Funções e tabelas de hash: implementações e aplicações

Breno Helfstein Moura

Universidade de São Paulo

2 de dezembro de 2019

(Supervisor: José Coelho de Pina Júnior)

Three-Sum

O problema

Escreva uma função que dado um vetor de números inteiros V e um número inteiro S, devolva se existe três elementos diferentes em V que somem S.

Podemos resolver esse problema facilmente com uma solução brute-force, tentando todas as triplas. Essa solução seria $O(N^3)$ em tempo e $O(1)$ em memória.

```
1 bool threeSumWithoutHashTable(vector<int>& v, int S) {
2    for (int i = 0; i < v.size(); i++)
3       for (int j = i + 1; j < v.size(); j++)
4          for (int k = j + 1; k < v.size(); k++)
5             if (v[i] + v[j] + v[k] == S) return true;
6    return false;
7 }
```

Contudo podemos substituir o último loop por uma verificação em uma tabela de hash, onde podemos verificar se o existe um terceiro elemento no vetor que complete a soma. Essa solução utilizaria memória linear.

```
1 bool threeSumWithoutHashTable(vector<int>& v, int S) {
2     for (int i = 0; i < v.size(); i++)
3         for (int j = i + 1; j < v.size(); j++)
4             for (int k = j + 1; k < v.size(); k++)
5                 if (v[i] + v[j] + v[k] == S) return true;
6     return false;
7 }
```

Contudo qual a complexidade de de inserção e busca de uma tabela de hash? Isso varia muito de implementação pra implementação, contudo, com a tabela da STL de C++ já conseguimos ver um grande aumento de performance em relação a implementação brute-force. A solução com tabela de hash é na ordem de $O(N^2)$

ArraySize	Time Without Hash Table	Time with Hash Table
128	4.231ms	6.494ms
256	34.223ms	26.665ms
512	267.499ms	99.130ms
1024	1742.688ms	302.453ms
2048	7345.126ms	683.197ms
4096	25029.888ms	761.363ms

Tabela de hash

A ideia principal de uma tabela de hash é associar um objeto a um número inteiro através de uma função de hash. Esse número é então usado para mapear o objeto a um vetor.

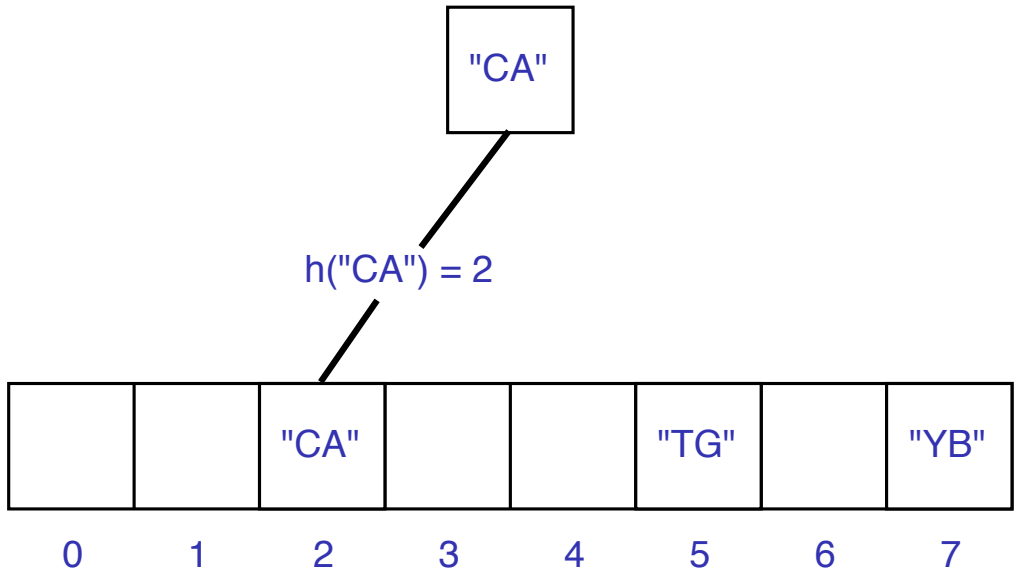

Figura: Exemplo de uma tabela de hash onde h é a função de hash.

A implementação mais simples é chaining hashing. Ela é a implementação de tabelas de hash em linguagens de programação como Java, C++, Scala e Golang.

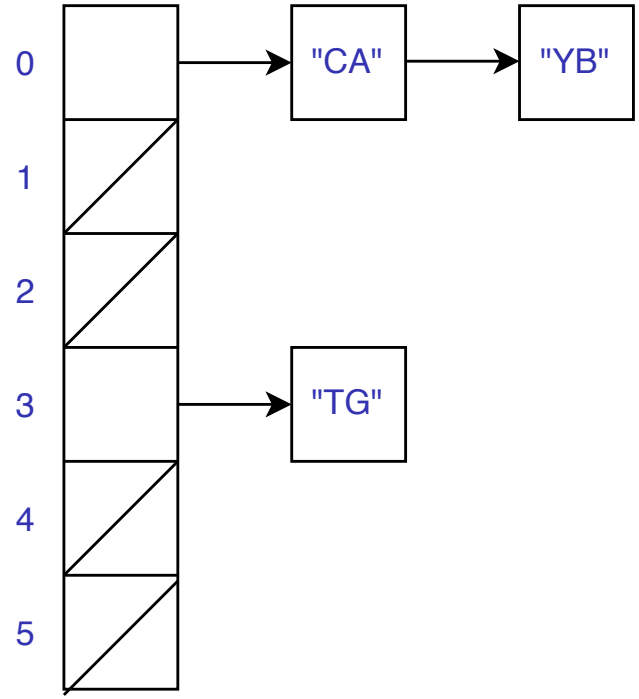

Figura: Ilustração de uma tabela de hashing implementada com chaining hashing

- Chaining hashing delega as operações de busca, inserção e remoção para contêineres contidos em cada posição.
- A complexidade das operações de busca e remoção em chaining hashing é $O(N)$ no pior caso.
- Contudo como vimos no exemplo do three sum, na prática ela é muito mais rápido que no pior caso.
- A complexidade de chaining hashing é $O(1)$ esperado.

Cuckoo Hashing

Contudo nem todas as implementações de tabelas de hash tem custo linear na busca e remoção no pior caso. Esse é o caso da implementação conhecida como cuckoo hashing, em que a busca e a remoção tem custo constante no pior caso.

A implementação de cuckoo hashing se utiliza normalmente de dois vetores e duas funções de hashing. Um fato interessante é que cada objeto inserido vai estar sempre exatamente na sua posição. Contudo, essas garantias são pagas durante a inserção.

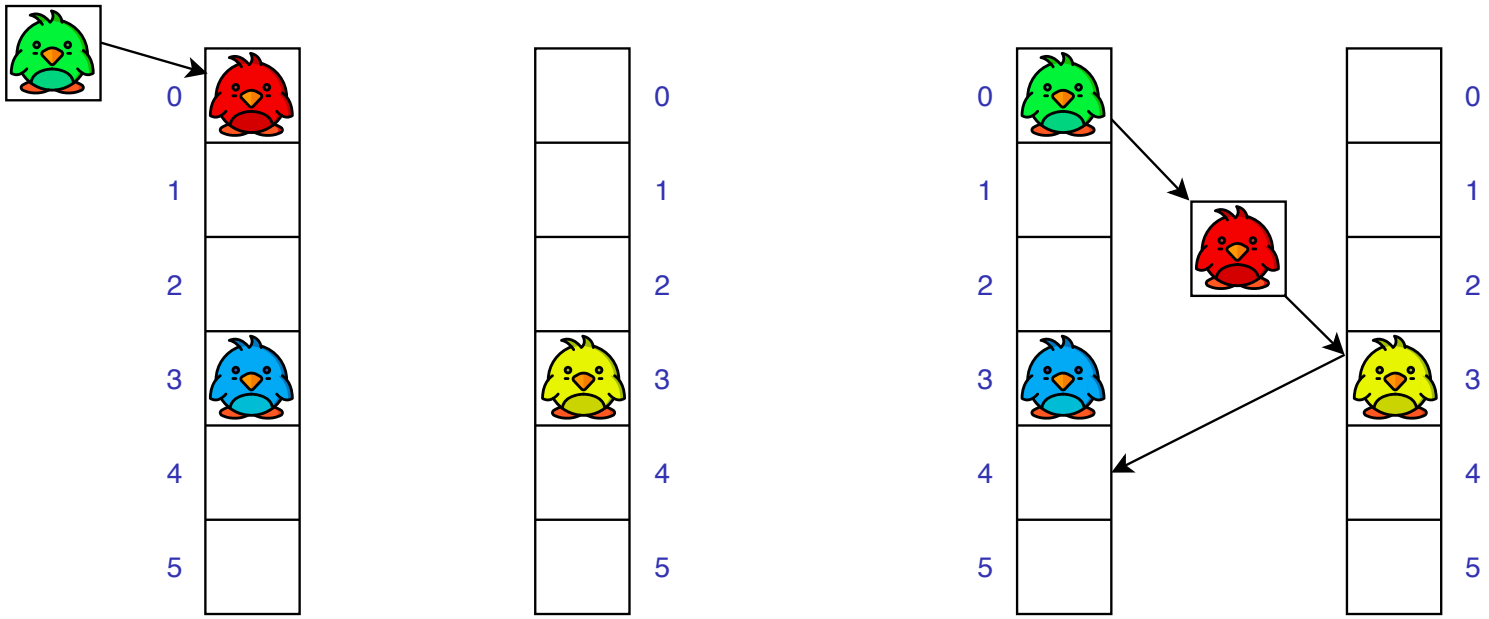

Figura: Ilustração da inserção do cuckoo hashing

Outras aplicações

- Hashing consistente
- Isomorfismo de árvores
- Matching de strings

Perguntas?

Obrigado!