Práctico 13 - Búsqueda

Programación 1
InCo - Facultad de Ingeniería, Udelar

Para los ejercicios 1 a 5, considerar las siguientes declaraciones de tipos:

```
const
    MAX = . . .;  {entero mayor estricto que 1}
type
      ArregloEnteros = array [1..MAX] of integer;
      ListaEnteros = ^Celda;
      Celda = record
                elem : integer;
                sig : ListaEnteros;
              end;
```

1. Escriba la función `buscarElemento` que determina si el elemento `e` se encuentra en:
   (a) el arreglo `a`

   ```
   function buscarElemento(e:integer; a:arregloEnteros):boolean;
   ```

   ```
   function buscarElemento(e:integer; a:arregloEnteros):boolean;
   var i:integer;
   begin
       i := 1;
       while (i <= MAX) and (A[i] <> e) do
           i := i + 1;
       buscarElemento := i <= MAX
   end;
   ```

   (b) el arreglo ordenado de menor a mayor `a`

   ```
   function buscarElemento(e:integer; a:arregloEnteros):boolean;
   ```

   ```
   function buscarElemento(e:integer; a:arregloEnteros):boolean;
   var inf, sup, medio: 0..MAX+1;
   begin
       inf := 1;
       sup := n;
       medio := (inf + sup) DIV 2;
       while (inf <= sup) and (A[medio] <> e) do
       begin
           if e < A[medio] then
               sup  := medio - 1
           else
               inf := medio + 1;
           medio := (inf + sup) DIV 2
       end;
       buscarElemento := inf <= sup
   end;
   ```

   (c) la lista de enteros `l`

   ```
   function buscarElemento(e:integer; l:ListaEnteros):boolean;
   ```

¿Cambiaría su solución si la lista estuviese ordenada?

```
function buscarElemento(e:integer; l:ListaEnteros):boolean;
var p: ListaEnteros;
begin
    p := l;
    while (p <> nil) and (p^.elem <> e) do
        p := p^.sig;
    buscarElemento := (p <> nil);
end;
```

2. Escriba la función `unSoloMayor` que determina si existe un único elemento mayor que el valor `v` en:
   (a) el arreglo `a`

   `function unSoloMayor(v:integer; a:arregloEnteros):boolean;`

```
function unSoloMayor(v:integer; a:arregloEnteros):boolean;
var i, cant :integer;
begin
  i := 1;
  cant := 0;
  while (i <= MAX) and (cant < 2) do
  begin
    if (A[i] >  v) then
      cant := cant + 1;
    i := i + 1;
  end;

    unSoloMayor := cant = 1;
end;
```

   (b) el arreglo ordenado de menor a mayor `a`

   `function unSoloMayor(v:integer; a:arregloEnteros):boolean;`

```
function unSoloMayor(v:integer; a:arregloEnteros):boolean;
begin
    unSoloMayor := (A[MAX] > v) and (A[MAX-1] <= v);
end;
```

   (c) la lista de enteros `l`

   `function unSoloMayor(v:integer; l:ListaEnteros):boolean;`

```
function unSoloMayor(v:integer; l:ListaEnteros):boolean;
var
    cant :integer;
    p :ListaEnteros;
begin
    p := l;
    cant := 0;
    while (p <> nil) and (cant < 2) do
    begin
        if (p^.elem > v) then
            cant := cant + 1;
        p := p^.sig
    end;
    unSoloMayor := cant = 1
end;
```

3. Escriba la función `todosMayores` que determina si todos los elementos son mayores que el valor `v` en:
   (a) el arreglo `a`

   ```
   function todosMayores(v:integer; a:arregloEnteros):boolean;
   ```

   ```
   function todosMayores(v:integer; a:arregloEnteros):boolean;
   var i : integer;
   begin
     i := 1;
     while (i <= MAX) and (A[i] > v) do
       i := i + 1;
     todosMayores := i > MAX
   end;
   ```

   (b) el arreglo ordenado de menor a mayor `a`

   ```
   function todosMayores(v:integer; a:arregloEnteros):boolean;
   ```

   ```
   function todosMayores(v:integer; a:arregloEnteros):boolean;
   begin
       todosMayores = A[1] > v;
   end;
   ```

   (c) la lista de enteros `l`

   ```
   function todosMayores(v:integer; l:ListaEnteros):boolean;
   ```

   ```
   function todosMayores(v:integer; l:ListaEnteros):boolean;
   var p : ListaEnteros;
   begin
     p := l;
     while (p <> nil) and (p^.elem > v) do
       p := p^.sig;
     todosMayores := p = nil;
   end;
   ```

4. Escriba la función `cuantosMayores` que determina cuántos son mayores que el valor `v` en:
   (a) el arreglo `a`

   ```
   function cuantosMayores(v:integer; a:arregloEnteros):integer;
   ```

   ```
   function cuantosMayores(v:integer; a:arregloEnteros):integer;
   var i, cant :integer;
   begin
     cant := 0;
     for i:=1  to MAX do
       if (A[i] >  v) then
         cant := cant + 1;

     cuantosMayores := cant;
   end;
   ```

   (b) el arreglo ordenado de menor a mayor `a`

   ```
   function cuantosMayores(v:integer; a:arregloEnteros):integer;
   ```

```
function cuantosMayores(v:integer; a:arregloEnteros):boolean;
var i: integer;
begin
    if (A[1] > v) then
        cuantosMayores := MAX
    else if (A[MAX] <= v) then
        cuantosMayores := 0
    else
    begin
        i := MAX;
        while (i >= 1) and (A[i] > v) do
            i := i - 1;
        cuantosMayores := MAX - i;
    end;
end;
```

(c) la lista de enteros `l`

`function cuantosMayores(v:integer; l:ListaEnteros):integer;`

```
function cuantosMayores(v:integer; l:ListaEnteros):integer;
var p: ListaEnteros;
var cant: integer;
begin
    cant := 0;
    p := l;
    while (p <> nil) do
        if (p^.elem > v) then
            cant := cant + 1;
    cuantosMayores := cant;
end;
```

5. Escriba la función `ultimaOcurrencia` que determina la última posición en la cual ocurre el elemento `e`. Si `e` no se encuentra, la función devuelve 0. Implemente para las siguientes estructuras:

(a) el arreglo `a`

`function ultimaOcurrencia(e:integer; a:arregloEnteros):integer;`

```
function ultimaOcurrencia(e:integer; a:arregloEnteros):integer;
var i : integer;
begin
  i := MAX;
  while (i > 0) and (A[i] <>  e) do
    i := i-1;

  ultimaOcurrencia := i
end;
```

(b) el arreglo ordenado de menor a mayor `a`

`function ultimaOcurrencia(e:integer; a:arregloEnteros):integer;`

```
function ultimaOcurrencia(e:integer; a:arregloEnteros):integer;
var inf, sup, medio, i, ultimo: integer;
begin
    if (A[1] > e) or (A[MAX] < e) then
        ultimo := 0
    else
```

```
    begin
        inf := 1;
        sup := MAX;
        medio := (inf + sup) DIV 2;
        while (inf <= sup) and (A[medio] <> e) do
        begin
            if e < A[medio] then
                sup  := medio - 1
            else
                inf := medio + 1;
            medio := (inf + sup) DIV 2
        end;
        if inf <= sup then
        begin
            i := medio;
            repeat
                i := i + 1;
            until (i > MAX) or (A[i] <> e);
            ultimo := i-1
        end
        else
            ultimo := 0
    end;
    ultimaOcurrencia := ultimo;
end;
```

(c) la lista de enteros `l`

```
function ultimaOcurrencia(e:integer; l:ListaEnteros):integer;
```

```
function ultimaOcurrencia(e:integer; l:ListaEnteros):integer;
var
    ultima, pos : integer;
    p : ListaEnteros;
begin
    p := l;
    pos := 1;
    ultima := 0;
    while (p <> nil) do
    begin
        if (p^.elem = e) then
            ultima := pos;
        p := p^.sig;
        pos := pos + 1
    end;
    ultimaOcurrencia := ultima;
end;
```

6. Sean las siguientes declaraciones:

```
const
    MAXCOL = . . .; {entero mayor estricto que 0}
    MAXFIL = . . .; {entero mayor estricto que 0}
type
    MatrizEnteros = array [1..MAXFIL, 1..MAXCOL] of integer;
```

Escriba la función `buscarElementoMatriz` que determina si el elemento `e` ocurre en la matriz `m`. Puede invocar las funciones de los ejercicios 1 a 5. Para ello puede suponer que las constantes `MAXCOL` y `MAX` coinciden.

```
function buscarElementoMatriz(e:Integer; m:matrizEnteros):boolean;
```

```
function buscarElementoMatriz(e:Integer; m:matrizEnteros):boolean;
var i : integer;
begin
  i := 1;
  while (i <= MAXFIL) and not buscarElemento(e,m[i]) do
    i := i+1;

  buscarElementoMatriz := i <= MAXFIL
end;
```

7. Sean las siguientes declaraciones:

```
const
    MAX = . . .; {entero mayor estricto que 0}
type
      TComparacion = (mayor, menor, igual);
      TTexto = record
                     nombre : array [1 .. MAX] of char;
                     tope     : 0 .. MAX;
                 end;
      ArregloNombres = array [1..MAX] of TTexto;
```

   (a) Escriba la función `comparacionNombres` que dados dos nombres `n1` y `n2` devuelva la comparación entre ellos. La comparación es en base al orden lexicográfico.

   ```
   function comparacionNombres(n1, n2:TTexto):TComparacion;
   ```

   ```
   function comparacionNombres(n1, n2:TTexto):TComparacion;
   var topeMin, i : integer;
       cp :TComparacion;
   begin

     if n1.tope <= n2.tope then
       topeMin := n1.tope
     else
       topeMin := n2.tope;

     i := 1;
     {recorre los nombres mientras no encuentre una letra diferente
     y tengas letras para comparar}
     while (i <= topeMin) and (n1.nombre[i] = n2.nombre[i]) do
       i := i + 1;
     if  i <= topeMin then {encontro una letra diferente}
       if n1.nombre[i] > n2.nombre[i] then
         cp := mayor
       else
         cp := menor
     else {no encotro letra diferente, es mayor el de mayor largo}
       if n1.tope = n2.tope then
         cp := igual
       else
         if n1.tope > n2.tope then
           cp := mayor
         else
           cp := menor;
   ```

```
  comparacionNombres := cp;
end;
```

(b) Escriba la función `buscarNombre` que devuelve `TRUE` si el nombre `n` ocurre dentro del arreglo `a` y `FALSE` en caso contrario. El arreglo de nombres `a` está ordenado en forma lexicográfica y creciente. ¿Realizará búsqueda lineal o binaria? ¿Por qué?

`function buscarNombre(n:TTexto; a:arregloNombres):boolean;`

En este caso se debe realizar búsqueda binaria, dado que el arreglo se encuentra ordenado.

```
function buscarNombre(n:TTexto; a:arregloNombres):boolean;
var inf, sup, medio: integer;
    encontre : boolean;
begin
  inf := 1;
  sup := MAX;
  medio := (inf + sup) DIV 2;
  encontre := false;

  while (inf <= sup) and not encontre do
    case comparacionNombres(n, a[medio]) of
      mayor: begin
        inf := medio + 1;
        medio := (sup + inf) div 2
      end;
      menor: begin
        sup := medio - 1;
        medio := (sup + inf) div 2
      end;
      igual: encontre := true
  end;

  buscarNombre := inf <= sup;
end;
```

(c) Para la parte anterior, ¿consideró si había nombres repetidos en el arreglo? Si no lo consideró, ¿qué cambia en su código si los hay? ¿Qué cambiaría si hay nombres repetidos y se le pide encontrar la primera ocurrencia de un nombre?

Si sólo tenemos que decir si lo encontramos, no cambia nada dado que con encontrar un elemento ya alcanza. Si tenemos que devolver el indice de la primera ocurrencia, podríamos recorrer linealmente hacia atrás hasta encontrar el primero, pero sería más eficiente utilizar la búsqueda binaria con el siguiente criterio:

```
function comparacionNombres2 (
  medio: integer; n: TTexto; a: arregloNombres
): TComparacion;
begin
  case comparacionNombres(a[medio], n) of
    menor: comparacionNombres2 := menor;
    igual: if (medio = 1) or (comparacionNombres(a[medio-1], n) = menor) then
        comparacionNombres2 := igual
      else
        comparacionNombres2 := mayor;
    mayor: comparacionNombres2 := mayor;
  end
end;
```

8. Sean las siguientes declaraciones:

```
const
    MAXE  = . . .; {entero mayor estricto que 0}
    MAXC = . . .;  {entero mayor estricto que 0}
    MAX = . . .; {entero mayor estricto que 0}

type
      TTexto = record
                     caracteres : array [1 .. MAX] of char;
                     tope     : 0 .. MAX;
                  end;

     TEstudiante = record
                     nombre     : TTexto;
                     generacion : integer;
                   end;

     TEstudiantes = record
                              estudiante : array [1 .. MAXE] of TEstudiante;
                              tope : 0 .. MAXE;
                            end;

      TCarrera = record
                       nombre : TTexto;
                       estudiantes : TEstudiantes;
                     end;

     TBedelias = array [1 .. MAXC] of TCarrera;
```

que representan los estudiantes que están inscriptos a una carrera de grado en facultad. Un elemento de tipo `TBedelias` tiene ordenadas las carreras en forma lexicográfica y creciente por su nombre. Luego, por cada carrera, la lista de estudiantes se encuentra ordenada por el año de ingreso en forma creciente.

(a) Escriba la función `buscarEstudiante` que determina si el estudiante de nombre `nombreE` está inscripto a la carrera de nombre `nombreC`. Para comparar los nombres de las carreras y de los estudiantes, puede usar la función definida en el ejercicio 7 (a).

```
function buscarEstudiante(nombreE:TTexto; nombreC:TTexto; bed:TBedelias):boolean;
```

```
function buscarCarrera(n:TTexto; a:TBedelias):integer;
var inf, sup, medio: integer;
begin
  inf := 1;
  sup := MAXC;
  medio := (inf + sup) DIV 2;
  while (inf <= sup) and (comparacionNombres(a[medio].nombre,n) <> igual) do
  begin
    if comparacionNombres(n,a[medio].nombre) = menor then
      sup := medio - 1
    else
      inf := medio + 1;
    medio := (inf + sup) DIV 2
  end;

  if (inf <= sup) then
    buscarCarrera := medio
  else
    buscarCarrera := -1;
end;


function buscarEstudianteCarr(nombreE:TTexto; estudiantes: TEstudiantes):boolean;
```

```
var i : integer;
begin
  i := 1;
  while (i <= estudiantes.tope) and
        (comparacionNombres(estudiantes.estudiante[i].nombre,nombreE) <> igual) do
    i := i + 1;
  buscarEstudianteCarr :=  i <= estudiantes.tope
end;


function buscarEstudiante(nombreE:TTexto; nombreC:TTexto; bed:TBedelias):boolean;
var indC : integer;
begin
  indC := buscarCarrera(nombreC,bed);
  if indC = -1 then
    buscarEstudiante := false
  else
    buscarEstudiante :=  buscarEstudianteCarr(nombreE, bed[indC].estudiantes);
end;
```

(b) En la parte anterior, ¿qué tipos de búsquedas tuvo que realizar? Explique.

Para buscar la carrera se realiza búsqueda binaria, dado que el arreglo está ordenado por nombre. Para buscar los estudiantes se realiza búsqueda lineal, dado que, si bien están ordenados por año, los tenemos que buscar por nombre.

(c) Escriba el procedimiento `buscarEstudiantesGeneracion` que devuelva en `bGen` todos los estudiantes que ingresaron en el año `gen`. Si no hay estudiantes que ingresaron en tal año para una carrera, la lista de estudiantes para dicha carrera debe quedar vacía. ¿Qué tipo de búsqueda debe realizar?.

```
procedure buscarEstudiantesGeneracion(gen:integer; bed:TBedelias; var bGen:TBedelias);
```

Se debe realizar una búsqueda binaria por año, dado que los estudiantes se encuentran organizados por año.

```
function comparacionEstudiante(i, gen: integer; est: TEstudiantes):TComparacion;
begin
  if est.estudiante[i].generacion = gen then
    if est.estudiante[i-1].generacion < gen then
      comparacionEstudiante := igual
    else
      comparacionEstudiante := mayor
  else
    if est.estudiante[i].generacion < gen then
      comparacionEstudiante := menor
    else
      comparacionEstudiante := mayor
end;

function buscarGeneracion(gen: integer; est: TEstudiantes):integer;
var inf, sup, medio: integer;
  encontre: boolean;
begin
  if est.estudiante[1].generacion < gen then
  begin
    inf := 2;
    sup := est.tope;
    medio := (inf + sup) DIV 2;
    encontre := false;
    while (inf <= sup) and not encontre do
```

```
      case comparacionEstudiante(medio, gen, est) of
        mayor: begin
          sup := medio - 1;
          medio := (inf + sup) div 2
        end;
        menor: begin
          inf := medio + 1;
          medio := (inf + sup) div 2
        end;
        igual: encontre := true
      end;
    buscarGeneracion := medio
  end
  else
    buscarGeneracion := 1
end;

procedure buscarEstudiantesGeneracion(
    gen: integer; bed: TBedelias; var bGen: TBedelias
);
var i, indEG : integer;
begin
  for i := 1 to MAXC do
  begin
    { asumimos existe la generacion buscada y buscamos el primer
      estudiante de la generacion }
    indEG := buscarGeneracion(gen, bed[i].estudiantes);

    { recorre todos los estudiantes de la generacion }
    bGen[i].nombre := bed[i].nombre;
    bGen[i].estudiantes.tope := 0;
    while(indEG <= bed[i].estudiantes.tope) and
        (bed[i].estudiantes.estudiante[indEG].generacion = gen) do
    begin
      bGen[i].estudiantes.tope := bGen[i].estudiantes.tope + 1;
      bGen[i].estudiantes.estudiante[bGen[i].estudiantes.tope] :=
        bed[i].estudiantes.estudiante[indEG];
      indEG := indEG + 1
    end
  end
end;
```

9. Sean las siguientes declaraciones:

```
const
    MAX = . . .; {entero mayor estricto que 0}
type
      Rango = 1 .. MAX;
      ArregloEnteros = array [rango] of integer;
```

(a) Escriba la función `estaOrdenado` que determina si el arreglo `a` está ordenado en forma ascendente.

```
function estaOrdenado(a:arregloEnteros):boolean;
```

```
function estaOrdenado(a:arregloEnteros):boolean;
var i : integer;
begin
  i := 1;
```

```
  while (i < MAX) and (a[i] <= a[i+1]) do
    i := i + 1;

  estaOrdenado := i = MAX;
end;
```

10. (a) Escriba un programa donde el usuario ingresa números enteros positivos, terminando la secuencia con el número `-1`. Puede asumir que como máximo el usuario ingresará una cantidad `N` de números, siendo `N` una constante declarada al inicio del programa. Luego, el usuario ingresa otra secuencia de números enteros positivos que termina en `-1`. El programa debe mostrar, por cada número de la segunda secuencia, si éste pertenece a la primera secuencia.

Ejemplo:

```
1 9 100 20 88 100 1 -1
6
El número 6 no pertenece a la secuencia.
9
El número 9 pertenece a la secuencia
100 2
El número 100 pertenece a la secuencia
El número 2 no pertence a la secuencia
-1
```

Analice las distintas formas de implementar dicho programa teniendo en cuenta los algoritmos de búsquedas y ordenación vistos en el curso y justifique la elección de los algoritmos empleados, de forma de tener en cuenta el tiempo de ejecución del programa.

(b) Modifique su programa para que imprima la cantidad de comparaciones que fueron necesarias para determinar si un número pertenece a la secuencia. ¿Cómo relaciona la cantidad de comparaciones con los algoritmos de búsqueda vistos en el curso y la forma de implementar su programa?

```
program ej11;
const N = ...; {entero mayor estricto que 0}
type ArregloNumeros = record
  numeros : array [1..N] of integer;
  tope : 0..N
end;

procedure insertarOdenado(n: integer; var a: ArregloNumeros);
var i: integer;
begin
  i := a.tope;
  while (i > 0) and (n < a.numeros[i]) do
  begin
    a.numeros[i+1] := a.numeros[i];
    i := i-1
  end;
  a.numeros[i+1] := n;
  a.tope := a.tope+1
end;

procedure leerNumeros (var a: ArregloNumeros);
var n: integer;
begin
  read(n);
  a.tope := 0;
  while n <> -1 do
  begin
```

```
    insertarOdenado(n,a);
    read(n)
  end;
  readln;
end;

procedure pertenece (m: integer; a: ArregloNumeros;
  var pert: boolean; var comp: integer);
var i_inf, i_sup, i_med: 0..N+1;
begin
  i_inf := 1;
  i_sup := a.tope;
  i_med := (i_sup+i_inf) div 2;
  comp := 0;
  while (i_inf <= i_sup) and (a.numeros[i_med] <> m) do
  begin
    comp := comp+3;
    if a.numeros[i_med] > m then
      i_sup := i_med-1
    else
      i_inf := i_med+1;
    i_med := (i_sup+i_inf) div 2
  end;
  pert := i_inf <= i_sup
end;

var m, comp: integer; pert: boolean; a: ArregloNumeros;

begin
  leerNumeros(a);
  readln(m);
  while m <> -1 do
  begin
    pertenece(m, a, pert, comp);
    write('Con ', comp:1, ' comparaciones se determino que ');
    if pert then
      writeln('el numero ', m:1, ' pertenece a la secuencia.')
    else
      writeln('el numero ', m:1, ' no pertenece a la secuencia.');
    readln(m)
  end
end.
```