|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
| МИНОБРНАУКИ РОССИИ | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | |
| **Институт искусственного интеллекта** | | |  |
| **Кафедра высшей математики** | | |  |

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| **по дисциплине** | |
| **«**Алгоритмы и теория сложности**»** | |
| **Тема курсовой работы**  **«Алгоритм Грэхэма»** | |
| Студент группы КМБО-07-22 | *Баттур Ц.* |
| Руководитель курсовой работы | *Драгилева И.П.* |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2024 г. | *(подпись студента)* |
|  |  |  |
| «Допущен(ы) к защите» | «\_\_»\_\_\_\_\_\_\_2024г. | *(подпись руководителя)* |

Москва – 2023

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | | **Институт искусственного интеллекта** | | | | **Кафедра высшей математики** | | | | | | |  |
|  | | **Утверждаю** | | |
|  | | Заведующий  кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_Шатина А.В. | | |
|  | | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2024г. | | |
| **ЗАДАНИЕ** | | | | |
| **на выполнение курсовой работы** | | | | |
| **по** **дисциплине** «Алгоритмы и теория сложности» | | | | |
|  | | | | |
| Студент Баттур Ц. Группа *КМБО-07-22* | | | | |
|  | | | | |
| 1. **Тема: «Алгоритм Грэхэма»** | | | | |
| **2. Исходные данные:** тестовые примеры и наборы данных для испытаний | | | | |
| **3**. **Перечень вопросов, подлежащих обработке, и обязательного графического материала:**  1) Алгоритм(ы) решения задачи на языке высокого уровня.  2) Получение и сравнение количественных результатов при различных исходных данных.  3) Асимптотические оценки временной сложности. | | | | |
|  | | | | |
| **4. Срок представления к защите курсовой работы:** **до** «23 » декабря 2024 г. | | | | |
|  | | | | |
| Задание на курсовую  работу выдал | « 1» октября 2024г . | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (Драгилева) | |
| Задание на курсовую  работу получил | «\_\_\_»\_\_\_\_\_\_2024г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (Баттур) | |

**Оглавление**

Введение

Постановка Задачи

Описание Алгоритма Грэхэма

Доказательство Корректности

Асимптотическая Оценка Временной Сложности Алгоритма

Результаты Запуска Программы

Заключение

Список Литературы

Приложения

**Введение**

Выпуклой оболочкой множества *X*называется наименьшее выпуклое множество, содержащее *X*. Наименьшее множество здесь означает наименьший элемент по отношению к вложению множеств, то есть такое выпуклое множество, содержащее данную фигуру, что оно содержится в любом другом выпуклом множестве, содержащем данную фигуру. Это концепция в вычислетельной геометрии, которая находит применение в различных областях, как компьтерная графика, обработка изображений и обнаружение столкновений.

Выпуклый многоугольник – это многоугольник, у которого все внутренние углы меньше 180 градусов. Выпуклую оболочку можно построить для любого набора точек, независимо от их расположения.

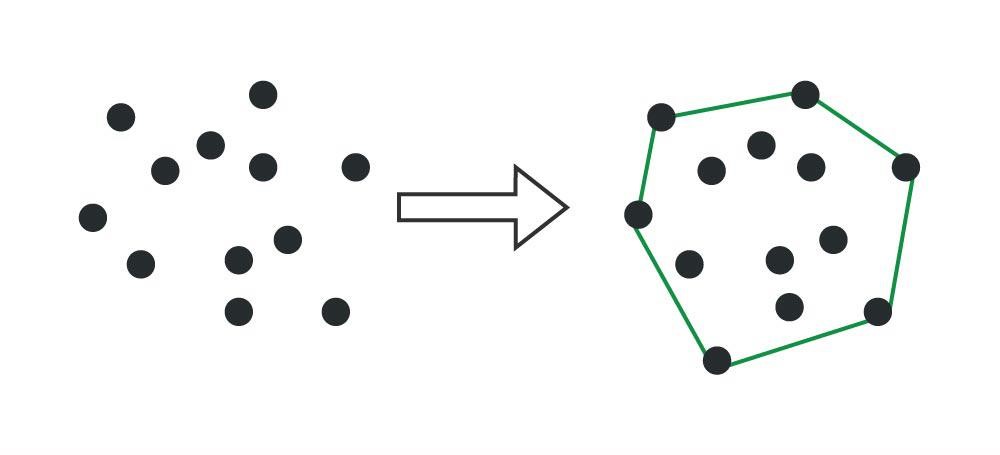


Рисунок 1. Выпуклая оболочка

**Постановка Задачи**

* 1. Построение минимальной выпуклой оболочки на плоскости
     1. Тестовый пример: 8 точек с координатами (1, 1), (2, 5), (1, 9), (4, 3), (6, 4), (5, 7), (8, 2), (8, 9)
     2. Набор данных испытания: равномерно выбрать 1000 точек из квадрате [-100, 100] x [-100; 100]
  2. Сравнить временные затраты для с затратами задач, использующих рекурсивный алгоритм и алгоритм Джарвиса
  3. Привести и обосновать асимптотическую оценку временной сложности алгоритма.

**Описание Алгоритма Грэхэма**

Алгоритм Грэхэма – это простой и эффективный алгоритм вычиления выпуклой оболочки множества точек на плоскости с временной сложностью. Он работает путем итеративного добавления точек к выпуклой оболочке до тех пор, пока не все точки не добавлены.

Основные шаги добавления:

1. Поиск опорной точки: Выбтрается токи с наименьшей координатой y. Эта точка всегда находится на выкуклой оболочке.
2. Сортировка точки по полярному углу: Все остальные точки сортируются относительно опорной точки по их полярному углу.
3. Построение выпуклой оболочки: Алгоритм итеративно добавляет точки к выпуклой оболочки. На каждом шаге алгоритм проверяет, образуют ли последние две точки, добавленные к выпуклой оболочки, правый поворот. Если это так, то последняя точка удаляется из выпуклой оболочки. В противном случае следующая точка в отсортированном списке добавляется к выпуклой оболочки.
4. После обработки всех точек в стеке остаются вершины выпуклой оболочки в порядке обхода.

Псевдокод алгоритма Грэхэма для построения выпуклой оболочки:

GrahamScan(S)

1. Найти точку P0 с наименьшей y-координатой (при равенстве — с наименьшей x-координатой)

2. Отсортировать остальные точки в S по возрастанию полярного угла относительно P0

3. Создать пустой стек Stack

4. Поместить P0, P1, P2 в Stack

5. Для i от 3 до n:

a. Пока размер Stack > 1 и поворот (NextToTop(Stack), Top(Stack), Pi) не является левым:

i. Удалить Top(Stack)

b. Добавить Pi в Stack

6. Возвратить Stack (вершины выпуклой оболочки в порядке обхода)

**Доказательства Корректности**

* Лемма о сортировке: Сортировка точек по полярному углу относительно P₀ гарантирует, что обход точек в отсортированном порядке соответствует обходу внешней границы множества точек.
* Монотонность стека: Алгоритм поддерживает инвариант, что вершины в стеке всегда образуют выпуклый многоугольник.
* Удаление внутренних точек: При обнаружении поворота направо удаляются точки, не принадлежащие выпуклой оболочке, что обеспечивает минимальность конечного результата.

**Асимптотическая Оценка Временной Сложности Алгоритма**

1. Поиск опорной точки: . Для нахождения такой точки необходимо пройти один раз по всем точкам.
2. Сортировка точек: . После выбора опорной точки все остальные точки сортируются по полярному углу относительно неё.
3. Обход и построение стека: . Стек используется для хранения вершин выпуклой оболочки, и каждая точка добавляется или удаляется из стека не более одного раза

Общая временная сложность складывается из указанных этапов и составляет , где наиболее затратным по времени является этап сортировки.

**Результаты Запуска Программы**

Три алгоритма были протестированы для построения выпуклой оболочки: Алгоритм Жарвиса, Алгоритм Грэхэма и Рекурсивный алгоритм для построения выпуклой оболочки. Первое тестирование проводилось на данных из 8 точек: (1, 1), (2, 5), (1 ,9), (4, 3), (6, 4), (5,7), (8, 2), (8, 9).

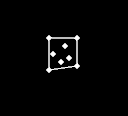


Рисунок 2. Выпуклая оболочка

Тестирование проводилось на тех же данных из 1000 точек для обеспечения корректного сравнения.

* Алгоритм Грэхэма: 1476 микросекундов
* Алгоритм Жарвиса: 386 микросекундов
* Рекурсивный Алгоритм: 880 микросекундов

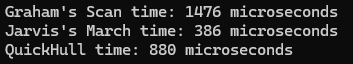


Рисунок 3. Результаты запуска алгоритмов

Анализ результатов:

* Алгоритм Жарвиса показал наилучшее время выполнения – 386 микросекундов. Его временная сложность - , где h – количество вершин выпуклой оболочки
* Рекурсивный Алгоритм занял 880 микросекундов. Он имеет хорошую производительность на случайных наборах точек благодаря разделяй и властвуй подходу.
* Алгоритм Грэхэма занял 1476 микросекундов, что является самым большим временем. Несмотря на теоретическую сложность в данном тесте он показал менее эффективный результат, возможно, из-за накладных расходов на сортировку или особенной реализации

**Заключение**

Алгоритм Грэхэма эффективно решает задачу построения выпуклой оболочки для множества точек на плоскости с временной сложностью . Он прост в реализации и демонстрирует высокую производительность на практике. Однако, как показали результаты тестирования, на некоторых наборах данных другие алгоритмы, такие как обход Джарвиса и Рекурсивный Алгоритм, могут показывать лучшую производительность. Выбор оптимального алгоритма зависит от характеристик входных данных, таких как размер набора и количество вершин выпуклой оболочки. Дополнительная оптимизация и адаптация алгоритмов к специфике задачи могут значительно повысить их эффективность.

**Список Литературы**

1. Convex Hull Using Graham Scan – URL: <https://www.geeksforgeeks.org/convex-hull-using-graham-scan/>
2. Выпуклая Оболочка – URL: <https://ru.wikipedia.org/wiki/Выпуклая_оболочка>
3. Построение Минимальных Выпуклых Оболочек – URL: <https://habr.com/ru/articles/144921/>
4. Algorithm and data structures – URL: <https://www.inf.ed.ac.uk/teaching/courses/ads/Lectures/lec16.pdf>

**Приложения**

std::vector<point> GrahamAlgo(std::vector<point> points) {

size\_t n = points.size();

if (n < 3) return points;

// Найти точку с минимальной y-координатой (и минимальной x при равенстве)

size\_t ymin = 0;

for (size\_t i = 1; i < n; i++) {

if ((points[i].y < points[ymin].y) || (points[i].y == points[ymin].y && points[i].x < points[ymin].x)) {

ymin = i;

}

}

// Переместить точку P0 в начало массива

std::swap(points[0], points[ymin]);

point p0 = points[0];

// Сортировка точек по полярному углу относительно P0

std::sort(points.begin() + 1, points.end(), [p0](point a, point b) {

int o = orientation(p0, a, b);

if (o == 0) return (std::hypot(p0.x - a.x, p0.y - a.y) < std::hypot(p0.x - b.x, p0.y - b.y));

return (o == 2);

});

// Удаление коллинеарных точек, оставляя самую дальнюю

size\_t m = 1; // Размер модифицированного массива

for (size\_t i = 1; i < n; i++) {

while (i < n - 1 && orientation(p0, points[i], points[i + 1]) == 0)

i++;

points[m] = points[i];

m++;

}

if (m < 3) return std::vector<point>(points.begin(), points.begin() + m);

// Инициализация стека выпуклой оболочки первыми тремя точками

std::vector<point> hull;

hull.push\_back(points[0]);

hull.push\_back(points[1]);

hull.push\_back(points[2]);

// Обход оставшихся точек и построение выпуклой оболочки

for (size\_t i = 3; i < n; i++) {

while (hull.size() > 1 && orientation(\*(hull.rbegin() + 1), hull.back(), points[i]) != 2) hull.pop\_back();

hull.push\_back(points[i]);

}

return hull;

}