MC202 - Estrutura de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Uma função de espalhamento (*hash function*) é qualquer função que mapeia dados de tamanho arbitrário em dados de tamanho fixo. Por exemplo:

Funções de Espalhamento

Uma função de espalhamento (*hash function*) é qualquer função que mapeia dados de tamanho arbitrário em dados de tamanho fixo. Por exemplo:

Suponha que cada letra do alfabeto (minúscula/maiúscula) vale um número de 0 a 25, $a/A = 0, b/B = 1, \ldots, z/Z = 25$, e que a palavra vale um número na base 26. Podemos mapear *strings* (nomes) em números na base 10 e calcular o resto da divisão deles por 1000. Esta função estaria mapeando qualquer palavra em um inteiro de 0 a 999.

Uma função de espalhamento (*hash function*) é qualquer função que mapeia dados de tamanho arbitrário em dados de tamanho fixo. Por exemplo:

Suponha que cada letra do alfabeto (minúscula/maiúscula) vale um número de 0 a 25, $a/A = 0, b/B = 1, \ldots, z/Z = 25$, e que a palavra vale um número na base 26. Podemos mapear *strings* (nomes) em números na base 10 e calcular o resto da divisão deles por 1000. Esta função estaria mapeando qualquer palavra em um inteiro de 0 a 999.

Ex: $Ana = (0 \times 26^2 + 13 \times 26^1 + 0 \times 26^0)\%1000 = 338$.

Agenda

- Quais os objetivos do espalhamento?
- O que é uma tabela de espalhamento?
- Quais são as aplicações?
- Qual é a dificuldade principal?
- Métodos principais de espalhamento.

Objetivos

As funções de espalhamento são usadas em aplicações que envolvem operações de dicionário (inserção, remoção, e busca).

Objetivos

As funções de espalhamento são usadas em aplicações que envolvem operações de dicionário (inserção, remoção, e busca).

- A chave de indexação da informação é mapeada em m possíveis posições de uma tabela de espalhamento.

Objetivos

As funções de espalhamento são usadas em aplicações que envolvem operações de dicionário (inserção, remoção, e busca).

- A chave de indexação da informação é mapeada em m possíveis posições de uma tabela de espalhamento.
- Como mais de uma chave pode ser mapeada em uma mesma posição da tabela (colisão), cada posição da tabela pode armazenar uma lista ligada de chaves (ou outra estrutura).

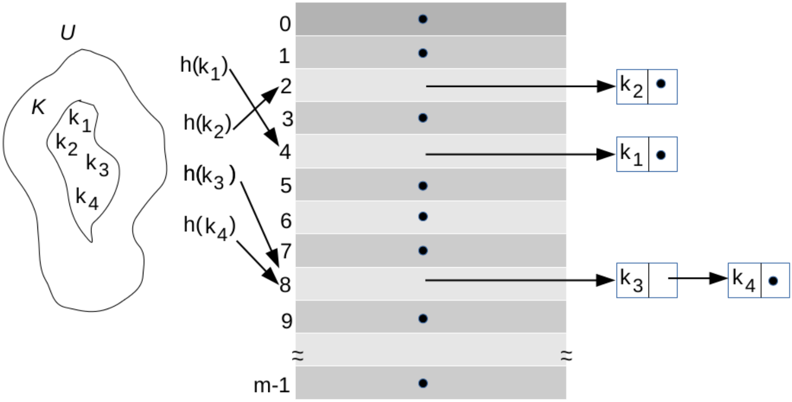
Objetivos

As funções de espalhamento são usadas em aplicações que envolvem operações de dicionário (inserção, remoção, e busca).

- A chave de indexação da informação é mapeada em m possíveis posições de uma tabela de espalhamento.
- Como mais de uma chave pode ser mapeada em uma mesma posição da tabela (colisão), cada posição da tabela pode armazenar uma lista ligada de chaves (ou outra estrutura).
- As chaves devem ser espalhadas o mais uniformemente possível em diferentes posições da tabela de modo a reduzir o custo das operações de dicionário, idealmente para $O(1)$ (caso sem colisão, espalhamento perfeito).

Tabela de espalhamento

Seja U o universo de chaves possíveis de indexação, K o conjunto das chaves de indexação, e $h(k)$ uma função de espalhamento, que associa m possíveis valores para qualquer chave $k \in K$.



As aplicações podem armazenar a informação desejada na própria tabela, em outra região de memória, ou no disco. Exemplos de aplicações são:

Aplicações

As aplicações podem armazenar a informação desejada na própria tabela, em outra região de memória, ou no disco. Exemplos de aplicações são:

- Tabela de espalhamento com símbolos de um compilador cujas chaves de indexação dos símbolos são os próprios identificadores da linguagem de programação.

Aplicações

As aplicações podem armazenar a informação desejada na própria tabela, em outra região de memória, ou no disco. Exemplos de aplicações são:

- Tabela de espalhamento com símbolos de um compilador cujas chaves de indexação dos símbolos são os próprios identificadores da linguagem de programação.
- Tabela de espalhamento como índice primário que armazena as chaves e os *offsets* de acesso aos registros de um arquivo em disco.

As aplicações podem armazenar a informação desejada na própria tabela, em outra região de memória, ou no disco. Exemplos de aplicações são:

- Tabela de espalhamento com símbolos de um compilador cujas chaves de indexação dos símbolos são os próprios identificadores da linguagem de programação.
- Tabela de espalhamento como índice primário que armazena as chaves e os *offsets* de acesso aos registros de um arquivo em disco.
- Tabela de espalhamento que armazena as senhas de vários usuários, onde o *username* é a chave de indexação.

Qual é a dificuldade principal?

O problema, portanto, consiste em encontrar uma função de espalhamento $h : U \rightarrow \{0, 1, \ldots, m-1\}$, tal que $h(k_i) \neq h(k_j)$ para todo $i, j \in \{0, 1, \ldots, n-1\}$, onde $n = |K|$, $K \subseteq U$.

Qual é a dificuldade principal?

O problema, portanto, consiste em encontrar uma função de espalhamento $h : U \rightarrow \{0, 1, \ldots, m-1\}$, tal que $h(k_i) \neq h(k_j)$ para todo $i, j \in \{0, 1, \ldots, n-1\}$, onde $n = |K|$, $K \subseteq U$.

- A solução ideal seria o espalhamento perfeito, $m = |U|$ e $h(k_i) = i$, mas esta solução não é boa.

Qual é a dificuldade principal?

O problema, portanto, consiste em encontrar uma função de espalhamento $h : U \rightarrow \{0, 1, \ldots, m-1\}$, tal que $h(k_i) \neq h(k_j)$ para todo $i, j \in \{0, 1, \ldots, n-1\}$, onde $n = |K|$, $K \subseteq U$.

- A solução ideal seria o espalhamento perfeito, $m = |U|$ e $h(k_i) = i$, mas esta solução não é boa.

- Não temos controle sobre U, então se $|U|$ for muito grande, a solução fica inviável.

Qual é a dificuldade principal?

O problema, portanto, consiste em encontrar uma função de espalhamento $h : U \rightarrow \{0, 1, \ldots, m-1\}$, tal que $h(k_i) \neq h(k_j)$ para todo $i, j \in \{0, 1, \ldots, n-1\}$, onde $n = |K|$, $K \subseteq U$.

- A solução ideal seria o espalhamento perfeito, $m = |U|$ e $h(k_i) = i$, mas esta solução não é boa.
- Não temos controle sobre U, então se $|U|$ for muito grande, a solução fica inviável.
- Se $n \ll |U|$, vai haver disperdício de memória.

Qual é a dificuldade principal?

O problema, portanto, consiste em encontrar uma função de espalhamento $h : U \rightarrow \{0, 1, \ldots, m-1\}$, tal que $h(k_i) \neq h(k_j)$ para todo $i, j \in \{0, 1, \ldots, n-1\}$, onde $n = |K|$, $K \subseteq U$.

- A solução ideal seria o espalhamento perfeito, $m = |U|$ e $h(k_i) = i$, mas esta solução não é boa.
- Não temos controle sobre U, então se $|U|$ for muito grande, a solução fica inviável.
- Se $n \ll |U|$, vai haver disperdício de memória.

Temos que escolher $m \ll |U|$ e tratar o problema inevitável de colisão.

- Escolha de uma função de espalhamento que minimiza a probabilidade de colisão.
    - Método da divisão.
    - Seleção de algarismos.

Métodos de espalhamento e tratamento de colisões

- Escolha de uma função de espalhamento que minimiza a probabilidade de colisão.
  - Método da divisão.
  - Seleção de algarismos.
- Tratamento de colisão por encadeamento.

Métodos de espalhamento e tratamento de colisões

- Escolha de uma função de espalhamento que minimiza a probabilidade de colisão.
  - Método da divisão.
  - Seleção de algarismos.
- Tratamento de colisão por encadeamento.
- Tratamento de colisão por endereçamento aberto.
  - Reespalhamento linear.
  - Reespalhamento quadrático.
  - Reespalhamento duplo.

Escolha da função de espalhamento

A função de espalhamento $h(k)$ deveria mapear as chaves $k \in K$ com mesma probabilidade $P(i) = 1/m$ para qualquer posição $i \in \{0, 1, \ldots, m-1\}$ da tabela.

Escolha da função de espalhamento

A função de espalhamento $h(k)$ deveria mapear as chaves $k \in K$ com mesma probabilidade $P(i) = 1/m$ para qualquer posição $i \in \{0, 1, \ldots, m-1\}$ da tabela.

- Se $k \in [0, 1)$, por exemplo, $h(k) = \lfloor k \times m \rfloor$ seria uma boa função. Porém, não temos controle sobre a distribuição de k, então heurísticas são aplicadas.

A função de espalhamento $h(k)$ deveria mapear as chaves $k \in K$ com mesma probabilidade $P(i) = 1/m$ para qualquer posição $i \in \{0, 1, \ldots, m-1\}$ da tabela.

- Se $k \in [0, 1)$, por exemplo, $h(k) = \lfloor k \times m \rfloor$ seria uma boa função. Porém, não temos controle sobre a distribuição de k, então heurísticas são aplicadas.

- Uma heurística é fazer com que a posição mapeada seja independente de qualquer padrão inerente das chaves. Ex: $h(k) = k\%100$ não é uma boa ideia, pois $h(k)$ será sempre os dois últimos dígitos de k.

Escolha da função de espalhamento

A função de espalhamento $h(k)$ deveria mapear as chaves $k \in K$ com mesma probabilidade $P(i) = 1/m$ para qualquer posição $i \in \{0, 1, \ldots, m-1\}$ da tabela.

- Se $k \in [0, 1)$, por exemplo, $h(k) = \lfloor k \times m \rfloor$ seria uma boa função. Porém, não temos controle sobre a distribuição de k, então heurísticas são aplicadas.
- Uma heurística é fazer com que a posição mapeada seja independente de qualquer padrão inerente das chaves. Ex: $h(k) = k\%100$ não é uma boa ideia, pois $h(k)$ será sempre os dois últimos dígitos de k.
- Outra heurística é mapear chaves próximas em posições distantes da tabela.

Escolha da função de espalhamento

Método da divisão:

$$h(k) = k\%m,$$

onde m deve ser primo e o mais distante possível de 2^b (uma potência de 2), para que $h(k)$ não corresponda aos b bits de mais baixa ordem de k.

Método da divisão:

$$h(k) = k\%m,$$

onde m deve ser primo e o mais distante possível de 2^b (uma potência de 2), para que $h(k)$ não corresponda aos b bits de mais baixa ordem de k.

Por exemplo, se o número de chaves $n = 2000$ e desejamos ter em média 3 colisões por posição da tabela, no máximo, então podemos escolher $m = 769$, que é primo, maior que $\frac{2000}{3}$, e o mais distante possível de 2^9 e 2^{10}.

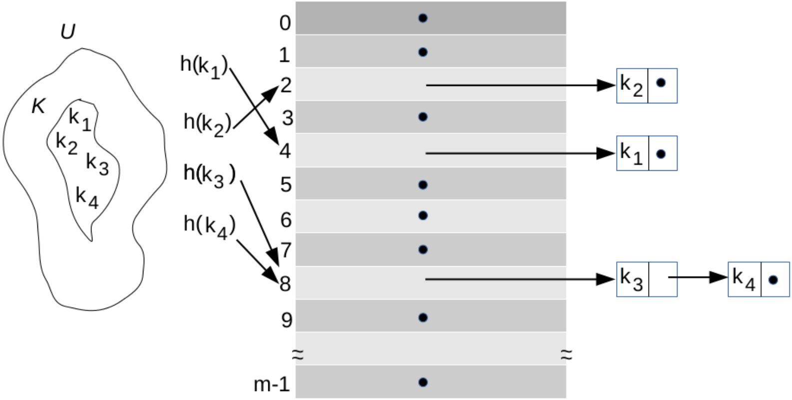
Método de Seleção de Algarismos: A chave k é vista como uma sequência de dígitos $d_0 d_1 d_2 \ldots$ e alguns deles são selecionados para formar o valor de $h(k)$.

Escolha da função de espalhamento

Método de Seleção de Algarismos: A chave k é vista como uma sequência de dígitos $d_0 d_1 d_2 \dots$ e alguns deles são selecionados para formar o valor de $h(k)$.

Por exemplo, se $k = Pedro$, na base 26, $k = 1504031714$, então podemos escolher $h(k) = d_3 d_5 d_2 = 430$.

As colisões são tratadas por listas ligadas.



Problemas do encadeamento

- Gasto de memória com ponteiros (memória $O(m + 2n)$).

Problemas do encadeamento

- Gasto de memória com ponteiros (memória $O(m+2n)$).
- Mesmo supondo que a inserção na lista é sempre no início, em $O(1)$, temos no pior caso (todas as chaves mapeadas para uma mesma posição), remoção e busca em $O(n)$.

Problemas do encadeamento

- Gasto de memória com ponteiros (memória $O(m + 2n)$).
- Mesmo supondo que a inserção na lista é sempre no início, em $O(1)$, temos no pior caso (todas as chaves mapeadas para uma mesma posição), remoção e busca em $O(n)$.
- Por outro lado, se a distribuição de chaves for uniforme, teremos um fator de carga $\alpha = \frac{n}{m}$ chaves por posição da tabela.

Problemas do encadeamento

- Gasto de memória com ponteiros (memória $O(m + 2n)$).
- Mesmo supondo que a inserção na lista é sempre no início, em $O(1)$, temos no pior caso (todas as chaves mapeadas para uma mesma posição), remoção e busca em $O(n)$.
- Por outro lado, se a distribuição de chaves for uniforme, teremos um fator de carga $\alpha = \frac{n}{m}$ chaves por posição da tabela.
- Como α pode ser pequeno, considerando agora o custo $O(1)$ do cômputo de $h(k)$, o custo de busca e remoção é reduzido para $O(1 + \alpha)$.

Problemas do encadeamento

- O custo médio de busca na tabela de espalhamento é $O(1 + \frac{\alpha}{2})$.

Problemas do encadeamento

- O custo médio de busca na tabela de espalhamento é $O(1 + \frac{\alpha}{2})$.
- Para $m = 1000$ e $n = 2000$ teremos o equivalente a 2 comparações, quando em uma árvore B com $b = 10$, considerando a busca linear em cada nó, este número médio de comparações seria $5 \log_{10}^{10^3} = 15$.

Problemas do encadeamento

- O custo médio de busca na tabela de espalhamento é $O(1 + \frac{\alpha}{2})$.

- Para $m = 1000$ e $n = 2000$ teremos o equivalente a 2 comparações, quando em uma árvore B com $b = 10$, considerando a busca linear em cada nó, este número médio de comparações seria $5 \log_{10}^{10^3} = 15$.

- Para $n = 10^6$, o custo médio da tabela é equivalente a 501 comparações, quando na mesma árvore B este número seria de 30 comparações.

Tratamento de colisão por endereçamento aberto

- A ideia é evitar gasto de memória com ponteiros, aproveitando espaços vazios na tabela de espalhamento com $m \geq n$ posições. O custo de memória é $O(m)$.

Tratamento de colisão por endereçamento aberto

- A ideia é evitar gasto de memória com ponteiros, aproveitando espaços vazios na tabela de espalhamento com $m \geq n$ posições. O custo de memória é $O(m)$.
- A estratégia é usar uma segunda função de espalhamento (*rehash function*) para os casos de colisão, a qual será chamada até encontrar uma posição disponível, limitada a m tentativas.

Tratamento de colisão por endereçamento aberto

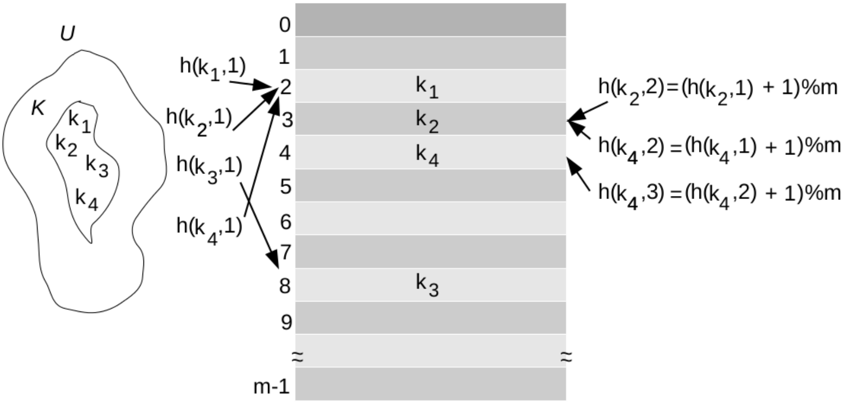
- A ideia é evitar gasto de memória com ponteiros, aproveitando espaços vazios na tabela de espalhamento com $m \geq n$ posições. O custo de memória é $O(m)$.

- A estratégia é usar uma segunda função de espalhamento (*rehash function*) para os casos de colisão, a qual será chamada até encontrar uma posição disponível, limitada a m tentativas.

- O espalhamento pode ser visto como uma sequência de $j = 1$ até m tentativas de achar uma posição vazia na tabela.

$$h(k, j) = \langle h(k, 1), h(k, 2), \ldots, h(k, m) \rangle$$

Tratamento de colisão por endereçamento aberto



- O número de inserções é limitado ao tamanho m da tabela.
- Se uma chave é removida, uma marca deve ser colocada na célula correspondente da tabela para não bloquear uma futura busca de chaves que seguem o mesmo caminho de busca.

Tipos de endereçamento aberto

- Reespalhamento linear.
- Reespalhamento quadrático.
- Reespalhamento duplo.

Seja $h(k,1) = k\%m$, o reespalhamento linear pode ser definido como

$$h(k,j) \quad = \quad (h(k,j-1)+1)\%m.$$

Isto equivale a buscar de 1 em 1 uma posição vazia na tabela.

Reespalhamento linear

Seja $h(k,1) = k\%m$, o reespalhamento linear pode ser definido como

$$h(k,j) \quad = \quad (h(k,j-1)+1)\%m.$$

Isto equivale a buscar de 1 em 1 uma posição vazia na tabela.

Um problema desta abordagem é o risco de aglomeração primária: Se todas as posições são vazias, a probabilidade de uma posição i ser escolhida é $P(i) = \frac{1}{m}$, no caso de espalhamento uniforme. Mas se esta posição for precedida de $p > 1$ outras posições já preenchidas, esta probabilidade sobe para $P(i) = \frac{p+1}{m}$.

Reespalhamento quadrático

Visando evitar a aglomeração primária, o reespalhamento quadrático considera

$$h(k,j) \quad = \quad (h(k,1) + c_1 \times (j-1) + c_2 \times (j-1)^2)\%m,$$

$c_1, c_2 \neq 0$. Porém, a técnica não evita aglomeração secundária — problema similar considerando agora as chaves que seguem o mesmo caminho de busca.

Reespalhamento duplo

O reespalhamento duplo é uma das melhores técnicas para evitar aglomerações primária e secundária, com vários variantes na literatura.

O reespalhamento duplo é uma das melhores técnicas para evitar aglomerações primária e secundária, com vários variantes na literatura.

A função de espalhamento $h(k,j)$ depende de duas funções, $h_1(k)$ e $h_2(k)$. Por exemplo:

$$\begin{aligned} h(k,j) &= (h_1(k) + (j-1) \times h_2(k))\%m, \\ h_1(k) &= k\%m, \\ h_2(k) &= 1 + (k\%m'), \end{aligned}$$

onde $m' = m - \delta$ (um número um pouco menor que m) e m deve ser primo e o mais distante possível de potência de 2.

Reespalhamento duplo

Um exemplo seria

$$h(k,j) \quad = \quad ((k\%m) + (j-1) \times (1 + k\%m'))\%m,$$

para $m = 13$ e $m' = 11$. A inserção da chave $k = 14$ por exemplo teria

- $h(14,1) = 1$,
- $h(14,2) = 1 + (1+3) = 5$,
- $h(14,3) = 1 + 2 \times (1+3) = 9, \ldots$.