Algoritmos e Estruturas de Dados I

Tabelas de Dispersão

versão 2.13

Fabiano Oliveira
fabiano.oliveira@ime.uerj.br

Tabelas de Dispersão

- **Motivação**: Comparemos as operações de inserção, busca e remoção em listas lineares com N elementos:

Operação	Alocação Sequencial		Alocação Encadeada		Árvore Balanceada
	Ordenada	Não-Ordenada	Ordenada	Não-Ordenada	
Inserção	O(N)	θ(1)	O(N)	θ(1)	O(lgN)
Busca	O(lg N)	O(N)	O(N)	O(N)	O(lgN)
Remoção	O(N)	O(N)	O(N)	O(N)	O(lgN)

Tabelas de Dispersão

- **Motivação**: Comparemos as operações de inserção, busca e remoção em listas lineares com N elementos:

Operação	Alocação Sequencial		Alocação Encadeada		Árvore Balanceada	**"Estrutura dos Sonhos"**
	Ordenada	Não-Ordenada	Ordenada	Não-Ordenada		
Inserção	O(N)	θ(1)	O(N)	θ(1)	O(lgN)	**θ(1)**
Busca	O(lg N)	O(N)	O(N)	O(N)	O(lgN)	**θ(1)**
Remoção	O(N)	O(N)	O(N)	O(N)	O(lgN)	**θ(1)**

AcessoDireto <TElem>

função Busca(**ref** T: AcessoDireto, c: Inteiro): <TElem>

Obtém o elemento de L com chave c ou NULO se inexistente

procedimento Insere(**ref** T: AcessoDireto, c: Inteiro, x: <TElem>)

Insere x com chave c em L

função Remove(**ref** T: AcessoDireto, c: Inteiro): <TElem>

Remove e retorna o elemento de L com chave c

função Tamanho(**ref** T: AcessoDireto): Inteiro

Obtém o número de elementos em L

Tabelas de Dispersão

- Exemplo (Solução do Exercício 6):

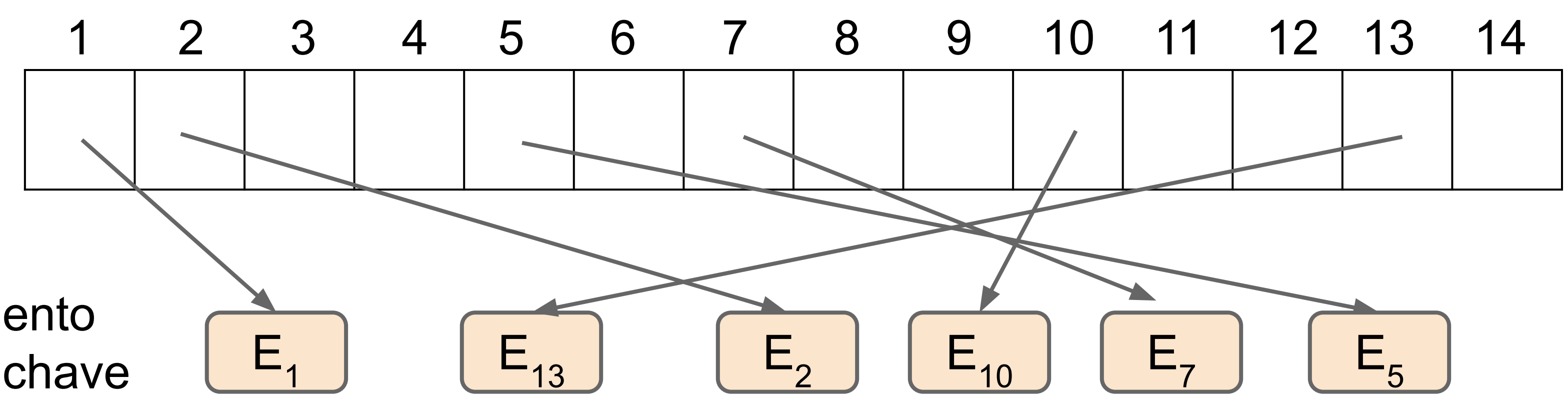
```
função EhAnagrama(ref A[], B[]: Caracter, N: Inteiro): Lógico
    var T1, T2: AcessoDireto<Inteiro>
    para c ← 1 até 256 faça
        Insere(T1, c, 0); Insere(T2, c, 0)
    para i ← 1 até N faça
        c1, c2 ← ASCII(A[i])+1, ASCII(B[i])+1
        Insere(T1, c1, Remove(T1, c1)+1)
        Insere(T2, c2, Remove(T2, c2)+1)
    Anagramas ← V
    para c ← 1 até 256 faça
        se Busca(T1, c) ≠ Busca(T2, c) então
            Anagramas ← F
            sair-para
    retornar Anagramas
```

Tempo:
$\theta(N)$

Tabelas de Dispersão

- Uma ***tabela de acesso direto*** para um conjunto de N chaves (com valores entre 1 a M) é um vetor H[1..M] tal que o elemento de chave c é armazenado em H[c]

```
H[1..14]: <TElem>
```



E_c = elemento com chave c

Tabelas de Dispersão

Espaço:
θ(M)

```
var M: Inteiro ← <MAIOR CHAVE POSSÍVEL>

estrutura AcessoDireto <TElem>:
   H[1..M]: <TElem>
   N: Inteiro //número de elementos na lista

procedimento Constroi(ref T: AcessoDireto)
   T.H[1..M], T.N ← "NULO", 0

procedimento Destroi(ref T: AcessoDireto)
   //nada a ser feito
```

Tabelas de Dispersão

```
procedimento Insere(ref T: AcessoDireto,
                   c: Inteiro, x: <TElem>)
   T.H[c], T.N ← x, T.N+1

função Busca(ref T: AcessoDireto, c: Inteiro): <TElem>
   retornar T.H[c]

função Remove(ref T: AcessoDireto, c: Inteiro): <TElem>
   var e: <TElem>
   e, T.H[c], T.N ← T.H[c], "NULO", T.N-1
   retornar (e)

função Tamanho(ref T: AcessoDireto): Inteiro
   retornar (T.N)
```

Tempo:
$\theta(1)$

Tabelas de Dispersão

- **Problema**:

    Dados um vetor A[1..N]: Inteiro (positivos) e um inteiro positivo K, determinar se existem dois valores neste vetor que somem K.

Tabelas de Dispersão

- **Solução 1:** Para cada $1 \le i \le j \le N$, verificar se $A[i]+A[j] = K$
  - **Tempo:** $O(N^2)$

- **Solução 2:** Ordenar A[1..N]. Para cada $1 \le i \le N$, verificar se $K-A[i]$ existe em A[1..N] por pesquisa binária.
  - **Tempo:** $O(N \lg N)$

- **Solução 3:** Ordenar A[1..N]. Fazendo $i, j \leftarrow 1, N$ inicialmente, manter o invariante "se existe a, b tais que $A[a]+A[b] = K$, então $i \le a \le b \le j$" e ir iterativamente incrementando i (se $A[i]+A[j]<K$) ou decrementando j (se $A[i]+A[j]>K$)
  - **Tempo:** $O(N \lg N)$, mas $O(N)$ se A já estiver ordenado

Tabelas de Dispersão

- **Solução 4:**

```
função DeterminaSoma(ref A[]: Inteiro, N, K: Inteiro): Lógico
    M ← máx { A[i] | 1 ≤ i ≤ N }
    var T: AcessoDireto<Lógico>

    para i ← 1 até N faça
        Insere(T, A[i], V)
    ExisteSoma ← F
    para i ← 1 até N faça
        se (K-A[i]≤M) e Busca(T, K-A[i]) então
            ExisteSoma ← V
            sair-para

    retornar ExisteSoma
```

Tempo:
$\theta(N + M)$

Tabelas de Dispersão

- Quando usar Acesso Direto?
  - M - N é pequeno
    (isto é, o desperdício é desprezível)
  - M - N é grande, mas o benefício supera o custo
    (operações de tempo constante são de vital importância, por exemplo, numa aplicação servidora de autenticação de usuários)

- E se o custo do "M - N grande" for proibitivo?
  Talvez ainda haja uma solução próxima: usar uma Tabela de Dispersão!

TabelaDispersao <TChave, TElem>

```
função Busca(ref T: TabelaDispersao, c: <TChave>): <TElem>
```

Obtém o elemento de L com chave c ou "NULO" se inexistente

```
procedimento Insere(ref T: TabelaDispersao, c: <TChave>, x: <TElem>)
```

Insere x com chave c em L

```
função Remove(ref T: TabelaDispersao, c: <TChave>): <TElem>
```

Remove e retorna o elemento de L com chave c

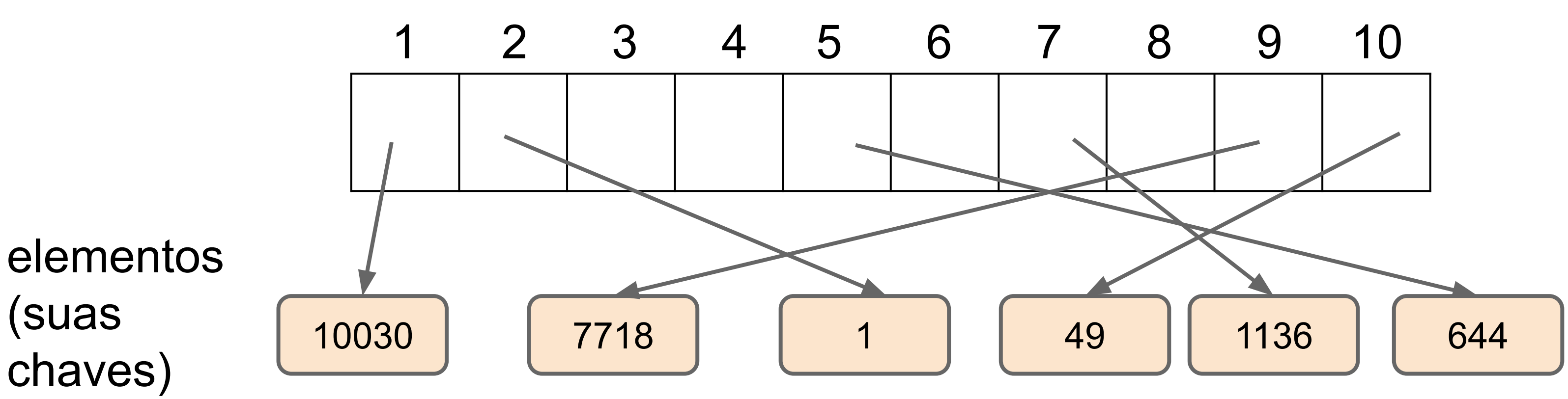
```
função Tamanho(ref T: TabelaDispersao): Inteiro
```

Obtém o número de elementos em L

Tabelas de Dispersão

- Uma ***tabela de dispersão*** para um conjunto C com N chaves é um vetor H[1..M] e uma ***função de dispersão*** h: C → {1,…,M} tal que o elemento de chave c ∈ C é armazenado em H[h(c)]

var H[1..10]: <TElem>, h(x) = x **mod** 10 + 1 // C ⊂ ℕ



Tabelas de Dispersão

- Numa tabela de dispersão, podem haver duas chaves x e y tais que $h(x) = h(y)$. Desta maneira, se x for inserido na tabela e, em um momento posterior, y for inserido, haverá uma ***colisão***

- Uma tabela de dispersão precisa especificar também como fará o tratamento das colisões!

Funções de Dispersão (características essenciais)

Tabelas de Dispersão

- Características de uma função adequada:
  - facilmente computável
    - se para conseguir a vantagem do uso da tabela de dispersão for dispendioso a computação, o ganho poderá ser comprometido
      - $h(x) = x \bmod M + 1$ é facilmente computável

Tabelas de Dispersão

- Características de uma função adequada:
  - ser uniforme
    - para qualquer subconjunto S de C "típico", tal que |S| = N, cada posição H[1..M] recebe cerca de N/M chaves
      - Exemplo: h(x) = x mod 10 + 1 é uma função de dispersão uniforme para conjuntos de números consecutivos, mas não se, por exemplo, o conjunto de chaves pares é considerado um subconjunto típico

Funções de Dispersão (métodos clássicos)

Tabelas de Dispersão

- **Método da Divisão**
    - $h(x) = x \bmod M + 1$
        - Fácil de computar? Sim!
        - Uniforme? Nem sempre. Exemplos:
            - Se M é par, então x é par $\Leftrightarrow$ h(x) é ímpar
            - Se $M = b^i$ onde b é a base da numeração, então h(x) - 1 é o número formado pelos últimos i algarismos de x
            - Se:
                - M é um primo que não esteja perto de uma potência de 2; ou
                - M não possui divisores primos menores que 20,

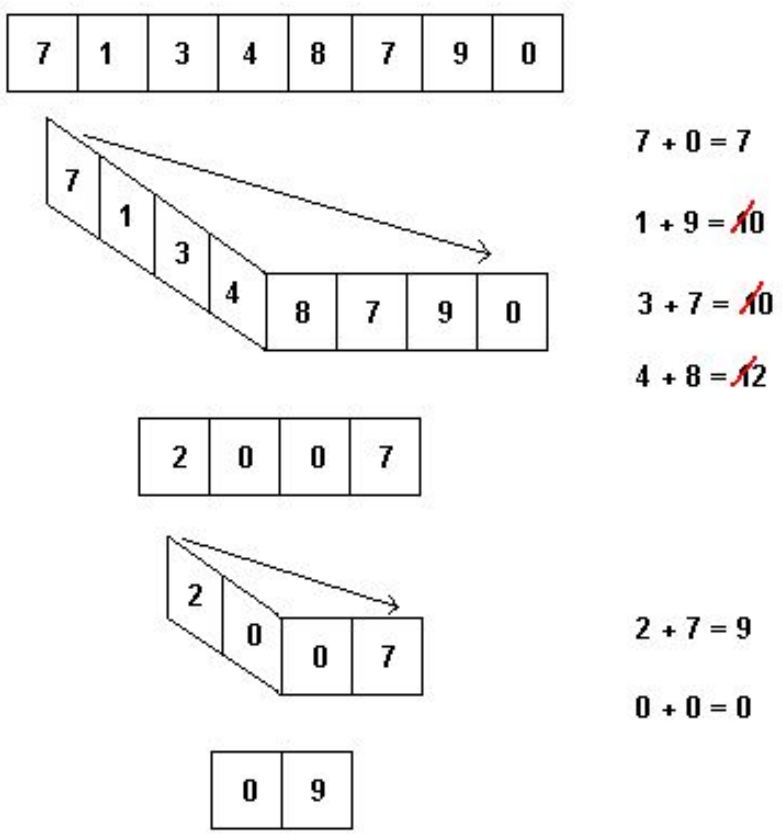
              então empiricamente comprova-se boa uniformidade

Tabelas de Dispersão

- **Método da Dobra**
    - Dobre os algarismos da representação numérica de x pela metade de forma que o último dígito coincida com o primeiro. Cada par de algarismos que são postos um por cima do outro são somados e eventuais "vai-um" são descartados, formando novos algarismos para a respectiva posição. Repita esta operação até que o número gerado seja menor ou igual a M

Tabelas de Dispersão

- **Método da Dobra**



fonte: http://www.lcad.icmc.usp.br/~nonato/ED/Hashing/node36-b.html

Tabelas de Dispersão

- **Método da Dobra**

  Java utiliza uma função de dispersão desta categoria

```
/**
 * Applies a supplemental hash function to a given hashCode, which
 * defends against poor quality hash functions.  This is critical
 * because HashMap uses power-of-two length hash tables, that
 * otherwise encounter collisions for hashCodes that do not differ
 * in lower bits. Note: Null keys always map to hash 0, thus index 0.
 */
static int hash(int h) {
    // This function ensures that hashCodes that differ only by
    // constant multiples at each bit position have a bounded
    // number of collisions (approximately 8 at default load factor).
    h ^= (h >>> 20) ^ (h >>> 12);
    return h ^ (h >>> 7) ^ (h >>> 4);
}
```

http://stackoverflow.com/questions/9364134/what-hashing-function-does-jav
a-use-to-implement-hashtable-class

Tabelas de Dispersão

- **Método da Multiplicação**
  - Chave é multiplicada por ela mesma ou uma constante resultando no número A e os dígitos necessários para compor h(x) são retirados da parte central da representação numérica de A

Tabelas de Dispersão

- **Método da Análise de Dígitos**
  - Utiliza-se alguma análise estatística das chaves. Por exemplo, calcula-se o desvio padrão da amostra em relação ideal de que cada posição decimal deveria ter N/10 chaves com valor 0, 1, …, 9. Se M necessita de d dígitos, toma-se então as d posições decimais com os menores desvios-padrão para formar a chave.
    - Exemplo, suponha que os melhores dígitos para se usar são o 3.º, 5.º, e 7.º menos significativos. Então se x = 82359104, então h(x) = 251

Tabelas de Dispersão

- **Método da Análise de Dígitos**
  - **Desvantagem**: ou tem que se conhecer todas as chaves a priori para se fazer a análise estatística, ou então ela deve ser recomputada a cada nova chave. No primeiro caso, pode ser impossível saber de antemão as chaves que serão usadas. No segundo caso, a função pode não atender o quesito de ser fácil de computar.

Tratamento de Colisões

Tabelas de Dispersão

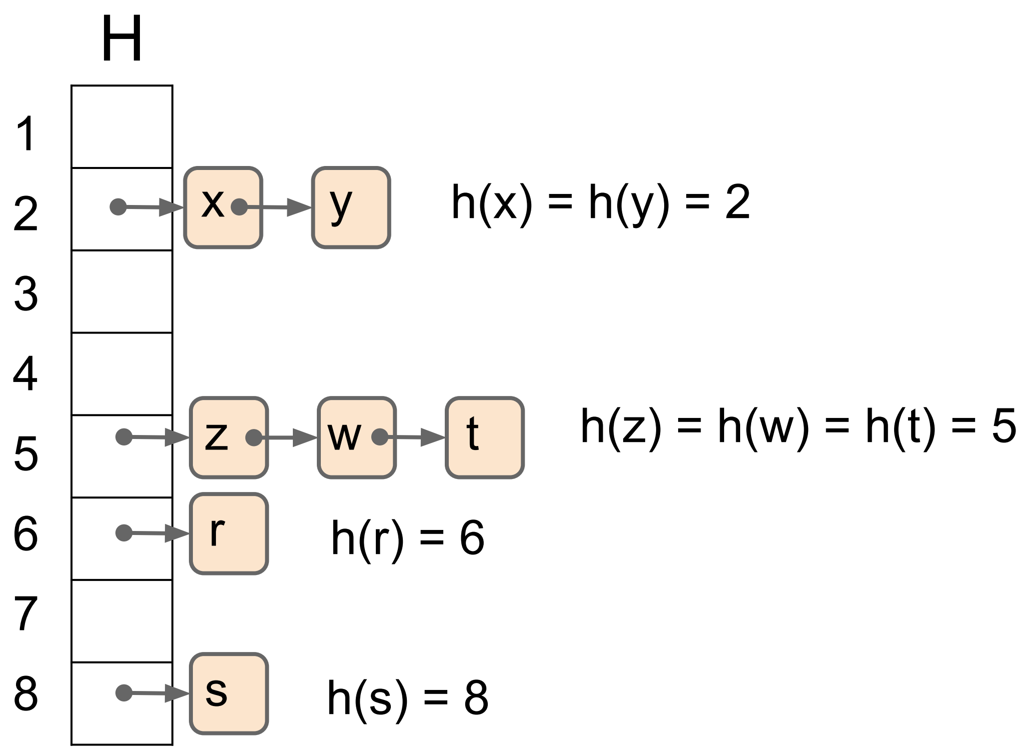
- Se $N > M$, não há como evitar colisões. Mesmo que $N \leq M$, duas chaves podem ser distribuídas para a mesma posição

- Como fazer o tratamento de colisão?
  - Encadeamento Exterior (HashTable's em Java)
  - Encadeamento Interior
  - Endereçamento Aberto (dicionários em Python)

- Seja $\alpha = N/M$ o ***fator de carga***

Tabelas de Dispersão

- **Encadeamento Exterior**
  - H[1..M] é um vetor de listas lineares tal que se x é um elemento da lista linear H[k], então h(x) = k



Tabelas de Dispersão

Espaço:
θ(M+N)
N = # de elementos

```
var M: Inteiro ← <TAMANHO DA DISPERSÃO>

estrutura TabelaDispersao <TChave, TElem>:
    H[1..M]: ListaLinear <TChave, TElem>
             //listas não-ordenadas com alocação encadeada
    N: Inteiro

procedimento Constroi(ref T: TabelaDispersao)
    para i ← 1 até M faça
        Constroi(T.H[i])
    T.N ← 0

procedimento Destroi(ref T: TabelaDispersao)
    para i ← 1 até M faça
        Destroi(T.H[i])
```

Tabelas de Dispersão

```
função Busca(ref T: TabelaDispersao, c: <TChave>): <TElem>
    retornar (Busca(T.H[h(c)], c))

procedimento Insere(ref T: TabelaDispersao,
                    c:<TChave>, x:<TElem>)
    Insere(T.H[h(c)], c, x)
    T.N ← T.N + 1

função Remove(ref T: TabelaDispersao, c: <TChave>): <TElem>
    T.N ← T.N - 1
    retornar (Remove(T.H[h(c)], c))

função Tamanho(ref T: TabelaDispersao): Inteiro
    retornar T.N
```

Tabelas de Dispersão

- **Encadeamento Exterior**

Teorema:
Numa tabela de dispersão uniforme com encadeamento exterior, o tamanho de cada lista encadeada tem tamanho médio $\theta(\alpha)$

Tabelas de Dispersão

- **Encadeamento Exterior**

Corolário:
Numa tabela de dispersão uniforme com encadeamento exterior, **se $M = \theta(N)$**, as operações de inserção, busca e remoção são executadas em tempo médio $\theta(1)$

Tabelas de Dispersão

```
função Busca(ref T: TabelaDispersao, c: <TChave>): <TElem>
    retornar (Busca(T.H[h(c)], c))

procedimento Insere(ref T: TabelaDispersao,
                    c:<TChave>, x:<TElem>)
    Insere(T.H[h(c)], c, x)
    T.N ← T.N + 1

função Remove(ref T: TabelaDispersao, c: <TChave>): <TElem>
    T.N ← T.N - 1
    retornar (Remove(T.H[h(c)], c))

função Tamanho(ref T: TabelaDispersao): Int
    retornar T.N
```

Tempo:
(Insere, Busca, Remove)
Caso Médio: $\theta(\alpha)$
Pior Caso: $\theta(N)$

Tabelas de Dispersão

```
função Busca(ref T: TabelaDispersao, c: <TChave>): <TElem>
    retornar (Busca(T.H[h(c)], c))

procedimento Insere(ref T: TabelaDispersao,
                    c:<TChave>, x:<TElem>)
    Insere(T.H[h(c)], c, x)
    T.N ← T.N + 1

função Remove(ref T: TabelaDispersao, c: <TChave>): <TElem>
    T.N ← T.N - 1
    retornar (Remove(T.H[h(c)], c))

função Tamanho(ref T: TabelaDispersao): Int
    retornar T.N
```

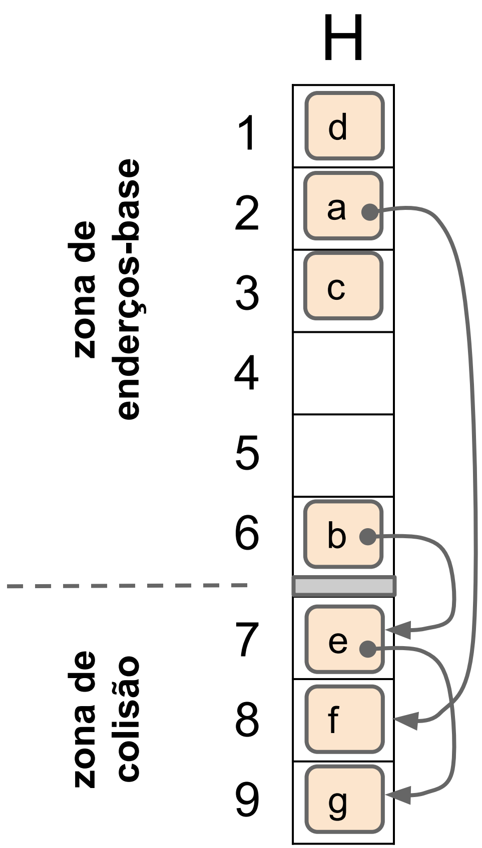
se M = O(N):
Tempo:
(Insere, Busca, Remove)
Caso Médio: θ(1)
Pior Caso: θ(N)

Tabelas de Dispersão

- **Encadeamento Interior**
  - Usada quando a memória alocada para a tabela de dispersão não pode crescer
    - $\alpha = N/M \leq 1$
  - Abordagens
    - **com** separação das colisões
    - **sem** separação das colisões

Tabelas de Dispersão

- **Encadeamento Interior**
  - Com separação de colisão



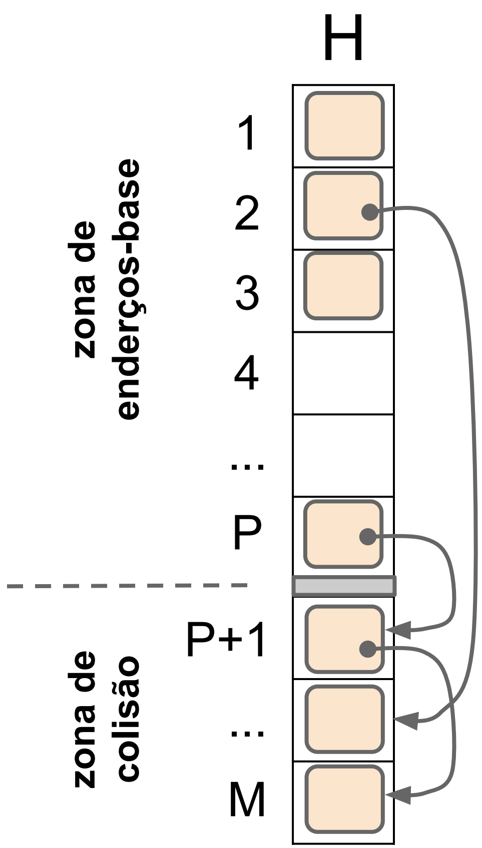
Quando há colisão, o elemento vai para a primeira posição livre da zona de colisão

Ex.:
Inserções das chaves a, b, c, d, e, f, g (nesta ordem) tais que:

- h(a) = h(f) = 2
- h(b) = h(e) = h(g) = 6
- h(c) = 3
- h(d) = 1

Tabelas de Dispersão

- **Encadeamento Interior**
  - Com separação de colisão

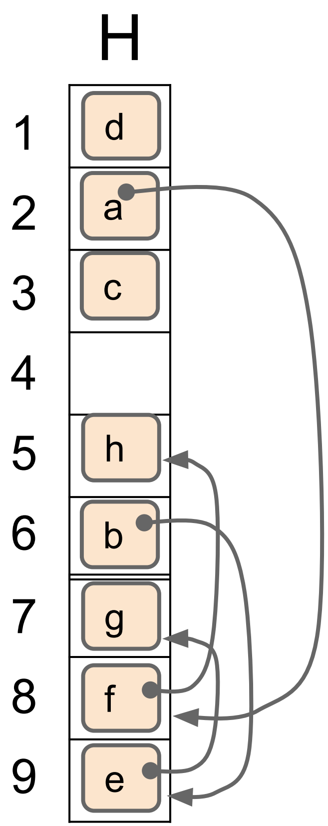


Encadeamento Interior se transforma gradualmente em uma:

- lista linear com alocação encadeada, se P → 1
- tabela de acesso direto, se P → M

Tabelas de Dispersão

- **Encadeamento Interior**
  - Sem separação de colisão



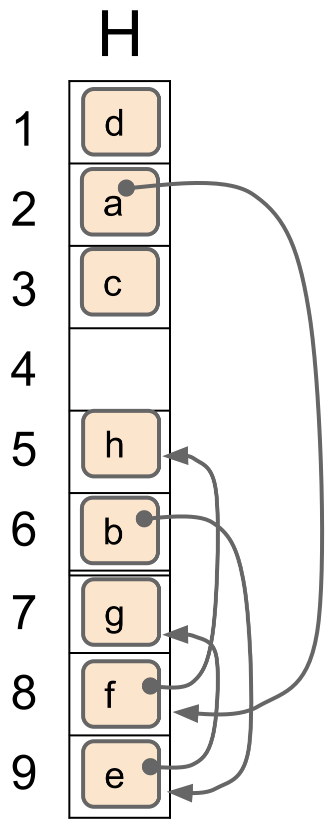
Quando há colisão, o elemento vai para a última posição livre da tabela

Ex.:
Inserções das chaves a, b, c, d, e, f, g, h (nesta ordem) tais que:

- h(a) = h(f) = h(h) = 2
- h(b) = h(e) = h(g) = 6
- h(c) = 3
- h(d) = 1

Tabelas de Dispersão

- **Encadeamento Interior**
  - Sem separação de colisão

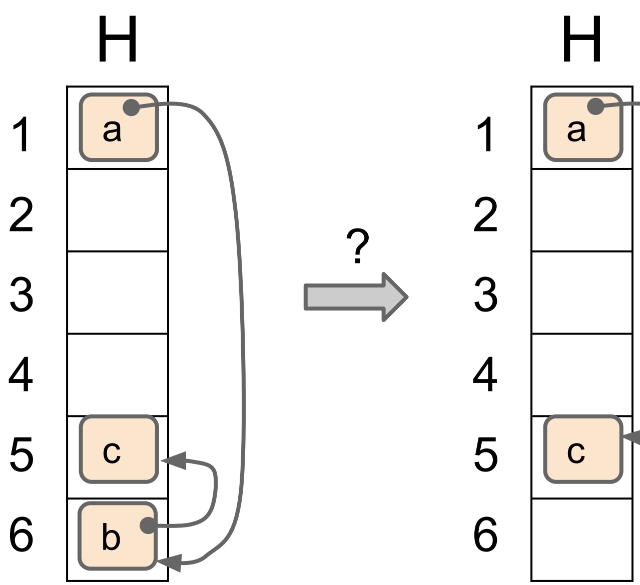


Desvantagem: união de listas de endereços-base diferentes

Ex.:
Se um novo elemento i com h(i) = 5 entrar na tabela, i será alocado na mesma lista de a, sendo que h(a) ≠ 5

Tabelas de Dispersão

- **Encadeamento Interior**
  - As operações de Inserção e Busca são diretas, se remoção não é uma operação considerada
  - Se remoção é considerada, o problema passa a ser de como separar listas fundidas

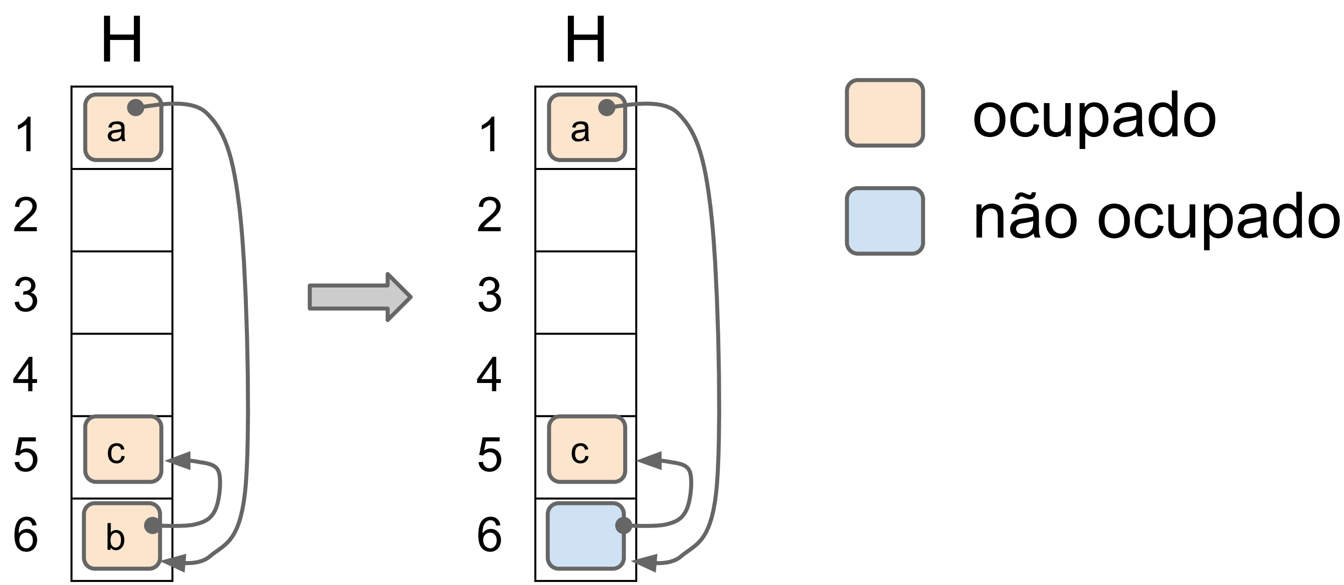


A remoção de b poderia ser simplesmente a retirada de b da lista? Dois inconvenientes:

- Não é possível manter tempo constante para decidir a última posição livre
- Suponha que h(a) = 1, h(b) = 1, h(c) = 6. Com tal remoção de b, como decidir se c pertence à tabela?

Tabelas de Dispersão

- **Encadeamento Interior**
  - **Ideia**: A remoção apenas marca uma posição como "não ocupada", cujo espaço pode ser aproveitado. Enquanto isto não ocorre, mantém a lista encadeada que existia antes da remoção do elemento.

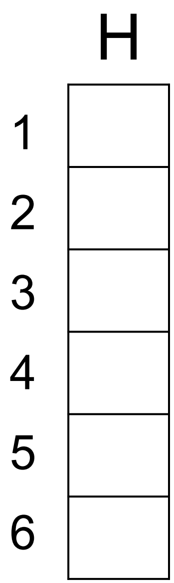


Tabelas de Dispersão

- **Endereçamento Aberto**
  - Como no Encadeamento Interior, Endereçamento Aberto também pressupõe que a memória alocada para a tabela não cresce
  - Ao contrário das Tabelas de Encadeamento Interior e Exterior que usam ponteiros para encadear as listas de colisão, a ideia é que a próxima posição para encadear uma colisão seja implícita, obtida através de um cálculo, economizando assim o espaço do ponteiro
  - Este cálculo é feito pela função de dispersão

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **h(x, k)** mapeia a posição da chave x na k-ésima tentativa de alocação



h(a,6) = h(b,6) = h(c,3) = 1

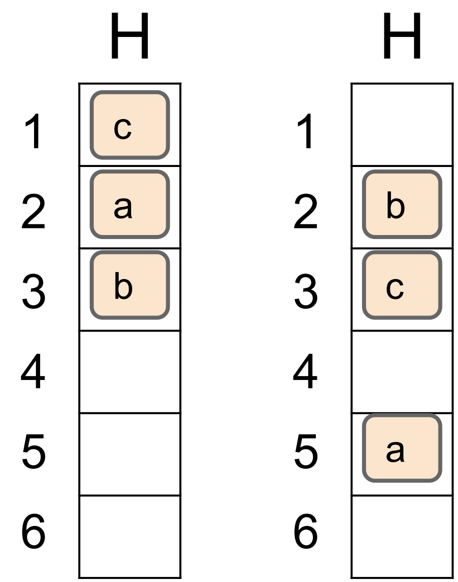
h(a,1) = h(b,1) = h(c,2) = 2

h(a,3) = h(b,2) = h(c,1) = 3

h(a,4) = h(b,3) = h(c,6) = 4

h(a,2) = h(b,4) = h(c,5) = 5

h(a,5) = h(b,5) = h(c,4) = 6



Inserção de: (na ordem) **a, b, c** **b, a, c**

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa Linear:**
    Seja $h(x)$ uma função de dispersão
    $h(x, k) = ((h(x)-1) + (k-1))$ **mod** $M + 1$

  - No slide anterior, $h(b, k)$ com $k=1,2,...,6$, $h(b)=2$ segue tentativa linear, mas não para os demais elementos! Portanto, $h(x, k)$ **não é** de tentativa linear

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa Linear:**
    Desvantagem: tendem a criar agrupamentos ao invés de dispersar as chaves (se um grupo é aumentado, a probabilidade de aumentá-lo ainda mais cresce)

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa Quadrática:**
    Seja $h(x)$ uma função de dispersão
    $h(x, k) = ((h(x)-1) + c_1(k-1) + c_2(k-1)^2)$ **mod** $M + 1$

  - No slide anterior, $h(c, k)$ com $k=1,2,...,6$, $h(c)=3$, $c_1=2$, $c_2=3$ segue tentativa quadrática, mas não para os demais elementos! Portanto, $h(x, k)$ **não é** de tentativa quadrática

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa Quadrática:**
    c_1, c_2, M devem ser bem escolhidos para que h(x,1), h(x,2), …, h(x, M) seja uma permutação de 1,2,...,M

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa com Dispersão Dupla:**
    Seja h'(x) e h''(x) funções de dispersão
    $h(x, k) = ((h'(x)-1) + (k-1)(h''(x)-1))$ **mod** $M + 1$

  - Uma melhor dispersão que aquela das tentativas linear e quadrática, pois aqui se $h(a,k) = h(b,k)$, não necessariamente $h(a,k+1) = h(b,k+1)$!

Tabelas de Dispersão

- **Endereçamento Aberto**
  - **Tentativa com Dispersão Dupla:**
    h'(x), h''(x), M devem ser bem escolhidos para que h(x,1), h(x,2), …, h(x, M) seja uma permutação de 1,2,...,M

Aplicação
Notas de Resgate (Exercício 10)

Tabelas de Dispersão

a.
```
função EhPossivelCriarNota (Texto[], Nota[]: Caracter,
                             N_A,N_B: Inteiro): Lógico
    var T_A, T_B: AcessoDireto<Inteiro>
    var M: Inteiro ← 256
    var i, c: Inteiro
    para c ← 1 até 256 faça
        Insere(T_A, c, 0); Insere(T_B, c, 0)
    para i ← 1 até N_A faça
        c ← ASCII(Texto[i])+1
        Insere(T_A, c, Remove(T_A, c) + 1)
    para i ← 1 até N_B faça
        c ← ASCII(Nota[i])+1
        Insere(T_B, c, Remove(T_B, c) + 1)
    resultado ← V
    para c ← 1 até 256 faça
        se Busca(T_A, c) ≤ Busca(T_B, c) então
            resultado ← F; sair-para
    retornar resultado
```

Tempo:
Pior Caso: $\theta(N_A+N_B)$
Espaço:
$\theta(1)$

Tabelas de Dispersão

b.

```
função EhPossivelCriarNota(Texto[], Nota[]: Caracter, N_A,N_B: Inteiro): Lógico
    var T_A, T_B: TabelaDispersao<Caracter[20], Inteiro>
    var M: Inteiro ← (N_A + N_B) div 1000; i, UPosLida: Inteiro
    var c[20]: Caracter
    UPosLida ← 0
    enquanto UPosLida ≤ N_A faça
        c[1..20] ← ObterProximaPalavra(Texto, N_A, UPosLida)
        LidaNovaPalavra(T_A, c)
    UPosLida ← 0
    enquanto UPosLida ≤ N_B faça
        c[1..20] ← ObterProximaPalavra(Nota, N_B, UPosLida)
        LidaNovaPalavra(T_B, c)
    UPosLida ← 0
    resultado ← V
    enquanto UPosLida ≤ N_B faça
        c[1..20] ← ObterProximaPalavra(Nota, N_B, UPosLida)
        se Busca(T_A, c) ≤ Busca(T_B, c) então
            resultado ← F; sair-enquanto
    retornar resultado
```

Tabelas de Dispersão

b.

```
função ObterProximaPalavra(Texto[]: Caracter, N: Inteiro,
                           ref UPosLida: Inteiro): Caracter[20]
    Inicio, UPosLida ← UPosLida+1, UPosLida+1
    enquanto UPosLida ≤ N e Texto[UPosLida] ≠ " " faça
        UPosLida ← UPosLida + 1
    retornar Texto[Inicio..UPosLida-1]

procedimento LidaNovaPalavra(T: TabelaDispersao, Texto[]: Caracter)
    se Busca(T, Texto) = "NULO" então
        Insere(T, Texto, 1)
    senão
        Insere(T, Texto, Remove(T, Texto) + 1)
```

Tabelas de Dispersão

```
b.  função EhPossivelCriarNota(Texto[], Nota[]: Caracter, N_A,N_B: Inteiro): Lógico
        var T_A, T_B: TabelaDispersao<Caracter[20], Inteiro>
        var M: Inteiro ← (N_A + N_B) div 1000; i, UPosLida: Inteiro
        var c[20]: Caracter
        UPosLida ← 0
        enquanto UPosLida ≤ N_A faça
            c[1..20] ← ObterProximaPalavra(Texto, N_A, UPosLida)
            LidaNovaPalavra(T_A, c)
        UPosLida ← 0
        enquanto UPosLida ≤ N_B faça
            c[1..20] ← ObterProximaPalavra(Nota, N_B, UPosLida)
            LidaNovaPalavra(T_B, c)
        UPosLida ← 0
        resultado ← V
        enquanto UPosLida ≤ N_B faça
            c[1..20] ← ObterProximaPalavra(Nota
            se Busca(T_A, c) < Busca(T_B, c) então
                resultado ← F, sair-enquanto
        retornar resultado
```

Tempo:
como $M = \theta(N_A+N_B)$,
$\alpha_A = \theta(1)$, $\alpha_B = \theta(1)$
Caso Médio: $\theta(N_A+N_B)$

Espaço:
$\theta(N_A+N_B)$

Exercícios

Exercícios

1. Escreva os algoritmos (e calcule as complexidades de tempo e espaço) de inserção e busca em tabelas de dispersão com encadeamento interior sem separação de colisões nos seguintes casos:
   a. a operação de remoção não é permitida
   b. a operação de remoção é permitida e escreva também o algoritmo de remoção

2. Escreva os algoritmos (e calcule as complexidades de tempo e espaço) de Inserção, Busca e Remoção em tabelas de dispersão com endereçamento aberto:
   a. geral (utilize a função h(x, k))
   b. com dispersão linear (no lugar da função h(x, k), utilize sua definição, especializando e simplificando os algoritmos do item (a))

Exercícios

3. Escreva um algoritmo que leia N registros, cada um contendo a matrícula e o nome de um aluno. Em seguida, o algoritmo deve ler M matrículas e, para cada matrícula, deve escrever o nome do aluno correspondente, buscando nos registros previamente lidos. O número de alunos (N) e de consultas à matrículas (M) será muito grande, de modo que nenhum algoritmo de tempo quadrático é aceitável. O algoritmo, portanto, deve ter complexidade média de tempo linear, isto é, $\theta(N + M)$. Resolva este problema para os seguinte casos:
   a. N faz parte da entrada: N é lido inicialmente
   b. N não faz parte da entrada: os registros de alunos terminam quando o valor -1 é lido como próxima matrícula

4. Dado um vetor de N inteiros, determinar se há elementos repetidos usando espaço auxiliar O(N) e em tempo:
   a. de pior caso O(N) se os números variam entre 10 e 1000N
   b. de caso médio O(N) se os números são aleatórios

Exercícios

5. Escreva um algoritmo que leia um texto e conte a frequência de cada palavra. O algoritmo deve ter complexidade de tempo O(N) (caso médio), onde N é o número de caracteres do texto. Sugestão: baixe um livro de domínio público em formato texto e execute seu programa.

6. Duas palavras são anagramas se uma palavra pode se tornar igual a outra por um rearranjo na ordem de suas letras. Por exemplo, as palavras computação e taãopmoçuc são anagramas. Dados dois vetores A[1..N]: Caracter e B[1..N]: Caracter representando duas palavras com N caracteres, determinar se são anagramas em tempo O(N) e espaço auxiliar constante.

Exercícios

7. Dado um vetor A[1..N] de naturais entre 1 e N, sabe-se que os valores no vetor estão em número par de ocorrências, exceto por um valor. Determinar o valor com número ímpar de ocorrências em tempo O(N) e espaço auxiliar que ocupa aproximadamente 32 vezes menos espaço que aquele ocupado por A.

8. Dada uma tabela M[1..N, 1..N]: Inteiro, zerar todos os valores numa linha e numa coluna de M que possua originalmente algum valor 0, ou seja, atribuir 0 a todos os valores da linha i e da coluna j se M[i, j] = 0. O algoritmo deve executar em tempo $O(N^2)$ e espaço auxiliar O(N).

9. Dado um vetor B com N ≤ 4 bilhões de naturais, encontrar um natural que falte em B. Ex.: Entrada: N = 4; B = [1, 4, 1000, 711]; Saída: 601 (qualquer natural que não esteja em B). O algoritmo deve executar em tempo O(N) e espaço auxiliar: (a) 1 GB; (b) 128 KB

Exercícios

10. Uma nota de resgate (do inglês, *ransom note*) é uma mensagem escrita com recortes de um texto fonte (um livro, uma revista, etc.) de forma que a grafia da mensagem não revele a identidade do criminoso. Escreva um algoritmo que determine se é possível criar uma nota de resgate a partir de um texto. Como entrada, o algoritmo recebe um vetor de caracteres A representando o texto fonte e outro vetor de caracteres B representando o texto da nota de resgate. O algoritmo deve escrever se é possível criar tal nota de resgate a partir deste texto fonte no seguintes casos:
    a. podemos recortar letras individuais do texto fonte
    b. podemos recortar apenas palavras inteiras do texto fonte

    O algoritmo deve ter complexidade de tempo $O(N_A + N_B)$, onde N_A e N_B são respectivamente o número de letras do texto fonte e da nota de resgate.