```java
public class HashTable<T> {
 int length;
 Object[] table;
 int cont;
 //double fullFactor;

 HashTable(){
  // si el numero de la tabla es primo es mejor
  length = 101;
  table = new Object[length];
  cont = 0;
 }
 HashTable(int length){
  this.length = length;
  table = new Object[length];
  cont = 0;
 }
 public int size(){
  return cont;
 }
 // checar si es mejor dejar la funcion aqui adentro o mejor como metodo externo
 /*private int fnHash(T elem){
  return 0;
 }*/
 public void add(T elem, double clave){
  cont++;
  /*if(cont/length > fullFactor)
   increase();*/
  //int pos = fnHash(elem) % length;
  int pos = (int)(clave % length);
  HashNode<T> temp = (HashNode<T>) table[pos];
  HashNode<T> novo = new HashNode<T>(elem);
  novo.setNext(temp);
  table[pos] = novo;
}
/*private void increase(){
  HashNode<T>[] aux = (HashNode<T>[]) new Object[2*length];
  HashNode<T> actual, auxNode, next;
  int newPos;

  for(int i = 0; i<length; i++){
    actual = (HashNode<T>)table[i];
    while(actual != null){
      newPos = fnHash(actual.getElem()) % aux.length;
      auxNode = aux[newPos];
      next = actual.getNext();
      actual.setNext(auxNode);
      aux[newPos] = actual;
      actual = next;
    }
  }
  table = aux;
  length*=2;
}*/
public boolean find(T elem, double clave){
  HashNode<T> actual;
  //int pos = fnHash(elem) % length;
  int pos = (int)(clave % length);
  actual =(HashNode<T>) table[pos];
  while (actual != null && !actual.elem.equals(elem))
    actual = actual.getNext();
  return actual != null;
}
private HashNode<T> findNode(T elem, double clave){
  HashNode<T> actual, previous;
  //int pos = fnHash(elem) % length;
  int pos = (int)(clave % length);
  actual = (HashNode<T>) table[pos];
  previous = (HashNode<T>) table[pos];
  while (actual != null && !actual.elem.equals(elem)){
    previous = actual;
    actual = actual.getNext();
  }
  if (actual == null)
    return null;
  else
    return previous;
}
public void delete(T elem, double clave){
  HashNode<T> preFind = findNode(elem, clave);
  if (preFind != null){
    HashNode<T> toDelete = preFind.getNext();
    preFind.setNext(toDelete.getNext());
    toDelete.setNext(null);
    cont--;
  }
}
```

```
  public double promColisiones(){
    double prom = 0;
    if(this.size()==0)
      return 0;
    else{
      for(int i = 0; i<table.length; i++){
        prom+=conteoColisiones(i);
      }
      return prom/table.length;
    }
  }
  public int conteoColisiones(int pos){
    HashNode<T> actual =(HashNode<T>) table[pos];
    int cont = -1;
    while(actual != null){
      cont++;
      actual = actual.getNext();
    }
    return cont;
  }
  public double getFullFactor(){
    return this.size()/table.length;
  }

}
```