|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Практическое задание №7 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **кафедра теоретической и прикладной информатики** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-03 |
| Бригада: | Место для ввода текста. |
| Студенты: | Сидоров Даниил, |
|  | Малыгин Сергей |
|  |  |
| Преподаватели: | Щукин Георгий Анатольевич |
|  |  |
|  | | |
| Новосибирск | | |
| 2021 | | |

**1.Результаты замеров:**

|  |  |  |
| --- | --- | --- |
| N | Разделяй и властвуй | Наивная реализация |
| 1 000 | 0.0014145 sec | 0.0021779 sec |
| 10 000 | 0.0217923 sec | 0.223343 sec |
| 100 000 | 0.194795 sec | 22.8963 sec |
| 1 000 000 | 2.10759 sec | 2282.07 sec |

**2.Программа:**

#include <stdexcept>

#include "closest\_pair.h"

#include <algorithm>

using namespace std;

std::pair<Point, Point> closest\_pair(const std::vector<Point> &points) {

if (points.size() == 2)

{

auto result = std::make\_pair(points[0], points[1]);

return result;

}

if (points.size() < 2) {

throw invalid\_argument("few points");

}

vector<Point> pointscopy = points;

std::sort(pointscopy.begin(), pointscopy.end(), [](const Point a, const Point b)

{

return a.x > b.x;

});

int new\_size = pointscopy.size() / 2;

int count = 0;

vector<Point> PLeft;//разбиение на левых и правых

while (count < new\_size)

{

PLeft.push\_back(pointscopy[count]);

count++;

}

vector<Point> PRight;

while (count < pointscopy.size())

{

PRight.push\_back(pointscopy[count]);

count++;

}

pair<Point, Point> pr=make\_pair(PLeft[0], PRight[0]);//pl = closest\_pair(PLeft), pr = closest\_pair(PRight) - пары ближайших точек в левой и правой половине соответственно.

if (PRight.size() > 1) pr = closest\_pair(PRight);

pair<Point, Point> pl=make\_pair(PLeft[0], PRight[0]);

if (PLeft.size() > 1) pl = closest\_pair(PLeft);

double d = min(pr.first.distance(pr.second), pl.first.distance(pl.second));

pair<Point, Point>pb = closest\_pair\_between(PLeft, PRight, d);//pb- пара ближайших точек между половинами PLeft и PRight.

auto result = make\_pair(Point(), Point());

if (pb.first.distance(pb.second) < pl.first.distance(pl.second))

{

if (pb.first.distance(pb.second) < pr.first.distance(pr.second)) result = pb; else result = pr;

}

else

if (pr.first.distance(pr.second) < pl.first.distance(pl.second))result = pr; else result = pl;

return result;

}

std::pair<Point, Point> closest\_pair\_between(const std::vector<Point> &Pleft, const std::vector<Point> &Pright, double d)

{

double Xm;//Xm - граница между PLeft и PRight по x-координате.

if (Pleft[Pleft.size() - 1].x - Pright[0].x != 0) Xm = (Pleft[Pleft.size() - 1].x + Pright[0].x) / 2; else Xm = Pright[0].x;

vector<Point> PStripe;

for (int i = 0; i < Pleft.size(); i++)//PStripe - множество точек из PLeft и PRight, по x отстоящих от Xm менее чем на расстоянии d.

{

if (abs(Pleft[i].x - Xm) < d) PStripe.push\_back(Pleft[i]);

}

for (int i = 0; i < Pright.size(); i++)

{

if (abs(Pright[i].x - Xm) < d) PStripe.push\_back(Pright[i]);

}

std::sort(PStripe.begin(), PStripe.end(), [](const Point a, const Point b)

{

return a.y > b.y;

});

pair<Point, Point> pb = std::make\_pair(Pleft[0], Pright[0]);

for (int i = 0; i + 1 < PStripe.size(); i++)//Для каждой точки достаточно рассмотреть только следующие по y точки, отстоящие по y менее чем на расстоянии d.

if (PStripe[i].distance(PStripe[i + 1]) < pb.first.distance(pb.second))

pb = make\_pair(PStripe[i], PStripe[i + 1]);

return pb;

}

std::pair<Point, Point> Naive\_solution(const std::vector<Point> &points)

{

auto result = make\_pair(Point(), Point());

if (points.size() > 1)

{

if (points.size() == 2) return result = make\_pair(points[0], points[1]);

else

{

result = make\_pair(points[0], points[1]);

for (int i = 0; i+1 < points.size(); i++)

{

for (int j = i+1; j < points.size(); j++)

if (points[i].distance(points[j]) < result.first.distance(result.second)) result = make\_pair(points[i], points[j]);

}

return result;

}

}else throw invalid\_argument ("few points");

}

**Main.cpp**

#include "closest\_pair.h"

#include <iostream>

#include <chrono>

#include <ctime>

#define CATCH\_CONFIG\_RUNNER

#include "catch.hpp"

using namespace std;

int main(int argc, char \*\*argv) {

int result = Catch::Session().run(argc, argv);

setlocale(LC\_ALL, "rus");

std::vector<Point> P;

srand(time(0));

for (int i = 1000; i <= 1000000; i = i \* 10)

{

cout << "N = " << i << endl << endl;

for (int j = 0; j < i; j++)

{

double a = rand() % 100;

double b= rand() % 100;

P.push\_back({a,b});

}

cout << "Разделяй и властвуй "<< endl;

auto t1 = std::chrono::high\_resolution\_clock::now();

closest\_pair(P);

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout <<"Time: " << seconds << " sec." << endl;

cout << "Наивная реализация" << endl;

t1 = std::chrono::high\_resolution\_clock::now();

Naive\_solution(P);

t2 = std::chrono::high\_resolution\_clock::now();

seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Time: " << seconds << " sec." << endl << endl;

}

return result;

}