo de informação usada como chave

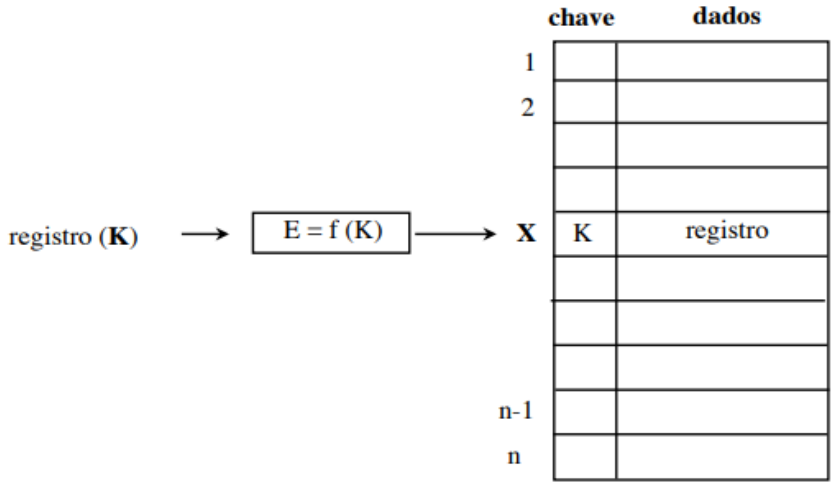
Representação

- Vetor ou lista sequencial
- Cada posição do vetor guarda uma informação.
- Se a função de hashing aplicada a um conjunto de elementos determinar as informações i 1, i2, ..., in " então o vetor V[1... n] é usado para representar a tabela hash

Função de Hashing

- A Função de Hashing é a responsável por gerar um **índice a partir de uma determinada chave**
- O ideal é que a **função forneça índices únicos** para o conjunto das chaves de entrada possíveis
  - sem colisões
  - fácil de computar
  - uniforme (todos os locais da tabela sejam igualmente utilizados)
  - extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash

Função de Hashing

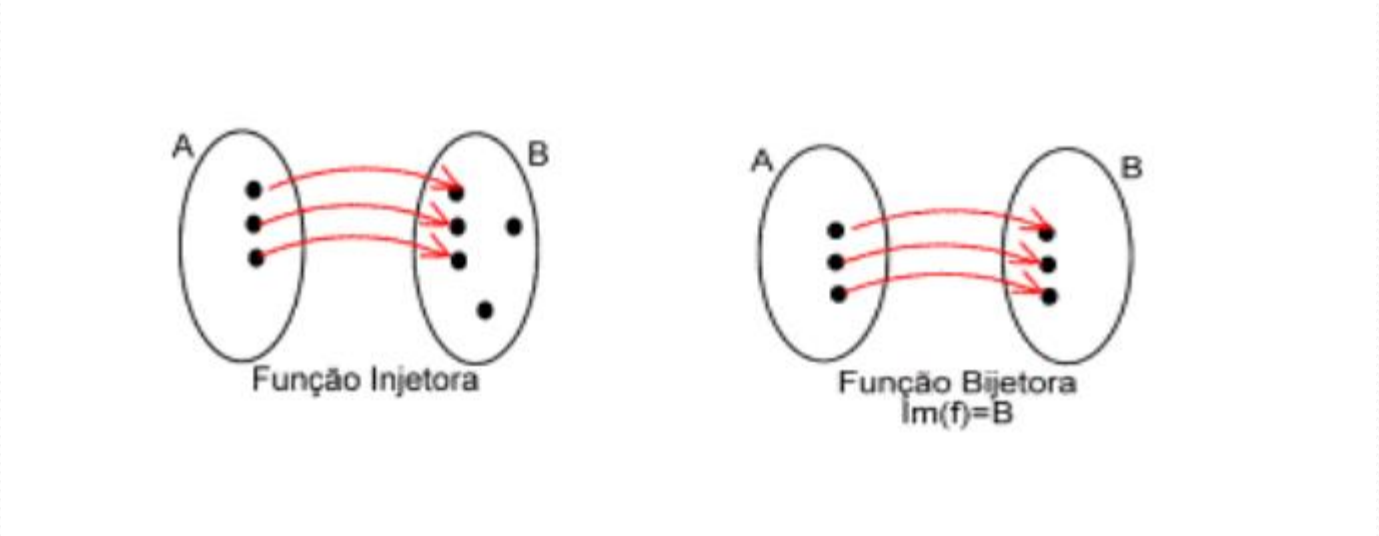


- Os valores da chave podem ser **numéricos**, **alfabéticos** ou **alfa-numéricos**.

- Executam a transformação do valor de uma chave em um endereço, pela aplicação de operações aritméticas e/ou lógicas
  - f: C -> E
  - f: função de cálculo de endereço
  - C: espaço de valores da chave (domínio de f)
  - E: espaço de endereçamento (contradomínio de f)

Hashing Perfeito

- Para quaisquer chaves x e y diferentes e pertencentes a A, a função utilizada fornece saídas diferentes;

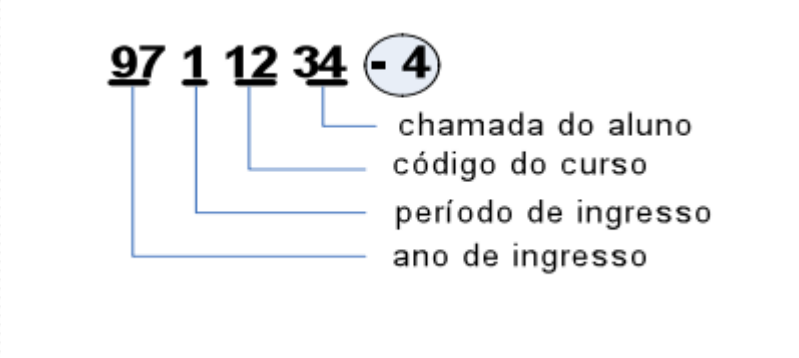


Exemplo

- Armazenamento de alunos de uma determinada turma de um curso específico
- Classe: Aluno
- Identificação: Número de matrícula com até 7 caracteres
- Visto que a matrícula é composta de 7 dígitos, então podemos esperar uma matrícula variando de ooooooo a 9999999.
- Portanto, precisaríamos dimensionar um vetor com DEZ Milhões de elementos

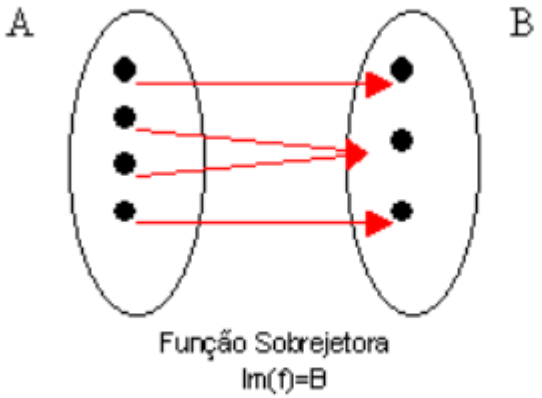
Exemplo

- Como economizar em espaço, mas ainda usando hashing perfeito?
- Identificando as partes significativas da chave para gerar a função de hashing



Hashing Imperfeito

- Existe chaves x e y diferentes e pertencentes a A, onde a função Hash utilizada fornece saídas iguais;



Hashing Imperfeito

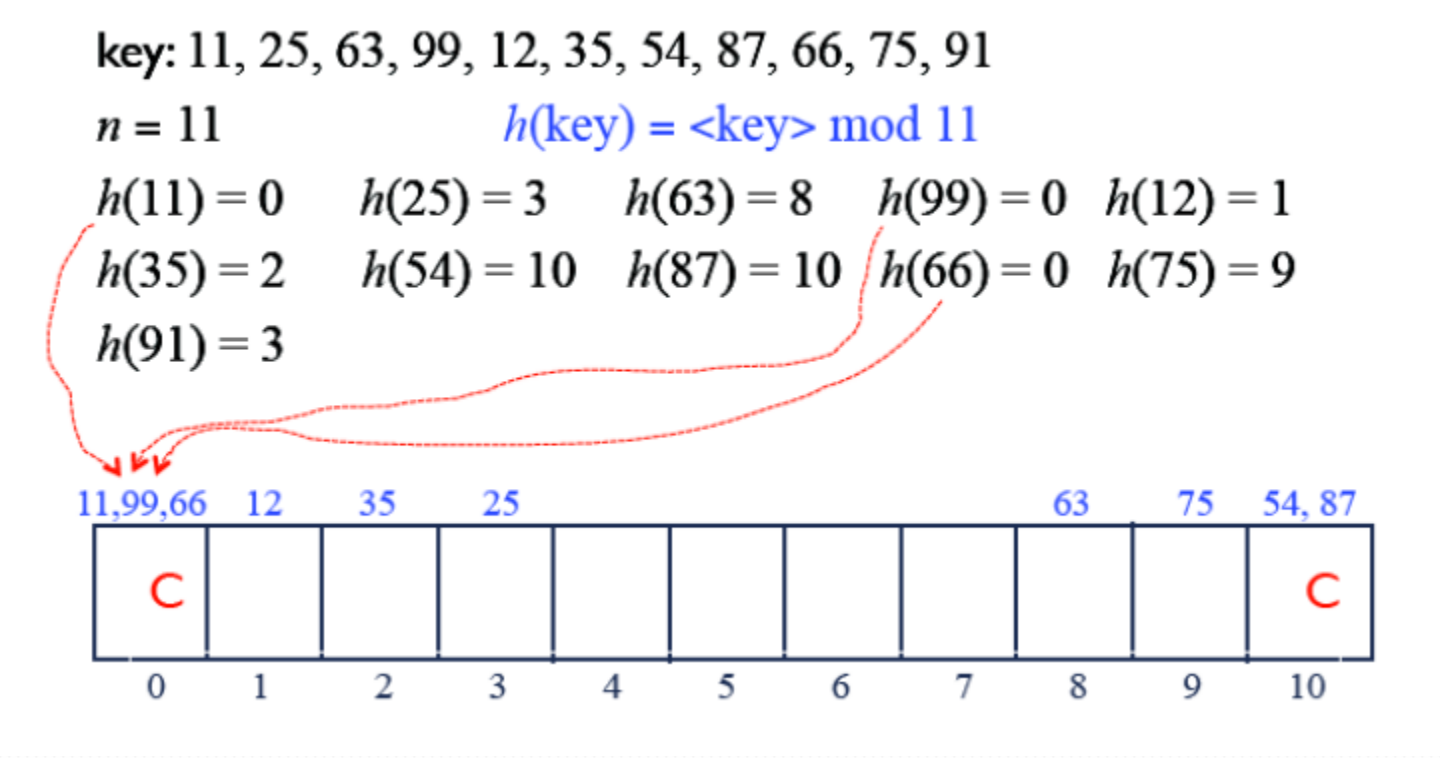
- Suponha que queiramos armazenar as seguintes chaves: C, H, A, V, E e S em um vetor de P.= 7 posições (0..6) conforme a seguinte
- Função f(k) = k(código ASCII) % P.

símbolo	ASCII	f(k)
C	67	4
H	72	2
A	65	2
V	86	2
E	69	6
S	83	6

Colisões

- E se duas (ou mais) chaves forem mapeadas para o mesmo índice ?

.

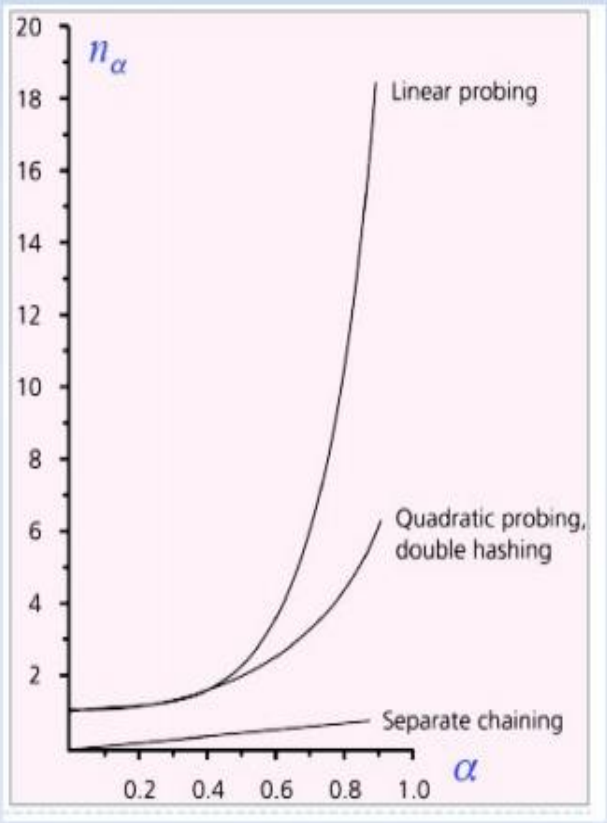


Colisões

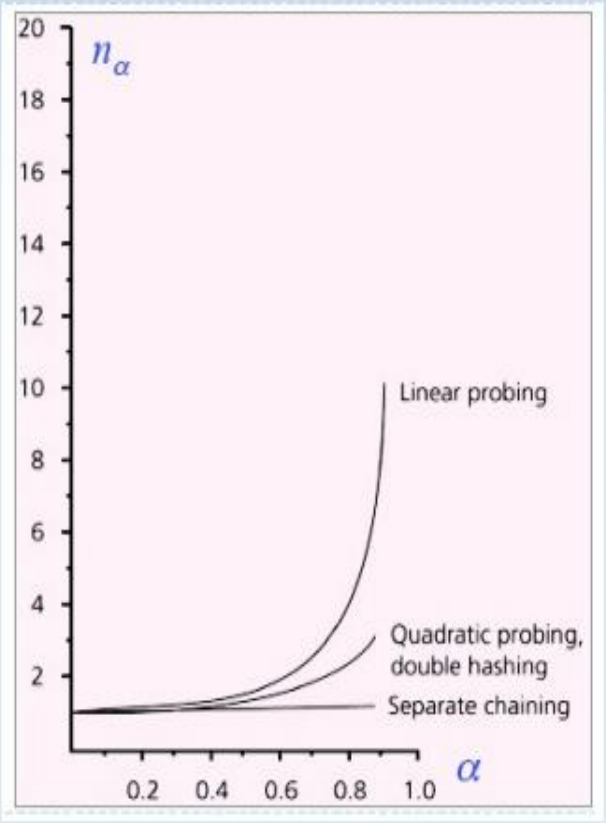
- Principais causas:
- Em geral o número N de chaves possíveis é muito maior que o número m disponíveis na tabela
- Não se pode garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento)

Colisões - Exemplo

- Dado um grupo de 23 (ou mais) pessoas escolhidas aleatoriamente, a chance de que duas pessoas terão a **mesma data de aniversário e de mais de 50%.**
- **Para 57 ou mais pessoas, a probabilidade é maior do que 99%.**
- A colisão pode ser evitada conhecendo-se todas as **chaves**
- Escolhe-se a função de dispersão ideal (perfeita) para obter a indexação perfeita
- Como isto não é prático, outras técnicas são utilizadas



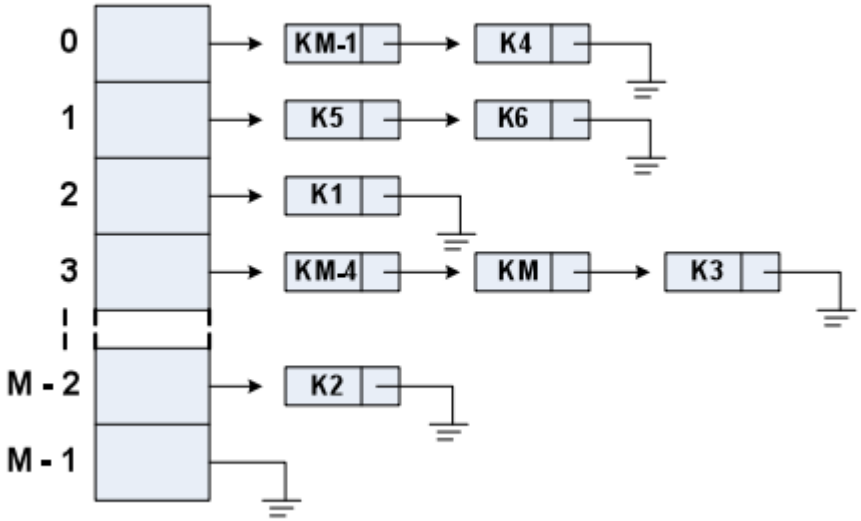
Pesquisa sem sucesso



Pesquisa com sucesso

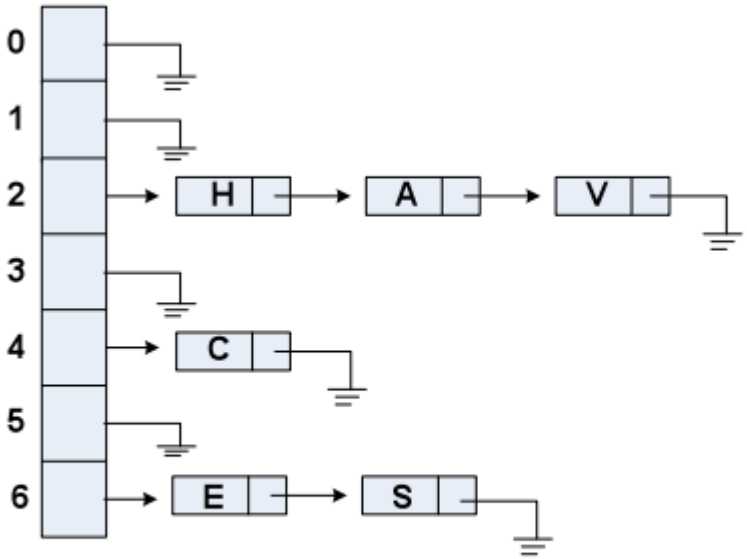
Colisões - Encadeamento

- Cada entrada na tabela aponta para uma lista encadeada
- Colisões geram uma nova entrada em uma lista
- A função utilizada deve ser uniforme para evitar uma grande lista encadeada em poucas posições da tabela
- Cada busca só será constante se o número de elementos em cada lista encadeada for pequeno

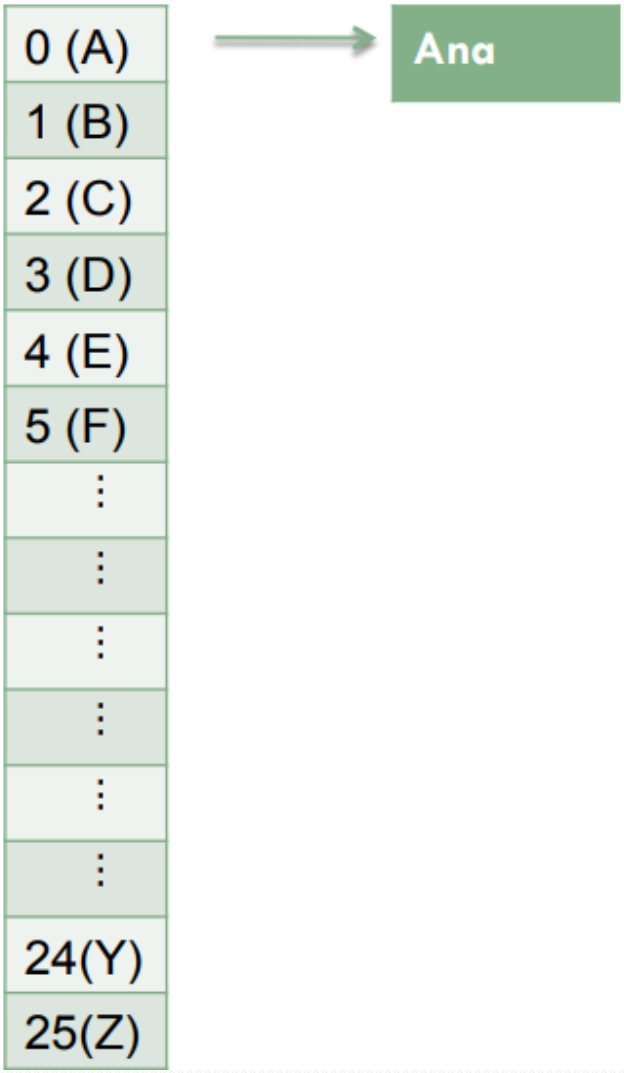


Colisões - Encadeamento

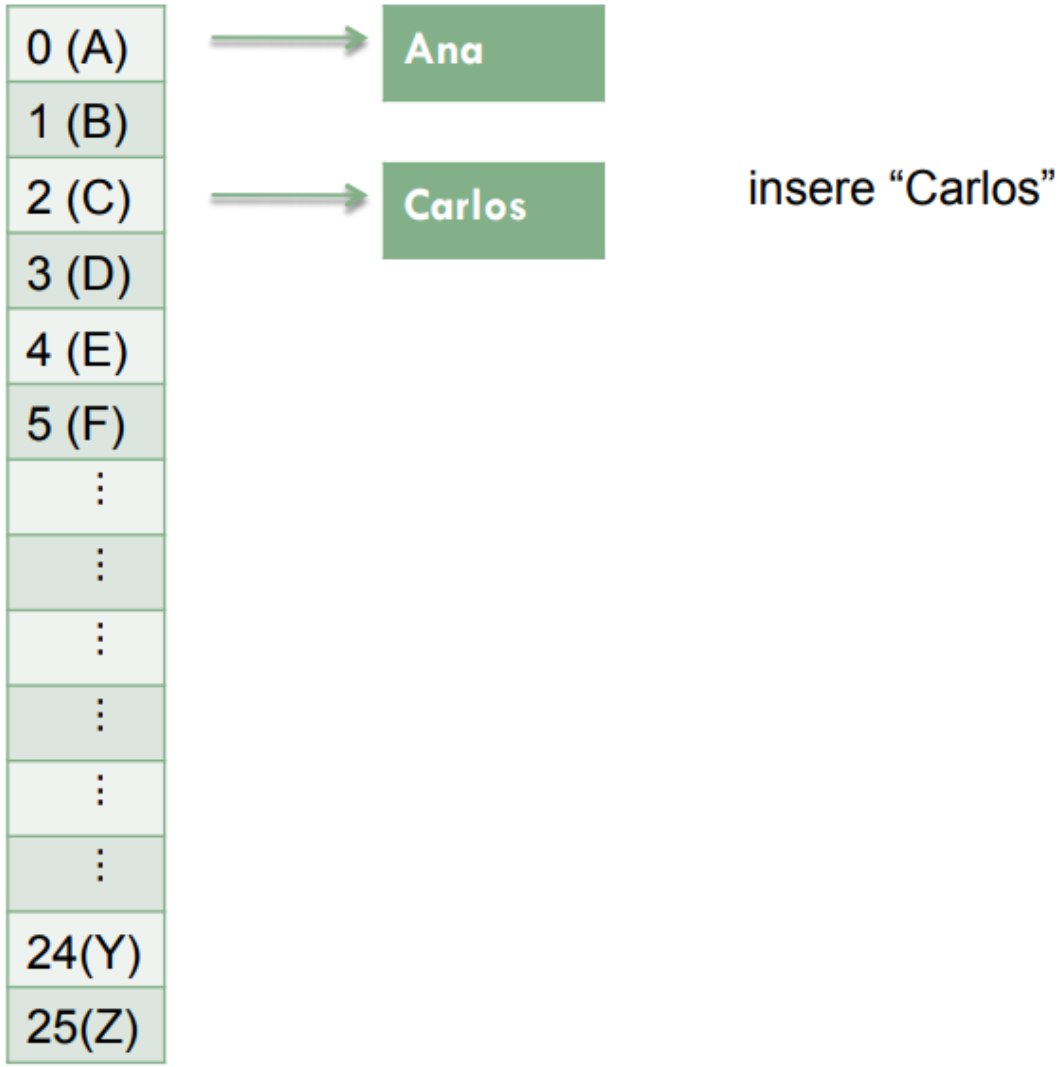
- P = 7 posições (0..6) e a função f(k) = k(código ASCII) % P.



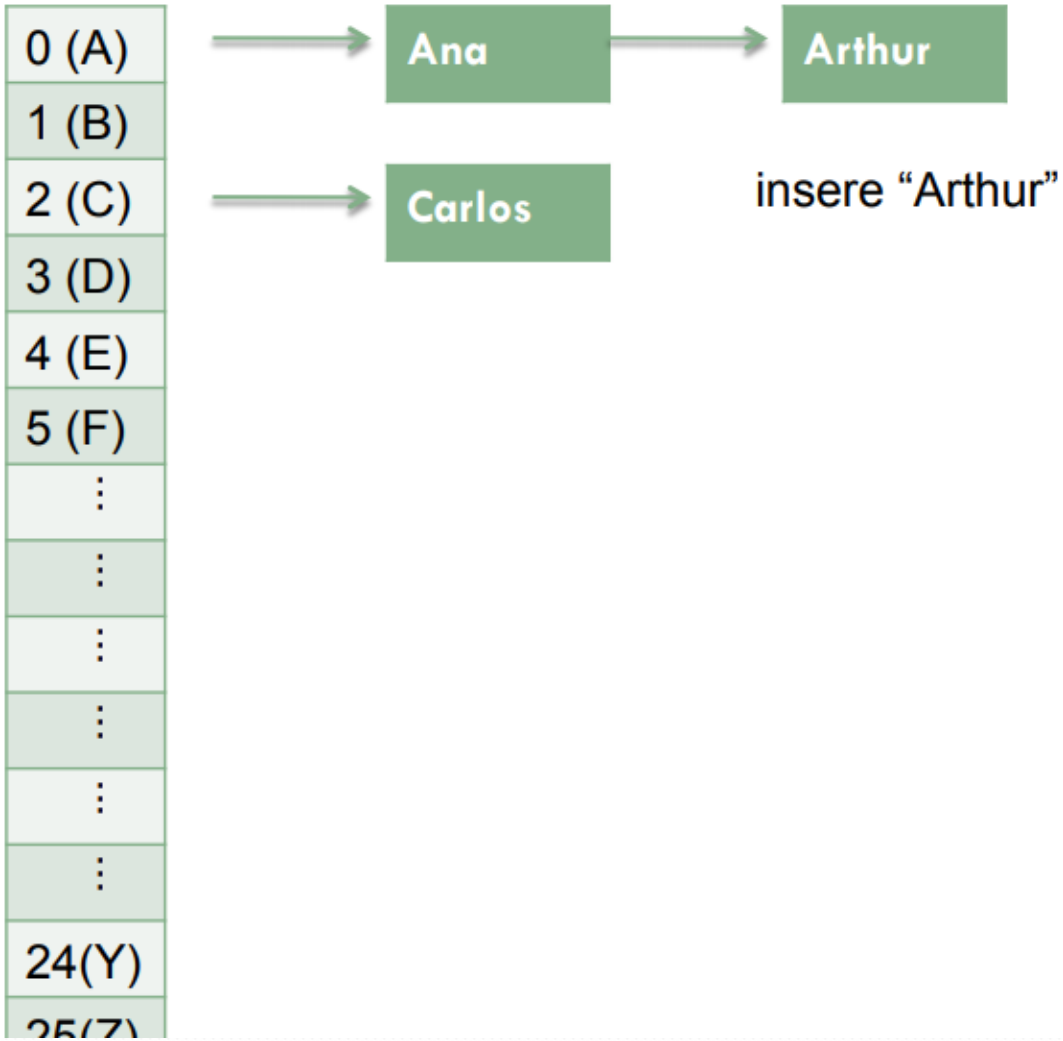
Colisões - Encadeamento



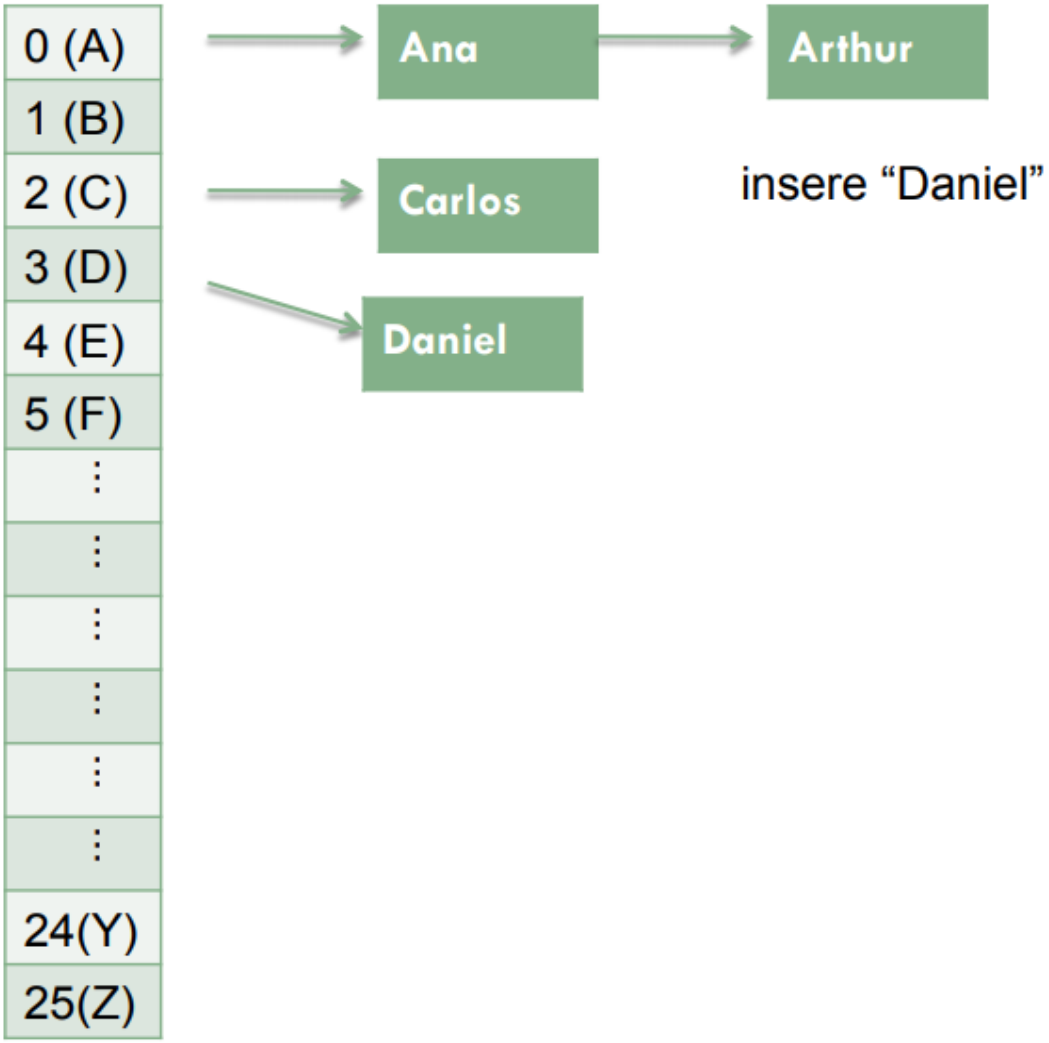
Colisões - Encadeamento



Colisões - Encadeamento



Colisões - Encadeamento



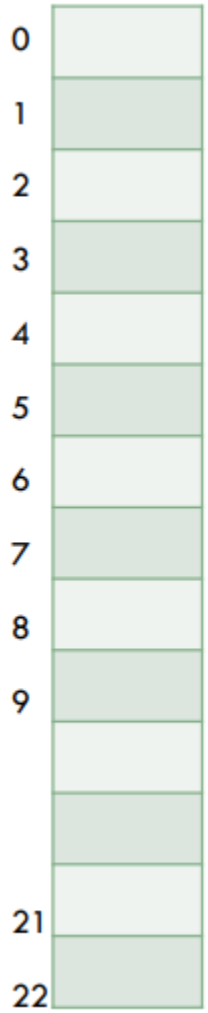
Colisões – Endereçamento Aberto

- Sem listas encadeadas
- Sem informação adicional
- Quando houver colisões – através de um cálculo qual o próximo local a ser examinado
- Sucesso: vai calculando até achar uma posição livre/ encontra a chave
- Sem sucesso: a tabela está cheia ou não se encontra a chave

Colisões – Sondagem Linear

- Ao verificar que uma posição h(k) da tabela está ocupada, tenta adicionar o elemento na primeira posição livre seguinte:
- h(k) + 1, h(k) + 2, h(k) + 3, ... até uma posição vazia, considerando a tabela circular
- **função hash h(x,k) = (h'(x) + k) mod m 0 <= k <= m-1**

Colisões – Sondagem Linear

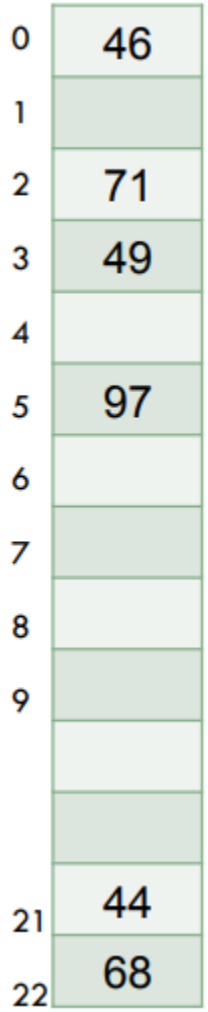


- m = 23

$h(x,k) = (h'(x) + k) \bmod m$

- Ao inserir
  - 44 → k = 0, 44 mod 23 = 21
  - 46 → k = 0, 46 mod 23 = 0
  - 49 → k = 0, 49 mod 23 = 3
  - 68 → k = 0, 68 mod 23 = 22
  - 71→ k = 0, 71 mod 23 = 3
  - 97 → k = 0, 44 mod 23 = 4

Colisões – Sondagem Linear



- m = 23

$$h(x,k) = (h'(x) + k) \bmod m$$

- Ao inserir
  - 44 → k = 0, 44 mod 23 = 21
  - 46 → k = 0, 46 mod 23 = 0
  - 49 → k = 0, 49 mod 23 = 3
  - 68 → k = 0, 68 mod 23 = 22
  - 71→ k = 0, 71 mod 23 = 3
  - 97 → k = 0, 44 mod 23 = 4

Colisões – Sondagem Linear

0	46
1	
2	71
3	49
4	26
5	97
6	
7	
8	
9	
21	44
22	68

$h(x,k) = (h'(x) + k) \bmod m$

- inserir 26
  - k = 0, 26 mod 23 = 3
  - k = 1, 26+1 mod 23 = 4

Colisões – Sondagem Linear

- Consequências:
- Cria grandes blocos de dados numa mesma região da tabela
- Dificulta a remoção de dados
- Aumenta a complexidade para a busca nos casos de colisão
- Limitado pelo tamanho da tabela

Colisões – Sondagem Quadrática

- Tentativa de se espalhar mais os elementos
- **função h(x,k) = (h'(x) + c1k + c2k2) mod m !**
- Onde c1 e c2 são constantes, c2 ≠ 0 e k = 0, ..., m-1
- exemplo: h(x,0) = h'(x) h(x,k) = (h(x, k-1) + k) mod m 0< k< m-1

Tabela de Espalhamento – Características

- Tabela de dispersão/espalhamento: um vetor que contém os itens nas posições indicadas pela função de dispersão
- Função de dispersão: função que mapeia cada chave em uma localização (índice) da tabela que contem o item
- Colisões: quando a função de dispersão mapeia mais do que uma chave para o mesmo índice
- Esquemas para tratar colisões: indicam posições diferentes para chaves envolvidas em uma colisão
- Requisitos para uma função de dispersão
- Ser fácil de computar (O(1))
- Uma boa função evita colisões
- Uma boa função tende a espalhar as chaves pelo vetor

Tabela de Espalhamento - Características

- Tabelas de espalhamento/dispersão têm tempo médio de busca constante.
- A avaliação de uma boa função de dispersão é complexa
- Os dados na memória ficam aleatoriamente distribuídos

Limitações

- Estrutura de dados do tipo dicionário, que não permite:
- Armazenar elementos repetidos
- Recuperar elementos sequencialmente (ordenação)
- Recuperar o elemento antecessor/sucessor
- Para otimizar a função de dispersão é necessário conhecer a natureza da chave a ser utilizada.
- No pior caso, a ordem das operações pode ser O(n)
- Podem necessitar de redimensionamento
- O custo da função de dispersão pode ser proibitivo

Aplicações

- Vetor associativo (cujos índices são strings arbitrárias), especialmente em linguagens interpretadas (AWK, Perl, e PHP)
- Indexação de bancos de dados (embora árvores sejam mais comuns)
- Cache de memória: memória auxiliar para acelerar o processamento
- Conjuntos (indica a existência e recupera rapidamente)
- Representação de objetos em linguagens de programação dinâmicas (Perl, Python, JavaScript, e Ruby)
- Representação única de dados (em LISP, para evitar a criação de strings repetidas)
- Etc.

Contatos

- Email: fabio.silva321@fatec.sp.gov.br
- Linkedin: https://br.linkedin.com/in/b41a5269
- Facebook: https://www.facebook.com/fabio.silva.56211