PAC2 – DANIEL GALEANO SANCHO

Format i data de lliurament

Cal lliurar la solució en un fitxer de tipus **pdf** a l'apartat de lliuraments d'AC de l'aula de teoria.

La data límit per lliurar la solució és el diumenge, 19 d'Abril de 2020 (a les 23:59 hores).

Presentació

El propòsit d'aquesta segona PAC és comprovar que has adquirit els conceptes explicats en els capítols 'Recursivitat' i 'TADs'.

Competències

Transversals

- Capacitat de comunicació en llengua estrangera.
- Coneixements de programació amb llenguatge algorísmic.

Específiques

- Capacitat de dissenyar i construir algorismes informàtics mitjançant tècniques de desenvolupament, integració i reutilització.

Objectius

Els objectius d'aquesta PAC són:

- Adquirir els conceptes teòrics explicats sobre les tècniques d'anàlisi d'algorismes i recursivitat.
- Dissenyar funcions recursives, identificant els casos base i recursius, sent capaços de simular la seqüència de crides donada una entrada.
- Implementar un algorisme iteratiu a partir d'un algorisme recursiu.
- Manipular operacions dels TADs bàsics implementats amb punters.

Descripció de la PAC a realitzar

Raona i justifica totes les respostes.

Les respostes incorrectes **no** disminueixen la nota.

Tots els dissenys i implementacions han de realitzar-se en llenguatge algorísmic. Els noms dels tipus, dels atributs i de les operacions s'han d'escriure en anglès. Els comentaris i missatges d'error no és obligatori fer-los en anglès, tot i que es valorarà positivament que es faci, ja que és l'estàndard.

Recursos

Per realitzar aquesta prova disposes dels següents recursos:

Bàsics

- Materials en format **web de l'assignatura**.
- **Fòrum de l'aula de teoria**. Disposes d'un espai associat en l'assignatura on pots plantejar els teus dubtes sobre l'enunciat.

Complementaris

- **Cercador web**. La forma més ràpida d'obtenir informació ampliada i extra sobre qualsevol aspecte de l'assignatura és mitjançant un cercador web.
- Solució de la PAC d'un semestre anterior.

Criteris de valoració

Per a la valoració dels exercicis es tindrà en compte:

- L'adequació de la resposta a la pregunta formulada.
- Utilització correcta del llenguatge algorísmic.
- Claredat de la resposta.
- Completesa i nivell de detall de la resposta aportada.

Avís

- Aprofitem per recordar que està totalment prohibit copiar en les PACs de l'assignatura. S'entén que hi pot haver un treball o comunicació entre els alumnes durant la realització de l'activitat, però el lliurament d'aquesta ha de ser individual i diferenciat de la resta.

- Així doncs, els lliuraments que continguin alguna part idèntica respecte a lliuraments d'altres estudiants seran considerats còpies i tots els implicats (sense que sigui rellevant el vincle existent entre ells) suspendran l'activitat lliurada.

[20%] Exercici 1: Conceptes bàsics de recursivitat

Tasca: Respon les preguntes següents justificant les respostes:

i) Què és la recursivitat simple i la múltiple?

La recursivitat simple es defineix com quan els algorismes fan una única crida a sí mateixos.
Per altra banda, la recursivitat múltiple és quan aquesta crida es realitza més d'una vegada.

ii) Quina és la missió de la funció equal en un TAD?

Serveix per comparar si dos objectes són iguals sigui quina sigui la definició del tipus abstracte.

iii) Quins avantatges i desavantatges tenen els algorismes recursius respecte als iteratius?

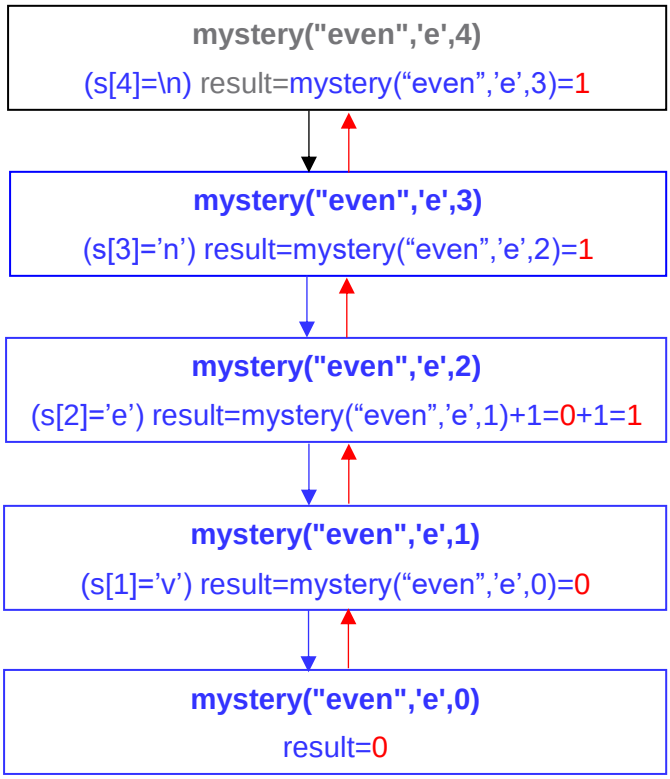
Alguns algorismes escrit de manera recursiva són més clars i intuïtius. Per tant, tot i que qualsevol algorisme recursiu es pot escriure de manera iterativa, s'aconsegueix aquest avantatge de claredat i intuïció extra.

El principal inconvenient de la recursivitat és que les successives crides recursives consumeixen més recursos que el mateix algorisme en forma iterativa.

iv) Donada la funció recursiva **mystery**, calcula quin valor retorna la crida **mystery("even",'e',4)** i completa el **model de les còpies** per veure com has arribat al resultat:

```
function mystery (s: string, c : char, p: integer ) : integer
var
    result : integer;
end var
    if p = 0 then
            result := 0;
    else
            result := mystery (s, c, p-1);
            if s[p] = c then
                    result := result + 1;
            end if
    end if
    return result;
end function
```

El resultat és **mystery("even",'e',3)** = 1.



[20%] Exercici 2: Disseny d'algorismes recursius

Tasca: Donada la descripció dels problemes següents, dissenya els algorismes recursius que els resolen.

Consell: Abans de començar a escriure cada algorisme, has d'identificar els casos base i recursius.

i) Dissenya la funció recursiva **square_number** que retorna cert quan **n** és un nombre quadrat perfecte, és a dir, retorna cert si n és el quadrat d'un nombre enter entre **start** i **n**.
Exemple: square_number *(49, 1)* retorna cert.

function square_number (n : **integer**, start: **integer**) : **boolean**
Pre: { n=N i N > 0 i start = START i START≤N}

Cas base: Quan *start* sigui igual *n* voldrà dir que ja hem comprovat tots els números que hi ha entre *start* i *n*.
Cas recursiu: Es retornarà el resultat de comprovar si la mateixa funció amb *start+1* és quadrat perfecte.

```
function square_number ( n : integer, start : integer ) : boolean
    if start = n then
            return false;
    else
            return start*start = n or square_number(n,start+1);
    end if
end function
```

ii) Dissenya l'acció recursiva **filter_stack** que desempila tots els enters de la pila p que estan en posicions parells començant des del fons de la pila. L'element que es troba al fons de la pila ocupa la posició 1.
Exemple: Si p conté la pila {3, -1, 2, 7} on 7 és el cim, la crida filter_stack(p, aux) retorna la pila {3, 2} al paràmetre p.

action filter_stack (**inout** p: **stack**(**integer**), **out** pos: **integer**)

Per dissenyar aquesta funció cal fer crides recursives fins desempilar tots els elements de la pila per tal de trobar el fons. Un cop la pila està buida, es comença el camí de retorn de la recursió on el paràmetre **pos** retornarà el nombre d'elements en la pila després de cada crida recursiva.

Cas base: Quan la pila està buida vol dir que ja tenim tots els elements que hem de considerar.

Cas recursiu: Es guarda el valor del cim de la pila en una variable local, es desempila i es crida a la funció amb la nova pila amb un element menys. Després s'empilen la variable local emmagatzemada si compleix la condició.

```
action filter_stack ( inout p: stack(integer), out pos: integer)
    var
        aux : integer;
    end var

    if emptyStack(p) then
        pos := 0;
    else
        aux := top(p);
        pop(p);
        filter_stack(p,pos);
        pos := pos + 1;
        if pos mod 2 ≠ 0 then
            push(p,aux);
        end if
    end if
end action
```

[20%] Exercici 3: Convertir algorismes recursius en iteratius

Tasca: Donada una acció recursiva transforma-la en iterativa.

i) Completa el disseny de la funció recursiva que calcula si un nombre **n** és un nombre primer.
La crida inicial a *is_prime* es fa de la manera següent:

is_prime (n, 2);

Exemple: el resultat de la crida *is_prime (7,2)* és cert, ja que el nombre 7 només és divisible per ell i per la unitat.

```
function is_prime (n: integer, d: integer): boolean
Pre: { n=N i N>1 i d=D i D≤N}
var
    result : boolean;
end var

    if d = n then
            result := true
    else
            result := is_prime(n,d+1);
            if n mod d = 0 then
                    result := result and false;
            end if
    end if
    return result;

end function
```

ii) Transforma la funció recursiva *is_prime* en una funció iterativa.

```
function is_prime(n: integer, d: integer): boolean
    var
            result : boolean;
    end var
    result := true;
    while (d ≠ n) do
            if n mod d = 0 then
                    result := result and false;
            end if
    end while
end function
```

[20%] Exercici 4: Modificació de TAD bàsics

Tasca: Donada la implementació del TAD pila amb punters (explicada en els apunts):

```
type
    node = record
        e : elem;
        next : pointer to node;
    end record

    stack = record
        first : pointer to node;
    end record
end type
```

Estén el tipus afegint les operacions següents:

i) **count**: funció que retorna el nombre d'elements emmagatzemats a la pila.

function count (p: **stack**(elem)) : **integer**

Per dissenyar aquesta funció no pots utilitzar les operacions del tipus **pila** (push, pop, top...). Així doncs, has de treballar directament amb la implementació del tipus que us hem facilitat en l'enunciat.

```
function count (p : stack(elem)) : integer
    var
        aux : stack;
        tmp : node;
        i : integer;
    end var

    duplicate(p, aux);
    tmp := aux.first;

    while (tmp ≠ NULL) do
        tmp := tmp^.next;
        i++;
    end while
end function
```

ii) **to_queue**: funció que donada una pila retorna una cua amb el contingut de la pila, mantenint l'ordre d'aquesta de manera que el primer element de la cua emmagatzemarà el cim de la pila.

function to_queue (p: **stack**(elem)) : **queue**(elem)

Per dissenyar aquesta funció has d'utilitzar les operacions del tipus cua i pila, és a dir, aquest cop desconeixes com s'han implementat internament aquests dos tipus.

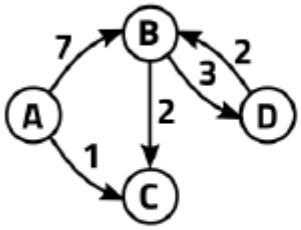
```
function to_queue (p : stack(elem)) : queue(elem)
    var
        q : queue(elem);
    end var

    q := createQueue();

    while not emptyStack(p) do
        enqueue(q,top(p));
        pop(p);
    end while

    return q;
end function
```

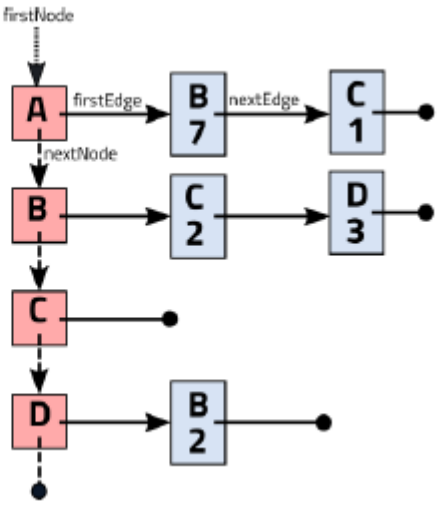
[20%] Exercici 5: Disseny d'un tipus amb punters

Tasca: Fins ara hem treballat amb els TADs pila, cua i llista, però a vegades aquests no ens permeten reflectir la realitat i necessitem crear tipus més complexos (com per exemple, una llista ordenada).

Definim el TAD **tGraph** que conté un graf dirigit i etiquetat amb strings. Un graf està format per un conjunt de nodes connectats per arestes dirigides. Cada node s'identifica amb una etiqueta (que en el nostre cas és un string) i cada aresta té un node origen, un node destí i un pes (que en el nostre cas és un enter).

Per facilitar la seva comprensió, us proporcionem un exemple de com podria ser la implementació d'un graf concret amb el tipus **tGraph**:





Així doncs, en el dibuix es pot veure que del node A surten dues arestes: una que va al node B amb pes 7 i una que va al node C amb pes 1.

i) A partir d'aquesta explicació, completa la definició del TAD **tGraph** utilitzant punters

```
type
    tEdge = record
            etiq: string;
            weight: integer;
            nextEdge : pointer to tEdge;
    end record
    tNode = record
            etiq: string;
            nextNode: pointer to tNode;
            firstEdge : pointer to tEdge;
    end record

    tGraph = record
            firstNode : pointer to tNode;
    end record
end type
```

ii) Aquest nou TAD oferirà diverses operacions, entre elles et demanem que implementis l'operació que retorna la suma dels pesos de totes les arestes que surten del node amb etiqueta **e**. Si el node no existeix o no té cap aresta sortint, llavors retorna 0.

function weights_node (g: tGraph, e:**string**) : **integer**

```
function weights_node (g : tGraph, e: string) : integer
    var
        wantedNode, auxNode : pointer to tNode;
        auxEdge : pointer to tEdge;
        sum : integer;
    end var

    sum := 0;

    wantedNode := NULL;
    auxNode := g.firstNode;

    while wantedNode = NULL and auxNode ≠ NULL do
        if auxNode^.etiq = e then
            wantedNode := auxNode;
        end if
        auxNode := auxNode^.nextNode;
    end while

    auxEdge := wantedNode^.firstEdge;

    if wantedNode ≠ NULL or auxEdge ≠ NULL then
        while auxEdge ≠ NULL do
            sum := sum + auxEdge^.weight;
            auxEdge := auxEdge^.nextEdge;
        end while
    end if

    return sum;
end function
```