LLamaCup 2024-I Editorial

GPC UNJFSC

25 de Mayo, 2024

Problema A: Imprimiendo enteros por paridad

Descripción

Escribe un programa que lea un arreglo de n enteros y un entero b. Si $b = 0$, tu programa deberá imprimir los valores del arreglo que sean pares; si $b = 1$, tu programa deberá imprimir los valores del arreglo que sean impares. El orden de aparición debe respetarse.

Entrada

Un entero n seguido de los n enteros del arreglo y posteriormente un entero b. Puedes suponer que $1 \leq n \leq 100$, que b puede ser 0 o 1 y que los elementos del arreglo están entre 0 y 1000.

Salida

Una secuencia de enteros separados por espacios que correspondan con lo solicitado en la descripción.

Ejemplo

ENTRADA	SALIDA
5 2 5 9 6 1 0	2 6
5 2 5 9 6 1 1	5 9 1

Resolución

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dato = sc.nextInt();
        int n1[] = new int[dato];
        for (int i = 0; i < dato; i++) {
            n1[i] = sc.nextInt();
        }
        int opc = sc.nextInt();
        if (opc == 0) {
            for (int i = 0; i < dato; i++) {
                if (n1[i] % 2 == 0) {
                    System.out.print(n1[i] + "-");
                }
            }
        } else if (opc == 1) {
            for (int i = 0; i < dato; i++) {
                if (n1[i] % 2 != 0) {
                    System.out.print(n1[i] + "-");
                }
            }
        }
    }
}
```

Explicación del Código

- **Lectura del tamaño del arreglo:** Se lee un entero n que indica el tamaño del arreglo.

- **Lectura de los elementos del arreglo:** Se inicializa un arreglo de tamaño n y se llenan sus elementos mediante un bucle `for`.
- **Lectura del valor de b:** Se lee un entero b que puede ser 0 o 1.
- **Impresión de valores según b:**
  - Si $b = 0$, se imprimen todos los elementos pares del arreglo.
  - Si $b = 1$, se imprimen todos los elementos impares del arreglo.
- El bucle `for` recorre cada elemento del arreglo y verifica si es par o impar usando el operador módulo (`%`).
- Los valores que cumplen con la condición se imprimen en una línea, separados por un espacio.

Problema B: Triples

Descripción

- Escribe un programa que lea una lista de enteros no negativos y pueda encuentrar todos los "triples"que serian (múltiplos de 3). Para el propósito de este problema, un triple se define como cualquier número que se pueda expresar como $3 \times n$. Por lo que 3, 6 y 9 se consideran triples.

- Para complicar un poco más las cosas, una vez que identifiques todos los triples debes imprimir su ubicación en la lista original.

Entrada

- La primera línea de entrada consiste de un entero n que representa el número de enteros en la lista.
- Las siguientes n líneas representan los enteros de la lista.

Salida

- Si hubo múltiplos de 3 en la lista, la primera línea de salida debe ser un número t que indica el número de triples encontrados.
- La segunda línea contiene t enteros separados por un espacio, describiendo la posición de los triples en orden ascendente.

Ejemplo

ENTRADA	**SALIDA**
7 10 12 8 9 3 29 30	4 2 4 5 7
7 16 49 2 10 28 55 31	no hay triples

Resolución

Con STL (vector)

El programa utiliza un vector para almacenar las posiciones de los triples encontrados en la lista. Itera sobre la lista de enteros, y si un número es divisible por 3, agrega su posición al vector. Al final, imprime el tamaño del vector seguido de sus elementos.

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n; cin >> n;
    vector<int> ans;
    for(int i = 0; i < n; i++){
        int x; cin >> x;
        if(x % 3 == 0) ans.push_back(i + 1);
    }
    if(ans.empty()) { cout << "no hay triples"; return 0; }
    int N = ans.size();
    cout << N << '\n';
    for(int i = 0; i < N; i++) cout << ans[i] << ' ';
}
```

Sin STL (array)

En esta versión, se utiliza un array estático para almacenar los enteros de la lista. Se cuenta el número de triples mientras se recorre la lista, y si hay alguno, se imprimen sus posiciones al final. Si no hay triples, se imprime "no hay triples".

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n; cin >> n;
    int count = 0;
    int data[n];
    for(int i = 0; i < n; i++){
        cin >> data[i];
        if(data[i] % 3 == 0) count++;
    }
    if(count == 0) { cout << "no hay triples"; return 0; }
    cout << count << '\n';
    for(int i = 0; i < n; i++){
        if(data[i] % 3 == 0) cout << i + 1 << ' ';
    }
}
```

Problema C: Dos Gatos y Un Raton

Descripción

Dos gatos y un ratón se encuentran en varias posiciones sobre una línea. Como valor de entrada se te darán sus posiciones iniciales. Tu tarea es determinar cuál gato alcanzará al ratón primero, asumiendo que el ratón no se mueve y ambos gatos se mueven a la misma velocidad.

Si ambos gatos llegan al mismo tiempo, el ratón podrá escapar mientras que los dos gatos pelean.

Entrada

Se te darán como valores de entrada tres enteros x, y y z separados por un espacio, los cuales representan respectivamente las posiciones del gato A, del gato B y del ratón C. Puedes suponer que $1 \leq x, y, z \leq 100$.

Salida

- Si el gato A alcanza primero al ratón, imprimir como salida "gato A".
- Si el gato B alcanza primero al ratón, imprimir como salida "gato B".
- Si ambos gatos alcanzan al ratón C al mismo tiempo, imprimir como salida ratón C".

Ejemplo

Entrada

```
1 2 3
```

Salida

```
gato B
```

Entrada

```
1 3 2
```

Salida

```
ratón C
```

Entrada

```
4 8 10
```

Salida

```
gato B
```

Entrada

```
8 3 12
```

Salida

```
gato A
```

Resolución

Analizar todas las posibles ubicaciones de los gatos respecto al ratón puede ser complejo. Es decir, analizar todos estos casos:

- A – B – C
- A – C – B
- B – A – C
- B – C – A
- C – A – B
- C – B – A

Una observación importante es que si desde el gato al ratón se demora X segundos, entonces desde el ratón a ese mismo gato se demorará lo mismo. Por lo que, en lugar de calcular la distancia de los gatos al ratón, es más simple calcular la distancia del ratón a los gatos.

Sea d_A la distancia del ratón al gato A, y d_B del ratón al gato B, entonces:

- Si $d_A == d_B$, ambos gatos llegan al ratón al mismo tiempo.
- Si $d_A < d_B$, el gato A llega antes.
- Si $d_A > d_B$, el gato B llega antes.

Implementación

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    int A, B, C;
    cin >> A >> B >> C;
    int da = abs(C - A), db = abs(C - B);

    if(da == db) cout << "raton C";
    else if(da < db) cout << "gato A";
    else cout << "gato B";
}
```

Implementación por Leonardo Valverde.

Problema D: Letras Faltantes

Descripción

Dada la palabra S que te escribieron y tu posible palabra P determina, si es posible, cuáles son las letras faltantes.

Entrada

En la primera línea dos números enteros positivos $0 < n, m < 50$, que indican la cantidad de letras en la palabra S y P respectivamente. En las siguientes dos líneas, la palabra S y la palabra P. Los caracteres que pueden tener S y P son: letras minúsculas, números comas y puntos.

Salida

Una línea que contenga las letras faltantes, recordar que las letras faltantes son aquellas que fueron omitidas mientras se escribían. Si la palabra P no puede ser generada agregando letras a S, entonces debes escribir "no entiendo"(sin comillas).

Ejemplo

Entrada

```
4 7
cptr
captura
1 4
b
abcd
5 7
ptir
captura
```

Salida

```
aua
acd
no entiendo
```

Resolución

Usar dos punteros, uno que itere la primera cadena (i) y el otro la otra cadena (j).

- Si $S[i] == P[j]$: aumentar ambos iteradores.
- Si $S[i] \neq P[j]$: considerar cadena $P[j]$ como faltante y aumentar el iterador j.

Si $i < n$, entonces una letra de S no pudo ser obtenida de P, por lo que se debe imprimir "no entiendo". Si $i == n$, mostrar los caracteres que se etiquetaron como faltantes.

Implementación

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    while (cin >> n >> m) {
        string S, P;
        cin >> S >> P;

        int i = 0, j = 0;
        string faltantes = "";

        while (j < m) {
            if (i < n && S[i] == P[j]) i++;
            else faltantes += P[j];
            j++;
        }

        if (i == n) cout << faltantes << '\n';
        else cout << "no entiendo\n";
    }
    return 0;
}
```

Implementación por Miguel Aimar.

Problema E: Binario a Decimal

Descripción

El sistema binario utiliza sólo dos dígitos: 0 y 1. El valor de cada posición se obtiene de una potencia de base 2, elevada a un exponente igual a la posición del dígito menos uno. Su popularidad radica en que es el utilizado por las computadoras y dispositivos electrónicos, internamente estos equipos usan el cero para inhibir y el 1 para generar impulsos eléctricos en su comunicación interna.

Entrada

Leer un número binario.

Salida

Imprimir el valor decimal resultante de la conversión del número ingresado.

Ejemplo

Entrada

```
101011
```

Salida

```
101011=43
```

Entrada

```
110011110000
```

Salida

```
110011110000=3312
```

Entrada

```
11011011001001111011
```

Salida

```
11011011001001111011=897659
```

Resolución

Para convertir un número binario a decimal, se puede usar el método de recorrer cada dígito del número binario y sumar el valor de cada dígito multiplicado por la potencia de 2 correspondiente a su posición.

- **Lectura del Número Binario:** Se lee un número binario como una cadena de texto.
- **Conversión a Decimal:** Se recorre cada dígito del número binario. Para cada dígito, se convierte el carácter a su valor numérico y se multiplica por la potencia de 2 correspondiente a su posición. Se acumulan los resultados de las multiplicaciones para obtener el valor decimal.
- **Impresión del Resultado:** Se imprime el número binario original junto con su valor decimal calculado.

Implementaciones

Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String d1 = sc.next();
        int acumul = 0;

        for (int i = 0; i < d1.length(); i++) {
            char un = d1.charAt(i);
            int ent = Character.getNumericValue(un);
            acumul = acumul * 2 + ent;
        }

        System.out.println(d1 + "=" + acumul);
    }
}
```

Implementación por Franco Quiquia.

C++: Manteniendo la potencia actual

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    string s; cin >> s;
    int ans = 0;
    int cur_pow = 1;
    for(int i = (int) s.size() - 1; i >= 0; i--){
        if(s[i] == '1') ans += cur_pow;
        cur_pow *= 2LL;
    }
    cout << s << '=' << ans << '\n';
}
```

C++: Usando Manipulacion de Bits (solamente manteniendo exponente)

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    string s; cin >> s;
    int n = (int) s.size();
    int ans = 0;
    for(int i = n - 1; i >= 0; i--){
        if(s[i] == '1') ans += (1 << (n - 1 - i));
    }
    cout << s << '=' << ans << '\n';
}
```

C++: Usando Bitset

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    string s; cin >> s;
    cout << s << '=' << bitset<32>(s).to_ulong() << '\n';
}
```

Implementación por Leonardo Valverde.

Problema F: Canchas V1

Descripción

El director te dará las coordenadas de las esquinas opuestas de cada cancha nueva y quiere que calcules cuál será el área total del patio que quedará cubierta por ambas canchas.

A pesar del tamaño del patio de la escuela, puede ser que ambas canchas puedan encimarse, por lo que deberás tomar eso en cuenta para no contar dos veces el área.

Entrada

Tu programa deberá leer del teclado dos líneas, cada una con 4 números enteros separados por un espacio que representan las coordenadas x_1, y_1, x_2, y_2 de cada una de las canchas.

Salida

Tu programa deberá escribir en la pantalla un único número entero que represente el área total cubierta por ambas canchas.

Ejemplo

Entrada

```
1 1 3 4
2 3 6 7
```

Salida

```
21
```

Resolución

Sean p_1, p_2 los puntos del primer rectángulo y q_1 y q_2 los puntos del segundo rectángulo. Los casos del problema garantizan que:

$$p_1.x < p_2.x \quad \text{y} \quad p_1.y < p_2.y$$

y lo mismo se cumple para q_1 y q_2. Si no se garantizaba esto, bastaba con hacer:

```
if(p1.x > p2.x) swap(p1.x, p2.x);
if(p1.y > p2.y) swap(p1.y, p2.y);
```

Del mismo modo con q_1 y q_2. Con esto se garantiza la relación mencionada.

La respuesta es $Area1 + Area2 - Interseccion$. Para las áreas, basta con calcular $base \times altura$:

$$\text{Area1} = (p2.x - p1.x) \times (p2.y - p1.y)$$

$$\text{Area2} = (q2.x - q1.x) \times (q2.y - q1.y)$$

- Para hallar la intersección, puede ser un poco más difícil. Para eliminar casos innecesarios, vamos a garantizar que $p1.x$ es el mínimo de todos los valores en el eje x.

- Es obvio que $p1.x < p2.x$ y que $q1.x < q2.x$, entonces para garantizar que $p1.x$ es el menor de todos, basta con evaluar $p1.x$ y $q1.x$. Si $p1.x > q1.x$, entonces intercambiamos todo el rectángulo:

```
if(p1.x > q1.x) { swap(p1, q1), swap(p2, q2); }
```

Luego los puntos en x pueden aparecer de los siguientes modos:

- $p1.x$ — $p2.x$ — $q1.x$ — $q2.x$
- $p1.x$ — $q1.x$ — $p2.x$ — $q2.x$
- $p1.x$ — $q1.x$ — $q2.x$ — $p2.x$

Sea dx la intersección en el eje x, entonces:

- Caso 1: $dx = 0$
- Caso 2: $dx = p2.x - q1.x$
- Caso 3: $dx = q2.x - q1.x$

Se pueden evaluar los 3 casos, pero un modo compacto de hacerlo es:

```
dx = max(0, min(p2.x, q2.x) - q1.x)
```

Hacemos algo similar para hallar la intersección en el eje y:

```
if(p1.y > q1.y) { swap(p1, q1); swap(p2, q2); }
dy = max(0, min(p2.y, q2.y) - q1.y);
```

Luego, $Interseccion = dx \times dy$.
Para la implementación es posible trabajar los puntos como pares, de modo que:

- `pair.first` = valor en eje x
- `pair.second` = valor en eje y

Implementación

```
#include <bits/stdc++.h>

#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif

using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    pair<int, int> p1, p2, q1, q2;
    cin >> p1.first >> p1.second;
    cin >> p2.first >> p2.second;
    cin >> q1.first >> q1.second;
    cin >> q2.first >> q2.second;

    int area = (p2.first - p1.first) * (p2.second - p1.second);
    area += (q2.first - q1.first) * (q2.second - q1.second);

    if(p1.first > q1.first) { swap(p1, q1); swap(p2, q2); }
    int dx = max(0, min(p2.first, q2.first) - q1.first);

    if(p1.second > q1.second) { swap(p1, q1); swap(p2, q2); }
    int dy = max(0, min(p2.second, q2.second) - q1.second);

    area -= dx * dy;

    cout << area << '\n';
}
```

Implementación

Implementación por Leonardo Valverde.

Problema F: Canchas V2

Resolución

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Leer coordenadas de las canchas
        int cancha1[] = new int[4];
        int cancha2[] = new int[4];
        int peri1, peri2, restan, total;
        for (int i = 0; i < 4; i++) {
            cancha1[i] = sc.nextInt();
        }
        for (int i = 0; i < 4; i++) {
            cancha2[i] = sc.nextInt();
        }

        // Calcular el area de cada cancha
        int x1 = cancha1[2] - cancha1[0];
        int y1 = cancha1[3] - cancha1[1];
        int x2 = cancha2[2] - cancha2[0];
        int y2 = cancha2[3] - cancha2[1];
        peri1 = x1 * y1;
        peri2 = x2 * y2;
        total = peri1 + peri2;

        // Calcular el area de superposicion si la hay
        int x[] = new int[4];
        int y[] = new int[4];
        x[0] = cancha1[0];
        x[1] = cancha1[2];
        x[2] = cancha2[0];
        x[3] = cancha2[2];
        y[0] = cancha1[1];
        y[1] = cancha1[3];
        y[2] = cancha2[1];
        y[3] = cancha2[3];
        Arrays.sort(y);
        Arrays.sort(x);
        restan = (x[2] - x[1]) * (y[2] - y[1]);

        // Verificar si hay superposicion
        if (cancha1[2] > cancha2[0] && cancha2[2] > cancha1[0]) {
            System.out.println(total - restan);
        } else {
            System.out.println(total);
        }
    }
}
```

Explicación del Código

- **Lectura de las Coordenadas:** Se leen dos conjuntos de cuatro números enteros, que representan las coordenadas de las esquinas opuestas de cada cancha.
- **Cálculo del Área de Cada Cancha:** Se calcula la longitud y la anchura de cada cancha utilizando las

diferencias entre sus coordenadas x e y. El área de cada cancha se calcula multiplicando la longitud por la anchura.

- **Cálculo del Área Total:** Se suman las áreas de las dos canchas para obtener el área total cubierta, sin considerar aún la posible superposición.
- **Cálculo del Área de Superposición:** Para determinar el área de superposición, se identifican las coordenadas de las posibles áreas de intersección y se ordenan. La longitud y la anchura de la superposición se calculan tomando la diferencia entre las coordenadas ordenadas.
- **Verificación de la Superposición:** Se verifica si hay una intersección entre las dos canchas comprobando si las coordenadas se solapan. Si hay superposición, se resta el área de superposición del área total. Si no hay superposición, se mantiene el área total sumada inicialmente.
- **Salida:** Se imprime el área total cubierta por ambas canchas, teniendo en cuenta la superposición si la hay.

Implementación

Implementación por Franco Quiquia.