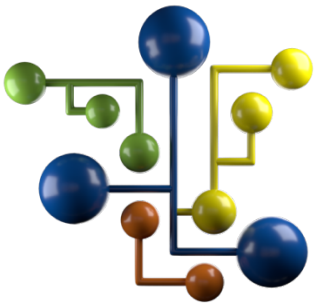

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Vamos para a Aula 8 do curso de Introdução à Lógica de Programação e estudar uma estrutura de dados cujo conceito é a base de diversas ferramentas de Big Data, como Hadoop MapReduce, Apache Spark, HBase, entre outros, e também um conceito fundamental em Blockchain. Tabelas Hash.

A aula de hoje vale ouro! Leia com atenção e saboreie o conhecimento!

Em Ciência da Computação a tabela hash (de hashing, no inglês), também conhecida por tabela de espalhamento, é uma estrutura de dados especial, que associa chaves de pesquisa (hash) a valores. Seu objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado.

Tabelas hash são tipicamente utilizadas para implementar vetores associativos. São tipicamente usadas para indexação de grandes volumes de informação (como bases de dados).

A implementação típica busca uma função hash que seja de complexidade O(1), não importando o número de registros na tabela (desconsiderando colisões). O ganho com relação a outras estruturas associativas (como um vetor simples) passa a ser maior conforme a quantidade de dados aumenta. Outros exemplos de uso das tabelas hash são as tabelas de transposição em jogos de xadrez para computador e até mesmo em serviços de DHCP (através do qual você provavelmente obteve o endereço IP que permitiu você estar conectado à internet neste exato momento).

A função de espalhamento ou função hash é a responsável por gerar um índice a partir de determinada chave. Caso a função seja mal escolhida, toda a tabela terá um mau desempenho. As funções hash são usadas em criptografia e formam a base por trás da tecnologia Blockchain!

O ideal para a função hash é que sejam sempre fornecidos índices únicos para as chaves de entrada. A função perfeita seria a que, para quaisquer entradas A e B, sendo A diferente de B, fornecesse saídas diferentes. Quando as entradas A e B são diferentes e, passando pela função hash, geram a mesma saída, acontece o que chamamos de colisão.

Na prática, funções hash perfeitas, ou quase perfeitas, são encontradas apenas onde a colisão é intolerável (por exemplo, nas funções hash de criptografia), ou quando conhecemos previamente o conteúdo da tabela armazenada). Nas tabelas hash comuns a colisão é apenas indesejável, diminuindo o desempenho do sistema. Muitos programas funcionam sem que seu responsável suspeite que a função de espalhamento seja ruim e esteja atrapalhando o desempenho.

Por causa das colisões, muitas tabelas hash são aliadas com alguma outra estrutura de dados, tal como uma lista encadeada ou até mesmo com árvore. Em outras oportunidades a colisão é solucionada dentro da própria tabela.

Vejamos um exemplo muito elucidativo que foi extraído do livro "Grokking Algorithms: An illustrated guide for programmers and other curious people", do autor Aditya Y. Bhargava, o qual aliás recomendamos a leitura.

Imagine uma atendente que trabalha em um mercado. Quando um cliente compra um produto, é preciso conferir o preço deste produto em um caderno (sim, eu sei que isso é antigo, mas vale o exemplo...rs).

Porém, se o caderno não estiver organizado alfabeticamente, a atendente levará muito tempo analisando cada linha até encontrar o preço da maçã, por exemplo. Procurando desta forma a atendente realizaria uma pesquisa simples, e por meio dela teria de analisar todas as linhas. No entanto, se o caderno estivesse ordenado alfabeticamente, poderia executar uma pesquisa binária para encontrar o preço da maçã (ou seja, começar a pesquisa pelo meio do caderno).

Você já sabe que a pesquisa binária é muito mais rápida. Mas procurar o preço de mercadorias em um caderno é uma tarefa chata, mesmo que este caderno esteja ordenado, pois o cliente fica impaciente enquanto a procura pelo preço dos itens é realizada. Assim, o que a atendente precisa é de um amigo que conheça todas as mercadorias e os seus preços, pois, dessa forma, não é necessário procurar nada: a atendente pede para este amigo e ele informa o preço imediatamente.

Para sorte da atendente, ela tem uma colega de trabalho chamada Maggie, que sabe tudo de cabeça e pode dizer o preço com tempo de execução quase instantâneo para todos os itens, não importando a quantidade de itens que compõem o caderno de preços. Dessa forma, Maggie é ainda mais rápida do que a pesquisa binária.

Que amiga maravilhosa! Mas, então, como você arranja uma “Maggie”? rsrs

Precisamos criar a "Maggie" na forma de uma estrutura de dados. Poderíamos criar um array no seguinte formato:

('maça', 12.80)
('laranja', 14.50)
('uva', 9.20)

Cada item neste array é, na realidade, uma dupla de itens: um é o nome do produto e o outro é o preço. Se ordenar este array por nome, será possível executar uma pesquisa binária para procurar o preço de um item. Logo, é possível pesquisar itens com tempo de execução O(log n). Entretanto nós queremos encontrar itens com tempo de execução O(1), que é bem mais rápido, ou seja, queremos uma “Maggie”, e é aí que entram as funções hash.

Uma função hash é uma função na qual você insere uma string e, depois disso, a função retorna um número.

Em uma terminologia mais técnica, diríamos que uma função hash “mapeia strings e números”. Você pode pensar que não existe um padrão indicando qual número será retornado após a inserção de uma string, mas existem alguns requisitos para uma função hash:

• Ela deve ser consistente. Imagine que você insere a string “maçã” e recebe o número 4. Todas as vezes que você inserir “maçã”, a função deverá retornar o número “4”; caso contrário, sua tabela hash não funcionará corretamente.

• A função deve mapear diferentes palavras para diferentes números. Desta forma, uma função hash não será útil se ela sempre retornar “1”, independentemente da palavra inserida. No melhor caso, cada palavra diferente deveria ser mapeada e ligada a um número diferente. Então, uma função hash mapeia strings e as relaciona a números. Mas qual é a utilidade disso? Bem, você pode usar esta funcionalidade para criar a sua “Maggie”! Seria assim:

1- Você armazenará os preços das mercadorias neste array. Vamos adicionar o preço de uma maçã. Insira “maçã” na função hash.

2- Ela retornará o valor “3”. Agora, vamos armazenar o preço da maçã no índice 3 do array.

3- Vamos adicionar a laranja agora. Insira “laranja” na função hash.

4- A função hash retornou “0”. Agora, armazene o preço da laranja no índice 0.

5- Continue e, eventualmente, todo o array estará repleto de preços de todas as frutas vendidas no mercado!

Agora, você poderá perguntar “Ei, qual é o preço de um abacate?” e não será necessário procurar o preço deste produto no array. Em vez disso, insira “abacate” na função hash.

Ela informará o preço armazenado no índice 4.

A função hash informará a posição exata em que o preço está armazenado. Assim, não precisará procurá-lo! Isto funciona porque:

• A função hash mapeia consistentemente um nome para o mesmo índice. Todas as vezes que você inserir “abacate”, ela retornará o mesmo número. Assim, a primeira execução da função hash servirá para identificar onde é possível armazenar o preço do abacate, e depois disso ela será utilizada para encontrar este valor armazenado.

• A função hash mapeia diferentes strings para diferentes índices. A string “abacate” é mapeada para o índice 4. A string “laranja” é mapeada para o índice 0. Todas as diferentes strings são mapeadas para diferentes lugares do array onde você está armazenando os preços.

• A função hash tem conhecimento sobre o tamanho do seu array e retornará apenas índices válidos. Portanto, caso o seu array tenha cinco itens, a função hash não retornará 100, pois este valor não seria um índice válido do array.

Você acabou de criar uma “Maggie”!

Coloque uma função hash em conjunto com um array e você terá uma estrutura de dados chamada tabela hash. Uma tabela hash é a primeira estrutura de dados que tem uma lógica adicional, visto que arrays e listas mapeiam diretamente para a memória, porém as tabelas hash são mais inteligentes.

Elas usam uma função hash para indicar, de maneira inteligente, onde armazenar os elementos. As tabelas hash são, provavelmente, as mais úteis e complexas estruturas de dados que você aprenderá. Elas também são conhecidas como mapas hash, mapas, dicionários e tabelas de dispersão. Além disso, as tabelas hash são muito rápidas! Você pode pegar um item de um array instantaneamente que as tabelas hash usarão arrays para armazenar os dados desse item; desta forma, elas são igualmente velozes.

Você provavelmente nunca terá de implementar uma tabela hash, pois qualquer linguagem de programação já terá uma implementação dela.

A linguagem Python contém tabelas hash chamadas de..... dicionários. Sim, isso mesmo que você leu! Aqueles dicionários que você usa em quase todas as aulas com Python aqui na DSA, especialmente nas aulas práticas de Spark, Hadoop e Deep Learning.

Para criar uma nova tabela hash você pode utilizar a função dict(). Os dicionários em Python são na verdade tabelas hash. E os detalhes técnicos de sua criação são explicados magistralmente no livro:

Beautiful Code
Leading Programmers Explain How They Think
Autores: Andy Oram, Greg Wilson

Uma tabela hash contém chaves e valores. Na hash caderno, os nomes dos produtos são as chaves e os preços são os valores. Logo, uma tabela hash mapeia chaves e valores. Aqui um exemplo em Python:

```
# Cria um dicionáro vazio
dicionario = dict()

# Adiciona pares de chaves/valores ao dicionário
# As letras são as chaves e os números os valores
dicionario['a'] = 1
dicionario['b'] = 2
```

```
dicionario['c'] = 3

# Podemos imprimir o dicionário
print(dicionario)

# Ou imprimir um valor com base em sua chave
print(dicionario['b'])

# Podemos usar um loop e imprimir as chaves
for chave in dicionario.keys():
--print(dicionario[chave])

# Podemos usar um loop e imprimir os pares chave/valor
for chave, valor in dicionario.items():
--print (chave, ':', valor)
```

O pacote Python chamado hashlib possui diversas funções que aplicam funções hash para Criptografia, como os padrões sha256 e sha512. Eles são estudados na prática no curso Desenvolvimento de Aplicações Descentralizadas na Formação Engenheiro Blockchain!

Aqui um material adicional caso você queira mais detalhes técnicos sobre as tabelas hash:

https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-hash.html

Ufa! Essa aula foi demais! Conhecimento puro e valioso. Da próxima vez que você se deparar com uma solução que armazene os dados no formato de chave/valor e permita pesquisa pela chave, provavelmente você estará diante de uma solução criada com tabela hash, para aumentar a performance.

Vejo você na Aula 9.

#aula08