UNIVERSIDAD
DE SANTIAGO
DE CHILE

Departamento de Matemática y Ciencia de la Computación

Tarea 2

Verificación de programas

Primer Semestre 2021

Lógica Computacional 22625
Licenciatura en Ciencia de la Computación

Rodrigo Kobayashi Araya
rodrigo.kobaayshi@usach.cl

1 Introducción

El objetivo de este trabajo es definir precondiciones, cabecera de función o procedimiento, postcondición y correctitud para dos problemas: La interpolación de lagrange y la generación de la matriz de opinion colectiva a partir de una permutación de un conjunto de números y un número de particiones definidas.

Para esto, se usará la verificación formal de programas, la cual consta de la descripción formal de las especificaciones de las funciones o procedimientos, tanto de entrada como de salida. Dichas descripciones formales son denominadas precondiciones y postcondiciones.

Al definir precondiciones y postcondiciones lo suficientemente fuertes, se minimiza el espacio para la mala interpretación de la funcion o el procedimiento, de esta manera hay menos espacio para errores y facilita el mantenimiento, extensión y reutilización del código.

Además, se requiere del cálculo de correctitud, que dada las precondiciones y postcondiciones de un programa, se calculan las condiciones intermedias de este. De esta manera, se calcula la correctitud del programa en su totalidad.

2 Método de Interpolación de Lagrange

El método de interpolación de lagrange se basa en encontrar un polinomio que, dado un conjunto de puntos, pase por todos estos. de esta manera se pueden aproximar nuevos puntos que no pertenecen al conjunto original. Además, entre mayor sea la cardinalidad del conjunto de puntos, mas exacta es la aproximación.

Dado un conjunto de $n+1$ puntos (x_i, y_i), para este caso, el polinomio de interpolación de Lagrange, de grado $g \leq n$, que pasa por todos los $n+1$ puntos está dado por:

$$P_n(x) = \sum_{i=0}^{n} L_i(x) y_i$$

Donde $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$ son los $n+1$ puntos pertenecientes al conjunto, y L_i está dado por:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_i}{x_i - x_j}$$

De esta manera, solo basta con ingresar el x deseado para obtener su evaluación. A continuación se definirán las precondiciones, cabecera de función y correctitud para el algoritmo que, ingresado un conjunto de puntos de largo $n+1$ y un x, retorne la evaluación de x.

2.1 Procedimiento

Se define un largo máximo de mil puntos. Además de esto, para el funcionamiento del algoritmo, se necesita saber la cantidad de puntos a ingresar, el conjunto de puntos y el número por evaluar en el polinomio. Se decide usar dos arrays para las abscisas y ordenadas. De esta manera:

$$\{Q \equiv 1 \leq n \leq 999\}$$
$$\textbf{func } Pn(n : float, xi : array, yi : array, x : float) \textbf{ return}(y : int)$$
$$\{R \equiv (\forall i, j \in \{0, ..., n\}\ \exists y \in \mathbb{R}\ .\ y = \sum_{i=0}^{n} L_i(x) y_i[i])\}$$

Ya que se decide hacer una funcion auxiliar para calcular L_i, se tiene:

$$\{Q \equiv 1 \leq n \leq 999\}$$
$$\textbf{func } \text{Li(n: float, i: int, xi: array, x: float)}\textbf{return}\text{(r: float)}$$
$$\{R \equiv (\forall j \in \{0, ..., n\}\ .\ r = \prod_{j=0, j\neq i}^{n} \frac{x - x_i[i]}{x_i[i] - x_i[j]})\}$$

2.2 Implementación

Como se menciona anteriormente, necesitamos definir una función que calcule $L_i(x)$, para luego usarla para calcular el polinomio P_n

```
1  float Li(int n, int i, float xi[n+1], float x)
2  {
3      int j = 0;          /*Se define j, para iterar sobre los valores de xi*/
4      float product = 1;  /*Se define el producto que sera retornado como Li*/
5      while (j<=n)
6      {
7          if(j != i)          /*para todo j != i*/
8          {
9              product = product*(x - xi[j])/(xi[i] - xi[j]);
10             /*   Li = PROD((x-xj)/(xi-xj) , j=0, j!= i, n)   */
11         }
12         j = j+1;
13     }
14     return product;
15 }
```

Luego de esto, implementamos la función para la evaluación de x en el polinomio P_n, con una lógica similar a la anterior, ya que tanto las sumatorias como las productorias se implementan de la misma manera, cambiando la suma por la multiplicación.

```
1  float Pn(int n, float xi[n+1], float yi[n+1], float x)
2  {
3      int i;            /*se define i, para iterar sobre los puntos*/
4      float result = 0; /*Se define result, que representa la evaluacion Pn(x)*/
5      while (i <= n)
6      {
7          result = result + Li(n, i, xi, x)*yi[i];
8          /*   y = SUM( Li(x)yi , i = 0, n)   */
9          i = i+1;
10     }
11     return result;
12 }
```

2.3 Correctitud

A continuación se desarrollará la correctitud para las funciones descritas anteriormente.

Correctitud para la función Li

1 $\{Q \equiv 1 \leq n \leq 999\}$
2
3 `Asignacion: j = 0`
4 `Asignacion: product = 1`
5 $(j \leq n)$`, j incrementa por la operacion de la linea 12`
6
7 $j \neq i$
8
9 $(\forall j \in \{1, ..., n\} \ . \ j \neq i \rightarrow product = \prod_{j=0, j \neq i}^{n} \frac{x - x_i[i]}{x_i[i] - x_i[j]})$
10
11
12 `Asignacion:` $j = j + 1$
13
14 $product \equiv (\forall j \in \{0, ..., n\} \ . \ r = \prod_{j=0, j \neq i}^{n} \frac{x - x_i[i]}{x_i[i] - x_i[j]})$

Correctitud para la función Pn

1 $\{Q \equiv 1 \leq n \leq 999\}$
2
3 `Asignacion: i = 0`
4 `Asignacion: result = 0`
5 $i \leq n$`, i incrementa por la operacion de la linea 9`
6
7 $\forall i \in \{1, ..., n\} \ . \ result = \sum_{i=0}^{n} L_i(x) y_i[i]$
8
9 `Asignacion: i = i+1`
10
11 $\{result \equiv (\forall i, j \in \{0, ..., n\} \ \exists y \in \mathbb{R} \ . \ y = \sum_{i=0}^{n} L_i(x) y_i[i])\}$

3 Matriz de opinión colectiva

Dado un número natural n, es posible particionar un conjunto de n números naturales $\{d_1, d_2, d_3, ..., d_n\}$ en p particiones $(2 \leq p \leq n-1)$ y luego construir la matriz de opinión colectiva Y. El conjunto $\{d_1, d_2, d_3, ..., d_n\}$ es una permutación del conjunto $\{1, 2, 3, ..., n\}$.

La matriz Y es una matriz de orden nxn que se construye del siguiente modo:

$$Y_{ii} = 1, 1 \leq i \leq n$$

$Y_{ij} = 1$, si i y j están en la misma partición. En caso contrario $y_{ij} = 0$.

Por ejemplo, sea $n = 5$, $p = 3$, la partición $1, 1, 3$ y la permutación $4, 5, 1, 2, 3$. La partición y la permutación genetan:

$$\{4\}, \{5\}, \{1, 2, 3\}$$

La matriz de opinión colectiva es:

$$Y = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

3.1 Procedimiento

Se define un largo máximo de mil números, por lo que n puede ir desde 1 a 1000 y p de 2 a 999. Ya que necesitamos encontrar la posición de un número dentro de un conjunto, se decide hacer una función auxiliar que haga lo descrito anteriormente.

Una forma de encontrar las particiones es mediante la suma de estas, el rango de las particiones se puede definir como la suma de las particiones anteriores como limite inferior y esta suma mas la cardinalidad de la partición en la cual se está trabajando como superior. De esta manera, para el conjunto $\{d_1, d_2, ..., d_{n-1}\}$ y el conjunto de particiones $\{p_1, p_2, ..., p_n\}$:

$$\forall a \in \{1, ..., n\} \ . \ d_a \in p_k \Leftrightarrow \sum_{i=0}^{k-1} p_i \leq d_a \leq \sum_{i=0}^{k} p_i$$

De esta manera, se puede formar la matriz usando el método descrito:

$$\{Q \equiv 1 \leq n \leq 1000\}$$
proc colOpMatrix(n: int, p: int, D: array, p: array, matrix: i/o array)
$$\{R \equiv (\forall i, j \in \{1, ..., n\} \ . \ (i = j \rightarrow matrix[i][j] = 1)$$
$$\vee(\exists x, y \in \{1, ..., n\} \wedge \exists k \ in\{1, ..., n-1\} \ .$$
$$(D[x] = i \wedge D[y] = j \wedge \sum_{a=0}^{k-1} p[a] \leq x, y \leq \sum_{a=0}^{k} p[a]) \rightarrow matrix[i][j] = 1)$$
$$\vee(\exists x, y \in \{1, ..., n\} \wedge \exists k \ in\{1, ..., n-1\} \ .$$
$$(D[x] = i \wedge D[y] = j \wedge !(\sum_{a=0}^{k-1} p[a] \leq x, y \leq \sum_{a=0}^{k} p[a]) \rightarrow matrix[i][j] = 0))\}$$

Ya que usaremos una función auxiliar para obtener la posición de un elemento en un arreglo, esta es de la manera:

$$\{Q \equiv 1 \leq n \leq 1000\}$$
func searchPos(num: int,n: int, arr: array)
$$\{R \equiv \exists x \in \{1, ..., n\} \ . \ arr[x] = num\}$$

3.2 Implementación

Se define la función para obtener la posición de un elemento de un arreglo.

```
int searchPos(int num, int n, int arr[n])
{
    int i = 0;
    while (i < n)
    {
        if (arr[i] == num)
        {
            return i;
        }
        i = i+1;
    }
}
```

Usando esta función auxiliar, se implementa la función para crear la matriz de opinión colectiva:

```
int *colOpMatrix(int n, int p, int D[n], int partitions[p], int **matrix)
{   /*Definicion de variables*/
    int pos1 = 0, pos2 = partitions[0], i=1, j=1, k=0, posi, posj;
    while (i <= n)
    {
        while (j <= n)
        {
            if (i == j)
            {
                matrix[i - 1][j - 1] = 1; /*si i=j, M_ij = 1*/
            }
            else
            {
                posi = searchPos(i, n, D); /*posiciones de las particiones*/
                posj = searchPos(j, n, D);
                while (k < p)
                {/*caso: la posicion de i esta por sobre el techo de la particion*/
                    if (pos2 < posi)
                    {
                        pos1 = pos2;
                        pos2 = pos2 + partitions[k + 2];
                    }
                    else
                    {/*caso: la posicion de i esta dentro de la particion*/
                        if (pos1 < posi && posi <= pos2)
```

```
26                          {/*caso: la posicion de j esta dentro de la particion*/
27                              if (pos1 < posj && posj <= pos2)
28                              {/*M_ij = 1*/
29                                  matrix[i - 1][j - 1] = 1;
30                              }/*caso: posiciones no estan dentro de la misma particion*/
31                              else
32                              {/*M_ij = 0*/
33                                  matrix[i - 1][j - 1] = 0;
34                              }
35                          }
36                      }
37                      k = k + 1;
38                  }
39                  k = 0;
40              }
41              j = j + 1;
42          }
43          j = 1;
44          i = i + 1;
45      }
46  }
```

3.3 correctitud

A continuación se desarrollará la correctitud para las funciones descritas anteriormente.

Correctitud para la funcion searchPos

1 $\{Q \equiv 1 \leq n \leq 1000\}$
2
3 `Asignacion: i = 0`
4 `(i < n), i incrementa po la operacion de la linea 10`
5
6 $\exists i \in \{1, ..., n\}$. $arr[i] = num$
7
8 $\{R \equiv \exists i \in \{1, ..., n\}$. $arr[i] = num\}$
9
10 `Asignacion: i = i+1`

Correctitud para el procedimiento colOpMatrix

1 $\{Q \equiv 1 \leq n \leq 1000\}$
2
3 `Asignacion: pos1=0, pos2=partitions[0], i=1, j=1, k=0`
4 $i \leq n$, `i incrementa por la operacion de la linea 44`
5
6 $j \leq n$, `j incrementa por la operacion de la linea 41`
7
8 $i = j \rightarrow matrix[i-1][j-1] = 1$
9
10 `asignacion:` $matrix[i-1][j-1] = 1$
11
12
13
14 `Asignacion: posi =  x .` $x \in \{1, ..., n\} \wedge D[x] = i$
15 `Asignacion: posj = x.` $x \in \{1, ..., n\} \wedge D[x] = j$
16 $(k < p)$, `k incrementa por la operacion de la linea 37`
17
18 $(pos2 < posi) \rightarrow pos1 = pos2 \wedge pos2 = pos2 + partitions[k+2]$
19
20 `Asignacion:` $pos1 = pos2$
21 `Asignacion` $pos2 = pos2 + partitions[k+2]$
22
23
24
25 $(pos1 < posi) \wedge ...$
26

27 $((pos1 < posi) \wedge (posj \leq pos2)) \rightarrow matrix[i-1][j-1] = 1$

28

29 `Asignacion:` $matrix[i-1][j-1] = 1$

30

31 $!((pos1 < posi) \wedge (posj \leq pos2)) \rightarrow matrix[i-1][j-1] = 0$

32

33 `Asignacion:` $matrix[i-1][j-1] = 0$

34

35

36

37 `Asignacion: k = k+1`

38

39 `Asignacion: k = 0`

40

41 `Asignacion: j = j+1`

42

43 `Asignacion: j = 1`

44 `Asignacion: i = i + 1`

Al final de la ejecución del procedimiento, dado por los ciclos de las líneas 4 y 6, en conjunto se obtiene:

$$\{R \equiv (\forall i, j \in \{1, ..., n\} \ . \ (i = j \rightarrow matrix[i][j] = 1)$$
$$\vee (\exists x, y \in \{1, ..., n\} \wedge \exists k \ in \{1, ..., n-1\} \ .$$
$$(D[x] = i \wedge D[y] = j \wedge \sum_{a=0}^{k-1} p[a] \leq x, y \leq \sum_{a=0}^{k} p[a]) \rightarrow matrix[i][j] = 1)$$
$$\vee (\exists x, y \in \{1, ..., n\} \wedge \exists k \ in \{1, ..., n-1\} \ .$$
$$(D[x] = i \wedge D[y] = j \wedge !(\sum_{a=0}^{k-1} p[a] \leq x, y \leq \sum_{a=0}^{k} p[a]) \rightarrow matrix[i][j] = 0))\}$$

4 Conclusiones

Luego de definir cabeceras, pre y postcondiciones y correctitud de los problemas planteados, se concluye que la correctitud es una forma eficaz para comprobar el funcionamiento de programas. comprobar que un programa funcione lógicamente es una herramienta útil para el desarrollo, mantenimiento y refacción de programas.

Finalmente, se concluye que estas herramientas trabajadas en conjunto son mas eficaces que cada una de estas por separado, dado que para definir la correctitud de un programa se necesitan las pre y postcondiciones. Y al tener las pre y postcondiciones definidas, no hay forma de comprobar que el funcionamiento interno del programa esté en orden con las entradas y salidas.