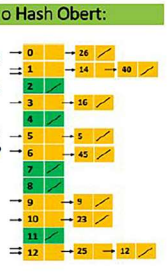
Col·lisions solució1: Encadenament separat o Hash Obert:

- La taula hash es converteix en un vector de llistes
- A cada posició del vector guardem, a una llista, tots els elements que s'han indexat a la mateixa posició.
- **Factor de càrrega λ = n / m**. És el nombre mig de claus per llista. S'ha d'intentar que sigui **menor a 1, 0.75** estaria be. (n és el nombre d'elements desats a la taula i **m** el nombre de posicions.]. Llistes d'1 element.
- **Problema:** si les llistes són molt llargues les cerques no són eficients.
- **Solució:** Incrementar mida de la taula i fer **rehashing**

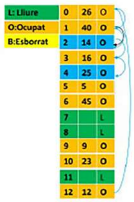
Rehashing: Si fem la taula més gran M varia i també varia h(x). Per tant hem de recalcular les posicions de totes les dades desades. Aquest procés s'anomena rehashing i és costós.

Exemple: 23,45,16,26,40,14,5,12,9,25.



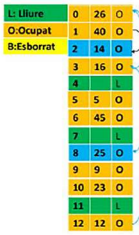
Exemple:
Amb h(x) = x%13 i M=13

Amb sondeig lineal: $h_i(x)=(h(x)+i)\%M$
Inserim:
23%13=10
45%13=6
16%13=3
26%13=0
40%13=1
14%13=1 !! COL·LISIÓ → Reassignació: (1+1)%13=2
5%13=5
12%13=12
9%13=9
25%13=12 !! COL·LISIÓ
→ Reassignació: (12+1)%13=0; (12+2)%13=1; (12+3)%13=2; (12+4)%13=3;(12+5)%13=4;



Exemple:
Amb h(x) = x%13 i M=13

Amb sondeig quadràtic: $h_i(x)=(h(x)+ i^2)\%M$
Inserim:
23%13=10
45%13=6
16%13=3
26%13=0
40%13=1
14%13=1 !! COL·LISIÓ → Reassignació: (1+ 1²)%13=2
5%13=5
12%13=12
9%13=9
25%13=12 !! COL·LISIÓ
→ Reassignació: (12+1)%13=0; (12+4)%13=3; (12+9)%13=8



solució: Implementar Inserció Hashing tancat. Insert

```
void Hash::insert(const string& clau, const string& descripcio)
{//Cerquem posicio amb collisions                                  Cerca element
  int index = codi(clau);
  int vegades = 1;                  Les posicions poden ser OCUPAT o ESBORRAT
  bool trobat = false;
  while ((m_estat[index] != LLIURE) && (!trobat))
  {
     if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
        trobat = true;
     else
     {
        index = codi2(index,vegades);
        vegades++;
     }                                                  Valida si trobat i modifica
  }
  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat es una modificacio de la seva descripcio
     m_diccionari[index].second = descripcio;
  }
  else{   //Inserim un nou element
     insertIntern(clau, descripcio);                    Insereix nou element
  }
}
```

solució: Implementar Inserció Hashing tancat. Cerca

```
int Hash::cerca(const string& clau) const
//Cerquem posicio amb collisions                                   Cerca element
  int index = codi(clau);
  int vegades = 1;                  Les posicions poden ser OCUPAT o ESBORRAT
  while ((m_estat[index] != LLIURE) && (!trobat))
  {
     if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
        trobat = true;
     else
     {
        index = codi2(index,vegades);
        vegades++;
     }
  }
  return index;
```

solució: Implementar Inserció Hashing tancat.Estructura

```
class Hash
(public:
    Hash(int numIni=10, const string& descripcioDefecte="Definicio no existent") ;
    ~Hash();
    string& operator[](const string& clau);
    void insert(const string& clau, const string& descripció);
    bool find(const string& clau, string& definicio) const;
    bool esborra(const string& clau);
    friend ostream& operator<<(ostream &out, const Hash& h);          Afegim cerca
  private:
    int codi(const string& clau) const;
    int codi2(int index, int vegades) const;
    void insertIntern(const string& clau, const string& descripcio);
    int cerca(const string& clau) const;
    std::vector<std::pair<string, string>> m_diccionari;
    std::vector<int> m_estat;
    int m_numOcupats;
    int m_maxElements;
    string m_descripcioDefecte;
    static const int LLIURE = 0;
    static const int OCUPAT = 1;
    static const int ESBORRAT = 2;
```

```
#include <iostream>
#include <vector>
#include <utility>
#include <string>
using namespace std;
```

```
string& Hash::operator[](const string& clau)                          Cerca element
{ int index = cerca(clau);

  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat
    return m_diccionari[index].second;                                Valida si trobat i retorna
  }
  else
  {//Element no trobat, l'inserim
    insertIntern(clau, m_descripcioDefecte);
    return m_descripcioDefecte;                                       No trobat i l'inserim
  }
```

solució: Implementar Inserció Hashing tancat. Insert

```
void Hash::insert(const string& clau, const string& descripcio)
{//Cerquem posicio amb collisions                                   Cerca element
  int index = cerca(clau);

  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat es una modificacio de la seva descripcio
     m_diccionari[index].second = descripcio;                         Valida si trobat i modifica
  }
  else
  {   //Inserim un nou element
     insertIntern(clau, descripcio);
  }                                                                   Insereix nou element
}
```

solució: Implementar Inserció Hashing tancat. Find

```
bool Hash::find(const string& clau, string& definicio) const
{                                                                     Cerca element
  int index = cerca(clau);

  bool trobat = false;
  if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
  {
     definicio = m_diccionari[index].second;
     trobat = true;                                                   Valida si trobat i retorna
  }
  else{     definicio = "";}
  return trobat;                                                      No trobat
}
```

solució: Implementar esborra Hashing tancat.

```
bool Hash::esborra(const string& clau)
{                                                                     Cerca element
  int index = cerca(clau);

  bool trobat = false;
  if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
  {//Trobat l'esborrem
     //m_diccionari[index].second="";
     //m_diccionari[index].first= "";
     m_estat[index] = ESBORRAT;
     m_numOcupats--;                                                  Valida si trobat i esborra
     trobat = true;                                                   Retorna true
  }

  return trobat;                                                      Si no el troba retorna false
}
```

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ofstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);


if (fitxer.is_open())
{
    int numero;
    while (<hi_ha_valors_per_escriure>)
    {
        <obtenir_numero>


        fitxer << numero;
    }

    fitxer.close();
}
```

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ifstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);

if (fitxer.is_open())
{
    int numero;
    fitxer >> numero;


    while (!fitxer.eof())
    {

        <tractar_numero>
        fitxer >> numero;
    }
    fitxer.close();
}
```

Mètodes de la classe `forward_list`

- **insert_after:** afegeix un element a la posició següent a la que indica un *iterador*. Retorna un *iterador* a l'element afegit.
- **erase_after:** elimina l'element següent a la posició que indica un iterador. Retorna un *iterador* a l'element següent a l'element eliminat.
- **push_front**: afegeix un element al principi de la llista.
- **pop_front**: elimina el primer element de la llista.
- **empty:** retorna si la llista està buida.
- **begin**: retorna un *iterador* al primer element de la llista.
- **before_begin**: retorna un *iterador* a la posició anterior al primer element.
- **end:** retorna un *iterador* a la posició següent al final de la llista.

Classe `list` (llista doblement encadenada)

```
#include <list>

std::list<tipus> llista;
```

Mètodes:

- **insert:** afegeix un element a la posició **anterior** a la que indica un *iterador*. Retorna un *iterador* a l'element afegit.
- **erase:** elimina l'element de la posició que indica un iterador. Retorna un *iterador* a l'element següent a l'element eliminat.
- **push_front/push_back**: afegeix un element al principi/final de la llista.
- **pop_front/pop_back**: elimina el primer/últim element de la llista.
- **empty:** retorna si la llista està buida.
- **begin/rbegin**: retorna un *iterador* al primer/últim element de la llista.
- **end/rend:** retorna un *iterador* a la posició següent al final de la llista (end) o a la posició anterior al principi de la llista (rend).

Mètodes de la classe stack:

- **void push (const <tipus_pila>& valor)**
  - Afegeix un element a dalt de tot de la pila
- **void pop()**
  - Elimina l'element de dalt de tot de la pila
  - La pila no pot estar buida
- **<tipus_pila>& top()**
  - Retorna l'element de dalt de tot de la pila
  - La pila no pot estar buida
- **bool empty()**
  - Retorna cert si la pila es buida, fals si no ho és

Operacions habituals de les piles:

- **push**: afegeix un element a dalt de tot de la pila.
- **pop:** elimina l'element de dalt de tot de la pila.
- **top:** retorna l'element de dalt de tot de la pila.
- **esBuida:** retorna cert si la pila es buida, fals si no ho és.

Mètodes de la classe **queue**:

- **void push (const <tipus_cua>& valor)**
  - Afegeix un element al final de tot de la cua com a últim element
- **void pop()**
  - Elimina el primer element de la cua
  - Suposa que la cua no està buida
- **<tipus_cua>& front()**
  - Retorna el primer element de la cua (sense eliminar-lo)
  - Suposa que la cua no està buida
- **<tipus_cua>& back()**
  - Retorna l'últim element de la cua (sense eliminar-lo)
  - Suposa que la cua no està buida
- **bool empty()**
  - Retorna cert si la cua es buida

ordered_map

```
map<std::string, std::string> diccionari =
una salutacio." },{"Adeu","Es un comiat."},
n periode de 24 hores."),
n periode de entre 28 i 31 dies."),
n periode de 12 mesos.")
};
string v[5] = { "Any","Hola","Mes","Dia","Adeu" };

cout << "=====CERQUES AL DICCIONARI=====" << endl;
for (int i=0;i<5;i++)
{  cout << "Definicio de " << v[i] << " es: "<< diccionari[v[i]] << endl;}

cout << "=====CONTINGUT DEL DICCIONARI=====" << endl;
for (auto& x : diccionari)
{  cout << x.first << ": " << x.second << std::endl;}
cout << "Definicio de caleidoscopi es: " << diccionari["caleidoscopi"] <<endl;

cout << "=====CONTINGUT DEL DICCIONARI=====" << endl;
for (auto& x : diccionari)
{ cout << x.first << ": " << x.second << std::endl;}
return 0;
}
```

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
```

```
std::unordered_map<string, string>::const_iterator itParaula=diccionari.find("caleidoscopi");
if (itParaula == diccionari.end())
    cout << "Definicio de caleidoscopi no hi es" <<endl;
Else    cout << "Definicio de caleidoscopi es: " <<itParaula->second <<endl;
```