Iteradores y Algoritmos Genéricos en C++

Algoritmos y Estructuras de Datos II

Colecciones

Conocemos los siguientes tipos de *colecciones*:

- Arreglo.
- Secuencia.
- Conjunto.
- Multiconjunto.
- Diccionario.

Operaciones sobre colecciones

Preguntas típicas sobre colecciones:

- Dado un elemento, ¿está en la colección?

  $x \in \texttt{conj}$

  $\text{def?}(x, \texttt{dicc})$

- Listar todos los elementos de una colección.
- Encontrar el elemento más chico de la colección.
- *etc.*

Operaciones sobre colecciones

¿Cómo recorremos una colección?

- **Arreglo.** Tamaño y acceder al i-ésimo (`operator[]`).
- **Secuencia.** prim y fin.
- **Conjunto.** dameUno y sinUno.
- **Multiconjunto.** dameUno y sinUno.
- **Diccionario.** claves y obtener.

¿Hay una manera uniforme de recorrerlas?

Operaciones sobre colecciones

¿Cómo hacemos para mirar los primeros 5 elementos de una lista simplemente enlazada en C++?

- Supongamos que la operación fin es destructiva:
  ```
  void Lista<T>::sacarPrimero() { ... }
  ```
- ¿Qué pasa con la complejidad?

Operaciones sobre colecciones

¿Cómo hacemos para mirar los primeros 5 elementos de una lista simplemente enlazada en C++?

- Supongamos que la operación fin es destructiva:
  `void Lista<T>::sacarPrimero() { ... }`
- ¿Qué pasa con la complejidad?

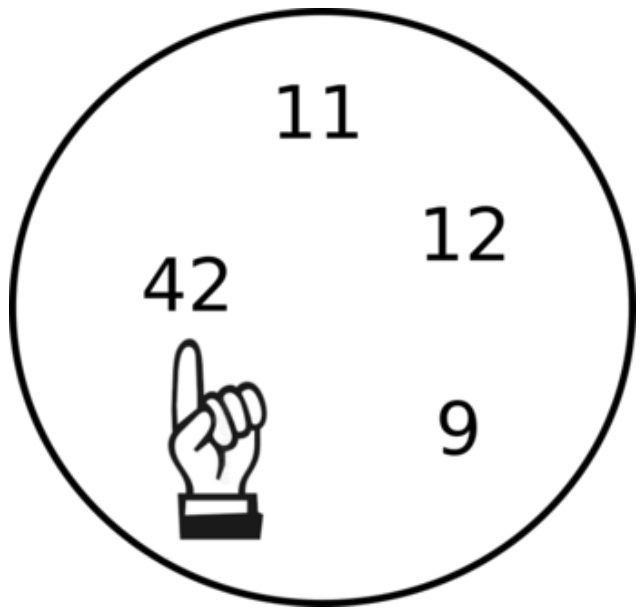
```
list<int> l = {1, 5, 2, 3, 4, 1, 2};
list<int> l2 = l;                          // O(n)
for (int i = 0; i < 5; i++) {
  cout << l.prim() << endl;                // O(1)
  l.sacarPrimero();                        // O(1)
}
```

Iteradores

Un **iterador** es una manera abstracta de recorrer colecciones, independientemente de su estructura.

Informalmente

$$\text{iterador} = \text{colección} + \text{dedo}$$



Iteradores

Operaciones con iteradores:

- ¿Está posicionado sobre un elemento?
- Obtener el elemento actual.
- Avanzar al siguiente elemento.
- Retroceder al elemento anterior. *(Bidireccional)*
- Modificar el valor del elemento actual. *(Mutable)*

Iteradores en C++

Tipos

Si `T` es un tipo de colección:

- `T::iterator`, `T::const_iterator`
  Tipo de los iteradores mutables e inmutables.
  Por ejemplo `vector<int>::iterator` es un tipo.
- `T::value_type`
  Tipo de los elementos que almacena la colección.
  Por ejemplo `vector<int>::value_type` es `int`.

Creación de iteradores

Si `col` es una colección de tipo `T`:

- `col.begin()`
  Iterador posicionado sobre el primer elemento de la colección.
- `col.end()`
  Iterador posicionado sobre el final de la colección.
  (Después del último elemento).

Iteradores en C++

Operaciones con iteradores

Si `it` es de tipo `T::iterator` o `T::const_iterator`:

- `*it` — Obtiene el elemento actual.
  Si `it` es un `T::iterator`, es un lvalue y modificable.
  Si `it` es un `T::const_iterator`, no es un lvalue y es no modificable.
- `it->campo` — Equivalente a `(*it).campo`.
- `++it` — Avanza al siguiente elemento.
- `--it` — Retrocede al elemento anterior.
- ...

Iteradores en C++

Operaciones de la colección usando iteradores

- `T::iterator T::insert(T::iterator pos, const T::value_type& elem)`

  Inserta un elemento en la posición indicada.
- `T::iterator T::erase(T::iterator pos)`

  Elimina el elemento en la posición indicada.
- ...

Iteradores en C++

Ejemplo (recorrer)

```
vector<int> v = {1, 2, 3, 4};

vector<int>::iterator it = v.begin();
while (it != v.end()) {
  cout << *it;
  ++it;
}

for(vector<int>::iterator it = v.begin(); it != v.end(); ++it){
  cout << *it;
}
```

Iteradores en C++

Ejemplo (eliminar)

```
vector<int> v = {1, 2, 3, 4};
vector<int>::iterator it = v.begin();
it += 2;
it = v.erase(it);                    // 1 2 4
cout << *it;                         // 4
```

Iteradores en C++

Ejemplo (insertar)

```
vector<int> v = {1, 2, 3, 4};
vector<int>::iterator it = v.end();
--it;
it = v.insert(it, 10);                    // 1 2 3 10 4
cout << *it;                              // 10
```

Iteradores en C++

Muchas veces el compilador puede inferir los tipos:

Ejemplo (auto)

```
vector<int> v = {1, 2, 3, 4};
auto it = v.end();
--it;
v.insert(it, 10);
```

(No abusar de esta funcionalidad).

Iteradores en C++

Ejemplo (iteradores mutables vs. inmutables)

```
void mostrar(const vector<int>& v) {
  for (vector<int>::iterator it = v.begin();
       it != v.end(); ++it) {
    cout << *it;
  }
}

int main() {
  vector<int> v{1, 2, 3, 4};
  mostrar(v);
  return 0;
}
```

Iteradores en C++

Ejemplo (iteradores mutables vs. inmutables)

```
void mostrar(const vector<int>& v) {
  for (vector<int>::iterator it = v.begin();
       it != v.end(); ++it) {
    cout << *it;
  }
}

int main() {
  vector<int> v{1, 2, 3, 4};
  mostrar(v);
  return 0;
}
```

```
In function void mostrar(const std::vector<int>&):
conversion from std::vector<int>::const_iterator [...]
to non-scalar type std::vector<int>::iterator [...]
```

Iteradores en C++

Ejemplo (iteradores mutables vs. inmutables)

```
void mostrar(const vector<int>& v) {
  for (vector<int>::const_iterator it = v.begin();
       it != v.end(); ++it) {
    cout << *it;
  }
}

int main() {
  vector<int> v{1, 2, 3, 4};
  mostrar(v);
  return 0;
}
```

Iteradores en C++

Ejemplo (for basado en rangos)

```
void mostrar(const vector<int>& v) {
  for (int x : v) {
    cout << x;
  }
}

int main() {
  vector<int> v{1, 2, 3, 4};
  mostrar(v);
  return 0;
}
```

Iteradores en C++

Ejemplo (for basado en rangos)

```
void mostrar(const vector<int>& v) {
  for (int x : v) {
    cout << x;
  }
}

int main() {
  vector<int> v{1, 2, 3, 4};
  mostrar(v);
  return 0;
}
```

- En general se acepta la sintaxis `for (T x : col)` siempre que `col` sea una colección con la interfaz de iteradores descripta arriba.

Algoritmos genéricos

Recibiendo una colección genérica:

```
template<class Coleccion>
bool pertenece(const Coleccion& c,
               typename const Coleccion::value_type& x) {
  for (auto& y : c) {
    if (x == y) {
      return true;
    }
  }
  return false;
}

int main() {
  vector<int> v{1, 2, 3, 4};
  int dos = 2;
  cout << pertenece(v, dos);
}
```

(Ojo con el typename).

Algoritmos genéricos

Recibiendo un iterador genérico:

```
template<class Iterador>
bool pertenece(Iterador desde, Iterador hasta,
               typename Iterador::value_type& x) {
  for (auto it = desde; it != hasta; ++it) {
    if (x == *it) {
      return true;
    }
  }
  return false;
}

int main() {
  vector<int> v{1, 2, 3, 4};
  int dos = 2;
  cout << pertenece(v.begin(), v.end(), dos);
}
```