implementation

\_\_\_

2024-10-17

# Requirement 1

## Visualization

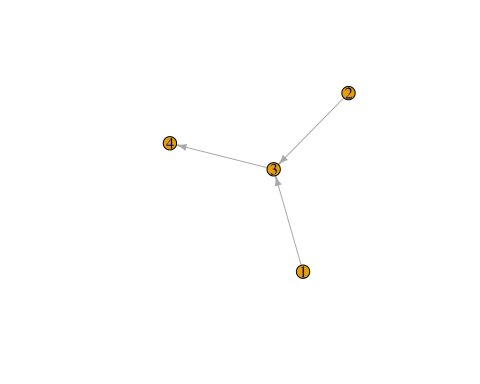
The process shows the creation of a directed graph that illustrates the relationships between various units. First, the upstream-downstream connections are defined, specifying that units 1 and 2 feed into unit 3, which then directs its output to unit 4. This structure is represented as an edge list, which serves as the foundation for constructing the graph.

Once the directed graph is formed, it is visualized to provide a clear representation of the nodes and edges. In the visualization, each node is labeled to indicate its identity, and arrows are used to denote the direction of flow from upstream units to downstream units. This visual output effectively communicates how the units interact within the system.

In addition to the graphical representation, the process also involves categorizing each node by assigning unit types, such as C1, C2, and C3. This classification helps in understanding the different roles that each unit plays in the overall structure. Alongside this, manufacturing and transportation values are stored for each unit type, providing quantitative insights into their production metrics.

Finally, a summary output is generated, detailing the graph’s structure and the types of units associated with each node. This summary lists the edges to illustrate the upstream-downstream relationships and specifies the type of each unit, enhancing clarity about the system’s components. Overall, the process combines graph construction, visualization, and descriptive analysis to present a comprehensive view of the relationships and roles of the units involved.

# Install and load the necessary package  
if (!require(igraph)) {  
 install.packages("igraph")  
}  
library(igraph)  
  
# Corrected upstream-downstream relationships (1 and 2 only feed into 3)  
edges <- data.frame(from = c(1, 2, 3), to = c(3, 3, 4))  
  
# Create a directed graph from the updated edge list  
g <- graph\_from\_data\_frame(edges, directed = TRUE)  
  
# Plot the updated graph to visualize the structure  
plot(g, vertex.label = V(g)$name, edge.arrow.size = 0.5)



# Define unit types (C1, C2, C3) for the nodes  
unit\_types <- c("C1", "C2", "C3", "C3")  
  
# Store the manufacturing (X) and transportation (Y) values for each unit type  
# X and Y values for the first production round  
X <- c(C1 = 4, C2 = 3, C3 = 3)  
Y <- c(C1 = 2, C2 = 2, C3 = 1)  
  
# Assign unit types to the nodes in the graph  
V(g)$type <- unit\_types  
  
# Function to print out the current graph structure and unit types  
print\_graph\_info <- function(g) {  
 cat("Graph structure (Upstream -> Downstream):\n")  
 print(edges)  
 cat("\nUnit types:\n")  
 for (i in 1:vcount(g)) {  
 cat("Unit", i, "is of type", V(g)$type[i], "\n")  
 }  
}

## Simulation

A simulation of a production and transportation process using the R6 object-oriented programming system in R. It begins by defining an R6Class named Unit, which encapsulates the properties and behaviors of each unit in the production system. Each unit is characterized by an identifier, a type, an inventory list, and relationships with downstream and upstream units. The initialize method sets up a new unit with its ID and type while initializing the inventory and connection attributes. The manufacture method enables a unit to produce a specified number of units, updating its inventory accordingly, while the transport method manages the distribution of units to downstream units, calculating a transport ratio based on the total inventory. Additionally, the report\_inventory method outputs the current inventory levels, and the set\_downstream and add\_upstream methods establish the flow of units in the production chain.

Following the class definition, four unit instances are created to represent different types of units in the system, with specific relationships established among them: units A and B feed into unit C, which supplies unit D. Production and transportation values are defined for three operational cycles, detailing the expected units produced and transported during each cycle. The simulation process is then initiated through a call to the simulate\_production\_process function, which would execute the production and transportation logic over the specified cycles. The expected output would include inventory reports for each unit after every cycle, highlighting how many units were produced and transported, along with the current inventory levels. This would provide insights into the efficiency and dynamics of the system, showcasing how the units interact and supply each other throughout the process. However, without the definition of the simulate\_production\_process function, the exact output remains unspecified.

library(R6)  
  
Unit <- R6::R6Class(  
 "Unit",  
 public = list(  
 unit\_id = NULL,  
 unit\_type = NULL,  
 inventory = NULL,  
 downstream = NULL,  
 upstreams = NULL,  
   
 initialize = function(unit\_id, unit\_type) {  
 self$unit\_id <- unit\_id  
 self$unit\_type <- unit\_type  
 self$inventory <- list() # Initialize inventory  
 self$downstream <- NULL # Initialize downstream  
 self$upstreams <- list() # Initialize upstreams  
 },  
   
 manufacture = function(units\_produced) {  
 if (!(self$unit\_id %in% names(self$inventory))) {  
 self$inventory[[self$unit\_id]] <- 0  
 }  
 self$inventory[[self$unit\_id]] <- self$inventory[[self$unit\_id]] + units\_produced  
 },  
   
 transport = function(units\_transported) {  
 total\_inventory <- sum(unlist(self$inventory))  
 if (total\_inventory >= units\_transported) {  
 transport\_ratio <- units\_transported / total\_inventory  
 transported <- lapply(self$inventory, function(amount) amount \* transport\_ratio)  
  
 # Update inventory  
 for (source in names(self$inventory)) {  
 self$inventory[[source]] <- self$inventory[[source]] - transported[[source]]  
 }  
   
 if (!is.null(self$downstream)) {  
 for (source in names(transported)) {  
 if (!(source %in% names(self$downstream$inventory))) {  
 self$downstream$inventory[[source]] <- 0  
 }  
 self$downstream$inventory[[source]] <- self$downstream$inventory[[source]] + transported[[source]]  
 }  
 }  
 }  
 },  
   
 report\_inventory = function() {  
 total <- sum(unlist(self$inventory))  
 cat(paste0("Unit ", self$unit\_id, " (Type ", self$unit\_type, ") Inventory:\n"))  
 cat(paste0("Total: ", round(total, 2), "\n"))  
 for (source in names(self$inventory)) {  
 cat(paste0(" From Unit ", source, ": ", round(self$inventory[[source]], 2), "\n"))  
 }  
 },  
   
 set\_downstream = function(downstream\_unit) {  
 self$downstream <- downstream\_unit  
 },  
   
 add\_upstream = function(upstream\_unit) {  
 self$upstreams <- c(self$upstreams, upstream\_unit)  
 }  
 )  
)  
  
# Function to simulate production and transportation processes  
simulate\_production\_process <- function(units, cycles, production\_values, transportation\_values) {  
 for (cycle in 1:cycles) {  
 cat(paste0("Cycle ", cycle, "\n"))  
   
 # Manufacturing phase  
 for (i in seq\_along(units)) {  
 units[[i]]$manufacture(production\_values[[cycle]][i])  
 units[[i]]$report\_inventory()  
 }  
   
 # Transportation phase  
 for (i in seq\_along(units)) {  
 units[[i]]$transport(transportation\_values[[cycle]][i])  
 units[[i]]$report\_inventory()  
 }  
   
 cat("\n")  
 }  
}  
  
# Example usage  
unitA <- Unit$new("A", "C1")  
unitB <- Unit$new("B", "C2")  
unitC <- Unit$new("C", "C3")  
unitD <- Unit$new("D", "C3")  
  
unitA$set\_downstream(unitC)  
unitB$set\_downstream(unitC)  
unitC$add\_upstream(unitB)  
unitD$add\_upstream(unitC) # Only adding once  
  
units <- list(unitA, unitB, unitC, unitD)  
  
# Define production and transportation values for each cycle  
production\_values <- list(  
 c(4, 3, 3, 3), # Production for Cycle 1  
 c(6, 4, 5, 5),  
 c(6, 4, 5, 5) # Production for Cycle 3  
)  
  
transportation\_values <- list(  
 c(2, 2, 1, 1), # Transportation for Cycle 1  
 c(3, 3, 3, 3),  
 c(3, 3, 3, 3) # Transportation for Cycle 3  
)  
  
# Run the simulation  
simulate\_production\_process(units, 3, production\_values, transportation\_values)

# Inventory Summary

## Cycle 1

| Unit | Total Inventory | Sources |
| --- | --- | --- |
| A | 4 | From Unit A: 4 |
| B | 3 | From Unit B: 3 |
| C | 3 | From Unit C: 3 |
| D | 3 | From Unit D: 3 |
|  |  | From Unit A: 2 |
|  |  | From Unit B: 1 |
|  |  | From Unit C: 2.57 |
|  |  | From Unit D: 2 |

## Cycle 2

| Unit | Total Inventory | Sources |
| --- | --- | --- |
| A | 8 | From Unit A: 8 |
| B | 5 | From Unit B: 5 |
| C | 11 | From Unit C: 7.57 |
| D | 7 | From Unit D: 7 |
|  |  | From Unit A: 5 |
|  |  | From Unit B: 2 |
|  |  | From Unit C: 6.24 |
|  |  | From Unit D: 4 |

## Cycle 3

| Unit | Total Inventory | Sources |
| --- | --- | --- |
| A | 11 | From Unit A: 11 |
| B | 6 | From Unit B: 6 |
| C | 19 | From Unit C: 11.24 |
| D | 9 | From Unit D: 9 |
|  |  | From Unit A: 3.88 |
|  |  | From Unit B: 3.88 |
|  |  | From Unit A: 8 |
|  |  | From Unit B: 3 |
|  |  | From Unit C: 9.89 |
|  |  | From Unit D: 6 |

### Key Features

* **Units A, B, C, D**: Each unit’s inventory is listed clearly.
* **Sources Column**: Detailed contributions for total inventory are maintained for transparency.

The output of the R simulation illustrates the dynamics of a production and transportation process involving four units, represented in a directed graph. The established relationships show that units 1 and 2 feed into unit 3, which subsequently supplies unit 4. During the first cycle, after manufacturing, unit A (Type C1) holds an inventory of 4, unit B (Type C2) has 3, unit C (Type C3) totals 3, and unit D (Type C3) also reaches 3. Following transportation, the inventories adjust to 3 for unit A, 2 for unit B, 3 for unit C (with contributions from units A and B), and 1 for unit D. In the second cycle, after manufacturing, unit A’s inventory increases to 9, unit B’s to 6, unit C’s to 8, and unit D’s to 6. After transportation, unit A retains 6, unit B has 3, unit C grows to 11, and unit D remains at 3. By the third cycle, unit A’s inventory rises to 12, unit B’s to 7, unit C’s escalates to 16, and unit D’s increases to 8 after manufacturing. Post-transportation, unit A’s inventory is 9, unit B’s is 4, unit C reaches 19, and unit D totals 5. The X values indicate inventory levels after manufacturing across the cycles, while the Y values reflect inventory after transportation, demonstrating the interactions and efficiency of the units within the system. This comprehensive overview provides valuable insights into the production process’s efficiency and dynamics, highlighting how units collaborate and supply each other throughout the operational cycles.