My Project

Generated by Doxygen 1.8.15

Chapter 1

Chan's Algorithm

This project aims to implement and test the famous Timothy Chan's algorithm for Convex Hull computation.

Introduction

Convex Hull

A convex hull of a given set of points is a minimal convex polygon which contains all the points within it.

Related Algorithms

Convex Hull computation is a very well studied and important problem in computational geometry. It has many real world applications which makes worthy to be sought after.

Some of the other well known algorithms for convex hull are:

1. Jarvis March (Gift wrapping) algorithm | `O(n * h)`
2. Graham Scan algorithm | `O(n log n)`

Here, `n` is the number of points in the input and `h` is the number of points in the output convex hull.

Algorithm & Complexity

Chan's algorithm is inspired by both `Graham Scan` and `Jarvis March`. It uses the ideas from both the algorithms to acheive the asymptotically optimal time complexity of `O(n log h)` where `n` is the number of points in the input and `h` is the number of points on the output convex hull. Here, since the complexity depends on the output hence, it is a output sensitive algorithm. Note, `h <= n` Hence, Chan's algorithm is atleast as good as `Graham Scan`.

Moreover, if `h \in o(n)` then Chan's algorithms is asymptotically better than `Graham Scan`.

How to run

Compilation

To compile the programs use the corresponding make command:

1. Chan's algorithm : `make chan`
2. Graham Scan algorithm : `make graham`
3. Jarvis March : `make jarvis`

Execution

Execute the code by giving an input via console or through input file.

```
./main_chan < ./tests/inputs/t0.txt
```

Run tests

1. Compile tester code `make test_fixed_h`
2. Execute `./test_fixed_h n h numTests` For example: `./test_fixed_h 10000 10 100`

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 files with brief descriptions:

Chapter 4

Class Documentation

4.1 Chan< T > Class Template Reference

`#include <Chan.hpp>`

Public Member Functions

- Chan (vector< Point< T >> &points)

  Construct a new Chan object.
- vector< Point< T > > getConvexHull ()

  Get convex hull computed using Chan's algorithm.

Private Member Functions

- bool restrictedConvexHull (int guessedHullSize)

  Tries to compute convex hull using given partition size (guessed size of hull)
- vector< vector< Point< T > > > computePartitions (int partitionSize)

  Divide input points equally into partitions of size 'partitionSize'.

Private Attributes

- vector< Point< T > > points

  Input points for computing convex hull.
- vector< Point< T > > convexHull

  Set of points in the convex hull in counter clockwise order.
- int numPoints

  Number of points in the input.
- int convexHullSize

  Convex Hull size.
- Point< T > pivot

  Pivot point in input points.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 Chan()

```
template<class T >
Chan< T >::Chan (
            vector< Point< T >> & points )
```

Construct a new Chan object.

Parameters

points	Set of input points

```
13 {
14     numPoints = points.size();
15     this->points = points;
16     assert(numPoints >= 3);
17 
18     // Compute pivot
19     Point<T> best(points[0]);
20     for (Point<T> &point : points)
21     {
22         if (point < best)
23             best = point;
24     }
25     pivot = best;
26 
27     int guessedHullSize = 1000;
28     while (!restrictedConvexHull(guessedHullSize))
29         guessedHullSize = guessedHullSize * guessedHullSize;
30 }
```

4.1.2 Member Function Documentation

4.1.2.1 computePartitions()

```
template<class T >
vector< vector< Point< T > > > Chan< T >::computePartitions (
            int partitionSize )  [private]
```

Divide input points equally into partitions of size 'partitionSize'.

Parameters

partitionSize	

Returns

vector<vector<Point<T>>> Partitions

```
73 {
74     vector<vector<Point<T»> partitions;
```

```
75      vector<Point<T» partition;
76      if (partitionSize >= numPoints)
77      {
78          partitions.push_back(points);
79          return partitions;
80      }
81      int numPartitions = numPoints / partitionSize;
82      partitions = vector<vector<Point<T»>(numPartitions);
83      for (int i = 0; i < numPoints; i++)
84          partitions[i % numPartitions].push_back(points[i]);
85      return partitions;
86  }
```

4.1.2.2 getConvexHull()

```
template<class T >
vector< Point< T > > Chan< T >::getConvexHull ( )
```

Get convex hull computed using Chan's algorithm.

Returns

vector<Point<T>> Set of points int convex hull in counter clockwise order

```
90  {
91      return convexHull;
92  }
```

4.1.2.3 restrictedConvexHull()

```
template<class T >
bool Chan< T >::restrictedConvexHull (
             int guessedHullSize )  [private]
```

Tries to compute convex hull using given partition size (guessed size of hull)

Parameters

guessedHullSize	Guessed size of hull

Returns

true If algorithm completes, i.e. guessed size if greater or equal to actual hull size
false If algorithm fails to complete

```
34  {
35      // Compute partitions
36      vector<vector<Point<T»> partitions = computePartitions(guessedHullSize);
37
38      // Graham's step O(numPoints * log(guessedHullSize))
39      int numPartitions = partitions.size();
40      vector<GrahamScan<T>*> grahamScans(numPartitions);
41      for (int i = 0; i < numPartitions; i++)
42          grahamScans[i] = new GrahamScan<T>(partitions[i]);
43
44      Point<T> currPivot = pivot;
45      vector<Point<T» hull;
46      for (int i = 1; i <= guessedHullSize; i++)
```

```
47      {
48          hull.push_back(currPivot);
49          vector<Point<T» candidatePoints(numPartitions);
50          for (int i = 0; i < numPartitions; i++)
51              candidatePoints[i] = grahamScans[i]->getRightTangentPoint(currPivot);
52
53          // Perform jarvis step
54          currPivot = JarvisStep<T>(currPivot, candidatePoints).getNext();
55          if (currPivot == pivot)
56              break;
57      }
58      if (currPivot == pivot)
59      {
60          // Finished successfully
61          convexHull = hull;
62          return true;
63      }
64      else
65      {
66          // Failed
67          return false;
68      }
69  }
```

4.1.3 Member Data Documentation

4.1.3.1 convexHull

```
template<class T >
vector<Point<T> > Chan< T >::convexHull  [private]
```

Set of points in the convex hull in counter clockwise order.

4.1.3.2 convexHullSize

```
template<class T >
int Chan< T >::convexHullSize  [private]
```

Convex Hull size.

4.1.3.3 numPoints

```
template<class T >
int Chan< T >::numPoints  [private]
```

Number of points in the input.

4.1.3.4 pivot

```
template<class T >
Point<T> Chan< T >::pivot  [private]
```

Pivot point in input points.

4.1.3.5 points

```
template<class T >
vector<Point<T> > Chan< T >::points  [private]
```

Input points for computing convex hull.

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

- include/Chan.hpp
- src/Chan.cpp

4.2 GrahamScan< T >::Comparator Class Reference

Compare ordering of points for graham's scan.

Public Member Functions

- Comparator (Point< T > pivot)
- bool operator() (Point< T > &p1, Point< T > &p2)

Private Attributes

- Point< T > pivot

4.2.1 Detailed Description

template<class T>
class GrahamScan< T >::Comparator

Compare ordering of points for graham's scan.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Comparator()

```
template<class T >
GrahamScan< T >::Comparator::Comparator (
            Point< T > pivot )  [inline]
78         {
79             this->pivot = pivot;
80         }
```

4.2.3 Member Function Documentation

4.2.3.1 operator()()

```
template<class T >
bool GrahamScan< T >::Comparator::operator() (
            Point< T > & p1,
            Point< T > & p2 )  [inline]
83         {
84             int o = Point<T>::orient(pivot, p1, p2);
85             if (o == 1)
86                 return 1;
87             else if (o == -1)
88                 return 0;
89             else
90                 return p1 < p2;
91         }
```

4.2.4 Member Data Documentation

4.2.4.1 pivot

```
template<class T >
Point<T> GrahamScan< T >::Comparator::pivot  [private]
```

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

- include/GrahamScan.hpp

4.3 GrahamScan< T > Class Template Reference

```
#include <GrahamScan.hpp>
```

Classes

- class Comparator

  Compare ordering of points for graham's scan.

Public Member Functions

- GrahamScan (vector< Point< T >> &points)

  Construct a new Graham Scan object.
- vector< Point< T > > getConvexHull ()
- Point< T > getRightTangentPoint (Point< T > pivot)

  Get the Right Tangent Point, using binary search on the convex hull from the pivot as reference.

Private Member Functions

- void insert (Point< T > &p)

  Insert new point into the stack of partially constructed convex hull.
- bool isBelow (Point< T > pivot, Point< T > a, Point< T > b)

  Determine if 'a' lies strictly to left of 'b' (in orientation) w.r.t pivot.
- bool isAbove (Point< T > pivot, Point< T > a, Point< T > b)

  Determine if a lies strictly to right of b (in orientation) w.r.t pivot.
- Point< T > getPoint (int idx)

  Get the Point from convex hull at position 'idx'.

Private Attributes

- stack< Point< T > > order

  Order stack for maintaining points in the partially constructed convex hull.
- int numPoints

  Number of points in input.
- vector< Point< T > > convexHull

  Counter clockwiser order of points in the convex hull.
- int convexHullSize

  Number of points in convex hull.

4.3.1 Constructor & Destructor Documentation

4.3.1.1 GrahamScan()

```
template<class T >
GrahamScan< T >::GrahamScan (
            vector< Point< T >> & points )
```

Construct a new Graham Scan object.

Parameters

points	set of points for constructing convex hull

```
14 {
15     numPoints = points.size();
```

```
16      assert(numPoints >= 3);
17
18      Point<T> best(points[0]);
19      for (Point<T> &point : points)
20          if (point < best)
21              best = point;
22
23      // Sort vertices by x-coordinate then by y-coordinate
24      sort(points.begin(), points.end(), Comparator(best));
25
26      order.push(points[0]);
27      order.push(points[1]);
28
29      // Construct Convex hull
30      for (int i = 2; i < numPoints; i++)
31          insert(points[i]);
32
33      // Get all points in convex hull avoiding repeatition for lowermost-leftmost point
34      while (!order.empty())
35      {
36          Point<T> top = order.top();
37          convexHull.push_back(top);
38          order.pop();
39      }
40      reverse(convexHull.begin(), convexHull.end());
41      convexHullSize = convexHull.size();
42  }
```

4.3.2 Member Function Documentation

4.3.2.1 getConvexHull()

```
template<class T >
vector< Point< T > > GrahamScan< T >::getConvexHull ( )
```

Returns

vector<Point<T>> set of points in convex hull in counter clockwise order

```
60  {
61      return convexHull;
62  }
```

4.3.2.2 getPoint()

```
template<class T >
Point< T > GrahamScan< T >::getPoint (
            int idx )  [private]
```

Get the Point from convex hull at position 'idx'.

Parameters

idx	

Returns

Point

```
138 {
139     if (idx >= convexHullSize)
140         idx -= convexHullSize;
141     else if (idx < 0)
142         idx += convexHullSize;
143     return convexHull[idx];
144 }
```

4.3.2.3 getRightTangentPoint()

```
template<class T >
Point< T > GrahamScan< T >::getRightTangentPoint (
            Point< T > pivot )
```

Get the Right Tangent Point, using binary search on the convex hull from the pivot as reference.

Parameters

pivot	

Returns

Point on the convex hull lying on the right tangent

```
92 {
93     if (convexHullSize < 3)
94         return JarvisStep<T>(pivot, convexHull).getNext();
95
96     // Check if convexHull[0] if local maximum
97     if (isBelow(pivot, getPoint(1), getPoint(0)) &&
98         !isAbove(pivot, getPoint(convexHullSize - 1),       getPoint(0)))
99         return getPoint(0);
100
101     int low = 0, high = convexHullSize;
102     while (true)
103     {
104         int mid = (low + high) / 2;
105         bool isMidDown = isBelow(pivot, getPoint(mid + 1),       getPoint(mid));
106         if (isMidDown && !isAbove(pivot, getPoint(mid - 1),       getPoint(mid)))
107             return getPoint(mid);
108         bool isLowUp = isAbove(pivot, getPoint(low + 1), getPoint(low));
109         if (isLowUp)
110         {
111             if (isMidDown)
112                 high = mid;
113             else
114             {
115                 if (isAbove(pivot, getPoint(low), getPoint(mid)))
116                     high = mid;
117                 else
118                     low = mid;
119             }
120         }
121         else
122         {
123             if (!isMidDown)
124                 low = mid;
125             else
126             {
127                 if (isBelow(pivot, getPoint(low), getPoint(mid)))
128                     high = mid;
129                 else
130                     low = mid;
131             }
132         }
133     }
134 }
```

4.3.2.4 insert()

```
template<class T >
void GrahamScan< T >::insert (
            Point< T > & p )  [private]
```

Insert new point into the stack of partially constructed convex hull.

Parameters

p	New point to be inserted

```
46 {
47     Point<T> top = order.top();
48     order.pop();
49     while (order.size() >= 1 && Point<T>::orient(order.top(), top, p) <= 0)
50     {
51         top = order.top();
52         order.pop();
53     }
54     order.push(top);
55     order.push(p);
56 }
```

4.3.2.5 isAbove()

```
template<class T >
bool GrahamScan< T >::isAbove (
            Point< T > pivot,
            Point< T > a,
            Point< T > b )  [private]
```

Determine if a lies strictly to right of b (in orientation) w.r.t pivot.

Parameters

pivot	
a	
b	

Returns

true If 'a' lies strictly to right of b
false otherwise

```
66 {
67     if (Point<T>::orient(pivot, a, b) > 0)
68         return true;
69     else if (Point<T>::orient(pivot, a, b) == 0)
70     {
71         if (pivot.squaredDistance(a) > pivot.squaredDistance(b))
72             return true;
73     }
74     return false;
75 }
```

4.3.2.6 isBelow()

```
template<class T >
bool GrahamScan< T >::isBelow (
            Point< T > pivot,
            Point< T > a,
            Point< T > b )  [private]
```

Determine if 'a' lies strictly to left of 'b' (in orientation) w.r.t pivot.

Parameters

pivot	
a	
b	

Returns

true If 'a' lies strictly to left of 'b'
false otherwise

```
79 {
80     if (Point<T>::orient(pivot, a, b) < 0)
81         return true;
82     else if (Point<T>::orient(pivot, a, b) == 0)
83     {
84         if (pivot.squaredDistance(a) < pivot.squaredDistance(b))
85             return true;
86     }
87     return false;
88 }
```

4.3.3 Member Data Documentation

4.3.3.1 convexHull

```
template<class T >
vector<Point<T> > GrahamScan< T >::convexHull  [private]
```

Counter clockwiser order of points in the convex hull.

4.3.3.2 convexHullSize

```
template<class T >
int GrahamScan< T >::convexHullSize  [private]
```

Number of points in convex hull.

4.3.3.3 numPoints

```
template<class T >
int GrahamScan< T >::numPoints  [private]
```

Number of points in input.

4.3.3.4 order

```
template<class T >
stack<Point<T> > GrahamScan< T >::order  [private]
```

Order stack for maintaining points in the partially constructed convex hull.

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

- include/GrahamScan.hpp
- src/GrahamScan.cpp

4.4 JarvisStep< T > Class Template Reference

```
#include <JarvisStep.hpp>
```

Public Member Functions

- JarvisStep (Point< T > pivot, vector< Point< T >> &points)
  Construct a new Jarvis March object.
- Point< T > getNext ()
  Get the next point in jarvis march.

Private Attributes

- int numPoints
  Number of points.
- Point< T > nextPoint
  Next point in jarvis march.

4.4.1 Constructor & Destructor Documentation

4.4.1.1 JarvisStep()

```
template<class T >
JarvisStep< T >::JarvisStep (
            Point< T > pivot,
            vector< Point< T >> & points )
```

Construct a new Jarvis March object.

Parameters

pivot	Reference pivot from which points are compared
points	Set of candidate points for Convex Hull

```
13 {
14     Point<T> best(points[0]);
15     for (Point<T> &point : points)
16     {
17         if (point == pivot)
18             continue;
19         // Selecting the right-most-oriented point w.r.t pivot
20         int orientation = pivot.orient(pivot, best, point);
21         if (orientation < 0)
22             best = point;
23         else if (orientation == 0)
24         {
25             // If points are collinear take the farthest one
26             if (pivot.squaredDistance(point) > pivot.        squaredDistance(best))
27                 best = point;
28         }
29     }
30     nextPoint = best;
31 }
```

4.4.2 Member Function Documentation

4.4.2.1 getNext()

```
template<class T >
Point< T > JarvisStep< T >::getNext ( )
```

Get the next point in jarvis march.

Returns

Point<T>

```
35 {
36     return nextPoint;
37 }
```

4.4.3 Member Data Documentation

4.4.3.1 nextPoint

```
template<class T>
Point<T> JarvisStep< T >::nextPoint  [private]
```

Next point in jarvis march.

4.4.3.2 numPoints

```
template<class T>
int JarvisStep< T >::numPoints  [private]
```

Number of points.

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

- include/JarvisStep.hpp
- src/JarvisStep.cpp

4.5 Point< T > Class Template Reference

```
#include <Point.hpp>
```

Public Member Functions

- Point ()
- Point (T a, T b)

  Construct a new Point object.
- bool operator< (const Point< T > &p)

  Compare two points, by order of x-coordinate then y-coordinate.
- bool operator== (const Point< T > &p)

  Check equality of points.
- pair< double, double > cross (Point< T > &p)

  Compute cross product of two vectors (represented as points)
- double squaredDistance (Point< T > &p)

  Compute squared Euclidean distance between points.
- void print ()

  Utility to print the point.

Static Public Member Functions

- static int orient (const Point< T > &p1, const Point< T > &p2, const Point< T > &p3)

  Compute orientation of the traversal of three points.

Public Attributes

- T x
- T y

4.5.1 Constructor & Destructor Documentation

4.5.1.1 Point() [1/2]

```
template<class T>
Point< T >::Point ( )  [inline]
14     {
15         x = 0;
16         y = 0;
17     }
```

4.5.1.2 Point() [2/2]

```
template<class T>
Point< T >::Point (
            T a,
            T b )  [inline]
```

Construct a new Point object.

Parameters

a	
b	

```
26     {
27         x = a;
28         y = b;
29     }
```

4.5.2 Member Function Documentation

4.5.2.1 cross()

```
template<class T>
pair<double, double> Point< T >::cross (
            Point< T > & p )  [inline]
```

Compute cross product of two vectors (represented as points)

Parameters

p	

Returns

int signed area of the parallelipiped formed by the vectors

```
64     {
65         double a = (double)x * (double)p.y;
66         double b = (double)p.x * (double)y;
67         return {a, b};
68     }
```

4.5.2.2 operator<()

```
template<class T>
bool Point< T >::operator< (
            const Point< T > & p )  [inline]
```

Compare two points, by order of x-coordinate then y-coordinate.

Parameters

p	

Returns

true If 'this' point is lesser in ordering
false otherwise

```
39     {
40         if (x != p.x)
41             return x < p.x;
42         return y < p.y;
43     }
```

4.5.2.3 operator==()

```
template<class T>
bool Point< T >::operator== (
            const Point< T > & p )  [inline]
```

Check equality of points.

Parameters

p	

Returns

true
false

```
53     {
54         return (x == p.x && y == p.y);
55     }
```

4.5.2.4 orient()

```
template<class T>
static int Point< T >::orient (
            const Point< T > & p1,
            const Point< T > & p2,
            const Point< T > & p3 )  [inline], [static]
```

Compute orientation of the traversal of three points.

Parameters

p1	
p2	
p3	

Returns

int value signifying orientation

```
79      {
80          Point<T> v1(p2.x - p1.x, p2.y - p1.y);
81          Point<T> v2(p3.x - p2.x, p3.y - p2.y);
82          auto o = v1.cross(v2);
83          if (o.first > o.second)
84              return 1;
85          else if (o.first < o.second)
86              return -1;
87          else
88              return 0;
89      }
```

4.5.2.5 print()

```
template<class T>
void Point< T >::print ( )  [inline]
```

Utility to print the point.

```
108     {
109         cout « x « " " « y « endl;
110     }
```

4.5.2.6 squaredDistance()

```
template<class T>
double Point< T >::squaredDistance (
            Point< T > & p )  [inline]
```

Compute squared Euclidean distance between points.

Parameters

p	

Returns

int distance

```
98      {
99          double dx = (double)x - (double)p.x;
100          double dy = (double)y - (double)p.y;
101          return Utils<double>::square(dx) +       Utils<double>::square(dy);
102      }
```

4.5.3 Member Data Documentation

4.5.3.1 x

```
template<class T>
T Point< T >::x
```

4.5.3.2 y

```
template<class T>
T Point< T >::y
```

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

- include/Point.hpp

4.6 Utils< T > Class Template Reference

```
#include <Utils.hpp>
```

Static Public Member Functions

- static T square (T x)

  Utility to compute square.

Private Member Functions

- Utils ()

  Private constructor to avoid instantiation.

4.6.1 Constructor & Destructor Documentation

4.6.1.1 Utils()

```
template<class T >
Utils< T >::Utils ( )  [inline], [private]
```

Private constructor to avoid instantiation.

```
11 {}
```

4.6.2 Member Function Documentation

4.6.2.1 square()

```
template<class T >
static T Utils< T >::square (
            T x )  [inline], [static]
```

Utility to compute square.

Parameters

x	

Returns

T

```
21     {
22         return x * x;
23     }
```

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

- include/Utils.hpp

Chapter 5

File Documentation

5.1 include/Chan.hpp File Reference

```
#include "GrahamScan.hpp"
#include "JarvisStep.hpp"
```

Classes

- class Chan< T >

5.2 include/GrahamScan.hpp File Reference

```
#include <bits/stdc++.h>
#include "Point.hpp"
#include "JarvisStep.hpp"
```

Classes

- class GrahamScan< T >
- class GrahamScan< T >::Comparator

  Compare ordering of points for graham's scan.

5.3 include/JarvisStep.hpp File Reference

```
#include <bits/stdc++.h>
#include "Point.hpp"
```

Classes

- class JarvisStep< T >

5.4 include/Point.hpp File Reference

```
#include "Utils.hpp"
```

Classes

- class Point< T >

5.5 include/Utils.hpp File Reference

```
#include <bits/stdc++.h>
```

Classes

- class Utils< T >

5.6 main.cpp File Reference

```
#include <bits/stdc++.h>
#include "Chan.hpp"
#include <vector>
```

Functions

- int32_t main ()

5.6.1 Function Documentation

5.6.1.1 main()

```
int32_t main ( )
10 {
11     // Number of points in the plane
12     int n;
13     cin » n;
14     vector<Point<int» points;
15     for (int i = 0; i < n; i++)
16     {
17         int x, y;
18         cin » x » y;
19         points.push_back(Point<int>(x, y));
20     }
21     vector<Point<int» convexHull = Chan<int>(points).getConvexHull();
22     cout « "Counter Clockwise order of Convex Hull: " « endl;
23     for (Point<int> &point : convexHull)
24     {
25         cout « point.x « " " « point.y « endl;
26     }
27 }
```

5.7 README.md File Reference

5.8 src/Chan.cpp File Reference

```
#include "Chan.hpp"
```

5.9 src/GrahamScan.cpp File Reference

```
#include "GrahamScan.hpp"
```

5.10 src/JarvisStep.cpp File Reference

```
#include "JarvisStep.hpp"
#include "Utils.hpp"
```

Index