ANÀLISI I DISSENY D'ALGORISMES (CODI: 102783)
SEGON PARCIAL: 14/01/2016

NOM I COGNOMS: ..... NIU: .....

IMPORTANT: Responeu en l'espai disponible a sota de cada pregunta

1. (5 punts) Són certes o falses, les següents afirmacions? Cada afirmació val 0.25 punts.
   1) El Branch & Bound s'aplica exclusivament a problemes d'optimització.
   2) Un algorisme de Las Vegas és preferible a un algorisme de Monte Carlo per resoldre un problema.
   3) La Programació Dinàmica no analitza tots els possibles casos, només aquells que intervenen en la solució òptima.
   4) El Branch & Bound està dissenyat per trobar totes les solucions factibles.
   5) Un algorisme Probabilístic és determinista.
   6) La finalització de l'algorisme de Branch & Bound sempre està garantida.
   7) A les funcions, les precondicions es solen referir als paràmetres de la funció.
   8) La recursivitat és la forma natural d'implementar la Programació Dinàmica.
   9) El Branch & Bound utilitza una llista de nodes vius per decidir el següent node a obrir.
   10) Les postcondicions permeten detectar més errors de programació que les precondicions.
   11) La Programació Dinàmica té una complexitat base polinòmica.
   12) Una precondició, postcondició o invariant és una condició que sempre s´ha de complir quan el programa l'executa.
   13) Una invariant de classe s'ha de complir al mig de l'execució d'un mètode de la classe.
   14) La Programació Dinàmica evita calcular subproblemes iguals més d'un cop.
   15) Branch & Bound i backtracking construeixen les solucions de la mateixa manera.
   16) Combinar algorismes probabilístics amb algorismes no probabilístics pot reduir el temps de càlcul en el cas mig.
   17) En Programació Dinàmica l'espai de solucions dels diferents subproblemes ha de ser disjunt.
   18) La Programació Dinàmica necessita que els problemes es puguin dividir en subproblemes del mateix tipus i de mida més petita.
   19) El Branch & Bound selecciona el següent node a obrir a partir d'acotacions aproximades del cost de la solució.
   20) Un algorisme probabilístic sempre ha de donar una solució.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
V		x	x				x		x		x	x		x		x		x	x	
F	x			x	x	x		x		x			x		x		x			x

2. (1 punt) Quins són els punts clau per desenvolupar un bon algorisme de Branch and Bound? Justifica la resposta.

   - Trobar un bon ordre de recorregut o ramificació dels nodes, és a dir, definir una bona funció de prioritat dels nodes vius per tal que les solucions bones es trobin ràpidament.
   - Trobar una bona funció d'acotació o poda perquè es produeixi el retrocés el més aviat possible.

3. (2 punts) Suposem que volem invertir exactament 1000 euros, que podem repartir en 4 fons d'inversió diferents en quantitats múltiples de 100. Els interessos que ens donen cadascun dels fons, en funció de la quantitat invertida, es resumeixen a la següent taula:

	0	100	200	300	400	500	600	700	800	900	1000
fons 1	0	0.01	0.01	0.01	0.01	0.04	0.05	0.06	0.10	0.11	0.11
fons 2	0	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.06	0.06
fons 3	0	0.01	0.01	0.01	0.05	0.05	0.05	0.05	0.05	0.05	0.05
fons 4	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10

Volem maximitzar el rendiment de la inversió dels nostres diners mitjançant la programació dinàmica. Per aquest motiu:

(a) Verifica que aquest problema compleix el principi d'optimalitat de Bellman.

Si $I_4(1000) = f_1(x_1) + \cdots + f_4(x_4)$ és el resultat d'una seqüència de decisions i és òptima per invertir 1000 euros en 4 bancs, aleshores $I_3(1000 - x_4) = f_1(x_1) + \cdots + f_3(x_3)$ és òptima per invertir $1000 - x_4$ euros en 3 bancs.

(b) Crea la funció recursiva de la solució.

$$I_n(x) = \begin{cases} f_1(x), & \text{n=1;} \\ Max_{\{t=0,100,\cdots,x\}}\{I_{n-1}(x-t) + f_n(t)\}, & n > 1. \end{cases}$$

(c) Omple la taula de resultats parcials.
La primera taula conté els interessos parcials per cada quantitat invertida (columnes) amb els fons disponibles (files). És a dir, la posició (i,j) ens diu quins interessos obtindrem d'invertir j euros en els i primers fons.

	0	100	200	300	400	500	600	700	800	900	1000
I_1	0	1	2	3	4	20	30	42	80	99	110
I_2	0	2	4	9	12	20	30	42	80	99	110
I_3	0	2	4	9	20	25	30	42	80	99	110
I_4	0	2	4	9	20	25	36	49	80	99	110

La segona taula conté la quantitat/100 a invertir a cada fons disponible. És a dir, a la posició (i,j) tindrem un vector de i posicions on cada posició conté la quantitat/100 a invertir per obtenir els interessos de la mateixa casella a la taula 1. En cas que tinguem vàries opcions, només n'he escrit una.

	0	100	200	300	400	500	600	700	800	900	1000
S_1	0	1	1	1	1	1	1	1	1	1	1
S_2	0	0,1	0,2	0,3	0,4	5,0	6,0	7,0	8,0	9,0	10,0
S_3	0	0,1,0	0,2,0	0,3,0	0,0,4	0,0,5	0,0,6	7,0,0	8,0,0	9,0,0	10,0,0
S_4	0	0,1,0,0	0,0,0,2	0,0,0,3	0,0,4,0	0,0,0,5	0,0,0,6	0,0,0,7	8,0,0,0	9,0,0,0	10,0,0,0

(d) Construeix la solució.

En aquest cas, tal i com hem construït les taules, no cal reconstruir la solució, ja que de la primera taula sabem que l'última casella és l'interès màxim que podem treure i per tant l'última casella de la segona taula ens dóna la solució.

4. (1 punt) Afegeix les precondicions i l'invariant de la classe CStack

```
public class CStack {
    private int m_Stack[];
    private int m_Size;
    private int m_Top;
    public CStack(int size) {
        m_Size=size;
        m_Stack=new int[m_Size];
        m_Top=-1;
    }
    public void Push(int element) {
        m_Stack[++m_Top]=element;
    }
    public int Pop() {
        return m_Stack[m_Top--];
    }
}
```

```
public CStack(int size)                 assert(Size>0);
public void Push(int element)           assert(m_Top<m_Size-1);
public int Pop()                        assert(m_Top>-1);

Invariant:
m_Stack!=null &&
m_Top>=-1 &&
m_Top<m_Size &&
m_Size>0
```

5. (1 punt) ¿Què és i com es pot aprofitar l'avantatge estocàstica en algorismes de Monte Carlo i Las Vegas? Posa la fórmula general pels dos casos per a un algorisme p-correcte. Fes un exemple amb un algorisme de Monte Carlo 0.7-correcte i un algorisme de Las Vegas 0.7-correcte.

   - Avantatge estocàstica: Suposem un algorisme p-correcte, amb $p > 1/2$. Es defineix com a avantatge de l'algorisme a $p - 1/2$. Qualsevol algorisme de Monte Carlo/Las Vegas amb un avantatge positiu es pot transformar en un altre amb una probabilitat d'error tan petita com vulguem.
   - Donat un algoritme de Monte Carlo p-correcte, podem millorar la probabilitat d'obtenir un resultat correcte si l'executem n vegades. La probabilitat d'encertar puja a $1-(1-p)n$. Com no sabem quin dels resultats és correcte, utilitzarem la moda (resultat més repetit) com a resultat de les n execucions. La probabilitat de que la moda coincideixi amb el resultat es calcula amb la següent formula de la resposta correcta majoritària: $\sum_{i=\lfloor \frac{n}{2} \rfloor+1}^{n} p^i(1-p)^{n-i} \begin{pmatrix} n \\ i \end{pmatrix}$
   - Exemple: Monte Carlo per n=5 i p=0,7, probabilitat de resposta correcta majoritària: 0,836920
   - Donat un algoritme de Las Vegas p-correcte, podem millorar la probabilitat d'obtenir un resultat correcte si l'executem n vegades. La probabilitat d'encertar puja a 1-(1-p)n i, a diferència dels algoritmes de Monte Carlo, els de las Vegas ens diuen si el seu resultat és correcte. Per tant, La probabilitat d'obtenir el resultat correcte per n execucions serà: 1-(1-p)n
   - Exemple: Las Vegas per n=5 i p=0,7, probabilitat de resposta correcta: 0,997570