Algoritmos e Estruturas de Dados I

TABELAS HASH

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Introdução

Problema

- Princípio de funcionamento dos métodos de busca
  - Procurar a informação desejada com base na comparação de suas chaves, isto é, com base em algum valor que a compõe.

Problema

- Algoritmos eficientes necessitam que os elementos estejam armazenados de forma ordenada
- Custo ordenação melhor caso é O(N log N)
- Custo da busca melhor caso é O(log N)

Problema

- Custo da comparação de chaves é alto
- O que seria uma operação de busca ideal?
  - Seria aquela que permitisse o acesso direto ao elemento procurado, sem nenhuma etapa de comparação de chaves.
  - Nesse caso, teríamos um custo O(1).
  - Tempo sempre constante de acesso

Problema

- Uma saída é usar arrays
  - São estruturas que utilizam índices para armazenar informações.
  - Permite acessar um determinada posição com custo O(1).
- Problema
  - Arrays não possuem nenhum mecanismo que permita calcular a posição onde uma informação está armazenada.
  - A operação de busca não é O(1).

Problema

- Precisamos do tempo de acesso do array juntamente com a capacidade de busca um elemento em tempo constante.
- Solução: usar uma **tabela hash**.

Tabela Hash

Tabela Hash

- Também conhecidas como tabelas de indexação ou de espalhamento
  - É uma generalização da ideia de array.
- Ideia central
  - Utilizar uma função, chamada de **função de hashing**, para espalhar os elementos que queremos armazenar na tabela.
  - Esse espalhamento faz com que os elementos fiquem dispersos de forma não ordenada dentro do array que define a tabela.

Tabela Hash

- Exemplo:

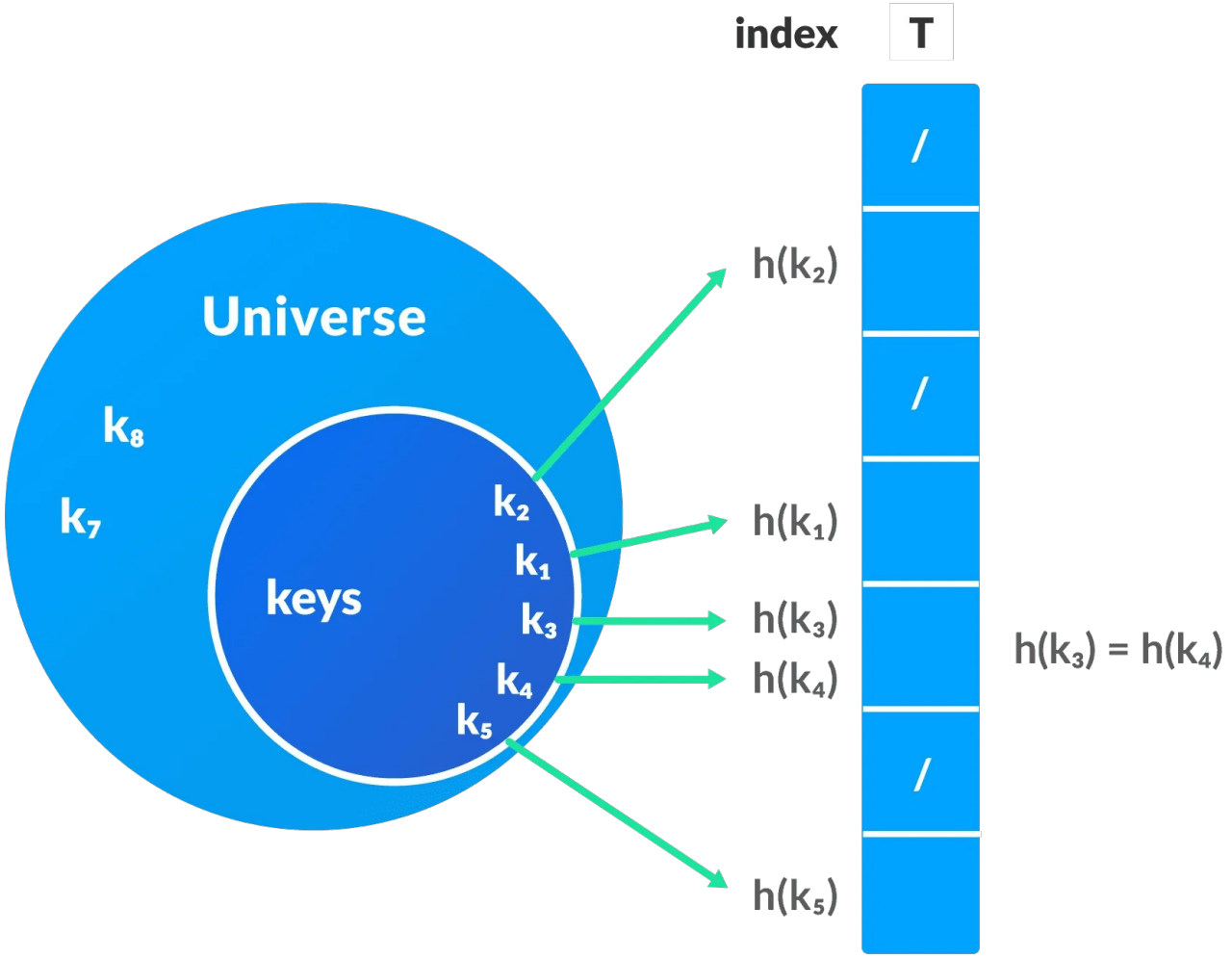


Tabela Hash

- Por que espalhar os elementos melhora a busca?
  - A tabela permite a associar valores a chaves
    - **chave**: parte da informação que compõe o elemento a ser inserido ou buscado na tabela.
    - **valor**: é a posição (índice) onde o elemento se encontra no array que define a tabela.
  - Assim, a partir de uma chave podemos acessar de forma rápida uma determinada posição do array.
    - Na média, essa operação tem custo **O(1)**.

Tabela Hash

- Vantagens
  - Alta eficiência na operação de busca
    - Caso médio é **O(1)** enquanto o da busca linear é **O(N)** e a da busca binária é **O(log N)**.
  - Tempo de busca é praticamente independente do número de chaves armazenadas na tabela.
  - Implementação *relativamente* simples.

Tabela Hash

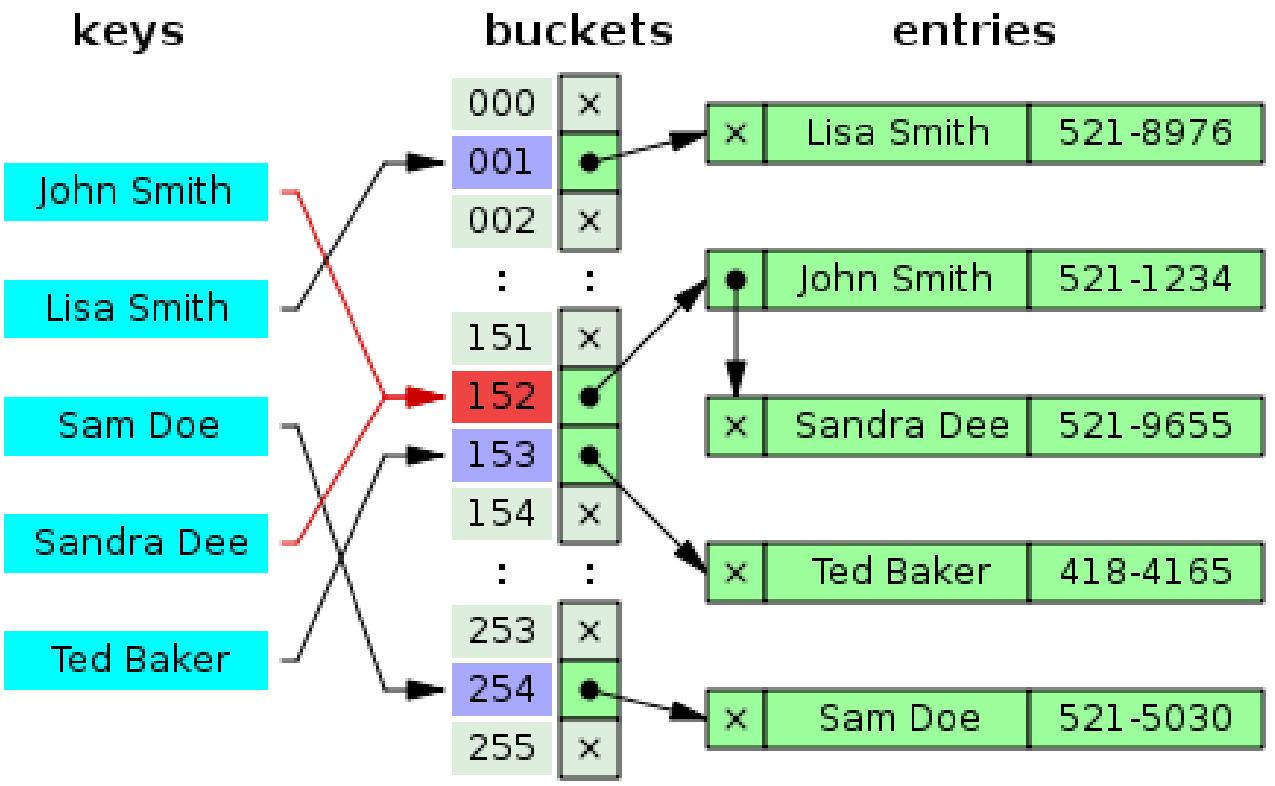
- Infelizmente, esse tipo de implementação também tem suas desvantagens:
  - Alto custo para recuperar os elementos da tabela ordenados pela chave.
    - Nesse caso, é preciso ordenar a tabela.
  - O pior caso é O(N), sendo N o tamanho da tabela.
    - Alto número de colisões.

Tabela Hash

- O que é uma colisão?
  - Uma colisão ocorre quando duas (ou mais) chaves diferentes tentam ocupar a mesma posição na tabela hash.
  - A colisão de chaves não é algo exatamente ruim, é apenas algo indesejável pois diminui o desempenho do sistema.

Tabela Hash

- Exemplo de colisão:



Aplicações

Aplicações

- A tabela hash pode ser utilizada para:
  - Busca de elementos em base de dados
    - Estruturas de dados em memória, bancos de dados e mecanismos de busca na Internet;
  - Verificação de integridade de dados e autenticação de mensagens
    - Os dados são enviados juntamente com o resultado da função de hashing.
    - Quem receber os dados recalcula a função de hashing usando os dados recebidos e compara o resultado obtido com o que ele recebeu.
    - Resultados diferentes: erro de transmissão

Aplicações

- A tabela hash pode ser utilizada para:
  - Armazenamento de senhas com segurança.
    - A senha não é armazenada no servidor, mas sim o resultado da função de hashing.
  - Implementação da tabela de símbolos dos compiladores.
  - Criptografia:
    - MD5 e família SHA (*Secure Hash Algorithm*).

Tabela Hash

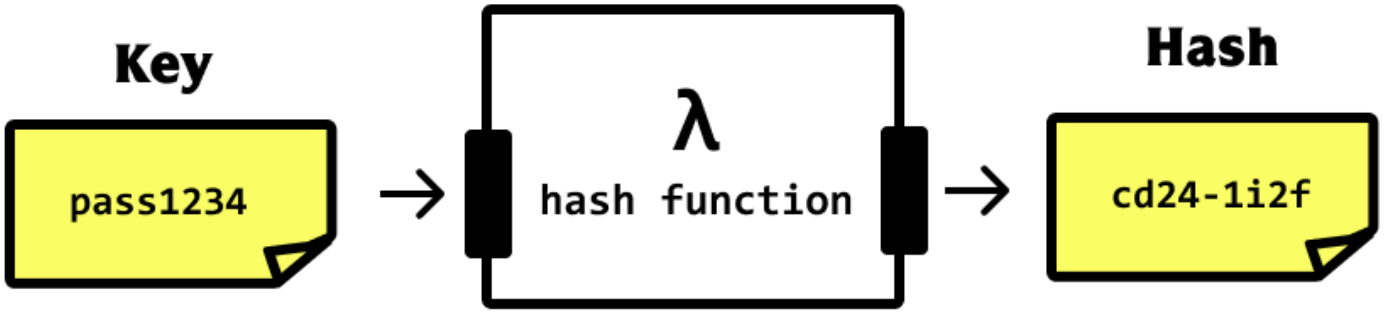
Tamanho da Tabela Hash

- O ideal é escolher um número primo e evitar valores que sejam uma potência de dois:
  - Número primo reduz a probabilidade de colisões, mesmo que a função de hashing utilizada não seja muito eficaz.
  - Potência de dois melhora a velocidade, mas pode aumentar os problemas de colisão se estivermos utilizando uma função de hashing mais simples.

Função de Hashing

Função de Hashing

- Inserção e busca: é necessário calcular a posição dos dados dentro da tabela.
- Função de Hashing.
  - Calcula a posição de uma chave escolhida a partir dos dados manipulados.

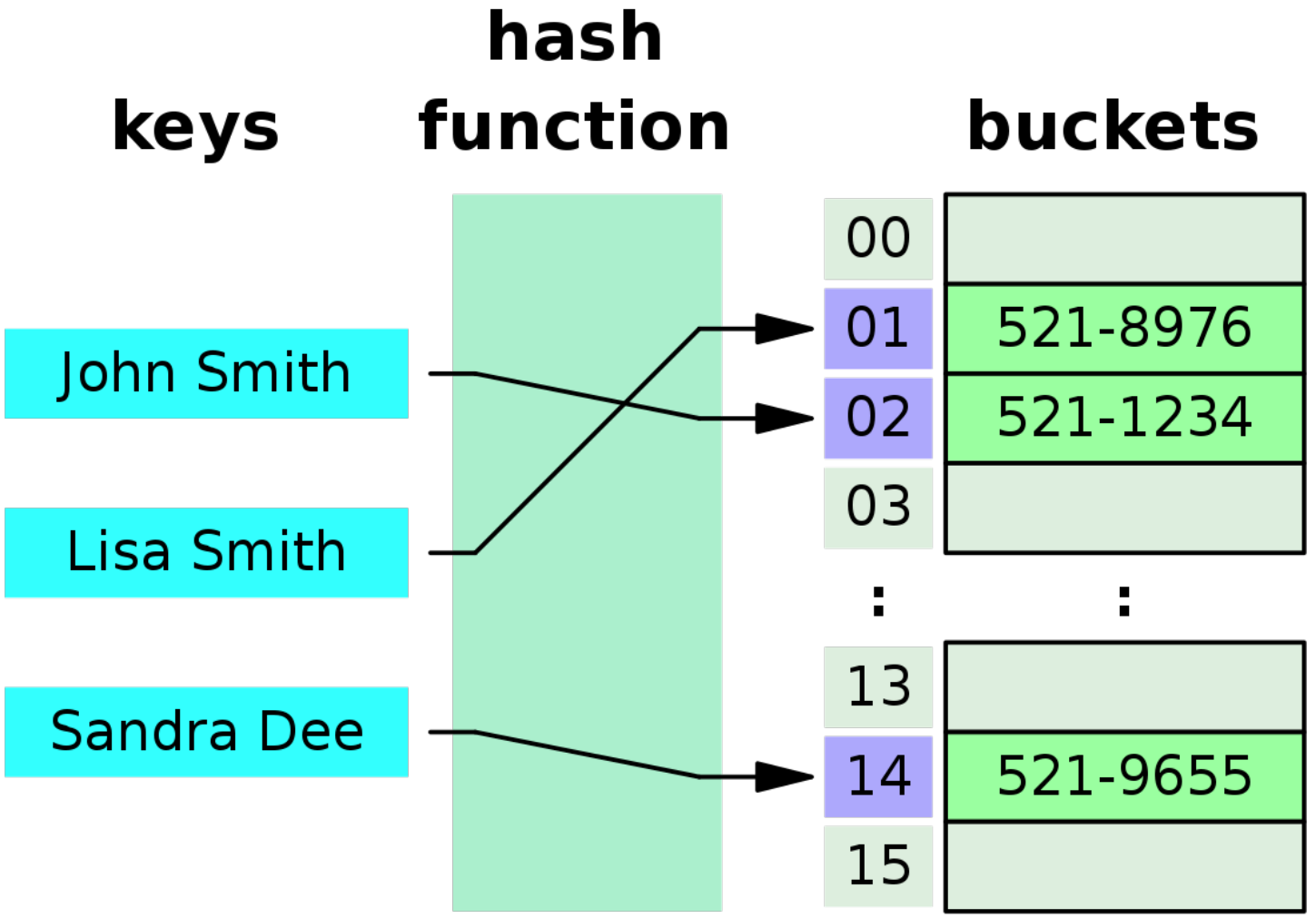


Função de Hashing

- Função de Hashing
  - É extremamente importante para o bom desempenho da tabela.
  - Ela é responsável por distribuir as informações de forma equilibrada pela tabela hash.

Função de Hashing

- Exemplo:



Função de Hashing

- Para um bom funcionamento, deve satisfazer as seguintes condições:
  - Ser simples e barata de se calcular.
  - Garantir que valores diferentes produzam posições diferentes.
  - Gerar uma distribuição equilibrada dos dados na tabela.
    - Cada posição da tabela tem a mesma chance de receber uma chave (máximo espalhamento).

Função de Hashing

- Sua implementação depende do conhecimento prévio da natureza e domínio da chave a ser utilizada.
- Exemplo: utilizar apenas os quatro dígitos do número de celular de uma pessoa para armazená-lo na tabela.
  - Neste caso, seria melhor usar os quatro últimos dígitos do que os quatro primeiros, pois os primeiros costumam se repetir com maior frequência e iriam gerar posições iguais na tabela.
  - Assim, o ideal é usar um cálculo diferente de hash para cada tipo de chave.

Função de Hashing

- Alguns exemplos de função de hashing comumente utilizadas:
  - Método da Divisão.
  - Método da Multiplicação.

Método da Divisão

- Uma função de hash precisa garantir que o valor retornado seja um índice válido para uma das células da tabela.
- A maneira mais simples é usar o módulo da divisão como h(k) = k%S, sendo K um número e S o tamanho da tabela.
- O método da divisão é bastante adequado quando se conhece pouco sobre as chaves.

Método da Divisão

- Apesar de simples, apresenta alguns problemas.
- Resto da divisão: valores diferentes podem resultar na mesma posição.
- Exemplo:
  - O resto da divisão de 11 por 10 e de 21 por 10 são o mesmo valor de posição: 1.
  - Uma maneira de reduzir esse tipo de problema é utilizar como tamanho da tabela, TABLE_SIZE, um número primo.

Método da Multiplicação

- Também chamado de método da congruência linear multiplicativo.
- Usa uma constante fracionária A, $0 < A < 1$, para multiplicar o valor da chave que representa o elemento.
- Em seguida, a parte fracionária resultante é multiplicada pelo tamanho da tabela para calcular a posição do elemento.

Método da Multiplicação

- Procedimento do método da multiplicação:
  1. Escolha uma constante de valor A tal que $0 < A < 1$.
  2. Multiplique o valor da chave por A.
  3. Extraia a parte fracionária de kA.
  4. Multiplique o resultado do passo anterior pelo tamanho M da tabela hash.
  5. O resultado do valor do hash é obtido por pegar o piso do resultado obtido no passo anterior (Passo 4).

Método da Multiplicação

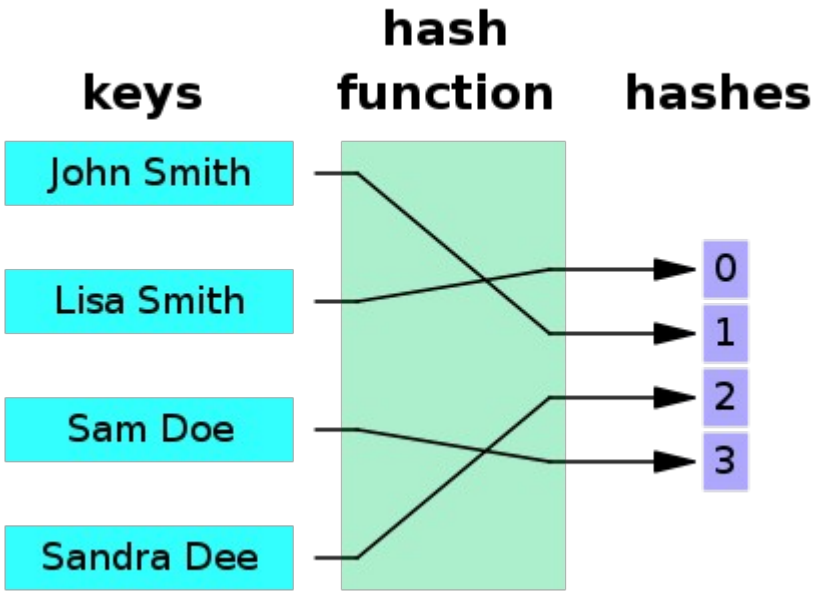
- h(K) = floor (M (kA mod 1))
  - M é o tamanho da tabela hash.
  - K é o valor da chave.
  - A é um valor constante.
    - Knuth sugere o valor da constante áurea: $A = (\sqrt{5} - 1)/2$

```
k = 12345
A = 0.357840
M = 100

h(12345) = floor[ 100 (12345*0.357840 mod 1)]
         = floor[ 100 (4417.5348 mod 1) ]
         = floor[ 100 (0.5348) ]
         = floor[ 53.48 ]
         = 53
```

Hashing Perfeito

- Se conhecermos todas as chaves antes de criar a função de hashing, é possível encontrar uma função de hashing injetora:
  - isto é, não temos colisões.
  - tais funções podem ser difíceis de encontrar.
- Exemplo:



Hashing Universal

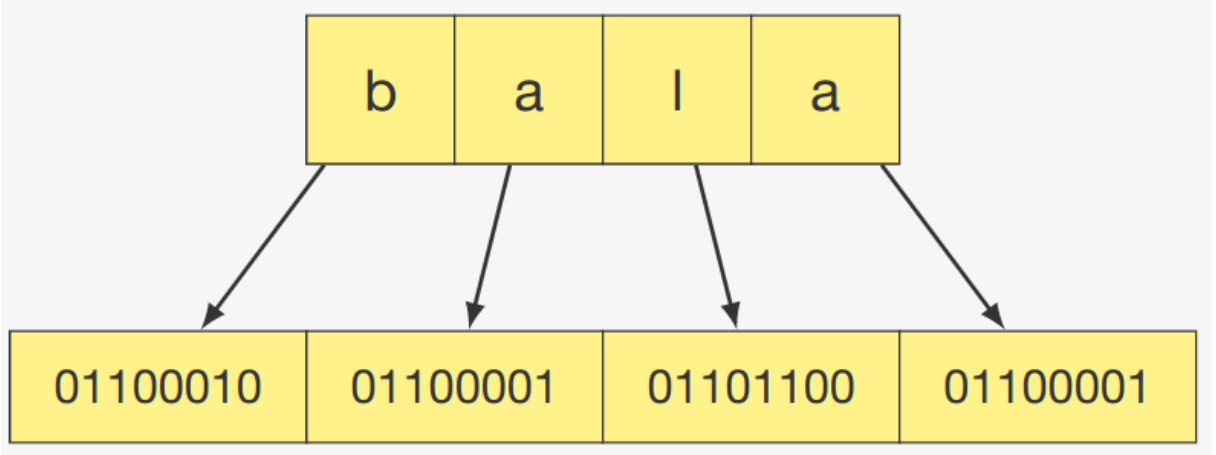
- Função de hashing está sujeita ao problema de gerar posições iguais para chaves diferentes.
  - Por se tratar de uma função determinística, ela pode ser manipulada de forma indesejada.
  - Conhecendo a função de hashing, pode-se escolher as chaves de entrada de modo que todas colidam, diminuindo o desempenho da tabela na busca para O(N).

Hashing Universal

- Hashing universal é uma estratégia que busca minimizar esse problema de colisões:
  - Basicamente, devemos escolher aleatoriamente (em tempo de execução) a função de hashing que será utilizada.
  - Para tanto, construímos um conjunto (ou família) de funções de hashing.

Chaves Não Numéricas

- Pressupomos que as chaves são números inteiros
  - E se não for?
- Reinterpretamos a chave como uma sequência de bits
- Exemplo:



- Assim, “bala” se torna o número 1.650.551.905

Tratamento de Colisões

Fator de Carga

- O fator de carga de uma tabela hash é F = n/m, onde n é o número de elementos armazenados na tabela de tamanho m.
  - O número de colisões cresce rapidamente quando o fator de carga aumenta.
  - Uma forma de diminuir as colisões é diminuir o fator de carga.
  - Mas isso não resolve o problema: colisões sempre podem ocorrer.
  - Como tratar as colisões?

Colisões

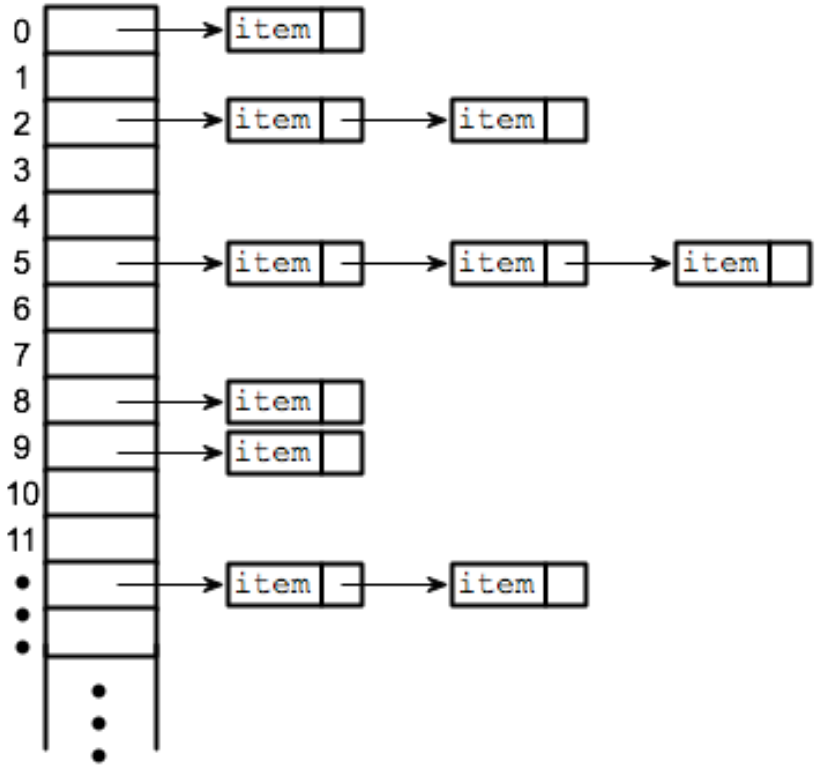
- Tratamento de colisões:
  - Por Encadeamento
  - Por Endereçamento Aberto

Encadeamento

- Manter m listas encadeadas, uma para cada possível endereço base.
- A tabela base não possui nenhum elemento, apenas os ponteiros para as listas encadeadas.
- A tabela base não armazena nenhum registro.

Encadeamento

- Cada nó da lista encadeada contém:
  - um elemento;
  - um ponteiro para o próximo nó (elemento);
- Exemplo:



Encadeamento

- Procedimento de busca:
  - Calcular o endereço aplicando a função de hash h(x);
  - Percorrer a lista encadeada associada ao endereço;
  - Comparar a chave de cada nó da lista encadeada com a chave x, até encontrar o nó (item) desejado;
  - Se o final da lista for atingido, o elemento não está na tabela;

Encadeamento

- Procedimento de inserção:
  - Calcular o endereço aplicando a função de hash h(x);
  - Buscar o elemento na lista associada ao endereço h(x);
  - Se registro for encontrado, sinalizar erro;
  - Se o registro não for encontrado, inserir no final da lista;

Análise de Complexidade

- Encadeamento (pior caso):
  - É necessário percorrer uma lista encadeada até o final para concluir que a chave não está na tabela.
  - Comprimento de uma lista encadeada pode ser O(n).
  - Complexidade no pior caso: O(n).

Análise de Complexidade

- Encadeamento (caso médio):
  - Assume-se que a função hash é uniforme.
  - Número médio de comparações feitas na busca sem sucesso é igual ao fator de carga da tabela F = n/m.
  - Número médio de comparações feitas na busca com sucesso também é igual a F = n/m.
  - Se assumirmos que o número de chaves n é proporcional ao tamanho da tabela m, então:
    - F = n/m = O(1)
    - Complexidade constante!

Encadeamento

- Procedimento de remoção:
  - Calcular o endereço aplicando a função h(x);
  - Buscar o elemento na lista associada ao endereço h(x);
  - Se o elemento for encontrado, excluir o elemento;
  - Se o elemento não for encontrado, sinalizar erro;

Endereçamento Aberto

- Motivação: a abordagem anterior utiliza ponteiros nas listas encadeadas.
  - Aumento no consumo de espaço
- Alternativa: armazenar apenas os elementos, sem os ponteiros.
- Quando houver colisão, determina-se, por cálculo de novo endereço, o próximo compartimento a ser examinado.

Funcionamento

- Para cada chave x, é necessário que todas as posições da tabela possam ser examinadas.
- A função h(x) deve fornecer, ao invés de um único endereço, um conjunto de m endereços base.
- Nova forma da função: h(x,k), onde k = 0, ..., m-1.
- Para se encontrar a chave x deve-se tentar o endereço base h(x,0).
- Se estiver ocupado com outra chave, tentar h(x,1), e assim sucessivamente.

Tratamento de Colisões

- A sequência h(x,0), h(x,1), ..., h(x, m-1) é denominada sequência de tentativas.
- A sequência de tentativas é uma permutação do conjunto {0, m-1}.
- Portanto: para cada chave x a função h deve ser capaz de fornecer uma permutação de endereços base.

Função de Hash

- Exemplos de funções hash para gerar sequência de tentativas:
  - Sondagem Linear
  - Sondagem Quadrática
  - Dispersão Dupla

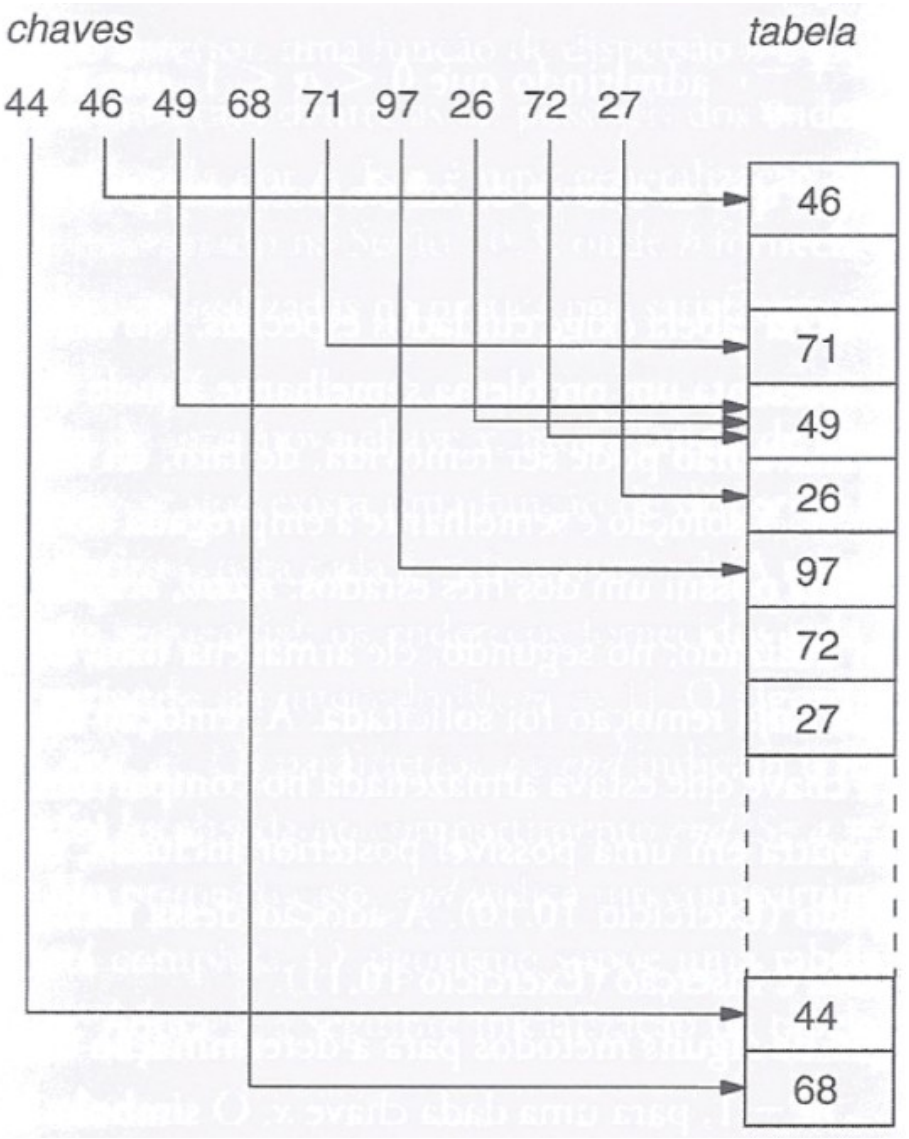
Sondagem Linear

- Suponha que o endereço base de uma chave x é h’(x).
- Suponha que já existe uma chave y ocupando o endereço h’(x).
- Ideia: tentar armazenar x no endereço consecutivo a h’(x). Se já estiver ocupado, tenta-se o próximo e assim sucessivamente.
- Considera-se uma tabela circular:
  - $h(x, k) = (h'(x) + k) \bmod m, 0 \leq k \leq m-1$

Sondagem Linear

- Exemplo:

Deve-se observar a ocorrência de colisões a partir da inserção do elemento 26.



Sondagem Linear

- Na presença de remoções, a inserção precisa que a busca percorra toda a tabela até ter certeza de que o elemento procurado não existe.
- Em situações onde não há remoção, a busca pode parar assim que encontrar uma posição livre (se a chave existisse, ela estaria ali).

Sondagem Linear

- Ponto Fraco
  - Suponha um trecho de j posições consecutivas ocupadas (chama-se agrupamento primário) e uma posição l vazia imediatamente seguinte a essas posições.
  - Suponha que uma chave x precisa ser inserida em uma das j posições:
    - x será armazenada em l
    - isso aumenta o tamanho do agrupamento primário para j + 1
    - Quanto maior for o tamanho de um agrupamento primário, maior a probabilidade de aumentá-lo ainda mais mediante a inserção de uma nova chave

Sondagem Quadrática

- Para mitigar a formação de agrupamentos primários, que aumentam muito o tempo de busca:
  - Obter sequências de endereços para endereços-base próximos, porém diferentes.
  - Utilizar como incremento uma função quadrática de k
  - $h(x,k) = (h'(x) + c1.k + c2.k2) \bmod m$, onde c1 e c2 são constantes, $c2 \neq 0$ e $k = 0, ..., m-1$

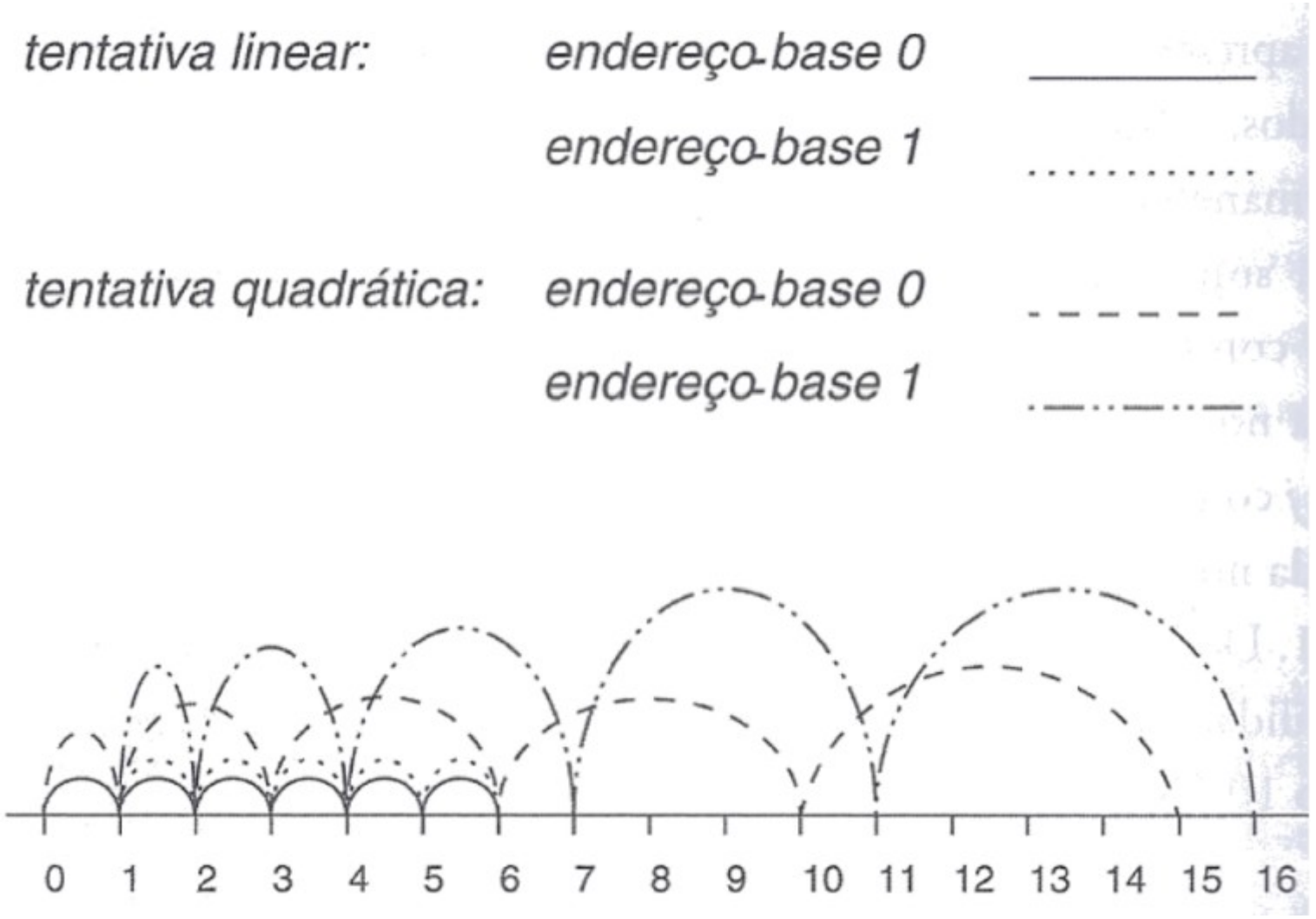
Sondagem Quadrática

- Método evita agrupamentos primários
- Mas... se duas chaves tiverem a mesma tentativa inicial, vão produzir sequências de tentativas idênticas: agrupamento secundário.

Sondagem Quadrática

- Valores de m, c1 e c2 precisam ser escolhidos de forma a garantir que todos os endereços-base serão percorridos:
- Exemplo:
  - h(x,0) = h’(x)
  - h(x,k) = (h(x,k-1) + k) mod m, para 0 < k < m
  - Essa função varre toda a tabela se m for potência de 2

Linear x Quadrática



Dispersão Dupla

- Utiliza duas funções de hash, h’(x) e h’’(x)
- h(x,k) = (h’(x) + k.h’’(x)) mod m, para 0 ≤ k < m
- Método distribui melhor as chaves do que os dois métodos anteriores
  - Se duas chaves distintas x e y são sinônimas (h’(x) = h’(y)), os métodos anteriores produzem exatamente a mesma sequência de tentativas para x e y, ocasionando concentração de chaves em algumas áreas da tabela
  - No método da dispersão dupla, isso só acontece se h’(x) = h’(y) e h’’(x) = h’’(y)

Exercícios

Exercícios

- Explique como uma tabela hash pode atingir uma complexidade de busca O(1) no caso médio. Quais fatores podem afetar esse desempenho?
- Descreva o que é uma colisão em uma tabela hash e explique duas técnicas comuns para tratá-las.
- Como o fator de carga afeta o desempenho de uma tabela hash?
- Diferencie hashing perfeito de hashing universal. Dê um exemplo de cada.
- Por que escolher um número primo como tamanho da tabela pode ser benéfico para o desempenho da tabela hash?
- Explique o conceito de sondagem linear e quadrática e discuta suas vantagens e desvantagens.

Exercícios

- Explique como uma tabela hash utiliza a função de hashing para armazenar e buscar elementos de forma eficiente.
- Compare a eficiência da busca em tabelas hash com a busca em arrays e listas encadeadas.
- Descreva as duas principais estratégias de tratamento de colisões em tabelas hash.
- Explique a diferença entre o encadeamento e o endereçamento aberto.
- O que é uma função de hashing perfeita? Como ela difere de uma função de hashing universal?
- Explique o conceito de fator de carga em uma tabela hash. Como ele afeta o desempenho da tabela?
- Quais são as condições necessárias para uma boa função de hashing?
- Discuta como a sondagem quadrática minimiza a formação de agrupamentos primários.

Exercícios

- Implemente um exemplo de função de hashing usando o método da multiplicação sugerido por Knuth.
- Qual é a ordem de complexidade de encontrar o maior elemento (maior chave) em uma estrutura de dados do tipo *hash*? Justifique a sua resposta.
- Dado o conjunto de elementos S = {42,39,57,3,18,5,67,13,70,26} e uma tabela hash T de tamanho 13 e com uma função de hash H = X mod 13. Mostre a inserção dos elementos de S em T usando o tratamento de colisões por encadeamento e por endereçamento aberto.

Exercícios

- Implementar uma função de hash básica que use o método da divisão. Teste essa função com chaves de diferentes tipos e observe como as colisões são tratadas.

```
def hash_division(key, table_size):
    return key % table_size
```

Exercícios

- Perguntas:
  - Explique como a função simple_hash calcula o índice de um dado key.
  - Qual seria a saída da função simple_hash se key fosse 1024 e size fosse 10? Justifique sua resposta.
  - A função simple_hash é uma boa função de hashing? Por quê? Discuta suas vantagens e desvantagens.

```
def simple_hash(key, size):
    return key % size

print(simple_hash(123, 10))  # Saída esperada: 3
print(simple_hash(456, 10))  # Saída esperada: 6
print(simple_hash(789, 10))  # Saída esperada: 9
```