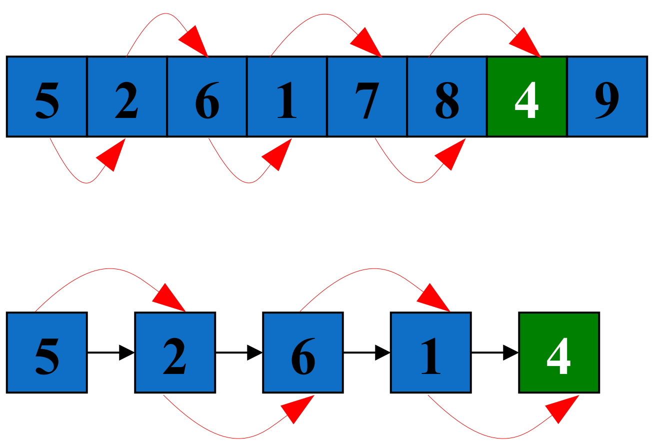
Hashing

Estrutura de Dados II

Jairo Francisco de Souza

Pesquisa sequencial

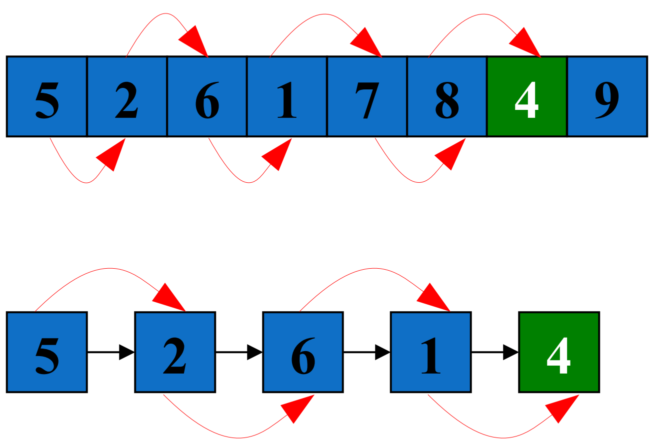
- Procedimento
  - Elementos pesquisados sucessivamente
  - Comparação determina se o elemento foi encontrado ou não
  - Exemplo: buscar 4 (Arrays e lista encadeada)
  - Importância no número médio de comparações



Pesquisa sequencial

- Número médio de comparações

$$\text{Comparações} = \frac{1 + 2 + 3 + \dots n}{n} = \frac{n\,((n+1)/2)}{n} = \frac{n+1}{2}$$

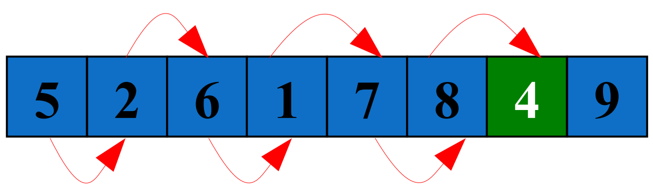


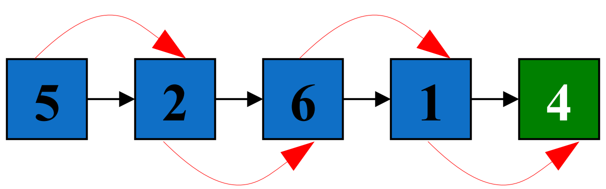
Tempo de busca:
O(n)
Comparações: (n + 1) / 2

Busca sequencial

- Pesquisa sequencial em vetor não ordenado
  - Necessário pesquisar em todo o vetor

```
for i := 1 to tamanho_vetor {
    if vetor[i] == chave then retorna chave
}
```





Tempo de busca:
O(n)

Comparações: (n + 1) / 2

Busca sequencial

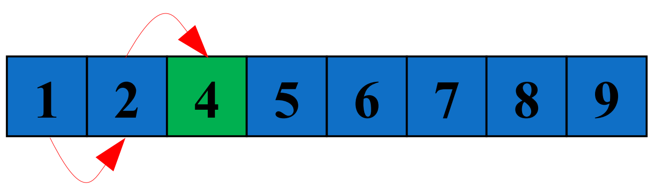
- Pesquisa sequencial em vetor ordenado
  - Caso encontre um número maior que o buscado, pare.

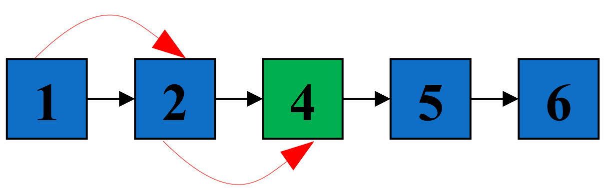
for i := 1 to tamanho_vetor {

    if vetor[i] > chave then return -1

    elseif vetor[i] == chave then retorna chave

}





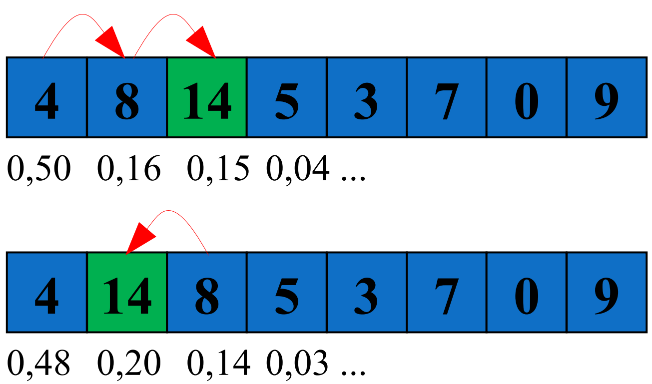
Ainda assim...

Tempo de busca: O(n)

Comparações: (n + 1) / 2

Busca sequencial usando frequência

- E se alguns valores foram mais acessados que outros?
  - Neste caso, podemos ordenar pela frequência de recuperação que uma chave possui. Ou seja, a cada busca de uma chave, aumentamos sua frequência e ordenamos pela frequência…



Ainda assim...

Tempo de busca: O(n)

Busca sequencial usando frequência

- Número de comparações usando frequência
- Exemplo:

Entrada	FRA
1	0,50
2	0,30
3	0,15
4	0,04
5	0,01

Número médio de endereços examinados para a localização de uma entrada, considerando-se que elas aparecem nessa seqüência dentro da tabela:

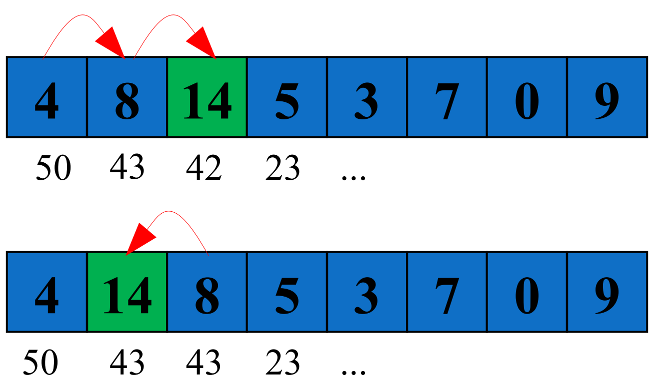
$$EE = 1*0{,}50 + 2*0{,}30 + 3*0{,}15 + 4*0{,}04 + 5*0{,}01 = 1{,}76$$

Se as entradas estivessem distribuídas na ordem inversa (o caso mais desfavorável), o número médio de endereços examinados seria:

$$EE = 1*0{,}01+ 2*0{,}04 + 3*0{,}15 + 4*0{,}30 + 5*0{,}50 = 4{,}24$$

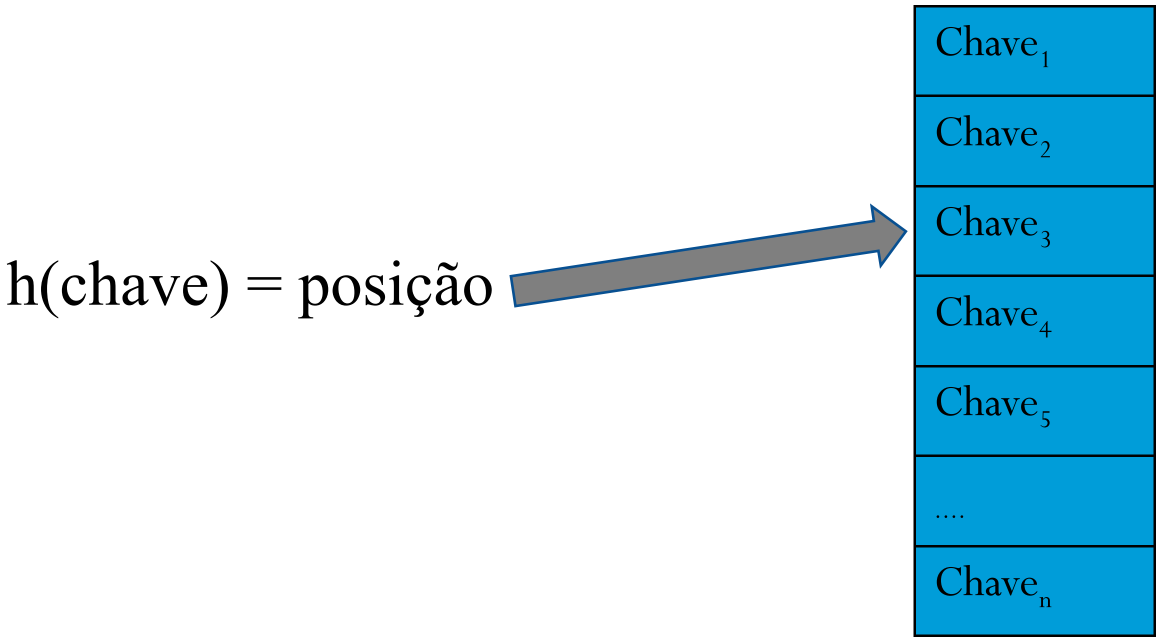
Busca sequencial usando frequência

- Necessidade de conhecer previamente a frequência ou calculá-la utilizando um histórico de buscas bem-sucedidas
- Auto-organização (numa abordagem mais simples):
  - Armazenar o número de buscas, ao invés da frequência
  - Alterar a posição das chaves dado o número de acessos em cada chave



Abordagem diferente

- Calcular a posição da chave na tabela baseado no valor da chave



- Posição na tabela pode ser acessada diretamente
- Não há necessidade de testes
- Complexidade é reduzida para O(1)
  - Busca sequencial: complexidade é O(n)
  - Busca binária: complexidade é O(lg n)

Hashing

- Segundo o Webster's New World Dictionary, significa:
  - Fazer picadinho de carne e vegetais para cozinhar;
  - Fazer uma bagunça
- Qualquer que seja a função de transformação, colisões irão acontecer
- O Paradoxo do aniversário (Feller, 1968, p.33) diz que em um grupo de 23 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%.

Hashing

- Segundo o Webster's New World Dictionary, significa:
  - Fazer picadinho de carne e vegetais para cozinhar;
  - Fazer uma bagunça
- Qualquer que seja a função de transformação, colisões irão acontecer
- O Paradoxo do aniversário (Feller, 1968, p.33) diz que em um grupo de 23 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%.

Paradoxo do aniversário

- A probabilidade p de se inserir N itens consecutivos sem colisão em uma tabela de tamanho M é:

$$p = \frac{M-1}{M} \times \frac{M-2}{M} \times \ldots \times \frac{M-N+1}{M} =$$

$$\prod_{i=1}^{N} \frac{M-i+1}{M} = \frac{M!}{(M-N)!M^N}$$

Paradoxo do aniversário

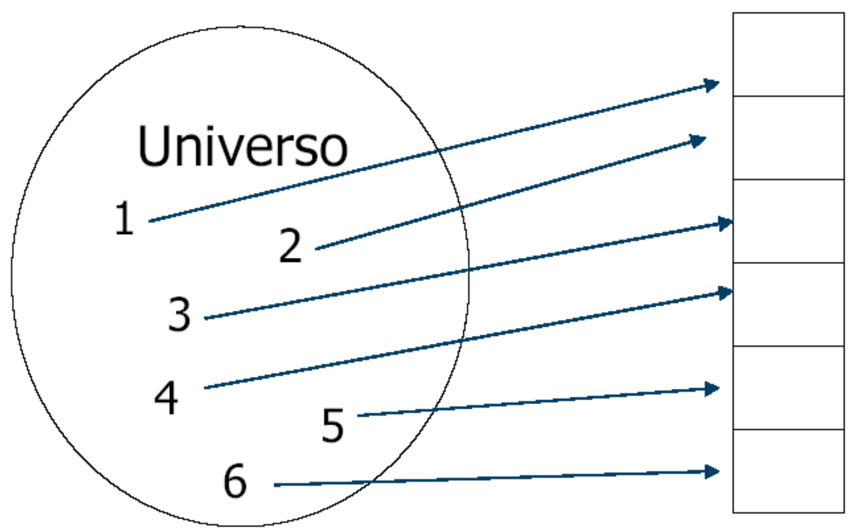
- Alguns valores de p para diferentes valores de N, onde M=365.

N	p
10	0,883
22	0,524
23	0,493
30	0,303

- Para N pequeno a probabilidade p pode ser aproximada por $p \approx N*(N-1) / 730$. Por exemplo, para $N=10$ então $p \approx 87,7\%$.

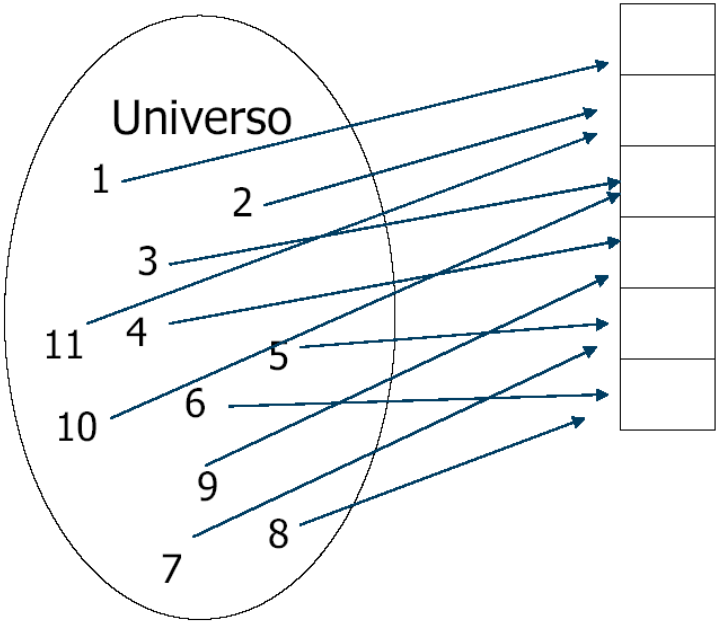
Hashing

- A função *h* deve ser tal que
  - Transforma uma chave em um índice na tabela
- Chave pode ser: cadeia de caracteres, um número, um registro …
- Função *h* é chamada de função *hashing* ou de escrutínio.
- Universo de chaves mapeado em posições



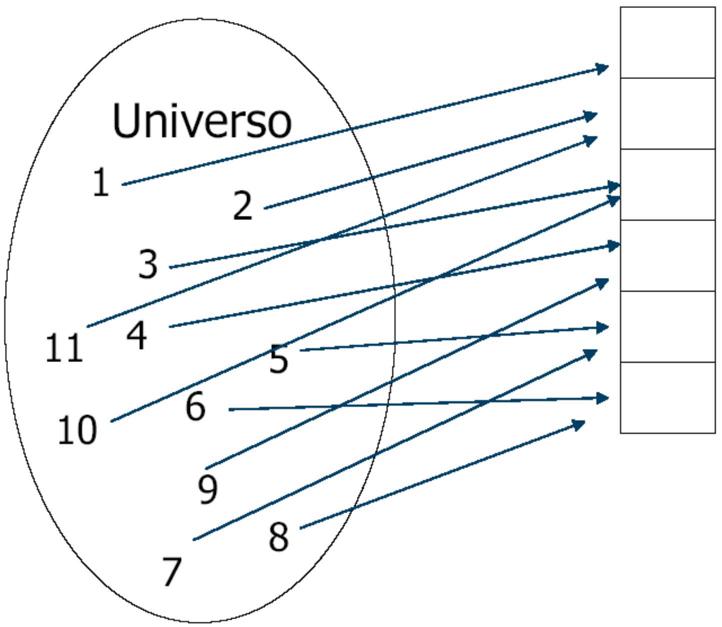
Hashing

- A função *h* deve ser tal que
  - Transforma uma chave em um índice na tabela
- Chave pode ser: cadeia de caracteres, um número, um registro …
- Função *h* é chamada de função *hashing* ou de escrutínio.
- Universo de chaves mapeado em posições
- Colisões podem ocorrer:



Hashing

- A função *h* deve ser tal que
  - Transforma uma chave em um índice na tabela
- Chave pode ser: cadeia de caracteres, um número, um registro …
- Função *h* é chamada de função *hashing* ou de escrutínio.
- Universo de chaves mapeado em posições
- Colisões podem ocorrer:



Hashing

- Função de *hashing* perfeita
  - Transforma diferentes chaves em diferentes posições sem colisões
- Para criar função perfeita
  - Tabela tem que conter mesmo número de posições que o número de elementos que sofreram *hashing*
- Quais são os problemas?
  - Pode-se não conhecer a priori o número total de elementos
  - O número possível de elementos pode ser muito maior do que o número total de elementos

Hashing

- Exemplo: Armazenamento dos empregados de uma empresa em um array
  - Quantas posições o array terá que ter?
  - Considerando que o número máximo de funcionários possíveis é igual à quantidade de CPFs diferentes que podem existir.
    - Quantas posições o array terá que ter?

Hashing

- Exemplo: Empregados identificados por CPF

```
Class Empregado{
  int cpf;
  char nome[80];
  char end[120];
   ...
}
```

  - CPF como índice: 1 bilhão de entradas 000 000 000 a 999 999 999
    - emp vet[1000000000]
  - Reservar espaço para o total de possíveis empregados
  - Mas, se empresa possui apenas 500 empregados, temos desperdício enorme de memória.