XXIII Olimpiada Murciana de Programación + AdaByron Murcia

Sesión de explicación de soluciones

Jueces de la XXIII Olimpiada Murciana de Programación + AdaByron Murcia

October 3, 2025

A: Cruzando el Río

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Dado un obstáculo a distancia d y n distancias de salto, comprobar cuántos pueden saltar la roca.

A: Cruzando el Río

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Dado un obstáculo a distancia d y n distancias de salto, comprobar cuántos pueden saltar la roca.
- Solución: Comparar los n números con d, si $a_i \leq d$, no pueden pasar.

A: Cruzando el Río

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Dado un obstáculo a distancia d y n distancias de salto, comprobar cuántos pueden saltar la roca.
- Solución: Comparar los n números con d, si $a_i \leq d$, no pueden pasar.
- Complejidad: $\mathcal{O}(n)$.

```cpp
int main() {
    int t; cin >> t;
    while (t--) {
        int n, d; cin >> n >> d;
        int no = 0;
        for (int i = 0; i < n; ++i) {
            int x; cin >> x;
            if (x ≤ d) {
                no++;
            }
        }
        cout << (no == 0 ? "TODOS\n" : to_string(no) + " NO PUEDEN\n");
    }
}
```

D: Calculadora de Beneficios

Autor del problema: Ginés García Mateos

El Problema

- **Enunciado:** Devolver el doble de un entero que puede tener hasta 1000 dígitos.

D: Calculadora de Beneficios

Autor del problema: Ginés García Mateos

El Problema

- **Enunciado:** Devolver el doble de un entero que puede tener hasta 1000 dígitos.
- Solución: En C++, implentar la operación del doble de un entero largo. En Python es trivial.

D: Calculadora de Beneficios

Autor del problema: Ginés García Mateos

El Problema

- **Enunciado:** Devolver el doble de un entero que puede tener hasta 1000 dígitos.
- Solución: En C++, implentar la operación del doble de un entero largo. En Python es trivial.
- Complejidad: $\mathcal{O}(n)$.

```cpp
void solve() {
    string s; cin >> s;
    string out = s;
    int carry = 0;
    for (int i = s.size() - 1; i >= 0; i--) {
        int d = ((s[i] - '0') * 2) + carry;
        carry = d / 10;
        out[i] = (d % 10) + '0';
    }
    if (carry) cout << carry;
    cout << out << "\n";
}
```

D: Calculadora de Beneficios

Autor del problema: Ginés García Mateos

El Problema

- **Enunciado:** Devolver el doble de un entero que puede tener hasta 1000 dígitos.
- Solución: En C++, implentar la operación del doble de un entero largo. En Python es trivial.
- Complejidad: $\mathcal{O}(n)$.

```cpp
void solve() {
    string s; cin >> s;
    string out = s;
    int carry = 0;
    for (int i = s.size() - 1; i >= 0; i--) {
        int d = ((s[i] - '0') * 2) + carry;
        carry = d / 10;
        out[i] = (d % 10) + '0';
    }
    if (carry) cout << carry;
    cout << out << "\n";
}
```

D: Calculadora de Beneficios

Autor del problema: Ginés García Mateos

El Problema

- **Enunciado:** Devolver el doble de un entero que puede tener hasta 1000 dígitos.
- Solución: En C++, implentar la operación del doble de un entero largo. En Python es trivial.
- Complejidad: $\mathcal{O}(n)$.

```cpp
void solve() {
    string s; cin >> s;
    string out = s;
    int carry = 0;
    for (int i = s.size() - 1; i >= 0; i--) {
        int d = ((s[i] - '0') * 2) + carry;
        carry = d / 10;
        out[i] = (d % 10) + '0';
    }
    if (carry) cout << carry;
    cout << out << "\n";
}
```

```python
for _ in range(int(input())):
    print(int(input()) * 2)
```

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n + 1)$

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n+1)$

- **Complejidad:** $\mathcal{O}(N)$

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n+1)$

- **Complejidad:** $\mathcal{O}(N)$
- **TLE:** $(N, M \leq 10^{18})$.

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n+1)$

- **Complejidad:** $\mathcal{O}(N)$
- **TLE:** $(N, M \leq 10^{18})$.

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n+1)$

- **Complejidad:** $\mathcal{O}(N)$
- **TLE:** $(N, M \leq 10^{18})$.

Error Común

Utilizar `int` en lugar de `long long`

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

El Problema

Determinar si un número N es oblongo (i.e., múltiplo de dos números consecutivos)

Solución por Fuerza Bruta

Probar todas las combinaciones $n \cdot (n+1)$

- **Complejidad:** $\mathcal{O}(N)$
- **TLE:** $(N, M \leq 10^{18})$.

Error Común

Utilizar `int` en lugar de `long long`

- **WA:** overflow (`int` solo cubre hasta $\approx 10^{12}$, pero $1 \leq N \leq 10^{18}$)

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

Comprobar : $\lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

$$\text{Comprobar} : \lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$$

- **Complejidad:** $\mathcal{O}(1)$

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

$$\text{Comprobar} : \lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$$

- **Complejidad:** $\mathcal{O}(1)$
- **AC.**

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

$$\text{Comprobar} : \lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$$

- **Complejidad:** $\mathcal{O}(1)$
- **AC.**

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

$$\text{Comprobar} : \lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$$

- **Complejidad:** $\mathcal{O}(1)$
- **AC.**

Alternativa: Búsqueda Binaria

- **Complejidad:** $\mathcal{O}(\log N)$ (10^{18})

I: Encriptación... ¿Oblonga?

Autor del problema: Carlos Durá Costa (Impl. by Adrián Fenollar Navarro)

Solución Ad-hoc

$$\text{Comprobar} : \lfloor \sqrt{N} \rfloor \cdot (\lfloor \sqrt{N} \rfloor + 1) == N$$

- **Complejidad:** $\mathcal{O}(1)$
- **AC.**

Alternativa: Búsqueda Binaria

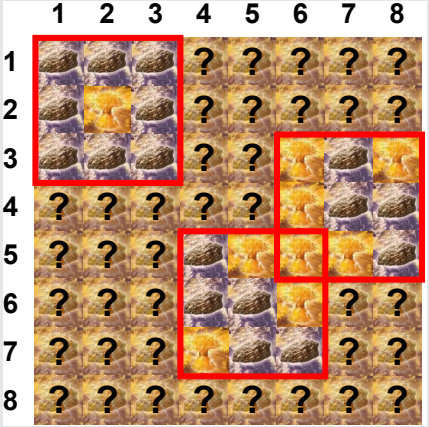
- **Complejidad:** $\mathcal{O}(\log N)$ (10^{18})
- **AC.**

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

El Problema

Dadas F fotografías descriptivas sobre las parcelas de un terreno de tamaño $N \times M$, determinar el número mínimo y máximo de parcelas con cultivo que puede haber en el terreno.



G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Solución por Fuerza Bruta

Crear una matriz de tamaño $N \times M$ y para cada fotografía, marcar las parcelas que contienen cultivo. Luego, iterar sobre la matriz contando.

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Solución por Fuerza Bruta

Crear una matriz de tamaño $N \times M$ y para cada fotografía, marcar las parcelas que contienen cultivo. Luego, iterar sobre la matriz contando.

- **Complejidad:** $\mathcal{O}(N \cdot M + F)$

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Solución por Fuerza Bruta

Crear una matriz de tamaño $N \times M$ y para cada fotografía, marcar las parcelas que contienen cultivo. Luego, iterar sobre la matriz contando.

- **Complejidad:** $\mathcal{O}(N \cdot M + F)$
- **Memoria:** $\mathcal{O}(N \cdot M) \rightarrow$ ¡hasta 10^{10} (1TB RAM)!

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Solución por Fuerza Bruta

Crear una matriz de tamaño $N \times M$ y para cada fotografía, marcar las parcelas que contienen cultivo. Luego, iterar sobre la matriz contando.

- **Complejidad:** $\mathcal{O}(N \cdot M + F)$
- **Memoria:** $\mathcal{O}(N \cdot M) \rightarrow$ ¡hasta 10^{10} (1TB RAM)!
- **TLE/MLE/RTE** ($N, M \leq 10^6$).

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

Solución eficiente con Estructuras de Datos

Almacenar las parcelas con cultivo en un conjunto (`std::set`) o mapa (`std::map`) para evitar duplicados y permitir búsquedas rápidas.

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

Solución eficiente con Estructuras de Datos

Almacenar las parcelas con cultivo en un conjunto (`std::set`) o mapa (`std::map`) para evitar duplicados y permitir búsquedas rápidas.

- **Complejidad:** $\mathcal{O}(F \log F)$

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

Solución eficiente con Estructuras de Datos

Almacenar las parcelas con cultivo en un conjunto (`std::set`) o mapa (`std::map`) para evitar duplicados y permitir búsquedas rápidas.

- **Complejidad:** $\mathcal{O}(F \log F)$
- **Memoria:** $\mathcal{O}(F)$

G: Control de Producción

Autor del problema: Carlos Agulló Domingo (Impl. by Adrián Fenollar Navarro)

Mejor aproximación por Fuerza Bruta

Solo guardar en una lista las parcelas que contienen cultivo (representación *sparse*). Para cada nuevo cultivo, comprobar si ya está en la lista.

- **Complejidad:** $\mathcal{O}(F^2)$
- **Memoria:** $\mathcal{O}(F)$
- **TLE:** ($F \leq 10^5$).

Solución eficiente con Estructuras de Datos

Almacenar las parcelas con cultivo en un conjunto (`std::set`) o mapa (`std::map`) para evitar duplicados y permitir búsquedas rápidas.

- **Complejidad:** $\mathcal{O}(F \log F)$
- **Memoria:** $\mathcal{O}(F)$
- **AC.**

B: Asignación de Grupos

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hacer la asignación de grupos con notas.

B: Asignación de Grupos

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hacer la asignación de grupos con notas.

Solución

- Paso 1: ordenar alumnos por nota $\mathcal{O}(nlog(n))$.

B: Asignación de Grupos

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hacer la asignación de grupos con notas.

Solución

- Paso 1: ordenar alumnos por nota $\mathcal{O}(nlog(n))$.
- Paso 2: asignar a cada alumno su primera opción disponible, $\mathcal{O}(n^2)$.

B: Asignación de Grupos

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hacer la asignación de grupos con notas.

Solución

- Paso 1: ordenar alumnos por nota $\mathcal{O}(nlog(n))$.
- Paso 2: asignar a cada alumno su primera opción disponible, $\mathcal{O}(n^2)$.
- Hay que tener cuidado con la capacidad del grupo.

B: Asignación de Grupos

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hacer la asignación de grupos con notas.

Solución

- Paso 1: ordenar alumnos por nota $\mathcal{O}(nlog(n))$.
- Paso 2: asignar a cada alumno su primera opción disponible, $\mathcal{O}(n^2)$.
- Hay que tener cuidado con la capacidad del grupo.

Sería más interesante si los laboratorios también tuvieran listas de preferencia...

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

El Problema

Dado el catálogo de precios de todas las CPUs y GPUs de una tienda, determinar cuál es la combinación CPU + GPU más cara que puedes comprar con un presupuesto D.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

El Problema

Dado el catálogo de precios de todas las CPUs y GPUs de una tienda, determinar cuál es la combinación CPU + GPU más cara que puedes comprar con un presupuesto D.

Solución por Fuerza Bruta

Probar todas las combinaciones de CPU y GPU, quedándonos con la más cara que no exceda el presupuesto D.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

El Problema

Dado el catálogo de precios de todas las CPUs y GPUs de una tienda, determinar cuál es la combinación CPU + GPU más cara que puedes comprar con un presupuesto D.

Solución por Fuerza Bruta

Probar todas las combinaciones de CPU y GPU, quedándonos con la más cara que no exceda el presupuesto D.

- **Complejidad:** $\mathcal{O}(N \cdot M)$, donde N es el número de CPUs y M el número de GPUs.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

El Problema

Dado el catálogo de precios de todas las CPUs y GPUs de una tienda, determinar cuál es la combinación CPU + GPU más cara que puedes comprar con un presupuesto D.

Solución por Fuerza Bruta

Probar todas las combinaciones de CPU y GPU, quedándonos con la más cara que no exceda el presupuesto D.

- **Complejidad:** $\mathcal{O}(N \cdot M)$, donde N es el número de CPUs y M el número de GPUs.
- **TLE:** $(N, M \leq 10^5)$.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.
- **AC.**

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.
- **AC.**

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.
- **AC.**

Opción 2: Búsqueda Binaria

Ordenar las GPUs y para cada CPU, buscar la GPU más cara que no exceda el presupuesto restante usando búsqueda binaria.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.
- **AC.**

Opción 2: Búsqueda Binaria

Ordenar las GPUs y para cada CPU, buscar la GPU más cara que no exceda el presupuesto restante usando búsqueda binaria.

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N \log M) = \mathcal{O}(N \log M)$.

F: Presupuestos Cuestionables

Autor del problema: Adrián Fenollar Navarro

Opción 1: Dos Punteros

Ordenar las CPUs y GPUs. Usar dos punteros para recorrer ambas listas y encontrar la combinación más cara que no exceda el presupuesto.

`ptr_cpu` apunta a la CPU más barata y `ptr_gpu` a la GPU más cara. Si la suma de ambos precios es mayor que `D`, mover `ptr_gpu` a la izquierda (GPU más barata). Si es menor o igual, actualizar la mejor combinación y mover `ptr_cpu` a la derecha (CPU más cara).

- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N + M) \approx \mathcal{O}(N \log M)$.
- **AC.**

Opción 2: Búsqueda Binaria

Ordenar las GPUs y para cada CPU, buscar la GPU más cara que no exceda el presupuesto restante usando búsqueda binaria.

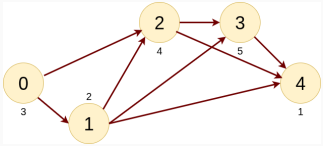
- **Complejidad:** $\mathcal{O}(N \log M) + \mathcal{O}(N \log M) = \mathcal{O}(N \log M)$.
- **AC.**

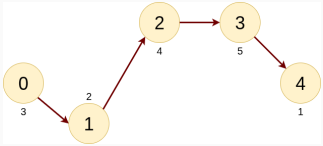
E: Proyecto Dolce

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hallar la Ruta Crítica en un diagrama de tareas.



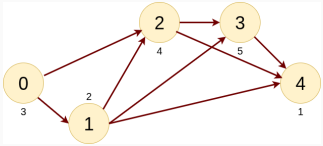


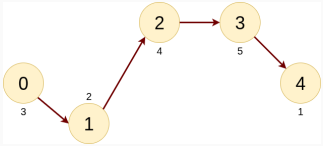
E: Proyecto Dolce

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hallar la Ruta Crítica en un diagrama de tareas.
- Es un problema de grafos (particularmente un DAG).



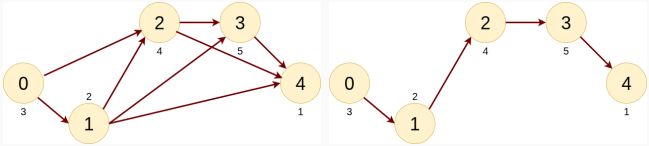


E: Proyecto Dolce

Autor del problema: Marcos Domínguez Velad

El Problema

- **Enunciado:** Hallar la Ruta Crítica en un diagrama de tareas.
- Es un problema de grafos (particularmente un DAG).



Solución

- Hay que tener en cuanta las dependencias antes de procesar nodos...
- Solución: variante de Dijkstra en DAG, encontrar la ruta más larga.
- Enfoque similar: orden topológico de las tareas + Programación dinámica, $\mathcal{O}(n+m)$ en tiempo y memoria.

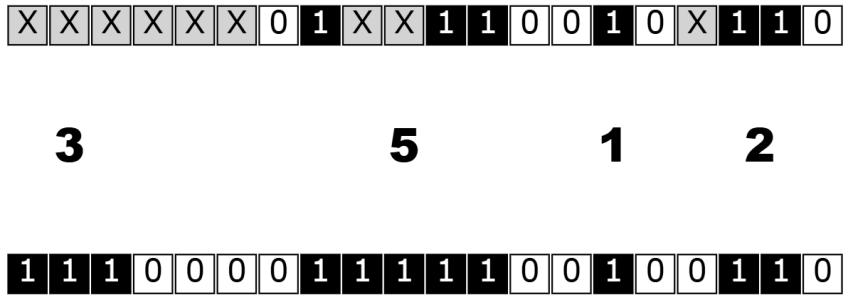
$$\text{dist}[v] = \max\big(\text{dist}[v],\ \text{dist}[u] + \text{len}[v]\big)$$

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

El Problema

- **Enunciado:** Resolver un nonograma unidimiensional (sólo hay pistas en la dimensión x)

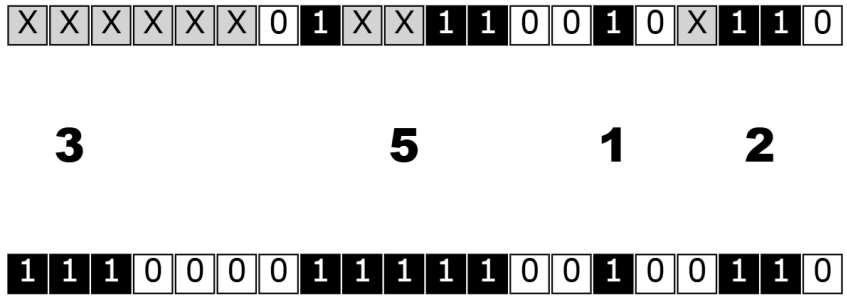


H: Lineanograma

Autor del problema: Carlos Agulló Domingo

El Problema

- **Enunciado:** Resolver un nonograma unidimiensional (sólo hay pistas en la dimensión x)
- **Enunciado:** Equivale a decidir si a la cadena $\sum_{i=1}^{M} 1^{a_i} 0^{i<M}$ se le pueden añadir ceros (sin separar los grupos de unos) para hacerla compatible con el patrón dado.



H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Primera aproximación: Expresión Regular

- Construir una expresión regular que represente la cadena de pistas.

```
string expr = "^[0X]*";
for(int i = 0; i < n-1; ++i){
    expr += "[1X]{" + to_string(v[i]) + "}[0X]+";
}
for (int i = max(0, n- 1); i < n; ++i) {
    expr += "[1X]{" + to_string(v[i]) + "}";
}
expr += "[0X]*$";
```

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Primera aproximación: Expresión Regular

- Construir una expresión regular que represente la cadena de pistas.

```
string expr = "^[0X]*";
for(int i = 0; i < n-1; ++i){
    expr += "[1X]{" + to_string(v[i]) + "}[0X]+";
}
for (int i = max(0, n- 1); i < n; ++i) {
    expr += "[1X]{" + to_string(v[i]) + "}";
}
expr += "[0X]*$";
```

- **Complejidad**: $\Omega(N^2), O(NP)$

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Primera aproximación: Expresión Regular

- Construir una expresión regular que represente la cadena de pistas.

```
string expr = "^[0X]*";
for(int i = 0; i < n-1; ++i){
    expr += "[1X]{" + to_string(v[i]) + "}[0X]+";
}
for (int i = max(0, n- 1); i < n; ++i) {
    expr += "[1X]{" + to_string(v[i]) + "}";
}
expr += "[0X]*$";
```

- **Complejidad**: $\Omega(N^2), O(NP)$
- TLE.

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.
- **Idea**: Programación Dinámica: una solución parcial válida hace coincidir los **c** primeros caracteres de la cadena 'pista' con los **d** primeros caracteres del patrón (necesariamente, se han añadido **d**−**c** '0' a la solución)

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.
- **Idea**: Programación Dinámica: una solución parcial válida hace coincidir los **c** primeros caracteres de la cadena 'pista' con los **d** primeros caracteres del patrón (necesariamente, se han añadido **d**−**c** '0' a la solución)
- **Caso base**: $DP(\mathbf{c}, \mathbf{d}) = 1$ si $c := 0, d := 0$

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.
- **Idea**: Programación Dinámica: una solución parcial válida hace coincidir los **c** primeros caracteres de la cadena 'pista' con los **d** primeros caracteres del patrón (necesariamente, se han añadido **d**−**c** '0' a la solución)
- **Caso base**: $DP(\mathbf{c}, \mathbf{d}) = 1$ si $c := 0, d := 0$
- **Caso recursivo**: $DP(\mathbf{c}, \mathbf{d}) := 1$ si: $DP(\mathbf{c}, \mathbf{d} - 1) = 1$ y tanto la pista como el patrón admiten añadir '0' en las posiciones **c** y **d**; o bien, $DP(\mathbf{c} - 1, \mathbf{d} - 1) = 1$ tanto la pista como el patrón admiten añadir '1' o '0' en las posiciones **c** y **d**.

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.
- **Idea**: Programación Dinámica: una solución parcial válida hace coincidir los **c** primeros caracteres de la cadena 'pista' con los **d** primeros caracteres del patrón (necesariamente, se han añadido **d**−**c** '0' a la solución)
- **Caso base**: $DP(\mathbf{c}, \mathbf{d}) = 1$ si $c := 0, d := 0$
- **Caso recursivo**: $DP(\mathbf{c}, \mathbf{d}) := 1$ si: $DP(\mathbf{c}, \mathbf{d} - 1) = 1$ y tanto la pista como el patrón admiten añadir '0' en las posiciones **c** y **d**; o bien, $DP(\mathbf{c} - 1, \mathbf{d} - 1) = 1$ tanto la pista como el patrón admiten añadir '1' o '0' en las posiciones **c** y **d**.
- **Complejidad**: $O((\sum a_i + M - 1) * (N - \sum a_i + M - 1)) = O(N^2)$

H: Lineanograma

Autor del problema: Carlos Agulló Domingo

Solución

- Vemos que la pista define una cadena de $\sum a_i + M - 1$ caracteres, luego, tenemos que añadir $N - \sum a_i + M - 1$ '0' en posiciones válidas.
- **Idea**: Programación Dinámica: una solución parcial válida hace coincidir los **c** primeros caracteres de la cadena 'pista' con los **d** primeros caracteres del patrón (necesariamente, se han añadido **d**−**c** '0' a la solución)
- **Caso base**: $DP(\mathbf{c}, \mathbf{d}) = 1$ si $c := 0, d := 0$
- **Caso recursivo**: $DP(\mathbf{c}, \mathbf{d}) := 1$ si: $DP(\mathbf{c}, \mathbf{d} - 1) = 1$ y tanto la pista como el patrón admiten añadir '0' en las posiciones **c** y **d**; o bien, $DP(\mathbf{c} - 1, \mathbf{d} - 1) = 1$ tanto la pista como el patrón admiten añadir '1' o '0' en las posiciones **c** y **d**.
- **Complejidad**: $O((\sum a_i + M - 1) * (N - \sum a_i + M - 1)) = O(N^2)$
- AC.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

El Problema

- **Enunciado:** Dado un juego por turnos, finito, con información completa, determinista, determinar el jugador ganador.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

El Problema

- **Enunciado:** Dado un juego por turnos, finito, con información completa, determinista, determinar el jugador ganador.
- **Enunciado:** Por las características del juego, el enunciado nos indica que existe una jugada óptima y se puede decidir quién ganará.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Solución

- **Idea inicial**: Backtracking con el algoritmo minimax.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Solución

- **Idea inicial**: Backtracking con el algoritmo minimax.
- Estado de juego: definido por (colocación de los luchadores, vida de los luchadoes, turno activo)

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Solución

- **Idea inicial**: Backtracking con el algoritmo minimax.
- Estado de juego: definido por (colocación de los luchadores, vida de los luchadoes, turno activo)
- **Caso base**: Un equipo de luchadores está vacío (pierde ese jugador)

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Solución

- **Idea inicial**: Backtracking con el algoritmo minimax.
- Estado de juego: definido por (colocación de los luchadores, vida de los luchadoes, turno activo)
- **Caso base**: Un equipo de luchadores está vacío (pierde ese jugador)
- **Caso recursivo**: Si la acción **Atacar** o **Intercambiar y atacar** te llevan a un estado perdedor (para el otro jugador) este es un estado ganador, de lo contrario, **Atacar** y **Intercambiar y atacar** te llevan a un estado ganador (para el otro jugador), y has ganado.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Solución

- **Idea inicial**: Backtracking con el algoritmo minimax.
- Estado de juego: definido por (colocación de los luchadores, vida de los luchadoes, turno activo)
- **Caso base**: Un equipo de luchadores está vacío (pierde ese jugador)
- **Caso recursivo**: Si la acción **Atacar** o **Intercambiar y atacar** te llevan a un estado perdedor (para el otro jugador) este es un estado ganador, de lo contrario, **Atacar** y **Intercambiar y atacar** te llevan a un estado ganador (para el otro jugador), y has ganado.
- **Complejidad**: Cada turno, la vida de un equipo se reduce en 1, en el peor caso, si tenemos que recorrer todo el árbol de estados. El factor de ramificación es 2, y la vida de ambos equipos es hasta 200 ($\sum a_i + \sum b_i$). $O(2^{200})$ estados, TLE.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Optimización

- **Observación**: Se puede llegar a estados similares (iguales) desde jugadas diferentes. Usar memoización. Para almacenar el estado actual eficientemente, nos podemos fijar en que aspectos del estado de juego son necesarios: quienes son los 4 (o menos) luchadores activos, la vida de los luchadores activos, el orden de los luchadores activos (sólo el de los defensores, por simetría de las acciones) y el jugador activo. Destacamos que no hay que almacenar información sobre el resto de los luchadores, ya que mantienen su orden y vida máxima mientras permanecen inactivos.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Optimización

- **Observación**: Se puede llegar a estados similares (iguales) desde jugadas diferentes. Usar memoización. Para almacenar el estado actual eficientemente, nos podemos fijar en que aspectos del estado de juego son necesarios: quienes son los 4 (o menos) luchadores activos, la vida de los luchadores activos, el orden de los luchadores activos (sólo el de los defensores, por simetría de las acciones) y el jugador activo. Destacamos que no hay que almacenar información sobre el resto de los luchadores, ya que mantienen su orden y vida máxima mientras permanecen inactivos.
- **Complejidad**:

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Optimización

- **Observación**: Se puede llegar a estados similares (iguales) desde jugadas diferentes. Usar memoización. Para almacenar el estado actual eficientemente, nos podemos fijar en que aspectos del estado de juego son necesarios: quienes son los 4 (o menos) luchadores activos, la vida de los luchadores activos, el orden de los luchadores activos (sólo el de los defensores, por simetría de las acciones) y el jugador activo. Destacamos que no hay que almacenar información sobre el resto de los luchadores, ya que mantienen su orden y vida máxima mientras permanecen inactivos.
- **Complejidad**:
  - Cota sobre el número de estados: $Num_{estados} \leq 2 * N^2 * \frac{M*(M-1)}{2} * HP_{max}^4 \leq 10^8$. En la práctica se visitan muchos menos estados, por la evaluación de minimax en cortocircuito y estados de juego inalcanzables.

C: La Final de Boxa-Fighters

Autor del problema: Carlos Agulló Domingo

Optimización

- **Observación**: Se puede llegar a estados similares (iguales) desde jugadas diferentes. Usar memoización. Para almacenar el estado actual eficientemente, nos podemos fijar en que aspectos del estado de juego son necesarios: quienes son los 4 (o menos) luchadores activos, la vida de los luchadores activos, el orden de los luchadores activos (sólo el de los defensores, por simetría de las acciones) y el jugador activo. Destacamos que no hay que almacenar información sobre el resto de los luchadores, ya que mantienen su orden y vida máxima mientras permanecen inactivos.
- **Complejidad**:
  - Cota sobre el número de estados: $Num_{estados} \leq 2 * N^2 * \frac{M*(M-1)}{2} * HP_{max}^4 \leq 10^8$. En la práctica se visitan muchos menos estados, por la evaluación de minimax en cortocircuito y estados de juego inalcanzables.
  - Tiempo $O(Num_{estados} \log Num_{estados})$ con `std::map` o $O(Num_{estados})$ con `std::unordered_map`.

XXIII Olimpiada Murciana de Programación

Sesión de explicación de soluciones

Jueces de la XXIII Olimpiada Murciana de Programación

October 3, 2025