PAC3: Tercera Prova d'Avaluació Continuada

Format i data de lliurament

Cal lliurar la solució en un fitxer de tipus **pdf**.

La data límit per lliurar la solució és el **dilluns, 3 de Desembre de 2018** (a les 23:59 hores).

Presentació

El propòsit d'aquesta tercera PAC és comprovar que has adquirit els conceptes explicats en els capítols '*Mètodes de Cerca'* i *'Complexitat'.*

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 els mètodes de cerca.
- Interpretar els algorismes de cerca, sent capaços de simular el seu funcionament donada una entrada.
- Implementar qualsevol mètode de cerca en un algorisme i avaluar-ne el seu rendiment.
- Adquirir els conceptes teòrics explicats sobre les tècniques d'anàlisi d'algorismes.
- Analitzar la complexitat d'una funció, calculant la seva funció de temps d'execució, i expressar aquesta complexitat amb notació asimptòtica.

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. No és obligatori escriure en anglès els comentaris i els missatges d'error, 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.**
- **El llibre *"Manual d'Algorísmica".*** En els capítols *Mètodes de Cerca* i *Tècniques d'Anàlisi d'Algorismes* pots trobar el material necessari per realitzar la prova.
- **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.
- La 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 treballs d'altres estudiants seran considerats còpies i tots els implicats (sense que sigui rellevant el vincle existent entre ells) suspendran la prova lliurada.

Exercici 1: Conceptes bàsics sobre cerca (10%)

Tasca: Digues si són certes o no les sentències següents. En cas que no ho siguin, especifica la raó:

i) En el cas pitjor, l'algorisme de cerca binària realitzarà ($\log_2 N+1$) accessos al vector, sent N la mida del vector.

Cert.

ii) Per realitzar una cerca binària en un vector no és necessari que estigui ordenat.

Fals. L'algorisme de cerca binària es basa en anar-hi descartant elements segons siguin majors o menors que l'element que està en el punt mitjà del fragment del vector que resta per tractar. Per tant, el vector ha d'estar ordenat.

iii) Les taules hash tenen una mida pre-fixada al tractar-se de taules i només poden emmagatzemar un nombre fixe d'elements.

Fals. Hi han opcions que permeten l'ús de memòria dinàmica com el hashing obert. En aquest cas, la taula no guarda directament els elements sinó els punters als elements.

iv) L'algorisme de cerca lineal o seqüencial es pot implementar en un vector o en una llista encadenada amb punters.

Cert.

Exercici 2: Algorismes de cerca (20%)

Tasca: Donat el següent vector, aplica l'algorisme indicat per buscar l'element que es desitja. Escriu el resultat i la seqüència d'índexs consultats en el procés de cerca pel vector:

1	2	3	4	5	6	7	8	9	10	11	12
7	8	9	28	34	35	37	45	60	73	80	96

i) Aplica l'algorisme de *cerca* **lineal** o **seqüencial** per trobar l'element 45.

L' element 45 està en la posició 8 del vector:

índex 1: 7 ≠ 45
índex 2: 8 ≠ 45
índex 3: 9 ≠ 45
índex 4: 28 ≠ 45
índex 5: 34 ≠ 45
índex 6: 35 ≠ 45
índex 7: 37 ≠ 45
índex 8: 45 = 45 Trobat!

ii) Aplica l'algorisme de *cerca* **lineal** per trobar l'element 12.

L'element 12 no es troba en el vector:

índex 1: 7 ≠ 12
índex 2: 8 ≠ 12
índex 3: 9 ≠ 12
índex 4: 28 ≠ 12
índex 5: 34 ≠ 12
índex 6: 35 ≠ 12
índex 7 37 ≠ 12
índex 8: 45 ≠ 12
índex 9: 60 ≠ 12
índex 10: 73 ≠ 12
índex 11 80 ≠ 12

índex 12: 96 ≠ 12

Si es coneix a priori que el vector està ordenat, llavors també és correcta la solució que atura la cerca en l'índex 4, ja que a partir d'aquest índex tots els elements seran majors que 28, i per tant el valor 12 no estarà entre ells.

iii) Aplica l'algorisme de **cerca binària** per trobar l'element 45.

L'element 45 es troba en la posició 8 del vector:

E=1 D=12	(1+12) /2 = 13/2 = 6	índex 6: 35 < 45
E=7 D=12	(7+12) /2 = 19/2 = 9	índex 9: 60 > 45
E=7 D=8	(7+8) /2 = 15/2 = 7	índex 7: 37 < 45
E=8 D=8	(8+8) /2 = 16/2 = 8	índex 8: 45 = 45. Trobat

iv) Aplica l'algorisme de **cerca binària** per trobar l'element 12.

L'element 12 no es troba en el vector:

E=1 D=12	(1+12) /2 = 13/2 = 6	índex 6: 35 > 12
E=1 D=5	(1+5) /2 = 6/2 = 3	índex 3: 9 < 12
E=4 D=5	(4+5) /2 = 9/2 = 4	índex 4: 28 > 12
E=4 D=3		No trobat

Exercici 3: Ús dels algorismes de cerca (20%)

Tasca: Tornant al sistema informàtic per a la matriculació d'escoles bressol, en aquest exercici s'utilitzaran els tipus de dades presentats a continuació i que estan basats en la solució de l'exercici 1 de la PAC1.

```
const
        HASH_SIZE : integer := 100;
        MAX_OPTIONS : integer := 4;
end const

tApplication = record
        idChild : integer;
        idNursery : integer;
        points : integer;
end record

tChild = record
        id : integer;
        applications : vector [MAX_OPTIONS] of tApplication;
        numApp: integer;
        next: pointer to tChild;
end record

tEnrollment = record
        children : vector [HASH_SIZE] of pointer to tChild;
        numChild : integer;
end record
```

Se'ns demana implementar la funció que a partir de la matriculació (**e**) i un codi d'alumne (**idChild**) retorna la màxima puntuació que ha aconseguit l'alumne en les seves sol·licituds de matriculació. Si l'alumne no existeix o no té cap sol·licitud de matriculació llavors es retornarà el valor -1.

El camp **children** del tipus **tEnrollment** està implementat com una taula hash oberta**.**

i) Indica quina funció hash definiries en la implementació de la taula hash **children** per accedir amb el codi de l'alumne.

La funció de hash hauria de ser 1 més el residu de la divisió del codi de l'alumne entre la mida del vector hash.

hash := 1 + (idChild **mod** HASH_SIZE);

ii) Implementa en **llenguatge algorítmic** la funció descrita en l'enunciat amb la següent capçalera seguint el mètode **hashing obert**:

function get_maximum_score (e:tEnrollment, idChild:**integer**): **integer**

```
var
    hash, ii: integer;
    stop: boolean;
    score: integer;
    c: pointer to tChild
end var

hash := 1 + (idChild mod HASH_SIZE);
c := e.children[hash];
stop := false;
score := -1;

while c ≠ NULL and not stop do
    if (c^.id = idChild) then
        for ii := 1 to c^.numApp do
            if (c^.applications[ii].points > score) then
                score :=  c^.applications[ii].points;
            end if
        end for
        stop := true;
    else
        c := c^.next;
    end if
end while

return score;
```

```
end function
```

Exercici 4: Conceptes bàsics sobre complexitat (20%)

Tasca: Respon les preguntes següents:

i) De qué depèn el temps d'execució d'un algorisme?

El temps d'execució d'un algorisme depèn habitualment de: 1) les dades d'entrada; 2) la qualitat del codi generada pel compilador; 3) la màquina on s'executa el programa; 4) la complexitat de temps de l'algorisme base del programa.

ii) Calcula la complexitat computacional de les següents funcions de temps d'execució usant la notació asimptòtica:

a.	$T(n) = 1000000 + n$	$O(n)$	lineal
b.	$T(n) = 1^n$	$O(1)$	constant
c.	$T(n) = 10^n$	$O(10^n)$	exponencial
d.	$T(n) = 2 + \log(n+1)$	$O(\log(n))$	logarítmica
e.	$T(n) = (2+n).\log(3n)$	$O(n.\log(n))$	quasi-lineal
f.	$T(n) = n + 1000.n^2$	$O(n^2)$	quadràtica

iii) Ordena les complexitats computacionals de l'apartat anterior de més eficient a menys eficient.

b, d, a, e, f, c

iv) Perquè per analitzar l'eficiència d'un algorisme es fa referència al pitjor cas, és a dir al pitjor escenari possible respecte a les dades d'entrada?

D'aquesta forma s'acota el temps d'execució de l'algorisme a analitzar. Si es fes referència al cas mitjà, seria necessari calcular la mitjana de tots els temps d'execució possibles de l'algorisme. Per calcular aquest temps d'execució mitjà s'hauria de tenir informació sobre l'entrada mitjana o la probabilitat de que cada possible entrada sigui utilitzada. Aquesta informació no sempre està disponible, i d'altra banda, si l'estigués, el càlcul pot ser matemàticament molt complex.

Exercici 5: Anàlisi d'algorismes (30%)

Tasca: Respon a les preguntes següents.

i) Donada la següent definició del temps d'execució, calcula la funció *T(n)* sense que aparegui cap definició recursiva i explica el procés que has seguit per obtenir el resultat:

$$T(n) = \begin{cases} 1, & \text{si } n = 1 \\ n \cdot T(n-1), & \text{si } n > 1 \end{cases}$$

El mètode per resoldre aquest tipus d'equacions consisteix en aplicar la definició recursiva de forma repetida un total de *i* vegades, fins que veiem quina forma pren l'expressió resultant:

$$\begin{aligned} T(n) &= \mathbf{n \cdot T(n-1)} = -n \cdot ((n-1) \cdot T((n-1)-1)) = \\ &= \mathbf{n \cdot (n-1) \cdot T(n-2)} = n \cdot (n-1) \cdot ((n-2) \cdot T((n-2)-1)) = \\ &= \mathbf{n \cdot (n-1) \cdot (n-2) \cdot T(n-3)} = \\ &= \ldots = \\ &= \mathbf{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-i) \cdot T(n-i)} \end{aligned}$$

Esbrinem quin valor ha de prendre *i* per poder aplicar el cas base de la definició de *T(n)*:

$$\begin{aligned} n - i &= 1 \\ n &= 1 + i \\ i &= n - 1 \end{aligned}$$

A continuació, completem el càlcul aprofitant que l'expressió *T(n - i)* quan *i = n - 1* es transforma en *T(1)*, que segons la definició de l'enunciat és 1:

$$\begin{aligned} T(n) &= n \cdot (n-1) \cdot (n-2) \cdot \ldots (n-(n-1)) \cdot T(n-(n-1)) = \\ &\quad n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \cdot T(1) = n! \cdot 1 = \mathbf{n!} \end{aligned}$$

ii) Calcula la complexitat computacional de la funció **mistery** i explica el procés que has seguit per obtenir el resultat.

```
function místery (v: vector[N]  of integer, num: integer, e: integer) : boolean
Pre: { 1 ≤ num ≤ N }
  var
        res: boolean;
  end var
        if num = 0 then                                  (1)
                res := FALSE                             (2)
        else                                             (3)
                if v[num] = e then                       (4)
                        res := TRUE;                     (5)
                else                                     (6)
                        res := mistery (v, num-1, e);    (7)
                end if                                   (8)
        end if                                           (9)
        return res;                                      (10)
end function
```

La funció mistery, és una funció recursiva, per la qual cosa hem de calcular el temps del cas base i el temps de la crida recursiva, per poder arribar a una equació de recurrència la qual resoldrem.

La funció comença amb l'avaluació d'una condició més una assignació (línies 1 i 2). Ambdues són operacions elementals amb temps constant, que agrupem en la constant k_1.
Si anomenem *n* al paràmetre *num*, el temps corresponent al cas base es pot expressar com:

$T(n) = k_1$ si $n=0$

El següent pas és calcular el cost de la crida recursiva. Aquest consta d'una avaluació d'una condició (línia 1), més una altra avaluació (línia 4). S'ha de sumar també el cost de la branca més costosa del condicional donat que considerem el pitjor cas, és a dir el cost de l'assignació del resultat de la crida recursiva (línia 7). Totes aquestes operacions elementals tenen cost constant, i els agrupem en la constant k_2.
La crida recursiva té temps donat per la funció:

$T(n) = k_2 + T(n-1)$ si $n \geq 1$

Per tant hem de resoldre ara la part recurrent de la expressió de temps, aplicant i vegades la funció de temps:

$T(n) = k_2 + T(n-1) = k_2 + (k_2 + T((n-1)-1)) = 2.k_2 + T(n-2) =$
$= 2.k_2 + (k_2 + T((n-2)-1) = 3.k_3 + T(n-3) =$
$= \ldots =$
$= i.k_2 + T(n-i)$

Calculem el valor de la variable i que fa que es pugui utilitzar la part no recurrent de la funció temps:

$n - i = 0$
$i = n$

Finalitzem el càlcul de T(n) amb el valor calculat per a la variable i:

$T(n) = n.k_2 + T(n - n) = n.k_2 + T(0) = n.k_2 + k_1$

La conclusió és que la funció té una complexitat **O(n)**, on n és el nombre d'elements (paràmetre *num*) a recòrrer del vector d'entrada *v*.