Tabelas de Dispersão

Fábio Henrique Viduani Martinez

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Estruturas de Dados

Conteúdo da aula

1. Introdução
2. Tabelas de acesso direto
3. Introdução às tabelas de dispersão
4. Tratamento de colisões com listas lineares encadeadas
5. Endereçamento aberto
6. Exercícios

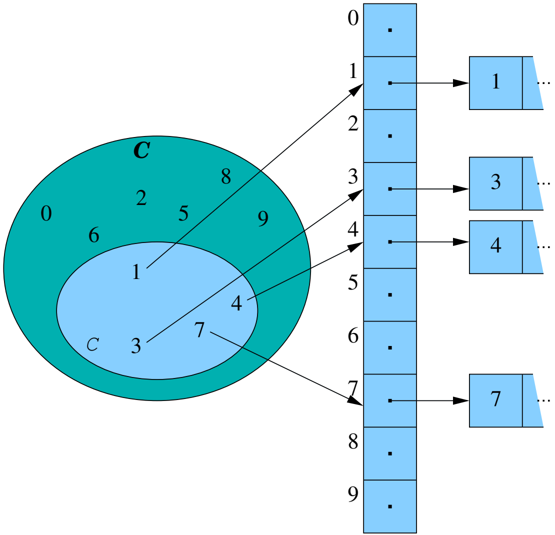
Introdução

- Conceito de dispersão foi primeiro usado por Hans P. Luhn em 1953
- Robert H. Morris usou-o em 1968, tornando o termo dispersão formal
- dispersão, espalhamento, do inglês *hash* ou *hashing*
- analogia com a palavra da língua inglesa que significa “corte e misture” ou “pique e misture”
- implementação de uma tabela de dispersão para tratar das operações básicas de busca, inserção e remoção
- uma tabela de dispersão é uma generalização da noção de vetor

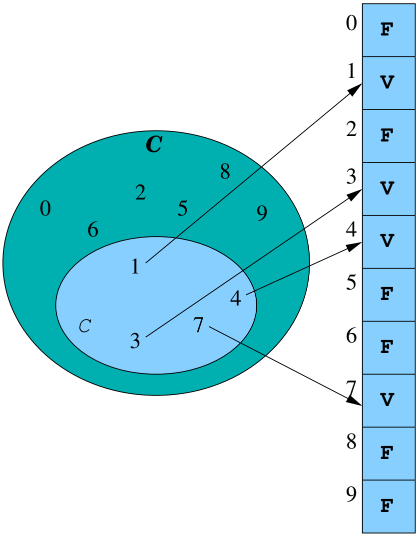
Tabelas de acesso direto

- uma aplicação em que cada informação é identificada por uma chave e que as operações essenciais realizadas sobre essas informações sejam a busca, inserção e remoção
- as chaves são numéricas
- suponha que todas as chaves possíveis na aplicação pertençam ao conjunto $\boldsymbol{C} = \{0, 1, \ldots, m-1\}$, onde m é um número inteiro relativamente pequeno
- usamos uma **tabela de acesso direto** para representar esse conjunto de informações, que nada mais é que um vetor $T[0..m-1]$ do tipo inteiro, onde o índice de cada compartimento de T corresponde a uma chave do conjunto $\boldsymbol{C}$
- durante a execução da aplicação, um determinado conjunto de chaves $C \subseteq \boldsymbol{C}$ está representado na tabela T

Tabelas de acesso direto



Tabelas de acesso direto



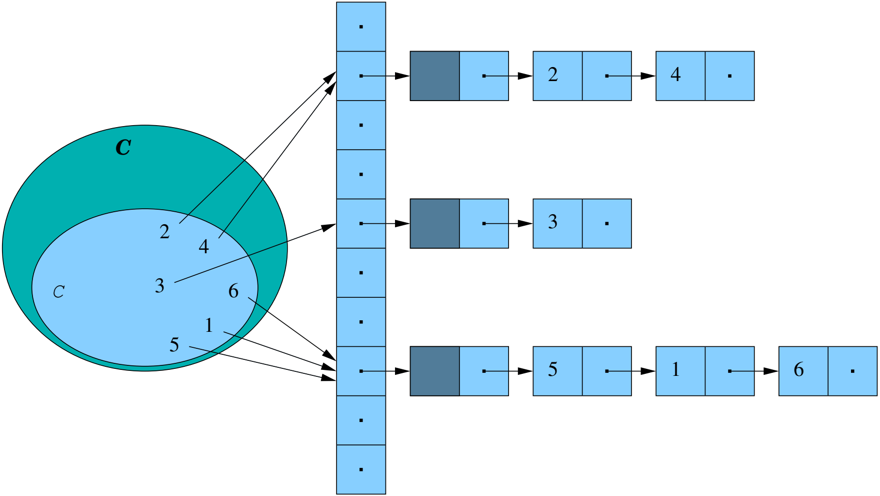
Tabelas de acesso direto

- operações básicas de busca, remoção e inserção são realizadas em tempo constante
- impossibilidade de alocação do vetor T quando o conjunto de todas as possíveis chaves $\boldsymbol{C}$ contém alguma chave que é um número inteiro muito grande
- além disso, se há poucas chaves representadas no conjunto C em um dado instante, a tabela de acesso direto terá muito espaço alocado mas não usado

Introdução às tabelas de dispersão

- em uma tabela de dispersão, uma chave x é armazenada no compartimento $h(x)$
- a função $h\colon \boldsymbol{C} \to \{0, 1, \ldots, m-1\}$ é chamada uma **função de dispersão**
- a função h mapeia as chaves de $\boldsymbol{C}$ nos compartimentos de uma **tabela de dispersão** $T[0..m-1]$
- o objetivo de uma função de dispersão é reduzir o intervalo de índices da tabela de dispersão

Introdução às tabelas de dispersão



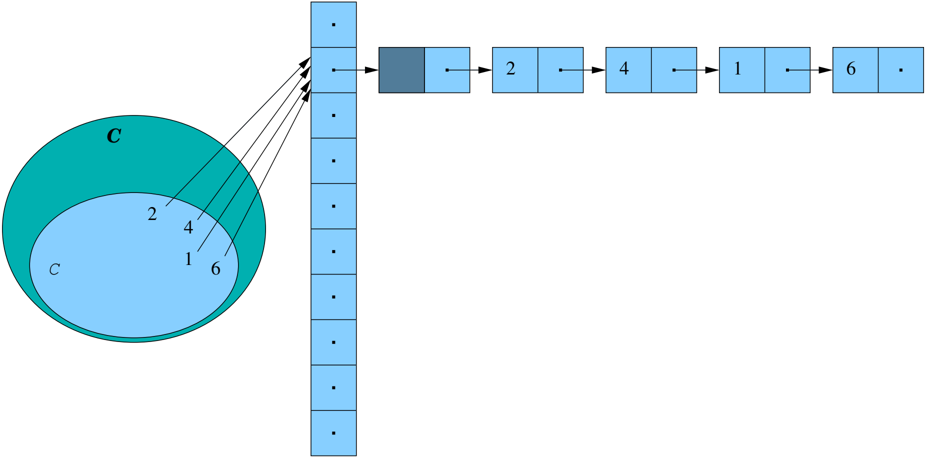
Introdução às tabelas de dispersão

- fazer uso de uma boa função de dispersão, permitindo distribuir bem as chaves pela tabela
- como $|C| > m$, duas ou mais chaves certamente terão o mesmo índice na tabela
- devem existir pelo menos duas chaves, digamos x_i e x_j, no conjunto das chaves possíveis C tais que $h(x_i) = h(x_j)$ (**colisão**)
- uma maneira de solucionar o problema das colisões é manter uma lista linear encadeada com cabeça para cada subconjunto de colisões

Introdução às tabelas de dispersão

- se a função de dispersão é muito ruim, podemos ter um caso degenerado em que todas as chaves têm o mesmo índice
- todas as operações básicas, de busca, remoção e inserção, gastam tempo proporcional ao número de chaves contidas na lista, isto é, tempo proporcional a n

Introdução às tabelas de dispersão

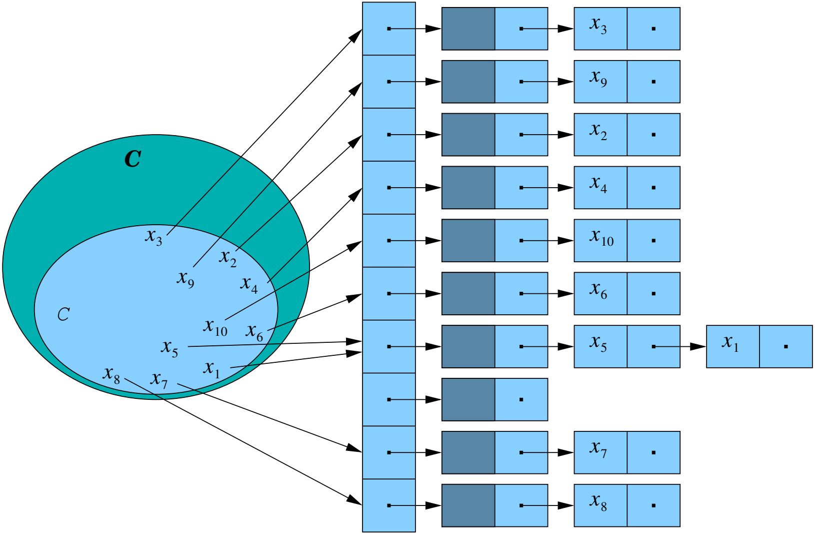


Introdução às tabelas de dispersão

Situação ideal:

- função de dispersão que maximiza o número de compartimentos usados em T e
- que minimiza os comprimentos das listas lineares encadeadas que armazenam as colisões

Introdução às tabelas de dispersão



Tratamento de colisões com listas encadeadas

- uma maneira intuitiva e eficiente de tratar colisões
- operações básicas de busca, remoção e inserção são operações de busca, remoção e inserção sobre listas lineares encadeadas
- modificamos levemente a operação de inserção, que em uma tabela de dispersão é sempre realizada no início da lista linear
- o tempo de execução no pior caso da operação de busca é proporcional ao tamanho da lista linear encadeada associada
- no pior caso tal lista contém todas as n chaves de C e, assim, o tempo de execução de pior caso da operação de busca é proporcional a n

Tratamento de colisões com listas encadeadas

- o **tempo de execução no caso médio** de uma busca em uma tabela de dispersão é proporcional a $1 + \alpha$
- α é chamado de **fator de carga** da tabela, definido como n/m, onde m é o tamanho da tabela de dispersão T e n é a quantidade de chaves em T
- se n é proporcional a m, então as operações básicas são realizadas em tempo esperado constante

Funções de dispersão

- o ideal é que uma função de dispersão seja **simples e uniforme**: cada chave é igualmente provável de ser armazenada em qualquer um dos m compartimentos da tabela de dispersão T, independentemente de onde qualquer outra chave foi armazenada antes
- necessidade de conhecimento da distribuição de probabilidades da qual as chaves foram obtidas, o que em geral não sabemos previamente
- uso de heurísticas para projetar funções de dispersão: não há garantia alguma de que uma tal função seja simples e uniforme, mas muitas delas são usadas na prática e funcionam bem na maioria dos casos
- quando a aplicação exige maior precisão, usamos o método universal para gerar uma classe de funções de dispersão que satisfazem essa propriedade

Funções de dispersão

- **método da divisão**:
  - $h(x) = x \bmod m$
  - escolhemos m um número primo grande não muito próximo a uma potência de 2
- **método da multiplicação**:
  - $h(x) = \left\lfloor m \left(xA - \lfloor xA \rfloor \right) \right\rfloor$
  - $m = 2^p$ para algum número inteiro p
  - $0 < A < 1$, $A \approx (\sqrt{5} - 1)/2 = 0{,}6180339887\ldots$
- outros métodos:
  - método da dobra
  - método da análise dos dígitos
  - método universal

Endereçamento aberto

- chaves são todas armazenadas na própria tabela
- cada compartimento ou contém uma chave ou um valor que indica que o compartimento está vazio
- em uma busca, examinamos sistematicamente os compartimentos da tabela até que a chave desejada seja encontrada ou que fique claro que a chave não consta na tabela
- computamos a sequência de compartimentos a serem examinados
- em uma inserção, examinamos sucessivamente a tabela até que um compartimento vazio seja encontrado
- a sequência de compartimentos examinada depende da chave a ser inserida e não da ordem dos índices $0, 1, \ldots, m-1$

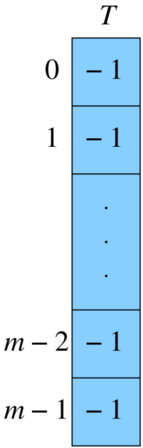
Endereçamento aberto

- a função de dispersão tem domínio e contra-domínio definidos como:

$$h\colon \boldsymbol{C} \times \{0,1,\ldots,m-1\} \to \{0,1,\ldots,m-1\}\,.$$

- é necessário que para toda chave x, a **sequência de tentativas** dada por $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ seja uma permutação de $\langle 0,1,\ldots,m-1\rangle$
- a tabela é inicializada com -1 em cada um de seus compartimentos, indicando que todos estão vazios

Endereçamento aberto



Endereçamento aberto

```
int insere_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j] == -1) {
         T[j] = x;
         return j;
      }
      else
         i++;
   } while (i != m);

   return m;
}
```

Endereçamento aberto

```
int busca_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j] == x)
         return j;
      else
         i++;
   } while (T[j] != -1 && i != m);

   return m;
}
```

Endereçamento aberto

- Problema: remoção de uma chave em um compartimento i da tabela T não permite que o compartimento i seja marcado com -1, já que isso acarreta problema em inserções subsequentes
- uma solução possível é fazer com que cada compartimento da tabela seja uma célula com dois campos: uma chave e um estado
- estado de uma célula pode ser um dos três: **`VAZIA`**, **`OCUPADA`** ou **`REMOVIDA`**

```
struct cel {
   int chave;
   int estado;
};

typedef struct cel celula;
```

Endereçamento aberto

T	chave	estado
0		**VAZIA**
1		**VAZIA**
	⋮	
$m-2$		**VAZIA**
$m-1$		**VAZIA**

Endereçamento aberto

```
int busca_aberto(int m, celula T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].chave == x && T[j].estado == OCUPADA)
         return j;
      else
         i++;
   } while (T[j].estado != VAZIA && i != m);

   return m;
}
```

Endereçamento aberto

```
int insere_aberto(int m, celula T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].estado != OCUPADA) {
         T[j].chave = x;
         T[j].estado = OCUPADA;
         return j;
      }
      else
         i++;
   } while (i != m);
   return m;
}
```

Endereçamento aberto

```
int remove_aberto(int m, celula T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].chave == x && T[j].estado == OCUPADA) {
         T[j].estado = REMOVIDA;
         return j;
      }
      else
         i++;
   } while (i != m && T[j].estado != VAZIA);
   return m;
}
```

Endereçamento aberto

Exemplo:

- tabela de dispersão T alocada com 10 compartimentos e contendo 6 chaves: $41, 22, 104, 96, 37, 16$
- função de dispersão usada h dada pelo método da tentativa linear: $h(x, i) = \big(h'(x) + i\big) \bmod 10$
- h' é uma função de dispersão ordinária dada pelo método da divisão: $h'(x) = x \bmod 10$

Endereçamento aberto

T

	chave	estado
0		VAZIA
1	41	OCUPADA
2	22	OCUPADA
3		VAZIA
4	104	OCUPADA
5		VAZIA
6	96	OCUPADA
7	37	OCUPADA
8	16	OCUPADA
9		VAZIA

Endereçamento aberto

- o papel da função de dispersão h é gerar uma sequência de compartimentos onde uma chave de interesse pode ocorrer
- em uma tabela de dispersão com endereçamento aberto temos sempre no máximo uma chave por compartimento
- ou seja, $n \leqslant m$, onde n é o número de chaves armazenadas e m o total de compartimentos da tabela
- assim, o fator de carga α da tabela sempre satisfaz $\alpha \leqslant 1$

Endereçamento aberto

- suponha que a tabela de dispersão é uniforme, isto é, que a sequência de tentativas $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ usada em uma operação básica é igualmente provável a qualquer permutação de $\langle 0, 1, \ldots, m-1\rangle$
  - número esperado de tentativas numa busca *sem* sucesso é

    $$\leqslant 1/(1-\alpha)\ ;$$

    por exemplo, se a tabela está preenchida metade, então o número médio de tentativas em uma busca *sem* sucesso é $\leqslant 1/(1-0.5) = 2$; se a tabela está 90% cheia, então o número médio de tentativas é $\leqslant 1/(1-0.9) = 10$
  - número esperado de tentativas numa busca *com* sucesso é

    $$\leqslant (1/\alpha)\ \ln\left(1/(1-\alpha)\right)\ ;$$

    por exemplo, se a tabela está preenchida pela metade, então o número médio de tentativas em uma busca *com* sucesso é < 1.387; se a tabela está 90% cheia, então o número médio de tentativas é < 2.559

Endereçamento aberto

- suponha que a tabela de dispersão é uniforme, isto é, que a sequência de tentativas $\langle h(x, 0), h(x, 1), \ldots, h(x, m-1)\rangle$ usada em uma operação básica é igualmente provável a qualquer permutação de $\langle 0, 1, \ldots, m-1\rangle$
  - número esperado de tentativas numa busca *sem* sucesso é
    $$\leqslant 1/(1-\alpha)\ ;$$
    por exemplo, se a tabela está preenchida metade, então o número médio de tentativas em uma busca *sem* sucesso é $\leqslant 1/(1-0.5) = 2$; se a tabela está 90% cheia, então o número médio de tentativas é $\leqslant 1/(1-0.9) = 10$
  - número esperado de tentativas numa busca *com* sucesso é
    $$\leqslant (1/\alpha)\ \ln\left(1/(1-\alpha)\right)\ ;$$
    por exemplo, se a tabela está preenchida pela metade, então o número médio de tentativas em uma busca *com* sucesso é < 1.387; se a tabela está 90% cheia, então o número médio de tentativas é < 2.559

Endereçamento aberto

- suponha que a tabela de dispersão é uniforme, isto é, que a sequência de tentativas $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ usada em uma operação básica é igualmente provável a qualquer permutação de $\langle 0, 1, \ldots, m-1\rangle$
  - número esperado de tentativas numa busca *sem* sucesso é

    $$\leqslant 1/(1-\alpha)\,;$$

    por exemplo, se a tabela está preenchida metade, então o número médio de tentativas em uma busca *sem* sucesso é $\leqslant 1/(1-0.5) = 2$; se a tabela está 90% cheia, então o número médio de tentativas é $\leqslant 1/(1-0.9) = 10$
  - número esperado de tentativas numa busca *com* sucesso é

    $$\leqslant (1/\alpha)\ \ln\big(1/(1-\alpha)\big)\,;$$

    por exemplo, se a tabela está preenchida pela metade, então o número médio de tentativas em uma busca *com* sucesso é < 1.387; se a tabela está 90% cheia, então o número médio de tentativas é < 2.559

Funções de dispersão

- gostaríamos que cada chave tivesse qualquer uma das $m!$ permutações igualmente prováveis de $\langle 0, 1, \ldots, m-1 \rangle$ como sua sequência de tentativas: **dispersão uniforme**
- funções de dispersão uniforme são difíceis de implementar e na prática são usadas algumas boas aproximações

Funções de dispersão

- **método da tentativa linear**: dada uma função de dispersão auxiliar $h' \colon \boldsymbol{C} \to \{0, 1, \ldots, m-1\}$, a função de dispersão é dada por
$$h(x, i) = \big(h'(x) + i\big) \bmod m$$
para $i = 0, 1, \ldots, m-1$
  - para uma chave x, o primeiro compartimento examinado é $T[h'(x)]$, que é o compartimento obtido pela função de dispersão auxiliar; em seguida, o compartimento $T[h'(x)+1]$ é examinado e assim por diante, até o compartimento $T[m-1]$; depois, a sequência de tentativas continua a partir de $T[0]$, seguindo para $T[1]$ e assim por diante, até $T[h'(x)-1]$
  - sofre de **agrupamento primário**

Funções de dispersão

- **método da tentativa quadrática**:

$$h(x,i) = \left(h'(x) + c_1 i + c_2 i^2\right) \bmod m$$

onde h' é uma função de dispersão auxiliar, c_1 e c_2 são constantes auxiliares e $i = 0, 1, \ldots, m-1$

  - para uma chave x, o primeiro compartimento examinado é $T[h'(x)]$; em seguida, os compartimentos examinados são obtidos pelo deslocamento quadrático de valores que dependem de i
  - a escolha dos valores c_1, c_2 e m é restrita
  - sofre de **agrupamento secundário**

Funções de dispersão

- **método de dispersão dupla**:

$$h(x, i) = \big(h_1(x) + ih_2(x)\big) \bmod m \,,$$

onde h_1 e h_2 são funções de dispersão auxiliares

  - compartimento inicialmente examinado é $T[h_1(x)]$; os compartimentos examinados em seguida são obtidos do deslocamento dos compartimentos anteriores da quantidade de $h_2(x)$ módulo m
  - o valor $h_2(x)$ deve ser um primo relativo ao tamanho da tabela m
  - permutações produzidas têm muitas das características de permutações aleatórias

Exercícios

3.1 Suponha que temos um conjunto de chaves C armazenado em uma tabela de acesso direto T de tamanho m. Escreva uma função que encontra a maior chave de C. Qual o tempo de execução de pior caso da sua função?

3.2 Suponha um conjunto de n chaves formado pelos n primeiros múltiplos de 7. Quantas colisões ocorrem mediante a aplicação das funções de dispersão abaixo?

(a) $x \bmod 7$

(b) $x \bmod 14$

(c) $x \bmod 5$

3.3 Ilustre a inserção das chaves $5, 28, 19, 15, 20, 33, 12, 17, 10$ em uma tabela de dispersão com colisões resolvidas por listas lineares encadeadas. Suponha que a tabela tenha 9 compartimentos e que a função de dispersão seja $h(x) = x \bmod 9$.

Exercícios

3.4 Considere uma tabela de dispersão de tamanho $m = 1000$ e uma função de dispersão $h(x) = \lfloor m\left(Ax - \lfloor Ax \rfloor\right) \rfloor$ para $A = (\sqrt{5} - 1)/2$. Compute as posições para as quais as chaves $61, 62, 63, 64$ e 65 são mapeadas.

3.5 Considere a inserção das chaves $10, 22, 31, 4, 15, 28, 17, 88, 59$ em uma tabela de dispersão com endereçamento aberto de tamanho $m = 11$ com função de dispersão auxiliar $h'(x) = x \bmod m$. Ilustre o resultado da inserção dessas chaves usando tentativa linear, tentativa quadrática com $c_1 = 1$ e $c_2 = 3$ e dispersão duplo com $h_2(x) = 1 + (x \bmod (m - 1))$.

Exercícios

3.6 A tabela abaixo é composta das palavras-chaves da linguagem C padrão.

`auto`	`double`	`int`	`long`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Escreva um programa que leia um arquivo contendo um programa na linguagem C e identifique suas palavras-chaves.

Exercícios

3.7 Veja animações do funcionamento de tabelas de dispersão nas páginas a seguir:
- VisuAlgo
- Open Hashing
- Hash Tables – animations that will make you understand how they work