Concurso ED y Grafos 2025/2026

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - Urgencias	Colas de prioridad
B - Óscar y los trenes	BFS/DFS
C - La Lista de la Compra	Ad-hoc, contar
D - CTF	BFS, Dijkstra
E - Enseñando a sumar	Conjuntos
F - La Ladrona de Libros	Cola, pila, mapa

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Urgencias	10	125 MB
B - Óscar y los trenes	8	1.8 MB
C - La Lista de la Compra	5	1.5 MB
D - CTF	10	20 MB
E - Enseñando a sumar	4	250 KB
F - La Ladrona de Libros	6	73 KB
- **Total**	43	(+-) 150MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Urgencias	HeapHunters	4
B - Óscar y los trenes	HeapHunters	15
C - La Lista de la Compra	a.izquierdor.2025	7
D - CTF	?	?
E - Enseñando a sumar	?	?
F - La Ladrona de Libros	?	?

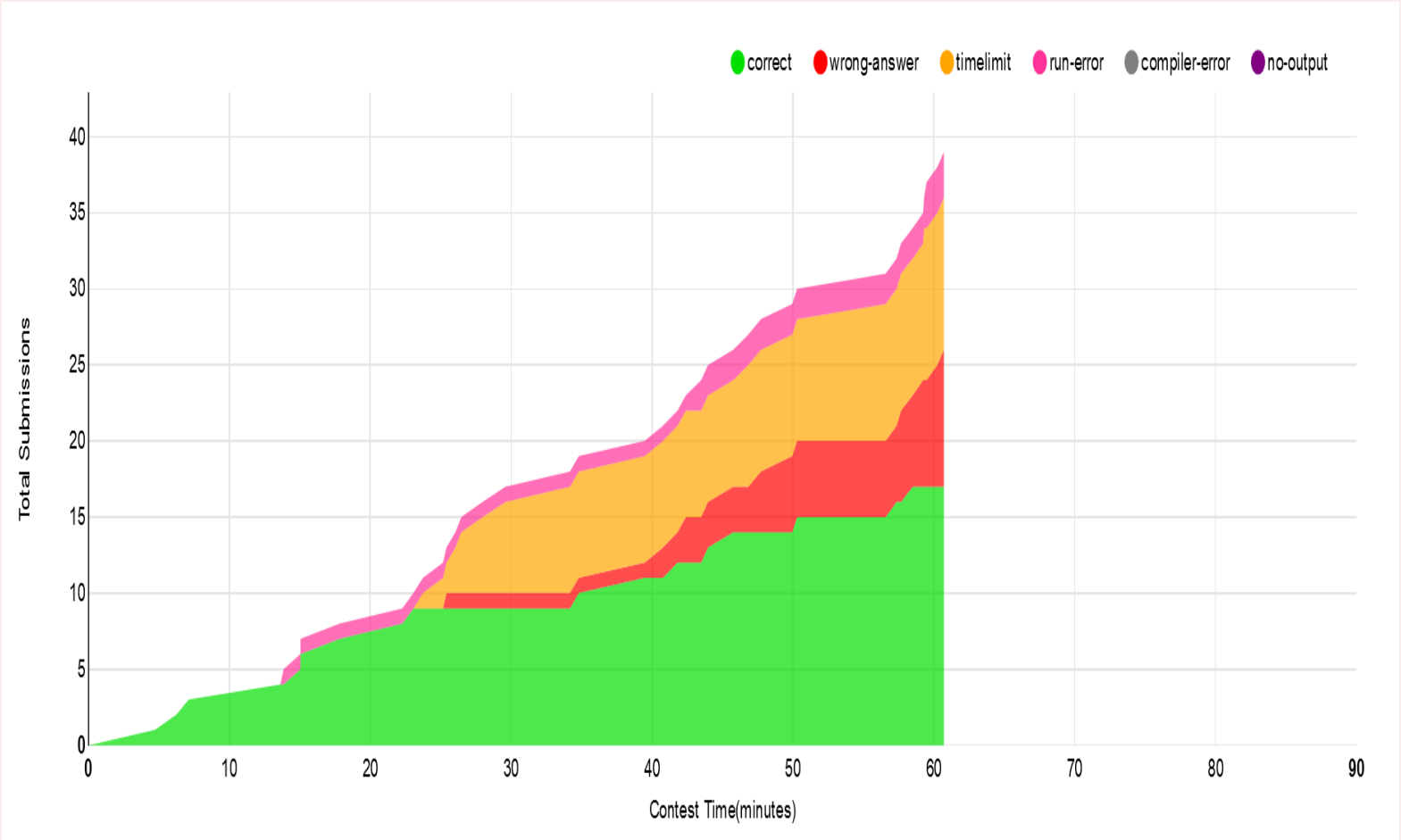
*Antes de congelar el marcador.

Estadísticas*

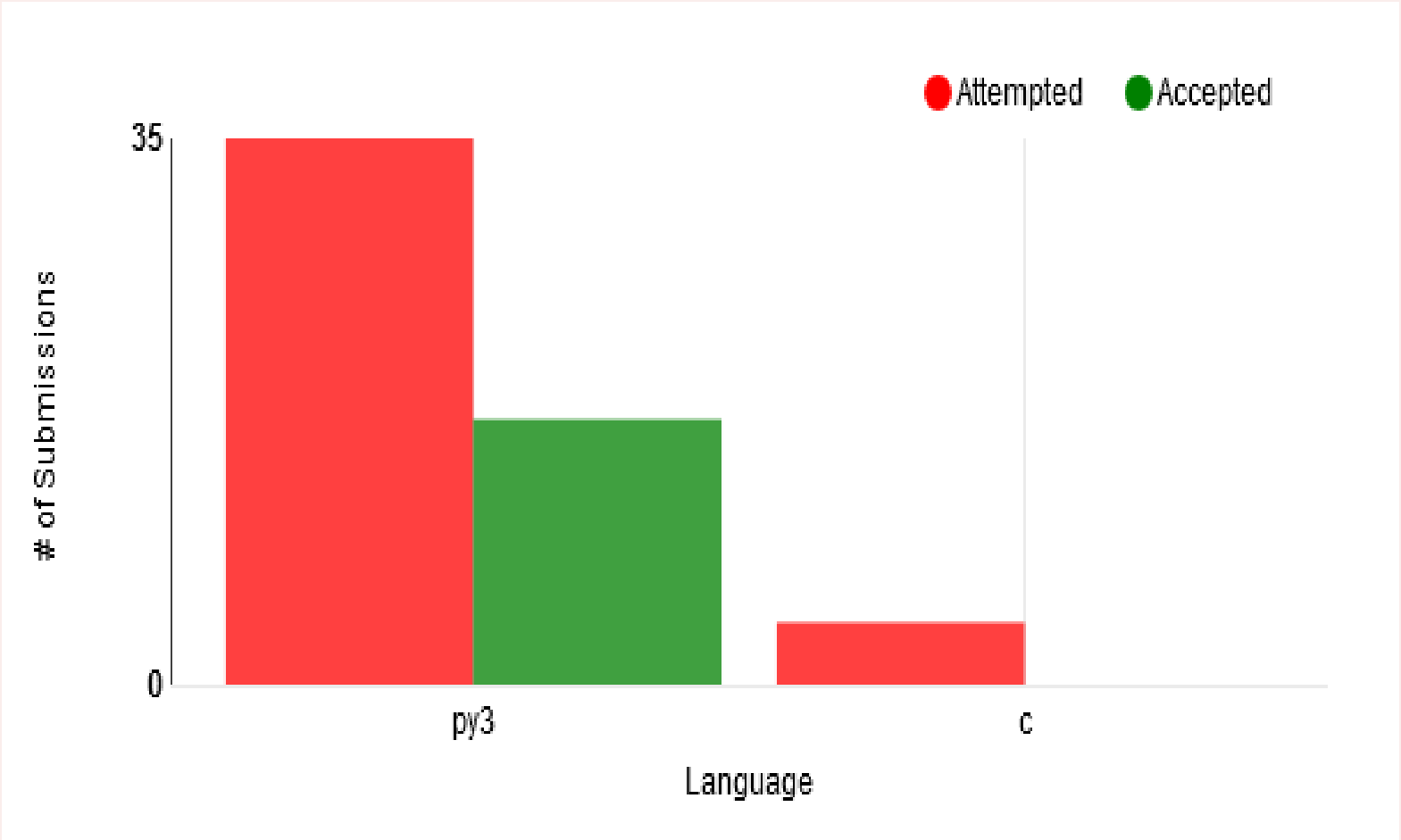
Problema	Válidos	Envíos	% éxito
A - Urgencias	6	19	32 %
B - Óscar y los trenes	2	3	66 %
C - La Lista de la Compra	9	15	60 %
D - CTF	0	0	0 %
E - Enseñando a sumar	0	1	0 %
F - La Ladrona de Libros	0	1	0 %

*Antes de congelar el marcador.

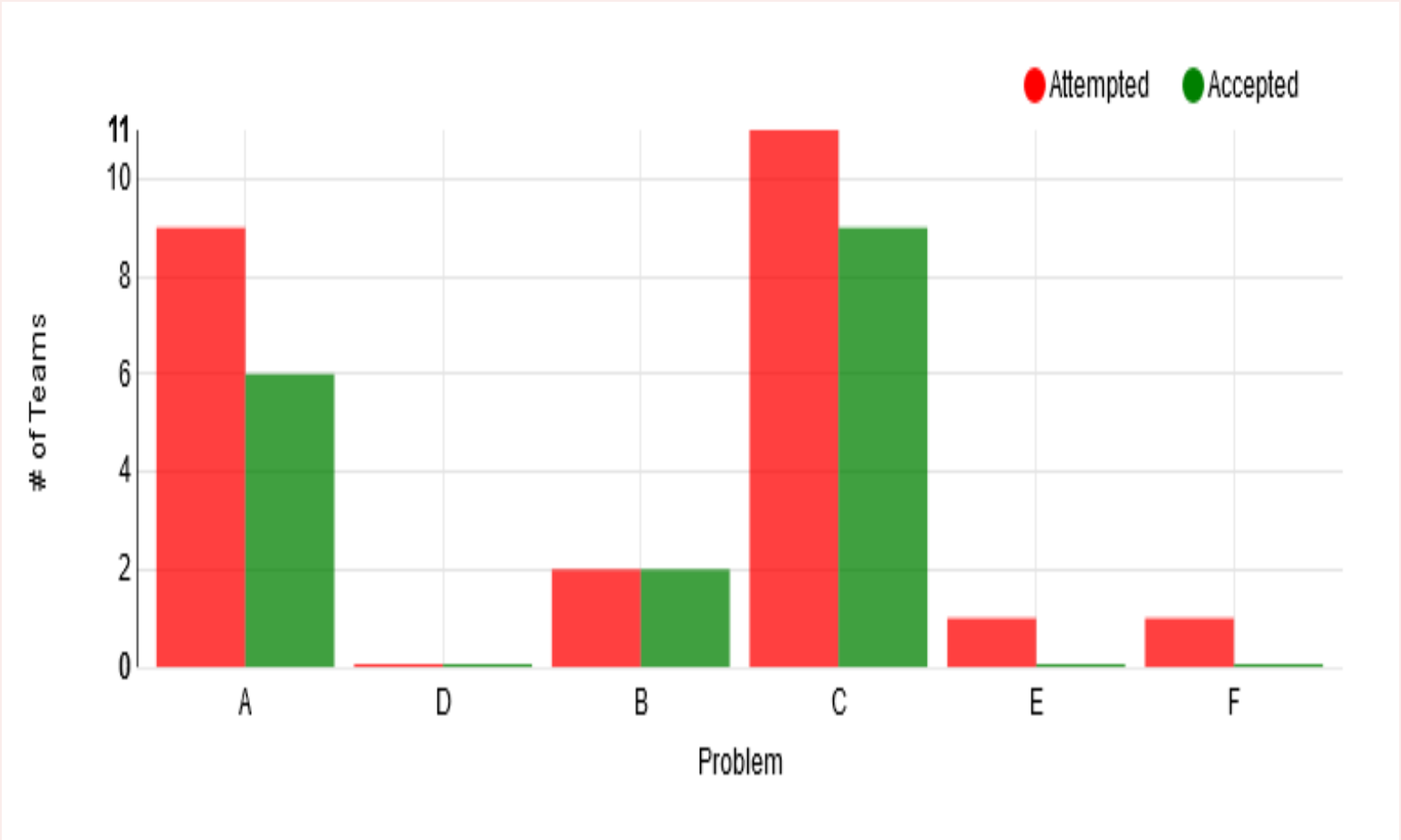
Estadísticas varias



Estadísticas varias



Estadísticas varias



A. Urgencias

Envíos	Válidos	% éxito
19	6	32 %

A. Urgencias

En las urgencias de un hospital:

- Llegan pacientes con un nivel de malestar.
- Se atiende a los pacientes de mayor a menor nivel de malestar.

A. Urgencias

En las urgencias de un hospital:

- Llegan pacientes con un nivel de malestar.
- Se atiende a los pacientes de mayor a menor nivel de malestar.

Los pacientes que esperan forman una **cola** para ser atendidos.

A. Urgencias

En las urgencias de un hospital:

- Llegan pacientes $\rightarrow$ Se añade el paciente a la cola.
- Se atiende a los pacientes $\rightarrow$ Se sacan pacientes de la cola.

¿Cómo hacerlo de forma eficiente?

A. Urgencias

En las urgencias de un hospital:

- Llegan pacientes $\rightarrow$ Se añade el paciente a la cola.
- Se atiende a los pacientes $\rightarrow$ Se sacan pacientes de la cola.

¿Cómo hacerlo de forma eficiente? $\rightarrow$ Pacientes en una **cola de prioridad**.

A. Urgencias

ATENCIÓN: Se atiende a los pacientes de **mayor a menor** nivel de malestar. Es decir, en el orden inverso al que tienen las colas por defecto (ascendente).

A. Urgencias

```
leer E
cola = []
for _ in range(E):
    leer pi, ni
    if pi == "PACIENTE":
        heapq.heappush(cola, -ni)
    else:
        for _ in range(ni):
            print(-cola[0])
            heapq.heappop(cola)
```

B. Óscar y los trenes

Envíos	Válidos	% éxito
2	3	66 %

B. Óscar y los trenes

Tras el temporal la red ferroviaria ha quedado dañada:

- Tenemos zonas desconectadas.
- Queremos **minimizar** las vías necesarias para arreglarlo.

¿Cómo hacerlo de forma eficiente?

B. Óscar y los trenes

- Hay que contar las **zonas conexas** de la red.
- Para identificar esas **componentes conexas** $\rightarrow$ **BFS/DFS**.
- El **número mínimo de vías** vendrá dado por el **número de componentes conexas - 1.**

B. Óscar y los trenes

```
if __name__ == '__main__':
    n, m = map(int, input().split())

    graph = [[] for _ in range(n + 1)]

    for _ in range(m):
        s,t = map(int, input().split())
        graph[s].append(t)
        graph[t].append(s)

    visited = [False] * len(graph)
    sol = -1
    for i in range(1, n + 1):
        if not visited[i]:
            sol += 1
            #DFS(graph, i, visited)
            BFS(graph, i, visited)
    print(sol)
```

B. Óscar y los trenes

```
def BFS(graph, start, visited):
    cola = deque()
    cola.append(start)
    while len(cola) > 0:
        node = cola.popleft()
        for aux in graph[node]:
            if not visited[aux]:
                visited[aux] = True
                cola.append(aux)
```

```
def DFS(graph, start, visited):
    visited[start] = True
    for node in graph[start]:
        if not visited[node]:
            DFS(graph,node,visited)
```

C. La lista de la compra

Envíos	Válidos	% éxito
9	15	60 %

C. La lista de la compra

David y Eva no se aclaran haciendo la compra:

- David tiene una lista de la compra con lo que hace falta.
- Eva compra lo que cree que falta en casa sin ninguna planificación.

Quieren contar cuántas cosas tenían que comprar dada la lista de la compra.

C. La lista de la compra

- La entrada nos da un número N de productos.
- Seguidos aparecen los N productos y un número Q_i indicando la cantidad.

C. La lista de la compra

- La entrada nos da un número N de productos.
- Seguidos aparecen los N productos y un número Q_i indicando la cantidad.

- Debemos sumar la cantidad de cada producto.
- El nombre de los productos nos dan igual! Solo nos interesan las cantidades!

C. La lista de la compra

```
N = leerEntero()
cantidad_total = 0
hasta N:
    palabras = leerPalabras()
    cantidad_total += entero(palabras[1])
imprimir(cantidad_total)
```

D. CTF

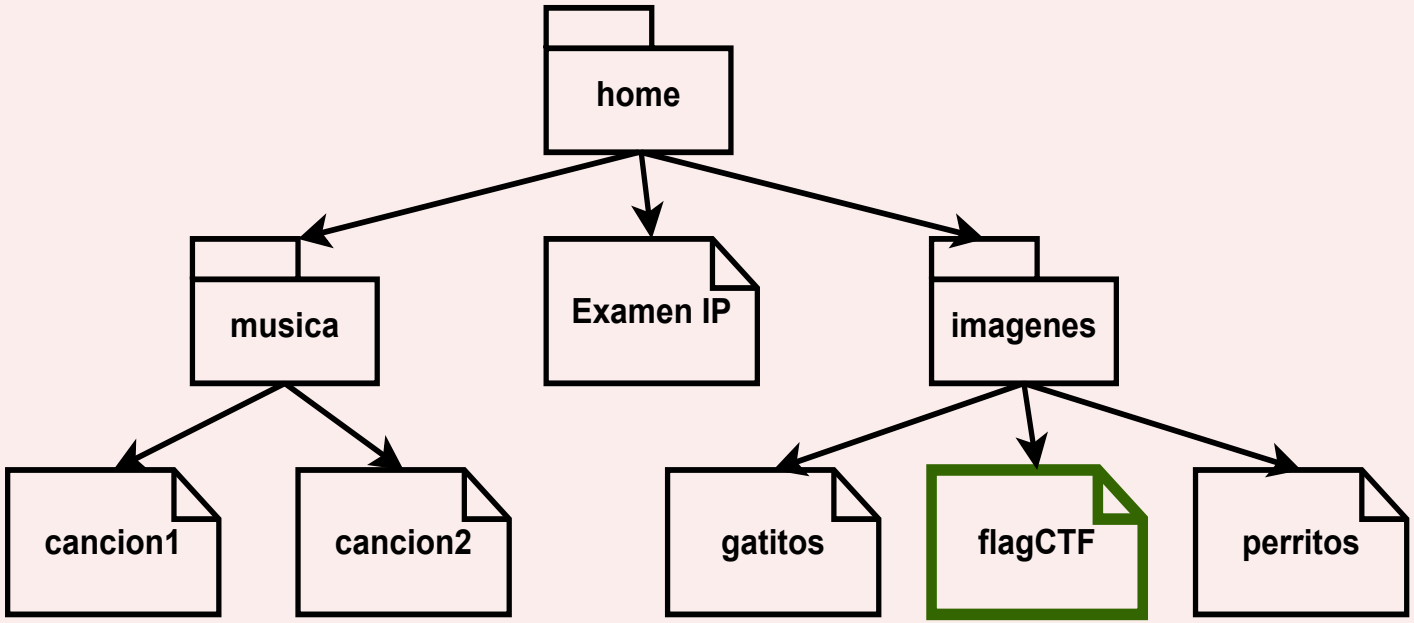
Envíos	Válidos	% éxito
0	0	0 %

D. CTF

Dada la descripción de los directorios de un ordenador, ¿puedes calcular cuantos pasos hay que dar para llegar al fichero flagCTF?

D. CTF

Si vemos los elementos del ordenador como nodos tenemos una estructura de grafo!!



D. CTF

- En el ordenador hay directorios y **ficheros**, pero solo se describen los directorios. O reservamos suficiente memoria, o utilizamos una estructura dinámica.
- El grafo puede tener ciclos! El enunciado nos avisa en: "Cabe destacar que gracias a los accesos directos, un directorio puede accederse desde más de un origen"
- Utilizar Strings como nodos es infernal!!!! Mapealos a enteros.

D. CTF

Idea inicial: Lanzamos un DFS desde cada padre, identificando por cuantos nodos pasamos en cada camino y nos quedamos el mínimo tras recorrernos el grafo:

$$\textit{Complejidad}: \quad \mathcal{O}(P \cdot |V|)$$

D. CTF

Idea inicial: Lanzamos un DFS desde cada padre, identificando por cuantos nodos pasamos en cada camino y nos quedamos el mínimo tras recorrernos el grafo:

$$\textit{Complejidad}: \quad \mathcal{O}(P \cdot |V|)$$

TLE

D. CTF

Idea básica: Lanzamos un BFS desde cada padre, identificando por cuantos nodos pasamos en cada camino y nos quedamos el mínimo:

$$\textit{Complejidad}: \quad \mathcal{O}(P \cdot |V|)$$

D. CTF

Idea básica: Lanzamos un BFS desde cada padre, identificando por cuantos nodos pasamos en cada camino y nos quedamos el mínimo:

$$\textit{Complejidad}: \quad \mathcal{O}(P \cdot |V|)$$

TLE

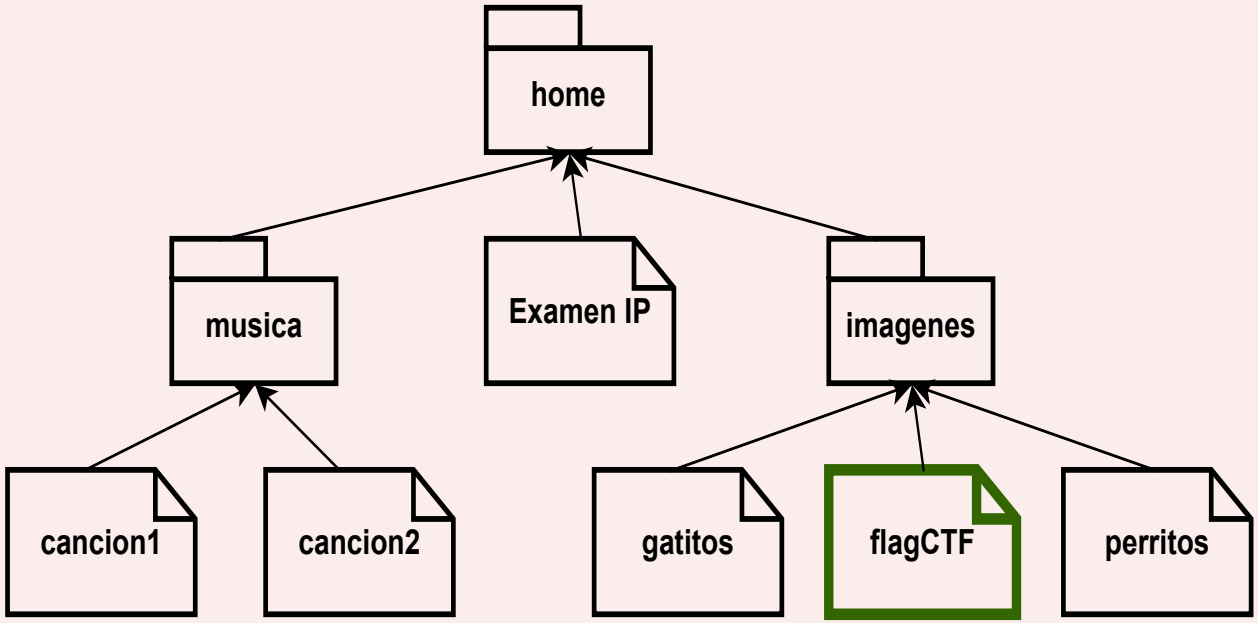
D. CTF

Podemos hacerlo mejor: En el grafo tenemos varias entradas pero una única salida!

D. CTF

Podemos hacerlo mejor: En el grafo tenemos varias entradas pero una única salida!

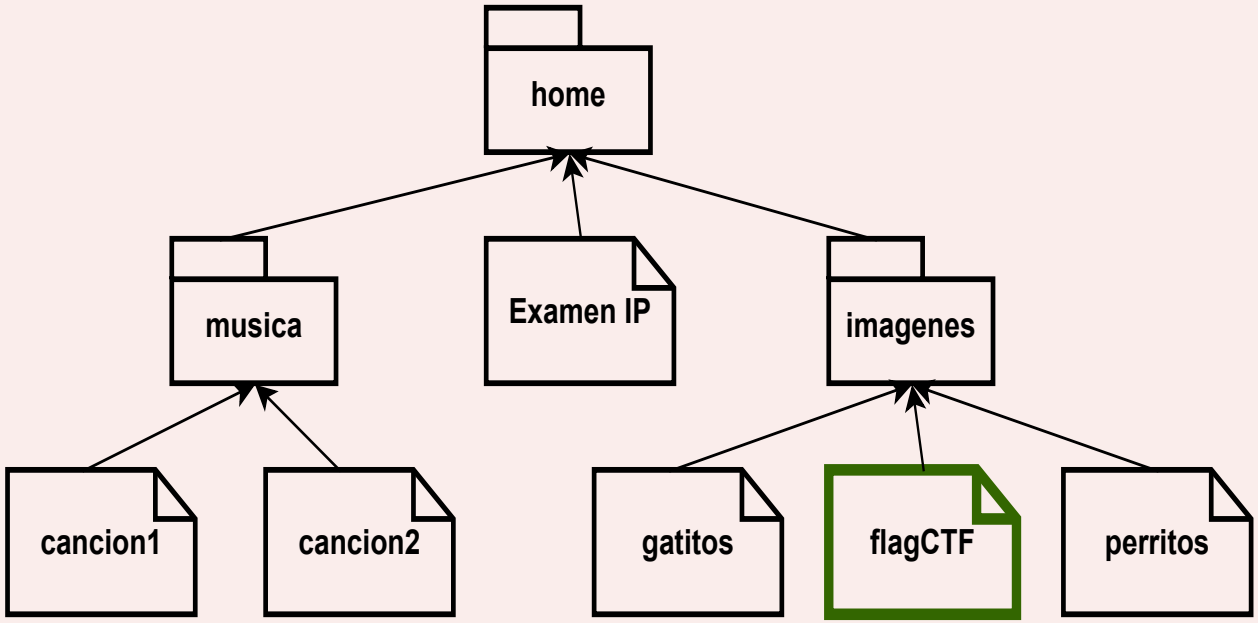
Idea: Podemos lanzar el BFS desde el fichero "flagCTF", pero debemos construir el grafo al reves!!!! Si llegamos a un nodo padre este será el más cercano.



D. CTF

Podemos hacerlo mejor: En el grafo tenemos varias entradas pero una única salida!

Idea: Podemos lanzar el BFS desde el fichero "flagCTF", pero debemos construir el grafo al reves!!!! Si llegamos a un nodo padre este será el más cercano.



$$\textit{Complejidad}: \quad \mathcal{O}(|V|)$$

D. CTF

Primero leemos el grafo al revés...

```
N, P = leerEnteros()

grafo = {}
para cada i entre 1 y N:
    linea = leerPalabras()
    directorio = linea[0]
    para cada hijo en linea[2:]:
        grafo.coger(hijo, []).añadir(nombre)

padres = set(leerPalabras())
```

D. CTF

...Y ahora aplicamos el BFS sobre la flag.

```
cola = deque([("flagCTF", 1)])
visitados = set(["flagCTF"])

mientras cola:
    actual, pasos = cola.sacar()

    para cada directorio en grafo.coger(actual, []):
        si directorio está en padres:
            imprimir(pasos)
            FIN

        si directorio no está en visitados:
            visitados.añadir(directorio)
            cola.añadir((directorio, pasos + 1))

imprimir(-1)
```

D. CTF

¿Y SI NO SE ME OCURRE?

D. CTF

Tal como estaba planteado el problema, también se podía lanzar un BFS sobre todos los directorios de los que partimos, en vez de empezar por la flag.

A diferencia del caso anterior que daba TLE, es 1 única ejecución, pero la cola ya cuenta con P elementos.

$$\textit{Complejidad}: \quad \mathcal{O}(|V|)$$

D. CTF

En este caso no es necesario leer el grafo al revés.

```
N, P = leerEnteros()

grafo = {}
para cada i entre 1 y N:
    linea = leerPalabras()
    directorio = linea[0]
    grafo[hijo] = linea[2:]

padres = leerPalabras()
```

D. CTF

Y al definir la cola del BFS, añadimos todos los padres.

```
cola = deque([(nodo, 1) para cada nodo en padres])
visitados = set(padres)

mientras cola:
    actual, pasos = cola.sacar()

    para cada directorio en grafo.coger(actual, []):
        si directorio es "flagCTF":
            imprimir(pasos)
            FIN

        si directorio no está en visitados:
            visitados.añadir(directorio)
            cola.añadir((directorio, pasos + 1))

imprimir(-1)
```

D. CTF

Este problema también se puede solucionar mediante el algoritmo de Dijkstra con un poco de cuidado y sin necesitar modelar el algoritmo al revés!

$$Complejidad: \quad \mathcal{O}(|V| \cdot log(|V|))$$

Este algoritmo será parte de la teoría de la última clase del curso :)

Es peor complejidad, pero en este problema el veredicto habría sido:

AC

E. Enseñando a sumar

Envíos	Válidos	% éxito
0	1	0 %

E. Enseñando a sumar

¿Dada una lista de números, podríamos decir eficientemente si otro número se puede obtener como la suma de dos de ellos?

E. Enseñando a sumar

Podemos resolver cada consulta con dos punteros

```
for target in Querys:
    found = False
    # doble bucle: probamos todas las parejas
    for i in range(n):
        for j in range(i + 1, n):
            if nums[i] + nums[j] == target:
                found = True
                break
        if found:
            break
    print("SI" if found else "NO")
```

E. Enseñando a sumar

Esta aproximación necesita de un doble bucle ($O(N^2)$) por cada query ($O(Q)$)

La complejidad total es cúbica: $O(N^2 \cdot Q)$

Pero para cada Query recalculemos todas las sumas... Memoricemoslas y hagamos una búsqueda eficiente (con un conjunto)

E. Enseñando a sumar

```
# Precomputamos todas las sumas de dos números distintos
sums = set()

for i in range(n):
    for j in range(i + 1, n):
        sums.add(nums[i] + nums[j])

# Respondemos cada consulta
for _ in range(q):
    target = int(sys.stdin.readline())
    print("SI" if target in sums else "NO")
```

E. Enseñando a sumar

Esta aproximación necesita de un doble bucle ($O(N^2)$) para memorizar todas las sumas. Cada query se resuelve en $O(1)$ porque usamos un conjunto

La complejidad total es cuadrática: $O(N^2)$

F. La ladrona de libros

Envíos	Válidos	% éxito
0	1	0 %

F. La ladrona de libros

Simplemente debemos guardar toda la información de la tienda de manera que podamos simular los eventos de manera eficiente:

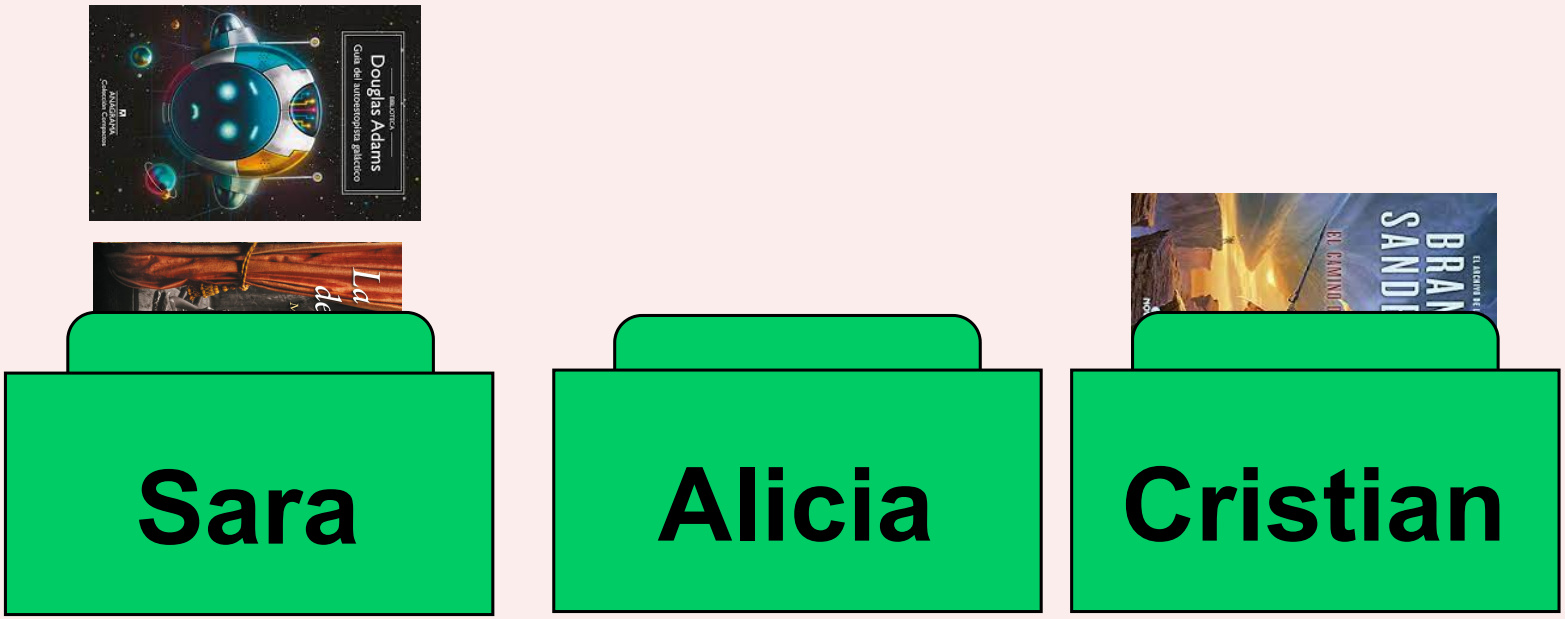
F. La ladrona de libros

Simplemente debemos guardar toda la información de la tienda de manera que podamos simular los eventos de manera eficiente:

F. La ladrona de libros

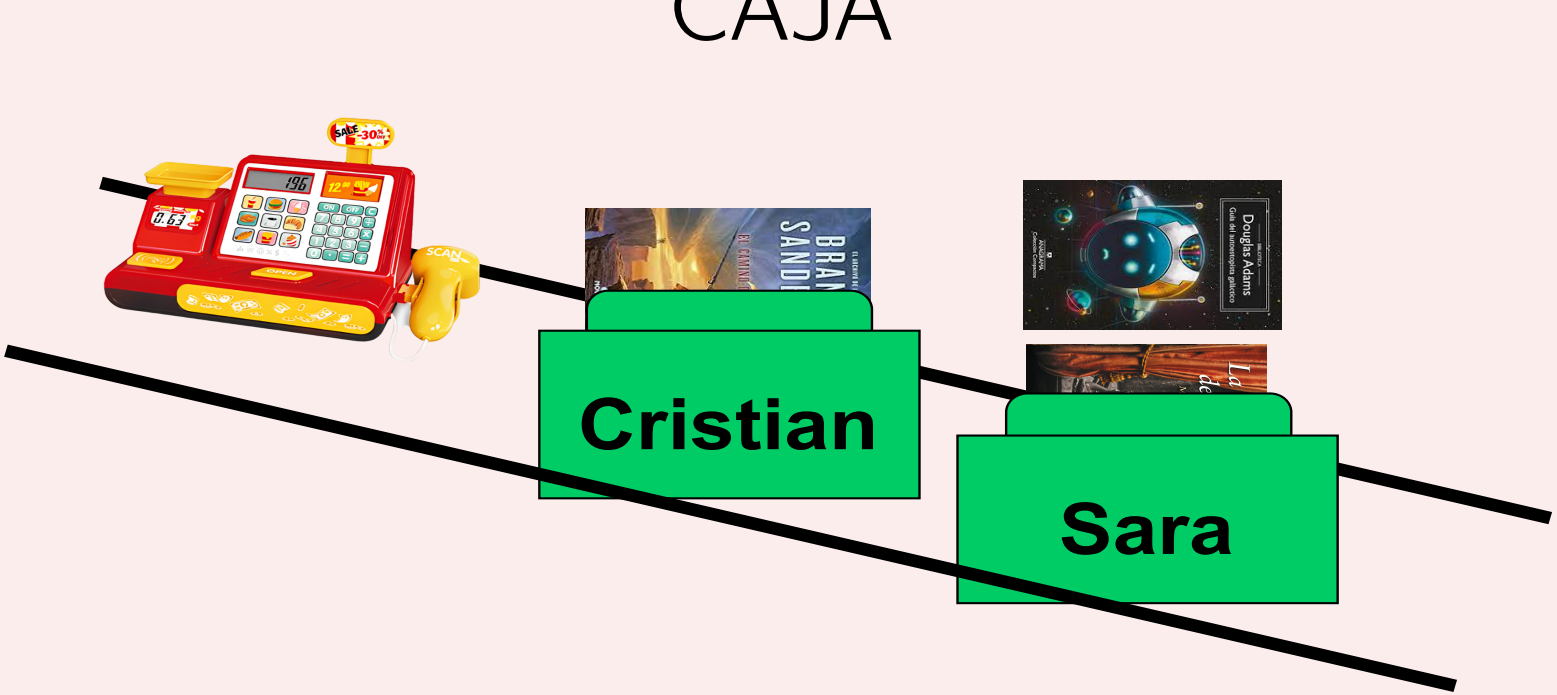
Simplemente debemos guardar toda la información de la tienda de manera que podamos simular los eventos de manera eficiente:

CESTAS



F. La ladrona de libros

Simplemente debemos guardar toda la información de la tienda de manera que podamos simular los eventos de manera eficiente:



F. La ladrona de libros

- Estantería: Array o Mapa
- Cestas de los clientes: Pilas ("Las cestas de tu tienda son demasiado estrechas por lo que si un cliente coge varios libros deberá apilarlos de forma que solo podrá acceder al último libro que ha cogido.")
- Caja registradora: Cola de cestas = Cola de Pilas! ("Tú recibirás los libros en la caja para cobrarlos, de esa manera si no se registran en el sistema de la caja en el orden natural, es que el muy desgraciado se ha guardado uno en algún lado.")

F. La ladrona de libros

- Evento 1: Introducir en la pila el libro indicado: **cesta[cliente].push(libro)**
- Evento 2: Eliminar en la pila el libro indicado: **cesta[cliente].pop()**
- Evento 3: Añadir cesta a la cola: **cola.push(cesta[cliente])**

F. La ladrona de libros

Una vez se han procesado todos los eventos, se debe comprobar la lista de la entrada, con la de la cola. En caso de que no coincida es porque falta un libro. La cesta del cliente que estemos revisando en ese momento es la ladrona!!!
Si todo coincide es que no ha venido la ladrona y no te está robando el poco dinero que ganas con tu tiendecita.
OJO!!: Cuidado con el caso de que el último libro de la caja sea el robado, en ese caso todo coincidirá, pero la cesta registrada por los eventos no estará vacía!!