

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación



IIC2115 - Programación como Herramienta para la Ingeniería

Estructuras de datos

Profesora: Francesca Lucchini
Prof. Coordinador: Hans Löbel

Tipos de dato

- Tipos numéricos

int

enteros

float

racionales

- Tipos textuales

str

texto

- Tipos lógicos (booleanos)

bool

lógico

Operadores

- Aritméticos (para ints y floats principalmente, pueden definirse para otros)



Suma



Resta o
Inverso
aditivo



Producto



División



División
Entera



Resto



Potencia

- Comparación



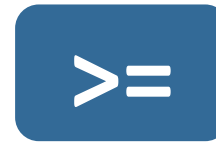
Menor



Menor
o igual



Mayor



Mayor
o igual



Igual



Distinto

Operadores

- Lógicos (para bools)

not

Negación

and

Y

or

O

| A | B | A AND B | A OR B | NOT A |
|-------|-------|---------|--------|-------|
| False | False | False | False | True |
| False | True | False | True | True |
| True | False | False | True | False |
| True | True | True | True | False |

- Texto (para strs)

+

Concatenación

Repetición

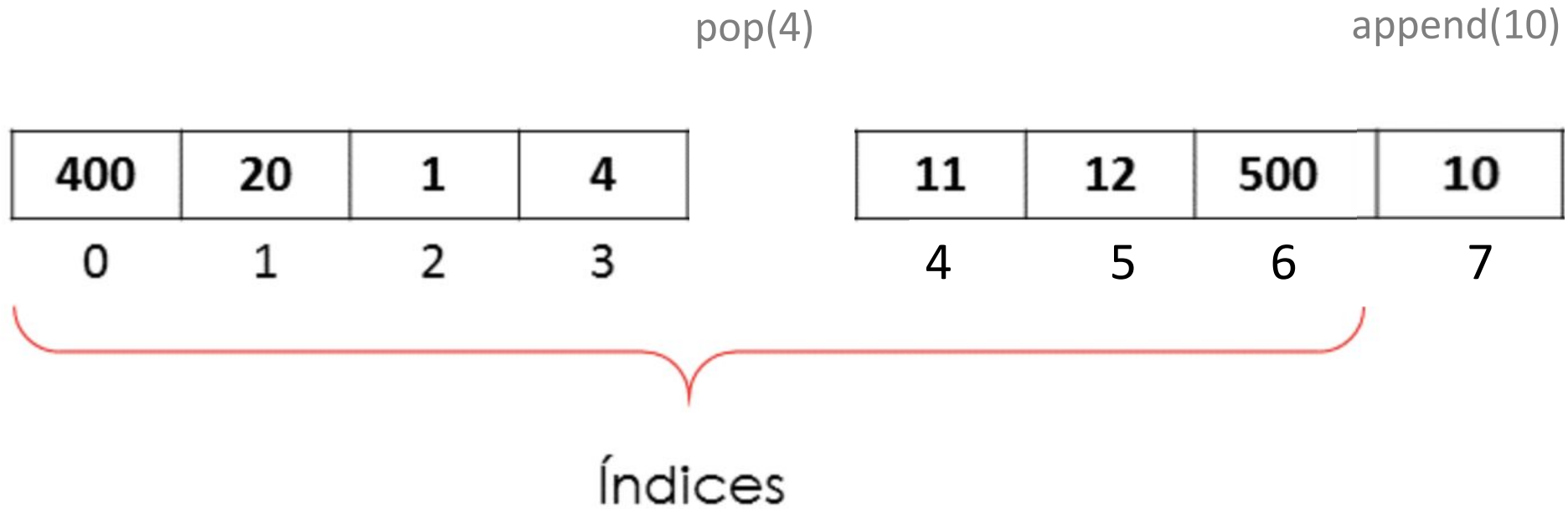
Estructuras de datos

Son **tipos de datos especializados**, diseñados para **agrupar, almacenar o acceder** a la información de manera más **eficiente** que un tipo de dato básico (como int, float, etc). Algunos ejemplos son los siguientes:

- Clases
- Listas
- Tuplas
- Diccionarios y sets
- Stacks y colas
- Árboles

Listas


- Las listas son estructuras que guardan datos de forma **ordenada**.
- Son mutables (modificables).



Tuplas

- Similares a las listas, permiten manejar datos de forma ordenada.
- Al igual que las listas, se accede a los datos mediante índices basados en el orden que fueron ingresados.
- A diferencia de las listas, son **inmutables**.

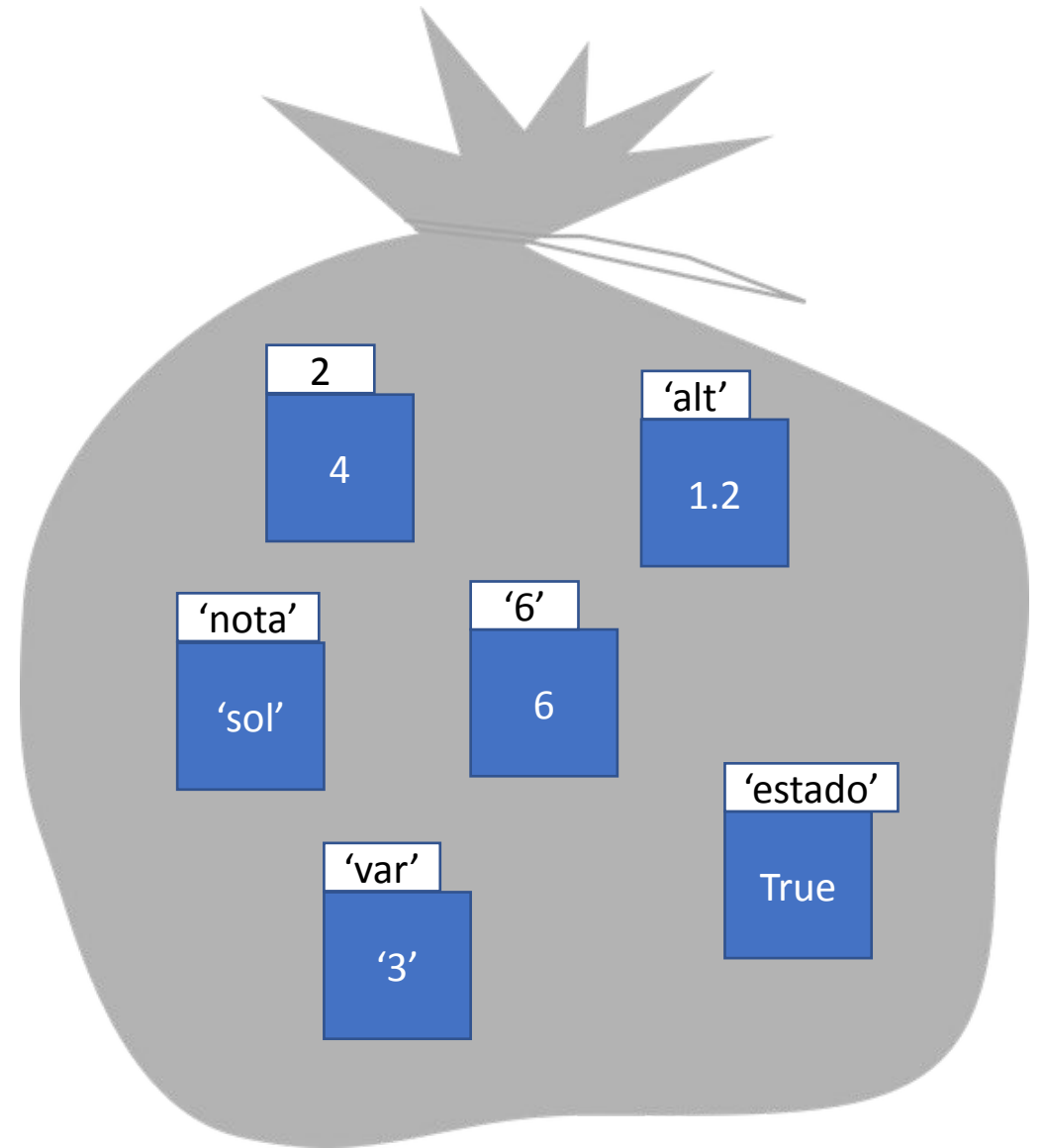
| | | | | | | | |
|------------|-----------|----------|----------|-----------|-----------|-----------|------------|
| 400 | 20 | 1 | 4 | 10 | 11 | 12 | 500 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



Índices

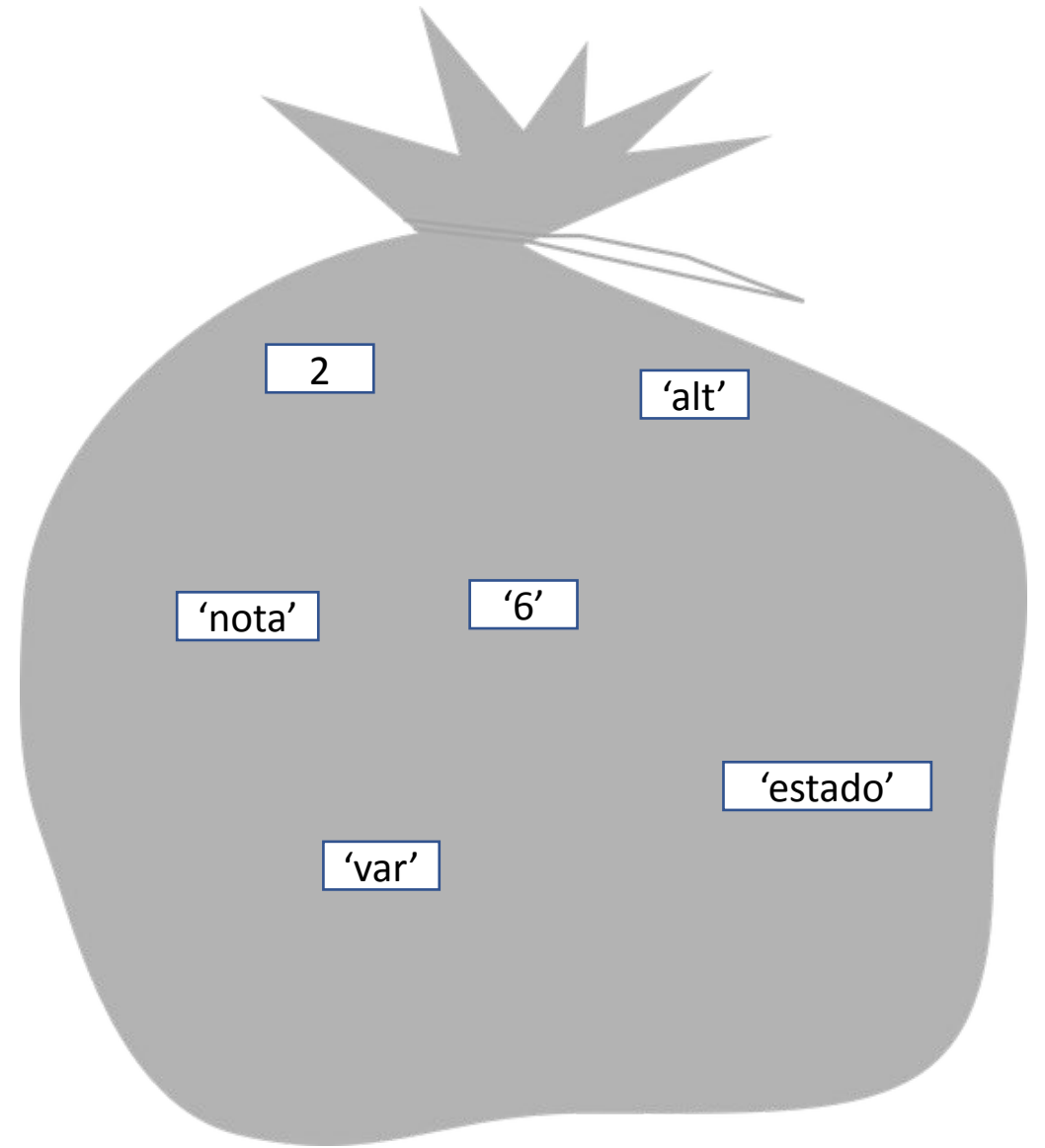
Diccionarios

- Permiten almacenar datos basados en una asociación de pares de elementos, a través de una relación **llave-valor**.
- Acceso a valores a través de la llave es instantáneo, no se necesita realizar una búsqueda (análogo a un índice).
- Se prefiere a una lista cuando el caso de uso más común no implica revisar todos los elementos, sino solo algunos fácilmente encontrables a través de la llave.



Sets

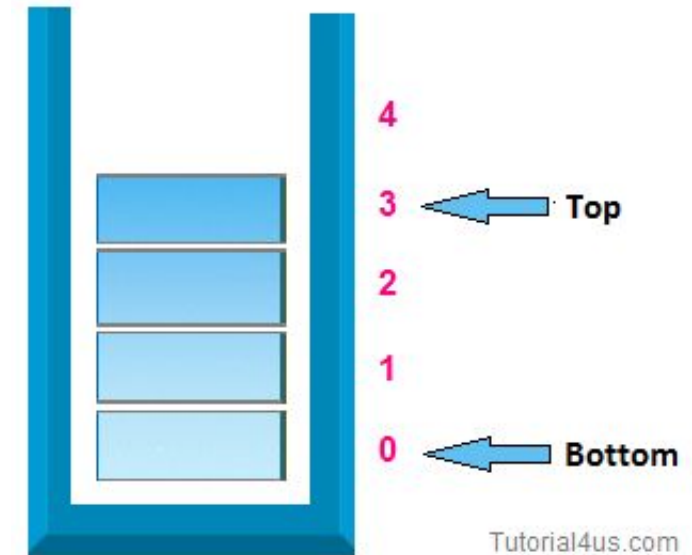
- Son como los diccionarios, pero solo consideran la llave (no hay valor asociado).
- Ideales para verificar la existencia de algo.
- Al igual que los diccionarios, no hay llaves repetidas.



Stacks y colas

- Ordenados linealmente como las listas, pero con reglas estrictas para la extracción de elementos
- Stacks: LIFO (implementados con list o dequeue)

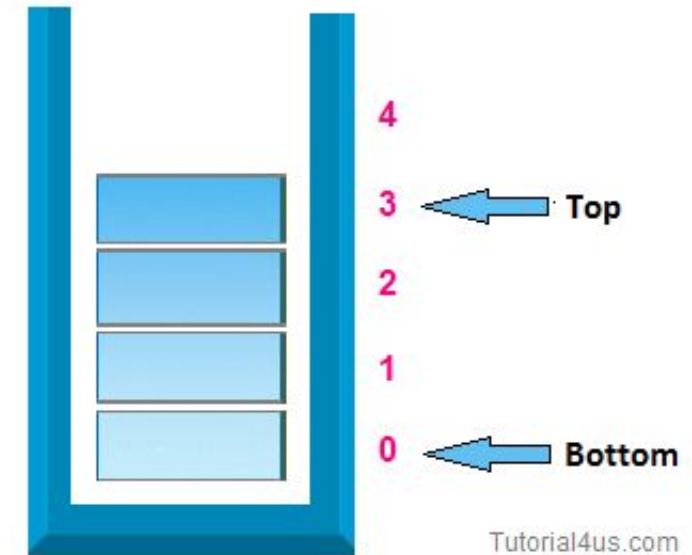
| | | | | | | |
|-----|-----|-----|-----|-----|------|------|
| 4 | 6 | 12 | 21 | ... | 1 | 1.2 |
| [0] | [1] | [2] | [3] | | [-2] | [-1] |



Stacks y colas

- Ordenados linealmente como las listas, pero con reglas estrictas para la extracción de elementos
- Stacks: LIFO (implementados con list o dequeue)

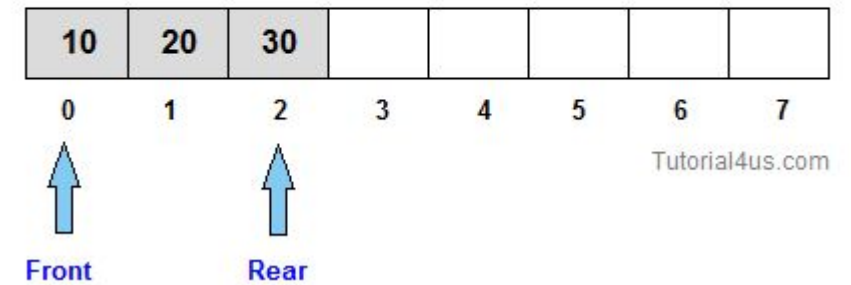
| | | | | | | |
|-----|-----|-----|-----|-----|------|------|
| 4 | 6 | 12 | 21 | ... | 1 | 1.2 |
| [0] | [1] | [2] | [3] | | [-2] | [-1] |



LIFO: Last IN, First OUT

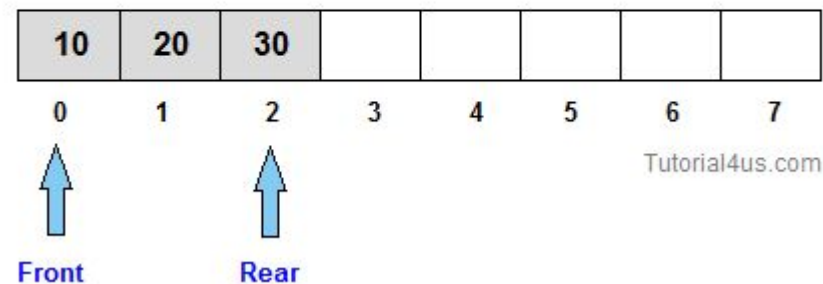
Stacks y colas

- Ordenados linealmente como las listas, pero con reglas estrictas para la extracción de elementos
- Colas: FIFO (implementadas con dequeue)



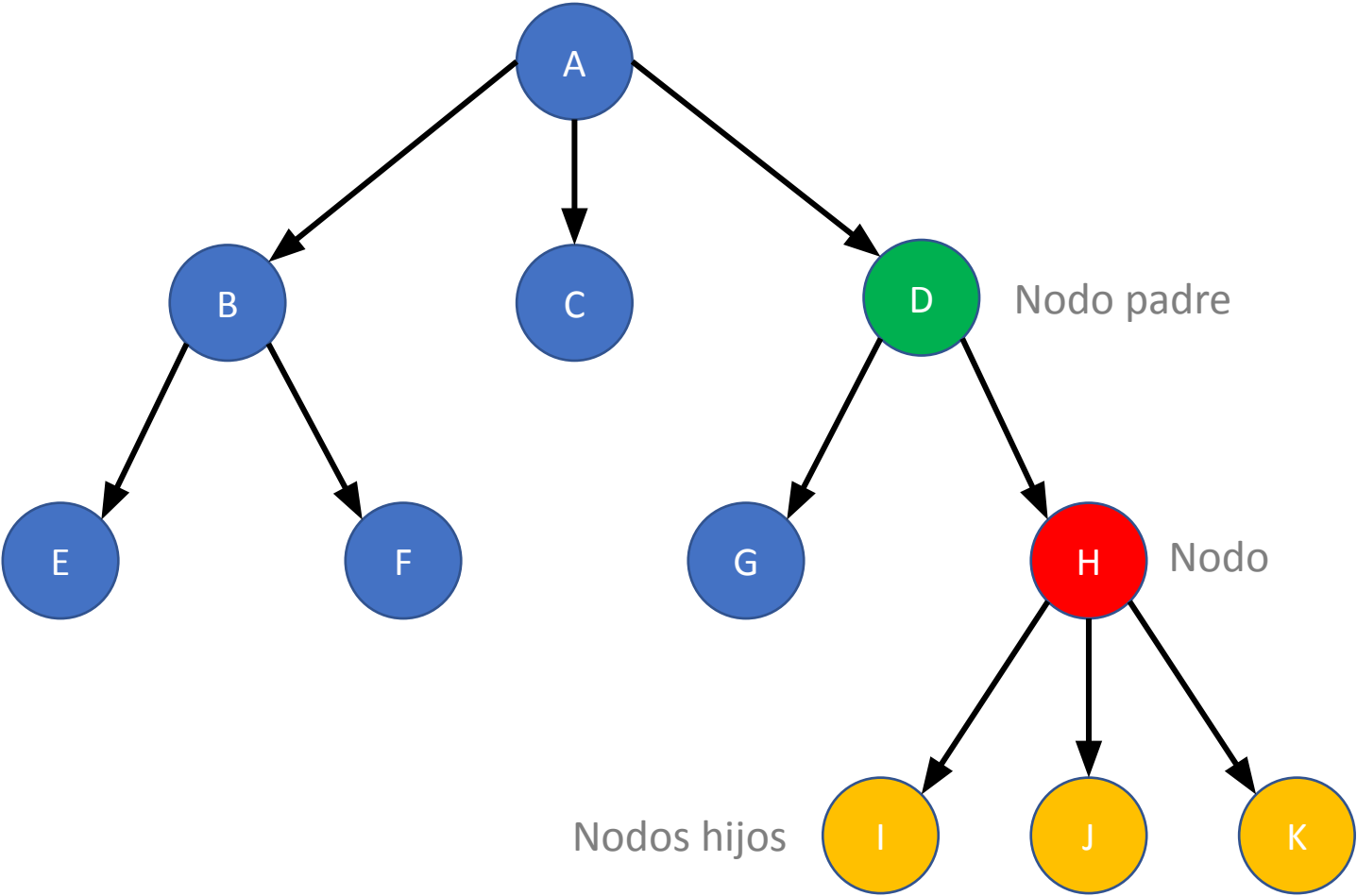
Stacks y colas

- Ordenados linealmente como las listas, pero con reglas estrictas para la extracción de elementos
- Colas: FIFO (implementadas con dequeue)

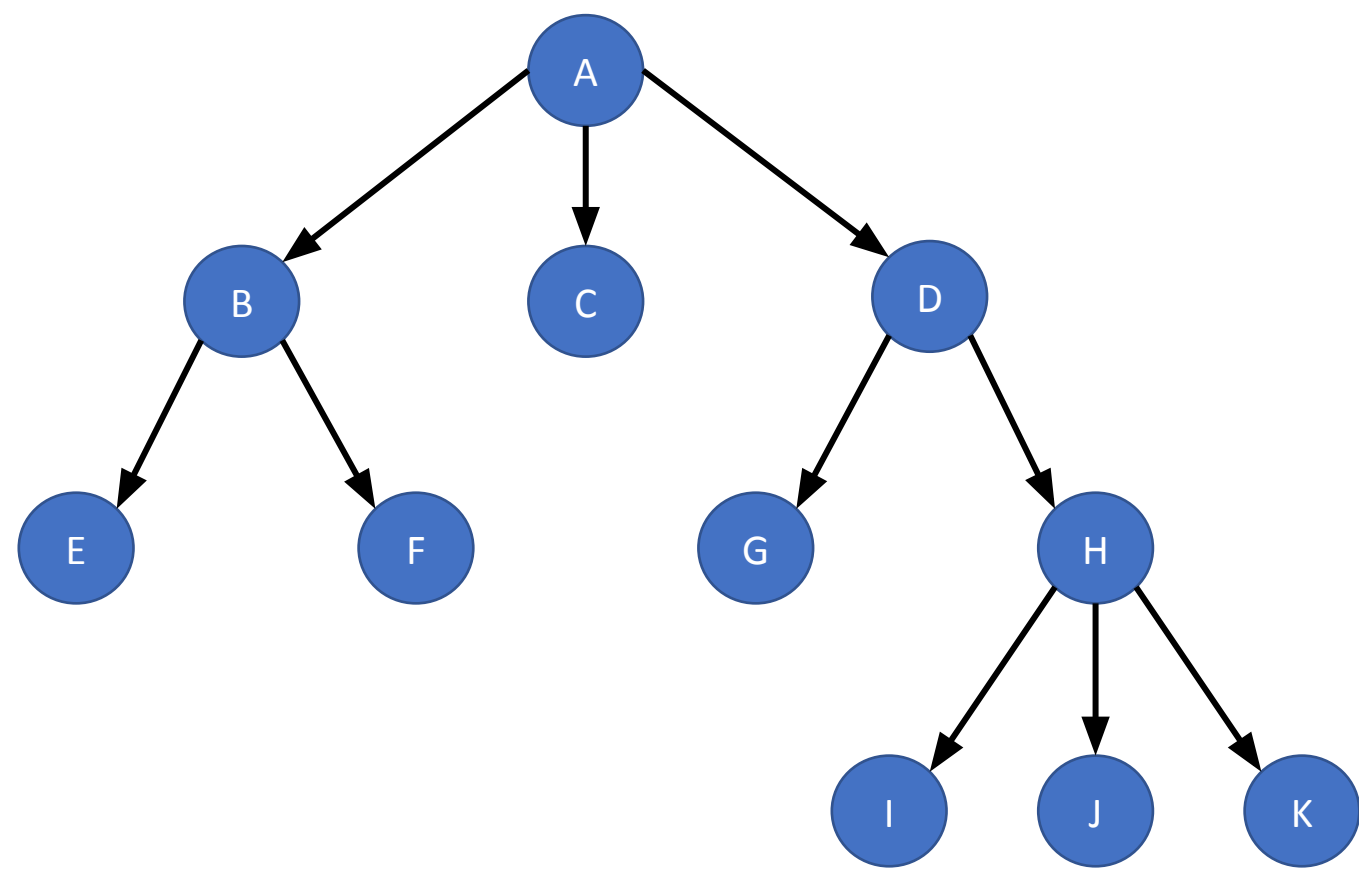


FIFO: First IN, First OUT

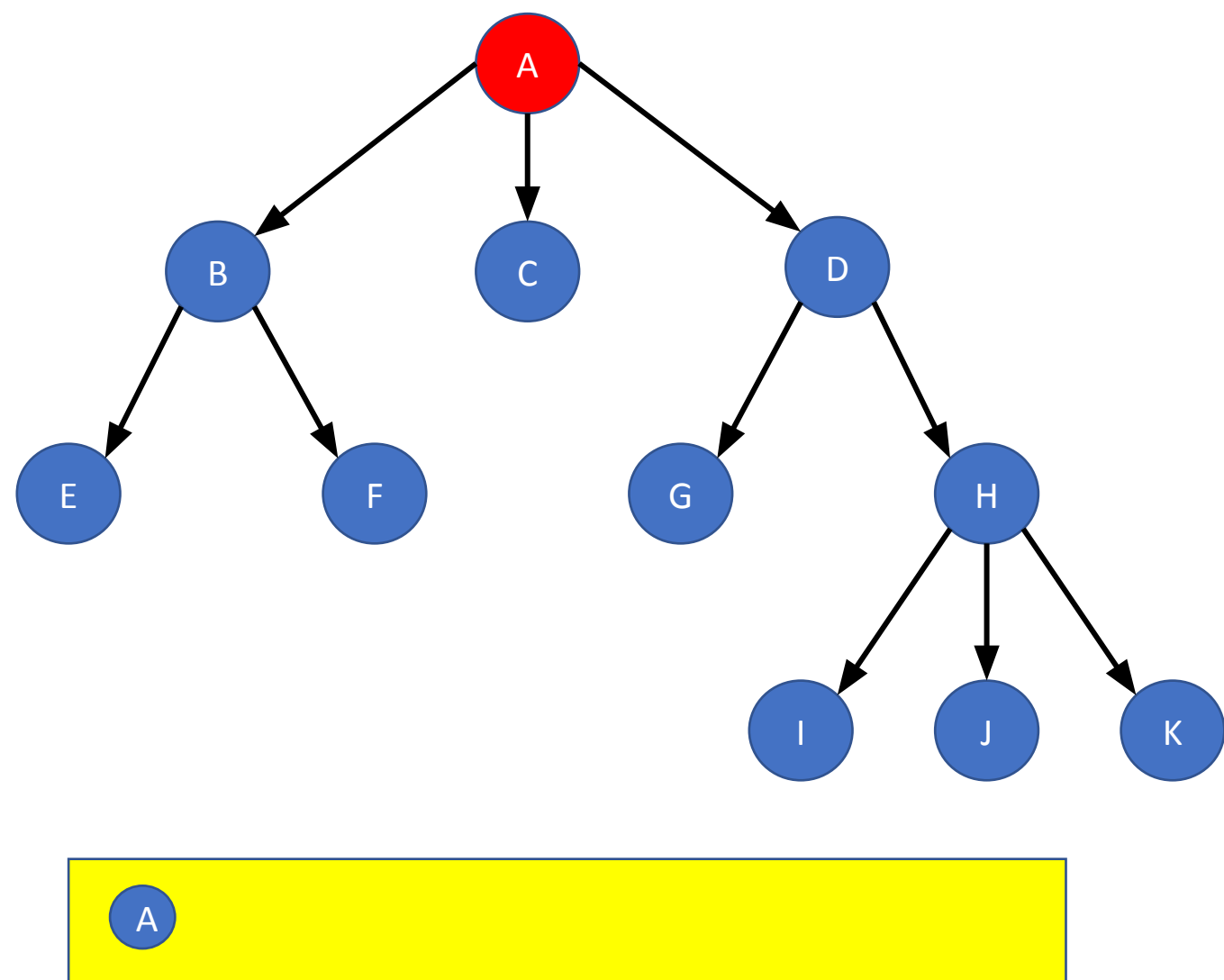
Árboles



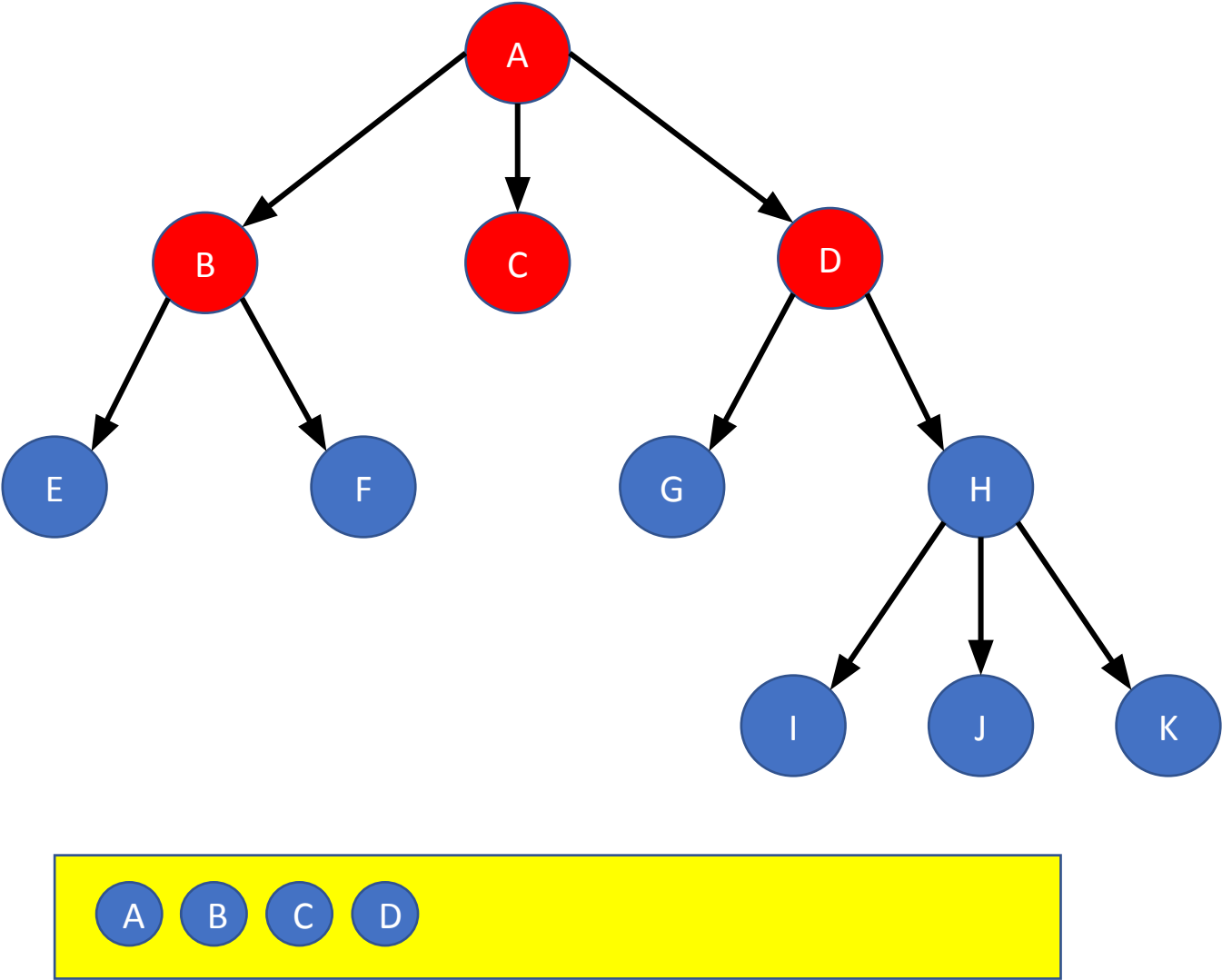
Recorriendo árboles – BFS (búsqueda en amplitud)



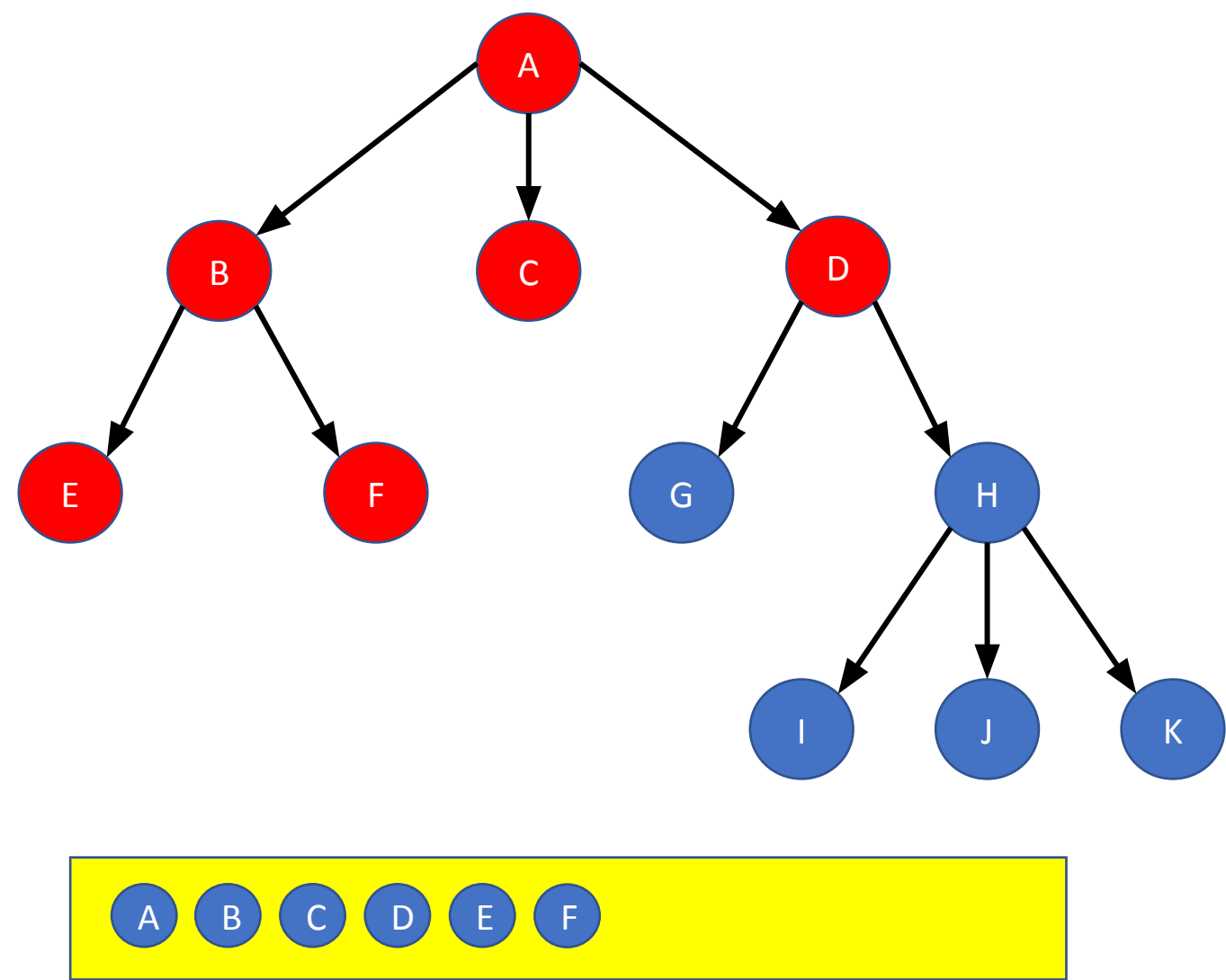
Recorriendo árboles – BFS (búsqueda en amplitud)



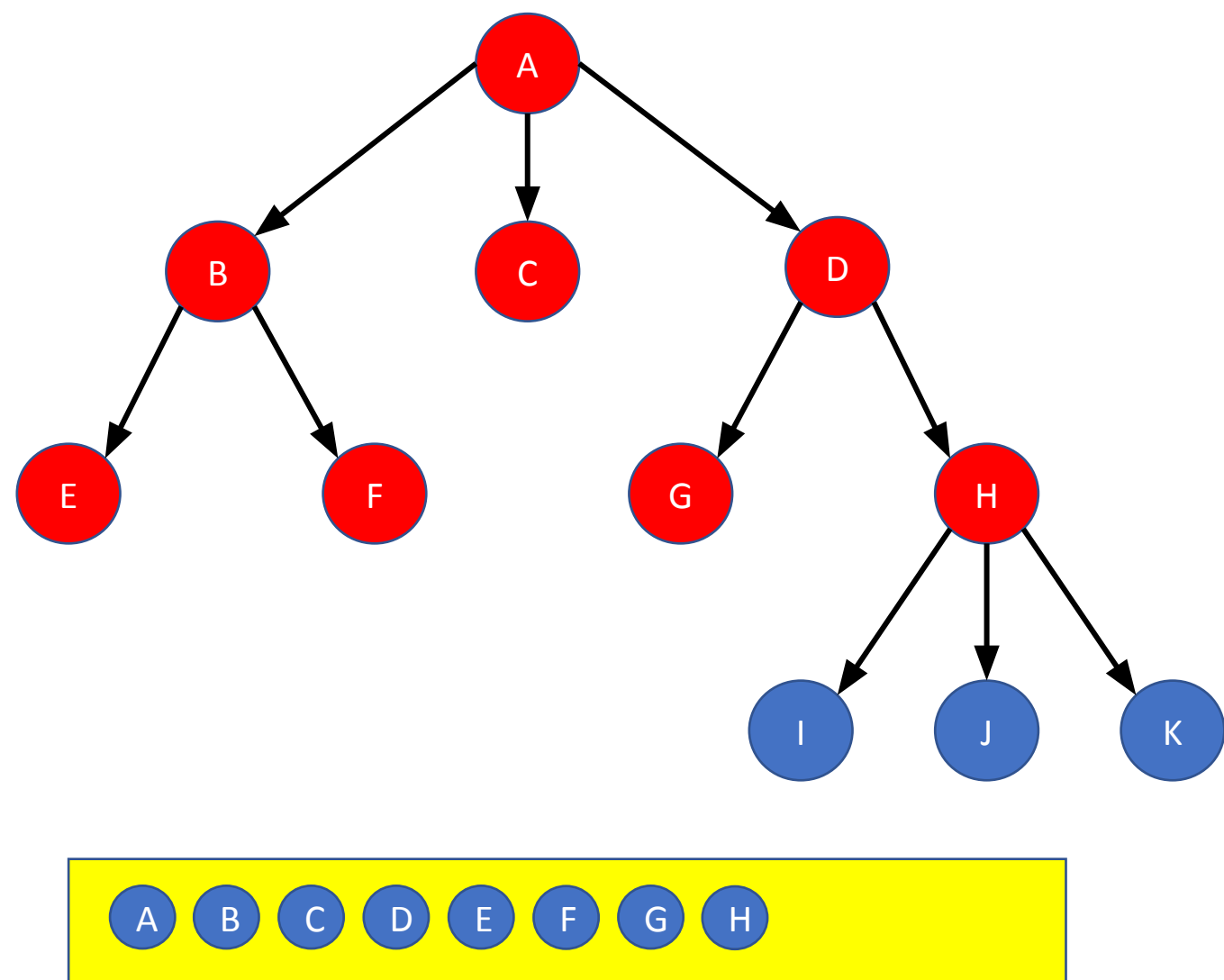
Recorriendo árboles – BFS (búsqueda en amplitud)



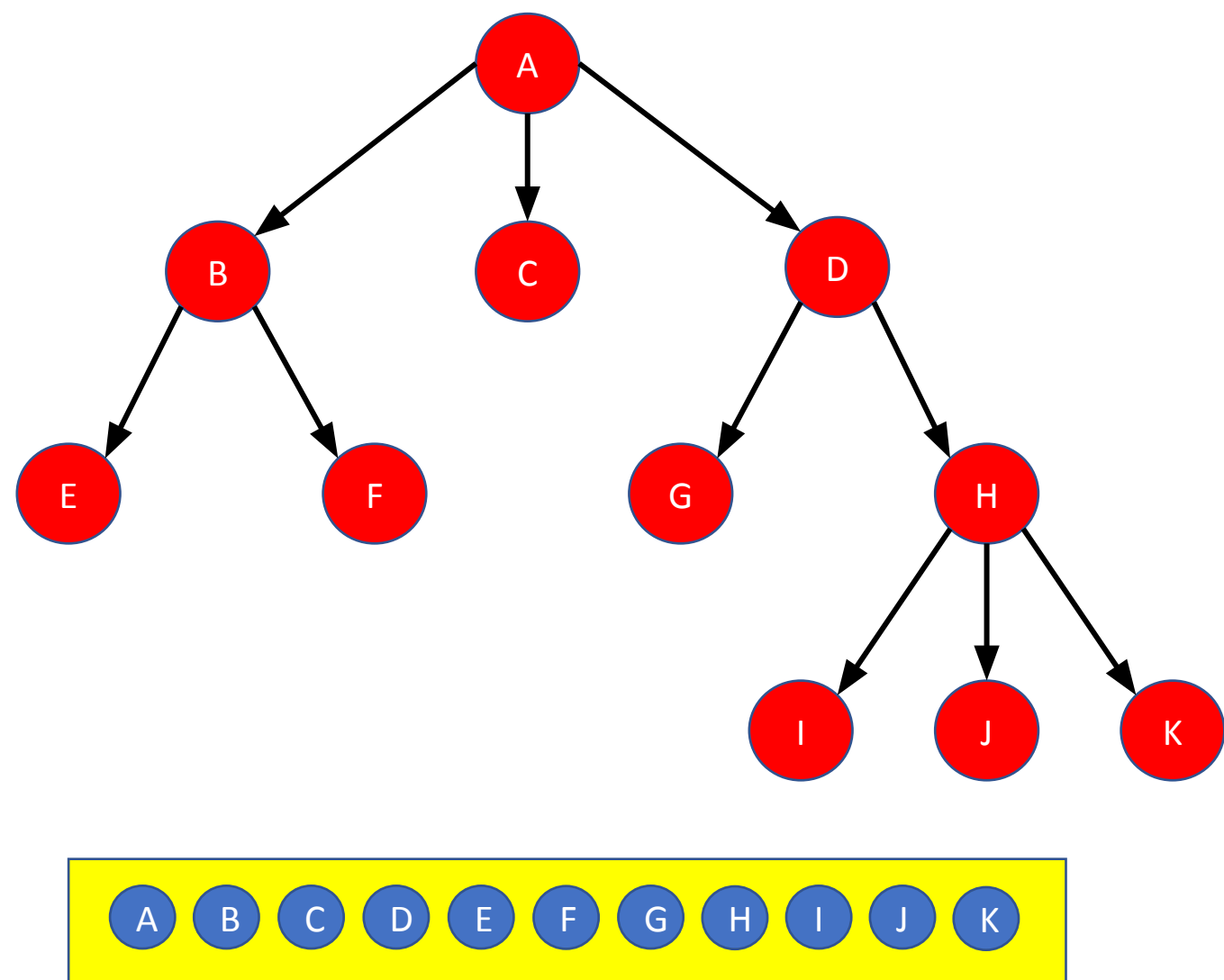
Recorriendo árboles – BFS (búsqueda en amplitud)



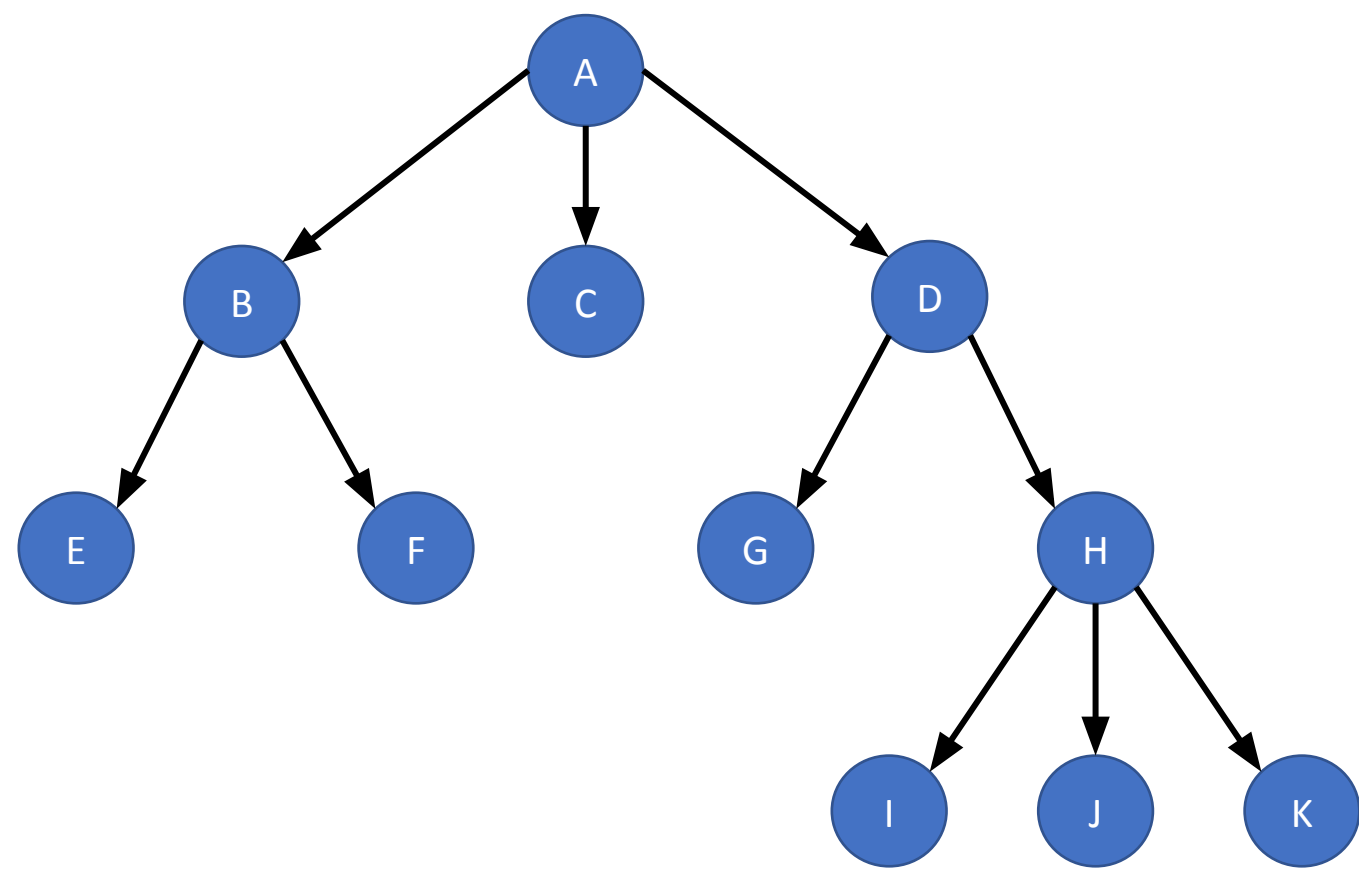
Recorriendo árboles – BFS (búsqueda en amplitud)



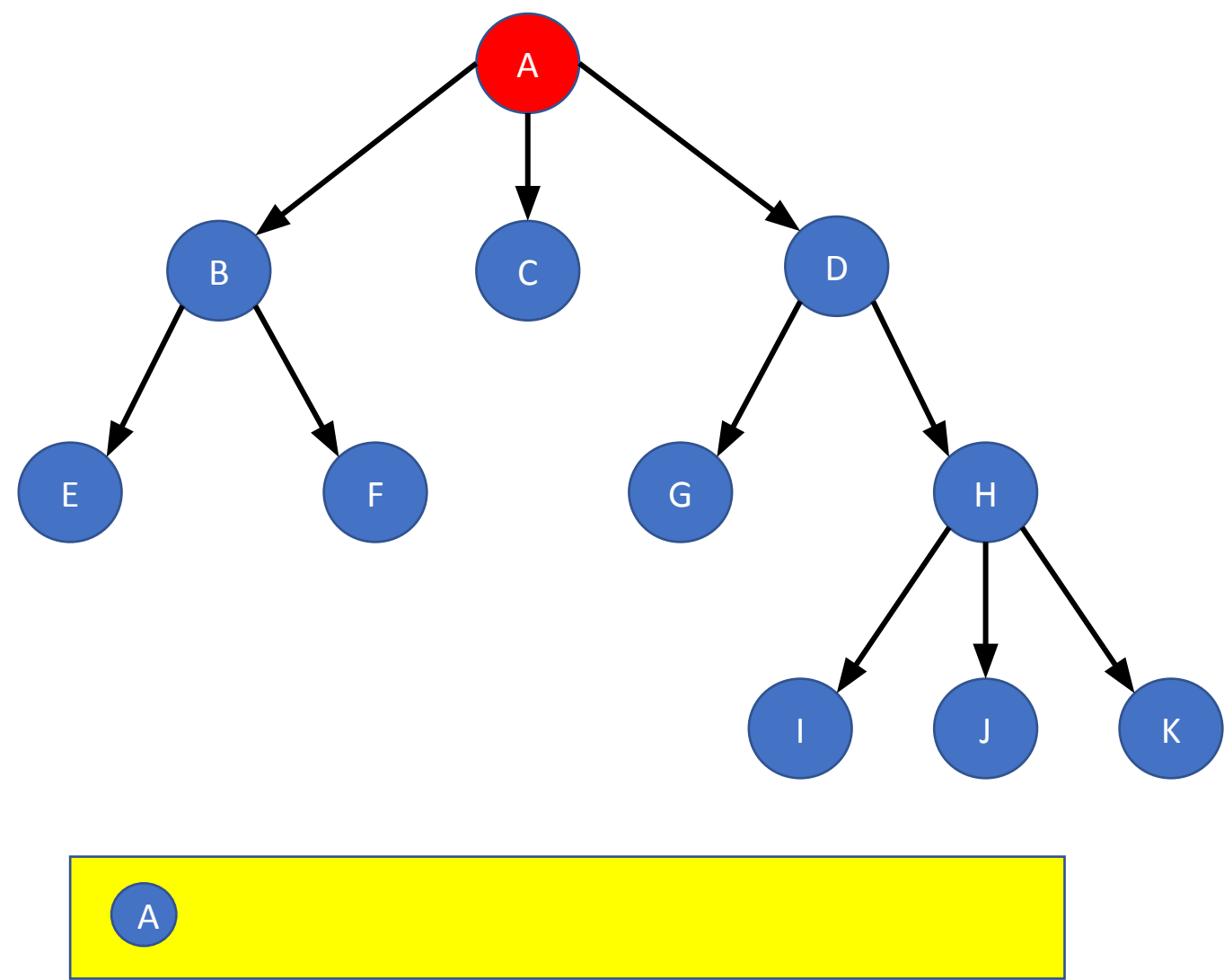
Recorriendo árboles – BFS (búsqueda en amplitud)



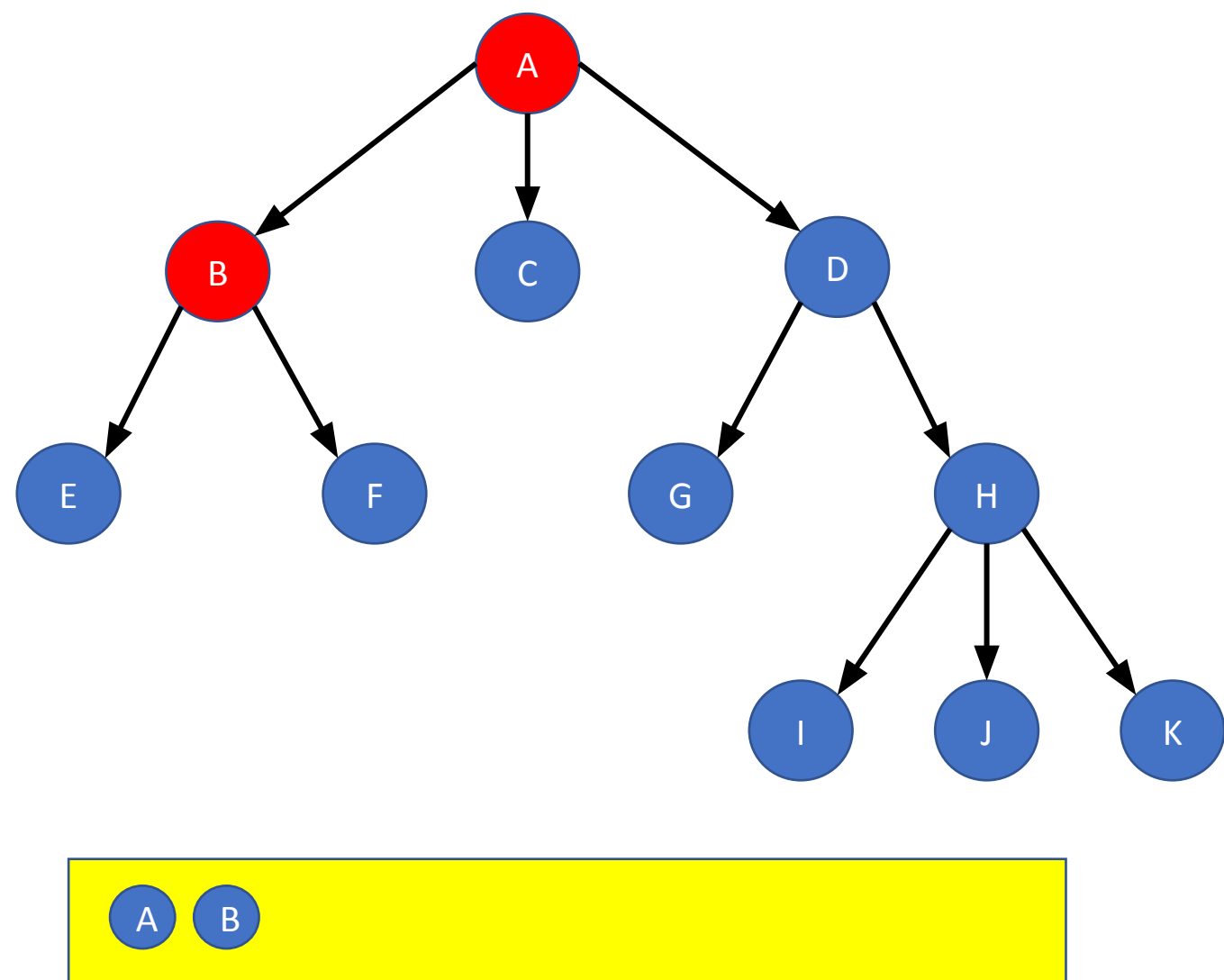
Recorriendo árboles – DFS (búsqueda en profundidad)



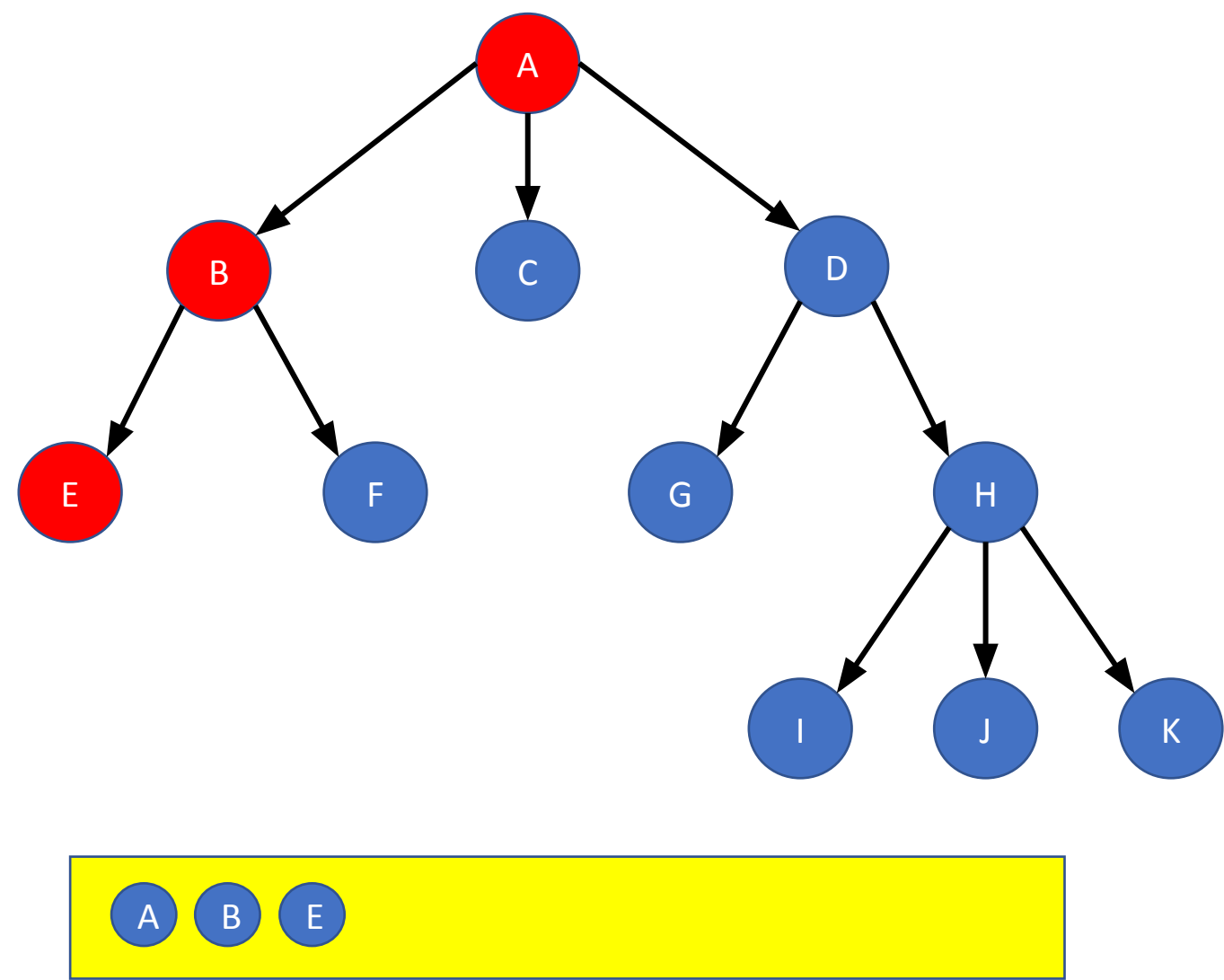
Recorriendo árboles – DFS (búsqueda en profundidad)



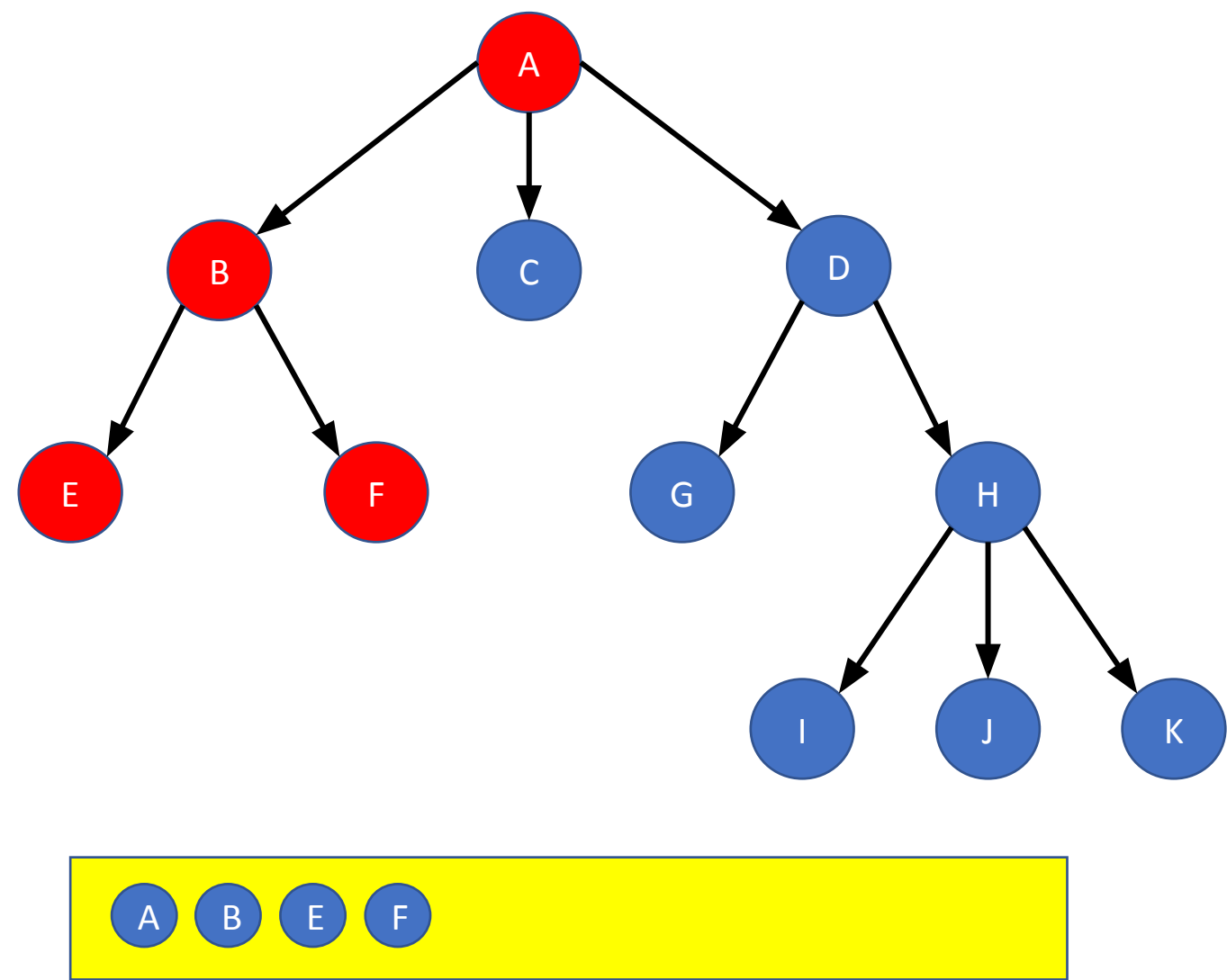
Recorriendo árboles – DFS (búsqueda en profundidad)



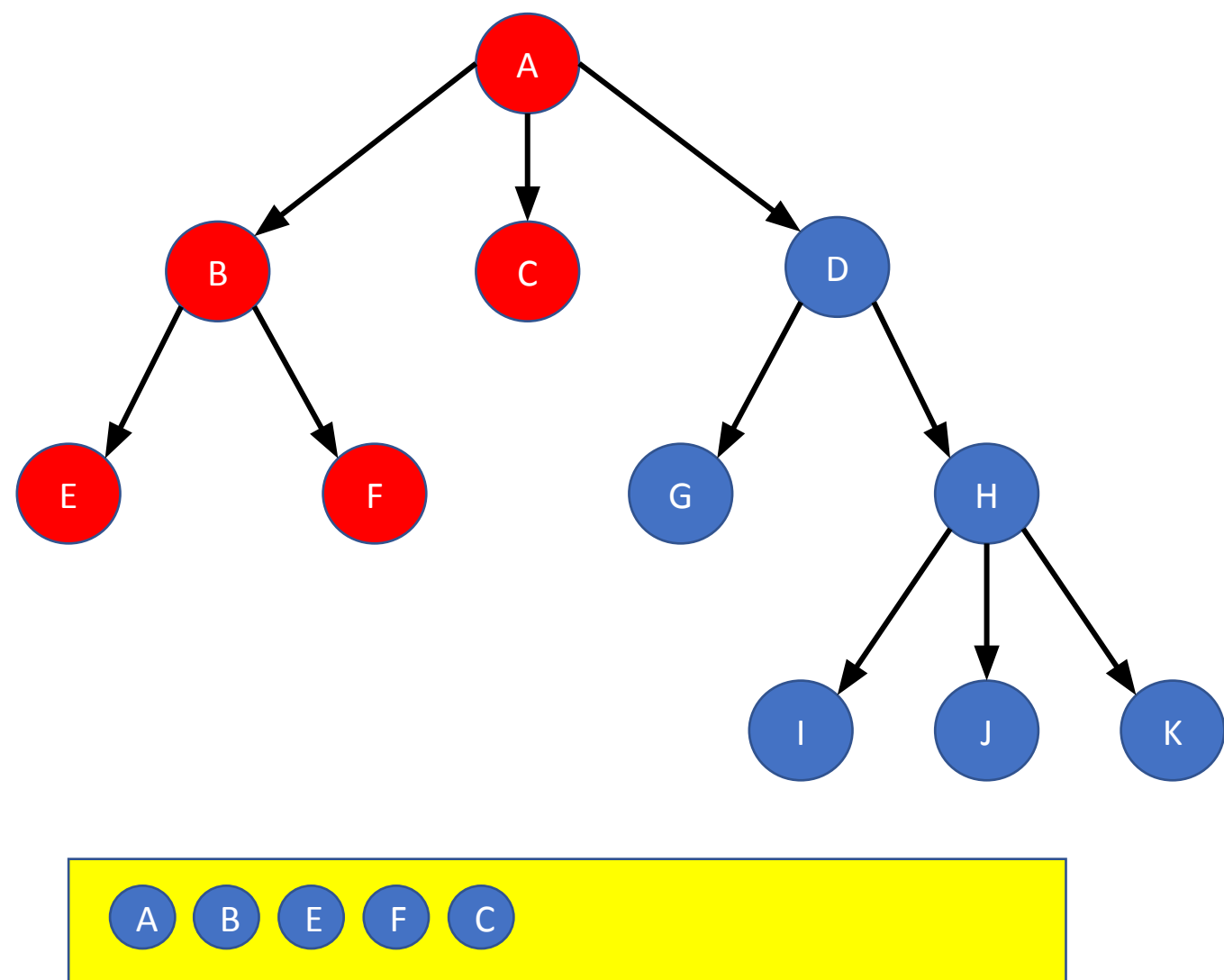
Recorriendo árboles – DFS (búsqueda en profundidad)



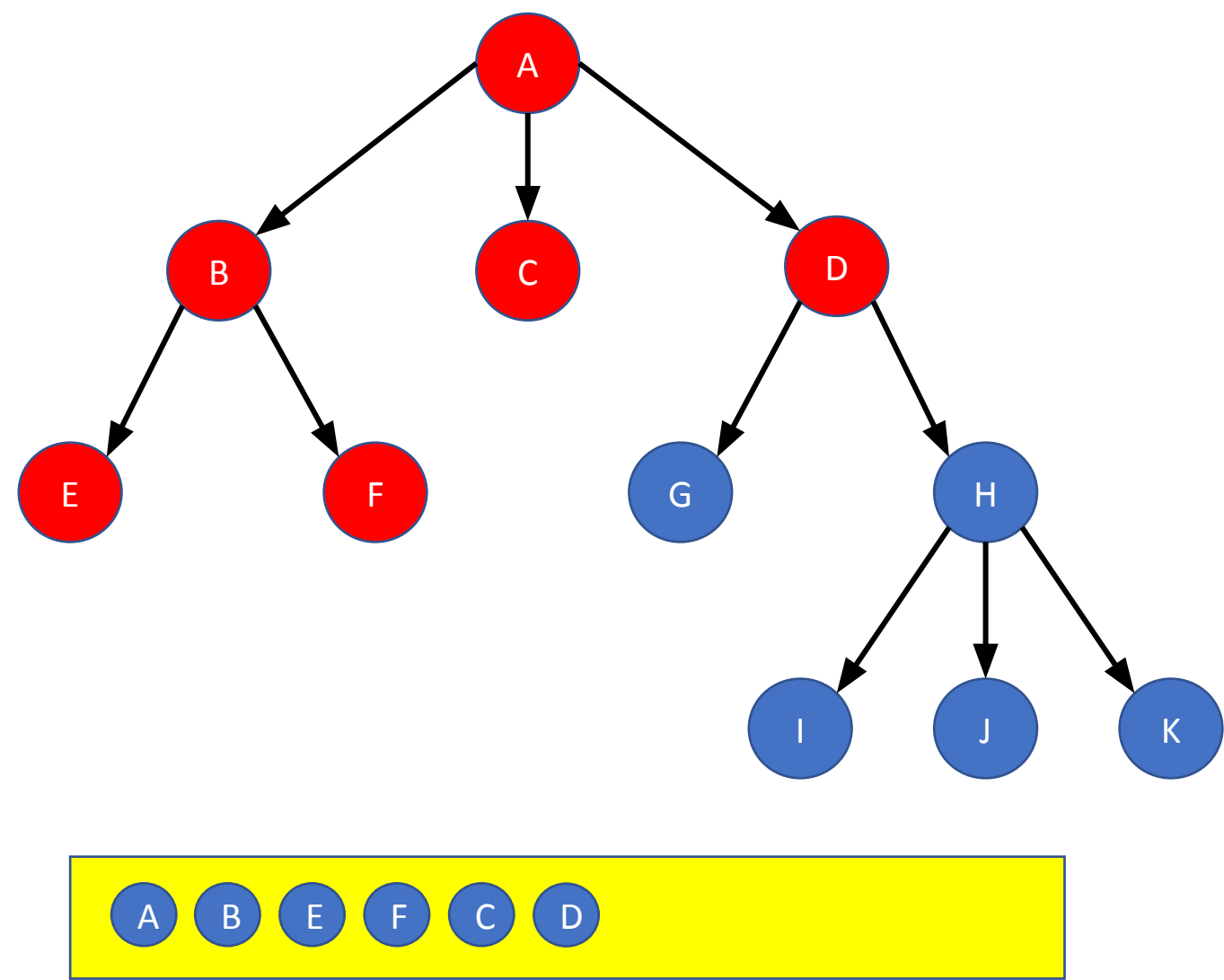
Recorriendo árboles – DFS (búsqueda en profundidad)



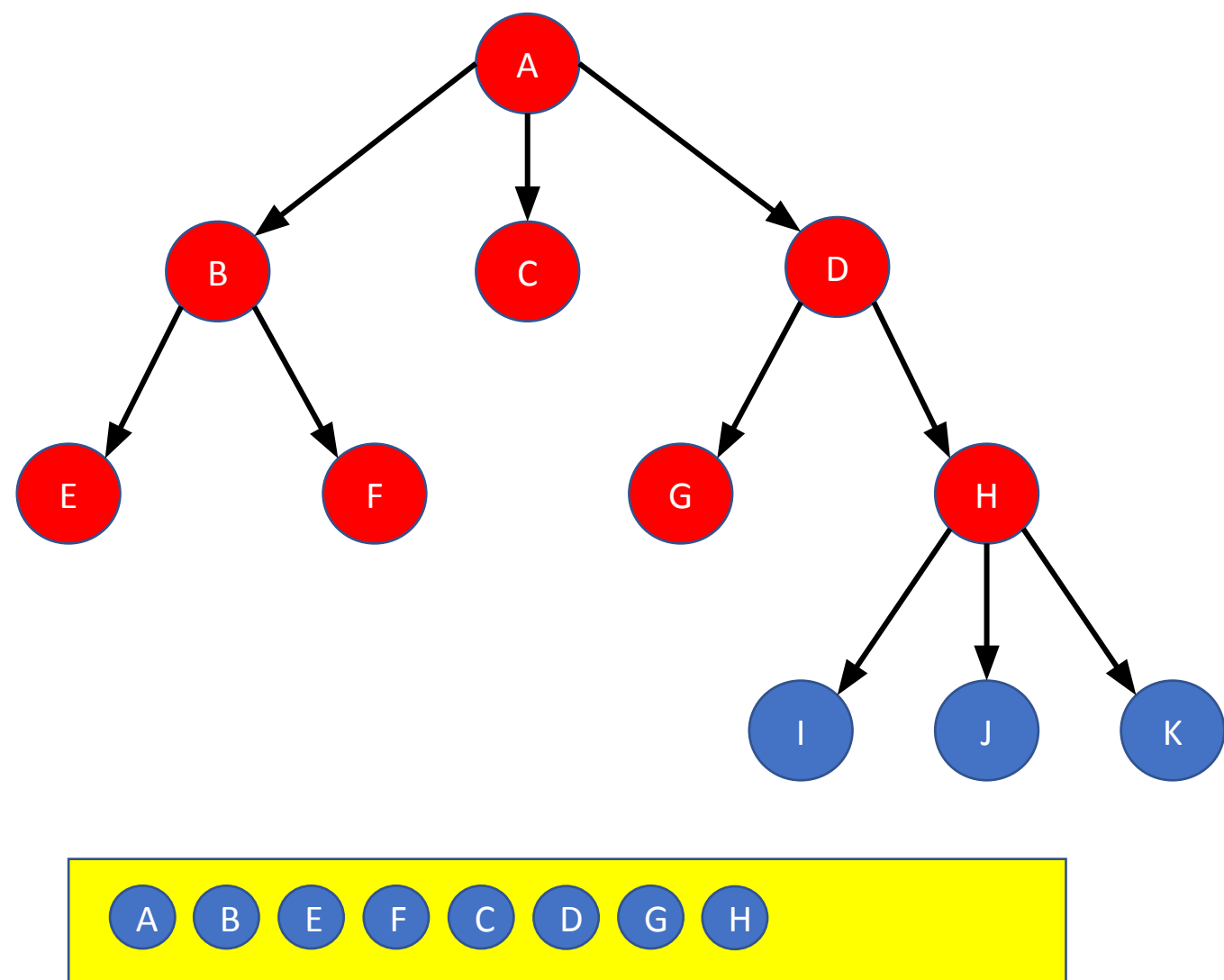
Recorriendo árboles – DFS (búsqueda en profundidad)



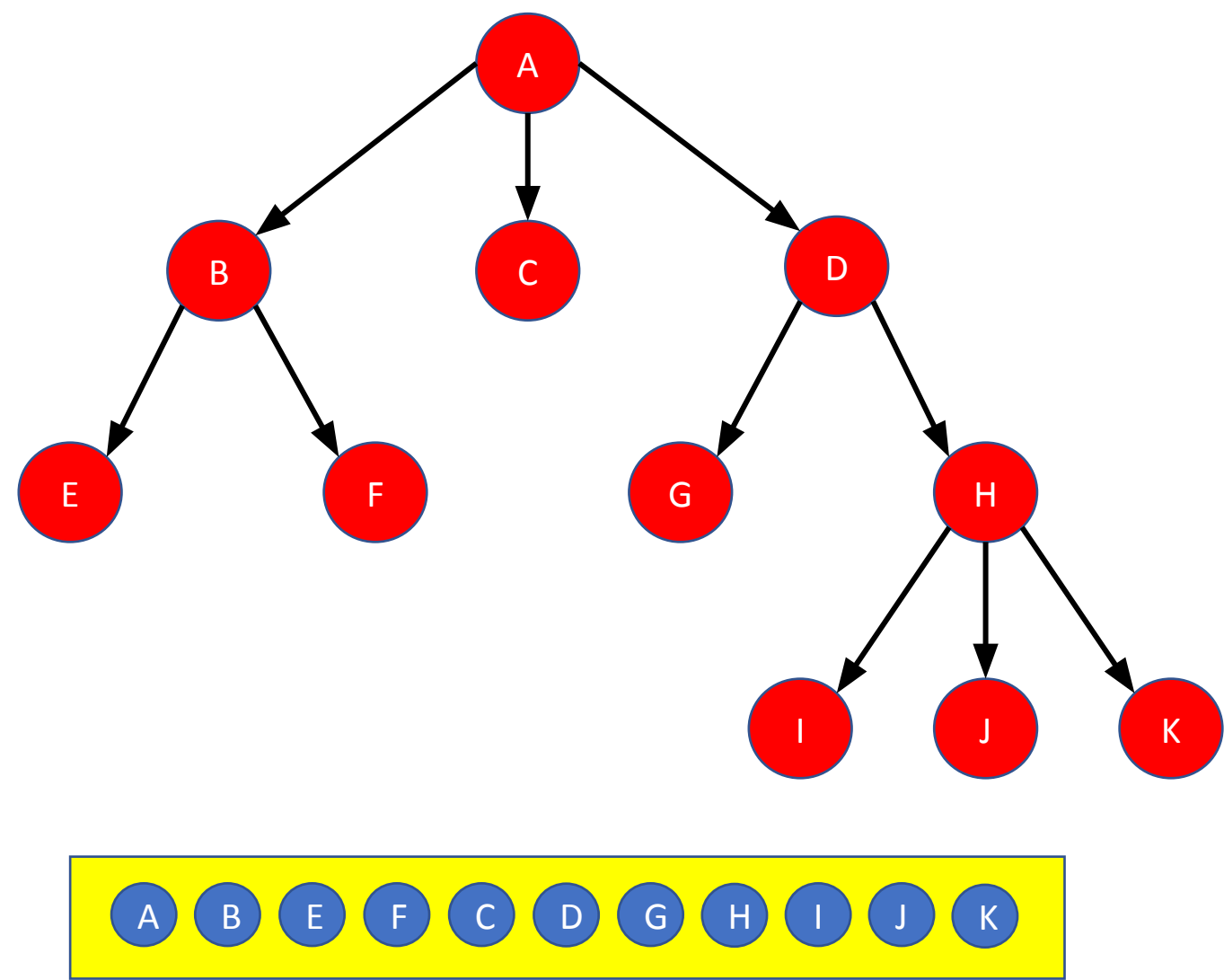
Recorriendo árboles – DFS (búsqueda en profundidad)



Recorriendo árboles – DFS (búsqueda en profundidad)



Recorriendo árboles – DFS (búsqueda en profundidad)



Un breve y somero resumen

- Las estructuras de datos corresponden a un tipo de dato especializado, **diseñado para agrupar, almacenar o acceder a la información de manera más eficiente** que un tipo de dato básico.
- La elección adecuada de la estructura de datos es fundamental para el desarrollo de un buen programa y muchas veces es la única posibilidad para solucionar un problema de forma realista.
- Pero siempre es conveniente pensar primero en una solución básica a los problemas, y luego incorporar las estructuras donde corresponda.

Un ejemplo práctico

“Dado un string que utiliza los parentesis:

() [] { }

determine si se encuentra balanceado o no”

1. Leer y releer el enunciado
2. Darnos ejemplo sencillos para entender la mecánica
3. Pensar formas de modelar y abordar el problema
4. Programo
5. ¿Puedo mejorarlo?
6. Sigo programando

*Si recibieramos el texto '()({[]})' debiesemos retornar **True**,
mientras que con '([])' o '({})' **False***

| | | | | | |
|---|---|---|---|---|---|
| (| [|] |) | { | } |
|---|---|---|---|---|---|

([]) { }

Stack

| | | | | | |
|---|---|---|---|---|---|
| (| [|] |) | { | } |
|---|---|---|---|---|---|

Stack

([]) { }

(

Stack

([]) { }

(

Stack

([]) { }

[

(

Stack

([]) { }

[

(

Stack

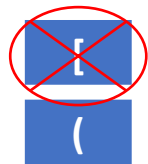
([]) { }

[
(

Stack

Si el stack estuviese vacío, el retorno sería False

([]) { }



Stack

Si el tope del stack corresponde al opuesto del que estamos analizando, lo sacamos y seguimos revisando, en caso contrario retornamos False

([]) { }

(

Stack

([]) { }



Stack

([]) { }

Stack

([]) { }

{

Stack

([]) { }

{

Stack

([]) { }



Stack

([]) { }

Stack

Seguimos iterando y si al finalizar el proceso el stack está vacío, retornamos True, en caso contrario, False

Cómo sigue la sesión de hoy

- Lectura del enunciado del ejercicio
- Control (15:15 a 15:30)
- Trabajo personal o grupal (15:30 a 16:30)
- Entrega del avance (16:40 a 16:50)

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación



IIC2115 - Programación como Herramienta para la Ingeniería

Estructuras de datos

Profesora: Francesca Lucchini
Prof. Coordinador: Hans Löbel

Algunas cosas antes de empezar el taller

- El objetivo parcial es el **desde** que consideramos para la materia. Así, sacarse un **7,0 no implica**, estar listo con el estudio.
- Por el contrario, **(casi) completar el taller** (ambos ejercicios, en algo más de un módulo) **sí es un indicador** de que el estudio está bien encaminado.
- Evaluaciones que no se entreguen por Github no serán corregidas.
- Evaluaciones que no respeten el formato de entrega (nombre y ubicación de carpeta y archivos) serán penalizadas.

Funciones

Nos permiten reutilizar partes de código tantas veces como queramos y generalizar su estructura.



Funciones

- Las funciones no necesariamente retornan un valor, pueden solo realizar cambios o actualizaciones sobre un parámetro (o en el caso de un método, de los atributos de un objeto).
- Los parámetros de entrada tampoco son obligatorios. Se pueden crear funciones que no reciban parámetros.

Listas en Python

```
l1 = [3,5,63,23]  
l2 = list()  
l1[1] # 5  
l1.append(4) #[3,5,63,23,4]  
l1.pop(2) #[3,5,23,4]
```

Tuplas en Python

```
t1 = (3,5,63,23)
t2 = tuple()
t1[1] # 5
t1.append(4) #error
t1.pop(2) #error
```

Diccionarios en Python

```
d1 = {'nom':'Jen', 'edad':63}  
d2 = dict()  
d1['nom'] # 'Jen'  
d1.update('edad': 53) #error
```


Sets

- Dictionarios: {}

