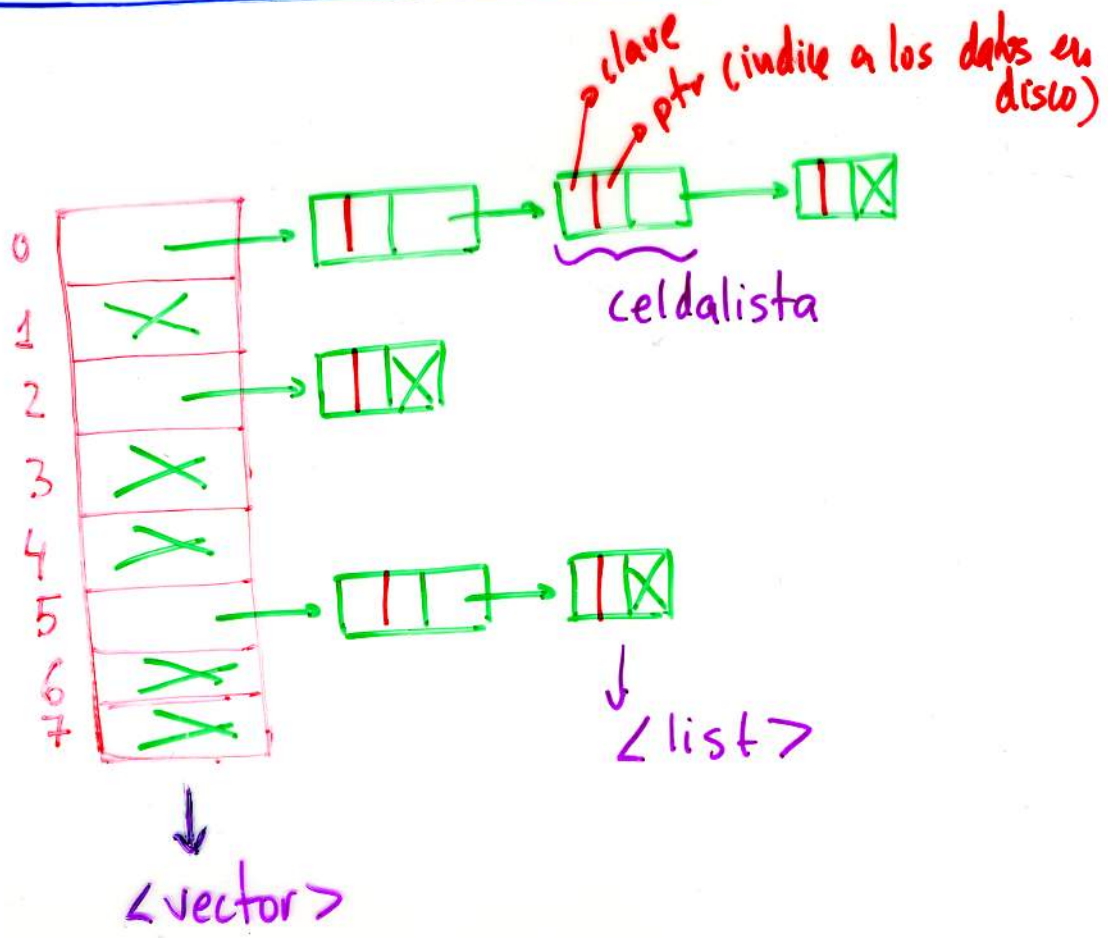
CLASE TABLA HASH ABIERTA



Definición usando la STL

```
vector < list<celdalista> > tablahash;
```

hash_abierto.h

```
#include <list>
#include <vector>
using namespace std;

class CeldaLista {
    private:
        int clave;
        int ptr; //indice a la posición de los datos
    public:
        CeldaLista() : clave(-1), ptr(-1) {};
        CeldaLista(int c, int p): clave(c), ptr(p) {};
       ~CeldaLista() {};
        ..........
        int & Clave() {return clave;};
        int & Datos() {return ptr;};
};
```

/** Cada celda de la lista enlazada contiene:

- una clave
- un indice a los datos

```
class TablaHash {

  private:
    vector < list <CeldaLista> > tabla;
      // La Tabla Hash es un vector de listas enlazadas
    int fhash (int clave); // función hash
    list <CeldaLista>:: iterator EstaEnFila (int clave, int pos);
  // Devuelve la posición (mediante un iterador) donde
  // se encuentra la clave en la fila pos de la tabla hash
  // Devuelve ∅, si no está

  public:
    TablaHash (int tam); //constructor de tabla con un
                         // tamaño fijo
    ~TablaHash() {};
    bool Existe (int clave); //comprueba si la clave está en
                             //la tabla
    bool Insertar (int clave, int ptrdatos);
    //Inserta un par (clave, ptrdatos). Devuelve true si
    // se ha insertado y false en caso contrario.
    bool Borrar (int clave); //borra la clave
    bool Cambiar (int clave, int ptrdatos);
      //cambia el valor ptrdatos asociado a la clave
    int Obtener (int clave);
      //Devuelve el ptrdatos asociado a la clave y -1 si la clave
                                                  //no está
    void imprimir(); // imprime en pantalla la tabla Hash
};
```

hash_abierto.cpp

```
#include <iostream>
#include <cassert>
#include <hash_abierto.h>
using namespace std;

TablaHash::TablaHash(int tam)
{
    assert(tam > 0);
    tabla.resize(tam); // Fijamos un tamaño para la Tabla
}

int TablaHash::fhash(int clave)
{
    return clave % tabla.size(); // f(x) = x % tamaño
}

list<CeldaLista>::iterator TablaHash::EstaEnFile(int clave,
                                                 int pos)
{
}
for (list<Celdalista>::iterator encontrado = tabla[pos].begin();
     encontrado != tabla[pos].end(); encontrado++)

    if (*encontrado).clave() == clave)
        return encontrado;

return 0;
}
                    //(*encontrado).clave() es equivalente
                    // a encontrado->clave()
```

```
bool TablaHash::Existe(int clave)
{
  int pos = fhash(clave);
  return (EstaEnFila(clave, pos) != 0);
}

bool TablaHash::Insertar(int clave, int ptrdatos)
{
  int pos = fhash(clave);
  if (EstaEnFila(clave, pos) == 0) {
    tabla[pos].push_back(CeldaLista(clave, ptrdatos));
    return true;
  } else return false; // No se ha insertado (clave repetida)
}

bool TablaHash::Cambiar(int clave, int ptrdatos)
{
  int pos = fhash(clave);
  list<CeldaLista>::iterator dondeesta = EstaEnFila(clave, pos);

  if (dondeesta != 0)
    {
      (*dondeesta).Datos() = ptrdatos;
      return true;          // dondeesta -> Datos()
    }
  else return false; // la clave no está en la tabla
}
```

```
int TablaHash::Obtener(int clave)
{
    int pos = fhash(clave);
    list<CeldaLista>::iterator fil = EstaEnFila(clave, pos);
    return (fil != 0) ? fil->Datos() : -1;
}

bool TablaHash::Borrar(int clave)
{
    int pos = fhash(clave);
    list<CeldaLista>::iterator dondeesta = EstaEnFila(
                                            clave, pos);
    if (dondeesta != 0)
    {
        tabla[pos].erase(dondeesta);
        return true;
    }
    else return false; // La clave no está en la tabla

void TablaHash::Imprimir()
{
    for (unsigned int i = 0; i < tabla.size(); i++)
    {
        cout << "Fila" << i << ":";
        for (list<CeldaLista>::iterator p = tabla[i].begin();
             p != tabla[i].end(); p++)
            cout << "(" << (*p).clave() << ","
                 << (*p).Datos() << ")";
        cout << endl;
    }
}
                    // p->clave(), p->Datos()
```

Estructuras duales

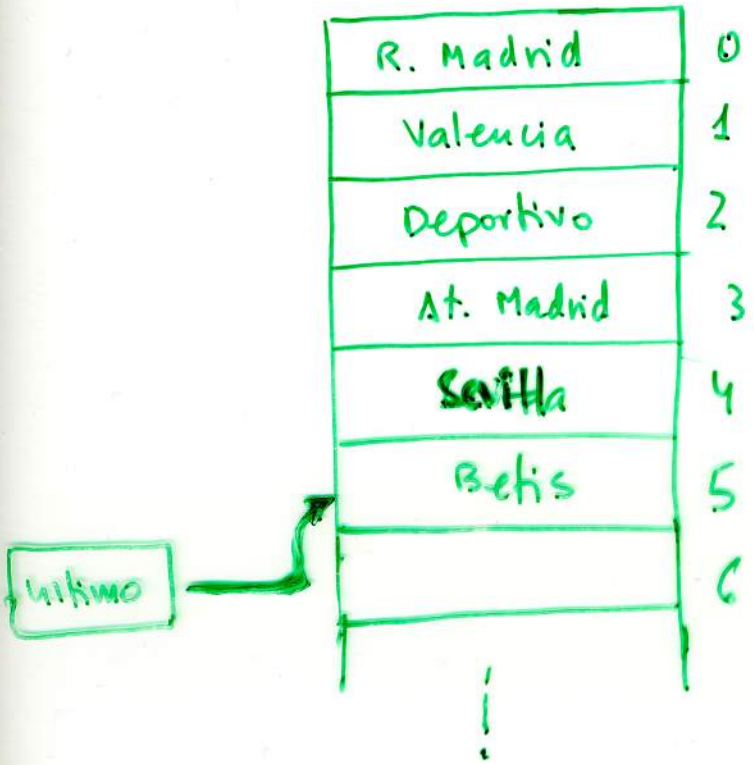
Problema: Mantener una escala de equipos de fútbol en la que cada equipo está en una posición. Los nuevos equipos se agregan al final. Cada equipo puede retar al que está inmediatamente encima y caso de ganarle se intercambia con el.

Operaciones

- **Agregar (equipo)**: Añade el equipo "equipo" en el último lugar
- **Retar (equipo)**: Devuelve el nombre del equipo en la posición $i-1$ si "equipo" está en la posición i, $i > 1$
- **Cambiar (i)**: Intercambia los equipos que están en las posiciones i e $i-1$ $i > 1$

Solución 1

Vector de equipos v con v[i] el nombre del equipo i-ésimo, manteniendo un cursor al último equipo insertado



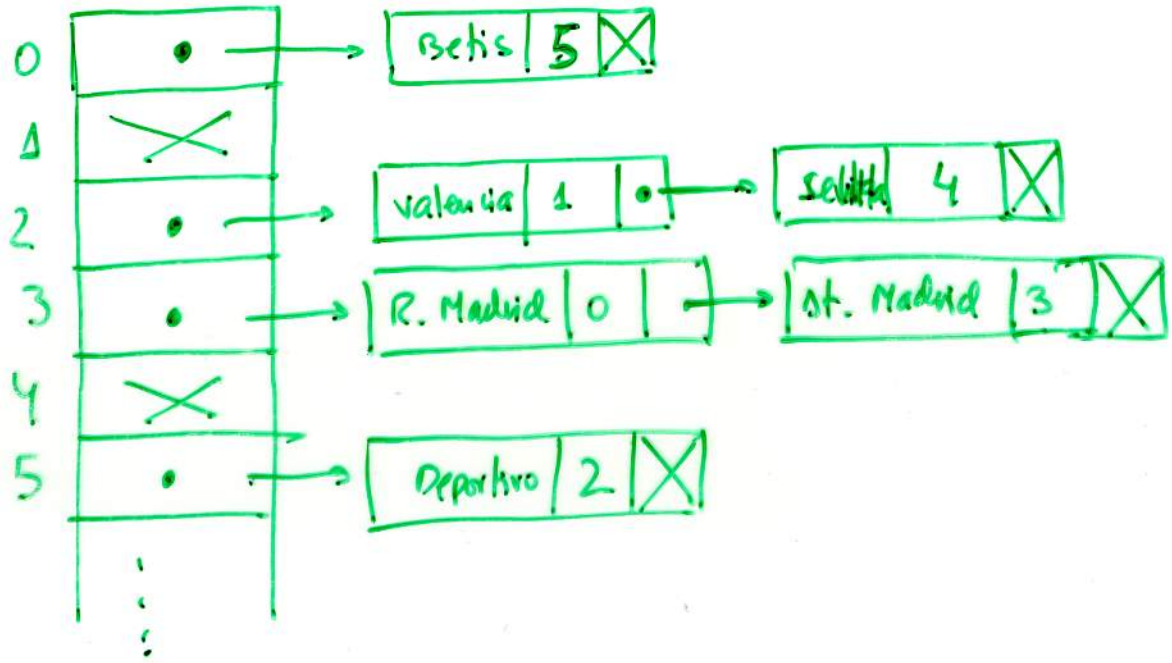
Agregar (equipo) ⟶ O(1)

Retabiar (i) ⟶ O(1)

Retar (equipo) ⟶ O(n)

Solución 2

Tabla hash abierta construida en base al nombre de los equipos. Cada elemento de las listas enlazadas contiene el nombre del equipo y su posición en la clasificación.

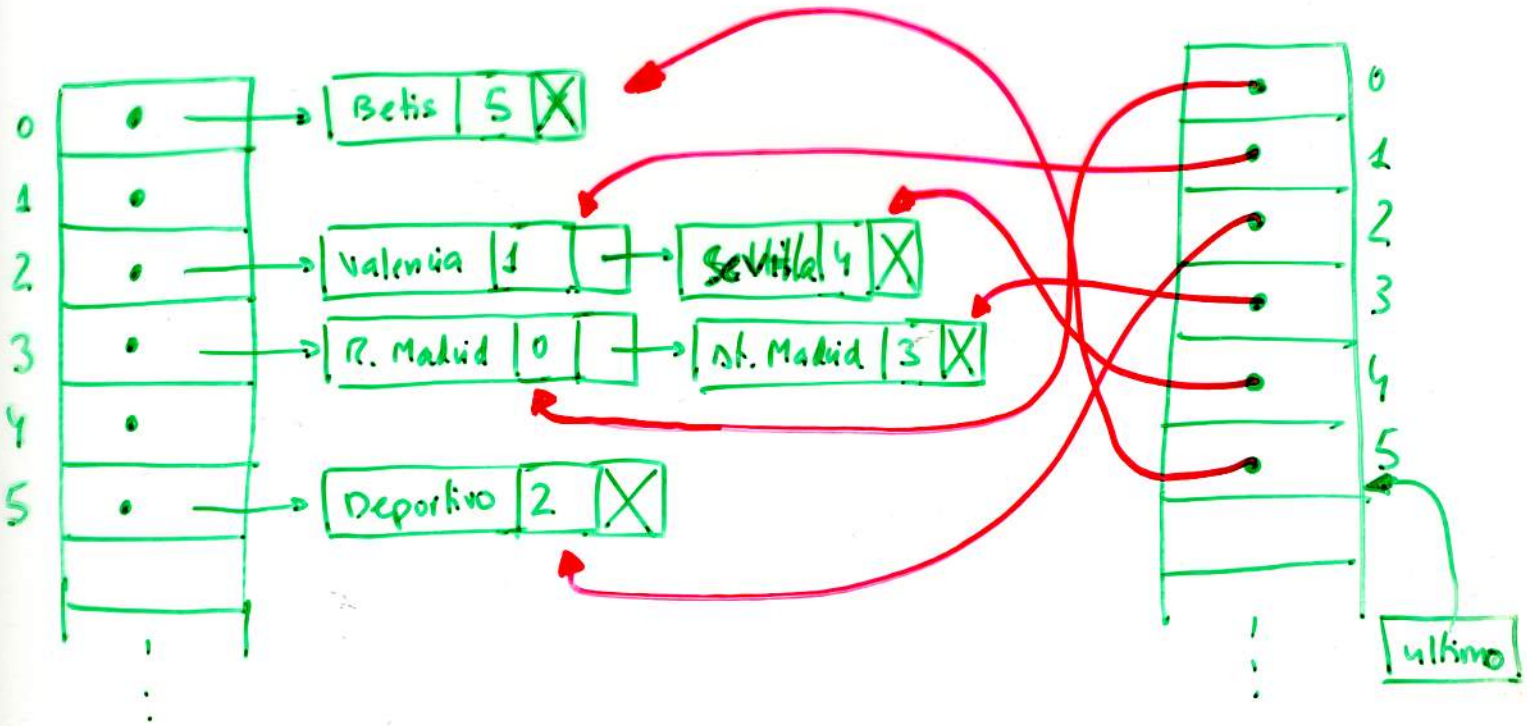


Agregar (equipo) ⟶ O(1)

Cambiar (i) ⟶ O(n)

Retar (equipo) ⟶ O(n)

Solución híbrida



Agregar (equipo) ⟶ O(1)

Cambiar (i) ⟶ O(1)

Retar (equipo) ⟶ O(1)