```cpp
#include "Edge.hpp"
// Insieme di tutti gli oggetti Point2D usati da tutti gli oggetti Edge
std::set<Point2D> Edge::Points;

// Costruttore
Edge::Edge(const Point2D& P1, const Point2D& P2)
{
    if(P1 == P2) throw std::invalid_argument("Error! Trying to create an edge with coincident vertices");
    A = Points.insert(P1).first;
    B = Points.insert(P2).first;
    std::cout << "Costruisco Edge " << *this << "\n";
}

// Distruttore
Edge::~Edge()
{
    std::cout << "Distruggo Edge " << *this << "\n";
}

// Costuttore di copia
Edge::Edge(const Edge& other)
{
    std::cout << "Copio un Edge "<< other << "\n";
    A = other.A;
    B = other.B;
}

Edge& Edge::operator=(const Edge& other)
{
    std::cout << "Copio un Edge tramite assignement operator "<< other << "\n";
    A = other.A;
    B = other.B;
    return *this;
}

// Metodi per l'accesso alle coordinate degli estremi
Point2D Edge::getA() const
{
    return *A;
}
Point2D Edge::getB() const
{
    return *B;
}

// Metodi per il calcolo della lunghezza del lato
double Edge::length() const
{
    return (*A-*B).norm();
}

```

```
52 // Metodi di confronto
53 bool Edge::Connected(const Edge& E1, const Edge& E2)
54 {
55     if (E1.A == E2.A) return true;
56     if (E1.A == E2.B) return true;
57     if (E1.B == E2.A) return true;
58     if (E1.B == E2.B) return true;
59     return false;
60 
61 }
62 bool operator==(const Edge& E1, const Edge& E2)
63 {
64     if ((E1.A == E2.A) and (E1.B == E2.B)) return true;
65     if ((E1.A == E2.B) and (E1.B == E2.A)) return true;
66     return false;
67 }
68 bool operator<(const Edge& E1, const Edge& E2)
69 {
70     if (*E1.A < *E2.A) return true;
71     if (*E1.B < *E2.B) return true;
72     return false;
73 }
```