DCEL

Generated by Doxygen 1.8.18

5 File Documentation

Chapter 1

Main Page

1.1 Overview

This program deals with building a DCEL data strutcure to store any type of planar graph given in the input.
How it works
A simple overview of the construction of DCEL:
Step 1: For each Edge, create two new HalfEdges, assign their twins and put them in corresponding source/tail vertex lists
Step 2: Sort all the HalfEdges present at each Vertex in clockwise order
Step 3: For Every pair of HalfEdges next to each other on a Vertex, assign their next and prev pointers
Step 4: Taking each HalfEdge, if its face points to nullptr, create new face and assign to all HalfEdges which can be obtained by traversing the next pointer. Set the represntative of Face object to any of theese halfedges and add this face to the list of Faces present in the DCEL

Steps to Compile and Run :
1) *cd* into the src directory
2) Run *g++ main.cpp* which generates an executable called *a.out* in the same directory
3) Run the executable using *./a.out* (on linux)
3.1) The executable takes a dataset from command line argument. For example, to use an existing dataset, run *./a.out ../datasets/1sq.txt*
3.2) If no command-line argument is given, it takes input from the shell directly (stdin)

Performance of the algorithm is documented in the report

1.1.1 Input

Input is given as follows:

- Input can be given both from file (via command-line args) or stdin

- First line must contain the no of Edges to be taken as input by the program.

- Each of next line must contain 4 integers, space seperated denoting the (x1, y1), (x2, y2) coordinates of endpoints of each edge.

- Each coordinate must be of integer type in the range -10^8 to 10^8.

- Number of coordinates must be less than 1 Billion.

1.1.2 Output

- The program prints all the Faces, Edges, HalfEdges which are incident and present in the DCEL structure to accurately show the representation of planar graph

1.1.3 Author

The algorithm is implemented and documented by **Rikil Gajarla (2017A7PS0202H)**.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 DCEL Class Reference

The main class for the DCEL data structure.
`#include <DCEL.h>`

Public Member Functions

- DCEL (vector< Edge > &edgelist)

  Construct a new DCEL object form the input list of edges.
- void setFace (HalfEdge ∗(&he))

  Set a new Face object for the given HalfEdge.
- Vertex ∗ getVertex (const Vertex &v)

  Get the pointer to Vertex object which matches/equals the given vertex.
- Vertex ∗ addVertex (Vertex ∗v)

  Add a new Vertex object to the existing DCEL.
- Vertex ∗ addVertex (Vertex v)

  Add a new Vertex object to the existing DCEL.
- Vertex ∗ addVertex (long x, long y)

  Add a new Vertex object to the existing DCEL.
- void addEdge (Vertex ∗a, Vertex ∗b)

  Add a new Edge to the existing DCEL data structure.
- void printBoundaryEdges (Face ∗f)

  Print the directional halfedges which surround the face.
- ∼DCEL ()

  Destroy the DCEL object.

Public Attributes

- map< Vertex ∗, vector< HalfEdge ∗ > > heMap
- vector< Vertex ∗ > vertices
- vector< Face ∗ > faces

4.1.1 Detailed Description

The main class for the DCEL data structure.
This is the main DCEL data structure. All basic geometric structures are created dynamically and stored as pointers so that all structures can refer to each other without extra storage.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DCEL()

```
DCEL::DCEL (
              vector< Edge > & edgelist )  [inline]
```

Construct a new DCEL object form the input list of edges.

Parameters

edgelist	a LIst of Edge objects which are used to build the DCEL structure

4.1.2.2 ∼DCEL()

```
DCEL::∼DCEL ( )  [inline]
```

Destroy the DCEL object.

This function deletes the DCEL data structure and all the dynamically allocated objects present in this structure.

4.1.3 Member Function Documentation

4.1.3.1 addEdge()

```
void DCEL::addEdge (
              Vertex * a,
              Vertex * b )  [inline]
```

Add a new Edge to the existing DCEL data structure.

This method dynamically creates two new HalfEdge objects representing the input Edge and sets all their attributes respectly. This method takes care to also create and assign new Faces (if formed) automatically.

Parameters

a	Vertex endpoint of the edge to be added
b	Vertex endpoint of the edge to be added

4.1.3.2 addVertex() [1/3]

```
Vertex* DCEL::addVertex (
              long x,
              long y )  [inline]
```

Add a new Vertex object to the existing DCEL.

Overloaded version of addVertex() function which takes coordinates vertex. This method builds a new Vertex object dynamically to store in DCEL

Parameters

x	x coordinate of the new vertex object
y	y coordinate of the new vertex object

Returns

Vertex∗ pointer to the added object in the DCEL

4.1.3.3 addVertex() [2/3]

```
Vertex* DCEL::addVertex (
             Vertex * v )  [inline]
```

Add a new Vertex object to the existing DCEL.
Addition of Vertex is only done if there is no similar/equal vertex existing in DCEL

Parameters

v	Pointer to Vertex object to be added

Returns

Vertex∗ pointer to the added object in the DCEL

4.1.3.4 addVertex() [3/3]

```
Vertex* DCEL::addVertex (
             Vertex v )  [inline]
```

Add a new Vertex object to the existing DCEL.
Overloaded version of addVertex() function which takes a Vertex object instead of a pointer. This method builds a new Vertex object dynamically to store in DCEL

Parameters

v	Vertex object to be added

Returns

Vertex∗ pointer to the added object in the DCEL

4.1.3.5 getVertex()

```
Vertex* DCEL::getVertex (
             const Vertex & v )  [inline]
```

Get the pointer to Vertex object which matches/equals the given vertex.

Parameters

v	Vertex object to which we need matching object in DCEL

Returns

Vertex∗ pointer to matching Vertex object in DCEL. Returns nullptr in case of no matching vertex

4.1.3.6 printBoundaryEdges()

```
void DCEL::printBoundaryEdges (
             Face * f )  [inline]
```

Print the directional halfedges which surround the face.

Parameters

f	Pointer to face object

4.1.3.7 setFace()

```
void DCEL::setFace (
            HalfEdge *& he )  [inline]
```

Set a new Face object for the given HalfEdge.

In case of new added HalfEdges, this function creates a face, traverses all the connected half edges and assigns the new face to all of them. Make sure that the input halfedge has no face assigned, else it will be lost. This function assigns the input halfedge as the representative of the new face

Parameters

he	HalfEdge to which new face must be created ans assigned

4.1.4 Member Data Documentation

4.1.4.1 faces

```
vector<Face *> DCEL::faces
```

List of pointers to all abailable faces

4.1.4.2 heMap

```
map<Vertex *, vector<HalfEdge *> > DCEL::heMap
```

Hash Map to store list of HalfEdge objects incident at each vertex

4.1.4.3 vertices

```
vector<Vertex *> DCEL::vertices
```

List of pointer to all available vertices

The documentation for this class was generated from the following file:

- DCEL.h

4.2 Edge Class Reference

Basic geometric Edge class to store endpoints of edges.

```
#include <Geometry.h>
```

Public Member Functions

- Edge ()

  Construct a new empty Edge object.
- Edge (Vertex &source, Vertex &destination)

  Construct a new Edge object from given vertices.
- Edge (double x1, double y1, double x2, double y2)

  Construct a new Edge object from vertex coordinates.

Public Attributes

- Vertex **src**
- Vertex **dst**

4.2.1 Detailed Description

Basic geometric Edge class to store endpoints of edges.
This class stores the 2 Vertex type objects to represent an edge. Directionality is NOT assumed. For the case of directionality, HalfEdge takes care of it.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Edge() [1/3]

```
Edge::Edge ( )  [inline]
```

Construct a new empty Edge object.

4.2.2.2 Edge() [2/3]

```
Edge::Edge (
            Vertex & source,
            Vertex & destination )  [inline]
```

Construct a new Edge object from given vertices.
To construct a Edge type object from 2 Vertex object. The parameter names might be misleading, but it is guarenteed that no directionality is assumed

Parameters

source	First Vertex object
destination	Second Vertex object

4.2.2.3 Edge() [3/3]

```
Edge::Edge (
            double x1,
            double y1,
            double x2,
            double y2 )  [inline]
```

Construct a new Edge object from vertex coordinates.

Parameters

x1	x coordinate of first vertex
y1	y coordinate of first vertex
x2	x coordinate of second vertex
y2	y coordinate of second vertex

The documentation for this class was generated from the following file:

- Geometry.h

4.3 Face Class Reference

Basic geometric Face class to store a Face structure.

```
#include <Geometry.h>
```

Public Member Functions

- Face ()

  Construct a new empty Face object.
- Face (long long Id, HalfEdge ∗representative)

  Construct a new Face object.

Public Attributes

- long long **id**
- HalfEdge ∗ **rep**

4.3.1 Detailed Description

Basic geometric Face class to store a Face structure.
This Face class has an id and a representative HalfEdge which can be traversed to get the surrounding vertices and edges.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Face() [1/2]

```
Face::Face ( )  [inline]
```

Construct a new empty Face object.

4.3.2.2 Face() [2/2]

```
Face::Face (
             long long Id,
             HalfEdge ∗ representative )  [inline]
```

Construct a new Face object.

Parameters

Id	id to be given to this face object (must be unique)
representative	A HalfEdge to represent this face (can be any of the incident half edges)

The documentation for this class was generated from the following file:

- Geometry.h

4.4 HalfEdge Class Reference

Basic geometric Face class to store a Face structure.

```
#include <Geometry.h>
```

Public Member Functions

- HalfEdge (const Vertex ∗src, const Vertex ∗dst)

  Construct a new Half Edge object.

Public Attributes

- HalfEdge ∗ prev

- HalfEdge ∗ next
- HalfEdge ∗ twin
- const Vertex ∗ tail
- const Vertex ∗ head
- Face ∗ face
- double **angle**

4.4.1 Detailed Description

Basic geometric Face class to store a Face structure.
Each Edge will be split into 2 HalfEdges and hence, these are directional. Each HalfEdge object has a next pointer, prev pointer, twin pointer, tail pointer, head pointer and face pointer to give a complete information about all its incident geometric objects. This class is the main component of the DCEL structure.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 HalfEdge()

```
HalfEdge::HalfEdge (
             const Vertex * src,
             const Vertex * dst )  [inline]
```

Construct a new Half Edge object.
Construct a HalfEdge object from the given 2 vertices. Note that directionality is important. First argument refers to the src/source/tail whereas the second argument refers to the dst/destination/head.

Parameters

src	source or the tail Vertex of the HalfEdge
dst	destination or the head Vertex of the HalfEdge

4.4.3 Member Data Documentation

4.4.3.1 face

```
Face* HalfEdge::face
```

prev->face == face && next->face == face

4.4.3.2 head

```
const Vertex* HalfEdge::head
```

Vertex present at head

4.4.3.3 next

```
HalfEdge* HalfEdge::next
```

Pointer to next HalfEdge such that: next->prev = this

4.4.3.4 prev

```
HalfEdge* HalfEdge::prev
```

Pointer to previous HalfEdge such that: prev->next = this

4.4.3.5 tail

`const Vertex* HalfEdge::tail`

Vertex present at tail such that: twin->next->tail == tail && prev->twin->tail == tail

4.4.3.6 twin

`HalfEdge* HalfEdge::twin`

Pointer to the other twin of this edge such that: twin->twin = this

The documentation for this class was generated from the following file:

- Geometry.h

4.5 Vertex Class Reference

Basic geometric Vertex class to store coordinates of vertex and halfedge.

`#include <Geometry.h>`

Public Member Functions

- Vertex ()

  Construct a new empty Vertex object.
- Vertex (double X, double Y)

  Construct a new Vertex object with the given coordinates.
- Vertex (const Vertex &p)

  Construct a new Vertex object from another Vertex object.
- bool operator== (const Vertex &v)

  To overload == operator on Vertex type objects.
- bool operator< (const Vertex &v) const

  To overload < operator on Vertex type objects.

Public Attributes

- double **x**
- double **y**
- HalfEdge ∗ **rep**

4.5.1 Detailed Description

Basic geometric Vertex class to store coordinates of vertex and halfedge.

This class stores the (x, y) coordinate of a vertex as taken from the user input. It also has a HalfEdge object which stores a representative HalfEdge for this vertex

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Vertex() [1/3]

`Vertex::Vertex ( ) [inline]`

Construct a new empty Vertex object.

4.5.2.2 Vertex() [2/3]

```
Vertex::Vertex (
                double X,
                double Y )  [inline]
```

Construct a new Vertex object with the given coordinates.
A vertex type object will be constructed with the given coordinates. The pointer to representative HalfEdge will be set to nullptr

Parameters

X	x coordinate of the input vertex
Y	y coordinate of the input vertex

4.5.2.3 Vertex() [3/3]

```
Vertex::Vertex (
                const Vertex & p )  [inline]
```

Construct a new Vertex object from another Vertex object.
This is effectively a copy constructor for the new Vertex object

Parameters

p	Vertex type object whose all parameters will be copied

4.5.3 Member Function Documentation

4.5.3.1 operator<()

```
bool Vertex::operator< (
                const Vertex & v ) const  [inline]
```

To overload < operator on Vertex type objects.
Vertex a is less than Vertex b (a < b) if x coordinate of a is lesser than b. In case x coordinate is equal, their y coordinates are compared.

Parameters

v	Vertex object to be compared with

Returns

true if current x coordinate is less than the other (y is used if x is same)

false otherwise

4.5.3.2 operator==()

```
bool Vertex::operator== (
                const Vertex & v )  [inline]
```

To overload == operator on Vertex type objects.
This returns true only if both x and y coordinates are equal between the objects being compared. Representative edge is not considered

Parameters

v	Vertex object to be compared with

Returns

true if both coordinates match

false if at least one coordinate differs

The documentation for this class was generated from the following file:

- Geometry.h

Chapter 5

File Documentation

5.1 Geometry.h File Reference

This file contains the basic geometric structure such as Vertex, Edge, Face and HalfEdge.

```
#include <iostream>
#include <cmath>
```

Classes

- class Vertex

  Basic geometric Vertex class to store coordinates of vertex and halfedge.
- class Edge

  Basic geometric Edge class to store endpoints of edges.
- class Face

  Basic geometric Face class to store a Face structure.
- class HalfEdge

  Basic geometric Face class to store a Face structure.

Functions

- ostream & **operator**<< (ostream &os, const HalfEdge &he)
- ostream & **operator**<< (ostream &os, const Vertex &v)
- ostream & **operator**<< (ostream &os, const Edge &e)
- ostream & **operator**<< (ostream &os, const Face &f)

Variables

- const double **epsillion** = 0.00001

5.1.1 Detailed Description

This file contains the basic geometric structure such as Vertex, Edge, Face and HalfEdge.

Author

Rikil Gajarla (f20170202@hyderabad.bits-pilani.ac.in)

5.2 main.cpp File Reference

This file is the main runner function which initializes the DCEL.

```
#include <vector>
#include <chrono>
```

```
#include <iostream>
#include "DCEL.h"
#include "Tools.h"
```

Functions

- int main (int argc, char *argv[ ])

  main runner function to run DCEL

5.2.1 Detailed Description

This file is the main runner function which initializes the DCEL.

Author

Rikil Gajarla (f20170202@hyderabad.bits-pilani.ac.in)

5.2.2 Function Documentation

5.2.2.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

main runner function to run DCEL

Parameters

argc	number of command line arguments, supplied by os
argv	command line arguments given by the user, supplied by the os

Returns

int returns 0 on successful execution

5.3 Tools.h File Reference

This file provides helper functions for reading input from the user.

```
#include <vector>
#include <fstream>
#include <sstream>
#include <iostream>
#include "DCEL.h"
```

Functions

- int orient (const Vertex &p, const Vertex &q, const Vertex &r)

  Return the clockwise or anti clockwise orientation of points.
- void **log** (string s)
- vector< Edge > readEdges (int argc, char *argv[ ])

  Helper to get input from the user.

5.3.1 Detailed Description

This file provides helper functions for reading input from the user.

Author

> Rikil Gajarla (f20170202@hyderabad.bits-pilani.ac.in) This file provides the readPoints() function which helps to read file and take input from it. In case any file is not provided, It tries to take input from the user via stdin.

5.3.2 Function Documentation

5.3.2.1 orient()

```
int orient (
            const Vertex & p,
            const Vertex & q,
            const Vertex & r )
```

Return the clockwise or anti clockwise orientation of points.

Parameters

p	first Vertex object
q	second Vertex object
r	third Vertex object

Returns

> int -1 if clockwise, +1 if anti-clockwise and 0 if colinear points

5.3.2.2 readEdges()

```
vector<Edge> readEdges (
            int argc,
            char * argv[] )
```

Helper to get input from the user.

If file path is given in command line argument, this function opens that file and reads input from it. In case no command line argument is given, It tries to take input from user (via stdin)

Parameters

argc	Number of command line arguments, to be passed from the main function
argv	Command line arguments given by the user, to be passed from the main function

Returns

> vector<Edge> List of edges of present in the planar map, taken from user/file input

Index