Tabela Hash

A tabela hash é uma estrutura de dados estática. Contudo, os dados não inseridos linearmente como em um array comum, existe uma função de transformação. O que é isso? A função de transformação tem como objetivo deixar os objetos bem distribuídos pelo hash, muitas vezes o objeto é inserido na posição de valor igual ao seu %tamanhoDaTabela. Por exemplo, tenho uma tabela de tamanho 5 e quero inserir o elemento 12. 12 % 5 = 2, o elemento é inserido na posição 2.
OBS: se o dado a ser inserido estiver em forma escrita, deve ser transformado para sua forma numérica.

O que acontece quando dois elementos possuem o mesmo resultado na função de transformação?

Existem 3 maneiras de resolver as colisões no hash

1) HASH COM RESERVA

O hash com reserva é um array que possui algumas posições, pré-determinadas, para o "array principal" e outras, para a reserva. Precisamos de 4 variáveis para ter o controle da inserção:

TamanhoTotal: o tamanho total é a soma do tamanho do array principal com o tamanho da reserva, vamos utilizar para criar o array e lembre-se: é preciso colocar "nulo" em todos as posições (pode ser um número negativo, uma palavra da sua preferência, depende do caso) para verificarmos se a posição está vazia no momento da inserção, ok?

TamanhoArray: o tamanho do array principal controla as inserções no mesmo. O elemento na posição elemento %TamanhoArray.

TamanhoReserva: essa variável é utilizada para verificar se existe espaço na reserva ainda, no momento da inserção.

Reserva: essa é um pouco mais complicada, seguinte a inserção na reserva é linear, não passa pela função de transformação, entao

precisamos acrescentar uma unidade nessa variável sempre que inserimos um elemento na reserva. A posição do elemento será Array[TamanhoArray + reserva].

// Construtor
criar as variáveis e o array do tipo em questão.
receber e setar os valores do array principal e da reserva.
setar a váriavel reserva (não o tamanho da reserva) como zero.
colocar "nulo" em todas as posições.

// Inserir
O método "inserir" é booleano, por que? Em determinadas situações não vamos conseguir inserir o elemento e porque a reserva estará e cheia. Por isso retornamos "true" para quando inserimos e "false" quando não.

```
boolean inserir(int elemento)
{
	boolean inseriu = true;

	if (elemento != NULO)
	{
		int i = funçãoTransformação(int elemento);

		if (Array[i] == NULO)
		{
			Array[i] = elemento;

		}else if (reserva < TamanhoReserva)
		{
			Array[TamnhoArray + Reserva];
			reserva++;
		}
		else
		{
			inseriu = false;
		}
	}

	return inseriu;
}
```

A função transformação é o que? Uma função de apenas umaq linha que irá retornar a posição do elemento em questão.

```
public int funçãoTransformação(int elemento)
{
```

```
        return elemento % TamanhoArray;
}

public boolean remover (int elemento)
{
        boolean presente = false;
        int i = funçãoTansformação(elemento);

        if (Array[i] == elemento)
        {
                presente = true;
                tam = TamanhoArray;
                boolean substituto = false;
                int pos = 0;

                // vou procurar um substituto levando em consideração o %tam

                for (int j = 0; j < reserva && !substituto;  j++)
                {
                        if ( funçãoTransformação(Array[ j+ tam]) == i)
                        {
                                Array[i] = Array[j + tam];
                                substituto = true;
                                pos = j + tam;
                        }
                }
                // vou subir os elementos da  reserva se eu tiver achado um substituto
                if (substituto)
                {
                        for (int j = 0; j < reserva - 1;  j++)
                        {
                                Array[pos + j ] = Array[pos + j + 1 ];
                        }

                        reserva--;
                        Array[tam + reserva] = NULO;

                }else {Array[i] = NULO};
        }

        else if (reserva > 0 )
        {
                int pos = 0;


                // percorrer a reserva em busca do elemento
                for (  ; pos < reserva && !presente;  pos++)
                {
                        if( Array[pos +TamanhoArray] == elemento) presente = true;
                }
```

```
                // subir os elementos
                if (presente)
                {
                        for (int j = 0; j < reserva - 1, j++)
                        {
                                A[TamanhoArray + pos] = A[TamanhoArray + pos + 1];
                        }
                        reserva--;
                        Array[TamanhoArray + reserva] = NULO;
                }
        }

        return presente;
}
```

O pesquisar é bem basiquinho

```
public boolean pesquisar (int elemento)
{
        boolean presente = false;
        i = FunçãoTransformação(elemento);

        if (Array[i] == elemento) resp = true;
        else
        {
                for (int j = 0; j < reserva; j++)
                {
                        if (Array[ TamnhoArray + j ] == elemento)
                        {
                                presente = true;
                                j = reserva;
                        }
                }
        }

        return presente;
}
```

complexidade
-inserção no array principal: O(1), na reserva: O(n).
-busca no array principal: O(1), na reserva varia. Melhor caso: O(1), pior O(n)

2) Hash com rehash

O hash com rehash não tem reserva. Então como ele resolve as colisões? Seguinte: a posição do elemento vai ser elemento %

TamArray, certo? Se a posição do elemento em questão estiver preenchida, podemos tentar inseri-lo na posição seguinte. Se a posição seguinte também estiver preenchida, não inserimos o elemento.

A função "reh" retorna a posição que o elemento deve ser inserido se a primeira estiver ocupada.

```
public int reh (int elemento)
{
	return ++elemento % TamanhoArray;
}

public int FunçãoTransformação(int elemento)
{
	return elemento % TamanhoArray;
}

public boolean inserir (int elemento)
{
	boolean inseriu = false;
	int i = FunçãoTransformação (elemento);

	if (elemento != NULO)
	{
		if (Array[i] == NULO)
		{
			Array[i] = elemento;
			inseriu = true;

		}else
		{
			int pos = reh (elemento);
			if (A[pos] == NULO)
			{
				inseriu = true;
				Array[pos] = NULO;
			}
		}
	}

	return inseriu;
}

public boolean pesquisar (int elemento)
{
	boolean presente = false;
	int i = FunçãoTransformação (elemento);

	if (A[i] == elemento) { presente = true; }
	else
```

```
	{
		int pos = reh (elemento);
		if (Array[pos] == elemento) presente = true;
	}
	return presente;
}
```

O método remover também vai ser boolean porque o elemento pode não ter sido inserido.

```
// não ta certo
public boolean remover (int elemento)
{
	boolean presente = false;

	if (pesquisar(elemento) == true)
	{
		presente = true;
		int i = FunçãoTransformação (elemento);

		if (Array[i] == elemento)
		{
			int pos = reh(elemento);

			if (Array[pos] != NULO)
			{
				Array[i] = Array[pos];

				if (Array[pos+1] != NULO)
		}
	}
}
```

3) Hash com lista encadeada

Nesse hash a nossa tabela vai ser de células. Por que? De cada posição vai sair uma lista encadeada que será a nossa solução para as colisões. Além disso, precisaremos de um array de ponteiros do tipo célula, que vão ser nossos ponteiros "último".

```
HashLista
{
	CelulaSimples[] Tabela;
	CelulaSimples[] Ultimo;
	int tamanho;
	int NULO = -1;

	public HashLista ()
	{
		this (10);
```

```
}

public HashLista(int tamanho)
{
	this.tamanho = tamanho;
	Tabela = new CelulaSimples[tamanho];
	Ultimo = new CelulaSimples[tamanho];

	for (int i = 0; i < tamanho; i++)
	{
		Tabela[i] .elemento = NULO;
		Ultimo[i] = Tabela[i];
	}
}


public void inserir (int elemento)
{
	int i = FunçãoTransformação(elemento);

	if (Tabela[i].elemento == NULO){Tabela[i].elemento = elemento;}
	else
	{
		Ultimo[i].prox = new CelulaSimples();
		Ultimo[i] = Ultimo[i].prox;
		Ultimo[i].elemento = elemento;
	}
}
```

Esse remover não é booleano pois todos os elementos foram inseridos.

```
public void remover (int elemento)
{
	int i = FunçãoTransformação(elemento);

	if (pesquisar(elemento))
	{
		int pos = 0;

		for (CelulaSimples j = Tabela[i]; j.prox != null; j = j.prox, pos++)
		{
			if (j.elemento == elemento) j = ultimo;
		}

		Tabela[i].remover(pos); // considera q a tabela é do tipo lista
	} else
	{
		new Exception (“Erro ao inserir, elemento inexistente);
	}

}
```