实验 1：SAT 求解器

1 环境配置

1. 请在本课程提供的虚拟机镜像中完成此实验以及后续实验，你可以按照 canvas 上提供的教程下载虚拟机镜像并将其导入至你的 VMware Workstation 中。
2. 进入虚拟机后，请从 canvas 上下载 *lab1.zip* 放至桌面并解压，然后进入 lab1 文件夹。
3. 在 lab1 文件夹中打开命令行，并在命令行中输入 `./install_minisat.sh` 来安装所有依赖，此脚本会帮你安装实验 1 所需要的所有依赖以及工具。

2 实验内容

通过本实验我们希望学生能实际使用 SAT 求解器解决一些简单的问题，我们使用的是 **MiniSat**，它是一个开源的 SAT 求解器，你可以在网络上找到它的源代码。（但这并不重要，我们只希望你能通过 SAT 求解器来解决实际问题，若你对其内部实现非常感兴趣可以尝试去阅读其源代码）。

2.1 MiniSat

我们将为你简单地介绍 **MiniSat** 的使用方法，在此之前，请你仔细阅读 *minisat.cpp* 的内容，这个实例程序求解了命题 $P = (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C) \wedge (A \vee \neg B \vee C)$ 通过注释简单理解每一行代码，这将有助于你理解 **MiniSat** 的使用方法。

首先，**MiniSat** 使用析取范式（CNF）作为其输入，要使用 **MiniSat** 求解命题逻辑表达式，首先我们需要初始化一个 Solver 对象，即 *minisat.cpp* 中的

```
    // Create a solver
    Solver solver;
```

Listing 1: 初始化 Solver 对象

然后我们需要定义一些命题变量，这里我们使用 **MiniSat** 提供的 `newVar` 函数，比如在 *minisat.cpp* 中，我们定义了文字 A, B, C：

```
    // Create variables
    auto A = solver.newVar();
    auto B = solver.newVar();
    auto C = solver.newVar();
```

Listing 2: 定义文字

接下来我们需要向 solver 中添加子句，这里我们使用 **MiniSat** 提供的 `mkLit` 和 `addClause` 函数，`mkLit` 可以将命题变量转化成子句需要的格式，`addClause` 可以向 solver 中添加子句。比如在 *minisat.cpp* 中，你可以看到我们添加了三次子句：

```
    // Add the clauses
    // (~A v ~B v C)
    solver.addClause(~mkLit(A), ~mkLit(B), mkLit(C));
    // (~A v ~B v ~C)
    solver.addClause(~mkLit(A), ~mkLit(B), ~mkLit(C));
    // (A v ~B v C)
    solver.addClause(mkLit(A), ~mkLit(B), mkLit(C));
```

Listing 3: 添加子句

接下来我们正式调用`solve` 函数进行求解，函数的返回值是一个 bool 类型，标志这此命题是否存在一组解释使其为真：

```
    // Solve the problem
    auto sat = solver.solve();
```

Listing 4: 求解

如果返回值为真，我们可以通过`modelValue` 函数获取一种解释，若返回值为假，我们直接输出 “UNSAT”：

```
    // Check solution and retrieve model if found
    if (sat)
    {
        std::clog << "SAT\n"
                  << "Model found:\n";
        std::clog << "A := " << (solver.modelValue(A) == l_True) << '\n';
        std::clog << "B := " << (solver.modelValue(B) == l_True) << '\n';
        std::clog << "C := " << (solver.modelValue(C) == l_True) << '\n';
        return 0;
    }
    else
    {
        std::clog << "UNSAT\n";
        return 1;
    }
```

Listing 5: 输出结果

（提示，在你要完成的实验中，你不能有任何使用`printf`，`cout`或者`clog`的输出，你需要做的是将结果通过函数参数的方式传出，这样我们才能够对你的程序进行测试。）

最后，你可以在 lab1 目录下打开命令行，输入`make example`来运行这个例子并看到输出结果如下：

```
SAT
Model found:
A := 0
B := 0
C := 0
```

2.2 问题描述

我们希望你能用 **MiniSat** 来求解一个实际问题，问题描述如下：

河中有 n 个石头，Tom 和 Jerry 想要踩着石头过河，这些石头有两种状态，要么浮在水面上，要么沉在水底。河边有 m 个开关，控制石头的沉浮。每个开关控制 1 或 2 个石头，每块石头被 1 或 2 个开关控制。每次打开或关闭一个开关，这个开关控制的石头都会改变状态，从水下浮到水上或者从水上沉到水下。

一开始所有开关都是关闭的，石头有的在水上有的在水下。请利用 **MiniSat** 来判断如何控制这些开关使得所有石头同时浮在水上。

2.3 代码框架

2.3.1 输入

输入数据在 *test.txt* 里面。第一行的整数是测试用例的数量，后面跟着所有测试用例，不同测试用例之间隔一行。

每个测试用例第一行是两个数字 m 和 n。m 是开关的数量，n 是石头的数量。下一行是 n 个数字，对应 n 个石头的初始状态，0 表示这个石头在水下，1 表示石头在水上。接下

来 m 行是 m 个开关的信息。每一行的第一个数字表示这个开关控制了几个石头，后面跟着石头的编号，石头的编号从 1 开始。

2.3.2 输出

框架代码会把结果输出到 *answer.txt* 里面，每个测试用例的结果占一行。

每一行包含几个 0 或 1，0 表示对应的开关应该关闭，1 表示对应的开关应该打开，这样每块石头都会浮在水上。如果没办法过河，那么这一行会只有一个 UNSAT。

2.3.3 举例说明

若 test.txt 的内容如下：

1

2 2

1 0

1 1

1 2

第一行表示有 1 个测试用例。第二行表示有 2 个开关和 2 个石头。第 3 行表示一开始，1 号石头在水上，2 号石头在水下。第 4 行表示第一个开关控制 1 块石头，是 1 号石头，第二个开关控制 1 块石头，是 2 号石头。

答案将输出到 *answer.txt* 里，内容如下。

0 1

这表示应该关闭第一个开关并打开第二个开关。

2.3.4 实现要求

main.cpp 已经完成了文件读取和答案输出功能，你需要完成 *lab1.cpp* 中的函数`lab1`，请不要修改除了函数`lab1`之外的代码。

函数`lab1`包括 5 个参数。

- n 是石头的数量。
- m 是开关的数量。
- 数组 states 是 n 个石头的初始状态。数组长度可能大于 n，可以忽略多出来的部分。
- Button 是一个二维数组，记录开关的控制信息，其中每个元素都是石头的编号。例如，button[3][0] 和 button[3][1] 表示被第 4 个开关控制的石头编号。石头的编号从 1 开始。如果 button[3][0] = 1 并且 button[3][1] = 0，那么这个开关只控制 1 号石头。
- 数组 answer 里面是问题的答案。例如，answer[2] = true 表示第 3 个开关应该打开。

- `lab1`函数的返回值类型是 bool，如果返回 true，表示有办法让所有石头同时浮在水上，如果返回 false，表示 Tom 和 Jerry 无法过河了。

假设这些参数的值都是合法的。

完成后请在 lab1 文件夹下打开命令行，输入`make lab1`就可以编译运行代码，运行结束后请到 *answer.txt* 中查看答案。

请注意，我们不提供测试用例的正确答案，且有些测试用例的正确答案可能不唯一。

3 作业提交

在 lab1 文件夹下打开命令行，输入`make handin`，会生成一个 *lab1.zip* 压缩包，里面保存了你的 *lab.cpp* 文件，请在截止日期前将其提交至 canvas。

请注意，若你违反了作业相关要求，将酌情扣分。