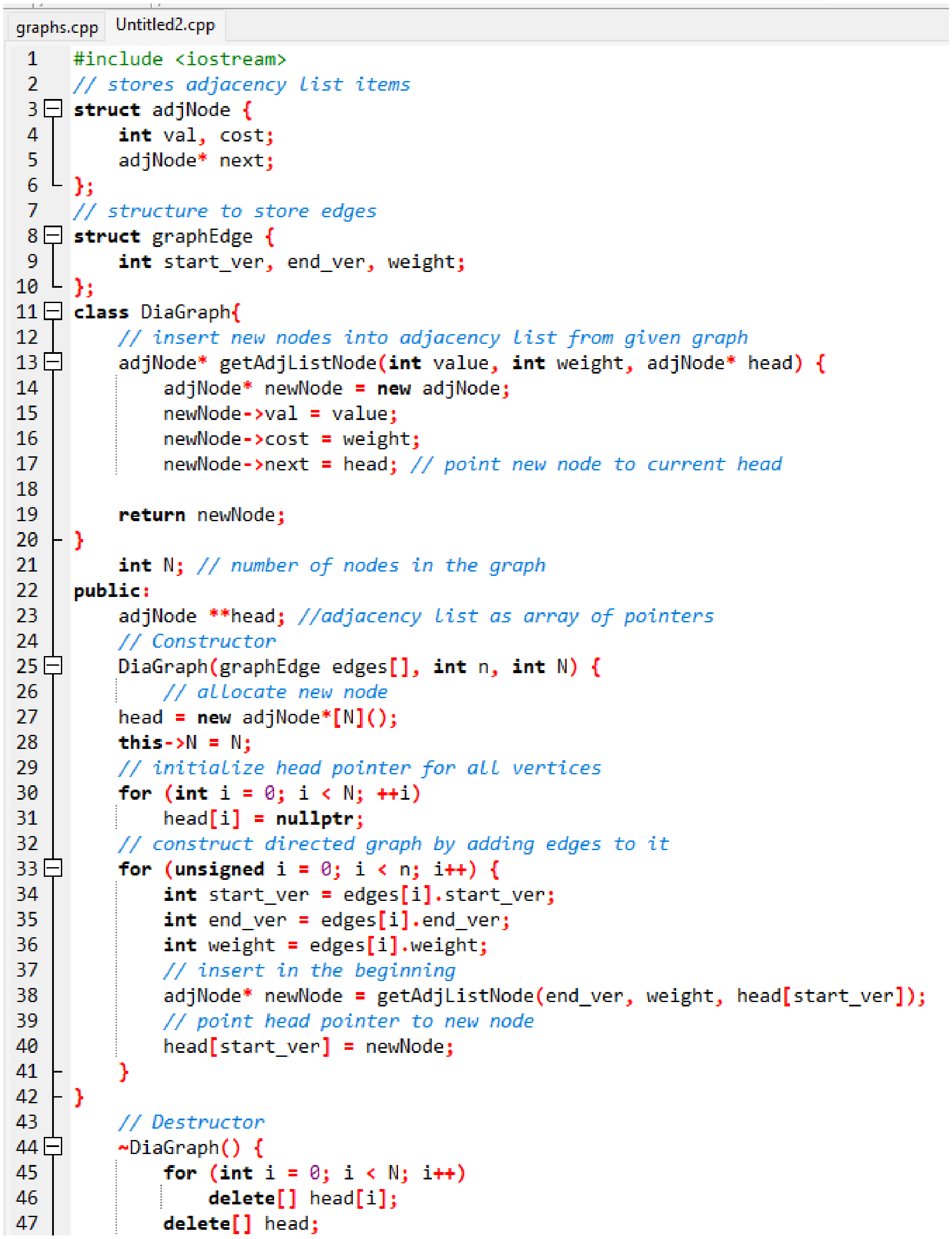
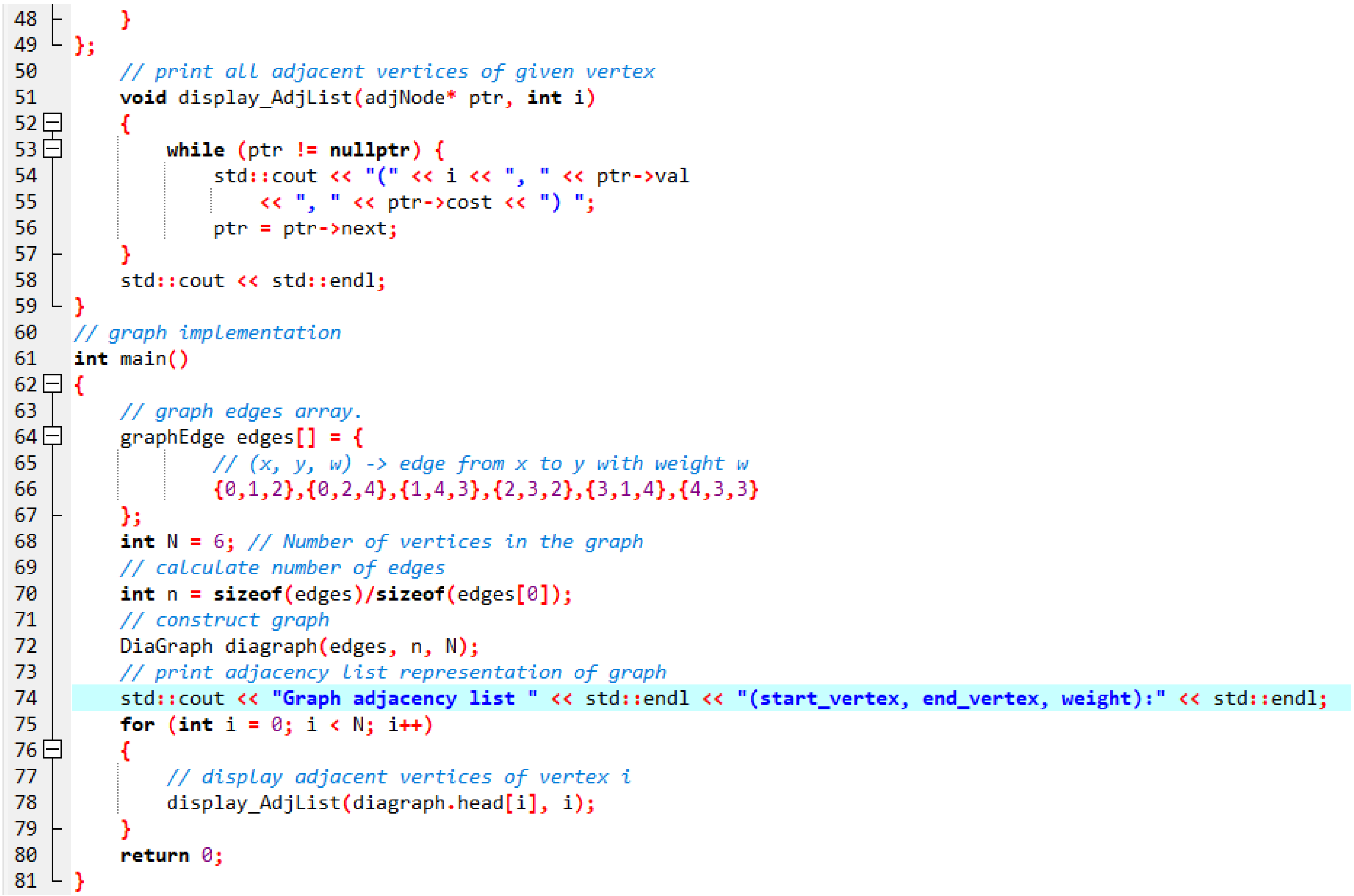
Activity No. 10.1	
Graphs	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** Sep 30, 2025
Section: CPE21S4	**Date Submitted:** Sep 30, 2025
Name(s): Bautista,Mariela S.	**Instructor:** Engr.Quejado
6. Output	

Code for ILO A: Create C++ code for graph implementation utilizing adjacency matrix and adjacency list

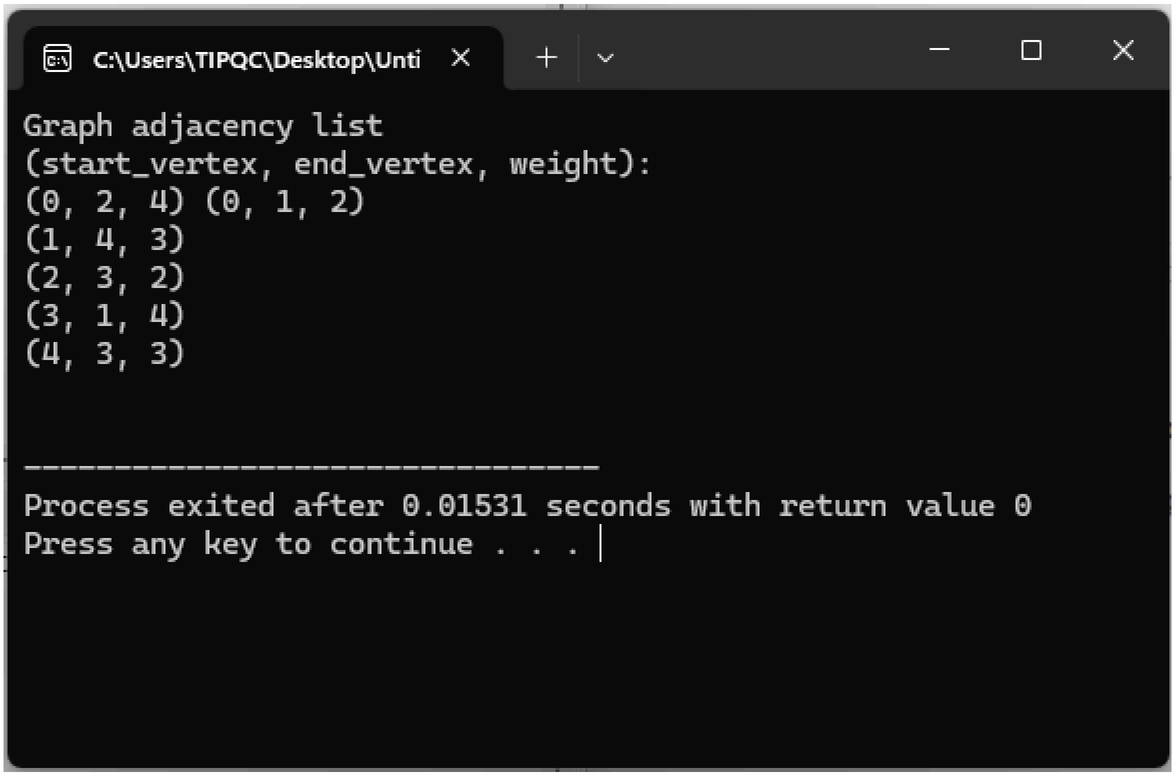


```cpp
#include <iostream>
// stores adjacency list items
struct adjNode {
    int val, cost;
    adjNode* next;
};
// structure to store edges
struct graphEdge {
    int start_ver, end_ver, weight;
};
class DiaGraph{
    // insert new nodes into adjacency list from given graph
    adjNode* getAdjListNode(int value, int weight, adjNode* head) {
        adjNode* newNode = new adjNode;
        newNode->val = value;
        newNode->cost = weight;
        newNode->next = head; // point new node to current head

    return newNode;
}
    int N; // number of nodes in the graph
public:
    adjNode **head; //adjacency list as array of pointers
    // Constructor
    DiaGraph(graphEdge edges[], int n, int N) {
        // allocate new node
    head = new adjNode*[N]();
    this->N = N;
    // initialize head pointer for all vertices
    for (int i = 0; i < N; ++i)
        head[i] = nullptr;
    // construct directed graph by adding edges to it
    for (unsigned i = 0; i < n; i++) {
        int start_ver = edges[i].start_ver;
        int end_ver = edges[i].end_ver;
        int weight = edges[i].weight;
        // insert in the beginning
        adjNode* newNode = getAdjListNode(end_ver, weight, head[start_ver]);
        // point head pointer to new node
        head[start_ver] = newNode;
    }
}
    // Destructor
    ~DiaGraph() {
        for (int i = 0; i < N; i++)
            delete[] head[i];
        delete[] head;
```



```
    }
};
    // print all adjacent vertices of given vertex
    void display_AdjList(adjNode* ptr, int i)
    {
        while (ptr != nullptr) {
            std::cout << "(" << i << ", " << ptr->val
                << ", " << ptr->cost << ") ";
            ptr = ptr->next;
    }
    std::cout << std::endl;
}
// graph implementation
int main()
{
    // graph edges array.
    graphEdge edges[] = {
        // (x, y, w) -> edge from x to y with weight w
        {0,1,2},{0,2,4},{1,4,3},{2,3,2},{3,1,4},{4,3,3}
    };
    int N = 6; // Number of vertices in the graph
    // calculate number of edges
    int n = sizeof(edges)/sizeof(edges[0]);
    // construct graph
    DiaGraph diagraph(edges, n, N);
    // print adjacency list representation of graph
    std::cout << "Graph adjacency list " << std::endl << "(start_vertex, end_vertex, weight):" << std::endl;
    for (int i = 0; i < N; i++)
    {
        // display adjacent vertices of vertex i
        display_AdjList(diagraph.head[i], i);
    }
    return 0;
}
```

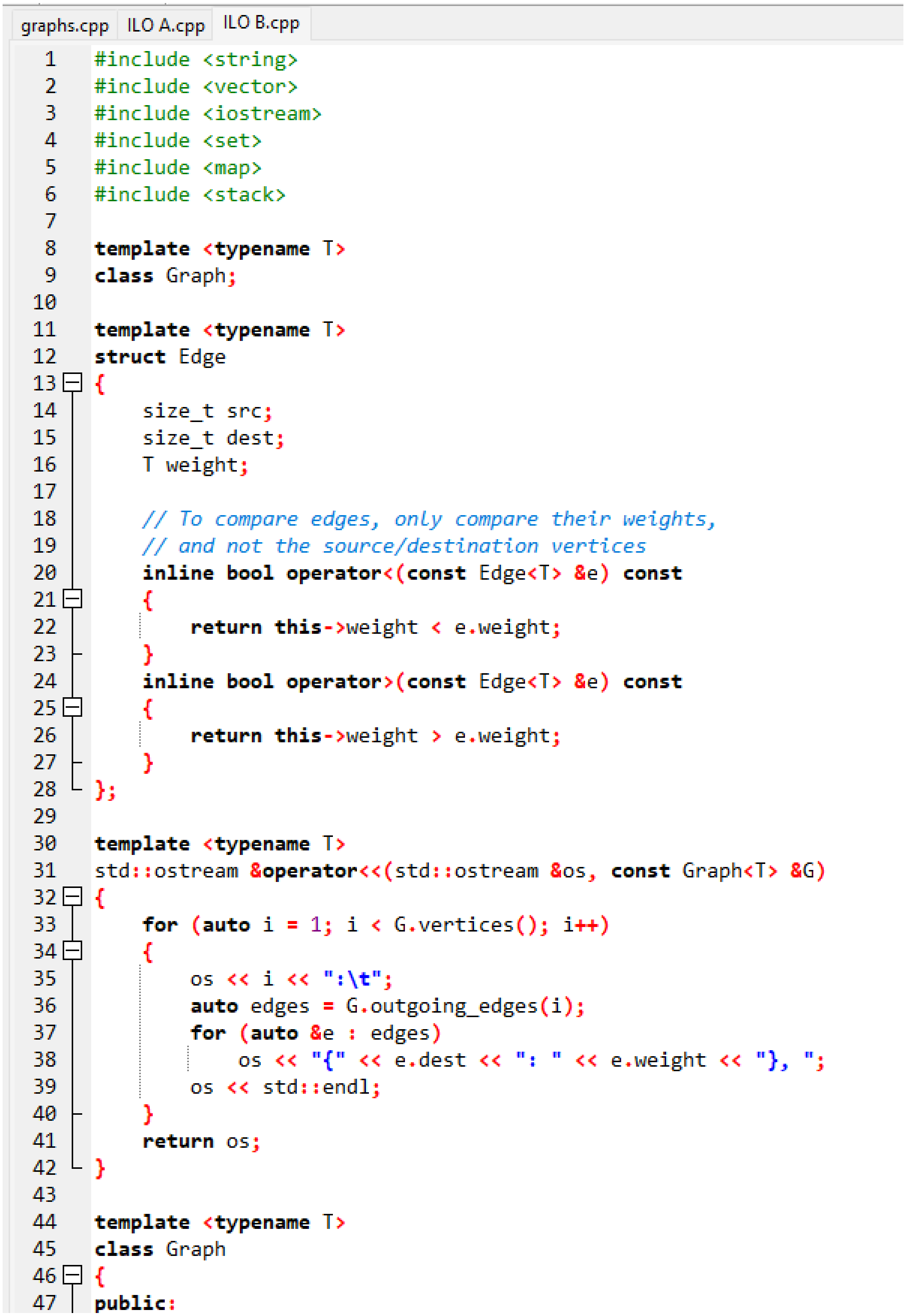
Ouput:



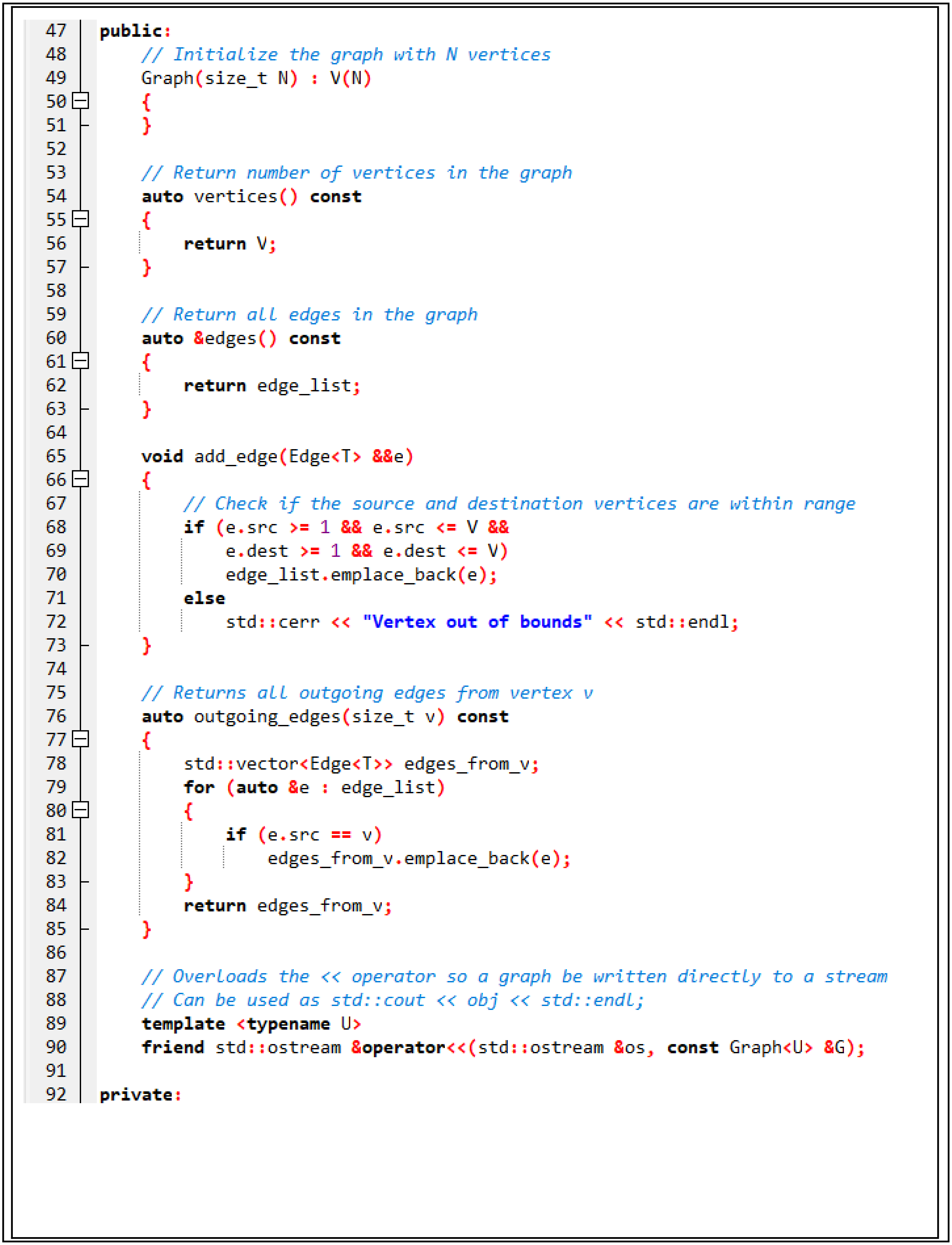
```
Graph adjacency list
(start_vertex, end_vertex, weight):
(0, 2, 4) (0, 1, 2)
(1, 4, 3)
(2, 3, 2)
(3, 1, 4)
(4, 3, 3)


--------------------------------
Process exited after 0.01531 seconds with return value 0
Press any key to continue . . .
```

Code: ILO B: Create C++ code for implementing graph traversal algorithms such as Breadth-First and Depth-First Search

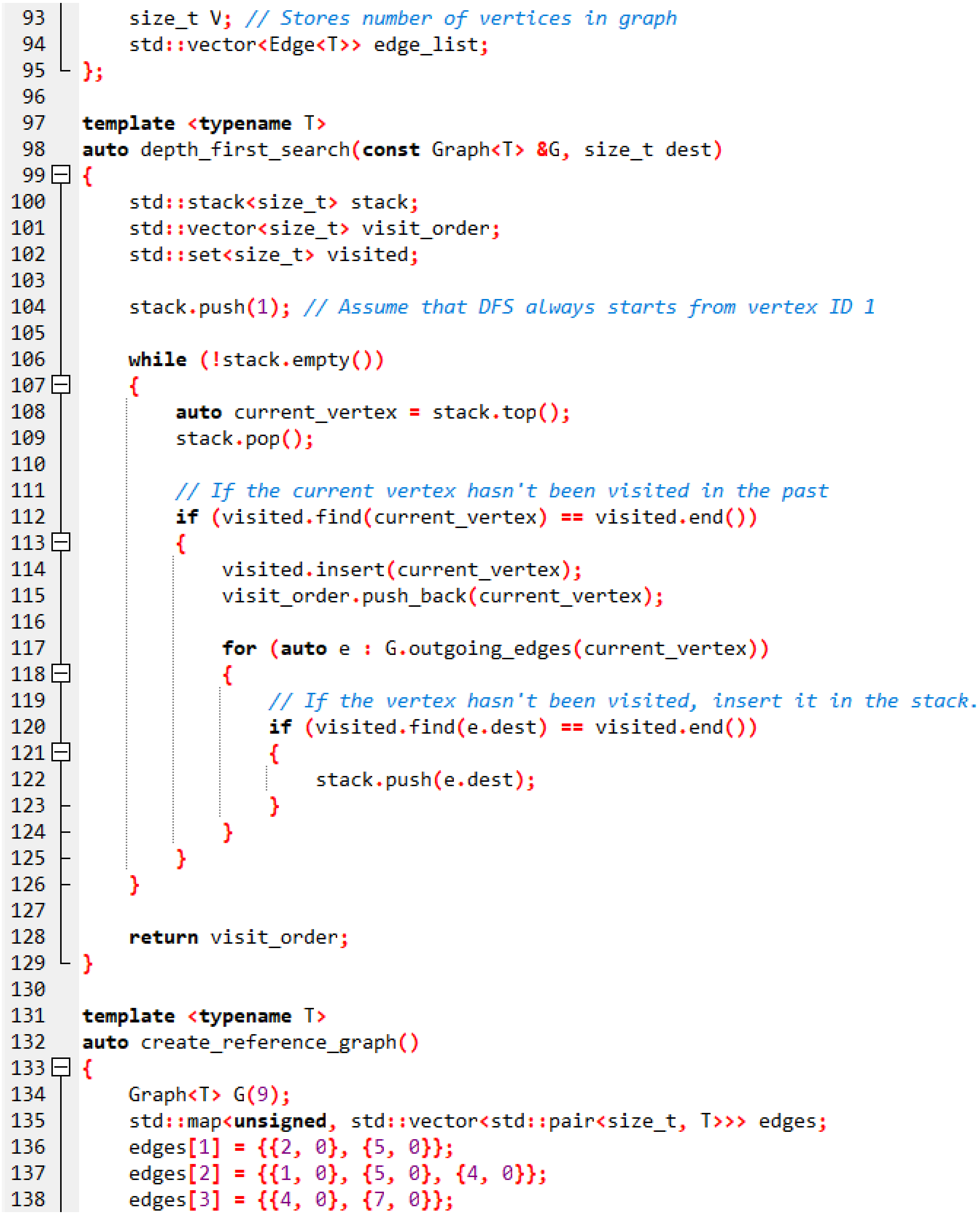
B.1. Depth-First Search



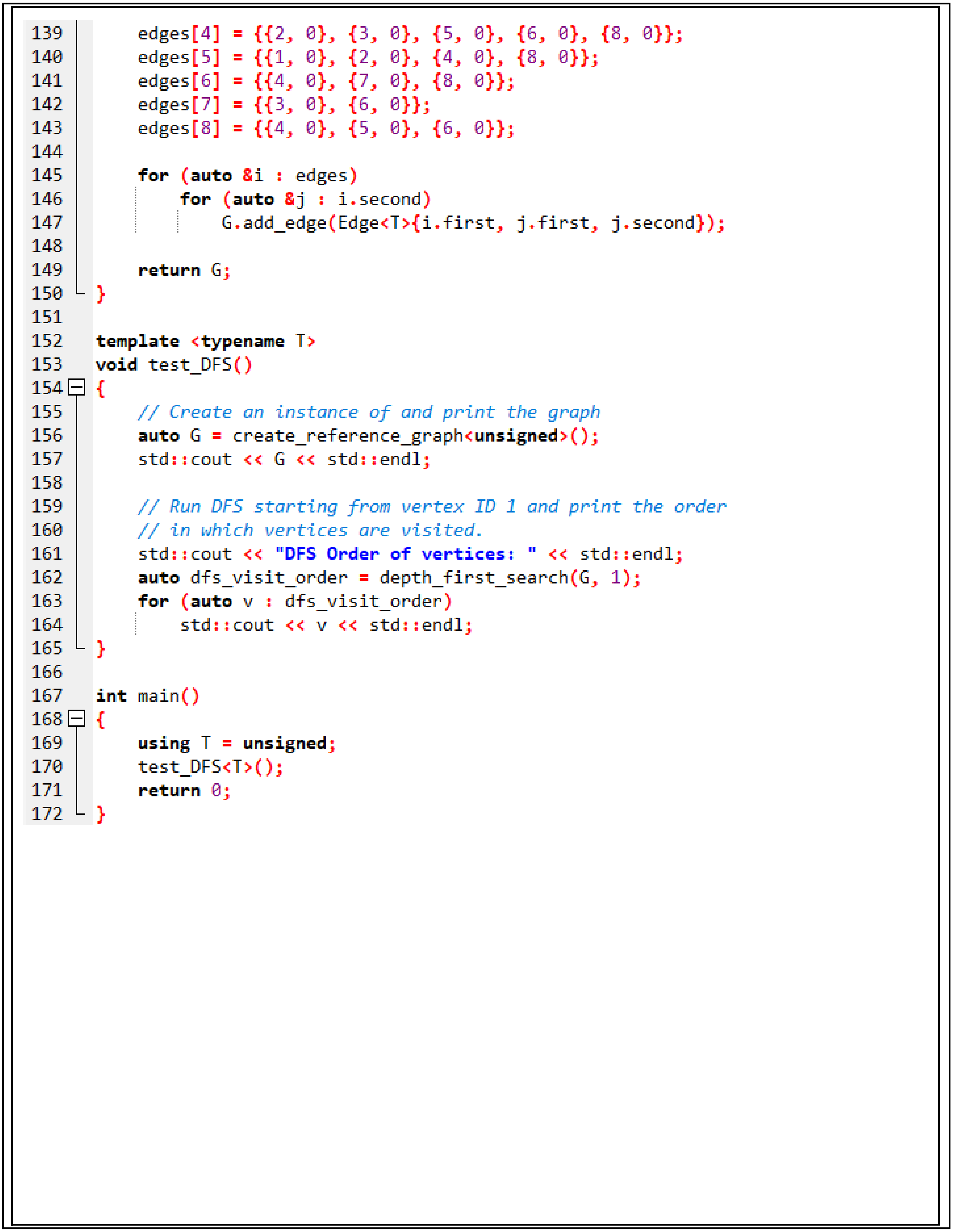
```cpp
#include <string>
#include <vector>
#include <iostream>
#include <set>
#include <map>
#include <stack>

template <typename T>
class Graph;

template <typename T>
struct Edge
{
    size_t src;
    size_t dest;
    T weight;

    // To compare edges, only compare their weights,
    // and not the source/destination vertices
    inline bool operator<(const Edge<T> &e) const
    {
        return this->weight < e.weight;
    }
    inline bool operator>(const Edge<T> &e) const
    {
        return this->weight > e.weight;
    }
};

template <typename T>
std::ostream &operator<<(std::ostream &os, const Graph<T> &G)
{
    for (auto i = 1; i < G.vertices(); i++)
    {
        os << i << ":\t";
        auto edges = G.outgoing_edges(i);
        for (auto &e : edges)
            os << "{" << e.dest << ": " << e.weight << "}, ";
        os << std::endl;
    }
    return os;
}

template <typename T>
class Graph
{
public:
```



```cpp
public:
    // Initialize the graph with N vertices
    Graph(size_t N) : V(N)
    {
    }

    // Return number of vertices in the graph
    auto vertices() const
    {
        return V;
    }

    // Return all edges in the graph
    auto &edges() const
    {
        return edge_list;
    }

    void add_edge(Edge<T> &&e)
    {
        // Check if the source and destination vertices are within range
        if (e.src >= 1 && e.src <= V &&
            e.dest >= 1 && e.dest <= V)
            edge_list.emplace_back(e);
        else
            std::cerr << "Vertex out of bounds" << std::endl;
    }

    // Returns all outgoing edges from vertex v
    auto outgoing_edges(size_t v) const
    {
        std::vector<Edge<T>> edges_from_v;
        for (auto &e : edge_list)
        {
            if (e.src == v)
                edges_from_v.emplace_back(e);
        }
        return edges_from_v;
    }

    // Overloads the << operator so a graph be written directly to a stream
    // Can be used as std::cout << obj << std::endl;
    template <typename U>
    friend std::ostream &operator<<(std::ostream &os, const Graph<U> &G);

private:
```

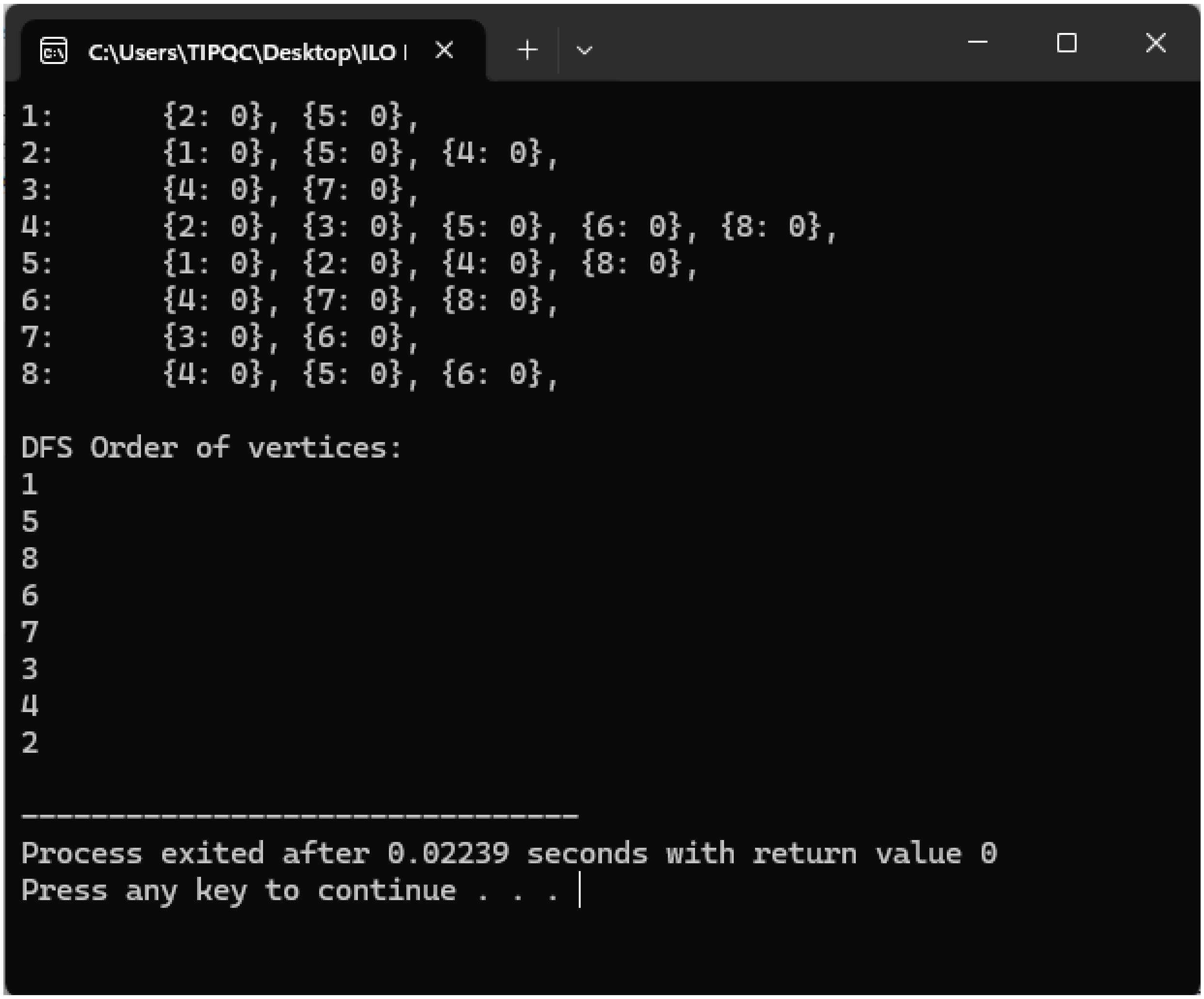


```cpp
    size_t V; // Stores number of vertices in graph
    std::vector<Edge<T>> edge_list;
};

template <typename T>
auto depth_first_search(const Graph<T> &G, size_t dest)
{
    std::stack<size_t> stack;
    std::vector<size_t> visit_order;
    std::set<size_t> visited;

    stack.push(1); // Assume that DFS always starts from vertex ID 1

    while (!stack.empty())
    {
        auto current_vertex = stack.top();
        stack.pop();

        // If the current vertex hasn't been visited in the past
        if (visited.find(current_vertex) == visited.end())
        {
            visited.insert(current_vertex);
            visit_order.push_back(current_vertex);

            for (auto e : G.outgoing_edges(current_vertex))
            {
                // If the vertex hasn't been visited, insert it in the stack.
                if (visited.find(e.dest) == visited.end())
                {
                    stack.push(e.dest);
                }
            }
        }
    }

    return visit_order;
}

template <typename T>
auto create_reference_graph()
{
    Graph<T> G(9);
    std::map<unsigned, std::vector<std::pair<size_t, T>>> edges;
    edges[1] = {{2, 0}, {5, 0}};
    edges[2] = {{1, 0}, {5, 0}, {4, 0}};
    edges[3] = {{4, 0}, {7, 0}};
```



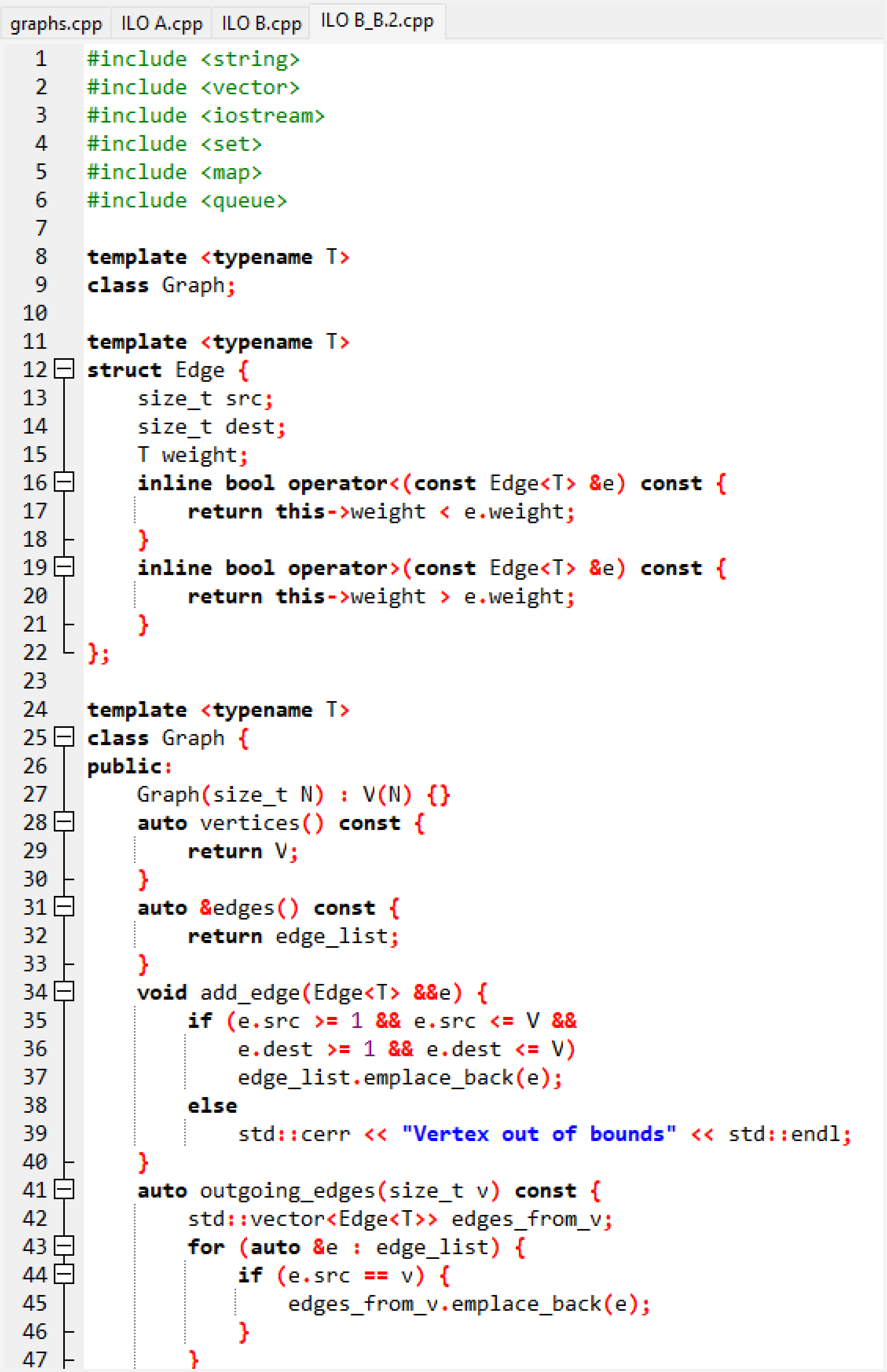
```
    edges[4] = {{2, 0}, {3, 0}, {5, 0}, {6, 0}, {8, 0}};
    edges[5] = {{1, 0}, {2, 0}, {4, 0}, {8, 0}};
    edges[6] = {{4, 0}, {7, 0}, {8, 0}};
    edges[7] = {{3, 0}, {6, 0}};
    edges[8] = {{4, 0}, {5, 0}, {6, 0}};

    for (auto &i : edges)
        for (auto &j : i.second)
            G.add_edge(Edge<T>{i.first, j.first, j.second});

    return G;
}

template <typename T>
void test_DFS()
{
    // Create an instance of and print the graph
    auto G = create_reference_graph<unsigned>();
    std::cout << G << std::endl;

    // Run DFS starting from vertex ID 1 and print the order
    // in which vertices are visited.
    std::cout << "DFS Order of vertices: " << std::endl;
    auto dfs_visit_order = depth_first_search(G, 1);
    for (auto v : dfs_visit_order)
        std::cout << v << std::endl;
}

int main()
{
    using T = unsigned;
    test_DFS<T>();
    return 0;
}
```

Output:

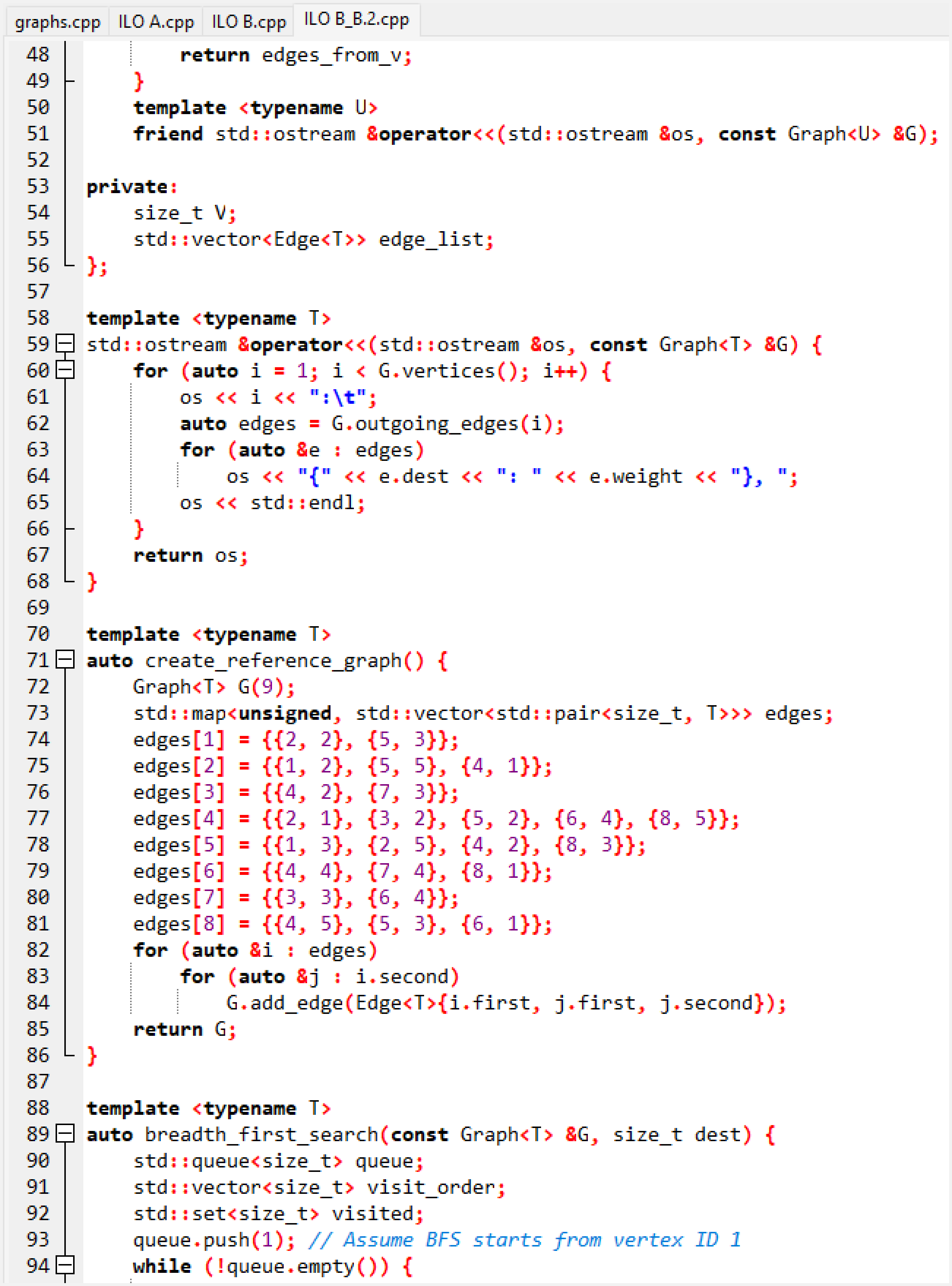


```
1:      {2: 0}, {5: 0},
2:      {1: 0}, {5: 0}, {4: 0},
3:      {4: 0}, {7: 0},
4:      {2: 0}, {3: 0}, {5: 0}, {6: 0}, {8: 0},
5:      {1: 0}, {2: 0}, {4: 0}, {8: 0},
6:      {4: 0}, {7: 0}, {8: 0},
7:      {3: 0}, {6: 0},
8:      {4: 0}, {5: 0}, {6: 0},

DFS Order of vertices:
1
5
8
6
7
3
4
2

--------------------------------
Process exited after 0.02239 seconds with return value 0
Press any key to continue . . .
```

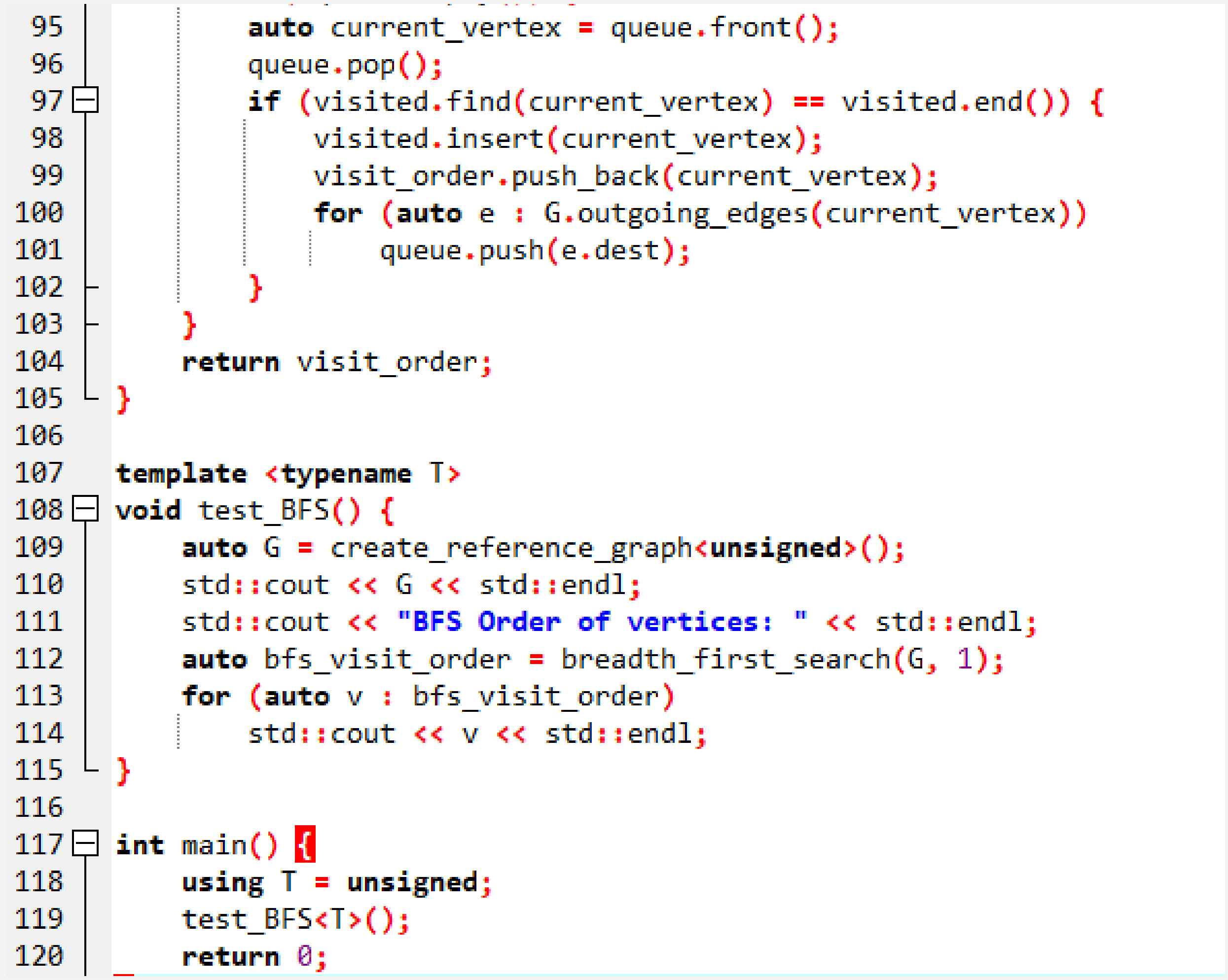
Code: B.2. Breadth-First Search



```cpp
#include <string>
#include <vector>
#include <iostream>
#include <set>
#include <map>
#include <queue>

template <typename T>
class Graph;

template <typename T>
struct Edge {
    size_t src;
    size_t dest;
    T weight;
    inline bool operator<(const Edge<T> &e) const {
        return this->weight < e.weight;
    }
    inline bool operator>(const Edge<T> &e) const {
        return this->weight > e.weight;
    }
};

template <typename T>
class Graph {
public:
    Graph(size_t N) : V(N) {}
    auto vertices() const {
        return V;
    }
    auto &edges() const {
        return edge_list;
    }
    void add_edge(Edge<T> &&e) {
        if (e.src >= 1 && e.src <= V &&
            e.dest >= 1 && e.dest <= V)
            edge_list.emplace_back(e);
        else
            std::cerr << "Vertex out of bounds" << std::endl;
    }
    auto outgoing_edges(size_t v) const {
        std::vector<Edge<T>> edges_from_v;
        for (auto &e : edge_list) {
            if (e.src == v) {
                edges_from_v.emplace_back(e);
            }
        }
```

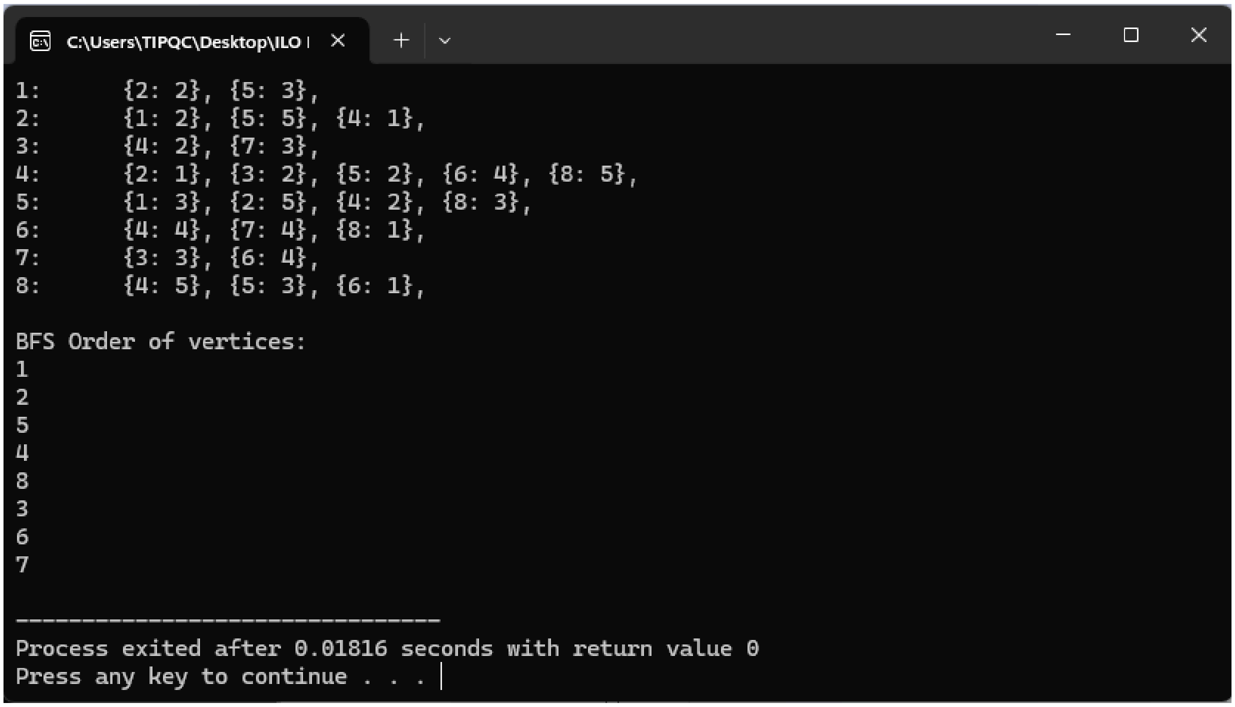


```cpp
        return edges_from_v;
    }
    template <typename U>
    friend std::ostream &operator<<(std::ostream &os, const Graph<U> &G);

private:
    size_t V;
    std::vector<Edge<T>> edge_list;
};

template <typename T>
std::ostream &operator<<(std::ostream &os, const Graph<T> &G) {
    for (auto i = 1; i < G.vertices(); i++) {
        os << i << ":\t";
        auto edges = G.outgoing_edges(i);
        for (auto &e : edges)
            os << "{" << e.dest << ": " << e.weight << "}, ";
        os << std::endl;
    }
    return os;
}

template <typename T>
auto create_reference_graph() {
    Graph<T> G(9);
    std::map<unsigned, std::vector<std::pair<size_t, T>>> edges;
    edges[1] = {{2, 2}, {5, 3}};
    edges[2] = {{1, 2}, {5, 5}, {4, 1}};
    edges[3] = {{4, 2}, {7, 3}};
    edges[4] = {{2, 1}, {3, 2}, {5, 2}, {6, 4}, {8, 5}};
    edges[5] = {{1, 3}, {2, 5}, {4, 2}, {8, 3}};
    edges[6] = {{4, 4}, {7, 4}, {8, 1}};
    edges[7] = {{3, 3}, {6, 4}};
    edges[8] = {{4, 5}, {5, 3}, {6, 1}};
    for (auto &i : edges)
        for (auto &j : i.second)
            G.add_edge(Edge<T>{i.first, j.first, j.second});
    return G;
}

template <typename T>
auto breadth_first_search(const Graph<T> &G, size_t dest) {
    std::queue<size_t> queue;
    std::vector<size_t> visit_order;
    std::set<size_t> visited;
    queue.push(1); // Assume BFS starts from vertex ID 1
    while (!queue.empty()) {
```



```cpp
        auto current_vertex = queue.front();
        queue.pop();
        if (visited.find(current_vertex) == visited.end()) {
            visited.insert(current_vertex);
            visit_order.push_back(current_vertex);
            for (auto e : G.outgoing_edges(current_vertex))
                queue.push(e.dest);
        }
    }
    return visit_order;
}

template <typename T>
void test_BFS() {
    auto G = create_reference_graph<unsigned>();
    std::cout << G << std::endl;
    std::cout << "BFS Order of vertices: " << std::endl;
    auto bfs_visit_order = breadth_first_search(G, 1);
    for (auto v : bfs_visit_order)
        std::cout << v << std::endl;
}

int main() {
    using T = unsigned;
    test_BFS<T>();
    return 0;
```

Output:



```
1:      {2: 2}, {5: 3},
2:      {1: 2}, {5: 5}, {4: 1},
3:      {4: 2}, {7: 3},
4:      {2: 1}, {3: 2}, {5: 2}, {6: 4}, {8: 5},
5:      {1: 3}, {2: 5}, {4: 2}, {8: 3},
6:      {4: 4}, {7: 4}, {8: 1},
7:      {3: 3}, {6: 4},
8:      {4: 5}, {5: 3}, {6: 1},

BFS Order of vertices:
1
2
5
4
8
3
6
7

--------------------------------
Process exited after 0.01816 seconds with return value 0
Press any key to continue . . .
```

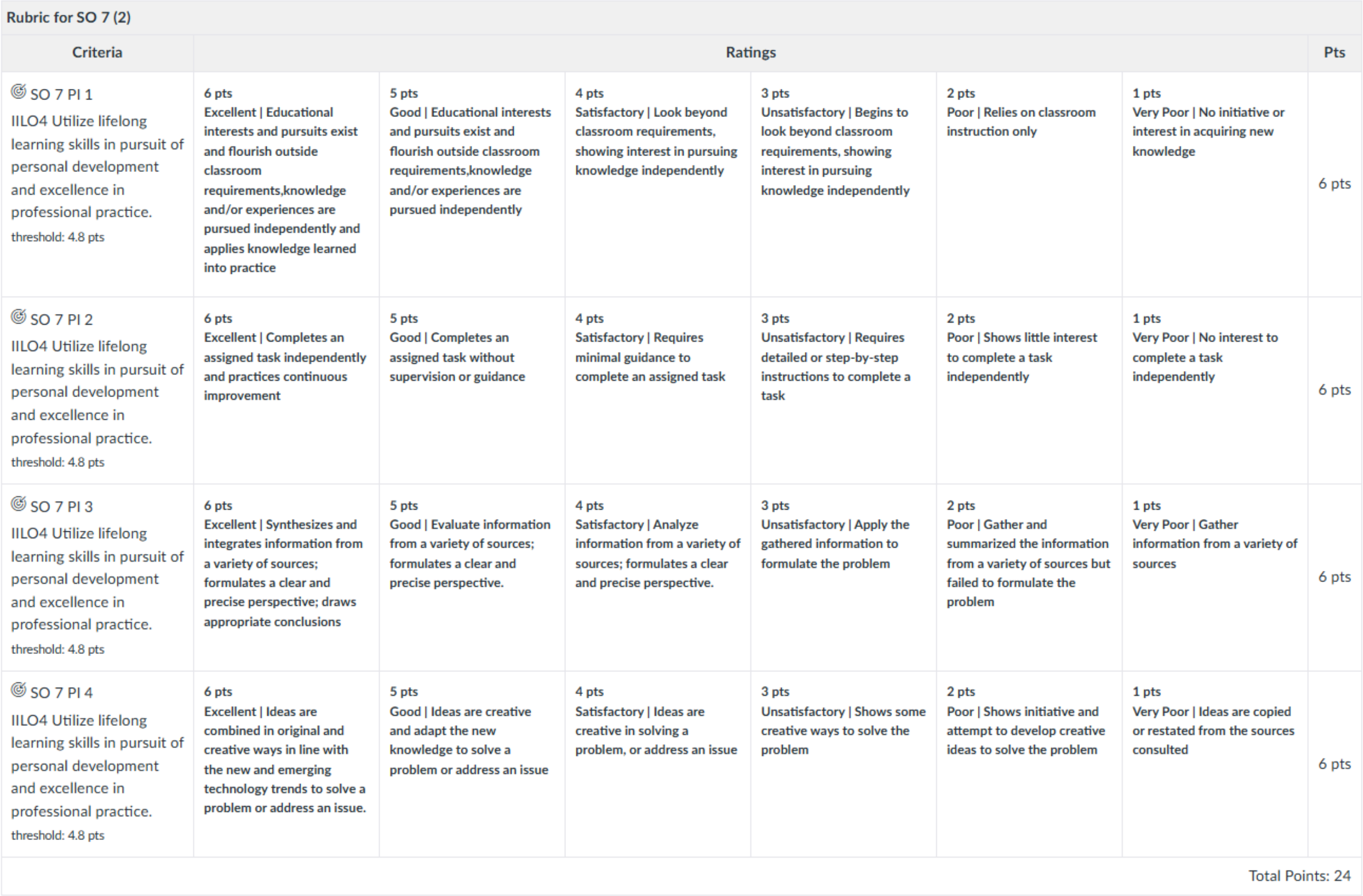
7. Supplementary Activity

ILO C: Demonstrate an understanding of graph implementation, operations and traversal methods.

1. A person wants to visit the

8. Conclusion

9. Assessment Rubric



Rubric for SO 7 (2)

Criteria	Ratings						Pts
SO 7 PI 1 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently and applies knowledge learned into practice	5 pts Good | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently	4 pts Satisfactory | Look beyond classroom requirements, showing interest in pursuing knowledge independently	3 pts Unsatisfactory | Begins to look beyond classroom requirements, showing interest in pursuing knowledge independently	2 pts Poor | Relies on classroom instruction only	1 pts Very Poor | No initiative or interest in acquiring new knowledge	6 pts
SO 7 PI 2 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Completes an assigned task independently and practices continuous improvement	5 pts Good | Completes an assigned task without supervision or guidance	4 pts Satisfactory | Requires minimal guidance to complete an assigned task	3 pts Unsatisfactory | Requires detailed or step-by-step instructions to complete a task	2 pts Poor | Shows little interest to complete a task independently	1 pts Very Poor | No interest to complete a task independently	6 pts
SO 7 PI 3 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Synthesizes and integrates information from a variety of sources; formulates a clear and precise perspective; draws appropriate conclusions	5 pts Good | Evaluate information from a variety of sources; formulates a clear and precise perspective.	4 pts Satisfactory | Analyze information from a variety of sources; formulates a clear and precise perspective.	3 pts Unsatisfactory | Apply the gathered information to formulate the problem	2 pts Poor | Gather and summarized the information from a variety of sources but failed to formulate the problem	1 pts Very Poor | Gather information from a variety of sources	6 pts
SO 7 PI 4 IILO4 Utilize lifelong learning skills in pursuit of personal development and excellence in professional practice. threshold: 4.8 pts	6 pts Excellent | Ideas are combined in original and creative ways in line with the new and emerging technology trends to solve a problem or address an issue.	5 pts Good | Ideas are creative and adapt the new knowledge to solve a problem or address an issue	4 pts Satisfactory | Ideas are creative in solving a problem, or address an issue	3 pts Unsatisfactory | Shows some creative ways to solve the problem	2 pts Poor | Shows initiative and attempt to develop creative ideas to solve the problem	1 pts Very Poor | Ideas are copied or restated from the sources consulted	6 pts

Total Points: 24