```julia
### s194624.jl

using Random

include("./IO.jl")
include("./SolutionBuilder.jl")

struct ArgumentException <: Exception
    message::String
end

function main()
    # Check arguments
    instanceLocation = ARGS[1]
    solutionLocation = ARGS[2]
    maxTime = parse(Int, ARGS[3])

    name, dim, LB ,rev, rev_pair, k, H, p = read_instance(instanceLocation)

    if (ARGS[2] == " ")
        solutionLocation = string("sols/", name, ".sol")
    end

    # Initialize solution
    sol = [Int[] for i in 1:k]
    revenue = 0
    remainingProducts = [i for i in 1:dim]
    manufacturingTimes = Int[0 for i in 1:k]

    destroyFunctions = [destroyPercentage, destroyPercentageAllProductionLines, destroyElementWithLeastRevenue]
    repairFunctions = [repairRandom, repairBestInsert]

    sol, revenue, manufacturingTimes, remainingProducts = repairRandom(sol, revenue, manufacturingTimes, remainingProducts, dim, rev, rev_pair, k, H, p)

    # ALSN
    println()
    println("Running ALSN for ", maxTime, " seconds...")
    println()

    iterations = 0
    elapsedTime = 0
    start = time_ns()

    while (elapsedTime < maxTime)

        # Print status message every 3 seconds
        if (elapsedTime % 3 == 0)
            println("\r", "Elapsed time: ", elapsedTime, "s, Iterations: ", iterations, ", Revenue: ", revenue, ", Lower bound: ", LB)
        end

        destroyFunction = destroyFunctions[rand(1:length(destroyFunctions))]
        repairFunction = repairFunctions[rand(1:length(repairFunctions))]

        newSol, newRevenue, newManufacturingTimes, newRemainingProducts =
            destroyFunction(sol, revenue, manufacturingTimes, remainingProducts, dim, rev, rev_pair, k, H, p)
        newSol, newRevenue, newManufacturingTimes, newRemainingProducts =
            repairFunction(newSol, newRevenue, newManufacturingTimes, newRemainingProducts, dim, rev, rev_pair, k, H, p)

        if (newRevenue > revenue)
            sol = newSol
            revenue = newRevenue
            manufacturingTimes = newManufacturingTimes
            remainingProducts = newRemainingProducts
        end
        elapsedTime = round((time_ns()-start)/1e9,digits=3)
        iterations += 1
    end

    println()
    println("Iterations: ", iterations)
    println("Final solution with revenue: ", revenue)
    println("Lower bound: ", LB)

    writeSolution(sol, solutionLocation)

    output = run(Cmd(["./POChecker.exe", instanceLocation, solutionLocation]))
    println(output)
end
main()
```

```julia
function read_instance(filename)
    f = open(filename)
    name = readline(f) # name of the instance
    size = parse(Int32,readline(f)) # number of order
    LB = parse(Int32,readline(f)) # best known revenue
    rev = parse.(Int32,split(readline(f)))# revenue for including an order
    rev_pair = zeros(Int32,size,size) # pairwise revenues
    for i in 1:size-1
        data = parse.(Int32,split(readline(f)))
        j=i+1
        for d in data
            rev_pair[i,j]=d
            rev_pair[j,i]=d
            j+=1
        end
    end
    readline(f)
    k = parse(Int32,readline(f)) # number of production lines
    H = parse(Int32,readline(f)) # planning horizon
    p = parse.(Int32,split(readline(f))) # production times
    close(f)
    return name, size, LB ,rev, rev_pair, k, H, p
end

function writeSolution(solution, solutionLocation)
    wDir = string(pwd())

    dir, file = splitdir(solutionLocation)
    if (!isdir(dir))
        mkpath(string("./", dir, "/"))
    end

    open(string(wDir, "/", solutionLocation), "w") do f
        for i in eachindex(solution)
            for j in solution[i]
                write(f, string(j, " "))
            end
            write(f, "\n")
        end
    end
end
```

```julia
### SolutionBuilder

using Random

# Destroy a percentage of elements from a random production line
function destroyPercentage(initSol, initRevenue, initManufacturingTimes, initRemainingProducts, dim, rev, rev_pair, k, H, p)
    percentage = rand([0.05, 0.10, 0.15, 0.20, 0.50])
    productionLine = rand(1:k)
    numOfElements = length(initSol[productionLine])
    elementsToRemove = round(Int, numOfElements * percentage)

    sol = deepcopy(initSol)
    revenue = initRevenue
    manufacturingTimes = deepcopy(initManufacturingTimes)
    remainingProducts = deepcopy(initRemainingProducts)

    # Remove percentage of elements from production line
    for i in 1:elementsToRemove
        element = rand(sol[productionLine])
        manufacturingTimes[productionLine] -= p[element]
        filter!(x -> x != element, sol[productionLine])
        push!(remainingProducts, element)
        for j in sol[productionLine]
            revenue -= rev_pair[j, element]
        end
        revenue -= rev[element]
    end
    return sol, revenue, manufacturingTimes, remainingProducts
end

# Remove fixed number of elements from all production lines
function destroyPercentageAllProductionLines(initSol, initRevenue, initManufacturingTimes, initRemainingProducts, dim, rev, rev_pair, k, H, p)
    percentage = rand([0.05, 0.10, 0.15, 0.20, 0.50])

    sol = deepcopy(initSol)
    revenue = initRevenue
    manufacturingTimes = deepcopy(initManufacturingTimes)
    remainingProducts = deepcopy(initRemainingProducts)

    # Iterate over all production lines
    for i in 1:k
        # Find number of elements to remove
        elementsToRemove = round(Int, length(sol[i]) * percentage)

        # Remove elements
        for j in 1:elementsToRemove
            element = rand(sol[i])
            manufacturingTimes[i] -= p[element]
            filter!(x -> x != element, sol[i])
            push!(remainingProducts, element)
            for j in sol[i]
                revenue -= rev_pair[j, element]
            end
            revenue -= rev[element]
        end
    end
    return sol, revenue, manufacturingTimes, remainingProducts
end

# For some random production line: Remove element that provides the least amount of total revenue + the bonus of elements in the same production line
function destroyElementWithLeastRevenue(initSol, initRevenue, initManufacturingTimes, initRemainingProducts, dim, rev, rev_pair, k, H, p)
    percentage = rand([0.05, 0.10, 0.15, 0.20, 0.50])
    numOfElementsToRemove = round(Int, dim * percentage)

    sol = deepcopy(initSol)
    revenue = initRevenue
    manufacturingTimes = deepcopy(initManufacturingTimes)
    remainingProducts = deepcopy(initRemainingProducts)

    # Do the code below a number of times in a for loop
    for i in 1:numOfElementsToRemove
        # Find production line
        productionLine = rand(1:k)

        if (length(sol[productionLine]) == 0)
            continue
        end

        # Find element with least revenue
        min = Inf
        minIdx = 0
        for i in sol[productionLine]
            revenueGain = rev[i]
            for j in sol[productionLine]
                revenueGain += rev_pair[j, i]
            end
            if (revenueGain < min)
                min = revenueGain
                minIdx = i
            end
        end

        # Remove element
        manufacturingTimes[productionLine] -= p[minIdx]
        filter!(x -> x != minIdx, sol[productionLine])
        push!(remainingProducts, minIdx)
        for j in sol[productionLine]
```

```julia
            revenue -= rev_pair[j, minIdx]
        end
        revenue -= rev[minIdx]
    end
    return sol, revenue, manufacturingTimes, remainingProducts
end

function repairRandom(initSol, initRevenue, initManufacturingTimes, initRemainingProducts, dim, rev, rev_pair, k, H, p)
    sol = deepcopy(initSol)
    revenue = initRevenue
    manufacturingTimes = deepcopy(initManufacturingTimes)
    remainingProducts = deepcopy(initRemainingProducts)

    productionLineOrder = [i for i in 1:k]
    shuffle!(productionLineOrder)

    while(length(remainingProducts) > 0)
        found = false
        for productionLine in productionLineOrder
            for remainingProduct in remainingProducts
                if (manufacturingTimes[productionLine] + p[remainingProduct] <= H)
                    found = true

                    # Update available time
                    manufacturingTimes[productionLine] += p[remainingProduct]

                    # Update solution
                    push!(sol[productionLine], remainingProduct)
                    filter!(x -> x != remainingProduct, remainingProducts)

                    # Update revenue
                    for j in sol[productionLine]
                        revenue += rev_pair[j, remainingProduct]
                    end
                    revenue += rev[remainingProduct]
                end
            end
        end
        if (!found)
            break
        end
    end
    return sol, revenue, manufacturingTimes, remainingProducts
end

function repairBestInsert(initSol, initRevenue, initManufacturingTimes, initRemainingProducts, dim, rev, rev_pair, k, H, p)
    sol = deepcopy(initSol)
    revenue = initRevenue
    manufacturingTimes = deepcopy(initManufacturingTimes)
    remainingProducts = deepcopy(initRemainingProducts)

    while(length(remainingProducts) > 0)
        bestProductionLine = 0
        bestRemainingProduct = 0
        bestRevenue = -Inf

        full = true
        for productionLine in 1:k
            for remainingProduct in remainingProducts
                if (manufacturingTimes[productionLine] + p[remainingProduct] <= H)
                    full = false

                    # Calculate total revenueGain from inserting this product here:
                    revenueGain = rev[remainingProduct]
                    for j in sol[productionLine]
                        revenueGain += rev_pair[j, remainingProduct]
                    end

                    if (revenueGain > bestRevenue)
                        bestRevenue = revenueGain
                        bestProductionLine = productionLine
                        bestRemainingProduct = remainingProduct
                    end
                end
            end
        end
        if (full)
            break
        end
        # Update available time
        manufacturingTimes[bestProductionLine] += p[bestRemainingProduct]

        # Update solution
        push!(sol[bestProductionLine], bestRemainingProduct)
        filter!(x -> x != bestRemainingProduct, remainingProducts)

        # Update revenue
        for j in sol[bestProductionLine]
            revenue += rev_pair[j, bestRemainingProduct]
        end
        revenue += rev[bestRemainingProduct]
    end
    return sol, revenue, manufacturingTimes, remainingProducts
end
```