Descripció de l'algorisme principal

Generar horari

Hem utilitzat un **Backtracking cronològic** amb **Forward Checking**.

Quan es vol generar un horari s'ha de cridar la funció `schedule.generateSchedule(sessions);` on **schedule** és un objecte de la classe Schedule i ***sessions*** és un Stack de sessions que conté totes les sessions que es volen col·locar a l'horari.

En llenguatge natural, l'algorisme realitza el següent:

L'algorisme el que fa és assignar una hora, dia i aula (ClassroomDayHour) a cada sessió, posar aquesta sessió a l'atribut **schedule** de la classe i eliminar-lo del stack.

Per assignar una hora, dia i aula cada sessió té un conjunt de ClassroomDayHour amb totes les combinacions d'hora, dia i aula que compleixen totes les restriccions de la sessió. Aquest Set<ClassroomDayHour\> s'anomena **allowedClassroomDayHour**. Els ClassroomDayHour d'aquest conjunt estan ordenats per una puntuació, que com més positiva és, millor compleix les restriccions i si és negativa, vol dir que no les compleix.

Per actualitzar **allowedClassroomDayHour** i comprovar les restriccions es crida `s.removeUnsatisfactoryCDH(this.schedule);` , on **s** és una sessió i **this.schedule** una llista de sessions. Aquesta funció retorna un conjunt de ClassroomDayHour, que són els que s'han eliminat de **allowedClassroomDayHour**.

L'algorisme simplement recorre **allowedClassroomDayHour**, assigna el ClassroomDayHour a la sessió i fa el mateix amb la següent sessió (fent una crida recursiva a `algorithm(sessions)`), fins que pila està buida, que vol dir que totes les sessions han estat assignades, per tant retorna `true` indicant que s'ha generat l'horari correctament.

Si en fer la crida recursiva a `algorithm(sessions)` rebem `true` retornem també `true` indicant que s'ha generat l'horari correctament. En canvi, si rebem `false` vol dir que assignant aquell ClassroomDayHour no es pot arribar a un horari que no violi cap restricció, per tant el que es fa és: treure aquella sessió de l'horari, desassignar-li l'hora, dia i aula que havíem assignat i provar el següent ClassroomDayHour.

Si en una sessió ja hem provat tots els **allowedClassroomDayHour** o no té res a **allowedClassroomDayHour**, significa que, en l'horari actual, no pot ser assignada a cap hora, dia i aula sense violar cap restricció. En aquest cas es tornen a posar a **allowedClassroomDayHour** els ClassroomDayHour eliminats en cridar `s.removeUnsatisfactoryCDH(this.schedule);` , es torna a posar la sessió a la pila i es retorna `false` indicant que no es pot continuar.

Fent aquest procediment **this.schedule** acabarà tenint les sessions amb hora, dia i aula assignats, o buit en cas de no poder-se generar l'horari.

Codi que genera l'horari

Aquesta és la part del codi que genera l'horari.

```
public class Schedule{
  private String name;
  private List<Session> schedule = new ArrayList<>();
```

```
  public boolean generateSchedule(Stack<Session> sessions){
    schedule.clear();
    return algorithm(sessions);
  }

  private boolean algorithm(Stack<Session> sessions){
    if(sessions.empty()) return true;
    else{
      Session s = sessions.pop();

      Set<ClassroomDayHour> removedCDH = s.removeUnsatisfactoryCDH(this.schedule);
      Set<ClassroomDayHour> allowedClassroomDayHour = s.getAllowedClassroomDayHour();

      for(ClassroomDayHour cdh : allowedClassroomDayHour) {
        s.setClassroomDayHour(cdh);
        this.schedule.add(s);
        if (algorithm(sessions) == true) return true;
        else{
          this.schedule.remove(s);
          s.setClassroomDayHour(null);
        }
      }

      s.addClassroomDayHour(removedCDH);
      sessions.push(s);
      return false;
    }
  }

    // ... more methods ... //
}
```

Comprovar restriccions

Quan es crida a `session.removeUnsatisfactoryCDH(sessions);` , on **session** és una sessió i **s** una llista de sessions, es crida a `checkAllBinaryRestrictions(s)` i `checkAllGlobalRestrictions(s)` , i es retorna el conjunt de la unió del que han retornat les dues funcions. Cada funció fa el mateix però per a una classe diferent, concretament per cada restricció **r** crida `checkBinaryRestriction(r, s)` o `checkGlobalRestriction(r, s)` i es retorna l'unió dels conjunts que retorna cada funció.

El que fan `checkBinaryRestriction(r, s)` i `checkGlobalRestriction(r, s)` és: per cada **cdh** ClassroomDayHour de **allowedClassroomDayHour** cridar a `r.check(s, this, cdh.getHour(), cdh.getDay(), cdh.getClassroom());` .

La funció `check()` retorna una puntuació, positiva si: la sessió **this** es pot realitzar a l'hora **cdh.getHour()**, dia **cdh.getDay()** i aula **cdh.getClassroom()** en un horari amb les sessions **s** sense violar cap restricció, i negativa altrement.

Si aquesta **puntuació** és mes petita que **restThreshold** (per defecte 0) aquell ClassroomDayHour es borrarà de **allowedClassroomDayHour**.

Les restriccions unaries es comproven al afegirla, ja que no varien segons les altres sessions de l'horari.

Codi que comprova les restriccions

```
public class Session {
  private ClassType classType;
  private int number;
  private String group;

  private List<RestrictionUnary>  restsU = new ArrayList<>();
  private List<RestrictionBinary> restsB = new ArrayList<>();
  private List<RestrictionGlobal> restsG = new ArrayList<>();

  private Classroom classroom;
  private int weekDay;
  private int hour;

  private int restThreshold = 0;

  public Set<ClassroomDayHour> removeUnsatisfactoryCDH(List<Session> s) {
    Set<ClassroomDayHour> removedCDH = new HashSet<>();
    removedCDH.addAll(checkAllBinaryRestrictions(s));
    removedCDH.addAll(checkAllGlobalRestrictions(s));
    return removedCDH;
  }

  /* checkAllGlobalRestrictions() and checkAllBinaryRestrictions() do the same*/
  private Set<ClassroomDayHour> checkAllBinaryRestrictions(List<Session> s) {
    Set<ClassroomDayHour> removedCDH = new HashSet<>();
    for (RestrictionBinary r : restsB) {
      removedCDH.addAll(checkBinaryRestriction(r, s));
    }
    return removedCDH;
  }

  /* checkGlobalRestriction() and checkBinaryRestriction() do the same*/
  private Set<ClassroomDayHour> checkBinaryRestriction(RestrictionBinary r, List<Session> s) {
    Set<ClassroomDayHour> removedCDH = new HashSet<>();
    for (ClassroomDayHour cdh : allowedClassroomDayHour) {
      int points = r.check(s, this, cdh.getHour(), cdh.getDay(), cdh.getClassroom());
      cdh.addPuntuation(points);
      if (points < restThreshold)  removedCDH.add(cdh);
    }
    allowedClassroomDayHour.removeAll(removedCDH);
    return removedCDH;
  }

  public void addRestrictionUnary(RestrictionUnary r) {
    this.restsU.add(r);
    checkUnaryRestriction(r);
  }
```

```
  public void setClassroomDayHour(ClassroomDayHour chd){
    if (chd == null) {
      this.classroom = null;
      this.weekDay   = 0;
      this.hour      = 0;
    }else{
      this.classroom = chd.getClassroom();
      this.weekDay   = chd.getDay();
      this.hour      = chd.getHour();
    }
  }

    // ... more methods ... //
}
```

Aquesta és la classe auxiliar ClassroomDayHour:

```
public class ClassroomDayHour implements Comparable<ClassroomDayHour>{
  private int day;
  private int hour;
  private Classroom classroom;
  private int puntuation = 0;

  public int compareTo(ClassroomDayHour o){ // to be ordered in a Set<ClassroomDayHour>
    return(o.puntuation - puntuation);
  }

  // ... more methods ... //
}
```