Implementación Grafos

List.h

```
#ifndef GRAPH_LIST_H
#define GRAPH_LIST_H

#include <iostream>
#include <cassert>

using namespace std;

template<typename T>
struct Node{
   T data;
   Node<T>* next;
};

template<typename T>
class List {
private:
   Node<T>* begin;
   int count;
   Node<T>* makeNode(const T& value);
public:
   List();
   ~List();
   void insert(int pos, const T& value);
   void erase(int pos);
   T& get(int pos) const;
   void print() const;
   int size() const;
   Node<T>* search(const T& value);
};


#endif //GRAPH_LIST_H
```

List.cpp

```
#ifndef GRAPH_LIST_CPP
#define GRAPH_LIST_CPP

#include "List.h"


template<typename T>
List<T>:: List(): begin(0), count(0){

}
template<typename T>
List<T>:: ~List(){
   Node<T>* del = begin;
   while (begin){
       begin = begin->next;
       delete del;
```

```
        del = begin;
    }
}
template<typename T>
Node<T>* List<T>::makeNode(const T &value) {
    Node<T>* temp = new Node<T>;
    temp->data = value;
    temp->next = 0;
    return temp;
}
template<typename T>
void List<T>::insert(int pos, const T &value) {
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    Node<T>* add = makeNode(value);
    if(pos == 0){
        add->next = begin;
        begin = add;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        add->next = cur->next;
        cur->next = add;
    }
    count++;
}

template<typename T>
void List<T>::erase(int pos) {
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    if(pos == 0){
        Node<T>* del = begin;
        begin = begin->next;
        delete del;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        Node<T>* del = cur->next;
        cur->next = del->next;
        delete del;
    }
    count--;
}
template<typename T>
T& List<T>::get(int pos) const{
    if(pos < 0 || pos>count-1){
        cout << "Error! The position is out of range." << endl;
        assert(false);
    }
    if(pos == 0){
        return begin->data;
```

```
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos; i++){
            cur = cur->next;
        }
        return cur->data;
    }
}
template<typename T>
void List<T>::print() const{
    if(count == 0){
        cout << "List is empty." << endl;
        return;
    }
    Node<T>* cur = begin;
    while(cur){
        cout << cur->data;
        cur = cur->next;
    }
}
template<typename T>
int List<T>::size() const {
    return count;
}

template<typename T>
Node<T> *List<T>::search(const T &value) {
    Node<T>* cur = begin;
    while(cur){
        if(cur == value) return cur;
        cur = cur->next;
    }
}

#endif
```

Graph.h

```
#ifndef GRAPH_GRAPH_H
#define GRAPH_GRAPH_H

#include <iostream>
#include "List.cpp"

using namespace std;

template<class T>
class Edge;
template<class T>
class Vertex;

template<class T>
class Edge{
public:
    Vertex<T>* to;
    int weight;
```

```
    friend ostream &operator<<(ostream &out, Edge<T>* edge) {
        out << "To: " << edge->to->data << ", Weight: " << edge->weight <<
endl;
        return out;
    }
};

template<class T>
class Vertex{
public:
    T data;
    int inDegree;
    int outDegree;
    List<Edge<T>*> connectedTo;
    Vertex(const T& value);
    ~Vertex();
    void addNeighbor(Vertex<T>* to, int weight=0);
    int getWeight(const T& value);
    friend ostream &operator<<(ostream &out, Vertex<T>* vertex) {
        out << vertex->data << endl;
        out << "In degree: " << vertex->inDegree << endl;
        out << "out degree: " << vertex->outDegree << endl;
        out << "Edges: " << endl;
        vertex->connectedTo.print();
        return out;
    }
};

template<class T>
class Graph {
public:
    int count;
    List<Vertex<T>*> vertexList;
    Graph();
    ~Graph();
    Vertex<T>* addVertex(const T& value);
    Vertex<T>* getVertex(const T& value);
    void addEdge(const T& from, const T& to, int weight=0);
};

#endif //GRAPH_GRAPH_H
```

Graph.cpp

```
#include "Graph.h"

template<class T>
Vertex<T>::Vertex(const T& value) {
    data = value;
    inDegree = 0;
    outDegree = 0;
    connectedTo = {};
}

template<class T>
Vertex<T>::~Vertex() {
```

```
}

template<class T>
void Vertex<T>::addNeighbor(Vertex<T> *to, int weight) {
   Edge<T>* temp = new Edge<T>;
   temp->to = to;
   temp->weight = weight;

   outDegree++;
   to->inDegree++;

   connectedTo.insert(connectedTo.size(), temp);
}

template<class T>
int Vertex<T>::getWeight(const T &value) {
   for(int i=0; i < connectedTo.size(); i++){
       Edge<T>* temp = connectedTo.get(i);
       if(temp->to->data == value){
           return connectedTo.get(i)->weight;
       }
   }
   return NULL;
}

template<class T>
Graph<T>::Graph() {
   count = 0;
   vertexList = {};
}

template<class T>
Graph<T>::~Graph() {
}

template<class T>
Vertex<T>* Graph<T>::addVertex(const T &value) {
   Vertex<T>* newVertex = new Vertex<T>(value);
   vertexList.insert(vertexList.size(), newVertex);
   count++;
   return newVertex;
}

template<class T>
void Graph<T>::addEdge(const T& from, const T& to, int weight) {
   Vertex<T>* fromVertex = getVertex(from);
   if(!fromVertex){
       fromVertex = addVertex(from);
   }
   Vertex<T>* toVertex = getVertex(to);
   if(!toVertex){
       toVertex = addVertex(to);
   }
   fromVertex->addNeighbor(toVertex, weight);
}

template<class T>
Vertex<T> *Graph<T>::getVertex(const T &value) {
   for(int i=0; i < vertexList.size();i++ ){
       if(vertexList.get(i)->data == value) return vertexList.get(i);
```

```
    }
    return NULL;
}
```

main.cpp

```
#include <iostream>
#include "Graph.cpp"

using namespace std;
int main() {
    Graph<int> g;
    for(int i=0; i < 6; i++){
        g.addVertex(i);
    }

    g.addEdge(0,1,5);
    g.addEdge(0,5,2);
    g.addEdge(1,2,4);
    g.addEdge(2,3,9);
    g.addEdge(3,4,7);
    g.addEdge(3,5,3);
    g.addEdge(4,0,1);
    g.addEdge(5,4,8);
    g.addEdge(5,2,1);

    for(int vertexPos=0; vertexPos < g.vertexList.size(); vertexPos++ ){
        Vertex<int>* vertex = g.vertexList.get(vertexPos);
        for(int edgePos=0; edgePos < vertex->connectedTo.size(); edgePos++){
            Edge<int>* edge = vertex->connectedTo.get(edgePos);
            cout << "(" << vertex->data << ", " << edge->to->data << ", " <<
edge->weight << ")" << endl;
        }
    }

    cout << "Weight of Vertex 3 -> 5: " <<g.getVertex(3)->getWeight(5) <<
endl;

    return 0;
}
```