Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Фундаментальные науки
КАФЕДРА Прикладная математика

ОТЧЕТ ПО ПРАКТИКЕ

Студент Акиньшин Никита Олегович
фамилия, имя, отчество

Группа ФН2-31Б

Тип практики: Ознакомительная практика

Название предприятия: Научно-учебный комплекс «Фундаментальные науки» МГТУ им. Н.Э. Баумана

Студент _____ Акиньшин Н.О.
подпись, дата *фамилия и.о.*

Руководитель практики _____ Марчевский И.К.
подпись, дата *фамилия и.о.*

Оценка _____

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Кафедра «Прикладная математика»

З А Д А Н И Е
на прохождение ознакомительной практики

на предприятии Научно-учебный комплекс «Фундаментальные науки» МГТУ им. Н.Э. Баумана

Студент Акиньшин Никита Олегович
фамилия, имя, отчество

Во время прохождения ознакомительной практики студент должен

1. Изучить на практике основные возможности языка программирования C++ и систем компьютерной алгебры, закрепить знания и умения, полученные в курсах «Введение в информационные технологии», «Информационные технологии профессиональной деятельности».
2. Изучить способы реализации методов построения минимальной выпуклой оболочки множества точек на плоскости.
3. Реализовать алгоритм заворачивания подарка (gift wrapping algorithm).

Дата выдачи задания «26» сентября 2023 г.

Руководитель практики _____ Марчевский И.К.
подпись, дата *фамилия и.о.*

Студент _____ Акиньшин Н.О.
подпись, дата *фамилия и.о.*

Содержание

Задание

Набор точек на плоскости задан парами своих координат.

Требуется построить выпуклую оболочку данного множества точек – т.е. выпуклый многоугольник наименьшей площади, содержащий все эти точки. В качестве ответа привести список точек по порядку (по часовой стрелке или против часовой стрелки), задающих многоугольник, являющийся границей выпуклой оболочки.

а) решить задачу «методом перебора», последовательно находя такие прямые, проходящие через пары точек, что все остальные точки лежат по одну сторону от этих прямых;

б) решить задачу эффективно, используя алгоритм заворачивания подарка (gift wrapping algorithm), известный также как алгоритм Джарвиса (Jarvis).

Сравнить результаты обоих алгоритмов для малого числа точек; оценить сложности обоих алгоритмов в зависимости от размерности задачи.

В исходном файле данных записны количество точек n в первой строке, и n строк из координат точек вида x_i y_i.

В файле результата записаны количество точек, задающих многоугольник, являющийся границей выпуклой оболочки, q в первой строке, и q строк из координат точек вида x_i y_i.

Введение

Основной целью ознакомительной практики 3-го семестра, входящей в учебный план подготовки бакалавров по направлению 01.03.04 – Прикладная математика, является знакомство с особенностями осуществления деятельности в рамках выбранного направления подготовки и получение навыков применения теоретических знаний в практической деятельности.

В ранее пройденном курсе «Введение в специальность» произошло общее знакомство с возможными направлениями деятельности специалистов в области прикладной математики и получен опыт оформления работ (реферата), который полезен при оформлении отчета по практике.

В рамках освоенного курса «Введение в информационные технологии» изучены основные возможности языка программирования C++ и сформированы базовые умения в области программирования на C++. Задачей практики является закрепление соответствующих знаний и умений и овладение навыками разработки программ на языке C++, реализующих заданные алгоритмы. Кроме того, практика предполагает формирование умений работы с системами компьютерной алгебры и уяснение различий в принципах построения алгоритмов решения задач при их реализации на языках программирования высокого уровня (к которым относится язык C++) и на языках функционального программирования (реализуемых системами компьютерной алгебры).

1. Алгоритм перебора

1.1. Описание алгоритма перебора

Самым простым и очевидным алгоритмом для решения задачи определения минимальной выпуклой оболочки для множества точек является алгоритм перебора. Он заключается в попарном переборе точек, являющихся концами отрезка, лежащего на прямой, и проверки условия, что все точки, не лежащие на этой прямой, лежат по одну сторону от нее. Алгоритм перебора можно описать следующим образом:

1. Цикл *for* по i от 1 до n, где n – количество точек.
2. Выбираем точку P_i.
3. Цикл *for* по j от $i+1$ до n.
4. Выбираем точку P_j.
5. Цикл *for* по k от 1 до n.
6. Выбираем точку P_k. Проверяем с какой стороны лежит точка относительно отрезка P_iP_j.
7. Конец цикла (5).
8. Если все точки лежат по одну сторону от прямой, содержащей отрезок P_iP_j, то он подходит.

Асимптотическая сложность равна $O(n^3)$, где n – количество точек.

1.2. Реализация алгоритма перебора на C++

Сначала происходит чтение данных из исходного файла в *std::vector<Point> points*, где *Point* – структура, хранящая координаты точки на плоскости. Далее происходит поиск точек с минимальной и максимальной координатами по оси Ox.

После этого происходит разделение точек по условию нахождения над или под прямой с граничными по абсциссе точками. Точки, которые находятся над

или на прямой записываются в *std::vector<Point> above_points*, остальные же точки – в *std::vector<Point> below_points*. Далее *above_points* сортируем алгоритмом быстрой сортировки по возрастанию координаты *x, below_points* сортируем таким же образом, но по убыванию координаты *x*. Это необходимо для обеспечения условия обхода точек выпуклой оболочки по часовой стрелке.

Для каждого вектора, полученного из точек, используем переборный алгоритм, который принимает вектор из точек *std::vector<Point> local_points*, в которых нужно выбрать оболочку, и все множество точек *std::vector<Point> all_points*. Его можно описать так:

1. Цикл *for* по $i = 0$ до *N1*, где *N1* – длина *local_points*.
2. Фиксируем точку *Point p1 = local_points[i]*.
3. Цикл for по $j = i+1$ до *N1*.
4. Фиксируем точку *p2 = local_points[j]*.
5. Если функция *check_segment* для *p1* и *p2* истинна, то отрезок *p1p2* подходит, тогда добавляем *p2* в *std::vector<Point> res* и прерываем цикл(3), иначе – переходим к следующей точке.
7. Конец цикла (3).
8. Конец цикла (1).

Опишем функцию *check_segment*, которая на вход принимает *Point& p1, Point p2, vector<Point> points*:

1. Инициализируем переменные: *ArePreviousBeenUpper* = 1, отвечающая за положение всех предыдущих точек относительно прямой *p1p2*, *isUpperThanLine* = 1, отвечающая за положение текущей точки относительно прямой, содержащей отрезок *p1p2*.
2. Цикл *for* по $i = 1$ до *N*, где *N* – длина вектора *points*.
3. Объявляем переменную $x = (points_i)_x$.
4. Если ($p1_x == p2_x$ и $x == p1_x$), то переходим к следующей итерации.

5. *isUpperThanLine* = ($p1_x$ == $p2_x$ И $x > p1_x$), т.е. считаем, что точка находится над прямой, если прямая вертикальная и точка правее.
6. Вычисляем значение *y* на прямой в точке, соответствующей по абсциссе текущей, и определяем значение *isUpperThanLine* = 1, если $y > p1_y$. Иначе – *isUpperThanLine* = 0.
7. Если (*i* == 0 И *isUpperThanLine* == 0), то необходимо переопределить *ArePreviousBeenUpper* = 0.
8. Если (*isUpperThanLine != ArePreviousBeenUpper*), то отрезок не подходит, возвращаем 0.
9. Конец цикла (2). Возвращаем 1.

Замер времени для случайно составленного набора входных данных:

1. 100 точек – 0.003 с
2. 1000 точек – 0.023 с
3. 10000 точек – 1.363 с

1.3. Реализация алгоритма перебора в системе компьютерной алгебры «Wolfram Mathematica»

При реализации алгоритма перебора в системе компьютерной алгебры «Wolfram Mathematica» сначала производим чтение данных из исходного файла в список points. Далее сортируем этот список по возрастанию абсцисс точек. Делим его на списки *aboveline*, *belowline* с помощью команды *Cases*, где *aboveline* – список точек, которые находятся либо на прямой, проходящей через две крайние по абсциссе точки, либо над ней. В свою очередь, *belowline* – список точек, находящихся под этой прямой. Применяем команду *Reverse* к списку и добавляем в его конец начальную точку *aboveline*.

Далее выполняется алгоритм перебора для каждого списка. Для его реализации была написана пользовательская функция *MyEnumerate*, которая

принимает на вход локальный список точек и все множество точек. Сначала мы создаем все возможные отрезки с помощью *Table*, далее применяем *Flatten*, потом последовательно выполняется *Flatten* с параметром 2. Среди этих отрезков, используя команду *Cases*, отбираем такие, которые удовлетворяют результату пользовательской функции *MyCheckPoints*.

Прежде, чем рассматривать функцию *MyCheckPoints*, рассмотрим другую пользовательскую функцию – *MyCheckPoint*. Она принимает на вход концы рассматриваемого отрезка p_1, p_2 и произвольную точку p. Она проверяет, с какой стороны находится данная точка относительно отрезка p_1p_2. Реализовано аналогично проверке точки в функции *check_segments* на C++ из предыдущего пункта. Возвращает 1, если точка лежит над отрезком p_1p_2, иначе – 0.

MyCheckPoints принимает на вход две точки p_1 и p_2, а также все множество точек *points*. Для начала создаем список *positions*, состоящий из значений функции *MyCheckPoint* для каждой точки из *points*. Далее заменяем заголовок *positions* на *Equal*, что дает результат итерационного применения *Equal* к каждой паре точек. Таким образом, отрезок подходит, т.е. функция возвращает *True*, если все точки находятся по одну сторону от p_1p_2, иначе – *False*.

Замер времени для случайно составленного набора данных:

1. 100 точек – 1.75 с
2. 200 точек – 31.23 с

2. Алгоритм заворачивания подарка

2.1. Описание алгоритма заворачивания подарка

Алгоритм заворачивания подарка или же алгоритм Джарвиса является более быстрым вариантом решения задачи нахождения минимальной выпуклой оболочки для заданного множества точек. Его можно описать так:

1. Берем самую нижнюю точку.

2. Цикл пока последняя выбранная точка не равна выбранной на предыдущем шаге.
3. Ищем такую точку, которая образует относительно последнего элемента в списке выбранных точек минимально полярный угол (рис. 2.1). Добавляем ее в этот список.

По итогу получаем список из точек оболочки. Алгоритм основан на идее проведения прямых по полярному углу от 0 радиан до пересечения с одной точкой на множестве для точки p_i.

Асимптотическая сложность равна *O(nh)*, где *h* – количество точек на оболочке, а *n* – количество точек.

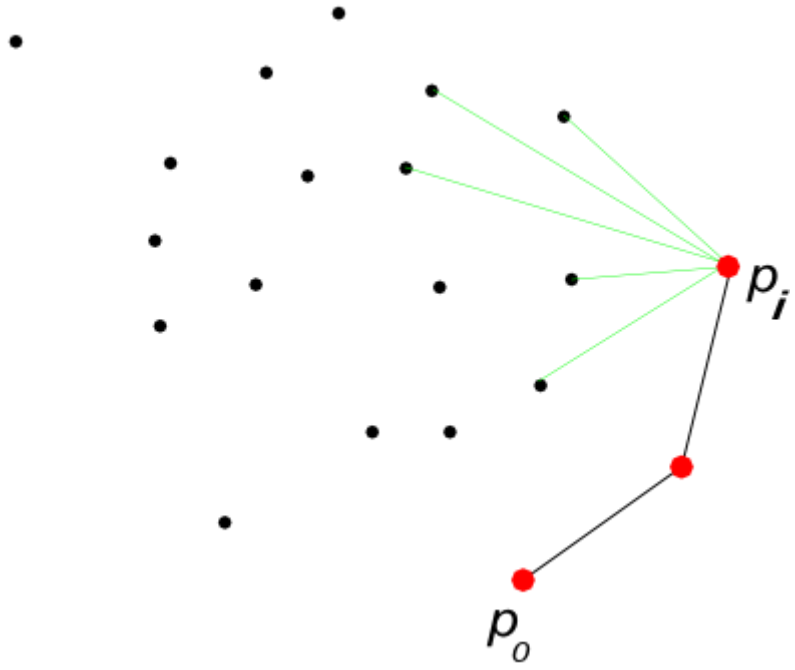

Рис. 2.1. Наглядный пример работы алгоритма Джарвиса

2.2. Реализация алгоритма заворачивания подарка на C++

При реализации алгоритма заворачивания подарка на C++ сначала считаем данные из исходного файла в *std::vector<Point> points*, где *Point* – структура, хранящая координаты *x*, *y* точки.

Сначала получаем номер самой нижней точки, то есть точки с минимальным значением *y*.

Создаем два вектора: *std::vector<Point> res и std::vector<int> res2*, где *res* – вектор, хранящий точки оболочки, а *res2* – вектор, хранящий номера точек оболочки.

Добавляем в *res* самую нижнюю точку. Рассмотрим работу следующего цикла:

1. Цикл *do while* с условием последний элемент res2 не совпадает с первым элементом *res2*.
2. Находим номер точки с минимальным значением косинуса с помощью функции *get_min_cos_point*, т.е. с наибольшим минимальным углом между отрезками, что гарантирует наименьший полярный угол.
3. Добавляем эту точку в *res* и номер в *res2*.

Рассмотрим работу функции *get_min_cos_point*. На вход она принимает все точки множества *vector<Point> points*, точку *Point p*, относительно которой необходимо найти другую точку с минимальным значением косинуса угла между двумя соседними отрезками оболочки. Номер этой точки – *int id_p*, номер предыдущей точки оболочки – *id_lastp*. Если у точки *p* нет соседней точки, принадлежащей оболочке, то считаем $id\_lastp = -1$.

Проходимся по $i = 0$ до $N-1$, где N – длина вектора *points*. Находим косинус между векторами, один из которых является вектором $points_{id\_lastp}points_{id\_p}$, другой вектор состоит из $points_i points_{id\_p}$. Если же $id\_lastp = -1$, то считаем, что вектор направлен горизонтально в отрицательную сторону отсчета оси *x*.

Минимальный косинус будем искать как косинус угла между двумя векторами по формуле (1) [4]:

$$\cos(\widehat{\vec{a},\vec{b}}) = \frac{(\vec{a},\vec{b})}{||\vec{a}||||\vec{b}||}, \quad (1)$$

где $\vec{a}$, $\vec{b}$ – векторы из R^n, (a, b) – скалярное произведение в R^n, $\|\vec{a}\|, \|\vec{b}\|$ – нормы векторов $\vec{a}$, $\vec{b}$ в R^n.

Таким образом получаем массив из точек, образующих выпуклую оболочку, упорядоченных против часовой стрелки.

Произведём замер времени для различного количества входных данных:

1. 1000 элементов – 0.003 с.
2. 10000 элементов – 0.031 с.
3. 100000 элементов – 0.24 с.

2.3. Реализация алгоритма заворачивания подарка в системе компьютерной алгебры «Wolfram Mathematica»

При реализации алгоритма заворачивания подарка в системе компьютерной алгебры «Wolfram Mathematica» сначала читаем данные из исходного файла в список *points*. Далее находим самую нижнюю точку и добавляем ее в список *res*. Дальше добавляем точку, образующую максимальный угол с самой нижней точкой с помощью пользовательской функции *GetMaxAnglePoint*, которая принимает на вход все множество точек *pts*, текущую точку *pt*, и предыдущую соседнюю на оболочке точку *lastpt*.

Рассмотрим работу функции *GetMaxAnglePoint* подробнее*:* сначала с помощью команды Cases избавляемся от точек, которые равны *lastpt* или *pt*, следующим шагом создаем список из всех возможных векторов, связанных с точкой *pt*. С помощью команды *Table* и команды *VectorsAngle* находим углы между векторами, один из которых связан с *pt* и *lastpt*, другой – взят из списка возможных векторов, таким образом получаем список angles, в котором на первой позиции элемента стоит угол, а на второй – выбранная точка. Удаляем из

этого списка углы, равные бесконечности, а затем сортируем его по убыванию углов. Тогда второе значение первого элемента *angles* и есть выбранная точка, которая соответствует максимально возможному наименьшему углу между отрезками, состоящими из *lastpt* и *pt*, а также *pt* и данной точки.

Таким образом получаем список из точек выпуклой оболочки. Произведём замер времени для различного количества входных данных:

1. 1000 точек – 0.25 с
2. 10000 точек – 2.43 с

Заключение

В ходе решения поставленной задачи были изучены и использованы основные возможности языка программирования C++ и системы компьютерной алгебры «Wolfram Mathematica». Также удалось закрепить знания и умения, полученные при прохождении учебных курсов «Введение в информационные технологии», «Информационные технологии профессиональной деятельности», изучить алгоритм перебора и заворачивания подарка (алгоритм Джарвиса) для построения выпуклой оболочки для множества точек.

Список используемых источников

1. Выпуклые оболочки // Алгоритмика.
   URL: https://algorithmica.org/ru/convex-hulls (Дата обращения: 10.12.2023)
2. Gift wrapping algorithm // Wikipedia
   URL: https://en.wikipedia.org/wiki/Gift_wrapping_algorithm (Дата обращения: 10.12.2023)
3. Wolfram Language & System. Documentation Center.
   URL: https://reference.wolfram.com/language/ (Дата обращения: 10.12.2023)
4. Векторная алгебра и аналитическая геометрия. Методические указания к решению задач // Э.И. Агаева, Р.Ф. Сперанская; 2-е изд. Москва: Издательство МГТУ им. Н.Э. Баумана, 2017г.