Trabalho de RA03:
Hash Table

André Thiago, Eduarda Dallagrana, Kauã da Silva Nunes

[1]Pontifícia Universidade Católica do Paraná (PUCPR)

Link GitHub: https://github.com/kauanunnes/ra03

1. Funções Hash

Foram criadas 3 instâncias de funções hash, uma delas usa o chamado dobramento. A nossa chave é igualada ao valor do registro de dentro da classe Registro, contendo agora um número de até 9 dígitos, até pq imaginamos que existam 0 (zeros) que completem as nove casas.

Seja key um número inteiro. Para calcular o hash com base nos dígitos de key, temos:

$$\text{a} = \left\lfloor \frac{\text{key}}{1,000,000} \right\rfloor$$

$$\text{b} = \left\lfloor \frac{(\text{key}/1,000)}{1,000} \right\rfloor \mod 1,000$$

$$\text{c} = \text{key} \mod 1,000$$

Calcule o hash utilizando a fórmula:

$$\text{hash} = (\text{a} \times 31) + (\text{b} \times 17) + (\text{c} \times 13)$$

Ajuste o hash para um valor positivo, garantindo que seja um número inteiro válido no intervalo de 0 a tamanho $- 1$:

$$\text{retorno} = (\text{hash} \& 0x7fffffff) \mod \text{tamanho}$$

As outras funções hash foram um pouco mais simples.

Seja key um número inteiro. Para calcular o hash utilizando o método 2, temos:

Inicialize hash = key.

Calcule o resultado usando a fórmula:

$$\text{result} = \left(\frac{\text{hash} \times 17}{3} \right) \mod \text{tamanho}$$

Retorne o valor de result como um número inteiro.

E a última função Hash criada ocorre desta maneira, ela manipula os bits do numero de forma que todos são 1, ou seja o número fica igual, menos o primeiro que é 0, o que torna o denominador sempre positivo.

Seja key um número inteiro. Para calcular o hash usando o método 3, temos:

Inicialize hash = key.

Calcule o resultado usando a fórmula:

$$\text{resultado} = ((\text{hash} \times 19) \& 0x7fffffff) \mod \text{tamanho}$$

Retorne o valor de resultado como um número inteiro.

2. Performance no Preenchimento

Preenchimento do menor array implementado:

Preenchendo com primeira função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
28571	20000	5628	3127300
28571	100000	72273	11533600
28571	500000	471429	178006900
28571	1000000	971429	973587300
28571	5000000	4971429	23054754800

Preenchendo com segunda função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
28571	20000	5630	2703900
28571	100000	72256	11468100
28571	500000	471429	204486200
28571	1000000	971429	937552200
28571	5000000	4971429	11983962600

Preenchendo com terceira função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
28571	20000	5626	2334000
28571	100000	72300	10111100
28571	500000	471429	205000100
28571	1000000	971429	941169700
28571	5000000	4971429	9667409500

Preenchimento do maior array implementado:

Preenchendo com primeira função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
7142857	20000	4206	752700
7142857	100000	60714	8019800
7142857	500000	448366	116914600
7142857	1000000	945568	780850000
7142857	5000000	4941896	10816170800

Preenchendo com segunda função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
7142857	20000	25	1576800
7142857	100000	652	8036200
7142857	500000	16908	29783700
7142857	1000000	66405	65146200
7142857	5000000	1403581	522729100

Preenchendo com terceira função Hash			
Tamanho do Vetor	Quantidade de dados	Número de colisões	Tempo (ns)
7142857	20000	28	1560900
7142857	100000	673	8626500
7142857	500000	16814	29120500
7142857	1000000	66401	61686800
7142857	5000000	1405392	471783900

Aqui fica claro que a primeira função hash implementada é de longe a menos eficiente para esse caso, mas ainda ocorre um considerável grande número de colisões nas outras duas funções também. Podemos observar que quanto mais espaço tem no array, menos colisões tendem a ocorrer.

3. Performance da Busca

Aqui é possível ver o número de comparações e tempo para a busca de cada elemento quando todos os elementos estão presentes na tabela, tanto na menor quanto na maior tabela implementada, ambas estão descritas com a quantidade respectiva de dados (5 milhões e 500 mil).

Performance da Busca			
ID	Quantidade de dados	Número de colisões	Tempo (ns)
1	5000000	4941896	9073578600
2	5000000	1403581	382483100
3	5000000	1405392	425590100
4	500000	448366	114625200
5	500000	16908	25687900
6	500000	16814	24592100