INSTITUTO FEDERAL
GOIÁS

INSTITUTO FEDERAL DE GOIÁS
CÂMPUS GOIÂNIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
TEORIA GERAL DA ADMINISTRAÇÃO

Nome do Aluno: Luiz Antônio Rodrigues dos Santos
Data: 18/06/2018
Prof. Renan Rodrigues

Trabalho em Sala Tabelas Hash

1) Suponha uma tabela *hash* armazenar valores de chaves. Insira as seguintes chaves nessa tabela: 36, 53, 70, 87, 54, 37, 71 e 40, nessa ordem. Considere os diferentes métodos:

   a) Endereçamento Aberto (tabela *hash* de tamanho 17)

   - Exploração linear: *Passo(k) = k % arraySize* + 1

		53	70	87	54	37	71	40								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

   - Exploração quadrática: *Passo(k) = k % arraySize* + i^2

		53	70	54		87	37			40		71				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

   - Hash duplo: *h1(k) = k % arraySize; h2(k) = 5 - (k % 5)*

		53	54	70	87	37	71		40							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

   b) Encadeamento separado (tabela *hash* de tamanho 5)

nó	nó	nó	nó	nó
0	1	2	3	4
	40	53	54	
		70	37	
		87		
		71		

2) Utilizando os resultados do exercício anterior, responda:

a) Qual o fator de carga final da tabela?
R: 8/17

b) Quais as estratégias geraram menos colisões na inserção dos itens?
R: Encadeamento separado com teve menos colisões, 4 no total, devido a dar somente uma colisão e ser colocada na lista encadeada. Em segundo lugar com 5 colisões o duplo hash. Em terceiro a quadrática com 11 colisões. Com 14 colisões e pior performance a linear.

c) Quais aparentemente realizaram melhor espalhamento das chaves?
R: Aparentemente a Quadrática

3) Se, no tratamento de colisões por encadeamento separado, as listas fossem mantidas de forma ordenada, isso afetaria o tempo de inserção na tabela *hash*? Por quê? E o tempo de pesquisa seria afetado de alguma forma?

R: Sim, ao ordenar as listas demandaria de mais recursos para inserção de novas chaves, pois as listas teriam que ser constantemente ordenas no encadeamente. O desempenho de pesquisas dependeria do modo de pesquisa, no caso da sequencial ou binária, se for sequencial o uso de recursos não seria alterado para pesquisa, já se for binária ela teria um otimização em que seria usado menos recursos para encontrar uma chave.

4) Descreva, comente e compare as vantagens e desvantagens entre as várias técnicas de endereçamento aberto (exploração linear, exploração quadrática e *hash* duplo) e encadeamento separado.

R: Em todos as explorações vantagens como algoritmos simples e eficientes para inserção, remoção e busca, alta eficiência no custo de pesquisa, que é O(1) para o caso médio, e desvantagens como grau de espalhamento depende da função hashing utilizada e ao tipo de informação usada com chave, o custo para recuperar os registros ordenados pela chave é alto, sendo necessário ordenar toda a tabela.
Outras desvantagens na exploração linear é que quando houver remoção de algum item será necessário inserir um número para setar que aquele item no vetor já foi usado e está vazio e a medida que o vetor ficar mais cheio os clusters ficarão maiores, isso significa que acessar células no fim da sequencia será muito lento reduzindo o desempenho.
A exploração quadrática elimina o problema de cluster como na exploração linear, no entanto, tem o problema de cluster secundário, onde segui a mesma sequencia ao tentar encontrar um espaço vago, ou seja, gera sempre os mesmos passos.
O duplo hashing elimina os clusters primário e secundário, as chaves que se convertem para o mesmo índice usarão diferentes sequencias de exploração, dentre as técnicas citadas é a melhor opção.

5) Seja uma empresa que tem o seu número de clientes limitado ao máximo de 1000. Porém o código do cliente (chave) é um número que começa com 4841200001 e vai em sequência até 4841201000. Como seria possível utilizar uma tabela *hash*, em vetores, para implementar tal situação?

R:
1°. Criando um vetor de tamanho 1.579 posições que é primo vetor[1578].
2°. Hash duplo: *h1(chave) = chave % 1.579; h2(chave) = 5 - (chave % 5)*

6) Dependendo da implementação, quando se permitir que itens de dados com chaves duplicadas sejam usadas em tabelas *hash,* apenas o primeiro item de dados pode ser acessado.

a) Descreva com suas palavras com resolver esse problema.
R: Quando houver chaves duplicadas na Tabela Hash, o algoritmo usado na busca de chaves deve ser aprimorado, ao buscar uma chave é efetuado o cálculo para encontrar o índice correspondente a chave, após encontrar o índice é feita uma comparação da chave existente neste mesmo índice do vetor com o valor da chave que deseja encontrar, caso não seja encontrada o cluster é percorrido baseado no passo determinado pela função de hash, que varia de acordo com estratégia de tratamento de colisão escolhida, onde ao encontrar chave a busca é finalizado.
O algoritmo pode ser melhorado na busca é o encerramento da mesma após percorrer todo o cluster ocasionado pelas possíveis colisões de índice calculado para esta chave, onde será implementado um vetor temporário dinamicamente alocado (pois não sabemos a quantidade valores que será armazenado), para que cada vez que a chave do índice for igual a chave buscada, seja armazenado neste vetor o valor do índice, para que retorne a busca com todos os índices correspondentes a chave pesquisada.

b) Escreva um pseudocódigo que retorna todos os itens com a mesma chave.

R:

```
int chave

busca(chave)

     int indice, x, resultado[x]

     indice <- calculaIndice()

     enquanto v[indice].indice != 0 faca

          se chave = v[indice].chave

               resultado[x] <- indice

               x <- x + 1

          Senao

               Indice <- indice + 1

          fimSe

     fimEnquanto

     Escreva (resultado)

fimFuncao
```

7) Fazendo buscas na tabela *hash* abaixo, preencha a tabela que se segue utilizando o duplo *hashing*. Tem-se que:
   - *h1(k) = k % arraySize;*
   - *h2(k) = 5 - (k % 5).*

51		*	85	4	39	*	24	*	19	27	46	38		11	32	*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

N° do Item	Busca	Valor do Hash	Tamanho do Passo	Células na Sequência de Exploração	Status da Busca
1	38	4	2	4, 6, 8, 10, 12	sim
2	78	10	3	10	não
3	24	7		7	sim
4	7	7	3	7, 10, 13	não
5	51	0		0	sim
6	30	13		13	não
7	19	2	1	2, 3, 4, 5, 6, 7, 8, 9	sim
8	8	8	2	8, 10, 12, 14, 16, 1	não
9	32	15		15	sim
10	46	12	4	12, 16, 3, 24, 11	sim
11	69	1		1	não
12	85	0	5	0, 5, 10, 15, 3	sim

Preencha a tabela abaixo (inserções e remoções) utilizando o duplo *hashing*, onde:

h1(k) = k % arraySize;

h2(k) = 5 - (k % 5).

(a)

33	41		25	15	5	27	36	19	11	
0	1	2	3	4	5	6	7	8	9	10

N° do Item	Chave	Valor do Hash	Tamanho do Passo	Células na Sequência de Exploração
1	5	5		5
2	33	0		0
3	19	8		8
4	25	3		3
5	27	5	3	3, 8, 0, 3, 6
6	41	8	4	8, 1
7	15	4		4
8	36	3	4	3, 7
9	11	0	4	0, 4, 8, 1, 5, 9

(b)

43	18	2	5	32	22	*	75	67	*	61	41	12	30		5	27
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

N° do Item	Chave	Valor do Hash	Tamanho do Passo	Células na Sequência de Exploração
1	22	5		5
2	61	10		10
3	18	1		1
4	75	7		7
5	5	5	5	5, 10, 15
6	1	1	4	1, 5, 9
7	32	15	3	15, 1, 4
8	5	5	5	5, 10, 15, 3
9	12	12		12
10	72	4	3	3, 7, 10, 13
11	2	2		2
12	27	10	3	10, 13, 16
13	41	7	4	7, 11
14	67	16	3	16, 2, 5, 8
15	25	8	5	8, 13, 1, 6
16	Del(72)	4	3	4, 7, 10, 13
17	30	13		13
18	43	9	2	9, 11, 13, 15, 0
19	Del(25)	8	5	8, 13, 1, 6
20	Del(1)	1	4	1, 5, 9
21	55	4	5	4, 9, 14