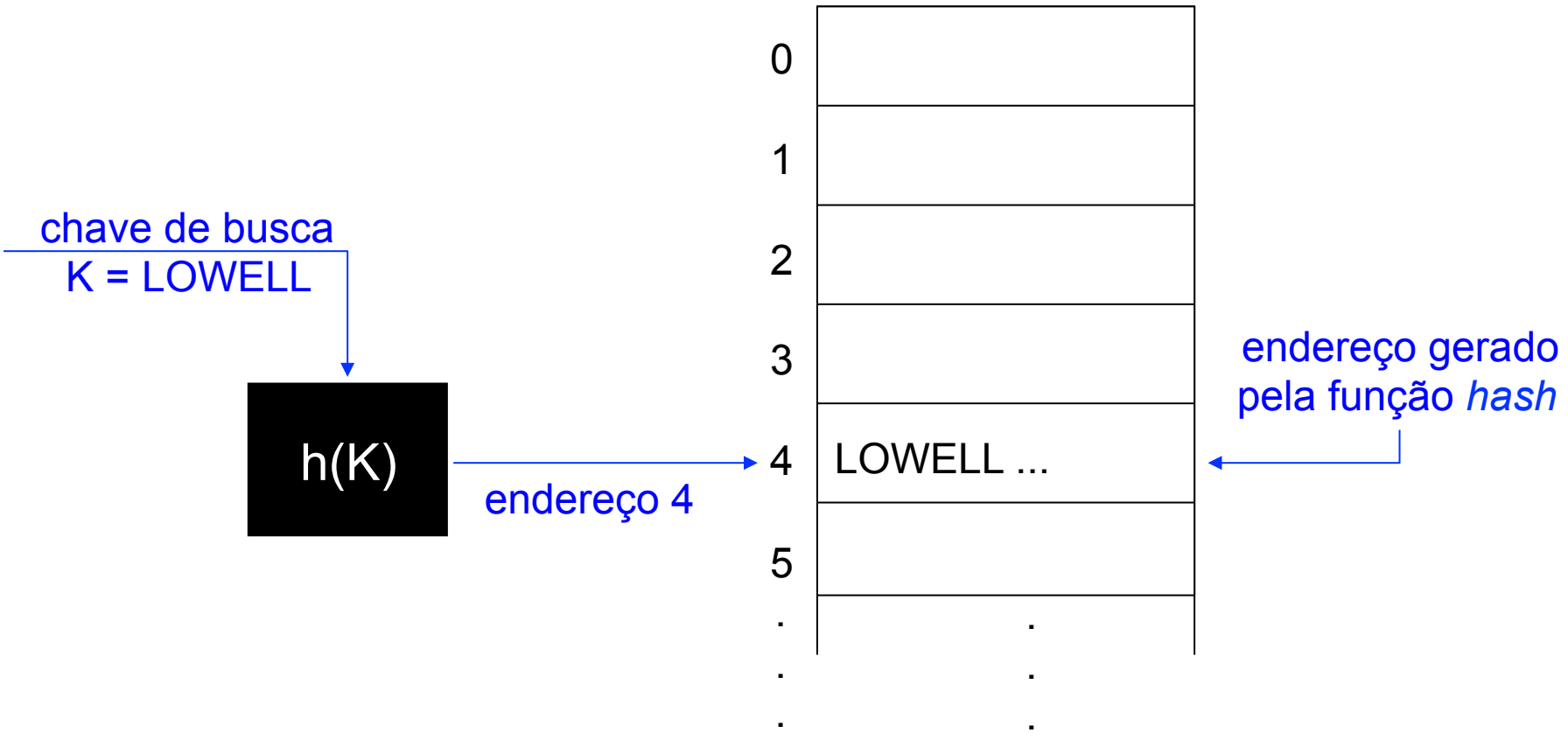
Hashing

Profa. Dra. Cristina Dutra de Aguiar Ciferri

Hashing

- Função *hash*
  - caixa preta que produz um endereço toda vez que uma chave de busca é passada como parâmetro
- Endereço resultante
  - usado para o armazenamento e a recuperação de registros no arquivo de dados
- Nomenclatura
  - h(K) → endereço
    - K: chave de busca

Hashing


chave de busca
K = LOWELL
h(K)
endereço 4
0
1
2
3
4
LOWELL ...
5
endereço gerado
pela função *hash*
espaço de endereçamento: registros de tamanho fixo

Hashing versus Indexação

- Semelhança
  - ambos envolvem a associação de uma chave de busca a um endereço de registro
- Diferença (*hashing*)
  - endereço gerado é aleatório
    - não existe relação óbvia entre a chave e a localização do registro no arquivo de dados
  - duas chaves diferentes podem ser transformadas para o mesmo endereço

COLISÃO

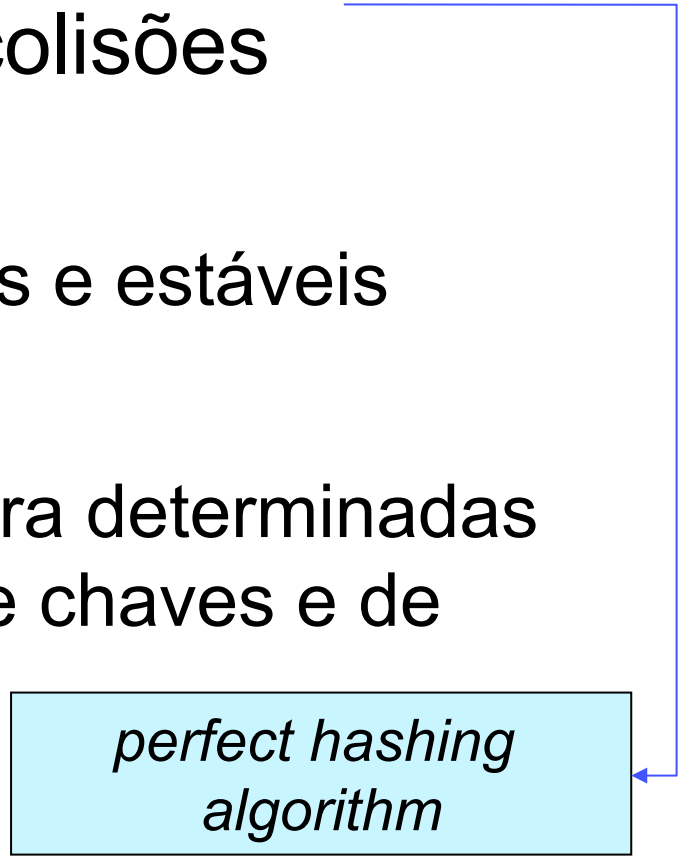
Exemplo de Colisão

nome	código ASC II 1ª e 2ª letras	produto	endereço gerado
BALL	66 65	66 x 65 = 4.290	290
LOWELL	76 79	76 x 79 = 6.004	004
TREE	84 82	84 x 82 = 6.888	888
OLIVER	79 76	79 x 76 = 6.004	004

chaves sinônimas: LOWELL e OLIVER

Colisão: Solução 1

- Encontrar um algoritmo de *hashing* perfeito que não produza colisões
- Cenário de uso
  - conjunto de dados pequenos e estáveis
- Limitação
  - abordagem não indicada para determinadas configurações de número de chaves e de dinâmica dos dados



Colisão: Solução 2

- Encontrar um algoritmo de *hashing* que produza poucas colisões
- Objetivo
  - evitar o agrupamento de registros em certos endereços
- Funcionalidade
  - espalhar os registros aleatoriamente no espaço disponível para armazenamento
  - distribuir o mais uniformemente possível

Colisão: Solução 3

- Ajustar a forma de armazenamento dos registros
- Possibilidade 1: uso de memória extra
  - aumentar o espaço de endereçamento para um mesmo conjunto de registros
  - cenário de uso
    - poucos registros para serem distribuídos entre muitos endereços

Colisão: Solução 3

- É muito fácil encontrar um algoritmo *hash* que evita colisões se existem poucos registros para serem distribuídos entre muitos endereços
- É muito mais difícil encontrar um algoritmo *hash* que evita colisões quando o número de registros e de endereços é aproximadamente o mesmo

Colisão: Solução 3

- Possibilidade 1: uso de memória extra
  - complexidade de espaço
    - perda de espaço de armazenamento
- Exemplo
  - registros: 75
  - espaço de endereçamento: 1.000
  - alocado *versus* usado = 7,5%
  - alocado *versus* não usado = 92,5%

Colisão: Solução 3

- Possibilidade 2: armazenamento de mais de um registro em um único endereço
  - uso de *buckets*
    - cada endereço é suficientemente grande para armazenar diversos registros
  - exemplo
    - registros de 80 bytes
    - *bucket* de 512 bytes
  - complexidade de espaço
    - perda de espaço para registros sem sinônimos

cada endereço pode armazenar até 6 registros!

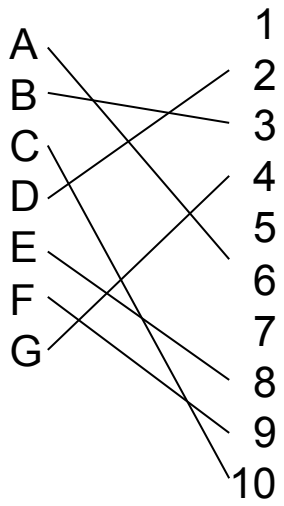
Colisão: Solução 3

- Para as possibilidades 1 e 2
  - soluções convencionais para o tratamento da colisão
    - overflow progressivo
    - *hashing* duplo
    - encademanto

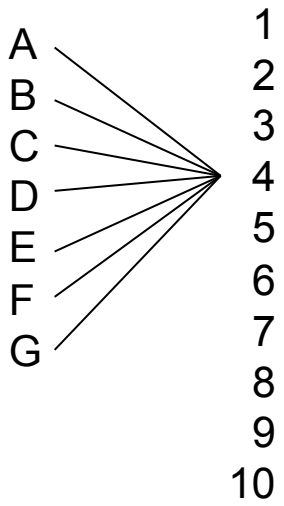
Distribuição de Registros

- Como uma função *hash* distribui (espalha) os registros no espaço de endereços?

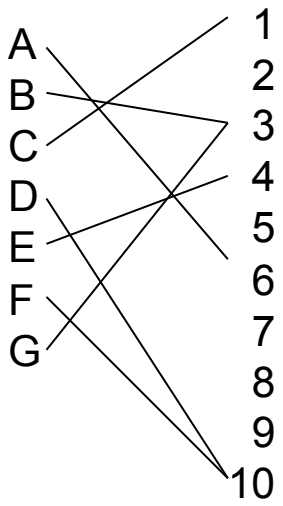
(a) melhor caso



(b) pior caso



(c) caso aceitável



7 registros x 10 endereços

Distribuição Uniforme (a)

- Os registros são espalhados uniformemente entre os endereços
- Características
  - sem colisão
  - muito difícil de ser obtida

Distribuição Aleatória (c)

- Os registros são espalhados no espaço de endereços com algumas colisões
- Propriedades (função randômica)
  - para uma certa chave, todos os endereços possuem a mesma probabilidade de serem escolhidos
  - a probabilidade de um endereço ser escolhido por uma outra chave não varia em função deste endereço já ter sido escolhido
  - na geração de um grande número de endereços, alguns são gerados mais frequentemente do que outros endereços

Outros Métodos de *Hash*

- Pegar o resto da divisão da chave pelo tamanho do espaço disponível
- Examinar as chaves em busca de um padrão
- Segmentar a chave em diversos pedaços e depois fundir os pedaços
- Dividir a chave por um número
- Elevar a chave ao quadrado e pegar o meio
- Transformar a base

Categorias

- Hashing estático
  - garante acesso O(1) para arquivos estáticos
- Hashing dinâmico
  - extensão do *hashing* estático para tratar arquivos dinâmicos, ou seja, arquivos que sofrem muitas inserções e remoções
  - estratégias: extensível, dinâmico, linear, etc.

o tamanho do espaço de endereçamento (número de *buckets*) pode aumentar