Pràctica obligatòria de Haskell. Point kd 2^n-Trees

1 Presentació

Volem representar un **conjunt** de punts en l'espai de k dimensions. Per això usarem una estructura anomenada kd 2^n-Trees, que és una variant dels kd-Trees que generalitza també els quadtrees i els octrees. Els kd 2^n-Trees són arbres generals de cerca on cada node conté un punt de k dimensions i una llista ordenada creixentment de les coordenades dels punts que tindrem en compte per distribuir la resta de punts entre els seus fills. Per exemple, si la llista en cada node conté sempre una única coordenada llavors tenim un kd Tree, si estem en l'espai 2D i conté sempre dues coordenades tindríem un quadtree i si estem en el 3D i totes les llistes tenen 3 coordenades tindríem un octree.

En el nostre cas podem tenir diferents mides de llista en cada node, però assumirem que els nombres que conté no són repetits, estan ordenats i són menors o iguals que la dimensió dels punts amb que estem treballant.

Com a exemple, suposem que estem en un espai 3D i considereu el punt $(3.1, -1.0, 2.5)$ i la llista de coordenades $[1, 3]$. Això vol dir que usarem la coordenada primera i la tercera, per repartir els punts en 4 quadrants centrats en el punt $(3.1, -1.0, 2.5)$, però només tenint en compte la primera i la tercera coordenada. En general si la llista té mida m, repartirem els punts en 2^m fills (arbres) segons les coordenades seleccionades, que estan numerats entre 0 i $2^m - 1$.

Així doncs, si el node conté el punt $(3.1, -1.0, 2.5)$ i la llista [1,3] llavors tindrà quatre fills i tindrem que

- el punt $(1.1, -1.5, 2.5)$ estarà en el fill 0
- el punt $(2.1, 3.5, 6.5)$ estarà en el fill 1
- el punt $(4.1, 2.0, 0.5)$ etarà en el fill 2
- el punt $(4.1, -2.0, 2.6)$ estarà en el fill 3

Per determinar el fill en que li toca farem el següent. Sigui p el punt del node i q el punt que volem afegir. Per a cada coordenada de la llista posem un 0 si el valor de q en aquesta coordenada és menor o igual que el de p i un 1 en cas contrari. El valor del nombre en binari obtingut aplicant-ho a totes les coordenades de la llista (amb el mateix ordre) ens diu el fill que li toca. En l'exemple, pel punt $(1.1, -1.5, 2.5)$ obtenim 00 que és 0, pel punt

$(2.1, 3.5, 6.5)$ obtenim 01 que és 1, pel punt $(4.1, 2.0, 0.5)$ obtenim 10, que és 2, i per $(4.1, -2.0, 2.6)$ obtenim 11 que és 3.

Per crear un kd 2^n-Tree ens donaran una llista de parells que contenen dues llistes. La primera són les coordenades del punt i la segona és la llista de coordenades que s'usarà per distribuir els següents punts entre els seus fills. Per exemple, la llista següent

$[([3.0, -1.0, 2.1], [1, 3]), ([3.5, 2.8, 3.1], [1, 2]), ([3.5, 0.0, 2.1], [3]), ([3.0, -1.7, 3.1], [1, 2, 3]),$
$([3.0, 5.1, 0.0], [2]), ([1.5, 8.0, 1.5], [1]), ([3.3, 2.8, 2.5], [3]), ([4.0, 5.1, 3.8], [2]),$
$([3.1, 3.8, 4.8], [1, 3]), ([1.8, 1.1, -2.0], [1, 2])]$

és representa amb un 3d 2^n-Tree que té com arrel un punt amb coordenades $3.0, -1.0, 2.1$, la llista $[1, 3]$ i quatre fills:

1. al primer fill (0) és un 3d 2^n-Tree amb arrel $3.0, 5.1, 0.0$ i la llista $[2]$ i només $1.8, 1.1, -2.0$ i la llista $[1, 2]$ al fill 0 i només $1.5, 8.0, 1.5$ i la llista $[1]$ al fill 1.
2. al segon fill (1) és un 3d 2^n-Tree amb només $3.0, -1.7, 3.1$ i $[1, 2, 3]$
3. al tercer fill (2) només conté $(3.5, 0.0, 2.1)$ i $[3]$.
4. al quart fill (3) és un 3d 2^n-Tree amb arrel $(3.5, 2.8, 3.1)$ i $[1, 2]$ i al fill 0 només $(3.3, 2.8, 2.5)$ i $[3]$, al fill 1 només $(3.1, 3.8, 4.8)$ i $[1, 3]$, res al fill 2 i al fill 3 només $(4.0, 5.1, 3.8)$ i $[2]$

2 Es demana

Feu el que es demana als següents apartats respectant els noms de les classes, els tipus i les funcions que s'indiquen. En el que resta, considerarem que els valors de les coordenades dels punts són del tipus `Double`.

1. Definiu en Haskell una nova classe de tipus anomenada `Point` de tipus p que representen punts i que tenen cinc funcions:
   (a) `sel` que donat un número que indica la coordenada i un element de tipus p retorna el valor de la coordenada que és un `Double`
   (b) `dim` que donat un element de tipus p retorna la seva dimensió
   (c) `child` que donats dos elements $e1$ i $e2$ de tipus p que representen punts i una llista de coordenades seleccionades retorna el número del fill de $e2$ que li toca a $e1$.
   (d) `dist` que donats dos elements $e1$ i $e2$ de tipus p ens retorna un `Double` que és la distància entre $e1$ i $e2$.
   (e) `list2Point` que rep una llista de `Double` i retorna un element de tipus p.

2. Com a exemple d'ús de la classe `Point` definiu el tipus `Point3d` i feu que sigui "instance" d'aquesta classe, cosa que us obligarà a definir les quatre funcions anteriors. Posteriorment haurem de fer que sigui de la classe `Eq` i de la classe `Show` (si us calen altres afegiu-les també).

3. Definiu en Haskell el nou tipus de dades genèric `Kd2nTree` seguint la descripció donada anteriorment i on el tipus del punt ha de ser genèric. Noteu que en algunes operacions ens caldrà que aquest tipus sigui de la classe `Point` o de la `Show`, etc.

   - definiu correctament la igualtat en el tipus `Kd2nTree` com a instància de la classe `Eq`. Fixeu-vos que la igualtat estructural no val, ja que dos conjunts són iguals si contenen els mateixos punts.
   - definiu correctament la funció de mostrar en el tipus `Kd2nTree` com a instància de la classe `Show`, de manera que el resultat sigui un String, que al fer `putStr` del `show` es mostri de la següent forma:

     ```
     (3.0,-1.0,2.1) [1,3]
      <0> (3.0,5.1,0.0) [2]
           <0> (1.8,1.1,-2.0) [1,2]
           <1> (1.5,8.0,1.5) [1]
      <1> (3.0,-1.7,3.1) [1,2,3]
      <2> (3.5,0.0,2.1) [3]
      <3> (3.5,2.8,3.1) [1,2]
           <0> (3.3,2.8,2.5) [3]
           <1> (3.1,3.8,4.8) [1,3]
           <3> (4.0,5.1,3.8) [2]
     ```

     és a dir, els fills sempre es mostren en el mateix ordre, però no s'han de mostrar els fills buits, i cal afegir els blancs i els salts de línia necessaris.

   Podeu definir constants al vostre programa per tal de no escriure cada vegada el mateix `Kd2nTree` amb que treballeu:

   ```
   exampleSet :: Kd2nTree Point3d
   exampleSet = ...
   ```

4. Feu una operació `insert` que donats (en aquest ordre) un `Kd2nTree` generic d'elements de la classe `Point`, un element d'aquest tipus (punt) i una llista d'enters (que representa les coordenades de selecció), retorna un `Kd2nTree` resultant d'inserir el punt al conjunt. Usant aquesta funció, feu una funció `build` que generi un `Kd2nTree` a partir d'una llista de parells que contenen un element de la classe point i una llista de coordenades. El `Kd2nTree` s'ha d'obtenir inserint els elements de la llista en l'ordre que apareixen (del primer al darrer). Finalment, feu una funció `buildIni` que

és com la `build` pero que rep una llista de parells `[([Double],[Int])]` com la de l'exemple anterior.

En les següents funcions ja no indicarem si el tipus dels elements que conté han de ser de la classe `Point` o no. Ho haureu de decidir vosaltres segons les operacions que us calguin quan les implementeu.

5. Feu una funció `get_all` que donat un `Kd2nTree` retorna la llista que conté tots els parells amb els punts i les llistes de coordenades del conjunt.

6. Feu una operació `remove` que donat un `Kd2nTree` i un punt (en aquest ordre), retorna un `Kd2nTree` resultant d'eliminar el punt del conjunt

7. Feu una operació `contains` que donat un `Kd2nTree` i un punt (en aquest ordre), ens diu si el punt pertany o no al conjunt.

8. Feu una operació `nearest` que donat un `Kd2nTree` (no buit) i un punt (en aquest ordre), ens diu quin és el punt més proper que pertany al conjunt.

9. Feu una operació `allinInterval` que donat un `Kd2nTree` i dos punt (en aquest ordre), ens retorna la llista ordenada amb tots els punts del conjunt que són més grans o iguals que el primer punt i més petits o iguals que el segon punt.

10. Feu una funció `kdmap` que donats (en aquest ordre) una funció de tipus `p -> q` i un `Kd2nTree` d'elements de tipus `p`, retorna un `Kd2nTree` de punts de tipus `q` resultant d'aplicar la funció a tots els punts mantenint l'estructura. Noteu que en general el resultat no té perquè seguir satisfent la propietat dels fills dels `Kd2nTree`, però hi ha força casos interessants on sí es manté.

    Utilitzant el `kdmap` feu una funció `translation` que aplica una translació a tots els punts d'un `Kd2nTree` (que serà el segon paràmetre). El primer paràmetre serà la translació i s'expressarà amb una llista de `Double`, que representen la translació per a cada coordenda. Per fer l'operació genèrica, heu de definir una nova operació `ptrans` a la classe `Point`.

    Utilitzant el `kdmap` feu una funció `scale` que aplica una escalat a tots els punts d'un `Kd2nTree` (que serà el segon paràmetre). L'escalat (que és el primer paràmetre) s'expressarà amb valor numèric. Igualment, per fer l'operació genèrica, heu de definir una nova operació `pscale` a la classe `Point`.