```
## TABU.jl

function insertElement(list, element, dist)
    n = size(list)[1]
    for i in n:-1:1
        if (dist[element,list[i]] == -1)
            insert!(list, i+1, element)
            return list
        end
    end
    insert!(list, 1, element)
    return list
end

function makeFeasible(s, dist)
    n = length(s)
    solution = [s[n]]
    for i in n-1:-1:1
        solution = insertElement(solution, s[i], dist)
    end
    return solution
end

function checkNeighbors(ID, visited, dist, dim)
    max = maximum(dist)+1
    currentList = zeros(dim)

    for i in eachindex(currentList)
        if i in visited
            currentList[i] = max
        else
            currentList[i] = dist[ID,i]
        end
    end
    return findmin(currentList)
end

function nearestNeighbor(dist, dim)
    visited = zeros(Int,1)
    visited[1] = 1

    nextNode = 1
    while(size(visited)[1] < dim)
        shortDist, nextNode = checkNeighbors(nextNode, visited, dist, dim)
        append!(visited,nextNode)
    end
    solution = makeFeasible(visited, dist)
    return solution, getObjectiveValue(solution, dist)
end

# Calculates the objective value of a given solution iteratively
function getObjectiveValue(solution, dist)
    val = 0
    n = size(solution)[1]
    for i in 1:n-1
        val = val + dist[solution[i], solution[i+1]]
    end
    return val
end

function legalPair(n, m)
    if (abs(n-m) < 2)
        return false
    end

    return true
end

function twoOpt(s, objectiveValue, edgeA, edgeB, dist)
    newSolution = copy(s)
    newSolution[edgeA+1], newSolution[edgeB] = newSolution[edgeB], newSolution[edgeA+1]
    newObjectiveValue = swapObjectiveValue(s, objectiveValue, edgeA, edgeB, dist)
    return newSolution, newObjectiveValue
end
```

```
# Given two edges 'n' and 'm' recalculate the relevant costs for the new solution
function swapObjectiveValue(s, sum, n, m, dist)
    s = copy(s)
    if (abs(n-m) == 2)
        sum = sum - (dist[s[n],s[n+1]] + dist[s[n+1],s[m]]
                   + dist[s[m],s[m+1]])
        sum = sum + (dist[s[n],s[m]] + dist[s[m],s[n+1]]
                   + dist[s[n+1],s[m+1]])
    else
        sum = sum - (dist[s[n],s[n+1]] + dist[s[n+1],s[n+2]]
                   + dist[s[m-1],s[m]] + dist[s[m],s[m+1]])
        sum = sum + (dist[s[n],s[m]] + dist[s[m],s[n+2]]
                   + dist[s[m-1],s[n+1]] + dist[s[n+1],s[m+1]])
    end
    return sum
end

# Returns true if solutions are equal, otherwise false
function compareSolutions(sol1, sol2)
    for i in eachindex(sol1)
            if (sol1[i] != sol2[i])
                return false
            end
        end
    return true
end

function solutionVisited(s, visitedSolutions)
    for prevSolution in visitedSolutions
        if (compareSolutions(s, prevSolution))
            return true
        end
    end
    return false
end

function isLegal(s, dist)
    for i in eachindex(s)
        for j in i:length(s)
            if (dist[s[i],s[j]] == -1)
                return false
            end
        end
    end
    return true
end

function beenVisited(OGSolution, previousMoves, dist)
    s = copy(OGSolution)
    for move in previousMoves
        newSolution, newObjectiveValue = twoOpt(s, objectiveValue, move[1], move[2], dist)
    end
end

function BestNonTABU(originalSolution, originalObjectiveValue, dim, dist, visitedSolutions)
    s = copy(originalSolution)
    objectiveValue = copy(originalObjectiveValue)
    noLegalNeighbors = true
    for i in 1:dim-1
        for j in i:dim-1
            if(legalPair(i, j))
                newSolution, newObjectiveValue = twoOpt(originalSolution, originalObjectiveValue, i, j, dist)
                if (isLegal(newSolution, dist) && !solutionVisited(newSolution, visitedSolutions))
                    if (noLegalNeighbors || newObjectiveValue < objectiveValue)
                        s = newSolution
                        objectiveValue = newObjectiveValue
                    end
                    noLegalNeighbors = false
                end
            end
        end
    end
    return s, objectiveValue, noLegalNeighbors
end
```

```
function getRandomEdgePair(dim)
    population = [i for i in 1:dim-1]
    edgeA = population[rand(1:length(population))]
    filter!(x -> abs(edgeA - x) > 2, population)
    edgeB = population[rand(1:length(population))]
    if (edgeB < edgeA)
        edgeA, edgeB = edgeB, edgeA
    end
    return edgeA, edgeB
end

function visitSolution(visitedSolutions, s, k)
    n = length(visitedSolutions)
    if (n == k)
        deleteat!(visitedSolutions, 1)
    end
    push!(visitedSolutions, s)
end
```