*Pregunta 1: VFVVVFVFFV

*Pregunta 2:

Dispersión abierta:

P0 -> 5
P1
P2 -> 7 -> 2
P3
P4 -> 9 -> 14

Dispersión cerrada:

h(x) = (x + i^2) % 5

P0: 14 (al reintento 1)
P1: 5 (al reintento 1)
P2: 7
P3: 2 (al reintento 1)
P4: 9

*Pregunta 3:

Apartado 1:

```
vector<T> ABB<T>::inordenNR() {
    vector<T> resultado;
    stack<Nodo<T> *> nodosPorProcesar;
    Nodo<T> *nodo = raiz;

    while (nodo != 0 || !nodos.empty()) {
        while (nodo != 0) {
            nodosPorProcesar.push(nodo);
            nodo = nodo->izq;
        }

        nodo = nodosPorProcesar.top();
        nodosPorProcesar.pop();
        resultado.push_back(nodo->dato);

        nodo = nodo->der;
    }

    return resultado;
}
```

Apartado 2:

```
void ListaEnlazada<T>::inserta(Iterador<T> &it1, ListaEnlazada<T> &l2) {
    // Precondición: suponemos que it1.nodo no apunta a 0
    Nodo<T> *anterior = it1.nodo, *resto = it1.nodo->sig;
    Iterador<T> it2 = l2.inicio();

    while (!it2.fin()) {
        anterior->sig = new Nodo<T>(it2.dato(), resto);
        anterior = anterior->sig;
```

```
        l2.siguiente();
    }

    // Caso especial si it1 apuntaba al final de la lista *this
    if (resto == 0) {
        cola = anterior;
    }
}
```

*Pregunta 4:
Hay un error en el diagrama UML y BloqueAsignado aparece con el parámetro T de plantilla cuando no es una plantilla.
En cualquier caso era irrelevante para la resolución del ejercicio.

```
class BloqueAsignado {
    long comienzo;
    long tam;

public:
    BloqueAsignado (long comienzo, long tam) {
        this->comienzo = comienzo;
        this->tam = tam;
    }

    long verComienzo() { return comienzo; }
    long verTam() {return tam; }
    long verFinal () { return comienzo + tam; }
};

template<typename T>
class GestorMemoria {
    // Vale una lista enlazada ordenada por comienzo, pero esta es la mejor elección
    map<long, BloqueAsignado> bloquesAsignados;
    T *bufferMemoria;
    long maxMem;

public:
    GestorMemoria(long maxMem) : bloquesAsignados() {
        this->maxMem = maxMem;
        bufferMemoria = new T[maxMem];
    }

    T *asignar(long tam) {
        // Variables para anotar la posición y tamaño del mejor hueco encontrado
        long comienzoMejorHueco = -1;
        long tamMejorHueco = +INFINITY;

        // Posición final del bloque anterior asignado
        long finalAnterior = 0;

        // Recorrer los bloques asignados y comprobar los huecos existentes entre ellos
        // Buscando el mejor (el de menor tamaño donde quepa el bloque solicitado)
        map<long, BloqueAsignado>::iterator i = bloquesAsignados.begin();
        while (i != bloquesAsignados.end()) {
            long tamHueco = i->verComienzo() - finalAnterior;
            if (tamHueco > tam && tamHueco < tamMejorHueco) {
                comienzoMejorHueco = finalAnterior;
```

```
                tamMejorHueco = tamHueco;
            }
            finalAnterior = i->verFinal();
            ++i;
        }
        // Comprobar también el hueco restante hasta el final del búffer
        // También sirve para el caso en que no haya ningún bloque asignado (todo libre)
        long tamHueco = maxMem - finalAnterior;
        if (tamHueco > tam && tamHueco < tamMejorHueco) {
            comienzoMejorHueco = finalAnterior;
            tamMejorHueco = tamHueco;
        }

        // Si no se ha encontrado ningún hueco, lanzar excepción
        if (comienzoMejorHueco = -1) {
            throw bad_alloc;
        }

        // Crear bloque e insertar en el mapa
        BloqueAsignado bloque(comienzoMejorHueco, tam);
        bloquesAsignados[comienzoMejorHueco] = bloque;

        // Devolver puntero a la posición correspondiente en el búffer
        return bufferMemoria + comienzoMejorHueco;
    }

    void eliminar(T *puntero) {
        // Sólo hay que borrar el bloque correspondiente del mapa
        // Truco: calcular la posición de comienzo del bloque
        // restando al puntero el puntero al comienzo del búffer
        // (los punteros son direcciones de memoria y se puede operar con ellos)
        //
        // Si no caemos en esto, hacer búsqueda secuencial (más lento)
        long comienzoBloque = (long)(puntero - bufferMemoria);
        bloquesAsignados.erase(comienzoBloque);
    }

    ~GestorMemoria() {
        // El mapa y su contenido se destruyen automáticamente
        delete[] bufferMemoria;
    }
}
```