Examen d'Anàlisi i Disseny d'Algorismes
segon parcial 17 de desembre de 2014

Nom:	NIU:

Cognoms:

Nota: les preguntes de la 1 a la 5 valen 1 punt sobre 10 mentre que la 6 val 5 punts.

1.- En què s'assembla i en què es diferencia la programació dinàmica i divide and conquer? Explica-ho fent servir l'exemple de fibbonacci.
PD i D&C s'assemblen en que els dos divideixen el problema a resoldre en subproblemes més simples amb la mateixa estructura, i la solució del problema és la composició de les solucions els subproblemes.

PD i D&C es diferencien en que mentre D&C necessita que els subproblemes siguin disjunts PD necessita que siguin dependents, i per no fer recàlculs innecessàris reaprofita els càlculs comuns, mitjançant una taula o vector.

Això ho veiem a fibbonnacci, a on si dividim el problema en subproblemes més simples: per exemple fib(n)=fib(n-1)+fib(n-2), tenen molts càlculs compartits. Per això veiem que, tot i que es pot solucionar el problema utilitzant les dues tècniques, PD és la tècnica adequada mentre D&C té una complexitat innecessàriament elevada pel fet de no dividir el problema en subproblemes disjunts i per tant haver de fer aquest recàlcul.

2.- Com han de ser els problemes per poder-los resoldre utilitzant la tècnica de programació dinàmica?
S'han de poder dividir en subproblemes simples amb la mateixa estructura.
La solució del problema ha de ser la composició de les solucions dels subproblemes (subestructura òptima dels subproblemes Bellman)
L'espai de solucions dels diferents subproblemes no és disjunt.

3.- Enumera dues diferències entre un algorisme de Monte Carlo 0,5-correcte esbiaixat i llençar una moneda a l'aire per decidir.
L'algorisme de MC 0,5-correcte esbiaixat ens assegura que és cert en alguna de les seves respostes, mentre que tirar una moneda no.
L'algorisme de MC-05-correcte esbiaixat si l'executem un nombre k de vegades millora la seva probabilitat d'encert a $0,5^k$

4.- Explica què és un algorisme de Sherwood i posa un exemple.
Són algorismes pels quals existeix una solució determinista que és molt més ràpida en mitjana que en el pitjor dels casos. Els algorismes de Sherwood, afegint aleatorietat redueixen temps entre entrades ("treuen eficiència de les entrades riques per donar-la a les pobres".
Un exemple és el quicksort amb la tria del pivot aleatòria.

5.- Als orígens de la teoria de la computabilitat es van plantejar 4 preguntes bàsiques i es va trobar la seva resposta. Quines varen ser aquestes 4 preguntes i quines implicacions tenen a la computació actual.
Les preguntes eren: és possible fer un algorisme que respongui a aquestes 4 preguntes de forma automàtica?
Problema d'equivalència: Calculen els algorismes A i B la mateixa funció?
Problema de parada: Pararà l'algorisme A per a una de les seves entrades?
Problema de la totalitat: Pararà l'algorisme A per a totes les seves entrades?
Problema de verificació: Calcula l'algorisme A la funció f?
La resposta va ser que no es podia construir l'algorisme que respongués a tot això.
Les implicacions actuals són que hem de depurar els programes i fer jocs de proves estrictes per tal de validar-ho nosaltres manualment.

6.- Volem comprar el bitllet de tren, per anar d'una ciutat europea a una altre, que ens surti més barat. El trajecte que va de la ciutat d'origen **C1** a la de destí **Cm** passa per m ciutats comptant la d'origen i destí: **C1,C2,...,Cm-1,Cm**. Podem comprar bitllets a qualsevol estació del trajecte per anar a qualsevol de les següents estacions del trajecte que tenim per davant, i cada bitllet té un preu diferent, i no sempre comprar el trajecte més llarg té un cost inferior. Tenim una taula de preus **P** de m-1xm-1 valors reals, tal que **P[i,j]** indica el preu de comprar un bitllet per anar de la ciutat **i** a la **j+1**, sent i<j+1. Volem fer un algorisme, utilitzant programació dinàmica, que calculi el preu més barat d'anar de la ciutat C1 a la Cm i ens digui quins bitllets hauríem de comprar.

Per exemple, suposem que tenim 4 ciutats, i volem anar de la ciutat 1 a la 4 amb la següent taula de preus:

	C2	C3	C4
C1	Preu anar de C1 a C2=20	15	50
C2	-	12	30
C3	-	-	15

El preu menor per comprar un bitllet per anar de C1 a C4 seria 30 i es faria comprant bitllet de C1 a C3 i de C3 a C4.

Feu l'algorisme de programació dinàmica que resolgui el problema genèric i doneu la taula o vector que donaria de resultat amb aquest problema concret. Indiqueu la funció recursiva que us servirà per omplir la taula o vector i quina interpretació té cada casella. Recordeu que necessitem el **valor final** del preu dels bitllets i **quins bitllets** hauríem de comprar.

Idea:Si tenim un vector B de n-1 posicions a on a B[1] tenim el preu mínim d'anar de l'estació 1 a la 2, a B[2] el d'anar de la primera estació a la 3a, i així successivament, al final a la última posició B[n-1] tindrem el preu mínim d'anar de la primera estació a la última. Això ho podem fer de manera incremental. Primer calculem B[1] que és el preu mínim d'anar de l'estació 1 a la 2 (que serà el bitllet directe que trobem a P[1,1]). Després calculem B[2], això implica mirar les diferents alternatives: primer anar directe de 1 a 3 (P[1,2]) o anar de 1 a 2 (ja calculat a B[1]) i sumar anar de 2 a 3 P[2,2]. Després calculem B[3], aquí mirarem el mínim entre: anar directes de 1 a 4 P[1,3], o anar de 1 a 2 (B[1]) +de 2 a 4 directe (P[2,3]), o de 1 a 3 B[2] +de 3 a 4 P[3,3]. I així successivament, es veu clar que d'aquesta manera tenim un càlcul progressiu que anem aprofitant a mesura que avancem al vector. Per tenir els bitllets a comprar necessitarem afegir un camp més al vector i guardar a cada cas de quina estació venim. Al final per trobar el preu el tindrem a B[n-1] i alla tindrem de quina estació venim (per exemple la j) de manera que anirem a B[j-1] i allà trobarem la següent estació anterior fins que arribem a 1.

Taula:

	Preu Mínim Fins 2	Preu Mínim Fins 3	Preu Mínim Fins 4
Preu	20	15	30
Estació origen	1	1	3

Funció Recursiva per omplir preu i origen:

$$B[i].preu = \text{mínim} \begin{cases} P[1,i], & \text{si aquest és el mínim el } B[i].origen=1 \\ B[1]+P[2,i], & \text{si aquest és el mínim el } B[i].origen=2 \\ \ldots. & \\ B[i-1]+P[i,i] & \text{si aquest és el mínim el } B[i].origen=i \end{cases}$$

```
//Suposem N>=3, sino no te sentit
Algorisme BitlletBarat(in: P:array[N-1,N-1]de real, out B: array[N-1] de structura{preu: real,origen:enter})

    B[1].preu=P[1,1] //Preu minim d'anar de l'estació 1 a la 2
    B[n-1].desti=1

    For i=2 to n-1
        B[i].preu=P[1,i-1]
        B[i].origen=1
        For j=1 to i-1
            If B[1].preu>B[j].preu+P[j+1,i-1] then
                B[i].preu= B[j].preu+P[j+1,i-1]
                B[i].origen=j+1
            Endif
        Endfor
    Endfor

    Write("El preu del trajecte mes barat es:",B[n-1].preu)
    K2=n-1
    K1=B[n-1].origen
    While k1>1 fer
        Write("has de comprar bitllet de",k1, "a",k2)
        K2=k1
        K1=B[k2-1].origen
    Fiwhile
    Write("has de comprar bitllet de 1 a",k2)
FiAlgorisme
```