CS-472: Design Technologies for Integrated Systems

Programming Assignment 3 Due date: 10/11/2022, 23:59

1 Introduction

In this homework, we are using ILP (integer linear programming) to minimize the latency of given sequencing graphs under resource constraints. Please refer to p.23 of slides 05 and p.199 of the textbook for the theoretical explanation of this formulation.

The constructed ILP system will be written into the `.lp` file format[1], which can then be accepted by and solved with an ILP solver (see Section 4 below). In the provided code and the following of this document, the term "constraint" is used interchangeably with the linear inequality in an ILP problem.

2 Input Format

The first line of the input file contains three numbers: number of resource types m, number of operations n, and upper bound of latency $\bar{\lambda}$.

Then, the following m lines specify each resource type with the following format: *name* d a, where *name* is a character briefing the resource type name, d is its delay (number of cycles an operation takes) and a is the number of the resource allowed (maximum number of operations of this type at each timeframe).

Finally, the following $n-1$ lines specify the sequencing graph. Each line describes an operation with a *type* character matching one of the resource type names followed by a set of operation IDs (ID 0 is NOP at timeframe 0 and the first operation has ID 1, etc. until the $(n-1)$-th operation) that have to be operated before this one. The last line encodes the final NOP (which is the n-th operation with ID n) with a type name of `0` and, similarly, the IDs of the operations computing the final results.

3 Coding Interface

The data structure representing the sequencing graph (`class Problem`) and the ILP constraint system (`class ILP`), as well as the parsing and dumping code, are already implemented for you. Your task is to finish the constraint encoding in `ILP.hpp`. The constraints should be encoded into the data members of this class, but not be printed out. Search for `TODO` to find where you should do your work.

The data structures are described as follows:

[1] `http://lpsolve.sourceforge.net/5.5/lp-format.htm`

3.1 class `Problem`

This class holds the information of the given sequencing graph with two internal structs `Resource` and `Operation`. `struct Resource` stores the delay (d) and number (a) of a resource type, and `resources` maps from `char` (the name of resource types) to `Resource`.

`struct Operation` records the type of resource an operation uses and the IDs of its predecessors (the operations that should be scheduled before it). The `Operation` objects can be accessed by its ID at `operations.at(ID)`. Note that the two NOPs have IDs 0 and n, meaning that there are in total $n + 1$ elements in `operations`.

You can print the sequencing graph with function `print()` (line 20 in `main.cpp`).

3.2 class `ILP`

This class formulates the system of linear inequalities from the given scheduling problem `prob`. There are two data members declared for your convenience:
`num_operations` $= n$, and `num_timeframes` $= \bar{\lambda} + 1$.

A `Variable` is represented with an operation ID `i` ($1 \leq$ `i` $\leq n$) and a timeframe number `l` ($1 \leq$ `l` $\leq \bar{\lambda} + 1$). Note that the first NOP (with ID 0) is always scheduled at timeframe 0, hence it is not involved in the ILP.

A `Constraint` consists of a list of involved `Variables` and their corresponding coefficients. The two vectors `variables` and `coefficients` should always be of the same size. The `type` of the constraint is encoded with an `enum Constraint_Type`, which is either `GE` (greater than or equal to; $\geq$), `LE` (less than or equal to; $\leq$) or `EQ` (equal to; $=$). The right-hand side of a constraint should always be a constant.

You can dump the formulated ILP problem into a file in the LP format with function `dump_lp()` (line 22 in `main.cpp`), or print it to the standard output with `print_lp()`.

4 (Optional) Solving with an ILP Solver

`lp_solve`[2] is an open-source (mixed) ILP solver. You can download its binary on their website[3]. Search for "lp_solve_5.5.2.11_exe" and choose the one that fits your platform. For example, choose the the `ux64` version to be run on the *selsrv1.epfl.ch* server.

You can solve the ILP problem formulated by your program with command `lp_solve scheduling.lp`, or simply uncomment line 23 in `main.cpp`. Feel free to play around with the solver. You can look up its list of options with command `lp_solve -h`.

5 Compiling, Testing, Submission and Grading

With a command line interface, type `make` to compile the code. An executable named `hw3` will be produced if compilation is successful. Type `./hw3 <path_to_test_file>` to test the code. For example, type `./hw3 testcases/test01.txt` to test on the first test case.

[2] `http://lpsolve.sourceforge.net/5.5/`

[3] `https://sourceforge.net/projects/lpsolve/files/lpsolve/5.5.2.11/`

Change the `0` in line `25` of `main.cpp` to `1` to run the grading code. Note that `lp_solve` has to be available for this part of code to run correctly. Expected results of the provided test cases are provided in the file `ref_output.txt`. Optionally, you can print the scheduled result with `print_schedule()` (line `29`).

Please submit `ILP.hpp`, compressed in one ZIP, to Moodle. Please do not change the names of the `hpp` files. Modifications to other files will not be considered during grading.

To avoid environmental issues, the grading will be done on the *selsrv1.epfl.ch* server. It is highly recommended to test your code on the server and compare to the reference outputs. No personalized modification will be made if there are compilation errors or redundant printing in your code that affects the automatic grading process.