```
## s194624.jl

using Random
include("IO.jl")
include("TABU.jl")

struct ArgumentException <: Exception
    message::String
end

function main()
    localSearchTime = 60
    instanceLocation = ARGS[1]
    solutionLocation = ARGS[2]
    totalTime = parse(Int, ARGS[3])

    name, UB, dim, dist = read_instance(instanceLocation)

    println("Dimension: ", dim)
    k = Int(round(dim/4))

    if (k == 0)
        k = 1
    end
    if (length(ARGS) > 3)
        k = parse(Int, ARGS[4])
    end
    println("k: ", k)

    diversifyFrequency = 1

    println("Running instance: ", name)
    println(string("Upper bound: ", UB))

    if (ARGS[2] == " ")
        vals = rsplit(name, ".", limit=2)
        solutionLocation = string("sols/", vals[1], ".sol")
    end

    # Initialize with solution using nearest neighbor
    println("Finding initial solution...")
    s, objectiveValue = nearestNeighbor(dist, dim)
    println("Initial solution found")

    # Find the initial local minimum
    iterations = 0
    elapsedTime = 0

    # Perform iterated local search
    println("Allowed time: ", totalTime, " seconds")

    bestSolution = copy(s)
    bestObjectiveValue = objectiveValue

    previousMove = (-1,-1)
    visitedSolutions = [s]

    noLegalNeighbors = true

    updates = 1
    shuffleFrequency = 10
    switch = 0

    lastUpdateTime = elapsedTime
    start = time_ns()
    while (elapsedTime < totalTime)

        s, objectiveValue, noLegalNeighbors = BestNonTABU(s, objectiveValue, dim, dist, visitedSolutions)
        visitSolution(visitedSolutions, s, k)
        if (objectiveValue < bestObjectiveValue)
            bestSolution = copy(s)
            bestObjectiveValue = objectiveValue
            println()
            println("Time: ", elapsedTime, " seconds")
            println("New best: ", bestObjectiveValue)
```

```
            println("Upper bound: ", UB)
            lastUpdateTime = elapsedTime
        end

        if (noLegalNeighbors || (elapsedTime - lastUpdateTime) > diversifyFrequency)
            # Diversification
            swaps = Int(ceil(dim/2))
            for i in 1:swaps
                edgeA, edgeB = getRandomEdgePair(dim)
                s, objectiveValue = twoOpt(s, objectiveValue, edgeA, edgeB, dist)
            end

            if (updates % shuffleFrequency == 0)
                shuffle!(s)
            end

            updates += 1
            s = makeFeasible(s, dist)
            objectiveValue = getObjectiveValue(s, dist)
            lastUpdateTime = elapsedTime
        end

        iterations += 1
        elapsedTime = round((time_ns()-start)/1e9,digits=3)
    end

    println("\nSearch completed.")
    println(string(iterations, " total iterations"))
    println(string("Final objective value: ", bestObjectiveValue))
    println(string("Upper bound: ", UB))

    writeSolution(bestSolution, solutionLocation)
end
main()
```