```cpp
#include <iostream>
#include "graph.h"

using namespace std;

void findPoint(Graph* tree, int from, int to);


int main() {
	Graph* tree = new Graph();

	int iChosenCommand = 0;

	while (iChosenCommand != 10) {
		cout << "\nChoose the command: "
			<< "\n1. Enter nodes"
			<< "\n2. Enter edges"
			<< "\n3. Show graph"
			<< "\n4. Find the shortest way"
			<< "\n5 Clear"
			<< "\n6. Autotest"
			<< "\n10. Exit" << endl;
		cin >> iChosenCommand;
		if (iChosenCommand == 10) break;

		switch (iChosenCommand) {
		case 1: {
			int count = 0;
			cout << "\nEnter nodes count (Nodes will be added from 0 -
> count): ";
			cin >> count;
			for (int i = 0; i < count + 1; i++) {
				tree->addNode(i);
			}
			break;
		}
		case 2: {
			int choser = 0;
			cout << "Choose orientation of graph ( 1 - orgraph, 2 -
neorgraph) : ";
			cin >> choser;
			cout << "\nEnter -1 -1 -1 , if you want to stop adding";
			cout << "\nEnter Key from, Key to, Weight : ";
			int iKeyFrom, iKeyTo, iWeight;
			cin >> iKeyFrom >> iKeyTo >> iWeight;
			while (iKeyFrom != -1 && iKeyTo != -1 && iWeight != -1) {
				if (choser == 2) tree->addEdge(iKeyTo, iKeyFrom,
iWeight);
				tree->addEdge(iKeyFrom, iKeyTo, iWeight);
				cout << "\nEnter Key from, Key to, Weight : ";
				cin >> iKeyFrom >> iKeyTo >> iWeight;
			}
			break;
		}
		case 3: {
			tree->showGraph();
			break;
		}
```

```
		case 4: {
			int iKeyFrom, iKeyTo;
			cout << "\nEnter Key From, Key To: ";
			cin >> iKeyFrom >> iKeyTo;
			map<int, pair<int, int>>* wayMap = new map<int, pair<int,
int>>();
			vector<int>* alreadyVisited = new vector<int>();
			for (int i = 0; i < tree->getSize(); i++) alreadyVisited-
>push_back(-1);
			tree->wayFinder(wayMap, tree->getNode(iKeyFrom), iKeyTo,
alreadyVisited);
			cout << "\nWay: ";
			if (tree->getNode(iKeyTo) == nullptr) {
				cout << "Key wasn't found";
				break;
			}
			int iCounter = iKeyTo;
			while (iCounter != iKeyFrom) {
				cout<<endl << iCounter;
				iCounter = (*wayMap)[iCounter].second;
			}

			cout << "\nWay weighted: " << (*wayMap)[iKeyTo].first;
			break;
		}
		case 5: {
			tree->clear();
		}
		case 6: {
			cout << "\nGraph 10";
			for (int i = 0; i < 9; i++) {
				tree->addNode(i);
			}
			tree->addEdge(0, 1, 0);
			tree->addEdge(1, 2, 23);
			tree->addEdge(1, 3, 12);
			tree->addEdge(2, 3, 25);
			tree->addEdge(2, 8, 35);
			tree->addEdge(3, 4, 18);
			tree->addEdge(4, 6, 20);
			tree->addEdge(5, 6, 23);
			tree->addEdge(5, 7, 14);
			tree->addEdge(6, 7, 24);
			tree->addEdge(7, 8, 16);

			tree->addEdge(2,1, 23);
			tree->addEdge(3, 1, 12);
			tree->addEdge(3, 2, 25);
			tree->addEdge(8, 2, 35);
			tree->addEdge(4, 3, 18);
			tree->addEdge(6, 4, 20);
			tree->addEdge(6, 5, 23);
			tree->addEdge(7, 5, 14);
			tree->addEdge(7, 6, 24);
			tree->addEdge(8, 7, 16);

			tree->showGraph();
```

```
findPoint(tree, 1, 8);

tree->clear();

cout << "\nGraph 7";
for (int i = 0; i < 11; i++) {
	tree->addNode(i);
}
tree->addEdge(0, 1, 0);
tree->addEdge(1, 2, 3);
tree->addEdge(1, 3, 4);
tree->addEdge(1, 4, 2);
tree->addEdge(2, 6, 3);
tree->addEdge(3, 6, 6);
tree->addEdge(4, 5, 5);
tree->addEdge(4, 6, 2);
tree->addEdge(5, 7, 6);
tree->addEdge(5, 9, 12);
tree->addEdge(6, 5, 1);
tree->addEdge(6, 7, 8);
tree->addEdge(6, 8, 7);
tree->addEdge(7, 10, 4);
tree->addEdge(8, 10, 3);
tree->addEdge(9, 8, 6);
tree->addEdge(9, 10, 11);

tree->showGraph();

findPoint(tree, 1, 6);

tree->clear();

cout << "\nGraph 2";
for (int i = 0; i < 7; i++) {
	tree->addNode(i);
}
tree->addEdge(0, 1, 0);
tree->addEdge(1, 4, 2);
tree->addEdge(1, 6, 4);
tree->addEdge(6, 4, 1);
tree->addEdge(6, 5, 8);
tree->addEdge(5, 4, 6);
tree->addEdge(5, 3, 3);
tree->addEdge(3, 4, 2);
tree->addEdge(3, 2, 1);
tree->addEdge(2, 4, 2);
tree->addEdge(2, 1, 7);

tree->addEdge(1, 0, 0);
tree->addEdge(4, 1, 2);
tree->addEdge(6, 1, 4);
tree->addEdge(4, 6, 1);
tree->addEdge(5, 6, 8);
tree->addEdge(4, 5, 6);
tree->addEdge(3, 5, 3);
tree->addEdge(4, 3, 2);
tree->addEdge(2, 3, 1);
tree->addEdge(4, 2, 2);
```

```
                tree->addEdge(1, 2, 7);

                tree->showGraph();

                findPoint(tree, 1, 6);

                tree->clear();

        }
        }
    }


    return 0;
}


void findPoint(Graph* tree, int from, int to) {
    map<int, pair<int, int>>* wayMap = new map<int, pair<int, int>>();
    vector<int>* alreadyVisited = new vector<int>();
    for (int i = 0; i < tree->getSize(); i++) {
        alreadyVisited->push_back(-1);
        (*wayMap)[i] = make_pair(-1, -1);
    }

    if (tree->getNode(to) == nullptr) {
        cout << "Key wasn't found";
        return;
    }

    tree->wayFinder(wayMap, tree->getNode(from), to, alreadyVisited);
    cout << "\nWay: ";
    int iCounter = to;
    while (iCounter != from) {
        cout << endl << iCounter;
        iCounter = (*wayMap)[iCounter].second;
    }
    cout << endl << from;
    cout << "\nWay weighted: " << (*wayMap)[to].first;
}
```