"The cake is a lie!"

Funções de Hash

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Propriedades

Quais propriedades esperamos de uma boa função de hash?

Propriedades

- Eficiente (fácil) de se computar.
- Satisfaz (ao menos aproximadamente) a hipótese do hash independente e uniforme.
  - Cada chave tem uma possibilidade aproximadamente igual de ser mapeada para qualquer slot.
  - Não temos “slots preferidos”.

Hash Estático

Com hash estático, usamos uma função de hash fixa.

Métodos clássicos:

- Método da divisão
- Método da multiplicação.

Método da divisão

Dado a Tabela Hash T[0:m-1], com *m* slots, a função de hash pode ser dada por:

$h(k) = k \bmod m$

A função funciona razoavelmente bem se *m* é um **número primo** que **não está próximo de uma potência de 2.**

Quando não se conhece nada sobre a distribuição das chaves, essa pode ser uma boa ideia (assumindo *m* primo).

Método da divisão

Dado a Tabela Hash T[0:m-1], com *m* slots, a função de hash pode ser dada por:

$h(k) = k \bmod m$

A função funciona razoavelmente bem se *m* é um **número primo** que **não está próximo de uma potência de 2.**

Quando não se conhece nada sobre a distribuição das chaves, essa pode ser uma boa ideia (assumindo *m* primo).

Se *m* fosse uma potência de 2 (má ideia), estaríamos pegando apenas m os bits mais baixos do valor como seu hash.

Precisaríamos garantir que todos os padrões de *m* bits de ordem mais baixa são igualmente prováveis.

Exemplo: se m = 8, e as chaves são palavras (strings), estamos pegando o último caractere da palavra.

Método da multiplicação

Escolher uma constante 0 ‹ A ‹ 1.

O valor de hash é dado por:

h(k) = ⌊m*parteFacionaria(Ak)⌋

Nesse caso, tamanho *m* da tabela não é crítico.

Independente do valor A escolhido.

Multiplicação com Deslocamento

Método *Multiply-Shift*.

Caso especial onde o tamanho *m* da tabela é uma potência de 2.

Considerando uma máquina com palavras de tamanho *w*.

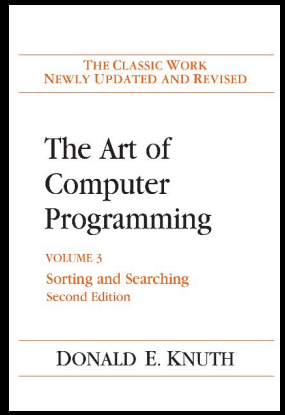
$m = 2^z$, onde $z \leq w$.

Escolher um inteiro positivo *a* de *w* bits, tal que $a = A2^w$, lembrando que $0 < A < 1$.

Logo, $0 < a < 2^w$.

Knuth (1998) sugere A = (sqrt(5)-1)/2 = 0,6180... como um bom valor genérico (funciona na maioria dos casos).

Knuth, D. The Art of Computer Programming: Volume 3: Sorting and Searching. 1998.



Exemplo

Suponha a chave $k = 123456$.

$w = 32$ (máquina de 32 bits), $z = 14$, $m = 2^{14} = 16384$.

Dessa forma, a = 2^{32} *0,6180 = 2654435769.

Exemplo

Suponha a chave $k = 123456$.

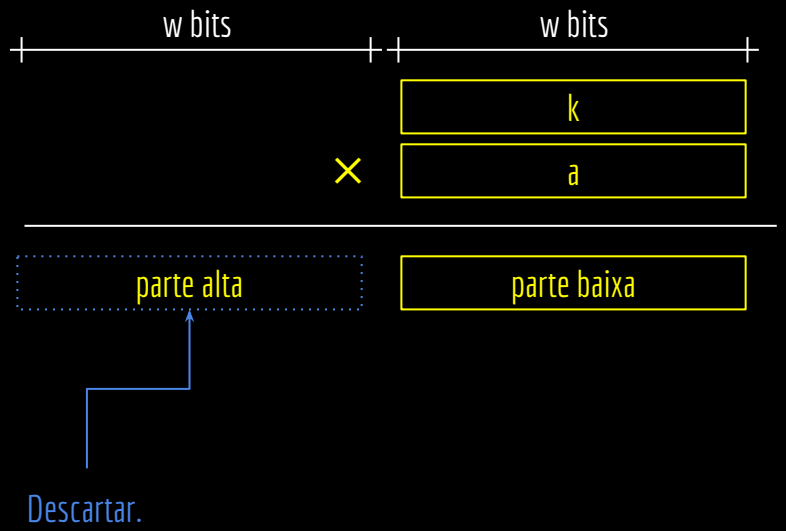
$w = 32$ (máquina de 32 bits), $z = 14$, $m = 2^{14} = 16384$.

Dessa forma, a = 2^{32} *0,6180 = 2654435769.

Multiplique k*a gerando um resultado de 2w bits.

Os w bits mais altos podem ser descartados.

k*a = *123456*2654435769 = 327706022297664



Exemplo

Suponha a chave $k = 123456$.

$w = 32$ (máquina de 32 bits), $z = 14$, $m = 2^{14} = 16384$.

Dessa forma, a = 2^{32} *0,6180 = 2654435769.

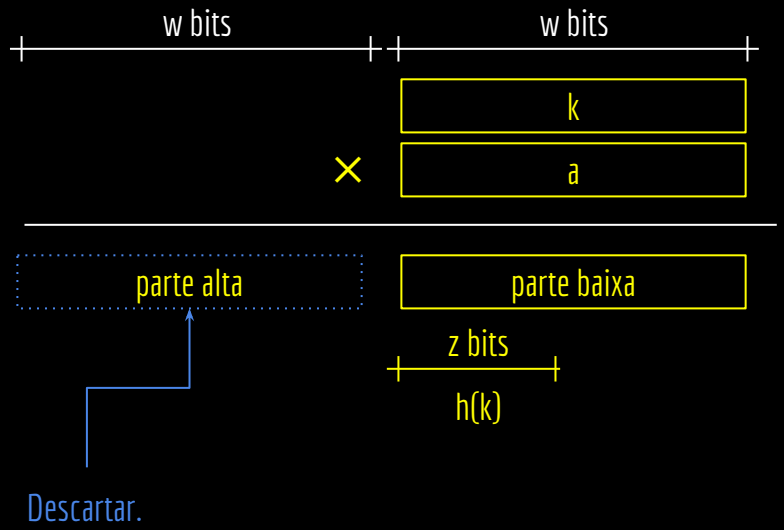
Multiplique k*a gerando um resultado de 2w bits.

Os w bits mais altos podem ser descartados.

k*a = *123456*2654435769 = 327706022297664

Os *z* bits mais significativos da parte baixa serão o hash.

No exemplo, h(k) = 67.



Exemplo

Suponha a chave $k = 123456$.

$w = 32$ (máquina de 32 bits), $z = 14$, $m = 2^{14} = 16384$.

Dessa forma, a = 2^{32} *0,6180 = 2654435769.

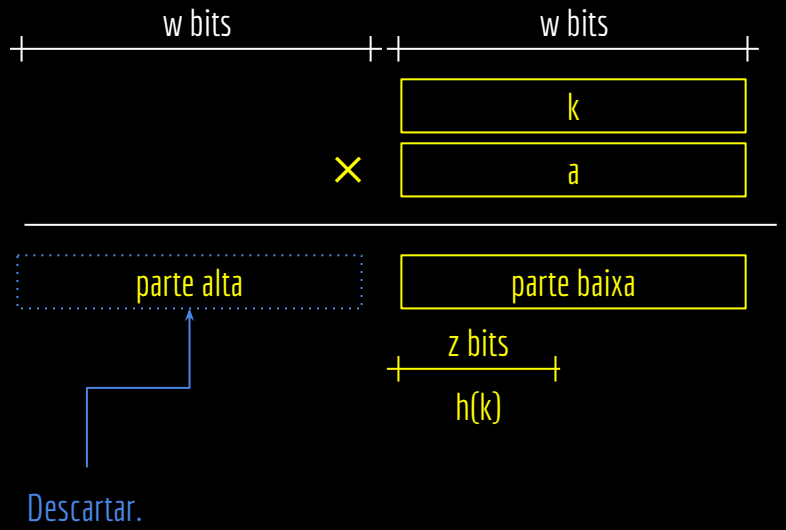
Multiplique k*a gerando um resultado de 2w bits.

Os w bits mais altos podem ser descartados.

k*a = *123456*2654435769 = 327706022297664

Os *z* bits mais significativos da parte baixa serão o hash.

No exemplo, h(k) = 67.



A operação pode ser implementada de forma eficiente como:

$h(k) = (ka \bmod 2^w) \gg (w-z)$.

O mod 2^w elimina a parte alta. Em C isso pode não ser necessário se você usar os tipos de variáveis corretos.

Hash Aleatório

Ao usar uma função de hash estática (fixa) corremos o risco de um adversário malicioso projetar chaves de forma que todas sejam mapeadas para um mesmo slot.

O tempo de busca se torna $\Theta(n)$.

Hash Aleatório

Para evitar isso, podemos implementar um Hash Aleatório.

Escolher a função de hash aleatoriamente e de forma independente das chaves.

Ideia

Criar múltiplas funções de hash.

Escolher uma das funções aleatoriamente quando o sistema é inicializado.

Universal

Seja uma família de funções de Hash *H*, que mapeiam o universo *U* de chaves para o intervalo {0, 1, ..., m-1}.

A família é **universal** se:

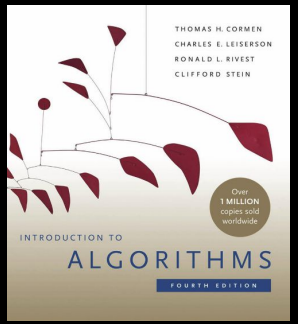
Dadas duas chaves k_1 e $k_2 \in U$, e uma função hash h $\in H$ escolhida aleatoriamente, a chance de $h(k_1)$ ser igual a $h(k_2)$ deve ser de no máximo 1/m.

Hash Universal

Dada uma família de hashes universal e resolução de colisões por encadeamento torna impossível para um adversário forçar o pior caso $\Theta(n)$.

Uma sequência s de operações de Inserção, busca e exclusão terá um custo $\Theta(s)$.

Veja a prova em Cormen (2022).



Inicialização

Uma ideia simples é escolher aleatoriamente a função de hash $h \in H$ na inicialização do programa, e usar essa função para todas operações enquanto o programa estiver executando.

Família Universal de Hashes baseada em Multiplicação

Método rápido e indicado para novas aplicações segundo Cormen et al. (2022).

Definir *H* como um conjunto de funções hash de multiplicação com deslocamento com constantes *a* ímpares.

$H = \{h_a : a \text{ é impar}, 1 \leq a < m, \text{ e } h_a \text{ possui as propriedades do Hash de multiplicação com deslocamento}\}$.

Família Universal de Hashes baseada em Multiplicação

Método rápido e indicado para novas aplicações segundo Cormen et al. (2022).

Definir *H* como um conjunto de funções hash de multiplicação com deslocamento com constantes *a* ímpares.

$H = \{h_a : a \text{ é impar}, 1 \leq a < m, \text{ e } h_a \text{ possui as propriedades do Hash de multiplicação com deslocamento}\}$.

Nesse caso, a família *H* vai ser **2/m Universal**.

Hash e-Universal:

Dadas duas chaves k_1 e $k_2 \in U$, e uma função hash $h \in H$ escolhida aleatoriamente, a chance de $h(k_1) = h(k_2)$ deve ser de no máximo *e*.

Dados de tamanho variável

Existem diversos algoritmos para criar funções de hash para dados de tamanho variável (ex.: strings).

A ideia geralmente é carregar os bytes do dado da memória em blocos, e interpretar cada bloco como um inteiro.

Exemplo: Algoritmo Fowler–Noll–Vo (FNV-1a).

Funções de Hash

Existem diversas funções de Hash prontas que você pode usar.

Funções de Hash não Criptográfico:

- Fowler–Noll–Vo hash;
- Pearson hashing;
- Jenkins hash;
- ...

Funções de Hash Criptográfico:

- SHA-256 e SHA-512;
- MD5 e MD6; ‹- MD5 é comprovadamente falho, mas ainda está na categoria de Hash Criptográfico.
- SWIFFT;
- ...

Exemplo de uso

Para usar uma função pronta para endereçar uma Tabela Hash podemos, por exemplo, fazer:

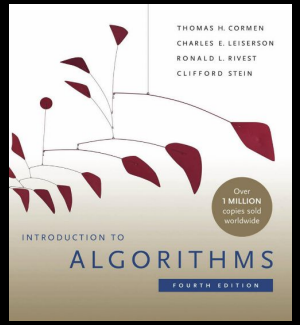
$h(k) = SHA256(k) \bmod m$.

Exercício

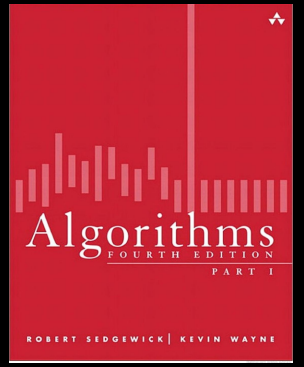
1. Implemente as funções de hash discutidas na aula em C.

Referências

T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 4a ed. 2022.

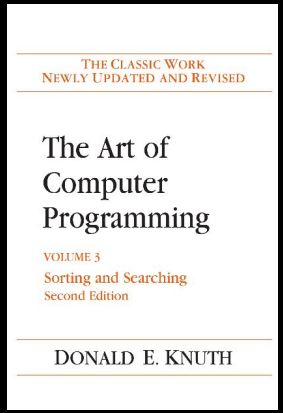


R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2011.



Estrutura de Dados e Algoritmos em C++. A. Drozdek. 4a ed. 2016.

Knuth, D. The Art of Computer Programming: Volume 3: Sorting and Searching. 1998.



Licença