Estrutura de Dados (CCA410)

Aula 4 – Tabela de Espalhamento (Hash)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2º Semestre de 2025

Reflexão sobre desempenho

Tabelas de Espalhamento (Hash)

Motivação: Estratégias de Busca

Estrutura de lista linear sem ordenação (**Busca Linear**)

5	25	35	70	95	57	23	54	1

Quanto tempo eu levaria para uma busca nesta lista, no pior caso?

Estrutura de lista linear com ordenação (**Busca Binária**):

1	5	23	25	35	54	57	70	95

Quanto tempo eu levaria para uma busca nesta lista, no pior caso?

Será que podemos fazer melhor?

Complexidade de algoritmos de busca:

Busca Linear: $O(n)$

Busca Binária: $O(\log_2 n)$

Hash: $O(1)$

Tabela de Endereçamento Direto

Tabelas de Espalhamento (Hash)

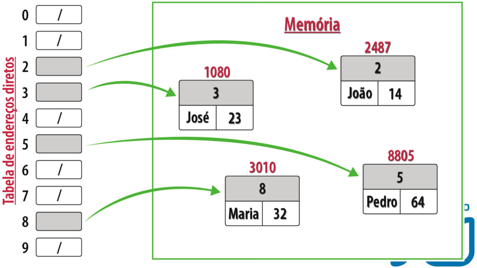
Tabela de Endereçamento Direto

```c
int main(void) {
  registro *tabela = malloc(10 * sizeof(registro));
  registro r1 = {2, "Joao", 14};
  tabela[2] = r1;
  registro r2 = {3, "Jose", 23};
  tabela[3] = r2;
  registro r3 = {5, "Pedro", 64};
  tabela[5] = r3;
  registro r4 = {8, "Maria", 32};
  tabela[8] = r4;

  for(int i = 0; i < 10; i++){
    printf("ID: %d\n", i);
    printf("Nome: %s\n", tabela[i].nome);
    printf("Idade: %d\n", tabela[i].idade);
    puts("");
  }
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int idx;
  char nome[15];
  int idade;
} registro;
```

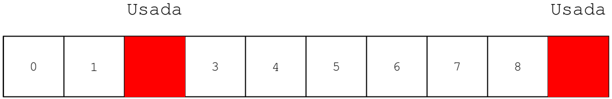


Limitações da Tabela de Endereçamento Direto

Problema Fundamental

A eficiência do endereçamento direto depende criticamente da dispersão das chaves no universo de valores possíveis.

Cenário problemático: Chaves esparsamente distribuídas



Universo de chaves possíveis

Resultado: Desperdício massivo de memória

Tabelas de Espalhamento (Hash)

Endereçamento Direto – Exemplo Prático

Caso Real

Sistema precisa armazenar registros com chaves $D = \{2, 100.002\}$

Requisitos de memória:

- Vetor: 100.003 posições
- Índices: 0 a 100.002
- Posições utilizadas: apenas 2

Análise de eficiência:

- Taxa de ocupação: $\sim 0,002\%$
- Posições vazias: 100.001
- Desperdício: $> 99,99\%$

Conclusão

Endereçamento direto é inviável para chaves esparsas

Tabelas de Espalhamento (Hash)

Quando Endereçamento Direto Falha

Condições de inviabilidade:

- Universo de chaves muito grande
- Chaves esparsamente distribuídas
- Baixa razão: $\frac{\text{chaves usadas}}{\text{espaço total}}$

Fórmula crítica:

$$\text{Eficiência} = \frac{n}{U}$$

n = chaves, U = universo

Necessidade de Alternativa

Estrutura que mantenha:

- Acesso eficiente $O(1)$
- Uso otimizado de memória
- Flexibilidade para diferentes distribuições

Solução: Tabelas de Espalhamento (Hash)

Tabela de Espalhamento (Hash)

Definição

É uma estrutura de dados especial, que associa chaves de pesquisa a valores;

O objetivo é promover uma busca eficiente.

Tabelas de Espalhamento (Hash)

Ideia geral

Cada chave ocupa a estrutura de dados através de uma **função**

Esta função mapeia uma **CHAVE** para um valor **INTEIRO**

Esta função se chama: **funcão HASH**.

A estrutura de dados que abriga as chaves chama-se: **Tabela Hash**

Definição

Mapeamento de chaves para índices da tabela

$$h : U \rightarrow \{0, 1, 2, ..., m - 1\}$$

Propriedades Essenciais:

- Eficiência: $O(1)$
- Distribuição uniforme
- Determinística
- Sensibilidade a mudanças

Métodos Principais:

- Divisão
- Multiplicação
- Hashing universal

Princípio de Design

A escolha da função hash deve considerar a natureza dos dados e o contexto da aplicação

Definição

$$h(k) = k \bmod m$$

onde k = chave e m = tamanho da tabela

Características:

- Simples implementação
- Uma operação de módulo
- Complexidade: $O(1)$

Exemplo ($m = 13$):

$h(123) = 123 \bmod 13 = 6$

$h(456) = 456 \bmod 13 = 1$

Escolha crítica do valor m**:**

Evitar

× Potências de 2 (2^p)

× Potências de 10

Preferir

✓ Números primos

✓ Distantes de 2^p e 10^p

Tabelas de Espalhamento (Hash)

Método da Multiplicação

Definição

$$h(k) = \lfloor m \times (k \times A \bmod 1) \rfloor$$

onde:

- A: constante com $0 < A < 1$
- $(k \times A \bmod 1)$: parte fracionária
- $\lfloor\cdot\rfloor$: função piso

Vantagens:

- Valor de m não é crítico
- Funciona com qualquer tamanho
- Menos sensível a parâmetros

Constante Ideal:

$$A = \frac{\sqrt{5}-1}{2} \approx 0.618$$

(Razão áurea - Knuth)

Exemplo: $(m = 8,\ A = 0.618)$:

$$k = 123 \quad (1)$$
$$123 \times 0.618 = 75.994 \quad (2)$$
$$\text{Fração} = 0.994 \quad (3)$$
$$h(123) = \lfloor 8 \times 0.994 \rfloor = 7 \quad (4)$$

Definição Informal

Família de funções hash onde a função específica é escolhida aleatoriamente

Definição Formal:

Uma família $\mathcal{H}$ é universal se:

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}$$

para chaves distintas x, y e h escolhido aleatoriamente de $\mathcal{H}$

Vantagens:

- Desempenho estatístico garantido
- Resistente a dados adversariais
- Análise probabilística rigorosa

Exemplo Clássico

- Para primo $p > \max(\text{chaves})$:
- $h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$

Parâmetros aleatórios:

- $a \in \{1, 2, ..., p - 1\}$
- $b \in \{0, 1, ..., p - 1\}$

Desvantagem

- Maior complexidade de implementação

Aspecto	Divisão	Multiplicação	Universal
Simplicidade	Alta	Média	Baixa
Velocidade	Alta	Média	Baixa
Sensibilidade a m	Alta	Baixa	Baixa
Distribuição	Boa$\star$	Boa	Excelente
Análise teórica	Limitada	Boa	Rigorosa
Uso prático	Muito comum	Comum	Especializado

Recomendações:

- **Divisão:** Aplicações gerais, escolha cuidadosa de m primo
- **Multiplicação:** Quando o tamanho da tabela varia ou não pode ser primo
- **Universal:** Aplicações críticas, dados adversariais, análise rigorosa

Colisões

Tabelas de Espalhamento (Hash)

Fator de Carga e Clustering

Fator de Carga (α)

Definição:

$$\alpha = \frac{n}{m}$$

onde n = elementos armazenados e m = tamanho da tabela

Interpretação:

- $\alpha = 0.5$: tabela 50% cheia
- $\alpha = 1.0$: tabela completa
- $\alpha > 1.0$: só com encadeamento

Impacto na performance:

- Encadeamento: $O(1 + \alpha)$
- Endereç. aberto: degrada rapidamente se $\alpha > 0.7$

Clustering

Clustering primário:

- Blocos contíguos de posições ocupadas
- Causa: sondagem linear
- Efeito: tempo $O(1) \to O(n)$

Clustering secundário:

- Elementos com mesmo hash inicial seguem mesma sequência
- Causa: sondagem quadrática
- Menos severo que primário

Soluções:

- Hash duplo elimina ambos
- Manter α baixo
- Função hash de qualidade

Exemplo visual do clustering primário:

[_] [X] [X] [X] [X] [_] [_] [_]

Novo elemento → posição 5 (estende cluster)

Princípio do Pombal

Com n chaves e m posições ($n > m$), colisões são matematicamente inevitáveis

Distribuição Ideal:

$$P(h(k) = i) = \frac{1}{m}$$

para $i = 0, 1, ..., m - 1$

Limitações Práticas:

- Custo computacional elevado
- Dependência dos dados
- Impossibilidade teórica

Necessidade

Métodos sistemáticos de tratamento

Estratégias:

- Encadeamento externo
- Endereçamento aberto
- Preservar eficiência $O(1)$

centro universitário FEI

Tratamento de Colisões

Tabelas de Espalhamento (Hash)

Encadeamento Separado

Princípio

Cada posição da tabela mantém uma lista ligada com todos os elementos que colidem

Estrutura:

- Vetor de ponteiros
- Listas ligadas por posição
- Inserção no início (eficiente)

Complexidade das Operações:

- **Inserção:** $O(1)$
- **Busca:** $O(1+\alpha)$
- **Remoção:** $O(1+\alpha)$

α = fator de carga

Vantagens

- Implementação simples
- Permite $\alpha > 1$
- Remoção direta

Desvantagem

Overhead de ponteiros e fragmentação de memória

Princípio

Se $h(k)$ está ocupada, busca sequencialmente a próxima posição livre

Função de Sondagem:

$$h(k,i) = (h(k) + i) \bmod m$$

onde $i = 0, 1, 2, ...$

Operações:

- **Inserção:** Busca linear até slot vazio
- **Busca:** Linear até encontrar ou vazio
- **Remoção:** Marca como "deletada"

Clustering Primário

Problema: Elementos formam blocos contíguos

Exemplo visual:

```
[_] [X] [X] [X] [_] [_]
[_] [X] [X] [X] [X] [_]
```

Performance degrada conforme tabela enche

Recomendação

Manter $\alpha < 0.7$

Tabelas de Espalhamento (Hash)

Sondagem Quadrática

Princípio

Usa incrementos quadráticos para evitar clustering primário

Função de Sondagem:

$$h(k,i) = (h(k) + c_1 i + c_2 i^2) \bmod m$$

Caso comum ($c_1 = c_2 = \frac{1}{2}$):

$$h(k,i) = \left(h(k) + \frac{i+i^2}{2}\right) \bmod m$$

Sequência: $+0, +1, +3, +6, +10, +15, \ldots$

Vantagens

- Elimina clustering primário
- Melhor distribuição

Clustering Secundário

Problema: Elementos com mesmo $h(k)$ seguem mesma sequência

Outras Limitações:

- Pode não explorar toda tabela
- Requer m primo para garantias

Garantia

Se m é primo e $\alpha \leq 0.5$, sempre encontra slot livre

Princípio

Usa segunda função hash para determinar incremento da sondagem

Função de Sondagem:

$$h(k,i) = (h_1(k) + i \times h_2(k)) \bmod m$$

onde:

- $h_1(k)$: posição inicial
- $h_2(k)$: incremento ($\neq 0$)

Exemplo Comum:

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod (m-1))$$

Garante $h_2(k) \in \{1, 2, ..., m-1\}$

Vantagens

- Elimina ambos os clusterings
- Distribuição próxima ao ideal
- Explora toda tabela[a]

[a] se $\gcd(h_2(k), m) = 1$

Desvantagem

Maior custo computacional (duas funções hash)

Melhor método para endereçamento aberto

Tabelas de Espalhamento (Hash)

Comparação dos Métodos de Tratamento de Colisões

Aspecto	Encadeamento	Linear	Quadrática	Hash Duplo
Simplicidade	Média	Alta	Média	Baixa
Uso de memória	+ ponteiros	Compacta	Compacta	Compacta
Fator de carga	>1 permitido	<0.7	<0.5	<0.7
Clustering	Não aplica	Primário	Secundário	Nenhum
Remoção	Simples	Complexa	Complexa	Complexa
Performance	Boa	Degrada	Melhor	Melhor

Recomendações de uso:

- **Encadeamento:** Remoções frequentes, dados dinâmicos
- **Linear:** Implementação simples, poucos dados
- **Quadrática:** Meio termo entre simplicidade e performance
- **Hash duplo:** Performance crítica, implementação mais complexa aceitável

Exemplo Prático

Especificações:

Função hash: $h(k) = k \bmod 7$

Tratamento de colisões: encadeamento externo

Conjunto: $\{190, 322, 172, 89, 13, 4, 769, 61, 15, 76\}$

Tarefas:

1. Calcule $h(k)$ para cada valor
2. Desenhe a tabela hash resultante
3. Simule inserção e remoção
4. Analise o fator de carga final

Objetivos: Compreender colisões, implementar encadeamento, analisar distribuição

Estrutura de Dados (CCA410)

Aula 4 – Tabela de Espalhamento (Hash)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2º Semestre de 2025