



Estrutura de Dados I

Busca sequencial e hash

Bruno Prado

Departamento de Computação / UFS

Introdução

- ▶ Estruturas de dados lineares
 - ▶ Lista
 - ▶ Fila
 - ▶ Pilha
 - ▶ ...

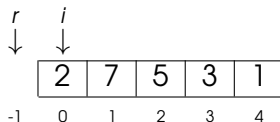
Sequências de elementos
que obedecem à regras de operação

Introdução

- ▶ O que é a busca sequencial?
 - ▶ Consiste em acessar sequencialmente os dados da estrutura utilizando um elemento inicial
 - ▶ A busca é finalizada com o elemento procurado é encontrado ou não existem mais elementos para serem comparados
 - ▶ É uma estratégia de força bruta

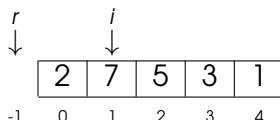
Busca Sequencial

- ▶ Busca em vetores sem ordenação
 - ▶ Parâmetro de busca: 1
 - ▶ O índice de resultado r possui o valor -1 e o de busca i recebe o valor 0
 - ▶ É feita a comparação do elemento da posição i com o valor do parâmetro de busca



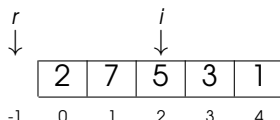
Busca Sequencial

- ▶ Busca em vetores sem ordenação
 - ▶ Parâmetro de busca: 1
 - ▶ Como o elemento procurado não foi encontrado na posição, o índice de busca i é incrementado e o índice de resultado r não é modificado



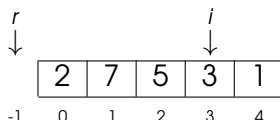
Busca Sequencial

- ▶ Busca em vetores sem ordenação
 - ▶ Parâmetro de busca: 1
 - ▶ Como o elemento procurado não foi encontrado na posição, o índice de busca i é incrementado e o índice de resultado r não é modificado



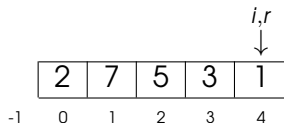
Busca Sequencial

- ▶ Busca em vetores sem ordenação
 - ▶ Parâmetro de busca: 1
 - ▶ Como o elemento procurado não foi encontrado na posição, o índice de busca i é incrementado e o índice de resultado r não é modificado



Busca Sequencial

- ▶ Busca em vetores sem ordenação
 - ▶ Parâmetro de busca: 1
 - ▶ O elemento procurado é encontrado na posição 4 e o índice de resultado r recebe o valor atual do índice de busca i



Busca Sequencial

► Implementação em C

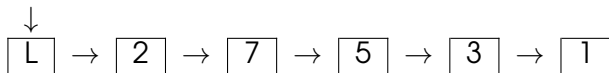
```
// Busca sequencial
int busca_seq(int* vetor, unsigned int tamanho, int valor) {
    // Ajustando índices
    int r = -1;
    int i;
    // Busca iterativa
    for(i = 0; i < tamanho && r == -1; i++)
        if(vetor[i] == valor)
            r = i;
    // Retornando índice de resultado
    return r;
}
```

Busca Sequencial

- ▶ Análise de complexidade
 - ▶ Espaço $O(1)$
 - ▶ Tempo $\Omega(1) \leq busca_seq \leq O(n)$

Busca Sequencial

- ▶ Busca em listas
 - ▶ Parâmetro de busca: 1
 - ▶ A busca tem início acessando a cabeça da lista que contém a referência para o primeiro elemento



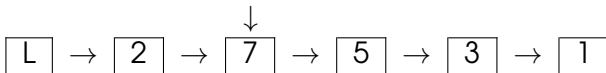
Busca Sequencial

- ▶ Busca em listas
 - ▶ Parâmetro de busca: 1
 - ▶ É feito o acesso ao próximo elemento da lista para comparação com o valor do parâmetro de busca até que o elemento seja encontrado ou que não existam mais elementos na lista



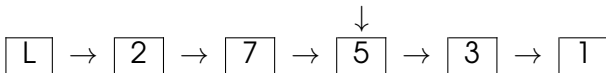
Busca Sequencial

- ▶ Busca em listas
 - ▶ Parâmetro de busca: 1
 - ▶ É feito o acesso ao próximo elemento da lista para comparação com o valor do parâmetro de busca até que o elemento seja encontrado ou que não existam mais elementos na lista



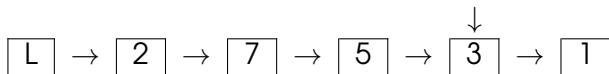
Busca Sequencial

- ▶ Busca em listas
 - ▶ Parâmetro de busca: 1
 - ▶ É feito o acesso ao próximo elemento da lista para comparação com o valor do parâmetro de busca até que o elemento seja encontrado ou que não existam mais elementos na lista



Busca Sequencial

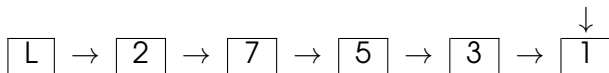
- ▶ Busca em listas
 - ▶ Parâmetro de busca: 1
 - ▶ É feito o acesso ao próximo elemento da lista para comparação com o valor do parâmetro de busca até que o elemento seja encontrado ou que não existam mais elementos na lista



Busca Sequencial

- ▶ Busca em listas

- ▶ Parâmetro de busca: 1
- ▶ O elemento procurado é encontrado e sua referência é armazenada para ser retornada pela função de busca, permitindo acesso direto ao seu conteúdo



Busca Sequencial

► Implementação em C

```
// Busca sequencial
elemento* busca_seq(lista L, int valor) {
    // Ajustando ponteiros
    elemento* r = NULL;
    elemento* i = L.P;
    // Busca iterativa
    while(i != NULL && r == NULL) {
        if(i->E == valor)
            r = i;
        i = i->P;
    }
    // Retornando ponteiro do elemento
    return r;
}
```

Busca Sequencial

- ▶ Análise de complexidade
 - ▶ Espaço $O(1)$
 - ▶ Tempo $\Omega(1) \leq busca_seq \leq O(n)$

Hash

- ▶ O que é uma estratégia de busca com hash?
 - ▶ É o cálculo em tempo constante da posição em que um determinado dado foi armazenado
 - ▶ O valor do parâmetro de busca é aplicado a uma função que mapeia o elemento procurado na estrutura de dados, permitindo seu acesso direto sem necessidade de busca

Elemento 1



Hash(1)

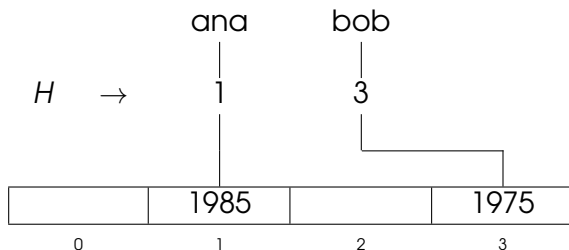


Posição 4

- ▶ Considere cadeias de caracteres utilizando somente letras minúsculas com tamanho máximo de até 100 caracteres
 - ▶ Calculando as possíveis combinações de texto são possíveis até $26^{100} \approx 3.14 \times 10^{141}$ padrões distintos
 - ▶ O uso de uma função hash é a solução mais adequada, uma vez que não é viável armazenar todas as posições de armazenamento que cada padrão possível de texto pode ocupar
 - ▶ Apesar do espaço possível de padrões ser muito grande, apenas uma pequena parte dos padrões de texto são normalmente utilizados na prática

Hash

- ▶ O que é uma função hash?
 - ▶ É o mapeamento de um conjunto de dados em índices ou em posições de uma estrutura de dados de tamanho limitado
 - ▶ A aplicação desta função permite o armazenamento e recuperação associativa dos dados em tempo constante



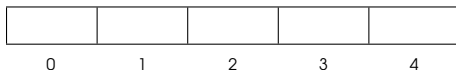
- ▶ Definições de uma função hash H
 - ▶ Possuir idealmente um comportamento injetivo, nunca mapeando duas entradas distintas em uma mesma posição da estrutura de dados
 - ▶ Utilizar uma função F com distribuição uniforme dos dados, reduzindo a chance de colisões de posições
 - ▶ Calcular valores de mapeamento limitados ao tamanho T da estrutura de dados

$$H(x) = F \bmod T$$

Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É feita a alocação de um vetor com 5 posições

$$H(n) = (33 \times x) \bmod 5$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 1

$$H(1) = (33 \times 1) \bmod 5 = 3$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 2

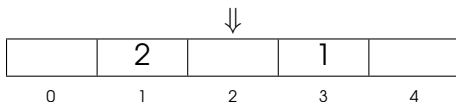
$$H(2) = (33 \times 2) \bmod 5 = 1$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 4

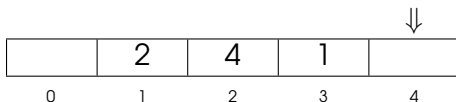
$$H(4) = (33 \times 4) \bmod 5 = 2$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 8

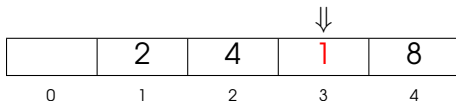
$$H(8) = (33 \times 8) \bmod 5 = 4$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 16

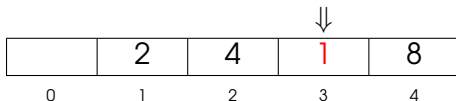
$$H(16) = (33 \times 16) \bmod 5 = 3$$



Hash

- ▶ Mapeando os elementos no vetor
 - ▶ Função hash $H(x) = 33x \bmod 5$
 - ▶ É calculado o índice para o elemento de valor 16

$$H(16) = (33 \times 16) \bmod 5 = 3$$



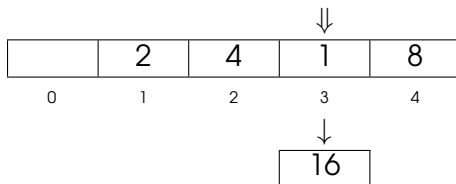
	2	4	1	8
0	1	2	3	4

Colisão de mapeamento!

Hash

► Endereçamento fechado

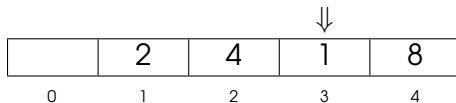
- O tratamento de colisão é feito através da utilização de uma estrutura de dados auxiliar que permita o ajuste incremental de capacidade
- Se uma estrutura de lista for utilizada, cada posição do vetor é a cabeça de uma lista
- Quando as colisões ocorrem os elementos são inseridos na estrutura de lista



Hash

- ▶ Endereçamento aberto
 - ▶ Técnica de linear probing
 - ▶ Função hash auxiliar $LP(x, i) = H(x) + i \bmod T$, com $i = 0, 1, \dots, T - 1$
 - ▶ É feito um novo cálculo de mapeamento utilizando a função hash auxiliar, evitando que espaço adicional seja alocado para o armazenamento dos elementos

$$LP(16, 0) = (H(16) + 0) \bmod 5 = 3$$

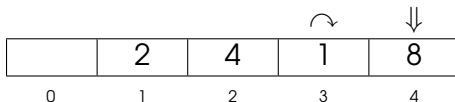


	2	4	1	8
0	1	2	3	4

Hash

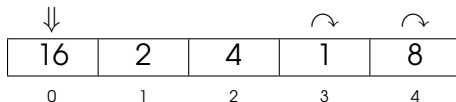
- ▶ Endereçamento aberto
 - ▶ Técnica de linear probing
 - ▶ Função hash auxiliar $LP(x, i) = H(x) + i \bmod T$, com $i = 0, 1, \dots, T - 1$
 - ▶ É feito um novo cálculo de mapeamento utilizando a função hash auxiliar, evitando que espaço adicional seja alocado para o armazenamento dos elementos

$$LP(16, 1) = (H(16) + 1) \bmod 5 = 4$$



- ▶ Endereçamento aberto
 - ▶ Técnica de linear probing
 - ▶ Função hash auxiliar $LP(x, i) = H(x) + i \bmod T$, com $i = 0, 1, \dots, T - 1$
 - ▶ É feito um novo cálculo de mapeamento utilizando a função hash auxiliar, evitando que espaço adicional seja alocado para o armazenamento dos elementos

$$LP(16, 2) = (H(16) + 2) \bmod 5 = 0$$



Hash

- ▶ Endereçamento aberto
 - ▶ Técnica de double hashing
 - ▶ Função hash dupla $DH(x, i) = H_1(x) + iH_2(x) \bmod T$
 - ▶ É aplicada uma função secundária de hash para deslocar o resultado para outra posição do vetor

$$DH(16, 0) = (33 \times 16 + 0 \times 7 \times 16) \bmod 5 = 3$$

	2	4	1	8
0	1	2	3	4

- ▶ Endereçamento aberto
 - ▶ Técnica de double hashing
 - ▶ Função hash dupla $DH(x, i) = H_1(x) + iH_2(x) \bmod T$
 - ▶ É aplicada uma função secundária de hash para deslocar o mapeamento para outro índice do vetor

$$DH(16, 1) = (33 \times 16 + 1 \times 7 \times 16) \bmod 5 = 0$$

\Downarrow					\rightarrow				
16	2	4	1	8					
0	1	2	3	4					

- ▶ Análise de complexidade
 - ▶ Espaço $O(1)$
 - ▶ Tempo $O(1)$
- ▶ Considerações
 - ▶ O espaço e tempo são constantes desde que os dados estejam uniformemente distribuídos e a capacidade da estrutura não seja muito utilizada, permitindo que o número de colisões sejam limitadas
 - ▶ A utilização de números primos para definir o tamanho das estrutura de dados utilizadas, reduzindo assim as chances de colisões nos mapeamentos

- ▶ Aplicações

- ▶ Memória cache
- ▶ Criptografia
- ▶ Checagem de integridade de dados
- ▶ Tabela de símbolos de compilador
- ▶ ...

Exercício

- ▶ A empresa de tecnologia Poxim Tech está criando um engenho de busca experimental para retornar ocorrências de padrões de texto em uma interface web
 - ▶ De acordo com o valor de chave gerado pelo texto é feito o mapeamento da requisição de busca para um dos servidores dedicados, utilizando um cálculo de checksum de 8 bits para cada um dos caracteres
 - ▶ Para atender as solicitações em tempo real, cada um dos servidores só é capaz de atender um número máximo de requisições ao mesmo tempo, sendo feita uma realocação do servidor por double hashing definida por $H_1(x) = 7919 \times \text{checksum}(x) \bmod T$ e $H_2(x) = 104729 \times \text{checksum}(x) + 123 \bmod T$
 - ▶ Todos os padrões pesquisados são compostos exclusivamente por letras e números com até 100 caracteres

Exercício

- ▶ Função de checksum de 8 bits
 - ▶ Realiza a operação de $xor \oplus$ com os valores numéricos ASCII dos caracteres

$$\begin{aligned}checksum("ufs") &= 'u' \oplus 'f' \oplus 's' \\ &= 117 \oplus 102 \oplus 115 \\ &= 96\end{aligned}$$

Probabilidade de duas strings diferentes gerarem o mesmo valor numérico é de $\frac{1}{2^8} \approx 0,4\%$

Exercício

- ▶ Formato do arquivo de entrada
 - ▶ $[\# \text{Servidores}] [\text{Capacidade máxima}]$
 - ▶ $[\# n]$
 - ▶ $[\# m_1] [P_1] [P_2] \dots [P_{m_1}]$
 - ▶ \dots
 - ▶ $[\# m_n] [P_1] [P_2] \dots [P_{m_n}]$

```
3 2
5
1 ufs
3 a b c
2 cd ef
2 e d
1 hash
```


Exercício

- ▶ Formato do arquivo de saída
 - ▶ É exibido o servidor alocado para realização da busca e os padrões de texto que estão sendo processados, além da realocação das requisições quando um servidor já atingiu o limite de operações

```
[S0] ufs  
[S0] ufs, a b c  
[S2] cd ef  
[S2] cd ef, e d  
S0->S1  
[S1] hash
```