CUCKOO HASHING COM PERFECT REHASH

Resumo

➢Apresentamos uma variante do Cuckoo Hashing tradicional em que o tempo de pior caso para a operação de inserção é O(sqrt(n)), ao contrário do esquema clássico, que, embora bom na média, tem tempo de execução ilimitado no pior caso. Nosso algoritmo baseia-se nos seguintes pontos:

- no uso de um número de tabelas hash que não é constante, mas cresce com o número de chaves;
- na aplicação de hashing perfeito nos momentos em que rehashes se fazem necessários.

O preço que pagamos no esquema proposto é o aumento do tempo de pior caso da consulta, que deixa de ser O(1) e passa a ser também O(sqrt(n)).

Após implementarmos o esquema proposto, pudemos compará-lo com o esquesma tradicional.

Cuckoo Hashing

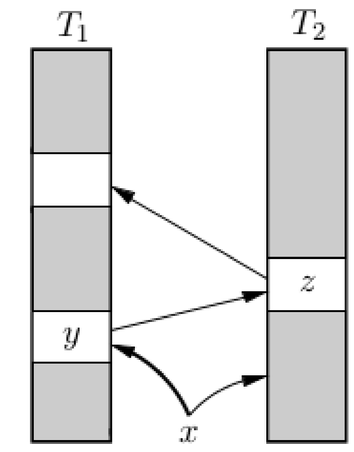
Inspirado no comportamento do pássaro cuckoo na natureza.

Um *esquema hashing* que garante tempo *O(1)* na consulta de uma chave.

É formado por duas tabelas de tamanho *(1 + c)n* cada, sendo *n* o número de chaves e *c* o fator de carga. Cada tabela tem sua função hash.

Ao consultar um elemento *x*, consulta-se a primeira tabela com a primeira função hash, e se não encontrar, consulta-se a segunda com a segunda função hash.

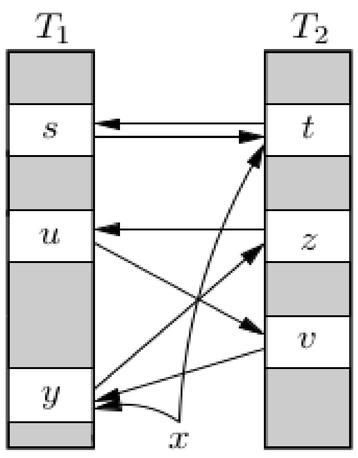
A inserção é simples de entender e implementar, mas complexa de analisar. Possui um comportamento aleatório.



-A chave *x* é mapeada em uma posição da primeira tabela pela função h1.

-Ela é inserida naquela posição, e se houver outra chave ali, digamos y, então a chave y é expulsa (removida) dessa tabela.

-A chave y é inserida na próxima tabela, repetindo o processo de remoção e reinserção na outra tabela enquanto for necessário, ou até que o número de tais operações excede certo limite configurado, situação essa em que uma redistribuição total das chaves será efetuada (*rehash*).



A chave pode não ter sucesso de inserção, quando os elementos ficam se reorganizando sem fim. Para evitar loops, faz-se um *rehash* (uma realocação) nas duas tabelas, utilizando-se duas novas funções hash.

Pseudo-Códigos

function lookup(x)
  return $T_1[h_1(x)] = x \ \vee \ T_2[h_2(x)] = x$
end

procedure insert(x)
  if lookup(x) **then return**
  loop MaxLoop **times**
    $x \leftrightarrow T_1[h_1(x)]$
    if $x = \perp$ **then return**
    $x \leftrightarrow T_2[h_2(x)]$
    if $x = \perp$ **then return**
  end loop
  rehash(); insert(x)
end

O algoritmo pode fazer uma sequência interminável de *rehashes*, tornando a inserção um procedimento irrazoavelmente custoso, em muitos casos.

O Cuckoo Hashing é frequentemente usado com um número constante de tabelas, o qual pode ser maior ou igual a dois. O tempo de consulta continua *O(1)*.

Perfect Hashing

Perfect Hashing garante economia de memória e tempo O(1) de consulta. As chaves devem ser conhecidas previamente. Ex: *tokens* em linguagens de programação.

Uma função hash perfeita é uma injeção, de forma a não ter colisões.

Uma função hash minimal é uma bijeção. O tamanho da tabela é igual ao número de chaves.

Com *n* chaves, o algoritmo para gerar uma função hash perfeita minimal, utilizado neste trabalho, é *O(n)*.

Perfect Rehash

Um *rehash* em um esquema Cuckoo Hashing significa escolher, de forma aleatória, uma nova função hash para cada uma das suas tabelas, de forma a tentar acomodar as chaves que já existiam mais a chave nova.

Como funciona o Perfect Rehash:

1) Escolher de forma aleatória uma tabela que tenha ao menos uma célula disponível para a chave a ser inserida.

2) Utilizar o método de Perfect Hashing para gerar uma função hash perfeita minimal para todas as chaves da tabela, incluindo a chave a ser inserida.

3) Essa nova função passa a ser a função hash da tabela.

O Perfect Rehash garante, para *r* chaves da tabela, uma solução garantida em tempo *O(r)*. O *rehash* tradicional pode nem sequer encontrar um conjunto de funções para acomodar todas as chaves, dependendo da família de funções de onde elas estão sendo escolhidas.

Cuckoo Hashing com Perfect Rehash

Cada tabela do Cuckoo Hashing tradicional possui *(1 + c)n* de células disponíveis para as chaves. Um Perfect Rehash leva *Ω(n)*, neste caso.

A segunda ideia para a construção da variante do Cuckoo Hashing é fazer o número de tabelas ser uma função de *n*. Isso piora o tempo de consulta, mas melhora o tempo de inserção, de forma a se chegar a um equilíbrio.

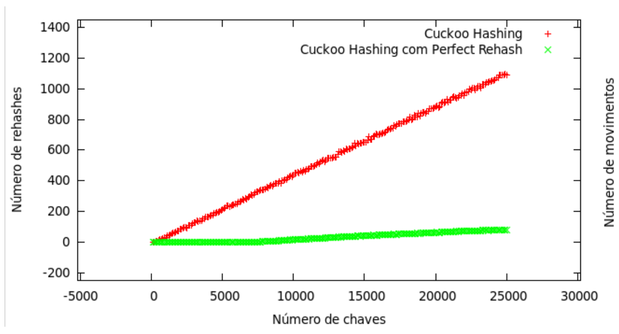
O número de tabelas passa a ser *√n*. Cada tabela possui um tamanho de *c√n, c > 0*.

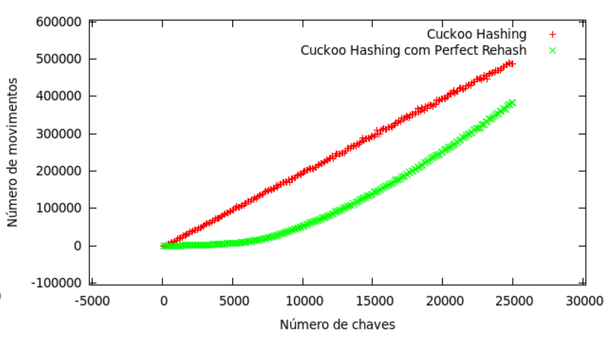
Com essa construção, o tempo de consulta se torna *O(√n)*. O Perfect Rehash tem uma execução de pior caso de *O(√n)*.

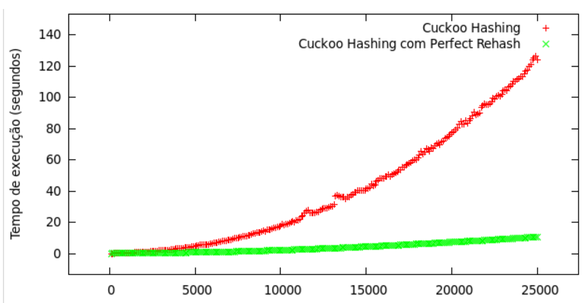
Análise de Operações em Hashing

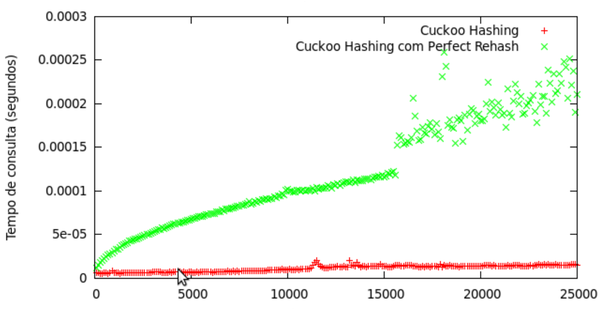
	Hashing com Lista Encadeada para colisão	Cuckoo Hashing	Cuckoo Hashing com Perfect Rehash
Consulta: caso médio	$O(1)$	$O(1)$	$O(\sqrt{n})$
Consulta: pior caso	$O(n)$	$O(1)$	$O(\sqrt{n})$
Inserção: caso médio	$O(1)$	$O(1)$	$O(\sqrt{n})$
Inserção: pior caso	$O(1)$	∞	$O(\sqrt{n})$

Resultados Computacionais









Conclusões e Trabalhos Futuros

- A inserção do novo esquema é mais rápida, na média, do que no Cuckoo Hashing tradicional.
- Pelos gráficos, fica claro que a complexidade de tempo do caso médio do novo esquema aparenta ser *O(1)*.
- O que torna a complexidade de tempo do novo esquema *O(√n)* é a necessidade de consulta no início da inserção.
- *Rehashes* perfeitos aparentam ser promissores para o Cuckoo Hashing tradicional, pois o *rehash* tradicional é *Ω(n)* e o rehash perfeito é *O(n)*.
- Número de movimentos no novo esquema pode ser melhorado modificando a quantidade de tentativa de movimentos que antecedem o *rehash* perfeito.
- As funções hash perfeitas minimais geradas podem ser tão boas quanto as de famílias hash universais?
- A complexidade do tempo de consulta pode ser melhorado.
- Cuckoo Hashing com *rehash* perfeito e número constante de tabelas pode ser promissor.

Equipe

Aluno:
Judismar Arpini Junior (jj.arpini@gmail.com)
Aluno de mestrado do PPGI da UFRJ

Orientador:
Prof° Dr Vinícius Gusmão Pereira de Sá (vigusmao@dcc.ufrj.br)
Professor do Departamento de Ciência da Computação da UFRJ