All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates, but you may not look at their code. If in doubt, ask the instructor.

Acknowledge all sources you found useful.

Your code should compute the correct results.

Partial credit is available, so attempt all exercises.

Submit your code and answers in a single archive file (ZIP/TGZ/TBZ2, etc.) that contains your name in the filename (e.g. `JRandom_A1.zip`). Your answers should be a PDF file in the archive.

The goal of this assignment is to introduce you to some challenges in scheduling parallel tasks. Your goal is to produce an offline scheduler that takes a dependence graph and produces a schedule mapping each task in the dependence graph to a processor.

The dependence graphs are individual files and have the following format:

```
nodes edges
taskid1 time ndeps dep1 dep2 dep3 .... depN
taskid2 time ndeps dep1 dep2 dep3 .... depN
```

Here, `nodes` is the number of tasks, and `edges` is the number of dependences (edges) in the dependence graph. Each line after this header introduces a task, beginning with its `taskid`, the time it takes to execute (`time`), and the number of tasks that depend on it (`ndeps`). The task IDs of each dependent task (total `ndeps`) then follows on the same line. The dependence graph will not contain cycles.

Your scheduler will be invoked as follows:

```
offsched dep.gr out.sched Nproc
```

where `dep.gr` is the input dependence graph, `out.sched` is the output schedule file to be written in the prescribed format (see below), and `Nproc` is the number of processors you're scheduling for. The special value of 0 for `Nproc` indicates infinite processors.

The format of the output schedule `out.sched` should follow the format below:

```
taskid proc starttime
...
```

Each line of the schedule file should contain the task ID (from the dependence graph), the processor it is scheduled on (between 0 and `Nproc` - 1 for `Nproc` > 0), and the time the task should begin execution (`starttime`). Time in the schedule starts at 0. When scheduling for infinite processors, `proc` must be a non-negative number.

Your schedule should be valid:

1. All tasks should be executed.
2. A processor should only execute one task at a time.
3. A dependent task should only begin after all tasks it is dependent on have finished. I.e., dependents of a task that starts at time t and takes s time-units to run can only start at or after $t + s$.
4. You should not exceed the number of processors specified if using a finite number of processors.

You can check the validity of your schedule by using the supplied `des.py` file.

```
des.py -n Nproc dep.gr out.sched
```

This tool will throw assertion failures if your schedule violates the validity constraints.

Exercise 1

(Infinite processor scheduling) Schedule the supplied input graphs on infinite processors.

1. The *critical path* (or *span*) of a dependence graph is the path that takes the longest time to execute on a machine with infinite processors. For each input graph, how many tasks are on the critical path?
2. For each input graph, what is the *minimum* finite number of processors needed to execute it without exceeding the time on infinite processors? (You can use the `des.py` tool on the schedules you produced for infinite graphs to determine the execution time for infinite processors).

Exercise 2

(Slack) On a machine with infinite processors, if a task can be delayed *without* increasing the overall execution time, the task is said to possess a non-zero slack.

Formally, if $T_{earliest}$ is the earliest time that a task can run, and T_{latest} is the latest time it can run without increasing overall execution time, then its slack is $T_{latest} - T_{earliest}$.

Let us define a *critical task* as a task that is on the critical path.

Assume three tasks A, B, C are dependent on each other in this manner: $A \rightarrow B \rightarrow C$ (i.e. C depends on B which depends on A, other dependences may exist and are not shown). If A and C are critical tasks, is B also a critical task?

Provide rationale.

Exercise 3

(Resource-constrained Scheduling) Produce schedules for all input dependence graphs for `Nproc` $\in 1, 2, 4, 8, 16$.

Ideally, try to lower the execution time while having the highest utilization per processor. You are allowed to compare your execution times and utilization levels with your classmates for motivation, but you may not reveal your scheduling strategy.

1. In a table, report the execution time (from `des.py`) vs the number of processors for each input graph.
2. How did you schedule tasks when the number of tasks that were ready to execute exceeded the number of processors?

Exercise 4

(EXTRA CREDIT) Write a program that takes an input dependence graph and outputs an integer linear programming (ILP) formulation for scheduling that can be solved by an off-the-shelf solvers (use the GNU Linear Programming Kit).

Evaluate your schedule using the supplied `des.py` program.

Compare the results of the solver-generated schedule with your scheduler.

END.