**Bài tập đánh giá quá trình**

1. Thuật toán divide conquer để tìm bao lồi - độ phức tạp O(nlogn)

Code:

*#include* <iostream>

*#include* <algorithm>

*#include* <vector>

*#include* <set>

*#include* <cmath>

*#include* <limits>

*#include* <cfloat>

*#include*<iomanip>

*using* *namespace* std;

pair*<int*, *int>* *mid*;

*int* *quad*(pair<*int*, *int*> p) {

*if* (p.*first* *>=* 0 *&&* p.*second* *>=* 0)

*return* 1;

*if* (p.*first* *<=* 0 *&&* p.*second* *>=* 0)

*return* 2;

*if* (p.*first* *<=* 0 *&&* p.*second* *<=* 0)

*return* 3;

*return* 4;

}

*int* *orientation*(pair<*int*, *int*> a, pair<*int*, *int*> b, pair<*int*, *int*> c) {

*int* *res* *=* (b.*second* *-* a.*second*) *\** (c.*first* *-* b.*first*) *-*

(c.*second* *-* b.*second*) *\** (b.*first* *-* a.*first*);

*if* (*res* *==* 0)

*return* 0;

*if* (*res* *>* 0)

*return* 1;

*return* *-*1;

}

*bool* *compare*(pair<*int*, *int*> p1, pair<*int*, *int*> q1) {

pair*<int*, *int>* *p* *=* *make\_pair*(p1.*first* *-* *mid*.*first*, p1.*second* *-* *mid*.*second*);

pair*<int*, *int>* *q* *=* *make\_pair*(q1.*first* *-* *mid*.*first*, q1.*second* *-* *mid*.*second*);

*int* *one* *=* *quad*(*p*);

*int* *two* *=* *quad*(*q*);

*if* (*one* *!=* *two*)

*return* (*one* *<* *two*);

*return* (*p*.*second* *\** *q*.*first* *<* *q*.*second* *\** *p*.*first*);

}

vector<pair<*int*, *int*>> *merger*(vector<pair<*int*, *int*>> a, vector<pair<*int*, *int*>> b) {

*int* *n1* *=* a.*size*(), *n2* *=* b.*size*();

*int* *ia* *=* 0, *ib* *=* 0;

*for* (*int* *i* *=* 1; *i* *<* *n1*; *i++*)

*if* (a*[i]*.*first* *>* a*[ia]*.*first*)

*ia* *=* *i*;

*for* (*int* *i* *=* 1; *i* *<* *n2*; *i++*)

*if* (b*[i]*.*first* *<* b*[ib]*.*first*)

*ib* *=* *i*;

*int* *inda* *=* *ia*, *indb* *=* *ib*;

*bool* *done* *=* 0;

*while* (*!done*) {

*done* *=* 1;

*while* (*orientation*(b*[indb]*, a*[inda]*, a*[*(*inda* *+* 1) *%* *n1]*) *>=* 0)

*inda* *=* (*inda* *+* 1) *%* *n1*;

*while* (*orientation*(a*[inda]*, b*[indb]*, b*[*(*n2* *+* *indb* *-* 1) *%* *n2]*) *<=* 0) {

*indb* *=* (*n2* *+* *indb* *-* 1) *%* *n2*;

*done* *=* 0;

}

}

*int* *uppera* *=* *inda*, *upperb* *=* *indb*;

*inda* *=* *ia*, *indb* *=* *ib*;

*done* *=* 0;

*while* (*!done*) {

*done* *=* 1;

*while* (*orientation*(a*[inda]*, b*[indb]*, b*[*(*indb* *+* 1) *%* *n2]*) *>=* 0)

*indb* *=* (*indb* *+* 1) *%* *n2*;

*while* (*orientation*(b*[indb]*, a*[inda]*, a*[*(*n1* *+* *inda* *-* 1) *%* *n1]*) *<=* 0) {

*inda* *=* (*n1* *+* *inda* *-* 1) *%* *n1*;

*done* *=* 0;

}

}

*int* *lowera* *=* *inda*, *lowerb* *=* *indb*;

vector*<*pair*<int*, *int>>* *ret*;

*int* *ind* *=* *uppera*;

*ret*.*push\_back*(a*[uppera]*);

*while* (*ind* *!=* *lowera*) {

*ind* *=* (*ind* *+* 1) *%* *n1*;

*ret*.*push\_back*(a*[ind]*);

}

*ind* *=* *lowerb*;

*ret*.*push\_back*(b*[lowerb]*);

*while* (*ind* *!=* *upperb*) {

*ind* *=* (*ind* *+* 1) *%* *n2*;

*ret*.*push\_back*(b*[ind]*);

}

*return* *ret*;

}

vector<pair<*int*, *int*>> *bruteHull*(vector<pair<*int*, *int*>> a) {

set*<*pair*<int*, *int>>* *s*;

*for* (*int* *i* *=* 0; *i* *<* a.*size*(); *i++*) {

*for* (*int* *j* *=* *i* *+* 1; *j* *<* a.*size*(); *j++*) {

*int* *x1* *=* a*[i]*.*first*, *x2* *=* a*[j]*.*first*;

*int* *y1* *=* a*[i]*.*second*, *y2* *=* a*[j]*.*second*;

*int* *a1* *=* *y1* *-* *y2*;

*int* *b1* *=* *x2* *-* *x1*;

*int* *c1* *=* *x1* *\** *y2* *-* *y1* *\** *x2*;

*int* *pos* *=* 0, *neg* *=* 0;

*for* (*int* *k* *=* 0; *k* *<* a.*size*(); *k++*) {

*if* (*a1* *\** a*[k]*.*first* *+* *b1* *\** a*[k]*.*second* *+* *c1* *<=* 0)

*neg++*;

*if* (*a1* *\** a*[k]*.*first* *+* *b1* *\** a*[k]*.*second* *+* *c1* *>=* 0)

*pos++*;

}

*if* (*pos* *==* a.*size*() *||* *neg* *==* a.*size*()) {

*s*.*insert*(a*[i]*);

*s*.*insert*(a*[j]*);

}

}

}

vector*<*pair*<int*, *int>>* *ret*;

*for* (*auto* *e* : *s*)

*ret*.*push\_back*(*e*);

*mid* *=* {0, 0};

*int* *n* *=* *ret*.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*mid*.*first* *+=* *ret[i]*.*first*;

*mid*.*second* *+=* *ret[i]*.*second*;

*ret[i]*.*first* *\*=* *n*;

*ret[i]*.*second* *\*=* *n*;

}

*sort*(*ret*.*begin*(), *ret*.*end*(), *compare*);

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*)

*ret[i]* *=* *make\_pair*(*ret[i]*.*first* */* *n*, *ret[i]*.*second* */* *n*);

*return* *ret*;

}

vector<pair<*int*, *int*>> *divide*(vector<pair<*int*, *int*>> a) {

*if* (a.*size*() *<=* 5)

*return* *bruteHull*(a);

vector*<*pair*<int*, *int>>* *left*, *right*;

*for* (*int* *i* *=* 0; *i* *<* a.*size*() */* 2; *i++*)

*left*.*push\_back*(a*[i]*);

*for* (*int* *i* *=* a.*size*() */* 2; *i* *<* a.*size*(); *i++*)

*right*.*push\_back*(a*[i]*);

vector*<*pair*<int*, *int>>* *left\_hull* *=* *divide*(*left*);

vector*<*pair*<int*, *int>>* *right\_hull* *=* *divide*(*right*);

*return* *merger*(*left\_hull*, *right\_hull*);

}

*double* *polygonArea*(*const* vector<pair<*int*, *int*>>*&* points) {

*double* *area* *=* 0.0;

*int* *n* *=* points.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*int* *j* *=* (*i* *+* 1) *%* *n*;

*area* *+=* (points*[i]*.*first* *\** points*[j]*.*second*) *-* (points*[j]*.*first* *\** points*[i]*.*second*);

}

*return* *abs*(*area*) */* 2.0;

}

*double* *distance*(pair<*int*, *int*> p1, pair<*int*, *int*> p2) {

*return* *sqrt*((p1.*first* *-* p2.*first*) *\** (p1.*first* *-* p2.*first*) *+*

(p1.*second* *-* p2.*second*) *\** (p1.*second* *-* p2.*second*));

}

*double* *shortestDistance*(*const* vector<pair<*int*, *int*>>*&* points) {

*double* *minDist* *=* *DBL\_MAX*;

pair*<int*, *int>* *p1*, *p2*;

*int* *n* *=* points.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*for* (*int* *j* *=* *i* *+* 1; *j* *<* *n*; *j++*) {

*double* *dist* *=* *distance*(points*[i]*, points*[j]*);

*if* (*dist* *<* *minDist*) {

*minDist* *=* *dist*;

*p1* *=* points*[i]*;

*p2* *=* points*[j]*;

}

}

}

*cout* *<<* "The two points with the shortest distance are: ("

*<<* *p1*.*first* *<<* ", " *<<* *p1*.*second* *<<* ") and ("

*<<* *p2*.*first* *<<* ", " *<<* *p2*.*second* *<<* "): ";

*return* *minDist*;

}

*double* *smallestEdge*(*const* vector<pair<*int*, *int*>>*&* points) {

*double* *minEdge* *=* *DBL\_MAX*;

*int* *n* *=* points.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*int* *j* *=* (*i* *+* 1) *%* *n*;

*minEdge* *=* *min*(*minEdge*, *distance*(points*[i]*, points*[j]*));

}

*return* *minEdge*;

}

*bool* *pointInConvexHull*(*const* vector<pair<*int*, *int*>>*&* hull, pair<*int*, *int*> point) {

*int* *n* *=* hull.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*int* *j* *=* (*i* *+* 1) *%* *n*;

*if* (*orientation*(hull*[i]*, hull*[j]*, point) *<* 0)

*return* false;

}

*return* true;

}

*int* *main*() {

vector*<*pair*<int*, *int>>* *a*;

*srand*(*time*(0));

set*<int>* *uniqueY*;

*while* (*a*.*size*() *<* 15) {

*int* *x* *=* *rand*() *%* 21;

*int* *y* *=* *rand*() *%* 21;

*if* (*uniqueY*.*find*(*y*) *==* *uniqueY*.*end*()) {

*a*.*push\_back*(*make\_pair*(*x*, *y*));

*uniqueY*.*insert*(*y*);

}

}

*cout<<*"Original Points:\n";

*for*(*auto* *p* : *a*){

*cout<<*"("*<<p*.*first<<*" "*<<p*.*second<<*")"*<<setw*(3);

}

*cout<<endl*;

*sort*(*a*.*begin*(), *a*.*end*());

vector*<*pair*<int*, *int>>* *ans* *=* *divide*(*a*);

*cout* *<<* "Convex Hull Points:\n";

*for* (*auto* *e* : *ans*)

*cout* *<<*"("*<<e*.*first* *<<* " " *<<* *e*.*second* *<<* ")\t" *<<endl*;

*cout* *<<* "Area of Convex Hull: " *<<* *polygonArea*(*ans*) *<<* *endl*;

*cout* *<<* "Shortest Distance Between Two Points: " *<<* *shortestDistance*(*a*) *<<* *endl*;

*cout* *<<* "Smallest Edge of Convex Hull: " *<<* *smallestEdge*(*ans*) *<<* *endl*;

pair*<int*, *int>* *randomPoint* *=* {2,0};

*if* (*pointInConvexHull*(*ans*, *randomPoint*)) {

*cout* *<<* "Point (" *<<* *randomPoint*.*first* *<<* ", " *<<* *randomPoint*.*second* *<<* ") is inside the convex hull.\n";

} *else* {

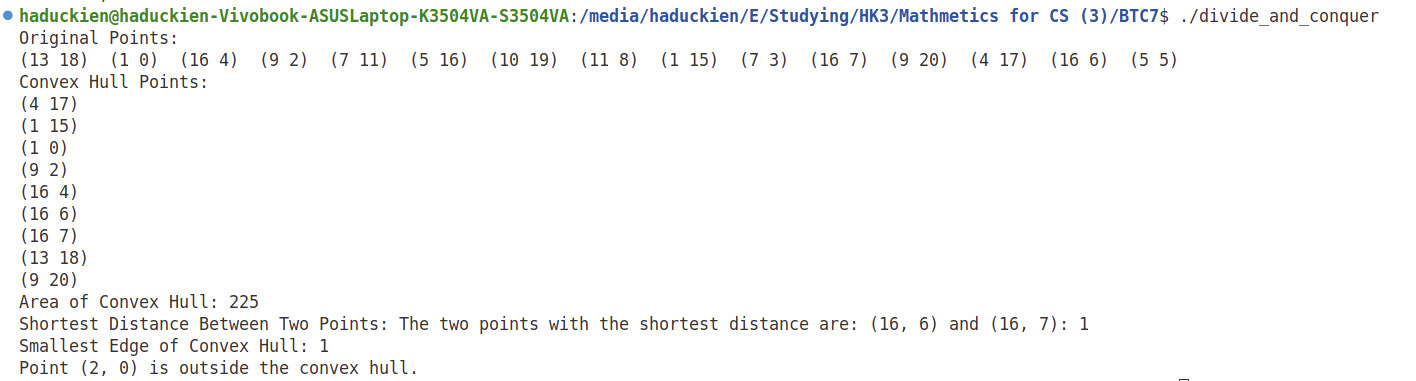
*cout* *<<* "Point (" *<<* *randomPoint*.*first* *<<* ", " *<<* *randomPoint*.*second* *<<* ") is outside the convex hull.\n";

}

*return* 0;

}

Kết quả thực thi:

2. Thuật toán Chan’s để tìm bao lồi: độ phức tạp O(nlogh) với h là số điểm trong bao lồi.

Code:

*#include* <iostream>

*#include* <stdlib.h>

*#include* <vector>

*#include* <algorithm>

*#include* <utility>

*#include* <set>

*#include* <cmath>

*#include* <limits>

*#include* <ctime>

*#define* *RIGHT\_TURN* *-*1

*#define* *LEFT\_TURN* 1

*#define* *COLLINEAR* 0

*using* *namespace* std;

*class* Point{

*public:*

*int* *x*;

*int* *y*;

*Point* (*int* newx*=*0,*int* newy*=*0){

*x=*newx;

*y=*newy;

}

*friend* *bool* *operator==* (*const* Point*&* p1,*const* Point*&* p2){

*return* (p1.*x==*p2.*x* *&&* p1.*y==*p2.*y*);

}

*friend* *bool* *operator!=* (*const* Point*&* p1,*const* Point*&* p2){

*return* (*!*(p1.*x==*p2.*x* *&&* p1.*y==*p2.*y*));

}

*friend* ostream*&* *operator<<*(ostream*&* output,*const* Point*&* p){

output*<<*"("*<<*p.*x<<*","*<<*p.*y<<*")";

*return* output;

}

} *p0*;

*int* *dist*(Point p1, Point p2){

*return* (p1.*x* *-* p2.*x*)*\**(p1.*x* *-* p2.*x*) *+* (p1.*y* *-* p2.*y*)*\**(p1.*y* *-* p2.*y*);

}

*int* *orientation*(Point p, Point q, Point r){

*int* *val* *=* (q.*y* *-* p.*y*) *\** (r.*x* *-* q.*x*) *-* (q.*x* *-* p.*x*) *\** (r.*y* *-* q.*y*);

*if* (*val* *==* 0) *return* 0;

*return* (*val* *>* 0)*?* *-*1*:* 1;

}

*int* *compare*(*const* *void* *\**vp1, *const* *void* *\**vp2){

Point *\*p1* *=* (Point *\**)vp1;

Point *\*p2* *=* (Point *\**)vp2;

*int* *orient* *=* *orientation*(*p0*, *\*p1*, *\*p2*);

*if* (*orient* *==* 0)

*return* (*dist*(*p0*, *\*p2*) *>=* *dist*(*p0*, *\*p1*))*?* *-*1 *:* 1;

*return* (*orient* *==* 1)*?* *-*1*:* 1;

}

*int* *tangent*(vector<Point> v,Point p){

*int* *l=*0;

*int* *r=* v.*size*();

*int* *l\_before* *=* *orientation*(p, v*[*0*]*, v*[*v.*size*()*-*1*]*);

*int* *l\_after* *=* *orientation*(p, v*[*0*]*, v*[*(*l* *+* 1) *%* v.*size*()*]*);

*while* (*l* *<* *r*){

*int* *c* *=* ((*l* *+* *r*)*>>*1);

*int* *c\_before* *=* *orientation*(p, v*[c]*, v*[*(*c* *-* 1) *%* v.*size*()*]*);

*int* *c\_after* *=* *orientation*(p, v*[c]*, v*[*(*c* *+* 1) *%* v.*size*()*]*);

*int* *c\_side* *=* *orientation*(p, v*[l]*, v*[c]*);

*if* ((*c\_before* *!=* *RIGHT\_TURN*) *&&* (*c\_after* *!=* *RIGHT\_TURN*))

*return* *c*;

*else* *if* ((*c\_side* *==* *LEFT\_TURN*) *&&* (*l\_after* *==* *RIGHT\_TURN* *||* *l\_before* *==* *l\_after*) *||* (*c\_side* *==* *RIGHT\_TURN* *&&* *c\_before* *==* *RIGHT\_TURN*))

*r* *=* *c*;

*else*

*l* *=* *c* *+* 1 ;

*l\_before* *=* *-c\_after*;

*l\_after* *=* *orientation*(p, v*[l]*, v*[*(*l* *+* 1) *%* v.*size*()*]*);

}

*return* *l*;

}

pair<*int*,*int*> *extreme\_hullpt\_pair*(vector<vector<Point> >*&* hulls){

*int* *h=* 0,*p=* 0;

*for* (*int* *i=*0; *i<*hulls.*size*(); *++i*){

*int* *min\_index=*0, *min\_y* *=* hulls*[i][*0*]*.*y*;

*for*(*int* *j=*1; *j<* hulls*[i]*.*size*(); *++j*){

*if*(hulls*[i][j]*.*y* *<* *min\_y*){

*min\_y=*hulls*[i][j]*.*y*;

*min\_index=j*;

}

}

*if*(hulls*[i][min\_index]*.*y* *<* hulls*[h][p]*.*y*){

*h=i*;

*p=min\_index*;

}

}

*return* *make\_pair*(*h*,*p*);

}

pair<*int*,*int*> *next\_hullpt\_pair*(vector<vector<Point> >*&* hulls, pair<*int*,*int*> lPoint){

Point *p* *=* hulls*[*lPoint.*first][*lPoint.*second]*;

pair*<int*,*int>* *next* *=* *make\_pair*(lPoint.*first*, (lPoint.*second* *+* 1) *%* hulls*[*lPoint.*first]*.*size*());

*for* (*int* *h=*0; *h<* hulls.*size*(); *h++*){

*if*(*h* *!=* lPoint.*first*){

*int* *s=* *tangent*(hulls*[h]*,*p*);

Point *q=* hulls*[next*.*first][next*.*second]*;

Point *r=* hulls*[h][s]*;

*int* *t=* *orientation*(*p*,*q*,*r*);

*if*( *t==* *RIGHT\_TURN* *||* (*t==COLLINEAR*) *&&* *dist*(*p*,*r*)*>dist*(*p*,*q*))

*next* *=* *make\_pair*(*h*,*s*);

}

}

*return* *next*;

}

vector<Point> *keep\_left* (vector<Point>*&* v,Point p){

*while*(v.*size*()*>*1 *&&* *orientation*(v*[*v.*size*()*-*2*]*,v*[*v.*size*()*-*1*]*,p) *!=* *LEFT\_TURN*)

v.*pop\_back*();

*if*(*!*v.*size*() *||* v*[*v.*size*()*-*1*]* *!=* p)

v.*push\_back*(p);

*return* v;

}

vector<Point> *GrahamScan*(vector<Point>*&* Points) {

*if* (Points.*size*() *<=* 1)

*return* Points;

*qsort*(*&*Points*[*0*]*, Points.*size*(), *sizeof*(Point), *compare*);

vector*<*Point*>* *lower\_hull*;

*for* (*const* *auto&* *p* : Points)

*lower\_hull* *=* *keep\_left*(*lower\_hull*, *p*);

*reverse*(Points.*begin*(), Points.*end*());

vector*<*Point*>* *upper\_hull*;

*for* (*const* *auto&* *p* : Points)

*upper\_hull* *=* *keep\_left*(*upper\_hull*, *p*);

*lower\_hull*.*insert*(*lower\_hull*.*end*(), *upper\_hull*.*begin*() *+* 1, *upper\_hull*.*end*());

*return* *lower\_hull*;

}

vector<Point> *chansalgorithm*(vector<Point> v) {

*for*(*int* *t* *=* 0; *t* *<* v.*size*(); *++t*) {

*for*(*int* *m* *=* 1; *m* *<* (1 *<<* (1 *<<* *t*)); *++m*) {

vector*<*vector*<*Point*>>* *hulls*;

*for*(*int* *i* *=* 0; *i* *<* v.*size*(); *i* *+=* *m*) {

vector*<*Point*>* *chunk*;

*if*(v.*begin*() *+* *i* *+* *m* *<=* v.*end*())

*chunk*.*assign*(v.*begin*() *+* *i*, v.*begin*() *+* *i* *+* *m*);

*else*

*chunk*.*assign*(v.*begin*() *+* *i*, v.*end*());

*hulls*.*push\_back*(*GrahamScan*(*chunk*));

}

vector*<*pair*<int*, *int>>* *hull*;

*hull*.*push\_back*(*extreme\_hullpt\_pair*(*hulls*));

*for*(*int* *i* *=* 0; *i* *<* *m*; *++i*) {

pair*<int*, *int>* *p* *=* *next\_hullpt\_pair*(*hulls*, *hull[hull*.*size*() *-* 1*]*);

vector*<*Point*>* *output*;

*if*(*p* *==* *hull[*0*]*) {

*for*(*int* *j* *=* 0; *j* *<* *hull*.*size*(); *++j*) {

*output*.*push\_back*(*hulls[hull[j]*.*first][hull[j]*.*second]*);

}

*return* *output*;

}

*hull*.*push\_back*(*p*);

}

}

}

*return* {};

}

vector<Point> *generateRandomPoints*(*int* numberOfPoints, *int* maxRange) {

vector*<*Point*>* *Points*;

*srand*(*time*(nullptr));

set*<int>* *existingYValues*;

*while* (*Points*.*size*() *<* numberOfPoints) {

*int* *x* *=* *rand*() *%* maxRange;

*int* *y*;

*do* {

*y* *=* *rand*() *%* maxRange;

} *while* (*existingYValues*.*find*(*y*) *!=* *existingYValues*.*end*());

*existingYValues*.*insert*(*y*);

*Points*.*push\_back*(Point{*x*, *y*});

}

*return* *Points*;

}

*double* *poly\_area*(*const* vector<Point> *&*P)

{

*int* *n* *=* P.*size*();

*double* *area* *=* 0;

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*)

{

*int* *j* *=* (*i* *+* 1) *%* *n*;

*area* *+=* P*[i]*.*x\** P*[j]*.*y* *-* P*[i]*.*y* *\** P*[j]*.*x*;

}

*return* *abs*(*area*) */* 2.0;

}

*bool* *is\_Point\_in\_convex\_hull*(*const* vector<Point>*&* hull, *const* Point*&* p) {

*int* *n* *=* hull.*size*();

*if* (*n* *<* 3) *return* false;

*for* (*int* *i* *=* 0; *i* *<* *n*; *++i*) {

*int* *next* *=* (*i* *+* 1) *%* *n*;

*int* *orient* *=* *orientation*(hull*[i]*, hull*[next]*, p);

*if* (*orient* *==* *RIGHT\_TURN*) {

*return* false;

}

}

*return* true;

}

*int* *main*(){

*srand*(*time*(0));

*int* *n* *=* 15;

*int* *max\_range* *=* 20;

vector*<*Point*>* *v* *=* *generateRandomPoints*(*n*, *max\_range*);

*cout* *<<* "Random Points: \n";

*for* (*auto* *p* : *v*) *cout* *<<* *p* *<<* "\t";

*cout* *<<* *endl*;

vector*<*Point*>* *output* *=* *chansalgorithm*(*v*);

*cout* *<<* "\nConvex hull: \n";

*for*(*auto* *p* : *output*) *cout* *<<* *p* *<<* "\t";

*cout* *<<* "\n";

*cout* *<<* "\nPolygon area: " *<<* *poly\_area*(*output*) *<<* *endl*;

*int* *m* *=* 1;

vector*<*Point*>* *testPoint* *=* *generateRandomPoints*(*m*, *max\_range*);

*for* (*auto* *p* : *testPoint*){

*cout* *<<* "Test Point: " *<<* *p* *<<* "\t";

}

*cout* *<<* *endl*;

*if* (*is\_Point\_in\_convex\_hull*(*output*, *testPoint[*0*]*)) {

*cout* *<<* "Point " *<<* *testPoint[*0*]* *<<* " is in the convex hull." *<<* *endl*;

} *else* {

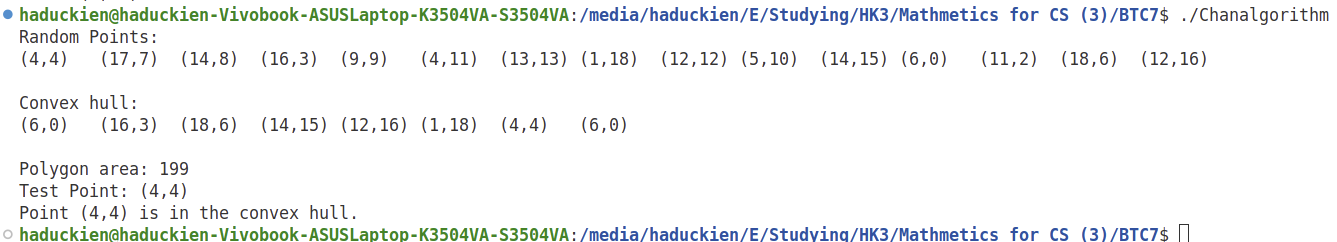
*cout* *<<* "Point " *<<* *testPoint[*0*]* *<<* " is not in the convex hull." *<<* *endl*;

}

*return* 0;

}

Kết quả thực thi chương trình:



3. Thuật toán Graham - scanning để tìm bao lồi: độ phức tạp O(nlogn)

*#include* <iostream>

*#include* <vector>

*#include* <cmath>

*#include* <algorithm>

*#include* <cstdlib>

*#include* <ctime>

*#include* <set>

*using* *namespace* std;

*struct* Point {

*int* *x*, *y*;

*bool* *operator<*(*const* Point*&* other) *const* {

*if* (*x* *!=* other.*x*) *return* *x* *<* other.*x*;

*return* *y* *<* other.*y*;

}

};

vector<Point> *generateRandomPoints*(*int* numberOfPoints, *int* maxRange) {

vector*<*Point*>* *points*;

*srand*(*time*(nullptr));

set*<int>* *existingYValues*;

*while* (*points*.*size*() *<* numberOfPoints) {

*int* *x* *=* *rand*() *%* maxRange;

*int* *y*;

*do* {

*y* *=* *rand*() *%* maxRange;

} *while* (*existingYValues*.*find*(*y*) *!=* *existingYValues*.*end*());

*existingYValues*.*insert*(*y*);

*points*.*push\_back*(Point{*x*, *y*});

}

*return* *points*;

}

*double* *angleToPoint*(*const* Point*&* pt1, *const* Point*&* pt2) {

*return* *atan2*(pt1.*y* *-* pt2.*y*, pt2.*x* *-* pt1.*x*);

}

*double* *getAngleBetween3Points*(*const* Point*&* pt1, *const* Point*&* pt2, *const* Point*&* pt3) {

*double* *ab* *=* *sqrt*(*pow*(pt2.*x* *-* pt1.*x*, 2) *+* *pow*(pt2.*y* *-* pt1.*y*, 2));

*double* *bc* *=* *sqrt*(*pow*(pt2.*x* *-* pt3.*x*, 2) *+* *pow*(pt2.*y* *-* pt3.*y*, 2));

*double* *ac* *=* *sqrt*(*pow*(pt3.*x* *-* pt1.*x*, 2) *+* *pow*(pt3.*y* *-* pt1.*y*, 2));

*return* *acos*((*bc* *\** *bc* *+* *ab* *\** *ab* *-* *ac* *\** *ac*) */* (2 *\** *bc* *\** *ab*));

}

*double* *getAreaBetween3Points*(*const* Point*&* pt1, *const* Point*&* pt2, *const* Point*&* pt3) {

*return* (pt2.*x* *-* pt1.*x*) *\** (pt1.*y* *-* pt3.*y*) *-* (pt3.*x* *-* pt1.*x*) *\** (pt1.*y* *-* pt2.*y*);

}

*bool* *checkIfLeftTurn*(*const* Point*&* pt1, *const* Point*&* pt2, *const* Point*&* pt3) {

*return* *getAreaBetween3Points*(pt1, pt2, pt3) *>* 0;

}

*bool* *isSamePoint*(*const* Point*&* p1, *const* Point*&* p2) {

*return* p1.*x* *==* p2.*x* *&&* p1.*y* *==* p2.*y*;

}

vector<Point> *grahamScan*(vector<Point>*&* points) {

Point *v\_lowest* *=* points*[*0*]*;

*for* (*size\_t* *i* *=* 1; *i* *<* points.*size*(); *i++*) {

*if* (points*[i]*.*y* *>* *v\_lowest*.*y* *||* (points*[i]*.*y* *==* *v\_lowest*.*y* *&&* points*[i]*.*x* *<* *v\_lowest*.*x*)) {

*v\_lowest* *=* points*[i]*;

}

}

*sort*(points.*begin*(), points.*end*(), [*&v\_lowest*](*const* Point*&* a, *const* Point*&* b) {

*double* *ang\_a* *=* *angleToPoint*(*v\_lowest*, a);

*double* *ang\_b* *=* *angleToPoint*(*v\_lowest*, b);

*return* *ang\_a* *<* *ang\_b*;

});

vector*<*Point*>* *stack*;

*if* (points.*size*() *<* 4) *return* points;

*stack*.*push\_back*(points*[*0*]*);

*stack*.*push\_back*(points*[*1*]*);

*size\_t* *index* *=* 2;

*while* (*index* *<* points.*size*()) {

*size\_t* *stacklen* *=* *stack*.*size*();

*if* (*stacklen* *>* 1 *&&* *checkIfLeftTurn*(*stack[stacklen* *-* 2*]*, *stack[stacklen* *-* 1*]*, points*[index]*)) {

*stack*.*push\_back*(points*[index]*);

*index++*;

} *else* {

*stack*.*pop\_back*();

}

}

*return* *stack*;

}

*double* *calculateHullArea*(*const* vector<Point>*&* hull) {

*double* *area* *=* 0.0;

*int* *n* *=* hull.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*int* *j* *=* (*i* *+* 1) *%* *n*;

*area* *+=* hull*[i]*.*x* *\** hull*[j]*.*y* *-* hull*[i]*.*y* *\** hull*[j]*.*x*;

}

*return* *abs*(*area*) */* 2.0;

}

*bool* *isPointInHull*(*const* vector<Point>*&* hull, *const* Point*&* p) {

*int* *n* *=* hull.*size*();

*for* (*int* *i* *=* 0; *i* *<* *n*; *i++*) {

*const* Point*&* *pt1* *=* hull*[i]*;

*const* Point*&* *pt2* *=* hull*[*(*i* *+* 1) *%* *n]*;

*if* (*checkIfLeftTurn*(*pt1*, *pt2*, p)) {

*return* false;

}

}

*return* true;

}

*int* *main*() {

*int* *n* *=* 15;

*int* *maxRange* *=* 20;

vector*<*Point*>* *points* *=* *generateRandomPoints*(*n*, *maxRange*);

*cout<<*"Random points:\n";

*for* (*const* *auto&* *point* : *points*) {

*cout* *<<* "(" *<<* *point*.*x* *<<* ", " *<<* *point*.*y* *<<* ") ";

}

cout<<”\n”;

*auto* *hull* *=* *grahamScan*(*points*);

*cout* *<<* "Convex Hull points:\n";

*for* (*const* *auto&* *point* : *hull*) {

*cout* *<<* "(" *<<* *point*.*x* *<<* ", " *<<* *point*.*y* *<<* ")" *<<* *endl*;

}

*double* *hullArea* *=* *calculateHullArea*(*hull*);

*cout* *<<* "Convex Hull Area: " *<<* *hullArea* *<<* *endl*;

*int* *m* *=* 1;

vector*<*Point*>* *testPoints* *=* *generateRandomPoints*(*m*, *maxRange*);

*for* (*const* *auto&* *point* : *testPoints*) {

*cout* *<<* "(" *<<* *point*.*x* *<<* ", " *<<* *point*.*y* *<<* ")" *<<* *endl*;

}

*if* (*isPointInHull*(*hull*, *testPoints[*0*]*)) {

*cout* *<<* "Point " *<<* *testPoints[*0*]*.*x* *<<* ", " *<<* *testPoints[*0*]*.*y* *<<* " is in the convex hull." *<<* *endl*;

} *else* {

*cout* *<<* "Point " *<<* *testPoints[*0*]*.*x* *<<* ", " *<<* *testPoints[*0*]*.*y* *<<* " is not in the convex hull." *<<* *endl*;

}

*return* 0;

}

Kết quả thực thi chương trình:

