Daniel Alejandro Osornio López
0244685@up.edu.mx
Universidad Panamericana
Matemáticas de la Computación

Actividad 2.3: Algoritmo de División

Específicamente, se trata de la implementación de división euclideana, que dice [1]:

> *Dados dos números $a, b \in \mathbb{Z}$ donde $b \neq 0$, existen dos números* ***únicos*** *$q, r \in \mathbb{Z}$ tal que*
> - $a = bq + r$
> - $0 \leq r < |b|$

Es importante recalcar que el algoritmo siempre dará un valor r mayor o igual a 0, esto es diferente a lo que realiza el operador `%` en muchos lenguajes, que devuelve el *remanente*, un valor que puede ser negativo.

Es así que lenguajes como Rust dan ambas posibilidades, con el operador `%` para obtener el *remanente* y funciones a parte como `rem_euclid()` para obtener el *modulo*:

- `println!("{}", -11 % 3); // -2`
- `println!("{}", (-11_i8).rem_euclid(3)); // 1`

La implementación dada en clase, escrita en `rust` es del tipo «por substracción repetida», donde en un bucle restamos el valor de b, obteniendo en el proceso cuántas veces cabe en a en el valor de q.

```rust
fn div(a: i64, b: i64) -> (i64, i64) {
    let (mut q, mut r) = (0, a.abs());
    if a == 0 { return (0, 0); }
    while r >= b {
        r -= b;
        q += 1
    }
    if a > 0 { }
    else if r == 0 { q = -q; }
    else {
        q = -q - 1;
        r = b - r;
    }

    (q, r)
}
```

Este algoritmo maneja bien casos, con $a, b \in \mathbb{Z}^+$, donde la entrada es `(0, b)`, `(-a, _)`, `(a, b)`. Sin embargo, el algoritmo se queda atrapado en un bucle infinito en casos de la forma `(_, -b)`.

Por esta razón decidí buscar una implementación más completa del algoritmo [2], la versión resultante es:

```rust
fn divide(n: i64, d: i64) -> (i64, i64) {
    match (n, d) {
        (_, 0) => panic!(),
        (_, d) if d < 0 => {
            let (q, r) = divide(n, -d);
            (-q, r)
        }
        (n, _) if n < 0 => {
            let (q, r) = divide(-n, d);
            if r == 0 {
                (-q, 0)
            } else {
                (-q - 1, d - r)
            }
        }
        (_, _) => divide_unsigned(n, d),
    }
}
```

```rust
fn divide_unsigned(n: i64, d: i64) -> (i64, i64) {
    let (mut q, mut r) = (0, n);
    while r >= d {
        q += 1;
        r -= d;
    }
    (q, r)
}
```

Puede ejecutar este código en línea en la liga: https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=3e3e5cf05cafcef5006460f1449015f3. El algoritmo resultante:

- En los casos donde se intente dividir entre 0, el programa *crashea*.
- En casos donde a o b sean negativos se aprovecha la llamada recursiva de la función para re-ajustar por valores positivos los valores de a, b de forma que se haga siempre entre valores positivos en `divide_unsigned`, cuando se obtiene el resultado se re-interpreta el para reflejar los valores correctos de q, r.

Es asi que se llama una y otra vez a si misma la función hasta tener dos a, b positivos, y al final se re-interpreta.

Bibliografía

[1] "Euclidean division," 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Euclidean_division&oldid=1162754895

[2] "Division algorithm, division by repeated subtraction," 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Division_algorithm&oldid=1169877646#Division_by_repeated_subtraction