Auxiliatura INF-143 “A”

Paradigmas en la Resolución de Problemas

Aux. Miguel Angel Quispe Mamani

Universidad Mayor de San Andrés

Carrera de Informática

I/2022

1 Prueba de Primalidad

El primer algoritmo presentado es para determinar si un numero natural N es primo. La version mas simple es probar por definición, es decir, probar que N solo tenga 2 divisores 1 y N, para hacer esto tendremos que dividir N veces. Esta no es la mejor forma para probar si un numero N es primo y hay varias posibles mejoras.
El primer algoritmo seria:
Java

```java
  public static boolean is_prime(int n) {
    int counter = 0;
    for(int i = 1; i <= n; i++)
      if(n % i == 0)
        counter++;
    return counter == 2;
  }
```

C++

```cpp
bool is_prime(int n) {
    int counter = 0;
    for(int i = 1; i <= n; i++)
        if(n % i == 0)
            counter++;
    return counter == 2;
}
```

La primera mejora es verificar si N es divisible por divisores $\in [2, ..\sqrt{N}]$, es decir, paramos de verificar cuando el divisor sea mas grande que $\sqrt{N}$.

La segunda mejora es verificar si N es divisible por divisores $\in [3, 5, 7, ..\sqrt{N}]$, es decir, solo verificaremos números impares hasta $\sqrt{N}$. Esto es porque solo hay un numero primo par, el numero 2, que puede ser verificado por separado.

El código es el siguiente:
Java

```java
public static boolean is_prime(int n) {
  if(n == 1)
    return false;
  if(n == 2)
    return true;
  if(n % 2 == 0)
    return false;
  for(int i = 3; i * i <= n; i += 2)
    if(n % i == 0)
      return false;
  return true;
}
```

C++

```cpp
bool is_prime(int n) {
    if(n == 1)
        return false;
    if(n == 2)
        return true;
    if(n % 2 == 0)
        return false;
    for(int i = 3; i * i <= n; i += 2)
        if(n % i == 0)
            return false;
    return true;
}
```

2 Criba de Eratóstenes

Si queremos generar una lista de números primos en rango de $[0..N]$, existe un mejor algoritmo que probar si cada numero en el rango es primo o no. Este algoritmo es llamado *“Criba de Eratóstenes”* inventado por Eratóstenes de Alexandria. Esto funciona de la siguiente manera:

- Primero, hacemos que todos los números en el rango sean *probablemente primos*, pero hacemos que los números 0 y 1 no sean primos.
- Luego, tomamos al 2 como primo marcamos todos los múltiplos de 2 empezando por $2 + 2 = 4$, 6, 8, 10,... hasta que el múltiplo sea mas grande que N.
- Luego tomamos el siguiente numero no marcado como primo que en este caso seria el 3 y marcamos todos los múltiplos de 3 empezando por $3 + 3 = 6$, 9, 12,...
- Luego tomamos a el siguiente número no marcado que en este caso seria el 5 y marcamos todos los múltiplos de 5 empezando por $5 + 5 = 10$, 15, 20,... y así sucesivamente hasta la raiz cuadrada de N.
- Después de esto cualquier numero no marcado dentro del rango $[0..N]$ sera primo.

El código seria el siguiente:
Java

```java
  public static int N = 1000000 + 10;
  public static boolean sieve[] = new boolean[N];
  public static void make_sieve() {
    //true primo; false no primo
    //Hacemos a todos probablemente primos
    for(int i = 0; i < N; i++)
      sieve[i] = true;
    sieve[0] = sieve[1] = false; //0 y 1 no son primos
    for(int i = 2; i * i <= N; i++)
      if(sieve[i] == true)
        for(int j = i + i; j < N; j += i)
          sieve[j] = false;
  }
```

C++

```cpp
const int N = 1000000 + 10;
bool sieve[N];
void make_sieve() {
    //true primo; false no primo
    //Hacemos a todos probablemente primos
    for(int i = 0; i < N; i++)
        sieve[i] = true;
    sieve[0] = sieve[1] = false; //0 y 1 no son primos
    for(int i = 2; i * i <= N; i++)
        if(sieve[i] == true)
            for(int j = i + i; j < N; j += i)
                sieve[j] = false;
}
```

3 Sumas en Subsegmento(Acumuladas)

Supongamos que dado:

- Un número entero positivo $n \leq 10^5$.
- Un vector a de n números enteros a_0, a_1,..., a_{n-1}, no exceden el valor absoluto de 10^9.
- Un número entero positivo $q \leq 10^5$.
- q consultas, cada uno de ellos consta de números enteros l, r, $(0 \leq l \leq r \leq n-1)$.

Para cada consulta, debemos imprimir una suma:

- $a_l + a_{l+1} + a_{l+2} + ... + a_r$

La solución “ingenua” funciona en $O(n*q)$ que es demasiado lenta, así se sería el algoritmo:
Java

```java
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int n = in.nextInt();
    int a[] = new int[n];
    for (int i = 0; i < n; i++)
      a[i] = in.nextInt();
    int q = in.nextInt();
    while (q-- > 0) {
      int l = in.nextInt();
      int r = in.nextInt();
      long ans = 0;
      for(int i = l; i <= r; i++)
        ans += a[i];
      System.out.println(ans);
    }
  }
```

C++

```cpp
int main(){
    int n, q;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cin >> q;
    while(q--){
        int l, r;
        cin >> l >> r;
        long long ans = 0;
        for(int i = l; i <= r; i++)
            ans += a[i];
        cout << ans << "\n";
    }
    return 0;
}
```

Construya un vector **acc**, donde $acc[i] = a_0 + a_1 + a_2 + ... + a_i$, $i = 0, 1, ..., n - 1$.

Si se calculan estas sumas parciales, entonces para cualquier l, r se cumple lo siguiente:

$$sumaParcial(l, r) = \begin{cases} acc[r] & \text{si } l = 0 \\ acc[r] - acc[l-1] & \text{si } l > 0 \end{cases} \quad (1)$$

Entonces, la respuesta para cada consulta se puede encontrar en $O(1)$, si se calculó el vector **acc**.

La solución optimizada tendría una complejidad de $O(n + q)$ y el código sería:

Java

```java
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int n = in.nextInt();
    int a[] = new int[n];
    for (int i = 0; i < n; i++)
      a[i] = in.nextInt();
    long acc[] = new long[n];
    acc[0] = a[0];
    for (int i = 1; i < n; i++) {
      acc[i] = a[i];
      acc[i] += acc[i - 1];
    }
    int q = in.nextInt();
    while (q-- > 0) {
      int l = in.nextInt();
      int r = in.nextInt();
      if(l == 0)
        System.out.println(acc[r]);
      else
        System.out.println(acc[r] - acc[l - 1]);
    }
  }
```

C++

```cpp
int main(){
    int n, q;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    long long acc[n];
    acc[0] = a[0];
    for (int i = 1; i < n; i++) {
        acc[i] = a[i];
        acc[i] += acc[i - 1];
    }
    cin >> q;
    while(q--){
        int l, r;
        cin >> l >> r;
        if(l == 0)
            cout << acc[r] << "\n";
        else
            cout << acc[r] - acc[l - 1] << "\n";
    }
    return 0;
}
```