CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2023

Sesión 2 (2ª Semana)

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín(raul.martin@urjc.es)
- Sergio Salazar (s.salazarc.2018@alumnos.urjc.es)
- Francisco Tórtola (f.tortola.2018@alumnos.urjc.es)
- Cristian Pérez(c.perezc.2018@alumnos.urjc.es)
- **Xuqiang Liu(x.liu1.2020@alumnos.urjc.es)**
- Alicia Pina(a.pinaz.2020@alumnos.urjc.es)
- Sara García(s.garciarod.2020@alumnos.urjc.es)
- Raúl Fauste(r.fauste.2020@alumnos.urjc.es)

IMPORTANTE!

- Dossier máximo 10 páginas en fuente Arial tamaño 10.

  https://www.overleaf.com/read/mhcdkrbxmfnv
- Sin acceso a internet
- Varias pantallas para leer problemarios
- Información y Contraseñas llegarán por correo

https://ada-byron.es/2023/reg/madrid/locales/urjc/index.php

Contenidos

- Complejidad -> DAA
- Estructuras de Datos Básicas
  - Strings -> IP
  - Arrays, Matrices -> IP
  - Listas, Pilas, Colas -> ED, IP2
  - Conjuntos y Mapas -> EDA, PA

Complejidad

- Medimos cuántas iteraciones hará nuestro programa en relación al tamaño (n) del problema
  - Un bucle realiza n iteraciones
  - Dos bucles anidados n^2 iteraciones

Complejidad

- Estimación de la eficiencia (cota)
- Sirve para evitar time limit exceeded (TLE)
- Se considera el peor caso (Notación O)
- No hace falta una demostración matemática, solo una idea intuitiva
  - Importa ser rápido
  - Consideramos bucles y llamadas a funciones para hacernos una idea

Complejidad

Ejemplo 1:

```
#include <stdio.h>

const int MAXN = 1000;
const int MAXM = 2500;
int mat[MAXN][MAXM];

int main(int argc, char **argv)
{
    int n,m;
    scanf("%d %d", &n, &m);
    for(int i=0; i<n;i++)
        for(int j=0; j<m; j++)
            scanf("%d", &mat[i][j]);

    return 0;
}
```

Complejidad

Ejemplo 1:

- Tenemos dos bucles anidados para leer datos de entrada
- Se realizan M lecturas un total de N veces
  - Su complejidad es O(N*M) ó $O(N^2)$

- En este curso solo hablaremos de la complejidad en **tiempo** pero existe también la complejidad en el espacio (memoria)

Complejidad

Ejemplo 2:

```
def f(n):
  if n <= 1: return 1
  return f(n-1) + f(n-2)

print(f(int(input())))
```

Complejidad

Ejemplo 2:

- Resolución del caso base es constante = O(1)
- Por cada estado, dos opciones, O(1) o 2 llamadas
- Cada llamada se ejecuta un número N de veces

```
def f(n):
  if n <= 1: return 1
  return f(n-1) + f(n-2)

print(f(int(input())))
```

Complejidad

Ejemplo 2:

Finalmente, podemos decir que, en recursión, la complejidad en tiempo será igual al número de transiciones, potenciado a la cantidad de veces que se puede llamar al estado recursivo.

Para este ejemplo: 2**N

```
def f(n):
  if n <= 1: return 1
  return f(n-1) + f(n-2)

print(f(int(input())))
```

Complejidad

Noción **aproximada** de complejidad

Complejidad	**Iteraciones por segundo**
O(1)	----
O(LgN)	$2^{1000000}$
O(N)	1,000,000
O(NLgN)	100,000
O(N^2)	1,000 ~ 3,000
O(N^3)	100 ~ 300
O(2^N)	20
O(N!)	12

Estructuras de Datos Básicas

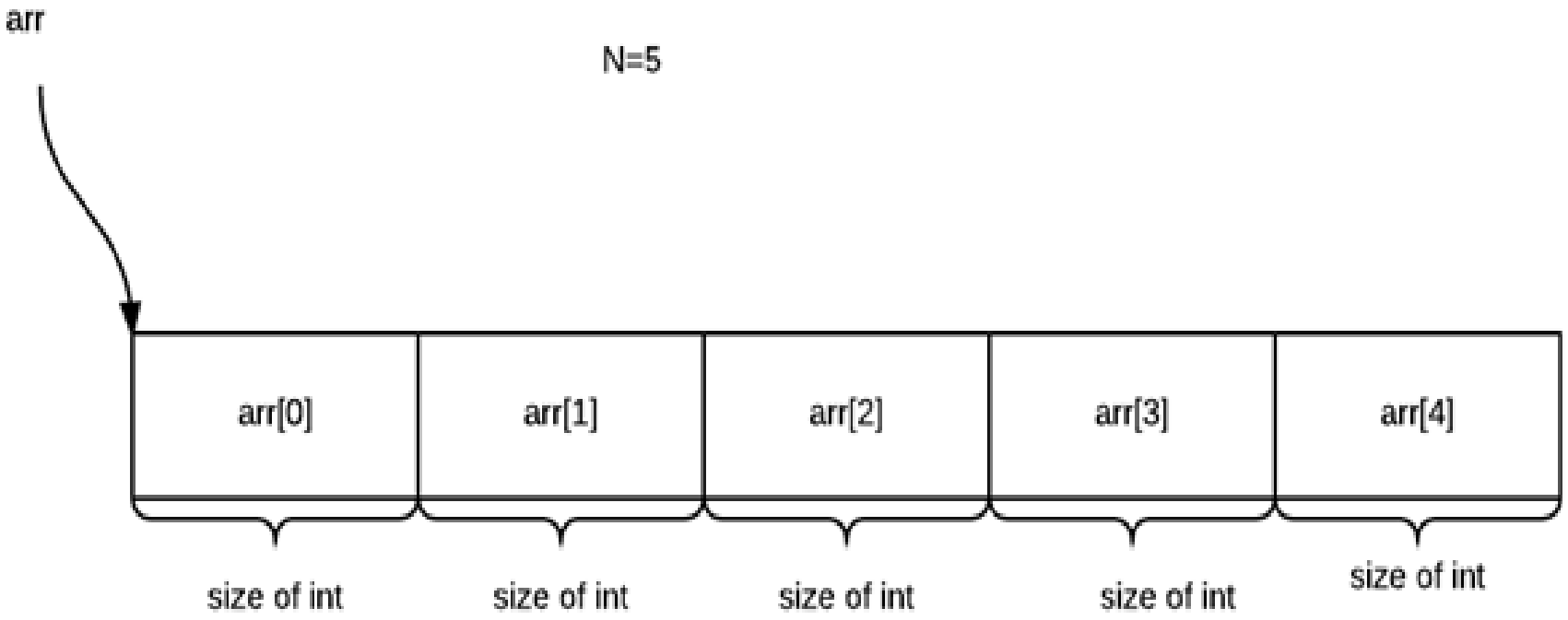
- Arrays
- Strings
- Vectores (ArrayList)
- Listas
- Pilas
- Colas
- Map (estructura <clave,valor>)
- Set

Estructuras de Datos Básicas

Arrays

- Estructura en la cual se guardan elementos
- Los valores se acceden mediante índices (`A[n]` en c++ y java)
- La reserva de memoria se hace una única vez
- La memoria utilizada es contigua

Estructuras de Datos Básicas



Estructuras de Datos Básicas

Arrays

- ¿Cuántas iteraciones hacen falta para?
  - Buscar un elemento
  - Eliminar un elemento
  - Insertar un elemento

Estructuras de Datos Básicas

Arrays

- ¿Cuántas iteraciones hacen falta para?
  - Buscar un elemento
    - -O(n)
    - -O(1) accediendo a los indices
  - Eliminar un elemento
    - -O(n) requiere otro array
  - Insertar un elemento
    - -O(n) requiere otro array

Estructuras de Datos Básicas

Strings

- Estructura en la cual se guardan caracteres
- Por simplicidad del curso, solo ASCII
- Los valores se acceden mediante índices (A[n] en c++, charAt(index) en java)
- Métodos útiles somo substring, find (indexOf)

Estructuras de Datos Básicas

- En C++ los string requieren de un carácter nulo \0 para delimitar su fin
  - Almacenar 6 caracteres requiere reservar al menos 7 espacios de memoria

Estructuras de Datos Básicas

Strings

- ¿Cuántas iteraciones hacen falta para?
  - Buscar una cadena en el string
  - Copiar una subcadena (substring)
  - Concatenar dos strings

Estructuras de Datos Básicas

Strings

- ¿Cuántas iteraciones hacen falta para?
  - Buscar una cadena en el string
    - O(n)
  - Copiar una subcadena (substring)
    - O(n)
  - Concatenar dos strings
    - O(n+m)

Estructuras de Datos Básicas

Vectores

- Estructura en la cual se guardan valores sobre un elemento
- Funciona como un array, en memoria se guarda en espacios contiguos

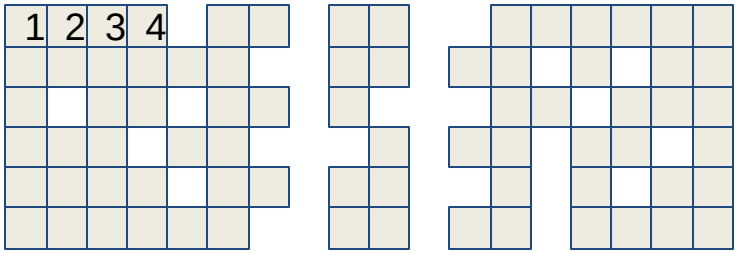
Estructuras de Datos Básicas

Vectores / Vector

- ¡Si crece en mucha medida sin una previa reserva puede ser mortal!
- Su tamaño puede cambiar dinámicamente y esto lo convierte en un objetivo ideal para crear matrices dispersas
- Equivalente en Java:
  - ArrayList
  - Vector (obsoleto, NO usar)

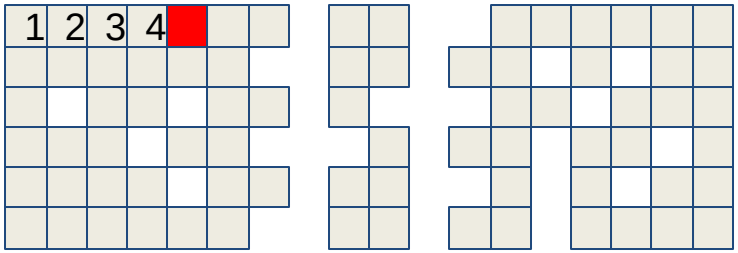
Estructuras de Datos Básicas

Vectores / Vector



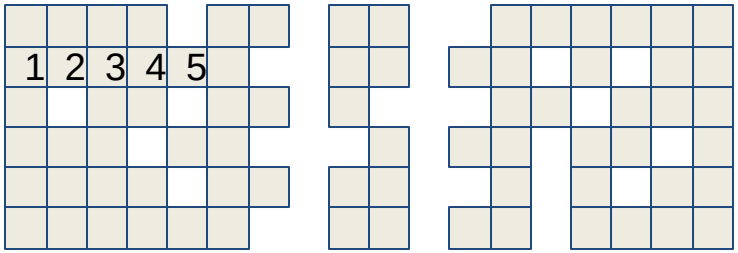
Estructuras de Datos Básicas

Vectores / Vector



Estructuras de Datos Básicas

Vectores / Vector



Estructuras de Datos Básicas

Listas (enlazadas) / List

- Estructura en la cual se guardan valores sobre un elemento, insertando en cualquier lugar un nuevo elemento
- Posiciones de memoria no-contiguas
- Inserciones y borrados mucho más fáciles de lograr internamente

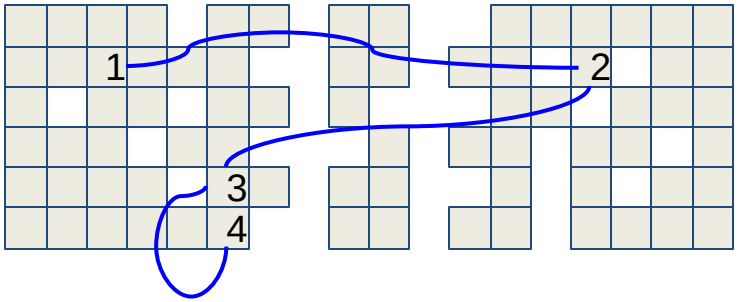
Estructuras de Datos Básicas

Listas (enlazadas) / List

- No se puede tener acceso directo O(1) a un elemento en específico (salvo primero o último)
- Puede servir tanto de pila como de cola
- Equivalente en Java: LinkedList

Estructuras de Datos Básicas

Listas (enlazadas) / List



Estructuras de Datos Básicas

Pilas / stack

- Estructura donde el último elemento en llegar es el primero en salir (LIFO)
- No utiliza espacio contiguo de memoria
- Sólo se puede insertar elementos apilándolos
- Sólo se puede pedir elementos del tope de la pila

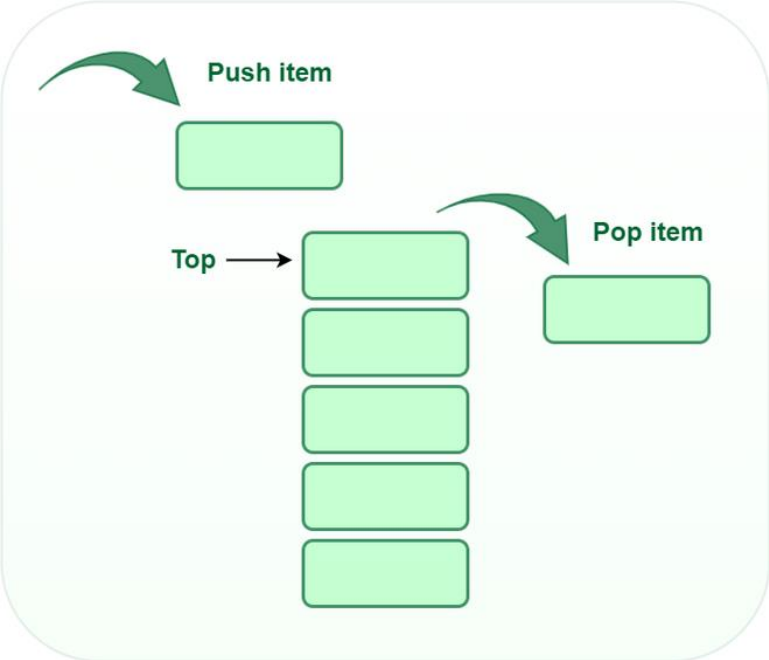
Estructuras de Datos Básicas

Pilas / stack

- En algunos casos, la pila puede “simular” una pila de sistema para reducir el tiempo de ejecución
- Ideal para DFS y otros algoritmos recursivos
- Equivalente en Java: LinkedList, **ArrayDeque** (métodos pop, peek, push)
  - Stack (obsoleto, no usar)

Estructuras de Datos Básicas

Pilas / stack



Estructuras de Datos Básicas

Colas / queue

- Estructura donde el primer elemento en llegar es el primero en salir (FIFO)
- No utiliza espacio contiguo de memoria
- Sólo se puede insertar elementos al final (encolando)

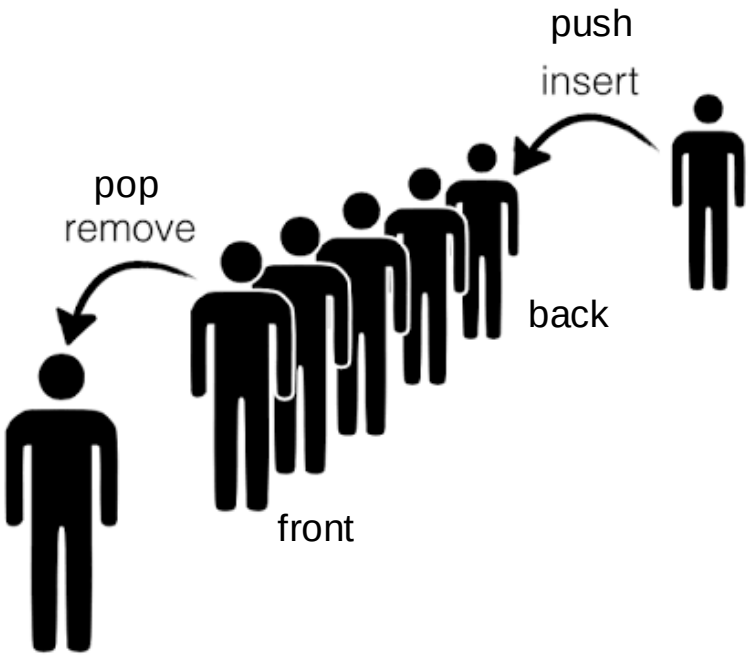
Estructuras de Datos Básicas

Colas / queue

- Sólo se puede pedir el elemento del principio de la cola
- Ideal para BFS
- Equivalente en Java: ArrayDeque (métodos de la interfaz Queue: offer, peek, poll)

Estructuras de Datos Básicas

Colas / queue



Estructuras de Datos Básicas

Cola de prioridad / priority_queue

- Estructura similar a una cola donde los elementos tienen asignado una prioridad.
- No utiliza espacio contiguo de memoria
- Sólo se puede insertar elementos al final (encolando)

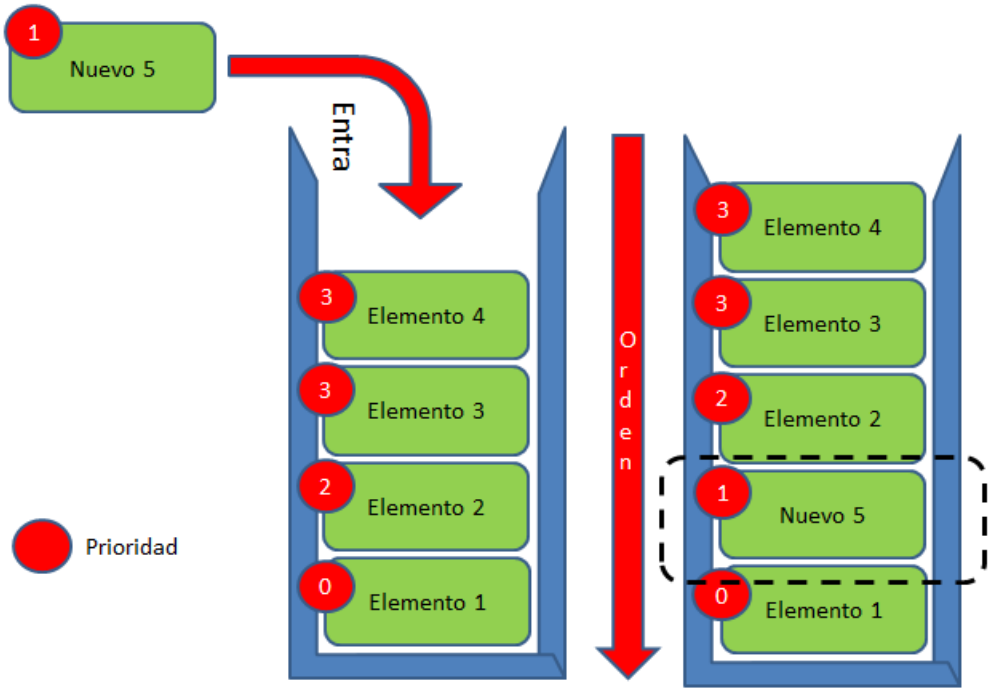
Estructuras de Datos Básicas

Cola de prioridad / priority_queue

- Sólo se puede pedir el elemento del principio de la cola
- Ideal para Dijkstra
- Equivalente en Java: PriorityQueue (métodos de la interfaz Queue: offer, peek, poll,add)
- Equivalente en C++: priority_queue (métodos de la interfaz : push,top,pop)

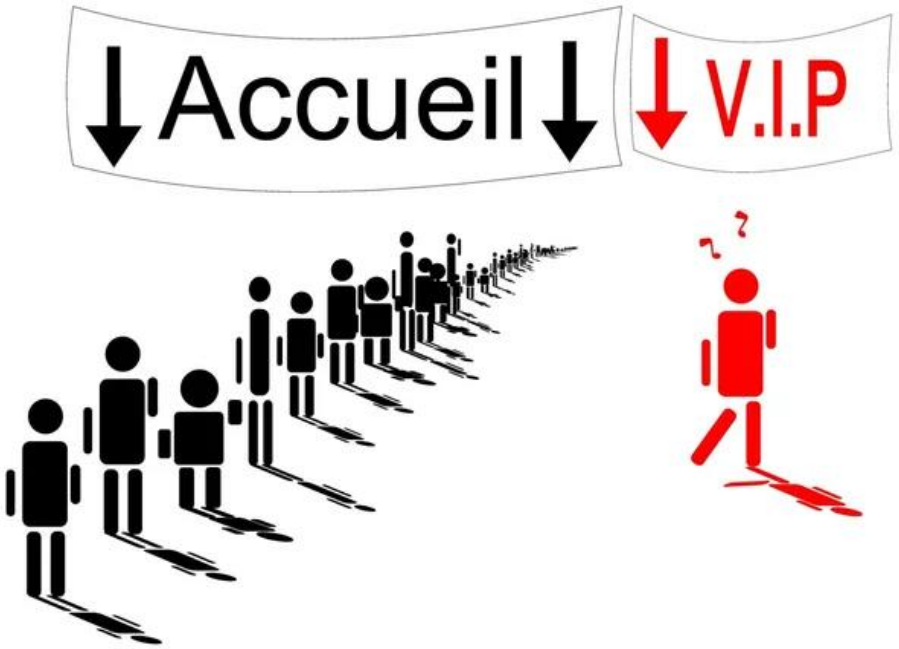
Estructuras de Datos Básicas

Cola de prioridad / priority_queue



Estructuras de Datos Básicas

Cola de prioridad / priority_queue



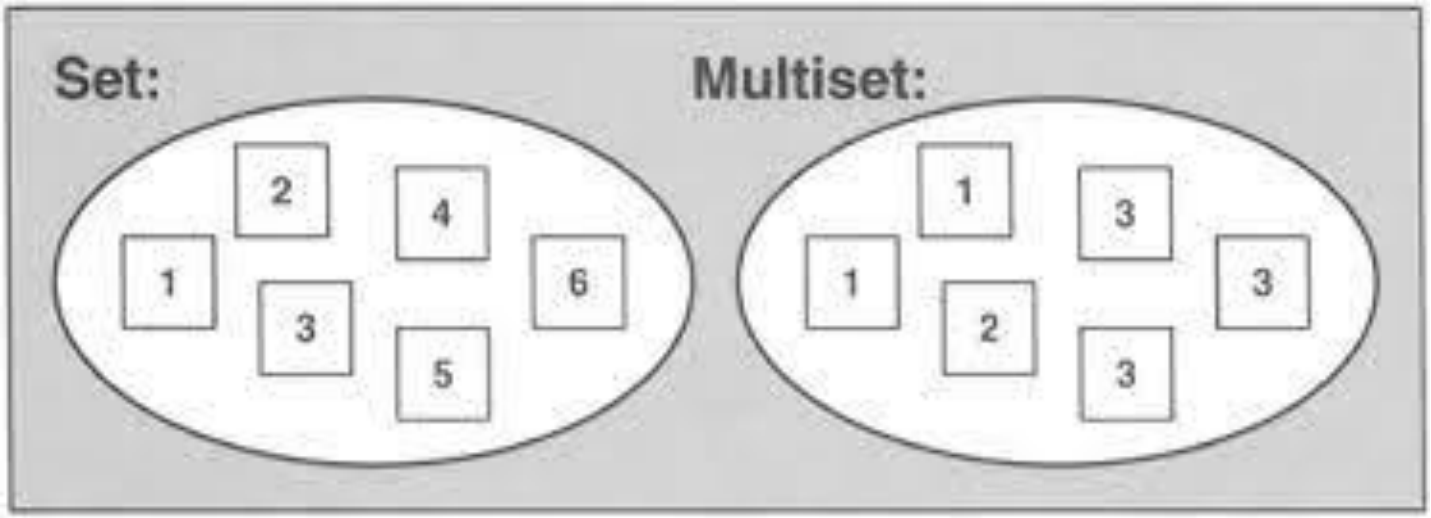
Estructuras de Datos Básicas

Conjuntos / set

- No admite elementos repetidos
- Árbol binario balanceado (C++), ordena naturalmente de menor a mayor. (podría ser una tabla hash con unordered_set)
- Equivalente en Java:
  - TreeSet: ordenado, operaciones O(logn)
  - HashSet: no ordenado, operaciones O(1)

Estructuras de Datos Básicas

Conjuntos / set



Estructuras de Datos Básicas

Mapa / map

- Contenedor asociativo que guarda claves únicas y les asocia a un valor
- Se puede acceder directamente a elementos guardados si se coloca su clave, en tal caso, se devolverá el valor asociado
- Al ser un árbol binario balanceado (C++), todas sus operaciones son logarítmicas (también podría ser tabla hash con unordered_map)
- Equivalente en Java:
  - HashMap: no ordenado, operaciones O(1)
  - TreeMap: ordenado, operaciones O(logn)

Estructuras de Datos Básicas

Typedef

Estructuras de Datos Básicas

Typedef

```
#include <bits/stdc++.h>
using namespace std;

typedef struct{
    int ball_num;
    string name;
    string flavors[3];
    float price;
    void print(){
        cout<<"Name: "<<name<<endl;
    }
}icecream;

int main() {
    icecream icecream1;
    icecream1.ball_num=3;
    icecream1.name="Ibiza";
    icecream1.flavors[0]="Mint";
    icecream1.flavors[0]="Chocolate";
    icecream1.flavors[0]="Vanilla";
    icecream1.price=4.5;
    icecream1.print();
}

```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return str + str2

function f(treeMap, k, newVal):
  if treeMap.contains(k):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return str + str2 -- O(N)

function f(treeMap, k, newVal):
  if treeMap.contains(k):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(N)

function f(treeMap, k, newVal):
  if treeMap.contains(k):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(N)

function f(treeMap, k, newVal):
  if treeMap.contains(k):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  if O(LgN):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  if O(LgN):
    treeMap.put(k, concat(treeMap.get(k), newVal))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  if O(LgN):
    treeMap.put(k, O(M+LgN))

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  if O(LgN):
    O(LgN + M)

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  O(1 + (LgN+M) + LgN)

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  O(LgN+M)

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

```
function concat(str, str2):
  return O(M)

function f(treeMap, k, newVal):
  O(LgN+M)

for i from 0 to N
  k = read(), newVal = read()
  f(treeMap, k, newVal)
```

Complejidad con Estructuras

Ejemplo: Cuál es la complejidad del siguiente código

O(N(LgN + M))
O(NM) para M > LgN

Problemas

https://www.aceptaelreto.com/problem/statement.php?id=418&cat=23

https://www.aceptaelreto.com/problem/statement.php?id=452&cat=13

https://www.aceptaelreto.com/problem/statement.php?id=270&cat=23

https://www.aceptaelreto.com/problem/statement.php?id=647&cat=23

Próxima semana

Clase Práctica (24/02)

- **Clasificatoria Adabyron**

Clase Teórica (03/03)

- Algoritmos de ordenamiento
- Búsqueda binaria
- Algoritmos voraces

¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente con copia a algunos / todos los docentes)

- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín(raul.martin@urjc.es)
- Sergio Salazar (s.salazarc.2018@alumnos.urjc.es)
- Francisco Tórtola (f.tortola.2018@alumnos.urjc.es)
- Cristian Pérez(c.perezc.2018@alumnos.urjc.es)
- Xuqiang Liu(x.liu1.2020@alumnos.urjc.es)
- Alicia Pina(a.pinaz.2020@alumnos.urjc.es)
- Sara García(s.garciarod.2020@alumnos.urjc.es)
- Raúl Fauste (r.fauste.2020@alumnos.urjc.es)