



Concurso Grafos 2024/2025

Estadísticas y Soluciones



Clasificación de los problemas

Problema	Categoría
A - Ventana Resbaladiza	Ventana deslizante.
B - Buscando el dorsal	Búsqueda binaria.
C - Grupos de Amigos	BFS/DFS.
D - Dijkstraídos	Camino mínimo en grafos.
E - Derrapando en la mediana	Ordenación/Ad hoc.

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Ventana Resbaladiza	4	10 KB
B - Buscando el dorsal	11	42 MB
C - Grupos de Amigos	5	58 MB
D - Dijkstraídos	8	172 KB
E - Derrapando en la mediana	8	2 MB
- Total	36	(+-) 102 MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Ventana Resbaladiza	o.somalo.2021	9
B - Buscando el dorsal	Equipo webo	10
C - Grupos de Amigos	Equipo webo	8
D - Dijkstraídos	BBC	26
E - Derrapando en la mediana	hap.py	9

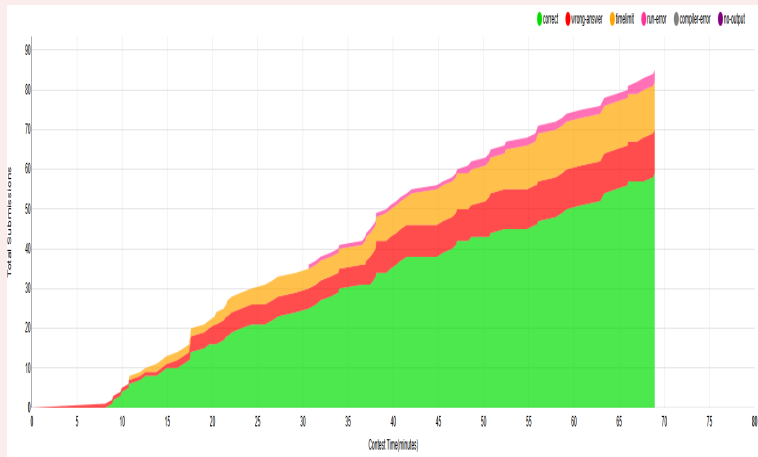
* Antes de congelar el marcador.

Estadísticas*

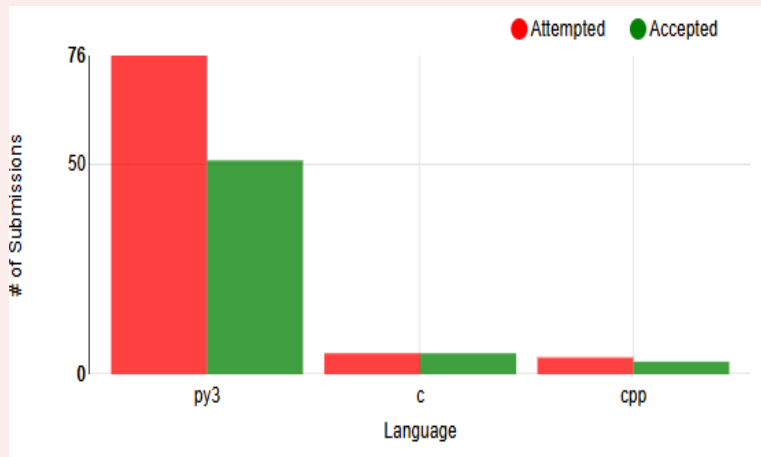
Problema	Válidos	Envíos	% éxito
A - Ventana Resbaladiza	14	25	56 %
B - Buscando el dorsal	13	21	62 %
C - Grupos de Amigos	7	8	87 %
D - Dijkstraídos	5	6	83 %
E - Derrapando en la mediana	11	14	79 %

* Antes de congelar el marcador.

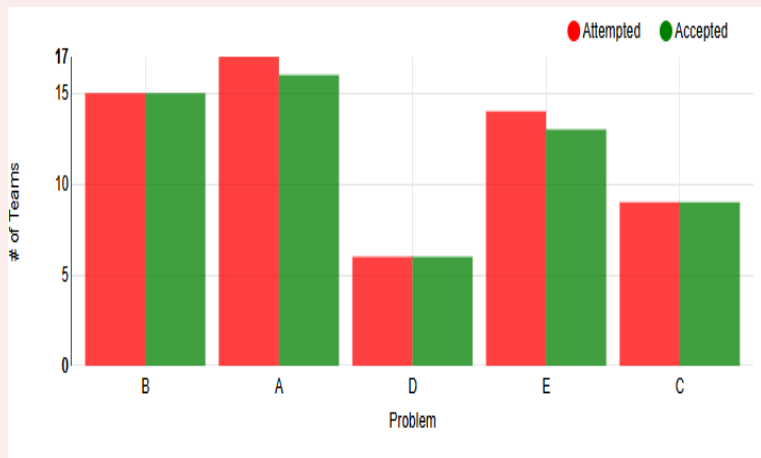
Estadísticas varias



Estadísticas varias



Estadísticas varias



● A. Ventana Resbaladiza

Envíos	Válidos	% éxito
25	14	56 %

A. Ventana Resbaladiza

Nos dan N números y tenemos que encontrar los V números consecutivos con mayor suma.

A. Ventana Resbaladiza

```
leer(N)
leer(V)
leer(A)

suma = 0
max = 0
sol = 0

for i in range(N-V):
    suma = 0
    for j in range(i,i+V):
        suma = suma + A[j]
    if suma > max:
        max = suma
        sol = i

imprimir((sol+1)+" "+max)
```

A. Ventana Resbaladiza

¡TIME LIMIT!

A. Ventana Resbaladiza

¡VENTANA DESLIZANTE!

A. Ventana Resbaladiza

```
leer(N)
leer(V)
leer(A)

suma = 0
ind = 0
for i in range(V):
    suma = suma + A[i]

max = suma
sol = 0
for i in range(V,N):
    suma = suma - A[ind]
    suma = suma + A[i]
    ind = ind + 1
    if suma > max:
        max = suma
        sol = ind

imprimir((sol+1)+" "+max)
```

● B. Buscando el dorsal

Envíos	Válidos	% éxito
21	13	62 %

B. Buscando el dorsal

Tenemos que buscar una serie de números en una lista ordenada.

B. Buscando el dorsal

```
leer(N)
leer(lista_numeros_cogidos)
leer(R)
leer(lista_numeros_raul)

for num in lista_numeros_raul:
    encontrado = buscar(num, lista_numeros_cogidos)
    if not encontrado:
        imprimir(num)
        break
```

B. Buscando el dorsal

```
buscar(num, lista_numeros_cogidos)
```

B. Buscando el dorsal

```
buscar(num, lista_numeros_cogidos)
```

- Recorrer la lista número a número hasta encontrarlo (o no)

B. Buscando el dorsal

```
buscar(num, lista_numeros_cogidos)
```

- Recorrer la lista número a número hasta encontrarlo (o no)

$$O(R * N) \rightarrow TLE$$

B. Buscando el dorsal

```
buscar(num, lista_numeros_cogidos)
```

- Recorrer la lista número a número hasta encontrarlo (o no)
- **BÚSQUEDA BINARIA**

$$O(R * \log(N)) \rightarrow \text{ACCEPTED}$$

B. Buscando el dorsal

```
buscar(num, lista_numeros_cogidos):  
    min = 0  
    max = len(lista_numeros_cogidos)  
    while max > min:  
        med = (max + min) / 2  
        if lista_numeros_cogidos[med] < num:  
            min = med + 1  
        else if lista_numeros_cogidos[med] > num:  
            max = med - 1  
        else:  
            return True  
    return False
```

● C. Grupos de Amigos

Envíos	Válidos	% éxito
8	7	87 %

C. Grupos de Amigos

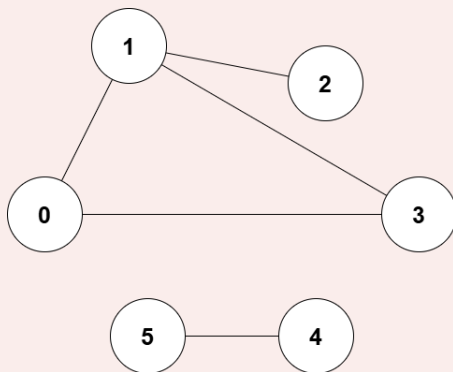
Quieres saber cuántos grupos distintos hay conociendo las relaciones de amistad que hay en la clase.

C. Grupos de Amigos

Idea:

- Ver la clase como un grafo: personas = nodos y amistades = aristas
- Grafo no dirigido

C. Grupos de Amigos



C. Grupos de Amigos

Solución:

- Con un recorrido no cubrimos el grafo completo.

C. Grupos de Amigos

Solución:

- Con un recorrido no cubrimos el grafo completo.
- Lanzamos un BFS o DFS por nodo no visitado.

C. Grupos de Amigos

```
# leer entrada
leer N, M
inicializar grafo
inicializar visitados
grupos = 0

# leer relaciones de amistad
for i in range(M)
    leer A, B
    añadir arista (A,B)
    añadir arista (B,A)

for i in range(M)
    if not visitados[i]:
        recorrido(i,grafo,visitados)
        grupos++

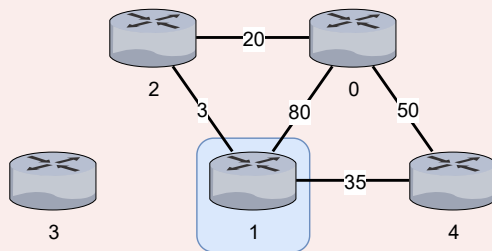
imprimir grupos
```

● D. Dijkstraídos

Envíos	Válidos	% éxito
6	5	83 %

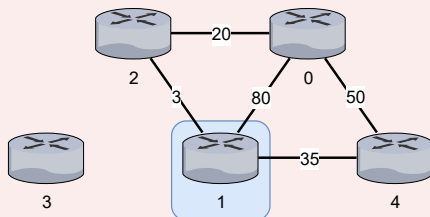
D. Dijkstraídos

El enunciado nos pedía implementar un algoritmo que se dio en clase de programación competitiva que pueda ayudarte a hacer los ejercicios de Redes de Computadores. En estos, te dan una red de dispositivos con sus distancias, y te piden dar la tabla de distancias desde uno de los *routers*.



D. Dijkstraídos

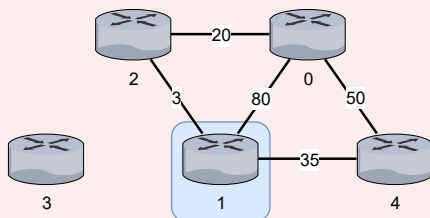
Problema: Encontrar las distancias mínimas de un *router* al resto de dispositivos.



0	
1	0
2	
3	?
4	

D. Dijkstraídos

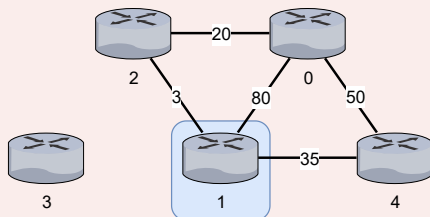
Problema: Encontrar las distancias mínimas de un *router* al resto de dispositivos.



0	80
1	0
2	3
3	?
4	35

D. Dijkstraídos

Problema: Encontrar las distancias mínimas de un *router* al resto de dispositivos.



0	$3+20=23$
1	0
2	3
3	?
4	35

D. Dijkstraídos

Problema: Encontrar las distancias mínimas de un *router* al resto de dispositivos.

Posibles aproximaciones:

- 1 Búsqueda por fuerza bruta → TLE, comprueba todos los caminos, incluidos ciclos, arcos en U innecesarios, ...
- 2 BFS → TLE, utiliza una cola normal, así que tampoco prioriza aquellos caminos con coste mínimo al explorar.

D. Dijkstraídos

Problema: Encontrar las distancias mínimas de un *router* al resto de dispositivos.

Posibles aproximaciones:

- 1 Búsqueda por fuerza bruta \rightarrow TLE, comprueba todos los caminos, incluidos ciclos, arcos en U innecesarios, ...
- 2 BFS \rightarrow TLE, utiliza una cola normal, así que tampoco prioriza aquellos caminos con coste mínimo al explorar.

¡ Dijkstra(ídos) !

D. Dijkstraídos

Algoritmo de Dijkstra: Con un grafo ponderado $G = (V, A)$ y un vector de distancias D , sigue la idea voraz de ir generando caminos utilizando la menor distancia de un nodo a otro no visitado.

Esta idea tendría una complejidad $O(|V| \cdot \log |V| + |A|) \approx O(|A| \cdot \log |V|)$ utilizando una cola de prioridad al escoger nodos con menor distancias.

D. Dijkstraídos

Algoritmo de Dijkstra

```
fun dijkstra(grafo, R):  
    cola = []  
    encolar(cola, (0, R))  
    distancias = [INF para nodo en grafo]  
    distancias[R] = 0  
  
    mientras cola:  
        distancia_actual, nodo_actual = desencolar(cola)  
        si distancia_actual > distancias[nodo_actual]:  
            saltar  
        para (vecino, peso) en grafo[nodo_actual]:  
            distancia = distancia_actual + peso  
            if distancia < distancias[vecino]:  
                distancias[vecino] = distancia  
                encolar(cola, (distancia, vecino))  
    devolver distancias
```

● E. Derrapando en la mediana




Envíos	Válidos	% éxito
14	11	79 %

E. Derrapando en la mediana




Se pide calcular el **doble** la mediana de un conjunto de datos.
La **mediana** es el valor central de los datos ordenados.

E. Derrapando en la mediana

- Número de datos impar: valor central

1	2	6	17	18		22		35	46	109	143	200
												
5 valores						Mediana= 22		5 valores				

- Número de datos par: media aritmética de los dos valores centrales

1	2	6	17	18		22	30		35	46	109	143	200
													
5 valores						Mediana= 26		5 valores					

E. Derrapando en la mediana

Se pide el doble para evitar trabajar con float/double.

Solución:

- **Ordenar** los datos de entrada → utilizando métodos del lenguaje
- Dos opciones:
 - Caso impar: devolver el doble del valor central
 - Caso par: devolver la suma de los valores centrales

E. Derrapando en la mediana

```
leer N
while N != 0:
    leer array
    ordenar array
    if N impar:
        mediana = array[N/2] * 2
    else:
        mediana = array[N/2] + array[N/2-1]
    imprimir mediana
    leer N
```