Estructures de dades i algorismes

Algorisme principal generador d'horaris

Descripció

L'algorisme està compost de 3 funcions principals. La primera (**generateSchedule**) prepara les dades per a generar l'horari. Per a fer-ho, comença assignant totes les possibles aules, hores i dies possibles per a cada Lecture (un grup pot tenir múltiples lecture), tenint en compte les restriccions unàries, de manera que eliminem qualsevol combinació que no cumpleixi les restriccions unàries. Anomenem a aquest procés **"primer filtrat"**, que estalviarà feina al forward checking. Durant aquest procés, anem contant el nombre d'assignacions possibles de cada lecture, i posteriorment introduirem cada Lecture amb el nombre d'assignacions en una PriorityQueue que anomenem **"heuristica"**, que ens ordenarà les lectures en ordre creixent d'assignacions. D'aquesta manera, intentarem posar primer les Lectures que tenen poques possiblitats,i així ens estalviem que quedin cap al final, on és molt probable que després de la inserció de la resta de lectures no tinguin opcions de ser inserides.

Un cop generades les possibles assignacions per a cada Lecture, les passem a la segona funció (**backjumping**). Aquesta va agafant les Lectures de la PriorityQueue "heurística" i les insereix en una de les seves possibles assignacions. Un cop fet això, es fa una crida a la tercera funció (**forwardChecking**), que borra per totes les Lectures les assignacions que consideràvem possibles però que ja no són compatibles amb la última inserció que hem fet. ForwardChecking elimina assignacions de manera que quan acaba, totes les assignacions compleixen totes les restriccions fins el moment. Si per alguna Lecture no queden assignacions possibles que compleixin les restriccions, forwardChecking retorna false. En tal cas, la funció backjumping prova una altre assignació perquè ja sabem que aquesta no ens porta enlloc. En el cas que que forwardChecking torni true (encara hi ha possibles assignacions per totes les lectures), fem una crida recursiva a backjumping per inserir la següent Lecture. Si no hi ha cap inserció possible per una Lecture, la crida de backjumping retorna false i tira enrere per provar una altra cosa. En cas que s'insereixin totes les Lectures, backjumping torna true, i sabem que s'ha generat un horari correctament.

En totes aquestes crides, per a emmagatzemar les possibilitats ho fem en un Map que té com a key la lecture i com a value un objecte de classe posAssig, que és un map que per cada dia té un map de hores que té un set de totes les aules possibles a cada hora.

Cal dir que quan controlador de domini fa la cridar a generarHorari, no li passarà totes les restriccions de manera incontrolada. Totes les restriccions negociables es poden activar i desactivar, de manera que el controlador de domini consulta quines estan activades i només li passarà com aquestes a generarHorari.

Altres estructures de dades

L'estructura de dades que guarda l'**horari** és un mapa de clau un String, que fa referència a una Aula, i de valor una matriu, que representa els 5 dies de la setmana i les 12 hores possibles, de manera que guardem l'horari de cada aula.

Per a guardar les Aules, Assignatures, Grups i Lectures al controlador de domini, utilitzem Maps que tenen com a key l'identificador de cada objecte, i el valor és el propi objecte. Les assignatures es relacionen amb grups i aquests amb lectures utilitzant Strings que els identifiquen. Així **evitem redundància** a memòria.

Ens basem en el mateix per a guardar les restriccions. Les unàries es guarden en un Map on la key és el Grup i el valor un Set de restriccions, d'aquesta manera podem accedir directament al es de cada grup en **temps constant** (encara no hem trobat cap escenari on el Hash code es repeteixi per algun objecte, de manera que **no tenim col·lisions** en cap dels Maps). Les nàries es guarden en un Set ja que s'apliquen per tots els grups.