Resolução de colisões:
Encadeamento interno (*Open Addressing*)

Inserção: examinar (probe) sucessivamente a tabela de dispersão até encontrar uma posição vazia. A sequência de posições examinadas dependente da chave a ser inserida. A função de dispersão inclui o número de vezes que a tabela foi examinada para aquela chave

```
HASH-INSERT(T, k)
i ← 0
repeat j ← h(k, i)
        if T[j] = NIL
          then T[j] ← k
               return j
          else i ← i + 1
  until i = m
error "hash table overflow"
```


Resolução de colisões:
Encadeamento interno (*Open Addressing*)

Procura: examinar a mesma sequência de posições que o algoritmo de inserção examinou para uma dada chave. A procura termina ou quando: ou encontra a chave; ou examinou M posições; ou encontra um espaço livre.

```
HASH-SEARCH(T, k)
i ← 0
repeat j ← h(k, i)
        if T[j] = k
          then return j
        i ← i + 1
  until T[j] = NIL or i = m
return NIL
```

Resolução de colisões:
Encadeamento interno (*Open Addressing*)

Eliminação.

- Não se pode colocar simplesmente a null na posição da chave a ser eliminada.
  - algoritmo de procura inconsistente.
- Uma solução será colocar a posição da chave a ser eliminada com um valor especial ao invés de ser eliminada.
  - implica modificação do algoritmo inserção anterior.

Algoritmos de resolução de colisões
Encadeamento interno (*Open Addressing*)

hash functions – $h(k,i) : U \rightarrow \{0,1,\ldots,m-1\}$

- Procura Linear (*Linear Probing*);

$$h(k,i) = (h'(k) + i) \bmod m,\ 0 \le i < m$$

- Procura Quadrática (*Quadratic Probing*);

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m,\ \ c_1, c_2 \neq 0_2 \text{ e } 0 \le i < m$$

- Dupla Dispersão (*Double Hashing*).

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m\ ,\ 0 \le i < m$$

h1 e ***h2***, caso se use um ***m*** primo, poderão ser:

$$h_1(k) = k \bmod m,$$

$$h_2(k) = 1 + (k \bmod m')$$

Resolução de Colisões:
Encadeamento Interno-Procura Linear

- N < M : método com índices livres.
  - Dado que há sempre posições livres na tabela, procurar outra posição.
- Procura linear:
  - se a posição correspondente ao índice devolvido pela função de dispersão estiver ocupada, ir incrementando o índice até se encontrar uma posição livre.

0	70
1	14
2	72
3	77
4	
5	19
6	75

Ordem de inserção: 70, 72, 14, 77, 19, 75

hash(k) = k mod 7

Cátia Vaz

Resolução de Colisões
Procura Linear (*Linear Probing)*

$$h(k,i) = (h'(k) + i) \bmod m,\ 0 \le i < m$$

- Desempenho:
  - os elementos tendem a ficar agrupados (*clusters*);
  - os agrupamentos grandes tendem a crescer ainda mais;
  - o tempo médio de procura tende a crescer para M à medida que a tabela enche;
  - operações na tabela de dispersão tornam-se demasiado lentas quando a tabela atinge 70% - 80% da sua capacidade.
  - existem m sequências de examinação (probe) diferentes

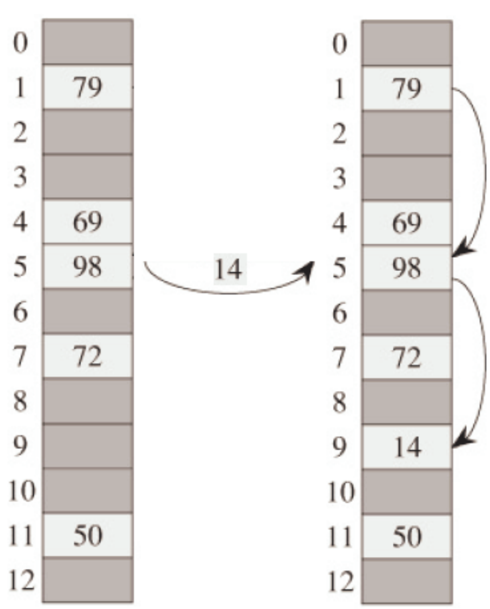
Resolução de Colisões
Procura Quadrática (*Quadratic Probing)*

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m, \quad c_1, c_2 \neq 0_2 \text{ e } 0 \leq i < m$$

- Desempenho:
  - tem melhores resultados que a procura linear;
  - *A* função de *hash* inicial define a sequência;
  - elementos que têm a mesma função de *hash* têm a mesma sequência logo também tendem a ficar agrupados (*secondary clusters*);
  - Em caso de colisão, existem m sequências de examinação (probe) diferentes

Resolução de Colisões
Dupla Dispersão (*Double Hashing*)

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m \;, \; 0 \leq i < m$$



$$h_1(k) = k \bmod 13$$
$$h_2(k) = 1 + (k \bmod 11)$$

Resolução de Colisões
Dupla Dispersão (*Double Hashing*)

$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m \ , \ 0 \leq i < m$$

- Desempenho:
  - é um dos melhores métodos de resolução;
  - a sequência também depende da chave;
  - elementos que têm a mesma função de *hash* têm sequências distintas
  - Em caso de colisão, existem m^2 sequências de exame (probe) diferentes
    - cada par possível (h1(k),h2(k)) assegura uma sequência de examinação diferente.
- Tem que se garantir que a tabela é pesquisada na totalidade:
  - Usando um ***m*** que seja potencia de dois e desenhando ***h2(k)*** para produzir sempre um número impar.
  - Usando um ***m*** primo e desenhando ***h2(k)*** para produzir sempre um número menor que ***m.***