# CS170 Project Part II - Ahmed Ali, Tony Zhang

We used the SAT4J library in this project. It is “the boolean satisfaction and optimisation library in Java.” There is a jar file in the folder and it should work fine if the folder is important into IntelliJ. We decided to use this library because our approach was to create 3sat clauses using the list of wizards as well as the constraints, this allowed us to then use SAT4J  to solve them.

**How to use SAT4J(Download & Installation):**

- First download SAT4j from http://www.sat4j.org/ ,and it will come as a zip file.
- Create a project in IntelliJ.
- Unzip the SAT4j and place the resultant folder inside the intelliJ project you just created.
- In intelliJ add the SAT4j folder as part of your library. You can do that by open file then go to project structure. Then add libraries and hit (+) sign to add the SAT4j.
- Now import the solver.java file into the project.
- To run a file, you need to have the file inside src folder in the intelliJ project.
- Every time you want to process folder just do the pervious step and change the name of the folder in the main function manually.

**The high level idea of our approach is as follows:**

- Each time we read an input file, we use the readingFile method which convert all the constraints in the input file into an Arraylist of String[ ] of constraints.

- Next, we use the randomOrdering routine to generate a random ordering of the wizards and also assigns each wizard to a unique integer ID.

- We then use the bigMap method to force a particular ordering for the wizards by building inequality statements such as (wizard A < wizard B).

- We then map these statements to a unique integers which the Sat4j uses to solve the problem. The reason we do this is because the Sat4j library takes in statements of integers such as “[2 9 3] [3 4]” where each integer within one of these clauses maps to a condition such as “wizard 3 < wizard 6”.

- We use generateClausesofTwo to generate SAT clauses of two numbers such as “[7 2]”, and we use generateClausesofThree to generate SAT clauses of three numbers such as “[7 9 4]”. The reason we do this is because the SAT4j Solver.

- If the given clauses is deemed solvable by the solver, we can then process the output by first calling the processing method on the model, which essentially just does the post processing after the Sat4j solver solves all the statements(sat clauses) and gives the orderings. This is done by building a graph of all the wizards in the form of an adjacency list that we built with a hashmap. Each key in the hashmap is a wizard, and the list attached to that key is all the wizards that are older than itself. In other words, processing generates an adjacency list graph that denotes the dependencies, or “smaller than” relationships between all the wizards. For example, the key that has an empty list as the item would be they key that denotes the oldest wizard, and the key that does not appear in the adjacency lists of any other key is the youngest wizard.

- This map/graph is then fed into the figuringOrdering method which uses the dependencies from the graph to generate the appropriate ordering out by sorting each key from the graph/map by the size of its adjacency list.

Note:
Our code could solve up to 180 optimally, but not 200 as a result we made a random ordering for the file from 200-400 and that why we use WriteRandomOrdering to the write those file.
Code:

```
import java.util.*;
import java.io.*;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.sat4j.core.*;
import org.sat4j.minisat.*;
import org.sat4j.specs.*;


/**
* Created by ahmedali89 on 11/22/17.
*/
public class Solver {

    public static HashMap<Integer,String> variables1 = new HashMap<>();
    public static HashMap<String,Integer> variables2 = new HashMap<>();
    public static HashMap<String,Integer> wizardsWorld1 = new HashMap<>();
    public static HashMap<Integer,String> wizardsWorld2 = new HashMap<>();
    public static ArrayList<String> wizards = new ArrayList<>();
    public static ArrayList<String[]> constraints = new ArrayList<>();
    public static ArrayList<int[]> clauses = new ArrayList<>();
    public static String filename= null;
    public static ArrayList<Integer> ordering = new ArrayList<>();
    public int num_wiz;

    public void readingFile(String filename) {

    int counter = 0;
    BufferedReader reader = null;
    try {
        FileInputStream file = new FileInputStream(filename);
        Scanner s = new Scanner(file);
        reader =  new BufferedReader(new InputStreamReader(file));
        String text = null;
        String num = s.nextLine();
        String num_const = s.nextLine();
        System.out.println(num_const);
        while (s.hasNext()) {
            String[] constraint = s.nextLine().split("\\s+");
            constraints.add(constraint);
            counter++;
        }
```

```
            System.out.println(counter);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
            }
        }
    }

    public static void print(ArrayList<String[]> constraints) {

        for (String[] cons : constraints) {
            for (String wizard : cons) {
                System.out.print(wizard + " ");
            }
            System.out.println();
        }
    }

    public void randomOrdering() {
        int count = 1;
        Set<String> uniqueList = new HashSet<>();
        ArrayList<String> random = new ArrayList<>();
        for (String[] cons : constraints) {
            for (String wizard : cons) {
                uniqueList.add(wizard);
            }
        }
        wizards.addAll(uniqueList);
        for(String wizard : wizards) {
            wizardsWorld1.put(wizard,count);
            wizardsWorld2.put(count,wizard);
            count++;
        }
    }

    public static void bigMap() {
            int count = 1;
            String condition = null;
            for (int i = 1; i <= wizards.size(); i++) {
                for (int j = 1; j <= wizards.size(); j++) {
                    if (i < j) {
                        condition = i + " < " + j;
                        variables1.put(count, condition);
                        variables2.put(condition, count);
                        count++;
```

```
            }
        }
    }
}

private int getValue(int wizardIDx, int wizardIDy) {
    if( wizardIDy < wizardIDx) {
        return (-1 * variables2.get(wizardIDy + " < " + wizardIDx));
    } else {
        return variables2.get(wizardIDx + " < " + wizardIDy);
    }
}

private  void generateClausesOfTwo() {
    int [] clause;
    int num;
    for (String[] constraint : constraints) {

        int wizardID1 = wizardsWorld1.get(constraint[0]);
        int wizardID2 = wizardsWorld1.get(constraint[1]);
        int wizardID3 = wizardsWorld1.get(constraint[2]);

        int val1 = getValue(wizardID1,wizardID3);
        int val2 = getValue(wizardID2,wizardID3);

        clause =new int[2];
        clause[0] = val1;
        clause[1] = -val2;
        clauses.add(clause);

        clause = new int[2];
        clause[0] = val2;
        clause[1] = -val1;

        clauses.add(clause);
    }
}

private void generateClausesOfThree() {
    int[] clause;

    for (int i = 1; i <= wizardsWorld1.size(); i++) {
        for (int j = 1; j <= wizardsWorld1.size(); j++) {
            for (int k = 1; k <= wizardsWorld1.size(); k++) {
                clause = new int[3];
                if(i != j && j != k && i != k) {
                    clause[0] = -1 * getValue(i,j);
                    clause[1] = -1 * getValue(k,i);
                    clause[2] = getValue(k,j);
                    clauses.add(clause);
                }
            }
```

```
        }
      }
  }

  //post Processing
  public HashMap<String, HashSet<String>> processing(int[] model) {
     HashMap<String, HashSet<String>> dictonary = new HashMap<>();
        for(int x : model) {
           if(x < 0) {
              String value = variables1.get(-1 * x);
              String[] wizards = value.split(" < ");
              String wiz = wizardsWorld2.get(Integer.parseInt(wizards[1]));
              if (!dictonary.containsKey(wiz)) {
                 dictonary.put(wiz, new HashSet<>());
                 dictonary.get(wiz).add(wizards[0]);
              } else {
                 dictonary.get(wiz).add(wizards[0]);
              }
           } else {
              String value = variables1.get(x);
              String[] wizards = value.split(" < ");
              String wiz = wizardsWorld2.get(Integer.parseInt(wizards[0]));
              if (!dictonary.containsKey(wiz)) {
                 dictonary.put(wiz, new HashSet<>());
                 dictonary.get(wiz);
              } else {
                 dictonary.get(wiz).add(wizards[1]);
              }
           }
        }
        for (String w : wizards) {
           if(!dictonary.containsKey(w)) {
              dictonary.put(w, new HashSet<>());
           }
        }
     return dictonary;
  }

  public List<Entry<String,Integer>> figuringOrdering(HashMap<String, HashSet<String>>
dictonary) {
        HashMap<String,Integer> lst = new HashMap<>();
        for (Map.Entry<String, HashSet<String>> entry : dictonary.entrySet()) {
           lst.put(entry.getKey(),entry.getValue().size());
        }
        List<Entry<String,Integer>> entries = new ArrayList<>(lst.entrySet());
        Collections.sort(entries, new Comparator<Entry<String, Integer>>() {
           @Override
           public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {
              return (o2.getValue()).compareTo( o1.getValue());
           }
        });
```

```
        return entries;
    }

    public void writeToFile(List<Entry<String,Integer>> entries, String filename) {
        StringBuilder order = new StringBuilder();
        Writer writer = null;
        try {
            writer = new BufferedWriter(new OutputStreamWriter(
                    new FileOutputStream(filename), "utf-8"));
            for(Map.Entry<String, Integer> entry: entries){
                order.append(entry.getKey() + " ");
            }
            writer.write(order.toString());
        } catch (IOException ex) {
            // report
        } finally {
            try {writer.close();} catch (Exception ex) {/*ignore*/}
        }
    }

    public void writeRandomOrdering() {
        StringBuilder order = new StringBuilder();
        String [] fileArray = filename.split("\\.");
        Pattern pattern = Pattern.compile("\\_\\d+");
        Matcher matcher = pattern.matcher(fileArray[0]);
        if (matcher.find()) {
            Writer writer = null;
            String str = matcher.group();
            filename = filename.replaceAll(filename, "staff" + str + ".out");
            try {
                writer = new BufferedWriter(new OutputStreamWriter(
                        new FileOutputStream(filename), "utf-8"));
                for (String wizard : wizards) {
                    order.append(wizard + " ");
                }
                writer.write(order.toString());
            } catch (IOException ex) {
                // report
            } finally {
                try {
                    writer.close();
                } catch (Exception ex) {/*ignore*/}
            }
        }
    }

    public static void main(String[] args) {

        Solver S = new Solver();
        String fileArray[] = null, str = null;
        filename = "src/staff_200.in";
```

```
    S.readingFile(filename);
    S.randomOrdering();
//       if (filename.contains("staff")) {
//           S.writeRandomOrdering();
//       }
//
        bigMap();

        S.generateClausesOfTwo();

        S.generateClausesOfThree();

        ISolver solver = SolverFactory.newDefault();
        try {
           for (int[] clause : clauses) {
              solver.addClause(new VecInt(clause));
           }
           IProblem problem = solver;
           if (problem.isSatisfiable()) {
              List<Entry<String, Integer>> entries =
S.figuringOrdering(S.processing(problem.model()));
              fileArray = filename.split("\\.");
              Pattern pattern = Pattern.compile("\\_\\d+");
              Matcher matcher = pattern.matcher(fileArray[0]);
              if (matcher.find()) {
                 str = matcher.group();
                 filename = filename.replaceAll(filename, "staff" + str + ".out");
                 S.writeToFile(entries, filename);
              } else {
                 System.out.println("Something Wrong With name of File!!!");
              }
           } else {
              System.out.print("problem is not satisfiable");
           }
        } catch (ContradictionException e) {
           System.out.println("Unsatisfiable (trivial)!");
        } catch (TimeoutException e) {
           System.out.println("time is out!");
        }
    }
}
```