Estructuras de Datos

Listas:

- Secuencia de valores que pueden ser recorridos arbitrariamente en el orden en el que están en la secuencia.
- Suelen ser implementados mediante estructuras enlazadas (listas enlazadas)
- Operaciones:
  - Agregar elemento al inicio: push_front
  - Agregar elemento al final: push_back
  - Eliminar elemento al inicio: pop_front
  - Eliminar elemento al final: pop_back
  - Insertar en posición arbitraria: insert
  - Eliminar de posición arbitraria: erase
  - Tamaño: size
  - Determinar si está vacía: empty

Operación	C++ (list)	C++ (deque)	Python

	`#include <list>`
	`using namespace std;`
	`list<int> l;`
		`#include <deque>`
		`using namespace std;`
		`deque<int> l;`
			`from collections import deque`
			`l = deque()`

push_back	`l.push_back(s);` O(1)	`l.push_back(s);` O(1)	`l.append(s)` O(1)
push_front	`l.push_front(3);` O(1)	`l.push_front(3);` O(1)	`l.appendleft(3)` O(1)
pop_back	`l.pop_back();` O(1)	`l.pop_back();` O(1)	`l.pop()` O(1)
pop_front	`l.pop_front();` O(1)	`l.pop_front();` O(1)	`l.popleft()` O(1)
insert	`l.insert(it, valor);` O(k)	`l.insert(it, valor);` O(k)	`l.insert(pos,valor)` O(k)
erase	`l.erase(it);` O(k)	`l.erase(it);` O(k)	`del l[pos]` O(k)
empty	`l.empty();` O(1)	`l.empty();` O(1)	`len(l) == 0` O(1)
size	`l.size();` O(1)	`l.size();` O(1)	`len(l)` O(1)
	`l.front();` O(1)	`l.front();` O(1)	`l[0]` O(1)
	`l.back();` O(1)	`l.back();` O(1)	`l[-1]` O(1)

Pilas:

- Secuencia de valores en la que solo se tiene acceso por un extremo (tope)
- LIFO: Last In First Out
- Típicamente se implementan como listas enlazadas
- Operaciones:
  - push
  - top
  - pop
  - empty
  - size

Operación	C++ (vector)	C++ (stack)	Python

	`#include <vector>`
	`using namespace std;`
	`vector<int> t;`
		`#include <stack>`
		`using namespace std;`
		`stack <int> t;`
			`from collections import deque`
			`t = deque()`

push	`t.push_back(2);` O(1) amort.	`t.push();` O(1)	`t.append(s)` O(1)
top	`t[t.size()-1];` O(1)	`t.top();` O(1)	`t[-1]` O(1)
pop	`t.pop_back();` O(1)	`t.pop();` O(1)	`t.pop()` O(1)
size	`t.size();` O(1)	`t.size();` O(1)	`len(t)` O(1)
empty	`t.empty();` O(1)	`t.empty();` O(1)	`len(t)==0` O(1)

Colas:

- Secuencia de valores en la que se tiene acceso por 2 extremos, uno para agregar elementos, otro para remover elementos.
- FIFO: First In First Out
- Típicamente se implementan como listas enlazadas
- Operaciones:
  - push
  - front
  - pop
  - size
  - empty

Operación	C++ (queue)	Python

	`#include <queue>`
	`using namespace std;`
	`queue<int> t;`
		`from collections import deque`
		`t = deque()`

push	`t.push();` O(1)	`t.append(s)` O(1)
front	`t.front();` O(1)	`t[0]` O(1)
pop	`t.pop();` O(1)	`t.popleft()` O(1)
size	`t.size();` O(1)	`len(t)` O(1)
empty	`t.empty();` O(1)	`len(t)==0` O(1)

Tablas de Direccionamiento Directo:

* Son estructuras de datos en las que se asocian claves a valores. Estas claves determinan la ubicación en la que se almacenan los valores.
* Es requerido que en la implementación se pueda obtener de forma eficiente los valores a partir de sus claves.
* Los arreglos (vectores) y mapas en C++, y las listas nativas y diccionarios de Python son implementaciones de esta estructura de datos.
* Operaciones:
  * assign
  * update
  * query

Operación	C++ (arreglos y vectores)	C++ (mapas)	Python (listas nativas)	Python (Diccionarios)

	`#include <vector>`
	`using namespace std;`
	`vector<int> t;`
	`int t[TAM];`
		`#include <map>`
		`using namespace std;`
		`map<string, int> t;`
			`t=[]`	`t=dict()  # t={}`

assign	`t.push_back(10);` O(1) amort. `t[0]=10;` O(1)	`t["hola"]=5;` O(log n)	`t.append(10)` O(1) amort. `t[0]=10` O(1)	`t["hola"]=5` O(1)
update	`t[0]=8;` O(1)	`t["hola"]=5;` O(log n)	`t[0]=8` O(1)	`t["hola"]=5` O(1)
query	`t[pos]` O(1)	`t[clave]` O(log n) `t.find(clave)==t.end()` O(log n)	`t[pos]` O(1)	`t[clave]` O(1) `clave in t`

Colas de Prioridad

* Corresponden a colas en las que los elementos no salen en el orden en el que entraron sino de acuerdo a una prioridad.
* La prioridad es determinada con respecto a algún criterio que puede ser el valor mismo del elemento o alguna cantidad que se pueda calcular sobre el valor del elemento.
* Suelen ser implementadas mediante montículos (heaps), árboles binarios de búsqueda balanceados u otras estructuras que soporten operaciones O(log n).
* Opciaciones:
  * push
  * top
  * pop
  * size
  * empty

Operación	C++ (priority-queue)		Python	
	#include <queue> using namespace std; priority_queue<int> t;		from heapq import heappush, heappop t = []	
push	t.push();	O(log n)	heappush(t,10)	O(log n)
front	t.top()	O(1)	t[0]	O(1)
pop	t.pop();	O(log n)	heappop(t)	O(log n)
size	t.size();	O(1)	len(t)	O(1)
empty	t.empty();	O(1)	len(t)==0	O(1)

Conjuntos:

* Agrupaciones de elementos en las que no hay elementos repetidos y no hay orden para los elementos.
* Asociación con conjuntos matemáticos
* Operaciones:
  - insert
  - erase
  - empty
  - size

Operación	C++		Python	
	#include <set> using namespace std; set<int> t;		t=set()	
insert	t.insert(10)	O(log n)	t.add(10)	O(1)
erase	t.erase(10)	O(log n)	t.discard(10) t.remove(10)	O(1)
size	t.size();	O(1)	len(t)	O(1)
empty	t.empty();	O(1)	len(t)==0	O(1)