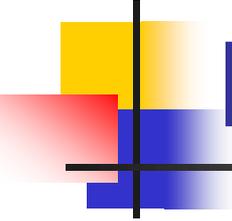


Hashing externo

Graça Nunes

Fonte: Folk & Zoelick, File Structures



Hashing convencional...

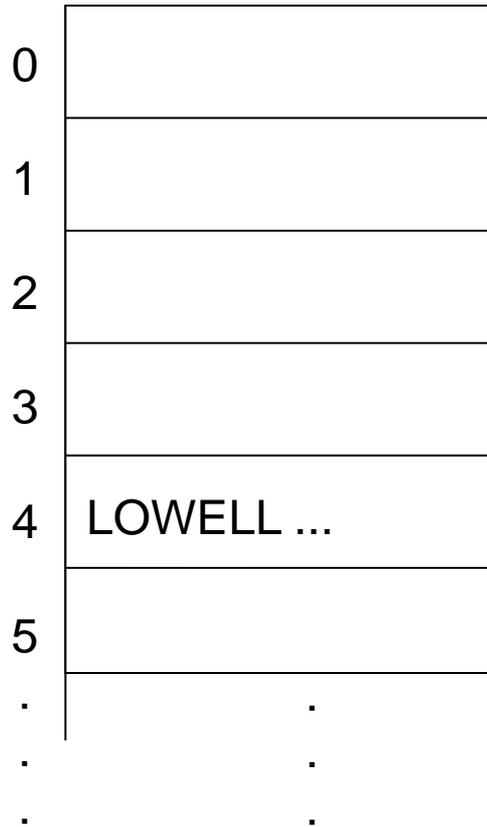
- Revisão...

Hashing

chave de busca
 $K = \text{LOWELL}$

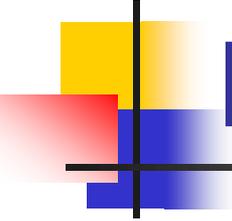
$h(K)$

endereço 4



endereço gerado
pela função *hash*

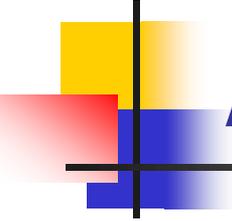
espaço de endereçamento: registros de tamanho fixo



Exemplo de espalhamento

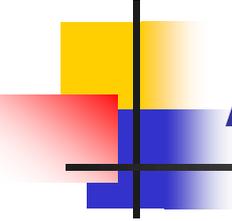
TABLE 10.1 A simple hashing scheme

Name	ASCII Code for First Two Letters	Product	Home Address
BALL	66 65	$66 \times 65 = 4,290$	290
LOWELL	76 79	$76 \times 79 = 6,004$	004
TREE	84 82	$84 \times 82 = 6,888$	888



Hashing

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



Hashing versus Indexação

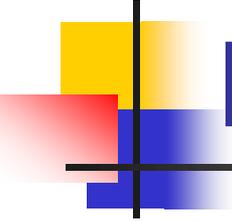
- Semelhança

- ambos envolvem associação de uma chave de busca a um endereço de registro

- Diferença (*hashing*)

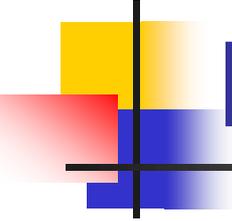
- endereço gerado é aleatório
 - não existe relação óbvia entre a chave e a localização do registro no arquivo de dados
- duas chaves diferentes podem ser transformadas para o mesmo endereço

colisão



Exemplo de Colisão

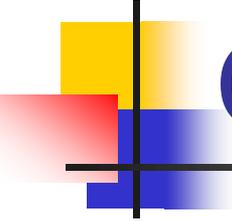
nome	código ASC II 1ª e 2ª letras	produto	endereço gerado
BALL	66 65	$66 \times 65 = 4.290$	290
LOWELL	76 79	$76 \times 79 = 6.004$	004
TREE	84 82	$84 \times 82 = 6.888$	888
OLIVER			



Exemplo de Colisão

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

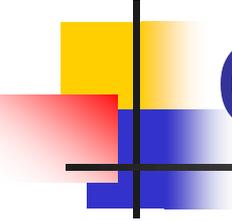
chaves sinônimas: LOWELL e OLIVER



Colisão: Solução 1

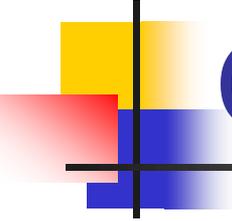
- Encontrar um algoritmo de *hashing* perfeito que não produza colisões
- Cenário de uso
 - conjunto de dados pequenos e estáveis
- Limitação
 - abordagem não indicada para determinadas configurações de número de chaves e de dinâmica dos dados

*perfect hashing
algorithm*



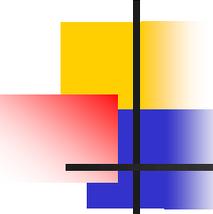
Colisão: Solução 2

- 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



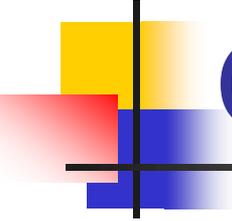
Colisão: Solução 3

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



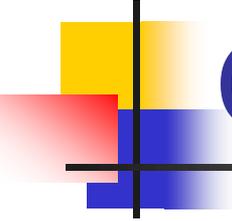
Colisão: Solução 3

- 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: armazenar 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

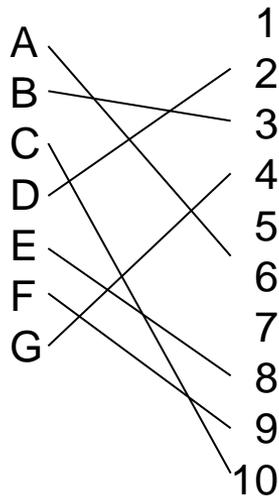
cada endereço pode armazenar até 6 registros!

Distribuição de Registros

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

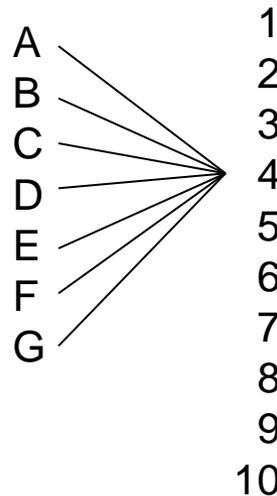
(a) melhor caso

registro endereço



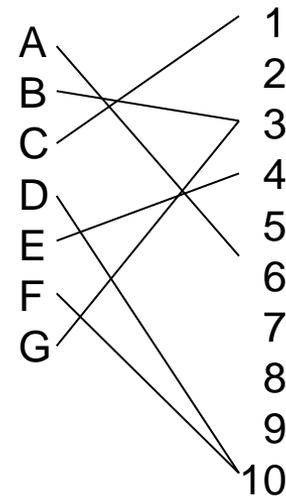
(b) pior caso

registro endereço

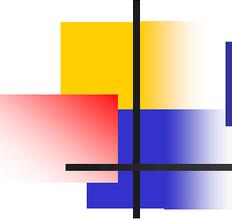


(c) caso aceitável

registro endereço

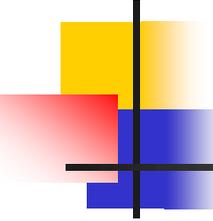


7 registros x 10 endereços



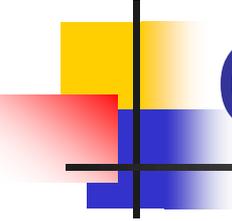
Distribuição Uniforme (a)

- Registros espalhados uniformemente entre os endereços
- Características
 - sem colisão
 - muito difícil de ser obtida



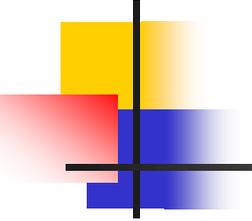
Distribuição Aleatória (c)

- Os registros 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 endereços são gerados mais frequentemente que outros



Outros Métodos de *Hash*

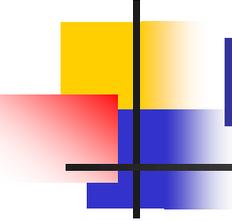
- Examinar as chaves em busca de um padrão
- Segmentar a chave em diversos pedaços e depois fundir os pedaços
- Dividir a chave por um número
- Elevar a chave ao quadrado e pegar o meio
- Transformar a base



Categorias de Hashing

- Hashing estático: garante acesso $O(1)$, para arquivos estáticos
- Hashing dinâmico: extensão do *hashing* estático para tratar arquivos dinâmicos

Organização de índices hashing



- único arquivo
 - Os dados e o índice *hashing* ficam no mesmo arquivo
- dois arquivos
 - Os dados ficam em um arquivo e o índice hashing das chaves fica em outro

Hashing Externo

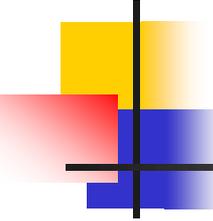
chave de busca
 $K = \text{LOWELL}$

$h(K)$

endereço 4

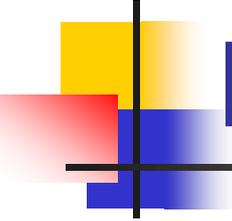
0		
1		
2		
3		
4	LOWELL	RRN
5		
.	.	
.	.	
.	.	

do registro no
arq. dados



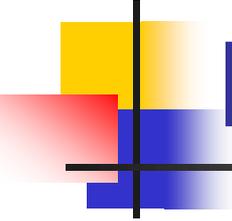
Possibilidades

- Índice cabe em RAM
 - Compreendendo tabela + espaço de tratamento de overflow (*buckets*, p.ex.)
 - *Seek* ocorre após obtenção do RRN em RAM; $O(1)$
- Índice não cabe na RAM
 - Como então gerar endereços fixos de tabela???



Problema

- Se arquivo de dados é estático (não muda após as inserções):
 - Basta encontrar a melhor função hash e o espaço de endereçamento que mais espalha as chaves
- Se o arquivo é dinâmico
 - O espaço de endereçamento inicialmente adequado pode ser insuficiente após algumas operações de inserção



Hashing Extensível

- **Espalhamento Extensível** (*Extendible Hashing*):
permite um auto-ajuste do espaço de endereçamento do espalhamento
 - Maior o número de chaves, maior o número de endereços
- Idéia chave é combinar o espalhamento convencional com uma técnica de recuperação de informações denominada **trie**