Universidad Nacional Autónoma de México

Análisis de Algoritmos
Tarea 3

Luis Sebastián Arrieta Mancera (318174116)
Zuriel Enrique Martínez Hernández (318056423)
Dania Paula Góngora Ramírez (318128274) Diana Laura Salgado Tirado (421110274)

30 de mayo de 2023

1. Ejercicios

1. Ejercicio 1

$$\frac{n!}{(r!(n-r)!)} \quad \text{mód } p$$

La complejidad en tiempo de tu algoritmo debe ser: $O(p + log(p * n))$
La complejidad en espacio de tu algoritmo debe ser: $O(p)$

Algóritmo:

- Si r es mayor que n regresamos 0 como resultado.
- Si n-r es mayor que r, establecemos r como n-r para calcular la formula más pequeña posible.
- Usamos la factorización de números primos para encontrar los menores de 1 a n.
- Utilizamos un diccionario para almacenar la potencia de cada primo en la factorización del numerador de n y r.
- Hacemos una división para obtener los factores primos y actualizamos el diccionario con la frecuencia de cada primo.
- Hacemos lo mismo para el denominador del coeficiente binomial, pero restamos la potencia de cada primo.
- Calculamos el resultado final tomando en cuenta la congruencia modular.

- Multiplicamos cada primo por su potencia correspondiente y se toma el módulo p en cada paso.

Pseudocódigo:

```
fun Ejercicio1(n, r, p):
// Caso base
si r > n:
    retornar 0

// r como n-r para calcular la formula mas corta posible.
si n - r > r:
    r = n - r

// encontrar los menores factores primos de 1 a n.
factores_primos = [0] * (n + 1)
factores_primos[1] = 1

   para i en rango(2, n + 1):
    si factores_primos[i] == 0:
        factores_primos[i] = i
        para j en rango(i * i, n + 1, i):
            si factores_primos[j] == 0:
                factores_primos[j] = i

// diccionario para almacenar la potencia de cada primo en la factorizacion
potencias_primos = {}

  para i en rango(r + 1, n + 1):
    t = i

    // division para obtener los factores primos
    mientras t > 1:
        si factores_primos[t] no esta en potencias_primos:
            potencias_primos[factores_primos[t]] = 1
        sino:
            potencias_primos[factores_primos[t]] += 1
        t //= factores_primos[t]

// restamos la potencia de cada factor primo
para i en rango(1, n - r + 1):
    t = i

          mientras t > 1:
        potencias_primos[factores_primos[t]] -= 1
        t //= factores_primos[t]

respuesta = 1

// Calculamos cada modulo
para i en potencias_primos:
    respuesta = (respuesta * pow(i, potencias_primos[i], p)) % p

regresamos respuesta
```

Explicación del algoritmo paso a paso:

- Lo primero que se hace es verificar los valores de r y n, si sucede que r es mayor que n se regresa 0 ya que no es posible calcular el coeficiente binomial en este caso.
- En el siguiente paso si $n-r$ es mayor que r entonces se intercambia el valor de r por $n-r$, se hace para obtener la fórmula más pequeña posible, esto se puede hacer ya que el coeficiente binomial es simétrico,es decir $C(n,r)$ es igual a $C(n,n-r)$
- Se utiliza la factorización de números primos para descomponer a n y r en factores primos. recordemos que la factorización de números primos consiste en descomponer un numero en la multiplicación de sus factores primos. (por ejemplos $n=15$ se descompone en $15=3^1*5^1$)
- Después utilizamos un diccionario para almacenar la potencia de cada primo en la factorización del paso anterior, las claves serian los factores primos y los valores asociados las potencias correspondientes. Por ejemplo si $n=10$ y $r=3$, la factorización de números primos sería $n=2^1*5^1 yr=3^1$. El diccionario se vería así: $2:1,5:1,3:1$.
- A continuación en los siguiente dos pasos hacemos la división para obtener los factores primos,la idea principal es cancelar los factores comunes en el numerador y denominador del coeficiente binomial.
  Por ejemplo, si tenemos $n=10$ y $r=3$, y hemos realizado la factorización de números primos $n=2^1*5^1 yr=3^1$, el diccionario inicial sería $2:1,5:1,3:1$.

  Al hacer la división para obtener los factores primos del numerador $(n!)$, se mantiene el diccionario sin cambios, pues todos los factores primos de n están en el numerador.

  Pero al realizar la división para obtener los factores primos del denominador $(r!(n-r)!)$, se resta la potencia de cada primo en el diccionario. es decir, el primo 3 tiene una potencia de 1 en el numerador, pero no está presente en el denominador. Por lo tanto, se resta 1 en la potencia correspondiente en el diccionario, quedando el diccionario actualizado como $2:1,5:1,3:0$.

  Este paso de división y actualización del diccionario garantiza que se cancelen correctamente los factores comunes entre el numerador y el denominador, para lograr el cálculo correcto del coeficiente binomial.
- Ahora para el cálculo del resultado final del coeficiente binomial, tomando en cuenta la congruencia modular. Se multiplican los primos por su potencia correspondiente y se toma el módulo p en cada paso. El resultado final es el valor obtenido después de todas las multiplicaciones y módulos.

El algoritmo funciona ya que se basa en la propiedad de que el coeficiente binomial se puede calcular dividiendo los factoriales de n y r por el factorial de (n-r). Por lo tanto al realizar la factorización de números primos y utilizar el diccionario para almacenar las potencias de los primos, se asegura un cálculo correcto del coeficiente binomial. Además, al aplicar la congruencia modular en cada paso, se evitan los cálculos innecesarios y se obtiene el resultado en forma de residuo. Esto es muy útil cuando el número p es grande y necesitamos que el resultado sea manejable.

2. Ejercicio 2

Desarrolla un segundo algoritmo con el problema anterior que tenga la siguiente complejidad:

La complejidad en tiempo de tu algoritmo debe ser: $O(n)$

La complejidad en espacio de tu algoritmo debe ser: $O(1)$

Preámbulo:

Nos piden calcular lo siguiente:

$$\frac{n!}{(r!(n-r)!)} \quad \text{mód } p$$

Sabemos que el cálculo de factoriales en computación es una operación que puede volverse bastante costosa para valores muy grandes. Es por esto que hay que encontrar una manera más eficiente de computar esto. Podemos utilizar el `Teorema de Lucas` para reducir la complejidad de este problema.

El `Teorema de Lucas` caracteriza el residuo del coeficiente binomial $\binom{n}{k}$ cuando es dividio por un número primo p. Fue enunciado por primera vez en 1878 en una publicación del matemático *Édouard Lucas*.

Enunciado

Sean n y r números enteros no negativos y p un número primo. Entonces, tenemos la siguiente relación de congruencia:

$$\binom{n}{r} \equiv \prod_{i=0}^{k} \binom{n_i}{r_i} \pmod{p}$$

Donde

$$N = n_0 + n_1 p + n_2 p^2 + ... + n_k p^k$$
$$R = r_0 + r_1 p + r_2 p^2 + ... + r_k p^k$$

Es decir N y R son la descomposición polinómica (la expresión de un número que se escribe como la suma de los valores relativos de cada una de sus cifras) de n y r en base p respectivamente. Se utiliza por convención que $\binom{n_i}{k_i} = 0$ si $n_i < k_i$.

Ejemplo

Consideremos $n = 15$, $r = 6$ y $p = 3$ para computar

$$\frac{n!}{(r!(n-r)!)} \quad \text{mód } p$$

Sabemos que $\frac{n!}{(r!(n-r)!)} = \binom{n}{r}$ por lo cual podemos aplicar el *Teorema de Lucas* comenzando por convertir n y r en base p.

$$\begin{array}{r|l} 3 & 15 \\ \hline & 5 \\ & 15 \\ \hline & 0 \end{array} \quad \Rightarrow \quad \begin{array}{r|l} 3 & 5 \\ \hline & 1 \\ & 3 \\ \hline & 2 \end{array} \quad \Rightarrow \quad \begin{array}{r|l} 3 & 1 \\ \hline & 0 \end{array}$$

Consideramos los residuos del último al primero para definir $15_3 = 120$

$$\begin{array}{r|l} 3 & 6 \\ \hline & 2 \\ & 6 \\ \hline & 0 \end{array} \quad \Rightarrow \quad \begin{array}{r|l} 3 & 2 \\ \hline & 0 \end{array}$$

Consideramos los residuos del último al primero para definir $6_3 = 20$. Ahora así como podemos hacer la descomposición polinómica de un número en base decimal....

$$86293 = 8 \cdot 10^4 + 6 \cdot 10^3 + 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$$

podemos hacer la descomposición polinómica en base p. Recordemos que para hacer la descomposición polinómica de un número se debe multiplicar cada cifra del número por 10 elevado a la cantidad de cifras que tiene a la derecha.

$$15_3 = 120 = 1 \cdot 3^2 + 2 \cdot 3^1 + 0 \cdot 3^0$$
$$6_3 = 20 = 0 \cdot 3^2 + 2 \cdot 3^1 + 0 \cdot 3^0$$

De manera que ya tenemos nuestros n_i, r_i así que procedemos a aplicar el *Teorema de Lucas*, donde

$$k = max(|n_i|,|r_i|) \text{ - } 1.$$

$$\binom{15}{6} \equiv \prod_{i=0}^{2} \binom{n_i}{r_i} \pmod 3 = \binom{n_0}{r_0} \cdot \binom{n_1}{r_1} \cdot \binom{n_2}{r_2} \pmod 3$$

$$= \binom{0}{0} \cdot \binom{2}{2} \cdot \binom{1}{0} \pmod 3$$

$$= 1 \cdot 1 \cdot 1 \pmod 3$$

$$= 1 \pmod 3$$

De manera que $\binom{15}{6} \equiv 1 \pmod 3$. Recordemos que $a \equiv b \pmod c$ quiere decir que a y b tienen el mismo residuo cuando son divididos por c, esto es $a \bmod c = b \bmod c$. Por lo tanto en nuestro ejemplo

$$1 \bmod 3 = \binom{15}{6} \bmod 3$$

Es más fácil computar $1 \bmod 3$ que $\binom{15}{6} \bmod 3$. Finalmente como $1 \bmod 3 = 1$ entonces

$$\binom{15}{6} \bmod 3 = 1$$

Podemos comprobarlo de la siguiente manera:

$$\binom{15}{6} = \frac{15!}{6! \cdot (15-6)!} = 5005 \Rightarrow 5005 \bmod 3 = 1$$

Ya que $5005 = k \cdot 3 + 1 \Rightarrow k = \frac{5004}{3} = 1668 \Rightarrow 5005 = 1668 \cdot 3 + 1$

Algoritmo

1. Si r es mayor que n regresamos 0 como resultado
2. Si $p \geq 50$ regresamos una excepción y el algoritmo termina
3. Si p no es un factor primo regresamos una excepción y el algoritmo termina
4. Convertimos n y r en base p y guardamos los dígitos en sus correspondientes arreglos, los espacios vacios los complementamos con 0's
5. Realizamos el coeficiente binomial por cada uno de los dígitos de derecha a izquierda de n_p y r_p (n base p y r base p)
6. Regresamos la evaluación del resultado del paso anterior con modulo p

- Siendo un poco más específicos en algunos aspectos del algoritmo y en su complejidad tanto en tiempo como en espacio, primero tenemos que el ayudante nos dio oportunidad de hacer este ejercicio 2 con espacio O(n) pero forzosamente tendremos que programarlo así que en esta parte desarrollaremos más a detalle el algoritmo y después procederemos a programarlo.
- Nos piden que p sea un factor primo y que no sea mayor a 50, por lo tanto la verificación $p50$ lo hacemos en tiempo constante, mientras que el determinar que p sea un factor primo es una verificación un poco más costosa. Como ya sabemos cuales son los factores primos menores a 50 esta verificación la podemos hacer utilizando un hashmap para guardar estos factores: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, donde la llave serán ellos mismos, de esta manera podemos verificar en tiempo constante si p cumple con esta condición pero con espacio $O(15) \in O(1)$.
- Posteriormente tenemos que convertir las entradas n y r en base p. Esto se hace de igual manera que como se mostró en el ejemplo anterior, dividimos $n = k$ entre p, guardamos el residuo en un arreglo correspondiente a n_p y el cociente tomara el lugar de k, repetimos este proceso hasta que el cociente sea cero. Análogamente hacemos lo mismo con r. Pero como $r < n$ entonces la cantidad de dígitos de n_p será menor o igual a r_p por lo tanto, completamos el arreglo de r_p con ceros hasta que tenga el mismo tamaño que el arreglo correspondiente a n_p si es necesario. En esta parte estamos utilizando espacio de $O(n)$
- Iteramos cada uno de los arreglos de n_p, r_p con dos apuntadores i,j respectivamente. Realizamos el producto del coeficiente binomial de cada $\binom{n_p[i]}{r_p[j]}$ y vamos acumulando el resultado en una variable. Estas operaciones tienen un costo computacional bastante bajo en comparación con el el coeficiente binomial de n y r.
- Finalmente realizamos realizamos la operación de la variable que utilizamos para acumular nuestro resultado del producto anterior con el modulo de p. De esta manera y como ya se explicó anteriormente habremos calculado $\binom{n}{r} \bmod p$ de una manera más eficiente y con tiempo y espacio $O(n)$. El tiempo se justifica de esta manera ya que iteramos por cada dígito del arreglo correspondiente a la entrada n.