ARITMÉTICA MODULAR

PECL3

Juan Casado Ballesteros

Álvaro Zamorano Ortega

Gabriel López Cuenca

TRABAJO REALIZADO

Se han implementado todos los requisitos establecidos en la documentación de la práctica:

- **Reducción a representante canónico**: dado un entero y un módulo expesamos dicho entero en el módulo indicado.
- **Suma de dos números en Zp**: dados dos números entero y su módulo se devuelver el resultado de la operación.
- **Multiplicación de dos números en Zp**: dados dos números entero y su múdulo se devuelver el resultado de la operación.
- **Decisión sobre la inversibilidad**: dado un número y un módulo podemos indicar si tendrá o no un inverso.
- **Cálculo del inverso**: dado un número y un módulo calcularemos su inverso en el caso de que lo tenga
- **Suma de matrices en Zp**: dadas dos matrices y un módulo se proporciona el resultado.
- **Multiplicación de matrices en Zp**: dadas dos matrices y un módulo se proporciona el resultado.
- **Rango de una matriz en Zp**: dada un matriz y un módulo se raliza el cálculo.
- **Decisión sobre la inversibilidad de matrices**: dada una matriz y un módulo determinamos si tendrá o no una matriz inversa.
- **Calculo del inverso de una matriz**: dada una matriz y un módulo calculamos su matriz inversa en el caso de que la tuviera.
- **Potencia de matrices en Zp con algoritmo binario**: se utiliza el algoritmo indicado para calcular la potencia n de una matriz en el módulo dado.

Para algunas de las funciones en álgebra matricial en Zp ha sido necesario crear otras como calcular el determinante, calcular la matriz adjunta y calcular la matriz transpuesta todas en Zp.

A continuación se detallan algunos detalles de la implementación.

ARITMÉTICA MODULAR

El trabajo en aritmética modular ha sido simple gracias a que los números enteros estaban ya definidos junto con sus operaciones, así como la función *restoent* mediante la cual se calcula el representante canónico.

Para el cálculo del inverso se utiliza una función recursiva que comprueba los desde el módulo hasta cero si es resultado es el inverso proporcionándolo en caso de que lo sea. En caso de que no haya un inverso para ese número en ese módulo se indicará mediante el texto “** Numero sin inverso **”.

Todas las funciones han sido implementadas utilizando las sintaxis del cálculo lamda de modo que es sencillo componer unas funciones a partir de otras o realizar operaciones de curring pudiendo reutilizar el código creado en esta fase de forma cómoda y sencilla a lo largo de la práctica.

MATRICES EN Z_p

Debido a que las matrices están siempre en la forma 2x2 los cálculos se han hecho sobre posiciones fijas de modo que las funciones creadas no son escalables a matrices nxn.

Se utilizan la función de matriz para ensamblar las matrices en un par de pares resultado cuando el resultado de la operación matricial es otra matriz.

Las operaciones son directas en todas las funciones propuestas excepto en el cálculo de potencias con el algoritmo binario del que se hablará más adelante. Para realizar algunas de las operaciones propuestas ha sido necesario crear previamente algunas funciones típicas del álgebra matricial como el determinante, la transpuesta o al adjunto.

RECURSIÓN

Para realizar la recursión no se ha utilizado el operador Y.

En su lugar se ha hecho que cada función sea su propio operador de punto fijo. En el cálculo lambda una función no puede ser recursiva, pero puede hacerse que lo sea. No puede serlo pues una función no puede utilizarse hasta que no ha sido definida completamente y por tanto mientras se está definiendo una función esta no podrá utilizarse a si misma. No obstante, nada impide que una función se reciba a si misma como parámetro pues para cuando dicha llamada sea realizada ella ya habrá sido definida y por tanto podrá hacerse que sea recursiva.

Cuando hemos necesitado utilizar recursión hemos empleado este sencillo *truco*.

No obstante, en scheme no hubiera sido necesario pues ya simula recursión y ni siquiera hubiera sido necesaria en sus versiones más puristas donde esta está limitada bajo la premisa de que cuando el compilado evalúe un término este haya tenido que ser definido previamente por completo pues el compilador nunca llegará a realizar dicha evaluación ya que en nuestro caso nuestra llamada recursiva estaba encapsulada por los parámetros lambda que permiten el curring.