**10)**  Определить АТД Полином, обеспечивающий метод calc для вычисления значения полинома в точке x𝑥 (используйте [схему Горнера или барицентрическую форму интерполяционного многочлена Лагранжа](https://ipc.susu.ru/51495.html)).

Реализовать полином через представление на значениях в точках. В конструкторе задается набор значений y0,...,yn−1𝑦0,...,𝑦𝑛-1, x0𝑥0 и ΔxΔ𝑥 (xi=x0+i⋅Δx𝑥𝑖=𝑥0+𝑖⋅Δ𝑥). Определить операцию +.

#include <iostream>

#include <vector>

using namespace std;

class Polynom

{

private:

vector<double> y;

double x0;

double deltaX;

public:

// Конструктор с заданием значений y, начальной точки x0 и шага dx

Polynom(const vector<double>& values, double x0, double deltaX)

: y(values), x0(x0), deltaX(deltaX) {}

// Метод для вычисления значения полинома в точке x

double calc(double x) const{

size\_t n = y.size();

if (n == 0){

throw runtime\_error("Многочлен не имеет значений");

}

double result = 0.0;

double term = 1.0

for (size\_t i = 0; i < n; i++){

result+=y[i]\*term;

term \*=(x-(x0+i\*deltaX));//хранение текущего значения множителя на шаге цикла

}

return result;

}

// Операция сложения двух полиномов

Polynom operator+(const Polynom& other) const

{

// Находим степень полинома, являющегося суммой двух слагаемых полиномов

int n = max(y.size() - 1, other.y.size() - 1);

// Новый начальная точка

double new\_x0 = min(x0, other.x0);

// Новый шаг

double new\_deltaX = min(deltaX, other.deltaX);

// Кол-во точек в новом полиноме

size\_t new\_size = static\_cast<size\_t>(n + 1);

// Новый вектор значений y

vector<double> new\_y(new\_size);

// Вычисление значений нового полинома в точках

for (size\_t i = 0; i < new\_size; i++){

double x = x0 + i \* new\_deltaX;

new\_y[i] = this->calc(x) + other.calc(x);

}

return Polynom(new\_y, new\_x0, new\_deltaX);

}

};

**Задание 4 исправленное**) Предложите структуры данных для представления АТД из задания 3. Перечислите поля, их типы и комментарии к каждому полю. Укажите оценку эффективности (амортизированную или среднюю) для каждого метода с учетом использованных структур данных. Хранимая в структуре информация не должна дублироваться.

*Структуры данных и их поля:*

Таблица StolbName: vector<string>— Массив названий столбцов

Stroki: vector<vector<string>> — Массив массивов строк, где каждая строка представляет собой массив значений.

StolbKey: string — Название столбца, по которому осуществляется сортировка.

indexStolbKey: int — Индекс ключевого столбца.

SortedIndex: vector<int> — Отсортированный массив индексов строк для итерации по ключевому столбцу.

GetColumnName(i: int) -> string — Имя таблицы

GetColumnIndex(StolbName: string) -> int — индекс стобца по имени

*Оценка эффективности методов:*

CreateTable:O(1)

Stradd:O(n) –n- количество строк в таблице

StrDel:O(n) - -//-

StrGet:O(n)

ValueUpd:O(1)

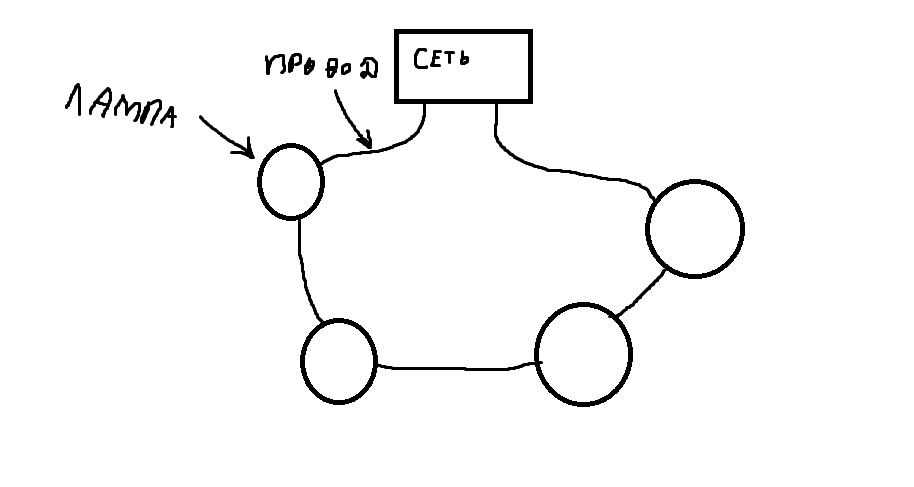
SetupSortKey:O(nlogn)

**17)** Укажите какой алгоритм нужно использовать для решения задачи и обоснуйте свой выбор. Из каких вершин и ребер граф будет состоять?  
Джон решил украсить дом иллюминацией к Рождеству. Он соединил проводами несколько ламп и подключил их к электросети. Затем Джон повернул рубильник, но некоторые лампы не загорелись. Напишите программу, которая определит, используя информацию о выполненных соединениях, какие лампы будут гореть.  
В первой строке ввода содержатся два целых числа, разделенных пробелом – количество ламп N𝑁 (1≤N≤1001≤𝑁≤100) и количество проводов K𝐾 (1≤K≤10001≤𝐾≤1000). Далее следует K𝐾 строк, в каждой строке сначала указывается номер элемента (0 для электросети, от 1 до N𝑁 для лампы), к которому был присоединен один из концов провода, потом через пробел – номер контакта 1 или 2 (все элементы имеют два контакта), затем указывается элемент и контакт, к которому был присоединен другой конец провода, в том же формате.  
В выходной файл в первой строке вывести N𝑁 целых чисел, разделяя их пробелами – если i𝑖-ая лампа будет гореть, то i𝑖-ое число должно быть равно 1, иначе 0. Лампа будет гореть, только если ее контакты подсоединены к вершинам с различными потенциалами. Провод имеет нулевое сопротивление, поэтому разность потенциалов на вершинах, соединенных между собой проводами без нагрузки в виде других ламп будет равна 0

Структура графа:

Вершины представляют собой лампу и электросеть с двумя контактами у каждого элемента

Ребра представляют собой провода между лампами и электросетью



Так как лампы будут гореть только если их контакты подсоединены к вершинам с различными потенциалами, то задача сводится к определению компонент связности и проверке, принадлежат ли контакты одной лампы к разным компонентам.

Для решения можно воспользоваться алгоритмом DFS(поиск в глубину), так как он будет обходить все вершины на максимальную глубину, что выгодно для задачи, где лампы подсоединены последовательно

**24)** Напишите функцию для нахождения стороны выпуклого многоугольника, ближайшей к его центру тяжести. В качестве параметра функции передается вектор координат вершин многоугольника против часовой стрелки. Многоугольник является сплошной фигурой.

Для точки использовать [класс из лекций и его методы](https://ipc.susu.ru/51512.html).

struct Point {

double x, y;

double len() const { return hypot(x, y); } // расстояние от начала координат

double phi() const { return atan2(y, x); } // угол

Point operator+(Point p) const { return {x + p.x, y + p.y}; }

Point operator-(Point p) const { return {x - p.x, y - p.y}; }

double operator\*(Point p) const { return x \* p.x + y \* p.y; } // скалярное произведение

double operator^(Point p) const { return x \* p.y - y \* p.x; } // векторное произведение

Point operator\*(double a) const { return {a \* x, a \* y}; } // "масштабирование"

Point turn(double a) const { double ca = cos(a), sa = sin(a); return {x \* ca - y \* sa, -x \* sa + y \* ca}; } // поворот

Point turn() const { return {-y, x}; } // поворот на п/2

Point operator-() const { return {-x, -y}; } // поворот на п

};

// Функция для нахождения центра тяжести многоугольника

Point findCentroid(const vector<Point>& vertices) {

Point centroid = {0, 0};

for (const Point& vertex : vertices) {

centroid = centroid + vertex;

}

centroid = centroid \* (1.0 / vertices.size());//среднее значение коорд всех вешин

return centroid;

}

// Функция для нахождения расстояния от точки до прямой

double distanceToLine(Point p, Point a, Point b) {

// Векторное произведение

double area = abs((b - a) ^ (p - a));

double base = (b - a).len();

return area / base;

}

// Функция для нахождения ближайшей стороны к центру тяжести

pair<Point, Point> findClosestSide(const vector<Point>& vertices) {

Point centroid = findCentroid(vertices);

double minDistance = numeric\_limits<double>::infinity();//установил начальное значение на бесконечность, но наверное, можно было бы и просто поставить какое-либо большое число

pair<Point, Point> closestSide;

for (size\_t i = 0; i < vertices.size(); ++i) {

Point a = vertices[i];

Point b = vertices[(i + 1) % vertices.size()];//если текущая вершина последняя в массиве, то след вершиной будет самая первая, то есть получим замкнутый многоугольник

double distance = distanceToLine(centroid, a, b);

if (distance < minDistance) {

minDistance = distance;

closestSide = {a, b};

}

}

return closestSide;

}

И вопрос по структуре, Point turn(double a) const { double ca = cos(a), sa = sin(a); return {x \* ca - y \* sa, -x \* sa + y \* ca}; }  
Почему для поиска поворота по оси ординат мы считаем по формуле –x\*sina+y\*cosa, у нас же тогда поворот будет по часовой стрелке, хотя обычно поворты идут против часовой стрелки, почему тут именно так сделано?