离散优化建模：作业十
寿宴准备

1 问题描述

为了庆祝女娲万岁诞辰，巨鳌也准备了丰盛的宴席，其中包含了女娲最喜欢的菜式。龙宫的主厨需要安排尽快完成寿宴准备，以保证在女娲太累而需要回床上休息之前完成。

主厨可以使用一组厨师，而每一道菜可以分成一些按照一定的顺序完成的不同步骤。每一步需要其中一名厨师完成，而这些厨师一次只能完成一件事情。

2 数据格式说明

寿宴准备的输入包含在一个名为 `data/banquet`p`.dzn` 的文件里面，其中 p 是问题的序号。数据文件定义了以下数量:

- $COOK$ 是一个表示可以使用的厨师的枚举类型，
- $DISH$ 是一个表示需要完成的菜式的枚举类型，
- no_tasks 是所有菜式需要的步骤总数，
- $steps$ 是一个步骤集合的数组，表示每道菜式需要的步骤（注意到一些步骤可能是某些菜共有的），
- $time$ 是一个二维数组表示对于每一个厨师来说，每一个步骤所需要的时间。如果是 0 则表示厨师不能完成这个步骤。

问题的目标是为每个步骤决定开始时间以及完成它的厨师。

数据声明和主要的决策变量如下:

```
enum COOK; % the set of cooks available
enum DISH; % the set of all dishes to prepare
int: no_tasks; % total number of tasks
set of int: TASK = 1..no_tasks;
array[DISH] of set of TASK: steps; % steps to making a dish
array[TASK, COOK] of int: time; % amount of time to complete task by cook

int: maxt = sum(array1d(time));
```

```
set of int: TIME = 0..maxt;
array[TASK] of var TIME:    s;  % start time for task
array[TASK] of var COOK:    c;  % cook for task
var TIME: obj;
```

每一个阶段的输出的形式如下：

```
s = [每一个步骤开始的时间];
c = [执行每一个步骤指定的厨师];
obj = 完成所有菜的时间;
```

目标是使完成时间最小化 (obj)。注意到每一道菜的步骤需要按照步骤序号的顺序来完成。

比如说对于给定的数据文件

```
COOK = { BOB, CAROL, TED, ALICE };
DISH = { MAPOTOFU, FRIEDEEL, STICKYRICE };
no_tasks = 9;
steps = [1..3, {2,4,5,6}, 7..9];
time = [| 17, 25,  0,  0
        |  0, 30,  0, 40
        |  0,  0,160,150
        | 30,  0, 50,  0
        |  0, 66,  0, 55
        |  0,  0, 65, 78
        | 56, 62,  0,  0
        |  0, 70, 80,  0
        |  0,  0,100, 90 |];
```

这里面有 3 道菜和 9 个步骤，其中一个是共有的。

一个最优解如下

```
s = [0, 17, 47, 73, 103, 207, 17, 73, 158];
c = [BOB, CAROL, TED, BOB, ALICE, TED, BOB, CAROL, ALICE];
obj = 272;
```

注意到步骤 2 要在步骤 3 和 4 之前完成。注意到没有任何厨师会在同一时间做两样事情：BOB 在时间 0..17 时做步骤 1，时间 17..73 做步骤 7，时间 73..103 做步骤 4；CAROL 在 17..30 的时候做步骤 2，73..143 的时候做步骤 8；TED 在 47..207 的时候做步骤 3，207..272 的时候做步骤 6；而 ALICE 在 103..158 的时候做步骤 5，在 158..248 的时候做步骤 9。

这次作业要求你为这个问题建立一个高效的模型，并找出一个有效的搜索策略可以在 30 秒以内找到一个好的时间表。

注意到这是一个调度问题，你可能需要利用关于调度的全局约束来使你的模型更加高效。同时你在搜索中可能会用到很多其他重要的概念，比如每一个步骤的结束时间，还有每一个步骤的长度。你可能需要在你的搜索中尝试使用一个好的重启策略。

除了在 IDE 中设置 "Solver flags" 或是使用于命令行的参数，MiniZinc 现在也支持以下注解来启动重启搜索：

- `restart_luby(int: scale)`
- `restart_geometric(float: base, int: scale)`
- `restart_constant(int: scale)`
- `restart_linear(int: scale)`

谨记引用头文件"gecode.mzn"。我们希望你可以在模型提交中使用这些注解。请确保你安装了最新的 MiniZinc 版本。

3 Instructions

你可以编辑 `banquet.mzn` 模型文件来解决上述优化问题。你实现的模型 `banquet.mzn` 可以用提供的数据文件进行测试。在 MiniZinc IDE 中，你可以通过点击 *Run* 按钮在本地测试和运行。或者在命令行中输入

```
mzn-gecode ./banquet.mzn ./data/<inputFileName>
```

进行本地测试和运行。两种情况下，你的模型都是用 MiniZinc 进行编译然后用 GECODE 求解器求解。

参考资料 你可以在 `data` 文件夹下找到讲义中的几个问题实例（的数据文件）。

提交作业 这次的作业包含有 4 个答案提交部分和 3 个模型提交部分。对于答案提交部分，我们将会提交求解器求解你的模型所得到的最好／最后的答案，然后检查它的正确性和质量。对于模型提交部分，我们将会提交你的模型文件 (.mzn) 然后用一些未公开的数据文件来做进一步测试。

在 MiniZinc IDE，点击 *Submit to Coursera* 图标可以用于提交作业。若采用命令行方式，`submit.py` 可以用于提交作业。无论采用那种方法，你都需要根据本指引中的要求完成作业各部分

的 MiniZinc 模型。你可以多次提交，最终作业分数是你的最高的一次。[1]作业的打分过程可能需要几分钟，请耐心等候。你可以在课程网站上的**编程作业** 版块查看你的作业提交状况。

4 软件要求

为了完成作业，你需要安装MiniZinc 2.1.x和GECODE 5.0.x 求解器。这些软件都会包含在MiniZinc IDE 2.1.X (`http://www.minizinc.org`)的集成版本中。如果你需要通过命令行提交作业，你需要安装Python 3.5.x。

[1]问题解的提交并没有次数限制。但是，**模型提交部分**只能提交有限次。