МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «Програмна інженерія та інформаційні технології управління»

Звіт з лабораторної роботи №7

З предмету «Алгоритми та структури даних»

Виконав Студент групи КН-36а

Рубан Ю.Д.

Перевірила:

ас. Бородіна І. О.

Харків 2017

ГЕОМЕТРИЧНІ АЛГОРИТМИ

Мета роботи: познайомитися із основними геометричними алгоритмами.

Завдання:

Розробити програму, яка читає з клавіатури число N (1 < N < 256) та N пар дійсних чисел — координати точок на площині. Програма виконує один за алгоритмів згідно варіанту.

Варіант завдання:

Побудувати опуклу оболонку наданих точок алгоритмом Грехема.

Хід виконання роботи:

Розроблено програму, яка читає з клавіатури число N (1 < N < 256) та N пар дійсних чисел — координати точок на площині. Програма виконує один за алгоритмів згідно варіанту.

Код програми:

```
#include<iostream>
using namespace std;
void swap(double&a, double&b);

struct  dot
{
        double x;
        double y;
        dot() {}
        dot(double a, double b) { x = a; y = b; }
        friend istream&operator >> (istream&is, dot &A)
        {
                is >> A.x >> A.y;
                return is;
        }
        friend ostream&operator << (ostream&os, dot &A)
        {
                os << "(" << A.x << "; " << A.y << ") ";
                return os;
        }
        static dot vector(dot &a, dot &b)
        {
                dot res(0,0);
                res.x = b.x - a.x;
                res.y = b.y - a.y;
                return res;
        }
};
double rotate(dot &a, dot &b, dot &c);
class lList
{
private:
        lList*head = 0;
        lList*tail = 0;
        lList*next = 0;
        lList*prev = 0;
        int count = 0;
        int data;
public:
        lList() {}
        void add(int x)
        {
                lList * temp = new lList;
                temp->data = x;
                if (head == NULL)
                {
```

```
                        head = tail = temp;
                }
                else
                {
                        temp->prev = tail;
                        tail->next = temp;
                        tail = temp;
                }
                count++;
        }
        void del(int x)
        {
                bool deleted = false;
                lList*temp = head;
                while (temp->data != x && temp->next!=NULL)
                {
                        temp = temp->next;
                }
                if (temp->next != NULL && temp->prev!=NULL)
                {
                        temp->prev->next = temp->next;
                        temp->next->prev = temp->prev;
                        deleted = true;
                }
                else if (temp->next == NULL && temp->prev != NULL)
                {
                        temp = temp->prev;
                        temp->next = NULL;
                        deleted = true;
                }
                else if (temp->next != NULL && temp->prev == NULL)
                {
                        temp = temp->next;
                        temp->prev = NULL;
                        head = temp;
                        deleted = true;
                }
                if (deleted) { count--; }

        }
        void delLast()
        {
                lList*temp = tail;
                temp = temp->prev;
                temp->next = NULL;
                tail = temp;
                count--;
        }
        int operator[](int index)
        {
                if (index >= 0)
                {
                        lList * temp = head;
                        for (int i = 0; i < index; i++)
                        {
                                temp = temp->next;
                        }
                        return temp->data;
                }
                else
                {
                        if (index == -1) { return tail->data; }
                        if (index == -2) { return tail->prev->data; }
                }
        }
        int getCount() { return count; }
};

int main()
{
        cout << "enter number of dots" << endl;
        int n;
        cin >> n;
        int *A = new int[n];
        dot* d = new dot[n];
        cout << "enter this dots (x;y)" << endl;
        for (int i = 0; i < n; i++)
```

```
	{
		cin >> d[i];
		A[i] = i;
	}
	int minX = d[0].x;
	int index=0;
	for (int i = 0; i < n; i++) //1 этап - найти минимальный x => самую левую точку
	{
		if (minX > d[i].x)
		{
			minX = d[i].x; index = i;
		}
	}
	swap(A[0], A[index]);//Конец первого этапа
	for (int i = 2; i < n; i++) //2 этап - отсортировать по "левизне" точки где последняя точка самая
левая относительно нулевой
	{
		int j = i;
		while(j>0 && rotate(d[A[0]], d[A[j - 1]], d[A[j]]) < 0)
		{
			swap(A[j - 1], A[j]);
			j--;
		}
	}//Конец второго этапа
	lList S;//3 этап - рассмотреть положение всех точек относительно всех точек
	S.add(A[0]);
	S.add(A[1]);
	for (int i = 2; i < n; i++)
	{
		while (rotate(d[S[-2]],d[S[-1]],d[A[i]])<0)//если точка d(A[i]) правее вектора (...), то
убрать точку d[S[-1]](последнюю добавленую), так как она не есть элементом МВО
		{
			S.delLast();
			if (i > S.getCount()) { break; }
		}
		S.add(A[i]);

	}
	cout << endl;
	cout << "res is:" << endl;
	for (int i = 0; i < S.getCount(); i++) //вывод последовательности вершин
	{
		cout << d[S[i]] << endl;
	}
	cout << endl;
	system("pause");
	return 0;
}
void swap(double&a, double&b)
{
	double temp = a;
	a = b;
	b = temp;
}
double rotate(dot &a, dot &b, dot &c) // функция сравнения двух векторов. Если результат <0, то точка C
левее вектора ab и наоборот
{
	dot ab = dot::vector(a, b);
	dot bc = dot::vector(b, c);
	return (ab.x*bc.y) - (ab.y * bc.x);
}
```

Висновки:

У даній лабораторній роботі я познайомився з одним із основних геометричних алгоритмів – побудова мінімальної опуклої оболонки методом Грехема