Assignment 6 –

// Assignment 7 Part A

#include <iostream>

#include <vector>

#include <algorithm>

#include <limits>

using namespace std;

const int INF = numeric\_limits<int>::max();

// Function to calculate the lower bound using Held-Karp relaxation

int calculateLowerBound(vector<vector<int>>& graph, vector<int>& path) {

int lowerBound = 0;

// Calculate the cost of the current path

for (int i = 0; i < path.size() - 1; ++i) {

if (graph[path[i]][path[i + 1]] == INF) {

return INF; // Path is not feasible

}

lowerBound += graph[path[i]][path[i + 1]];

}

// Add the cost of returning to the starting city

lowerBound += graph[path.back()][path.front()];

// Reduce the lower bound using Held-Karp relaxation

for (int i = 0; i < graph.size(); ++i) {

if (find(path.begin(), path.end(), i) == path.end()) {

int minEdge = INF;

for (int j = 0; j < graph.size(); ++j) {

if (i != j) {

minEdge = min(minEdge, graph[i][j]);

}

}

lowerBound += minEdge;

}

}

return lowerBound;

}

// Branch and Bound recursive function

void tspBranchAndBound(vector<vector<int>>& graph, vector<int>& path, int& minLength, int& optimalPathLength) {

int n = graph.size();

if (path.size() == n) {

int currentLength = 0;

for (int i = 0; i < n - 1; ++i) {

currentLength += graph[path[i]][path[i + 1]];

}

currentLength += graph[path.back()][path.front()];

if (currentLength < optimalPathLength) {

optimalPathLength = currentLength;

}

return;

}

// Iterate over all cities

for (int i = 0; i < n; ++i) {

// If the city is not visited yet

if (find(path.begin(), path.end(), i) == path.end()) {

path.push\_back(i);

int lowerBound = calculateLowerBound(graph, path);

// Only explore if the lower bound is smaller than the current optimal length

if (lowerBound < optimalPathLength) {

tspBranchAndBound(graph, path, minLength, optimalPathLength);

}

path.pop\_back();

}

}

}

void solveTSPBranchAndBound(vector<vector<int>>& graph) {

int n = graph.size();

vector<int> path;

int minLength = INF;

int optimalPathLength = INF;

path.push\_back(0);

tspBranchAndBound(graph, path, minLength, optimalPathLength);

cout << "Optimal TSP Path Length: " << optimalPathLength << endl;

}

int main() {

vector<vector<int>> graph = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

solveTSPBranchAndBound(graph);

return 0;

}