Diseño y Análisis de Algoritmos

MSc. Enrique Paiva - 2023

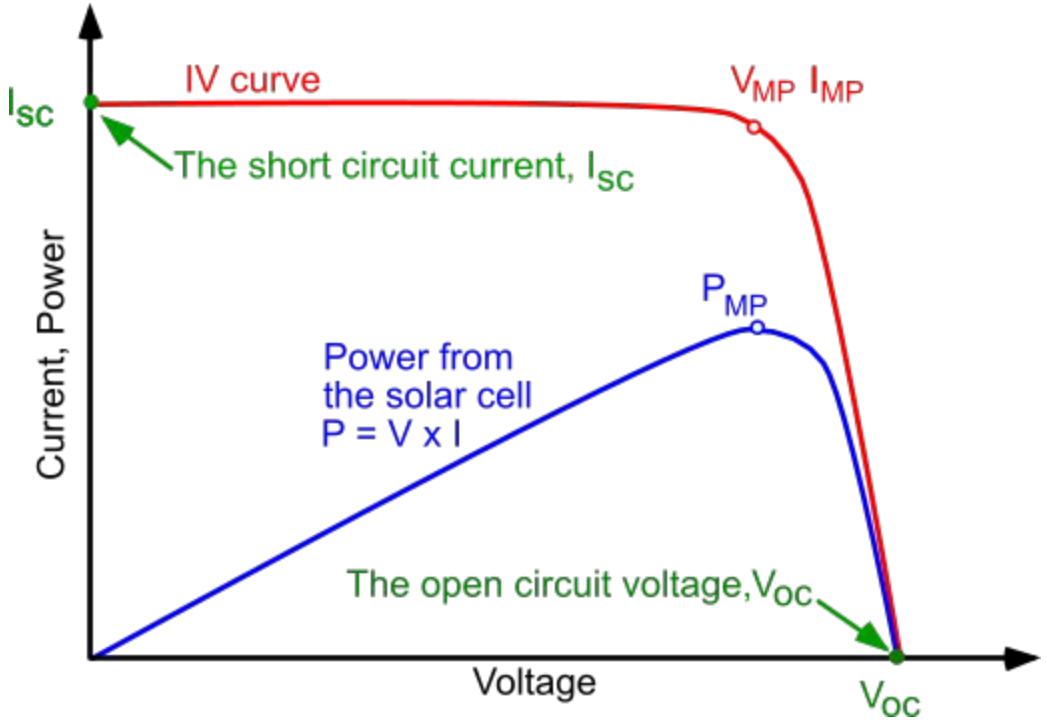
Ejercicio de Clases

Implementar una clase Panel Solar

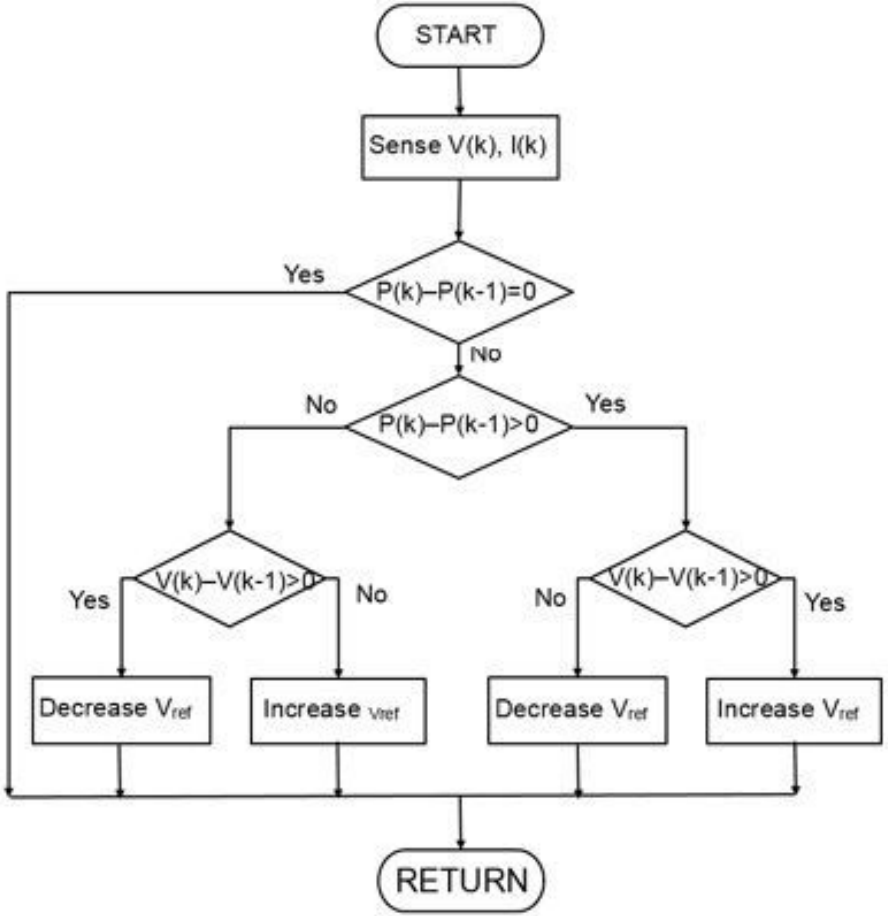
$$I = I_L - I_0 \left[\exp\left(\frac{qV}{nkT} \right) \right]$$

Donde:

- I_L (Corriente generada por la luz) = 0.5 A
- I_0 (Corriente de saturación oscura) = 10^(-7) A
- q (Carga del electrón) = 1.6*10^(-19) C
- n = 1
- k (constante de Boltzmann)= 1.38*10^(-23) J/K
- T = 300K



https://www.pveducation.org/pvcdrom/solar-cell-operation/iv-curve


START
Sense V(k), I(k)
P(k)–P(k-1)=0
Yes
No
P(k)–P(k-1)>0
No
Yes
V(k)–V(k-1)>0
Yes
No
V(k)–V(k-1)>0
No
Yes
Decrease Vref
Increase Vref
Decrease Vref
Increase Vref
RETURN

Estructura de Datos

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)

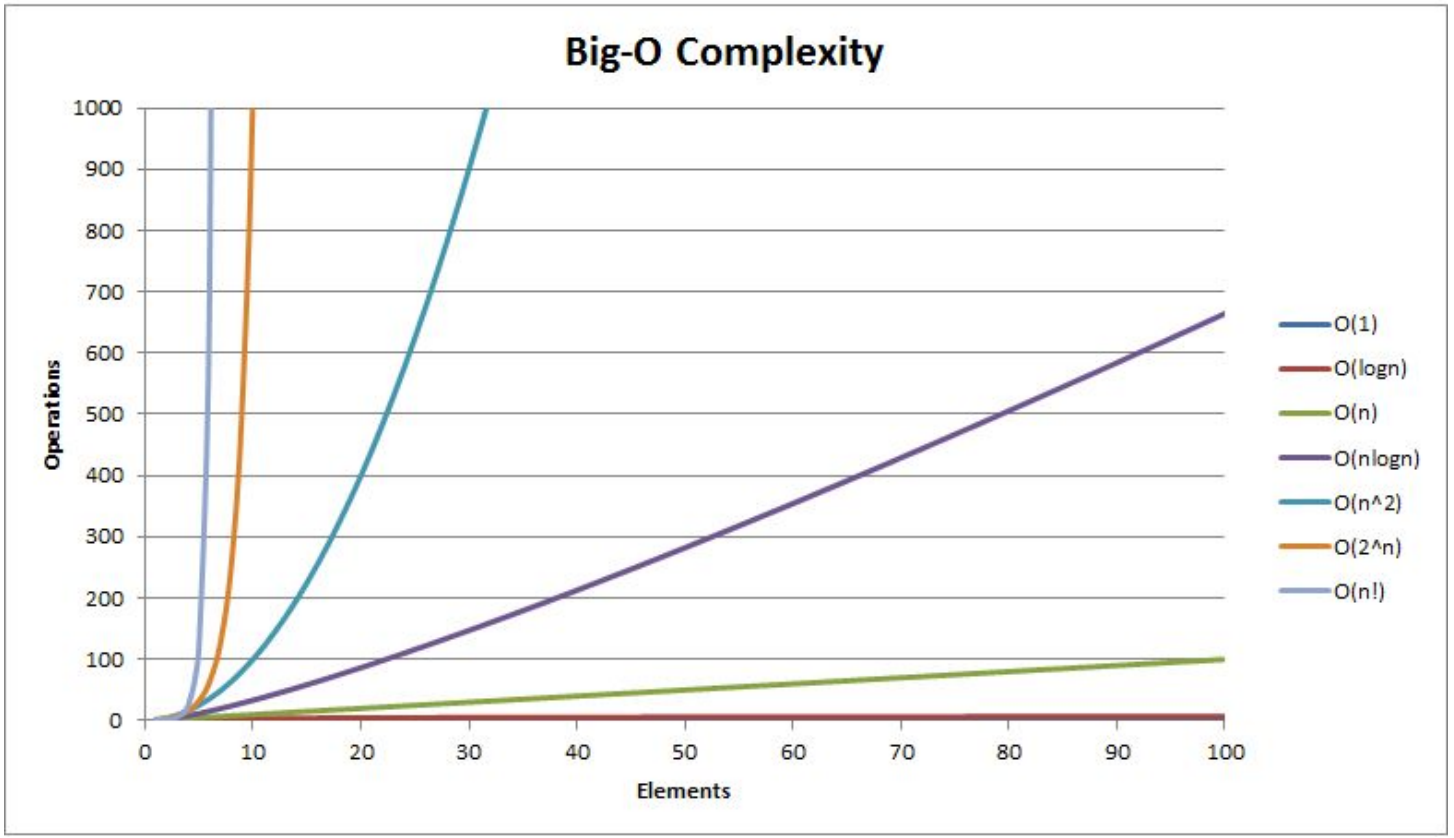
https://github.com/gibsjose/cpp-cheat-sheet/blob/master/Data%20Structures%20and%20Algorithms.md#13-deque-stddeque

Data Structures

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Indexing	Search	Insertion	Deletion	Indexing	Search	Insertion	Deletion	
Basic Array	O(1)	O(n)	-	-	O(1)	O(n)	-	-	O(n)
Dynamic Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartresian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

https://github.com/gibsjose/cpp-cheat-sheet/blob/master/Data%20Structures%20and%20Algorithms.md#13-deque-stddeque

Big-O Complexity Chart



Clasificación de contenedores

Contenedores de secuencias

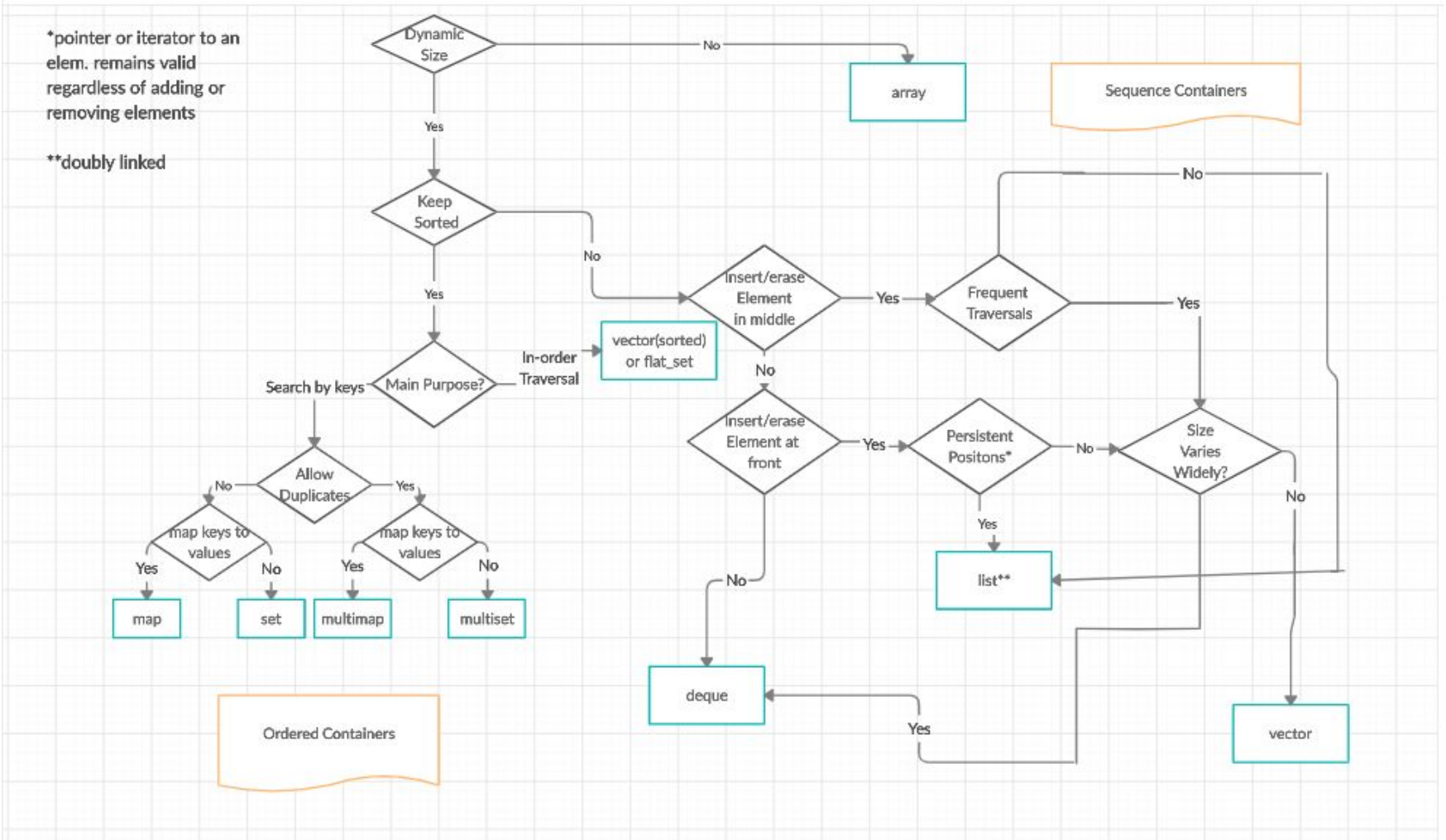
Contenedores Asociativos

Ordenados y Desordenados

Contenedores Adaptadores

Contenedores de secuencias

Implementan estructuras que pueden ser accedidas de forma secuencial.


*pointer or iterator to an elem. remains valid regardless of adding or removing elements
**doubly linked
Dynamic Size
No
array
Sequence Containers
Yes
Keep Sorted
No
Yes
Insert/erase Element in middle
Yes
Frequent Traversals
No
Yes
Main Purpose?
Search by keys
In-order Traversal
vector(sorted) or flat_set
No
Insert/erase Element at front
Yes
Persistent Positons*
No
Size Varies Widely?
No
Yes
Allow Duplicates
No
Yes
map keys to values
map keys to values
Yes
No
Yes
No
map
set
multimap
multiset
list**
No
deque
Yes
vector
Ordered Containers

std::vector

Usado para

- Simple almacenamiento
- Agregar pero no eliminar
- Serialización
- Búsquedas rápidas por índice
- Conversión sencilla a arrays de estilo C
- Recorrido eficiente (caché de CPU contigua)

No usar para

- Inserción/supresión en medio de la lista
- Cambio dinámico del almacenamiento
- Indexación no entera

https://hackingcpp.com/cpp/std/sequence_containers.html

std::deque

Usado para

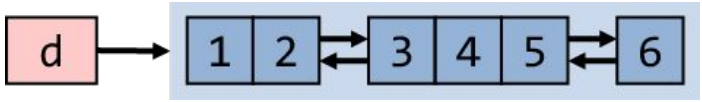
- Propósito similar de std::vector
- Básicamente std::vector con push_front y pop_front eficientes

No usar para

- Almacenamiento contiguo estilo C (no **garantizado)**

Notas

- Se pronuncia 'deck'
- Siglas: Double Ended Queue



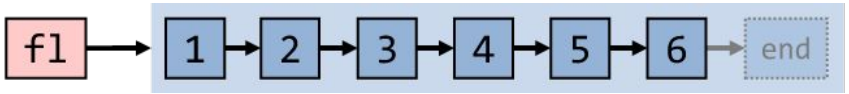
std::forward_list (singly-linked list)

Usado para

- Inserción en el centro/inicio de la lista
- Ordenación eficiente (intercambio de punteros frente a copia)

No usar para

- Acceso directo



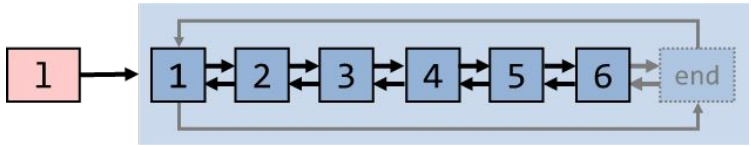
std::list (doubly-linked list)

Usado para

- Inserción en el centro/inicio de la lista
- Ordenación eficiente (intercambio de punteros frente a copia)

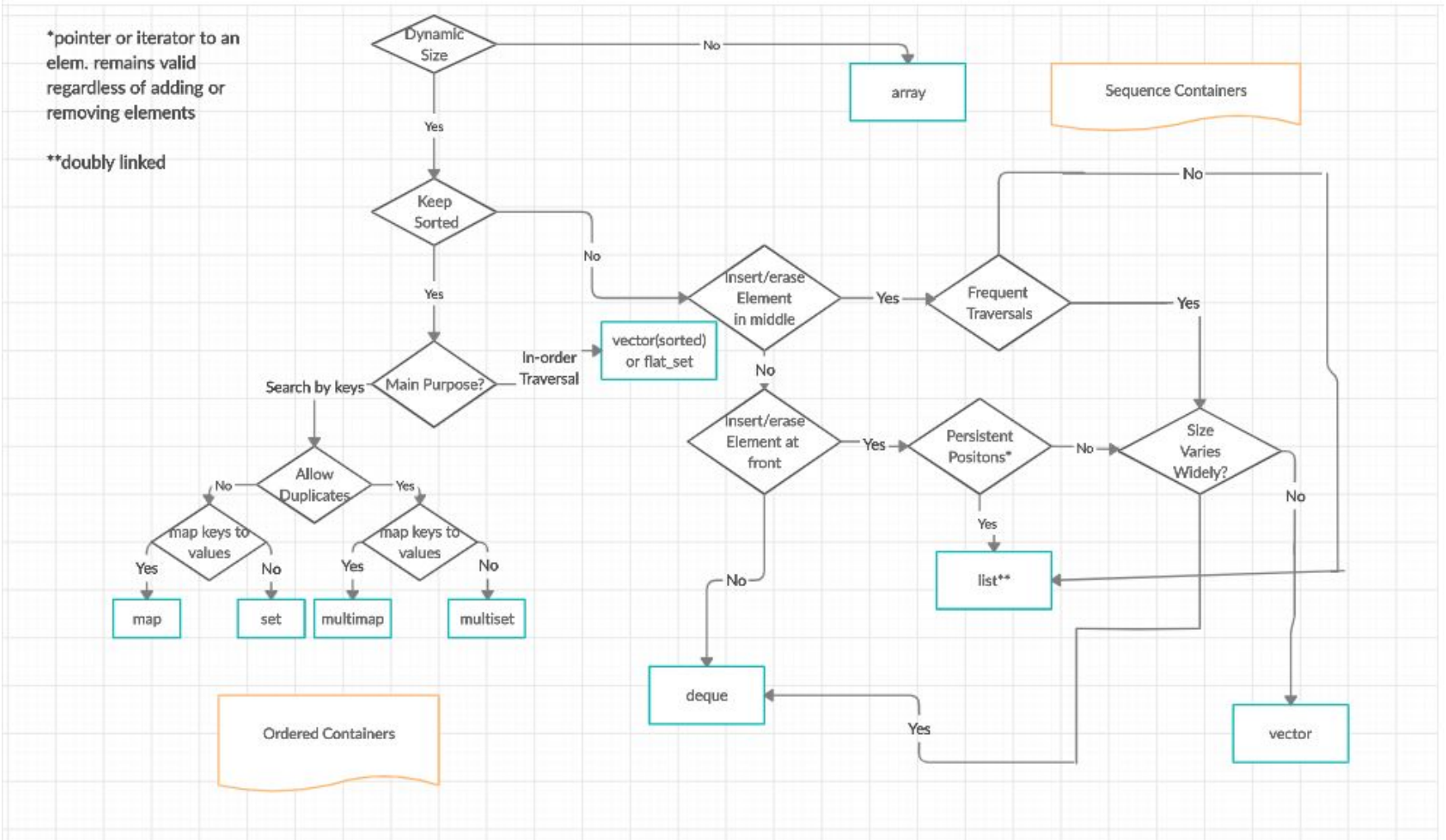
No usar para

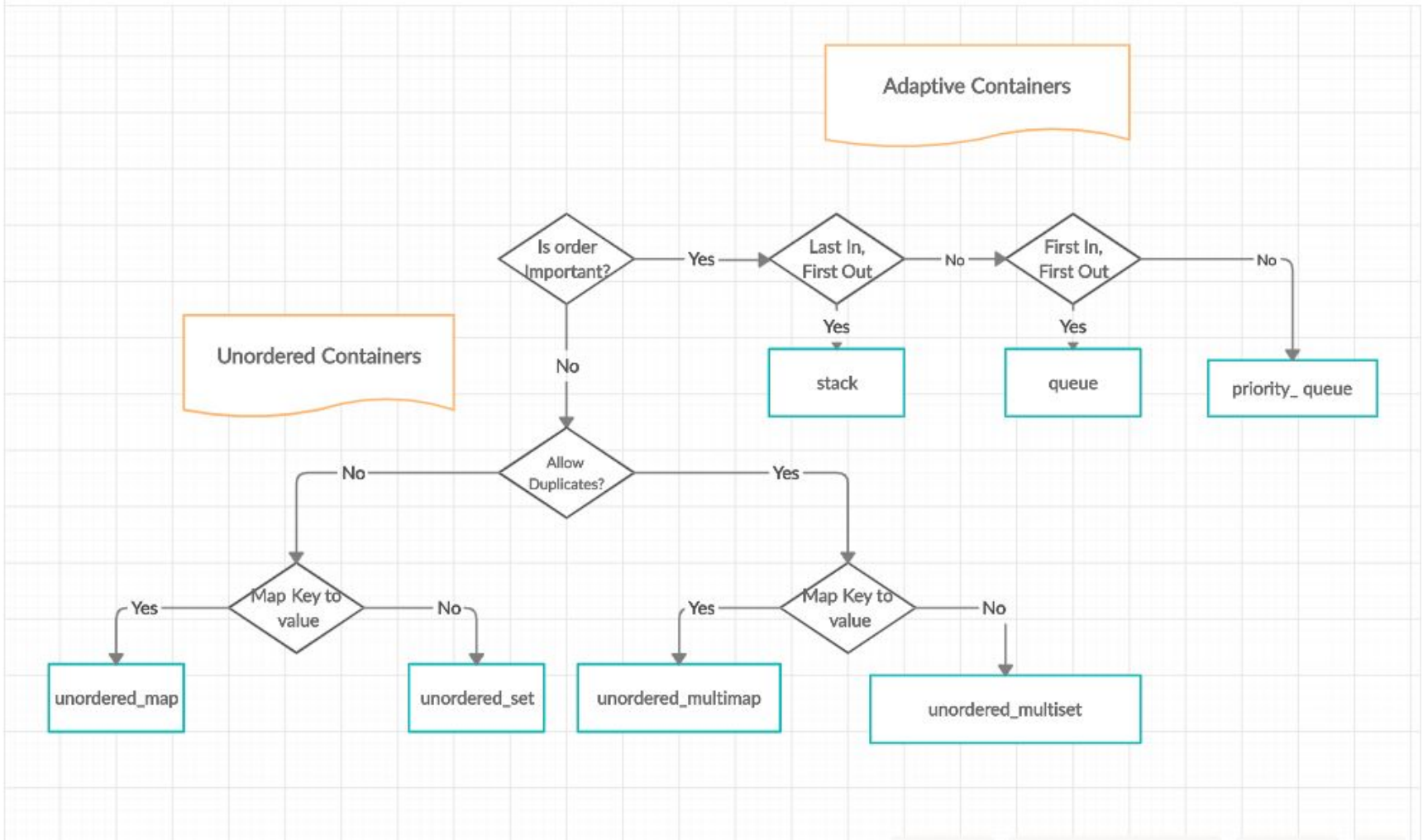
- Es mucho más lento que std::vector
- Acceso directo



Contenedores asociativos

Implementan estructuras de datos ordenados en donde se pueden buscar rápidamente

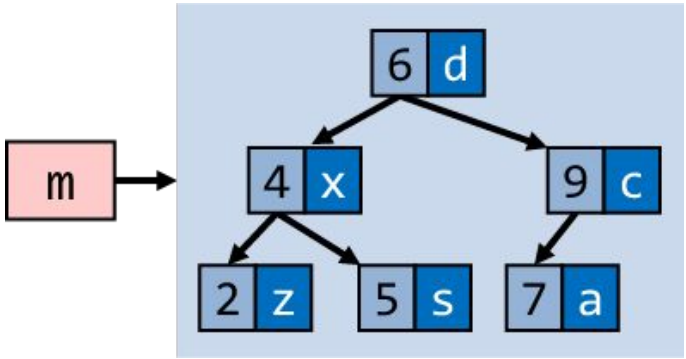

*pointer or iterator to an elem. remains valid regardless of adding or removing elements
**doubly linked
Dynamic Size
No
array
Sequence Containers
Yes
Keep Sorted
No
Yes
Insert/erase Element in middle
Yes
Frequent Traversals
No
Yes
Main Purpose?
Search by keys
In-order Traversal
vector(sorted) or flat_set
No
Insert/erase Element at front
Yes
Persistent Positons*
No
Size Varies Widely?
No
Yes
Allow Duplicates
No
Yes
map keys to values
map keys to values
Yes
No
Yes
No
map
set
multimap
multiset
list**
No
deque
Yes
vector
Ordered Containers


Adaptive Containers
Is order Important?
Yes
Last In, First Out
No
First In, First Out
No
Yes
Yes
stack
queue
priority_ queue
Unordered Containers
No
Allow Duplicates?
No
Yes
Map Key to value
Yes
No
Map Key to value
Yes
No
unordered_map
unordered_set
unordered_multimap
unordered_multiset

std::map - std::unordered_map

Usado para

- Pares Key-value
- Búsqueda constante por keys
- Búsqueda si el key/value existe
- Eliminar duplicados



No usar para

- Ordenamientos

Notas

- Típicamente el map ordenado (std::map) es más lento que el map desordenado (std::unordered_map)
- Los maps son utilizados para *binary search trees*

std::set - std::unordered_set

Usado para

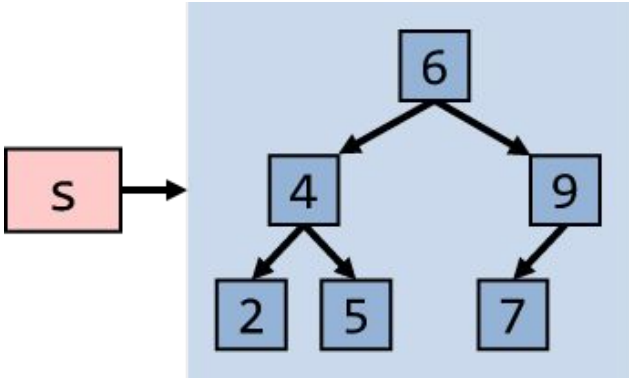
- Remover duplicados
- Almacenamiento dinámico ordenado

No usar para

- Almacenamiento Simple
- Acceso Directo por indexación

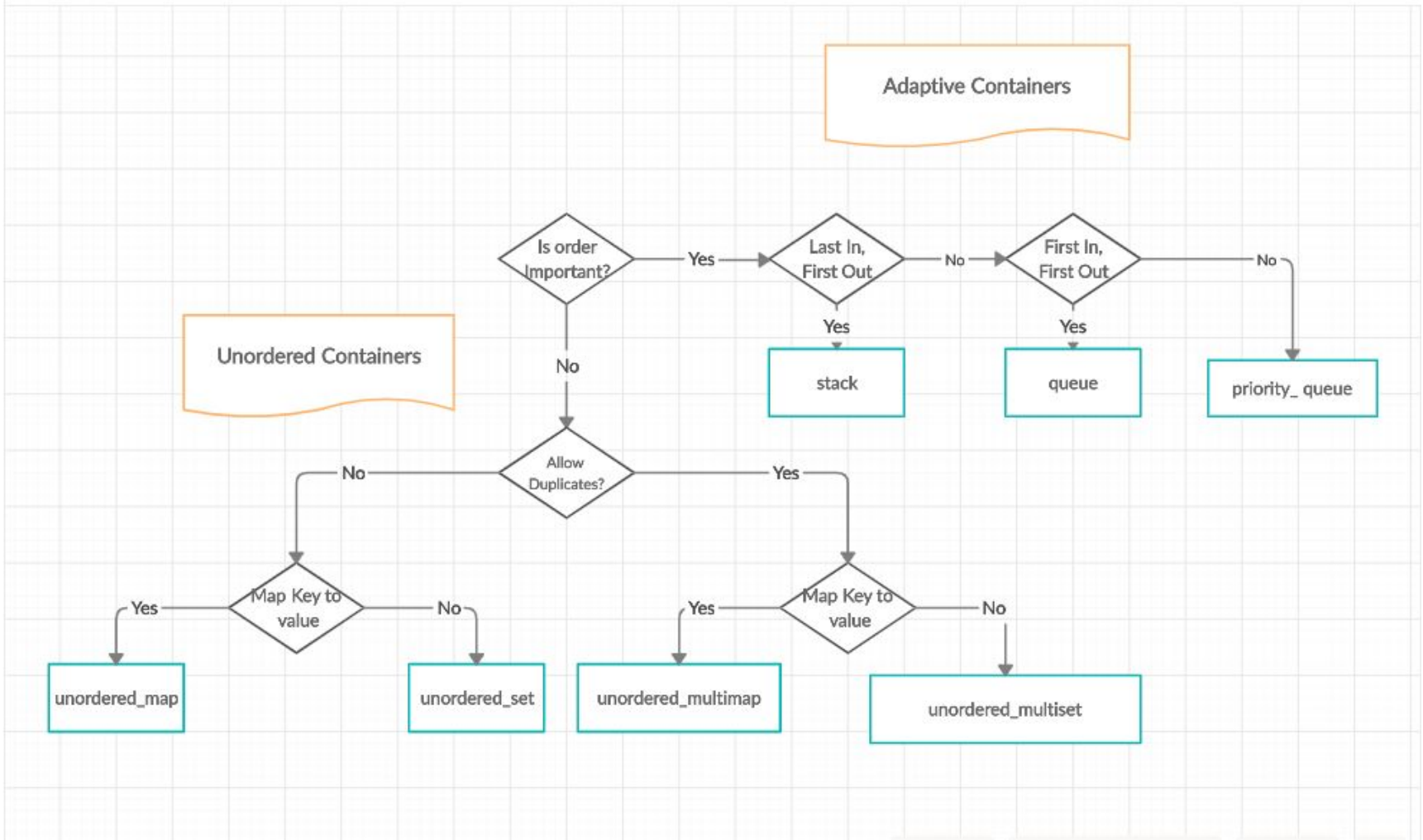
Notas

- Sets son usualmente implementados para binary search trees



Contenedores adaptadores

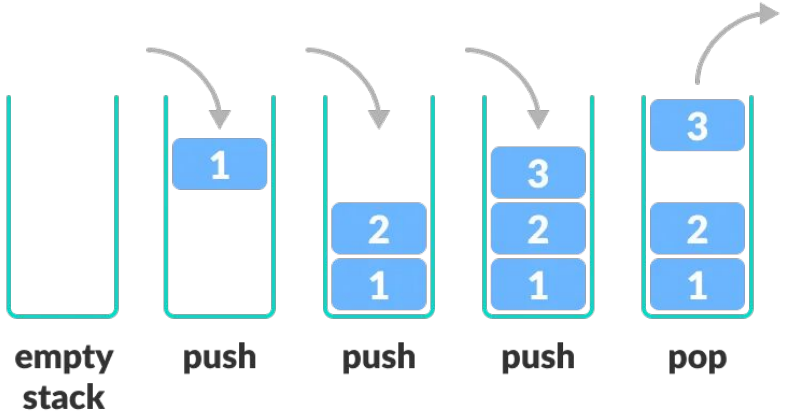
Proporcionan una interfaz diferente para los contenedores secuenciales


Adaptive Containers
Is order Important?
Yes
Last In, First Out
No
First In, First Out
No
Yes
Yes
stack
queue
priority_ queue
Unordered Containers
No
Allow Duplicates?
No
Yes
Map Key to value
Yes
No
Map Key to value
Yes
No
unordered_map
unordered_set
unordered_multimap
unordered_multiset

std::stack

Usar para

- Operaciones First-In Last-Out
- Inversión de elementos



std::queue

Usar para

- Operaciones First-In First-Out
- Ex: Sistema de ordenamiento simple online (el primero que llega es el primero en ser atendido)
- Ex: Semaphore queue handling
- Ex: CPU scheduling (FCFS)

Notas

- Usualmente implementado como un std::deque

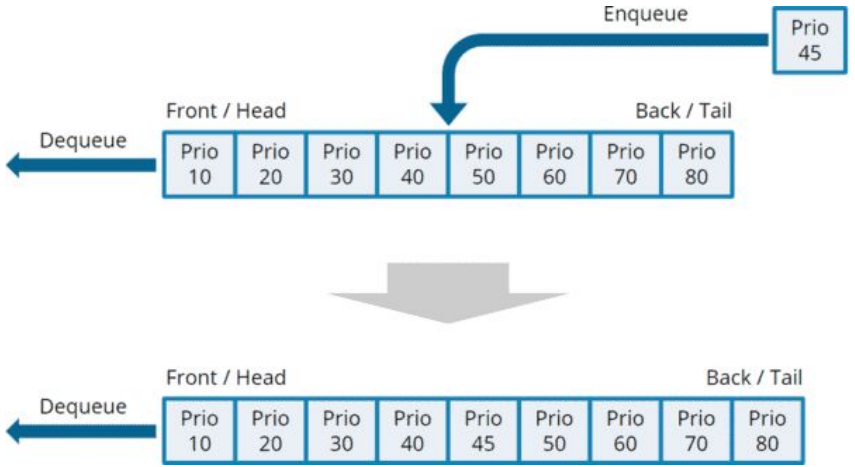
std::priority_queue

Usado para

- Operaciones First-In First-Out donde la prioridad prevalece sobre la hora de llegada
- Ex: CPU scheduling (primero trabajos pequeños, prioridad system/user)
- Ex: Emergencias medicas

Notas

- usualmente implementado como std::vector



Ejercicio 1

Agregar a la Clase Panel Solar

- **std::stack** para guardar los paneles solares (crear una nueva **Clase Deposito**)
- **std::set** para hacer un inventario de los paneles solares por ID. (agregar un nuevo atributo ID a cada panel)
- **std::priority_queue** para reparar paneles solares según la importancia del daño de cada una. (crear una nueva **Clase TallerPL**)