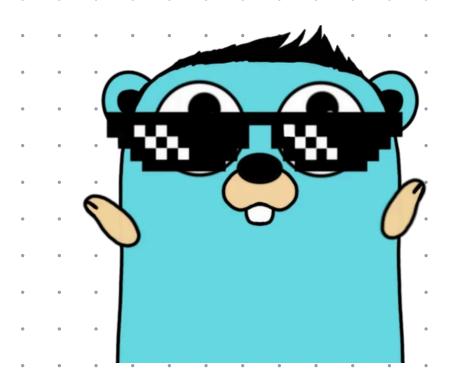


FUNÇÃO HASH

Baseado no livro Entendendo Algoritmos, Um guia ilustrado para programadores e outros curiosos. Aditya Y. Bhargava, Novatec, 2017.





FUNÇÃO HASH

Um armário mágico que transforma qualquer objeto em uma etiqueta única e irreversível. Conhecida como função de dispersão, a função hash funciona como um processador de dados que converte uma grande qualidade de informações em um código de tamanho fixo e possui tempo de execução O(1) para caso médio e O(n) para o pior caso (Big O).

Em outras palavras, ao inserir qualquer tipo de dados, a função retorna um número baseado em regras como retornar um número consistente para uma quantidade de dados e ideialmente mapear dados para diferentes números. Ao final, o array estará cheio e ao buscar um objeto, é necessário buscar somente pelo seu nome e a função hash é encarregado de retornar o índice (gaveta) onde o objeto se encontra. Na prática, funciona da seguinte forma:

- Mapeamento consiste de um nome para seu respectivo índice, sendo a primeira execução para verificar disponibilidade de armazenamento;
- Mapeamento de strings (sequência de bytes) para diferentes índices e;
- A função hash conhece o tamanho do array e retorna somente índices válidos.





FUNÇÃO HASH

Sendo assim, as funções hashs é uma escolha assertiva quando precisar mapear algum item com relação ao outro ou quando pesquisar algo, são constituídos de chave-valor e uma tabela hash mapeia chaves e valores. Ideal sabermos escolher uma boa função hash, caracterizada por menor incidência de colisões e um excelente fator de carga.

Colisões é considerado um comportamento inesperado da função hash, é quando uma função não é capaz de mapear diferentes chaves para diferentes espaços e, eventualmente, pode ocorrer de duas chaves estarem indicadas em um mesmo espaço do array. Uma forma para resolver o problema de colisão, é implementar a lista encadeada no espaço do array onde foi identificada a colisão, podendo comprometer o processamento caso a lista encadeada conter mais elementos do que a própria tabela hash.

Por outro lado, o fator de carga é um cálculo que considera o número de itens de uma tabela hash dividido pelo número total da estrutura.



FUNÇÃO HASH

É recomendável que se o fator de carga estiver acima de 0.7, seja realizado um redimensionamento pela duplicação do tamanho da estrutura para melhor desempenho e menor incidência de colisões.

APLICAÇÕES

Em estrutura de dados, podemos encontrar o algoritmo de função hash nos seguintes casos:

- Armazenamento seguro de senhas: protege contra acessos não autorizados e violações de dados;
- Verificação de integridade de arquivos: compara o hash do arquivo original com o hash do arquivo pós-transferência, identificando e descartando arquivos corrompidos;
- Detecção de duplicatas: compara hashes de arquivos ou registros para identificar duplicatas, otimizando uso de armazenamento e realizando exclusão de informações redindantes;





FUNÇÃO HASH

- Mineiração de criptomoedas: verifica transações e criar novos blocos de blockchain, garantindo a segurança e a imutabilidade da blockchain;
- Download seguro de arquivos: compara o hash do arquivo com o hash fornecido pelo servidor, assegurando autenticidade e integridade;
- Identificação única de objetos: geração de identificadores únicos para objetos digitais (imagem e documentos), permitindo organização, rastreamento eficiente bem como o gerenciamento de grandes volumes de dados;
- Autenticação de mensagens: autenticidade e integridade de mensagens eletrônicas durante a transmissão, protegendo contra falsificação de identidade e adulteração de informações;
- Detecção de malware: identificação de arquivos maliciosos através da comparação de hashes de arquivos com uma lista de hashes, bloqueando a execução de arquivos maliciosos e protegendo sistemas contra ataques cibernéticos.



Q FUNÇÃO HASH

```
package main
import (
    "fmt"
    "crypto/sha256"
type Hash interface {
 GenerateHash(input string) string
type HashSHA256 struct{}
```



Q FUNÇÃO HASH

```
func (HashSHA256 *HashSHA256) GenerateHash(input
string) string {
    hash := sha256.New()
    hash.Write([]byte(input))
    return fmt.Sprintf("%x", hash.Sum(nil))
}
func GenerateHash(input string, hash Hash) string {
    return hash.GenerateHash(input)
```



Q FUNÇÃO HASH

```
func main() {
   inputString := "hello, world!"
   hashSha256 := &HashSHA256{}
   hash := GenerateHash(inputString, hashSha256)
   fmt.Println("hash of", inputString, ":", hash)
}
```



Q SAÍDA DO PROGRAMA

CENÁRIO OTIMISTA

hash of hello, world!: 68e656b25le67e8358bef8483ab0d5lc66l9f3e7ala9f0e75838d4lff368 f728



CONSIDERAÇÕES

Para o exemplo apresentado, seguiu-se as boas práticas de desenvolvimento com clean code e SOLID, na tentativa de simular um cenário otimista.

Essa iniciativa vai de encontro com a ideia de trazer conteúdos relevantes altamente abordados em processos seletivos e desmitificar a ideia de algoritmos e estrutura de dados. Espero que seja de bom proveito e bons estudos.