Production Networks

# Introduction

A Global Production Network could be defined as:

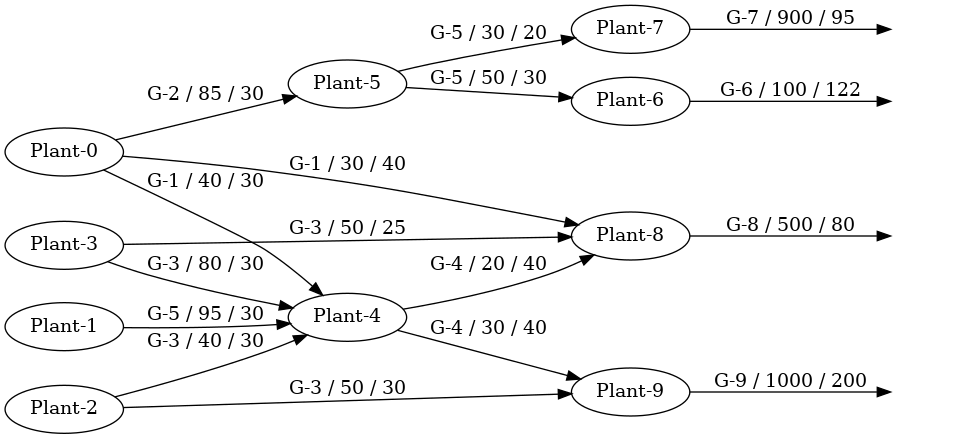
The nexus of interconnected functions, operations and transactions through which a specific product or service is produced, distributed and consumed.

<https://academic.oup.com/joeg/article/8/3/271/938286>

That could be thought of as a set of interconnected nodes and links where the nodes represent a producer or consumer and the links represent the commercial relationship between them.

## A general model

A production network could be modeled by a Graph where the nodes are the network participants and the arc are the exchange relationship between them. The following image shows a production network modeled by a graph:



Each arc is defined as follow:

* The source node is the seller.
* The target node is the buyer.
* The arc info is a tuple with the sold good, the amount sold, and its unitarian price.

The arcs in the last level (the rightmost) do not have a target node because they represent production for final consumers. Such arcs are only representative, they do not exist in the graph model.

# Practice

## The data model

To solve this problem, you have been provided with a data model with four elements: Good, Process, Plant, and Company.

* Good: represents any kind of product or service that could be exchanged in a production network.
* Process: represents the formula to produce a good. A process has a good as its output and a set of inputs. Each input is defined by a tuple of a good and the amount required to produce one unit of the output.
* Company: represents a registered name that produces goods.
* Plant: represents a branch of a company specialized in the production of some goods.

## The challenge

Your challenge here is to create a production network model by graphs given the model defined above.

This package is distributed as follows:

**lib/include/IOUtils.hpp y lib/src/IOUtils.cpp**: Contain the implementation of the output operator to write any of the data structures used in this project.

**Please, do not modify these files.**

**lib/include/TestUtils.hpp y lib/src/TestUtils.cpp**: Contain some basic operations for the unit tests.

**Plase, do not modify these files.**

**lib/include/Models.hpp** and **lib/src.Models.cpp**: Contain a class for each of the defined models. In order to ease the navigability between them, they have references in both directions. That means, for instance, a Process object has a reference to the produced good, it also contains a list of inputs. So, each good has a list of all of the processes that produce it and a list with all of the processes that consume it. Check the provided documentation to see more. In order to create objects of any of the models, I recommend you to use the class ModelBuilder and its methods create\_good, create\_process, create\_plant, and create\_company.

The files also contain the class ProductionData that has the set of goods, the set of processes, the set of plants, the set of companies, and a set with the purchase-sale relationships in the whole network.

**Please, do not modify these files.**

**lib/include/Generator.hpp** and **lib/src/Generator.cpp**: contain the function generate\_random\_production\_data to create random network models. Use it wisely.

**Please, do not modify these files.**

**lib/include/Rng.hpp** and **lib/src/Rng.cpp**: contain a singleton to generate random numbers.

**Please, do not modify these file.**

**lib/include/ProductionNetwork.hpp** and **lib/include/ProductionNetwork.cpp:** contain the class ProductionNetwork with all of its methods empty. They are ready to be solved by you.

* The aliases: You can see some aliases in this file in order to avoid writing a lot. **NodeInfoType** is the type that you should store in the nodes of the graph. **ArcInfoType** is the type that you should store in the arcs of the graph, this is provided as **void**, you should replace it with the type of graph that you want to use. **NetworkType** is the type that represents the graph. **NodeData**, **ArcData**, and **ChangedPriceData** are defined to retrieve information from your solution.
* The constructor: The class should be constructed from a **ProductionData** instance. You have to build a graph that satisfies all of the data in the argument. **NOTE: Not all of the plants will belong to the network. The graph should contain only the plants that have sale-purchase relationships.**
* The destructor: This is optional. If you need to clean your attributes before finishing the program execution, this is the place.
* Copy constructor, move constructor, and overloaded assignment operator are deleted for this class.
* **std::vector<NodeData> get\_node\_info\_list() const noexcept**: You have to solve this method. It should return a list with the info stored in all of the nodes of the graph.
* **std::vector<ArcData> get\_arc\_info\_list() const noexcept**: You have to solve this method. It should return a list with the info stored in all of the nodes of the graph. That is a tuple with the seller company, the buyer company, the good, the amount sold, and the unitarian price.
* **std::vector<ChangedPriceData> change\_price(std::shared\_ptr<Good>& good, float change)**: You have to solve this method. Given a good and a value to change its price, this method add this value to the price of the good, then it should change the prices of all of the products that use the goods as inputs, and so on.
* **std::ostream& to\_dot(std::ostream& output) const noexcept**: You have to solve this method. This is for you. It should write in the ostream object the representation of the graph in the dot language to be able to generate a graphic of your network.

**main/src/ProductionNetwork.cpp**: contains a main program that generates a random production data, builds the network and you can type a name of a first-level good and an amount to change it price to see the solution.

**test/src/ProductionNetwork.cpp**: This file contains a main program with a basic unit test.

**Makefile**: contains the rules to compile the project. **make main** will compile the program in the path **main/src**, **make test** wil compile your test program, and **make all** will compile both of them.

**Documentation.pdf**: contains the documentation of the models and utilities provided.

# Requirements

* C++14 or higher
* STL
* You are allowed to use the class Graph, Digraph, or any data structures of the library [DeSiGNAR](https://github.com/R3mmurd/DeSiGNAR).
* Make
* The suite clang to compile. If you use gcc, you should modify the variable CXX in the Makefile.
* Be familiarized with the standard smart pointers, specifically the type [std::shared\_ptr](https://en.cppreference.com/w/cpp/memory/shared_ptr).

# 

# Evaluation

This project could be solved by couples (choose wisely who you work with).

You have all the semester to submit this project and you are allowed to submit once a week. If you submit and fail, then you will be able to submit again the next week and so on until the semester is finished.

In order to evaluate your work, you should send the files ProductionNetwork.hpp and ProductionNetWor.cpp to the email [alejandro.j.mujic4@gmail.com](mailto:alejandro.j.mujic4@gmail.com) with the subject: [AYDA]-05-ProductionNetwork. The email body should content the names and the id numbers of the couple.