Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2020

Práctico 2 - Parte 1

1. Escribir un algoritmo que dada una matriz a: **array**$[1..n,1..m]$ **of int** calcule el elemento mínimo. Escribir otro algoritmo que devuelva un arreglo **array**$[1..n]$ con el mínimo de cada fila de la matriz a.

1)a)

```
fun minimo_matriz (a : array[1..n,1..m] of int) ret min : int
      min := a[1,1]
      for i := 1 to n do
            for j := 1 to m do
                  if (a[i,j] < min) →
                        min := a[i,j]
                  fi
            od
      od
end fun
```

Testeo
A = 1 2 3 n = 2, m = 3
4 5 6
min = 1
1.1 primera iteración i = 1
1.1.1 i = 1, j = 1 → a[1,1] < min
1.1.2 i = 1, j = 2 → a[1,2] < min
1.1.3 i = 1, j = 3 → a[1,3] < min
1.2 segunda iteración i = 2
1.1.1 i = 2, j = 1 → a[2,1] < min
1.1.2 i = 2, j = 2 → a[2,2] < min
1.1.3 i = 2, j = 3 → a[2,3] < min
min = 1

```
Testeo
A = 9    4   2    n = 2, m = 3
    15  -2  -12
min = 9
```

1.1 primera iteración i = 1
1.1.1 i = 1, j = 1 → a[1,1] < min nope
1.1.2 i = 1, j = 2 → a[1,2] < min → min = 4
1.1.3 i = 1, j = 3 → a[1,3] < min → min = 2
1.2 segunda iteración i = 2
1.1.1 i = 2, j = 1 → a[2,1] < min
1.1.2 i = 2, j = 2 → a[2,2] < min → min = -2
1.1.3 i = 2, j = 3 → a[2,3] < min → min = -12
min = -12

1)b)

```
fun (a : array[1..n,1..m] of int) ret b array[1..n] of int
      for j := 1 to n do
            b[j] := a[j,1]
            for i := 1 to m do
                  if (a[j,i] < b[j]) →
                        b[j] = a[j,i]
                  fi
            od
      od
end fun
```

Testeo

A =	1	2	3	n = 3, m = 3
	-5	10	-10	
	40	1	30	

1.1 primera iteración j = 1
b[1] = a[1,1] = 1
1.1.1 – i = 1 – a[1,1] < b[1]
1.1.2 – i = 2 – a[1,2] < b[1]
1.1.3 – i = 3 – a[1,3] < b[1]
b[1] = 1

1.2 segunda iteración j = 2
b[2] = a[2,1] = -5
1.1.1 – i = 1 – a[2,1] < b[2]
1.1.2 – i = 2 – a[2,2] < b[2]
1.1.3 – i = 3 – a[2,3] < b[2] → b[2] = -10
b[2] = -10

1.3 tercera iteración j = 3
b[3] = a[3,1] = 40
1.1.1 – i = 1 – a[3,1] < b[3]
1.1.2 – i = 2 – a[3,2] < b[3] → b[3] = 1
1.1.3 – i = 3 – a[3,3] < b[3]
b[3] = 1

j = 4 ya no entra en el bucle
b = [1,-10,1]

2. Dados los tipos enumerados

type mes = **enumerate**
enero
febrero
...
diciembre
end enumerate

type clima = **enumerate**
Temp
TempMax
TempMin
Pres
Hum
Prec
end enumerate

El arreglo med:**array**[1980..2016,enero..diciembre,1..28,Temp..Prec] **of nat** es un arreglo multidimensional que contiene todas las mediciones estadísticas del clima para la ciudad de Córdoba desde el 1/1/1980 hasta el 28/12/2016. Por ejemplo, med[2014,febrero,3,Pres] indica la presión atmosférica que se registró el día 3 de febrero de 2014. Todas las mediciones están expresadas con números enteros. Por simplicidad asumiremos que todos los meses tienen 28 días.

(a) Dar un algoritmo que obtenga la menor temperatura mínima (TempMin) histórica registrada en la ciudad de Córdoba según los datos del arreglo.

(b) Dar un algoritmo que devuelva un arreglo que registre para cada año entre 1980 y 2016 la mayor temperatura máxima (TempMax) registrada durante ese año.

(c) Dar un algoritmo que devuelva un arreglo que registre para cada año entre 1980 y 2016 el mes de ese año en que se registró la mayor cantidad mensual de precipitaciones (Prec).

(d) Dar un algoritmo que utilice el arreglo devuelto en el inciso anterior (además de med) para obtener el año en que ese máximo mensual de precipitaciones fue mínimo (comparado con los de otros años).

(e) Dar un algoritmo que obtenga el mismo resultado sin utilizar el del inciso (c).

2)a)

```
proc min_temp (in a:array[1..n,enero..diciembre,fst_day..lst_day, temp..prec) ret
result : nat
	var result : nat
	result = "infinito"
	for( i := 1 to n) do
		for( j := enero to diciembre) do
			for( m := fst_day to lst_day) do
				if a[i,j,m,TempMin] < result then
					result := a[i,j,m,TempMin]
				fi
			od
		od
	od
end proc
```

2)b)

```
proc high_temp(in a:array[1..n,enero..diciembre,fst_day..lst_day, temp..prec) ret
highest_temps : array[1..n] of nat
	var result : nat
	for( i := 1 to n) do
		result := "-infinito"
		for( j := enero to diciembre) do
			for( m := fst_day to lst_day) do
				if a[i,j,m,TempMax] > result then
					result := a[i,j,m,TempMax]
				fi
			od
		od
		highest_temp[i] := result
	od
```

2)c)

luego lo copio del lab03

2)d)

```
proc min_rainfall (in a:array[1..n,enero..diciembre,fst_day…lst_day, Temp..Prec] of
nat, b:array[enero..diciembre] of mes) ret minyear : nat
	var c : array[1..n] : nat
	var max_rainfall : nat
	var min_month : nat
	for( i := 1 to n ) do
		max_rainfall := 0
		for( day := fst_day to lst_day ) do
			max_rainfall := max_rainfall + a[i,b[i],day,Prec]
		od
		c[i] := max_rainfall
	od
	min_month := “infinito”
	for ( i := 1 to n) do
		if(c[i] < min_month) then
			min_month := c[i]
			minyear := 1980 + ( i - 1)
		fi
	od
```

2)e)

```
proc min_rainfall (in a:array[1..n,enero..diciembre,fst_day…lst_day, Temp..Prec] of
nat, b:array[enero..diciembre] of mes) ret minyear : nat
       var c : array[1..n] : nat
       var max_rainfall : nat
       var min_month : nat
       for( i := 1 to n ) do
              max_rainfall := 0
              for( day := fst_day to lst_day ) do
                     max_rainfall := max_rainfall + a[i,b[i],day,Prec]
              od
              c[i] := max_rainfall
       od
       min_month := "infinito"
       for ( i := 1 to n) do
              if(c[i] < min_month) then
                     min_month := c[i]
                     minyear := 1980 + ( i - 1)
              fi
       od
```

d)

```
proc min_max_rainfall ( in a:array[fst_year..lst_year,enero..febrero,
fst_day..lst_day,Temp..Prec]of nat) ret year_MinMaxMonthly_rainfall : nat
var rainfallsum : nat
var min_max_rainfall : nat
min_max_rainfall := "infinito"
for ( year := fst_year to lst_year) do
       high_monthly_rainfall := "-infinito"
       for ( month := enero to diciembre) do
              rainfallsum := 0
              for ( day := fst_day to lst_day) do
                     //precipitación mensual
                     rainfallsum := rainfallsum + a[year,month,day,Prec]
              od
              if( rainfallsum > high_monthly_rainfall) then
                     high_monthly_rainfall := rainfallsum
                     //guardo la máxima precipitación
              fi
       od
       if ( high_monthly_rainfall < min_max_rainfall )
              min_max_rainfall := high_monthly_rainfall
              //guardo la menor máxima prec.
              year_MinMaxMonthly_rainfall := year
              // guardo el año
       fi
od
end proc
```

3. Dado el tipo

type person = **tuple**
name: **string**
age: **nat**
weight: **nat**
end tuple

(a) escribí un algoritmo que calcule la edad y peso promedio de un arreglo a : **array**$[1..n]$ **of** *person*.
(b) escribí un algoritmo que ordene alfabéticamente dicho arreglo.

3)a)

```
proc age_weight_prom (in a : array[1..n] of person, out weight_prom
: float,out  age_prom : float)

weight_prom := 0
age_prom := 0

for i := 1 to n do
      weight_prom := weight_prom + a[i].weight
      age_prom := age_prom + a[i].age
od

weight_prom := weight_prom / n
age_prom := age_prom / n

end proc
```

3)b)

asumimos la existencia de una función que retorna un valor de verdad si una cadena esta antes que otra

```
less_string :: string -> string -> bool

proc sort_person (in/out a: array[1..n] of person)
      for i:= 2 to n do
            insert_person(a,i)
      od
end proc

proc insert_person (in/out a: array[1..n] of person, in i : nat)
      j:= i
      do j > 1 ∧ less_string(a[j].name, a[j−1].name)→ swap(a,j−1,j)
                                                      j:= j-1
      od
end proc
```

4. Dados dos punteros p, q : **pointer to int**

   (a) escribí un algoritmo que intercambie los valores referidos sin modificar los valores de p y q.

   (b) escribí otro algoritmo que intercambie los valores de los punteros.

   Sea un tercer puntero r : **pointer to int** que inicialmente es igual a p, y asumiendo que inicialmente $*p = 5$ y $*q =$ -4 ¿cuáles serían los valores de $*p$, $*q$ y $*r$ luego de ejecutar el algoritmo en cada uno de los dos casos?

4)a)

```
proc swap_reference (in p,q : pointer to int)
      var aux : int
      aux := *p
      *p := *q
      *q := aux
end proc
```

4)b)

```
proc swap_pointer (in p,q : pointer to int)
      var aux : pointer to int
      aux := p
      p := q
      q := aux
end proc
```

Algo que no anda:

```
  var temp : pointer to int
   temp := p     {- ahora temp vale DIRECCION(451) -}
   p := q        {- ahora p vale DIRECCION(760) -}
   q := temp     {- ahora q vale DIRECCION(451) -}
   {- hasta acá, nunca se cambiaron los valores de DIRECCION(451) ni 760.
-}
   free(temp)    {- AHORA DIRECCION(451) NO EXISTE MAS, ENTONCES q APUNTA A
LA NADA MISMA. QUEDA "COLGADO". -}
```

No hay que reservar ni liberar mas memoria, solo cambiar las direcciones

5. Dados dos arreglos a, b : **array**$[1..n]$ **of nat** se dice que a es "lexicográficamente menor" que b sii existe $k \in \{1 \dots n\}$ tal que $a[k] < b[k]$, y para todo $i \in \{1 \dots k-1\}$ se cumple $a[i] = b[i]$. En otras palabras, si en la primera posición en que a y b difieren, el valor de a es menor que el de b. También se dice que a es "lexicográficamente menor o igual" a b sii a es lexicográficamente menor que b o a es igual a b.

(a) Escribir un algoritmo lex_less que recibe ambos arreglos y determina si a es lexicográficamente menor que b.

(b) Escribir un algoritmo lex_less_or_equal que recibe ambos arreglos y determina si a es lexicográficamente menor o igual a b.

(c) Dado el tipo enumerado

type ord = **enumerate**
igual
menor
mayor
end enumerate

Escribir un algoritmo lex_compare que recibe ambos arreglos y devuelve valores en el tipo ord. ¿Cuál es el interés de escribir este algoritmo?

5)a)

```
fun lex_less ( a,b : array[1..n] of nat ) ret result : bool
      result := false
      for i := 1 to n do
            if(a[i] != b[i] && a[i] < a[b]) →
                  result := true
            fi
      od
end fun
```

5)b)

```
fun lex_less_or_equal ( a,b : array[1..n] of nat ) ret result : bool
      result := false
            if( lex_less(a,b) || equal(a,b)) →
                  result := true
            fi
end fun

fun equal ( a,b : array[1..n] of nat ) ret result : bool
      result := true
      for i := 1 to n do
            if( a[i] != b[i]) →
                  result := false
            fi
      od
end fun
```

5)c)

```
fun lex_compare ( a,b : array[1..n] of nat ) ret result : ord
      if (lex_less(a,b)) →
            result := menor
      else if (equal(a,b)) →
            result := igual
      else →
            result := mayor
      fi
end fun
```

6. Escribir un algoritmo que dadas dos matrices a, b: **array**[1..n,1..m] **of nat** devuelva su suma.

6)

```
fun sum_m ( a,b : array[1..n,1..m] of nat ) ret s : array
[1..n,1..m] of nat

      for i := 1 to n do
            for j := 1 to m do
                  s[i,j] := a[i,j] + b[i,j]
            od
      od
end fun
```

7. Escribir un algoritmo que dadas dos matrices a: **array**[1..n,1..m] **of nat** y b: **array**[1..m,1..p] **of nat** devuelva su producto.

7)

```
fun prod_m (a : array[1..n,1..m], b : array[1..m,1..p]) ret prod :
array[1..n,1..p]

      for i := 1 to n do
            for j := to p do
                  for k := 1 to m do
                        prod[i,j] := prod[i,j] + ( a[i,k] * b[k,j] )
                  od
            od
      od
end fun
```