FIB

PROGRAMACIÓ i ALGORÍSMIA - 2

Col·lecció de Problemes

Recull per Jordi Delgado
(Dept. CS, UPC)

Grau d'IA, 2022-23
FIB (UPC)

Problemes diversos (repassant PA1)

1. Donada una matriu quadrada (representada com a llista de llistes, amb una llista per cada fila), feu una funció que ens digui si la matriu és un *quadrat màgic* o no.
   Un quadrat màgic és una matriu quadrada N×N tal que *tots* els nombres entre 1 i N^2 estan disposats de manera que les seves files, les seves columnes i les seves diagonals principals sumen el mateix. Per exemple, [[6,1,8],[7,5,3],[2,9,4]] és un quadrat màgic.

2. Vam veure que l'*ordenació per fusió* tenia una complexitat Θ**(N·log(N))** en el cas general. Si filem una mica més prim, i posem condicions sobre la llista a ordenar, què podeu dir sobre el que costa aquest algorisme si:
   a) La llista a ordenar conté nombres triats a l'atzar
   b) La llista a ordenar ja està ordenada
   c) La llista a ordenar està ordenada a l'inrevés
   d) La llista a ordenar conté el mateix element en les seves N posicions

3. Una llista $[a_0,...,a_{N-1}]$ d'N elements (amb N > 2) s'anomena *unimodal* si existeix un índex p, amb $0 \leq p < N$, tal que $a_0 < a_1 < ... < a_p$ i $a_p > a_{p+1} > ... a_{N-1}$, és a dir, els elements de la llista creixen fins a p, i després decreixen. L'element a_p s'anomena el *cim*. Feu una funció de cost Θ**(log(N))** que, donada una llista unimodal, trobi el seu cim (fixeu-vos que podeu *argumentar* el cost de l'algorisme, però no sabeu *demostrar-lo* encara).
   Pista: Recordeu la `cerca_eficient`.

4. *Maximum Subsequence problem*: Sigui L una llista $L = [a_0,...,a_{N-1}]$ de nombres enters, que suposarem no pot ser buida (N > 0). Considerem totes les seves subllistes possibles. Volem trobar quant suma la subllista amb suma màxima. Per exemple, si L = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84] el resultat és 187, que correspón a la subllista L[2:7]. Formalment podem escriure que volem trobar `max{sum(L[i:j]), 0 ≤ i < N, 0 ≤ j ≤ N}`. En Python podríem resoldre el problema amb l'expressió:
   ```
   max([sum(ll[i:j]) for i in range(len(ll)) for j in range(i+1,len(ll)+1)])
   ```
   on `ll` és la llista en qüestió, però això no és gaire eficient.
   a) Quin cost penseu que té aquesta solució?
   b) Mireu de trobar un algorisme *millor* per resoldre aquest problema.
   c) Raoneu per què penseu que el vostre algorisme té un cost inferior al que trobeu a l'apartat a).
   Pista 1: L'expressió donada més amunt equival a aquest algorisme (on `alist` és la llista donada):
   ```
   maxSum = float('-inf')
   # provem totes les subseqüències possibles
   for i in range(len(alist)):
       for j in range(i,len(alist)):
           thisSum = 0
           for k in range(i,j+1):
               thisSum += alist[k]
           if thisSum > maxSum:
               maxSum = thisSum
   return maxSum
   ```
   Pista 2: Recordeu la `cerca_eficient`.

5. Feu una funció que, donada una llista, retorni **`True`** si tots els elements són diferents, i **`False`** en cas contrari (per simplificar suposarem que tots els elements de la llista són del mateix tipus, és a dir, o bé tots són nombres, o bé tots són *strings*, etc).
   Pista: No compareu tots amb tots. Feu servir un diccionari.

Heaps

6. Són ***min-heaps*** les següents llistes?
   a) `[None,12,24,34,45,24,79]`
   b) `[None,13,20,44,18,35,67,98,67,69,47]`
   c) `[None,10,14,15,23,38,44,46,51,98]`
   d) `[None,69,97,29,50,42,24,61,14,95,83,98,83,79,5,66]`
   e) `[None,2]`

7. Són ***max-heaps*** les següents llistes?
   a) `[None,76,34,56,12,21,11]`
   b) `[None,93,50,22,26,35,21,12,17,27]`
   c) `[None,98,51,46,44,38,23,15,14,10]`
   d) `[None,15,44,65,73,21,52,42,5,43,27,25,6]`
   e) `[None,12]`

8. Començant amb un ***min-heap*** buit, inserir, en l'ordre donat, els següents nombres: 45, 67, 23, 46, 89, 65, 12, 34, 98, 76. Quin és el ***min-heap*** resultant? El podeu dibuixar en forma d'arbre binari, o escriure la llista corresponent.

9. Començant amb un ***max-heap*** buit, inserir, en l'ordre donat, els següents nombres: 45, 67, 23, 46, 89, 65, 12, 34, 98, 76. Quin és el ***max-heap*** resultant? El podeu dibuixar en forma d'arbre binari, o escriure la llista corresponent.

10. Començant amb aquest ***max-heap***: [None,84,52,29,36,45,23,21,18,12,13], obtingueu el màxim element, un darrera l'altre fins que el ***max-heap*** es quedi buit. Detalleu com queda el ***max-heap*** a cada pas.

11. * Fer una funció que, donades **N** llistes ordenades creixentment i un nombre **k**, retorni el k-èssim element més petit, considerant *tots* els elements de *totes* les N llistes. La funció ha de tenir un cost $\Theta(k \cdot log(N)+N)$.

Arbres (I)

12. Els arbres binaris, tal com els vam implementar a la transparència 55 de teoria, són arbres immutables. Com que vam fer servir un diccionari per implementar-los, aquesta implementació es pot ampliar per poder modificar els arbres (i que aquests siguin *mutables*). Feu les funcions

```
def modifica_valor_arbre(a,v):
    # Pre: a és un arbre binari no buit
def modifica_fill_esq(a,fesq):
    # Pre: a i fesq són arbres binaris. a no és buit
def modifica_fill_dre(a,fdre):
    # Pre: a i fdre són arbres binaris. a no és buit
```

13. Feu una funció que, donat un arbre, retorni una llista amb els elements de les fulles de l'arbre (transparència 40 de teoria).

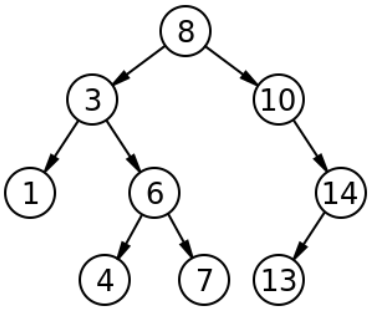
14. Donats els recorreguts en pre-ordre i post-ordre d'un arbre *binari*, podem determinar com és l'arbre? Si penseu que sí, demostreu-ho. En cas contrari, proporcioneu un contra-exemple.

15. Anomenem *full* (en anglès) a un arbre binari si cada node té o bé zero fills, o en té dos (no hi ha cap node amb un fill només). Dibuixeu els arbres *binaris full* corresponents a:
    a.– Pre-ordre: 2 7 3 6 1 4 5; Post-ordre: 3 6 7 4 5 1 2
    b.– Pre-ordre: 3 1 7 4 9 5 2 6 8; Post-ordre: 1 9 5 4 6 8 2 7 3

16. Dibuixeu els arbres *binaris* corresponents als recorreguts següents:
    a.– Pre-ordre: 3 6 1 8 5 2 4 7 9; In-ordre: 1 6 3 5 2 8 7 4 9
    b.– Nivells: 4 8 3 1 2 7 5 6 9; In-ordre: 1 8 5 2 4 6 7 9 3
    c.– Post-ordre: 4 3 2 5 9 6 8 7 1; In-ordre: 4 3 9 2 5 1 7 8 6

17. Feu una funció que, donats el recorregut en pre-ordre i el recorregut en in-ordre d'un arbre *binari*, construeixi l'arbre binari corresponent. Suposarem que els elements de l'arbre són tots diferents.

18. Un Arbre Binari de Cerca (BST, de *Binary Search Tree*) és un arbre binari tal que tots els elements del fill esquerre d'un node són més petits que el node, i tots els elements del fill dret del node són més grans que el node. I això es verifica *per a tot node* de l'arbre. Per exemple, aquest arbre binari:



    és un BST. Demostreu que el recorregut en in-ordre d'un BST visita els elements de l'arbre en ordre creixent (el resultat és una llista amb els elements de l'arbre ordenats).

19. Implementeu una funció que, donat un arbre binari i un nombre **x**, retorni el nombre d'elements de l'arbre que són estrictament més grans que **x**. Si l'arbre fos un BST (és a dir, un arbre binari que satisfà la propietat de BST), es podria fer una funció més eficient per resoldre el mateix problema?

20. Si considerem un ***min-heap*** com un arbre binari (tal com vam fer a la sessió 1 de laboratori) i us diuen que el recorregut en post-ordre d'aquest arbre és: 17 15 21 13 6 8 4 2, com és el ***min-heap***?
    Pista: Quina forma té l'arbre corresponent a un ***min-heap*** amb 8 elements?

Grafs

21. El *transposat* d'un graf dirigit $G = (V,E)$ és el graf dirigit $G^T = (V,E^T)$ on definirem
$$E^T = \{(v,u) \mid (u,v) \in E\}$$
Feu una funció que, donat un graf dirigit G, calculi el graf G^T transposat.
La funció ha de ser *destructiva* (no ha de retornar res, ha de modificar G), tot i que es permet la utilització d'un graf auxiliar mentre es fa el càlcul. Suposarem que el graf G està implementat amb llistes d'adjacència, en el format que hem fet servir a les sessions de laboratori.

22. El *quadrat* d'un graf $G = (V,E)$ és el graf $G^Q = (V,E^Q)$ on definirem
$$E^Q = \{ (u,v) \mid \exists\ w \in V\ t.q.\ (u,w) \in V \wedge (w,v) \in V \}$$
Feu una funció que, donat un graf G, calculi el quadrat G^Q d'un graf G.
La funció no ha de ser *destructiva* (ha de retornar el nou graf, no ha de modificar G). Suposarem que el graf G està implementat amb llistes d'adjacència, en el format que hem fet servir a les sessions de laboratori.

23. Considerem un graf no dirigit $G = (V,E)$, amb $n = |V|$ vèrtexos i $m = |E|$ arestes, implementat amb llistes d'adjacència, en el format que hem fet servir a les sessions de laboratori. Tenim la funció:
```
def misteri(G):
    x = 0
    for i in range(len(G)):
        for j in range(len(G[i])):
            x += 1
    return x
```
a) Explica breument què retorna la funció
b) Quin és el cost d'aquesta funció?

24. Modifiqueu l'algorisme de Dijkstra que hem vist a classe de manera que calculi el camí més curt des d'un vèrtex **x** a un vèrtex **y** donats: `def dijkstra_mod(G,x,y)`. Hauria de retornar la distància d'**x** a **y** (`inf` si no es pot anar d'**x** a **y**) i una llista amb el camí (`None` si no es pot anar d'**x** a **y**).

25. Modifiqueu l'algorisme de Dijkstra que hem vist a classe de manera que compti **quants** camins òptims (de mínim cost; pot haver-ne més d'un) hi ha entre dos vèrtexos **x** i **y**: `def dijkstra_camins(G,x,y)`. Hauria de retornar un nombre, 0 si no n'hi ha cap. Suposarem que les arestes del graf tenen costos *estrictament* > 0 (pareu atenció, hem de respondre ***quants*** camins, és a dir, hem de comptar-los; NO hem de retornar els camins).

Classes i Objectes

Suposem que definim una classe per representar punts en un plà:

```
class Punt:

    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def getX(self):
        return self.__x
    def getY(self):
        return self.__y

    def distancia(self, p):
        """ Pre: p és instància de la classe Punt """
        dx = self.getX() - p.getX()
        dy = self.getY() - p.getY()
        return sqrt(dx*dx + dy*dy)

    def radi(self):
        return sqrt(self.getX()*self.getX() + self.getY()*self.getY())

    def angle(self):
        if (self.getX() == 0 and self.getY() == 0):
            return 0;
        return atan(self.getY()/self.getX())

    def __add__(self, p):
        """ Pre: p és instància de la classe Punt """
        return Punt(self.getX() + p.getX(), self.getY() + p.getY())
```

26. Volem canviar la implementació de la classe `Punt` a una implementació que utilitzi coordenades *polars* i no coordenades cartesianes. Identifiqueu quins mètodes cal canviar.

27. Ara, canvieu la implementació i feu servir coordenades polars. Penseu que els mètodes de la classe han de tenir *la mateixa semàntica*, és a dir, s'han de comportar *exactament* igual. Per exemple, els mètodes `getX` i `getY`, que abans eren simples *getters*, ara ja no ho són. A més, recordeu que els punts s'han de continuar creant a partir de les coordenades cartesianes, p.ex. `p = Punt(3,-1)`

28. Què hagués passat si no haguéssim fet servir els *getters* a la resta de funcions? Imagineu que haguéssim escrit el mètode `distancia` així:

    ```
    def distancia(self, p):
        dx = self.__x - p.__x   # podem accedir a atributs privats d'objectes
        dy = self.__y - p.__y   # instància de la mateixa classe que estem definint
        return sqrt(dx*dx + dy*dy)
    ```

    Implementeu aquest mètode dins la classe amb la implementació basada en coordenades polars.

29. Afegiu a la classe `Punt` els mètodes `__str__`, `__repr__` i `__eq__`.

30. A la classe `Racional` (sessió 5 de laboratori) substituïu el mètode `suma_racional` pel mètode `__add__` de manera que pugueu utilitzar l'operador `+`. Feu el mateix amb el mètode `igual`, és a dir, elimineu-lo i escriviu el mètode `__eq__`. Així podreu fer servir l'operador `==`.

31. Afegiu a la classe `Racional` els mètodes `__str__` i `__repr__`.

32. Aquest és un problema divertit: Afegiu `__repr__` a la classe `BinTree`. (sessió 6 de laboratori).

Estructures Dinàmiques de Dades

33. Fer una funció `def palindrom(s)` que determini si una seqüència s (de Python: llista, tupla o *string*) és un palíndrom o no. És a dir, cal indicar si la seqüència es llegeix igual d'esquerra a dreta que a l'inrevés. Utilitzeu una pila (encara que es pugui fer sense).

34. Donat el programa:

    ```
    def escriu(n):
        if n > 0:
            print(' ', n, end='')
            escriu(n - 1)
            escriu(n - 1)
    ```

    Exemples de l'execució d'aquest programa:

    ```
    >>> escriu(2); print()
     2  1  1
    >>> escriu(3); print()
     3  2  1  1  2  1  1
    >>> escriu(4); print()
     4  3  2  1  1  2  1  1  3  2  1  1  2  1  1
    ```

    Reimplementeu `escriu` *iterativament*, de manera que la sortida del programa no canviï.

35. Donat el programa:

    ```
    def escriu(n):
        if n > 0:
            escriu(n - 1)
            print(' ', n, end='')
            escriu(n - 1)
    ```

    Exemples de l'execució d'aquest programa:

    ```
    >>> escriu(2); print()
     1  2  1
    >>> escriu(3); print()
     1  2  1  3  1  2  1
    >>> escriu(4); print()
     1  2  1  3  1  2  1  4  1  2  1  3  1  2  1
    ```

    Reimplementeu `escriu` *iterativament*, de manera que la sortida del programa no canviï.

36. Feu una funció `def avalua(s)` que, donada una *string* representant una expressió aritmètica (amb enters) en *notació polaca inversa*, l'avaluï i retorni el resultat (un nombre enter). Només considerarem les operacions de suma, resta i multiplicació.

    Per exemple, l'expressió `3 + 154` en notació polaca inversa és `3 154 +`, és a dir, primer es donen els dos operands, i després l'operador corresponent. Una expressió més complicada com ara

    `((3 + 4) * (2 - 8)) + (2 + 5)`

    s'escriuria: `3 4 + 2 8 - * 2 5 + +`

37. Fer un programa per resoldre el següent problema: Donada una cua de parells d'enters, on el primer és un identificador d'usuari i el segon el temps esperat de la gestió que vol fer, dissenyeu una operació que reparteixi els usuaris en dues noves cues de tal manera que:

    a) Cap persona esperi més a la nova cua que una que tenia al darrera inicialment.

    b) Totes esperin el mínim temps possible.

    c) Si un usuari pot anar a les dues cues, s'escollira la primera.

    L'entrada és una seqüència de parells d'enters acabada en `0 0` (que representa la cua inicial). La sortida és el contingut de les dues noves cues. Per exemple:

    Entrada: `3 2  6 3  2 5  11 1  8 4  5 3  9 2  1 3  7 4  15 2  4 3  0 0`

    Sortida:
    ```
    3 2
    2 5
    5 3
    1 3
    15 2

    6 3
    11 1
    8 4
    9 2
    7 4
    4 3
    ```

38. Diem que el resultat de *trenar* dues cues q1 i q2 és una cua q3 on els elements de q1 apareixen a les posicions senars (primera, tercera, cinquena, etc) i els elements de q2 apareixen a les posicions parelles (segona, quarta, sisena, etc). Després de l'últim element de la més curta apareixen la resta dels elements de la més llarga. Volem afegir un *mètode* a la classe `Cua` per trenar una altra cua amb **`self`**. Òbviament aquesta operació és *destructiva* per a totes dues cues. Manipuleu les referències a instàncies de **`_Node`** directament, no feu servir *setters* ni *getters*. `trenar` té la següent especificació:

    ```
    def trenar(self,c):
        """
        Operació destructiva per trenar self amb la cua c. c ha d'acabar buida.
        """
    ```

39. Fer una funció **`def`** `comptar(lst,n)` que, donada una llista `lst` de parelles d'enters (al tanto, `lst` és una instància de `Llista`, no una llista de Python) i un número `n`, que compti quants cops apareix `n` com a primer element d'una parella i sumi els segons elements d'aquestes parelles (de les que `n` és el primer element). Ha de retornar aquests dos valors (el comptatge i la suma).

40. Cal fer un programa que calculi estadístiques d'una seqüència de parell d'enters. Cada parell d'enters es composa de parells **<codi nombre>** un codi d'operació (un enter *negatiu*) i d'un enter (en aquest ordre). Si el codi és -1 això vol dir que el nombre que té com a parella compta com a vàlid. Si el codi és -2 llavors vol dir que cal invalidar qualsevol de les aparicions del nombre que segueix a continuació. (seria equivalent al fet que s'hagués esborrat una de les aparicions prèvies vàlides d'aquest element a la seqüència). Si el nombre s'ha d'invalidar però no té cap aparició prèvia vàlida, llavors no cal esborrar res. Cada vegada que processem una parella de la seqüència, cal treure per la sortida estàndard el mínim, al màxim i la mitjana dels elements vàlids que hi hagin al tros de seqüència que haurem processat. En cas que no hi hagi cap element vàlid, llavors cal escriure només un zero. Per exemple:

    Entrada: **-1 1 -1 1 -1 3 -1 2 -1 1 -2 1 -2 2 -2 3 -2 34 0 0**

    Sortida: **1 1 1**

    1 1 1

    1 3 1.66667

    1 3 1.75

    1 3 1.6

    1 3 1.75

    1 3 1.66667

    1 1 1

    1 1 1

    No podeu fer servir cap contenidor dels que Python ofereix (llistes, tuples, diccionaris, etc).

41. Fer una funció `def interseccio(lst1,lst2)` que, donades dues llistes (instàncies de **Llista**) *ordenades* d'enters, retorni una nova llista amb la intersecció de les esmentades llistes. No s'han de visitar elements de forma innecessària ni fer servir estructures auxiliars.

42. Volem implementar dins de la classe `Llista` un mètode nou amb la següent especificació:

```
def esborrar_tots(self,x):
    """
    Operació destructiva on s'han eliminat de self totes les aparicions d'x (la
    resta d'elements queda en el mateix ordre original); si el cursor de self
    referenciava a una aparicio d'x, passa a referenciar al primer element
    diferent d'x posterior a aquesta (si no hi ha cap element diferent d'x, passa
    a la dreta el tot); en cas contrari, el cursor no canvia

    """
```

43. En un problema anterior hem demanat fer un mètode de la classe `Cua` per *trenar* dues cues. Afegiu ara un mètode a la classe **Llista** per ***trenar*** dues llistes, **self** i la llista-paràmetre. L'operació és destructiva, és a dir, **self** és la llista trenada i la llista-paràmetre queda buida:

```
def trenar(self,l):
    """
    Operació destructiva per trenar self amb la llista l. l ha d'acabar buida.
    """
```

Recordar que el resultat de *trenar* dues llistes l1 i l2 és una llista l3 on els elements de l1 apareixen a les posicions senars (primera, tercera, cinquena, etc) i els elements de l2 apareixen a les posicions parelles (segona, quarta, sisena, etc). Després de l'últim element de la llista més curta apareixen la resta dels elements de la llista més llarga.

44. Afegeix a la classe **Llista** un mètode públic `def splice(self,x,lst)` per realitzar la següent operació: Donada una llista `lst`, inserir-la just davant de la primera aparició d' `x` a **self**. És una operació *destructiva*, **self** quedarà modificada i `lst` ha de quedar buida. Si no hi ha `x` a **self**, la llista `lst` s'insereix al final de **self**. El cursor de **self** ha de quedar inalterat. Aquest és un problema on s'han de manipular referències a **_Node**. No s'ha d'utilitzar memòria extra (a part de variables senzilles, com referències a **_Node**, o variables booleanes, etc.) ni s'han de crear ni destruir instàncies de **_Node**.

    Per exemple, si `l1 = [[3,7,-2,5,0,7]]` i `l2 = [[3,4,3,4]]` (representem les instàncies de **Llista** amb els seus elements dins de `[[...]]`), el resultat de fer `l1.splice(7,l2)` serà que `l1 = [[3,3,4,3,4,7,-2,5,0,7]]` i `l2 = [[]]` (llista buida). Si en canvi fem `l1.splice(8,l2)`, el resultat serà `l1 = [[3,7,-2,5,0,7,3,4,3,4]]` i `l2 = [[]]`. Si `l1 = [[]]` (és buida) i `l2 =` L2 i fem `l1.splice(x,l2)` (per a qualsevol x), el resultat serà que `l1 =` L2 i `l2 = [[]]`.

45. Fes una funció `def inverteix_prefix(lst,m)` que, donada una llista de mida `n` i un nombre $0 \leq m \leq n$, retorni una nova llista igual que `lst` però amb els `m` primers elements (prefix de mida `m`) de `lst` invertits.

    Per exemple, si `l1 = [[3,7,-2,5,0,7]]` i `m = 3`, la crida `inverteix_prefix(l1,m)` ha de retornar la (nova) llista `[[-2,7,3,5,0,7]]`.

46. Implementa el mètode `__eq__` en la classe **BinTree**. Així, donades dues instàncies de **BinTree** podrem comparar-les amb l'operador **==**. Fes-ho en les dues implementacions de **BinTree** que coneixem.

47. Feu un programa que llegeixi un arbre binari de paraules (*strings*) no buit, es posi a l'arrel de l'arbre, i després es mogui per l'arbre seguint les ordres que se li donin. L'entrada comença amb la descripció d'un arbre no buit segons es va explicar a classe. Després ve una seqüència de paraules que poden ser "amunt", "esquerra" o "dreta". Escriviu el contingut de la posició inicial, i de la posició després de cada pas. Les ordres que farien sortir fora de l'arbre cal ignorar-les, però escrivint igualment el contingut de la posició.

    Pista (I): No cal modificar la classe BinTree, però us caldrà una pila auxiliar.

    Pista (II): L'entrada són *strings*, així que la marca en la lectura de l'arbre no és `0`, sinò `'0'`

Entrada:	**Sortida**
va tenir sis 0 polls 0 0 xics 0 0 pics **pellarics 0 0 camatorts i 0 becarics 0 0 0** **esquerra** **dreta** **esquerra** **amunt** **amunt** **amunt** **dreta**	**va** **tenir** **xics** **xics** **tenir** **va** **va** **pics** **camatorts** **camatorts**

dreta **dreta** **esquerra** **dreta** **esquerra**	**i** **becarics** **becarics**

48. Feu una funció **def** reflex(a,b) que, donades dues instàncies de **BinTree**, ens digui si són ***reflexos***. Dos arbres binaris a i b són reflexos si les arrels són iguals i el fill dret d'a i el fill esquerre de b són reflexos, i el fill esquerre d'a i el fill dret de b són reflexos. Dos arbres binaris buits són reflexos.

49. Feu un programa que llegeixi diversos arbres binaris de naturals no buits i, per a cadascun, indiqui si és un arbre especular o no. Diem que un arbre és especular si el seu costat esquerre i el seu costat dret són l'un com el reflex de l'altre en un mirall vertical. L'entrada és un nombre enter que ens diu quants arbres hi ha a la seqüència, i una seqüència de descripcions d'arbres segons el format explicat a classe. La sortida és, per a cada arbre, escriure "ESPECULAR" o "NO ESPECULAR" segons convingui.

    Pista: Us anirà bé utilitzar com a funció auxiliar la solució del problema anterior. Fixeu-vos que a l'entrada la marca de final dels arbres és -1, no 0 com fem habitualment.

Entrada:	**Sortida**
3 **100 70 20 -1 30 -1 -1 -1 70 -1 20 30 -1 -1 -1** **100 200 -1 -1 300 -1 -1** **300 300 300 -1 -1 -1 300 300 -1 -1 -1**	**ESPECULAR** **NO ESPECULAR** **NO ESPECULAR**

50. Recordem que l'alçada d'un node en un arbre és la longitud (en nodes) del camí més llarg possible des del node fins a una fulla. L'alçada d'un arbre és l'alçada de l'arrel. Direm que un node en un arbre binari està ***equilibrat*** si (el valor absolut de) la diferència en l'alçada del fill esquerre i dret d'aquest node és com a molt 1. Un arbre binari està equilibrat si tots els seus nodes estan equilibrats. Feu una funció que ens digui si un arbre binari està equilibrat o no. Estem suposant que no tenim cap operació pública a la classe **BinTree** que ens retorni l'alçada. La funció ha de ser tan eficient com sigui possible (compteu quants cops es passa per un node a la vostra solució; es pot fer passant només un cop).

Implementació de Diccionaris i Conjunts

51. Considerem una taula de dispersió amb encadenament separat, amb `M = 10` posicions amb nombres enters com a claus, i funció de dispersió `h(x) = x mod M`.

    (a) Començant amb una taula buida, mostra com queda després d'inserir les següents claus (els valors associats els ignorem): 3441, 3412, 498, 1983, 4893, 3874, 3722, 3313, 4830, 2001, 3202, 365, 128181, 7812, 1299, 999 and 18267.

    (b) Mostra la taula resultant després de fer *rehashing* de la taula anterior en una taula amb `M = 20`.

52. Construeix una taula de dispersió amb encadenament separat amb les claus: 30, 20, 56, 75, 31, 19, 46, 88, 12, 25, de mida `M = 13` i funció de dispersió `h(x) = x mod M`.

    (a) Calcula el nombre *màxim* de comparacions que cal fer en una cerca amb èxit en aquesta taula.

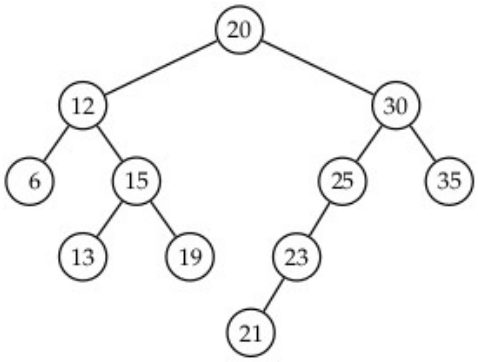
    (b) Calcula el nombre *mitjà* de comparacions que cal fer en una cerca amb èxit en aquesta taula. (suposem que cada un dels números apareix amb probabilitat 1/10).

53. Fes una funció que, donada una llista amb el recorregut en pre-ordre d'un BST, retorni el BST reconstruït:

```
def bst_a_partir_de_preordre(lst):
    """
    Pre: lst és el recorregut en pre-ordre d'un BST.
         Suposem que tots els elements d'lst són diferents
    Retorna una instància de BinTree
    """
```

54. Fusió de dos BST: La fusió de dos BSTs `a1` i `a2` és una *llista* que conté tots els elements d'`a1` i `a2`, ordenats. Implementar una funció **`def`** `bst_merge(a1,a2)` que retorna la combinació dels BST `a1` i `a2`. Quin és el cost de la funció? Intenta que sigui eficient.

55. Segment d'un BST: Implementeu una funció **`def`** `segment(a,x1,x2)` que, donat un BST `a` i dos elements `x1` i `x2` amb $x1 \leq x2$, retorna una llista amb tots els elements d'`a` que es troben entre `x1` i `x2`, en ordre *decreixent*. Per exemple, si `a` és el BST:



    la crida a `segment(a,18,26)` ha de retornar la llista `[25,23,21,20,19]`.

Complexitat Assimptòtica

56. Per a cada una de les següents funcions, doneu el seu ordre de magnitud de la manera més simple possible utilitzant la notació assimptòtica Θ. Per exemple, $3n^2 = \Theta(n^2)$.

    a) $23n^2 + 3n - 4$ b) 42 c) $2n$ d) $3n$

    e) 2^n f) 3^n g) $0.001 \cdot n \cdot \log(n) + 1000 \cdot n$

    h) $n^2 + n - \sqrt{n}$ i) $n! + 2n$ j) $n^n + n!$

57. Per a cada un d'aquests fragments de codi, analitzeu el seu cost en funció d'**n**

```
a)
 s = 0
 i = 1
 while i <= n:
     s += 1
     i *= 2
b)
 s = 0
 for i in range(n):
     for j in range(i):
         s += 1
c)
 s = 0
 for i in range(n):
     s += 1
 for j in range(n):
     s += 1
```

58. Escriviu la funció (trivial) que suma dues matrius (llista de llistes) de mida **n** × **n**. Expliqueu per quina raó diem que aquest algorisme, de cost $\Theta(n^2)$, és *lineal*. Argumenteu per quina raó qualsevol algorisme per sumar dues matrius requerirà $\Omega(n^2)$ passos.

59. Resol les següents recurrències:
    a) $T(n) = T(n-1) + \Theta(1)$ b) $T(n) = T(n-2) + \Theta(1)$
    c) $T(n) = T(n-1) + \Theta(n)$ d) $T(n) = 2T(n-1) + \Theta(1)$

60. Resol les següents recurrències:
    a) $T(n) = 2T(n/2) + \Theta(1)$ b) $T(n) = 2T(n/2) + \Theta(n)$
    c) $T(n) = 2T(n/2) + \Theta(n^2)$ d) $T(n) = 9T(n/3) + 3n + 2$
    e) $T(n) = T(9n/10) + \Theta(n)$

61. Resol la recurrència: $T(n) = T(\sqrt{n}) + 1$

62. Dissenyar i analitzar la complexitat d'un algorisme que utilitza una taula de dispersió per comprovar que tots els elements d'una llista són diferents (no cal codificar l'algorisme, una descripció abstracta ja està bé).