Ивановский Сергей Алексеевич,
Преображенский Алексей Семенович,
Симончик Сергей Константинович

АЛГОРИТМЫ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ. ВЫПУКЛЫЕ ОБОЛОЧКИ: СВЯЗЬ С ЗАДАЧЕЙ СОРТИРОВКИ И ОПТИМАЛЬНЫЕ АЛГОРИТМЫ

1. СВЯЗЬ ЗАДАЧИ ПОСТРОЕНИЯ ВЫПУКЛОЙ ОБОЛОЧКИ С ЗАДАЧЕЙ СОРТИРОВКИ

Все рассмотренные в [1] алгоритмы построения выпуклой оболочки в качестве ответа задачи порождали упорядоченную последовательность крайних точек выпуклой оболочки.

Такое рассмотрение задачи естественным образом связывает ее с задачей сортировки. Действительно, необходимость получения упорядоченной последовательности крайних точек требует от алгоритма способности прямо или косвенно проводить сортировку. Очевидна связь задачи сортировки и алгоритма Грэхема, который начинает с того, что упорядочивает исходное множество точек по полярному углу. Перед тем как с этой точки зрения анализировать другие алгоритмы построения выпуклой оболочки, вспомним основные алгоритмы, решающие задачу сортировки.

Алгоритм сортировки – это алгоритм для упорядочения элементов множества с определенным отношением порядка на этом множестве. Можно выделить три основных метода сортировки. Это сортировка *вставками*, *выбором и обменом*.

Идея метода вставок *(insertion sort)* заключается в следующем: на каждом шаге алгоритма выбирается один из элементов входных данных и вставляется на нужную позицию в ранее отсортированной части последовательности, такие шаги повторяются до тех пор, пока набор входных данных не будет исчерпан (см. рис. 1.1). Сложность такого алгоритма при сортировке n элементов составляет $O(n^2)$.

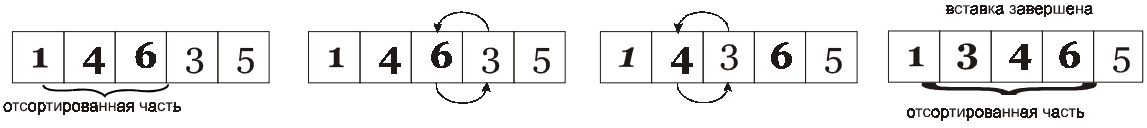


Рис. 1.1. Сортировка вставками. Вставка числа 3 в отсортированную последовательность (1, 4, 6)

Идея сортировки выбором *(selection sort)* заключается в поиске на каждом шаге наименьшего из ещё не упорядоченных элементов (при этом каждый из ранее упорядоченных элементов заведомо меньше каждого из еще не упорядоченных) и перемещении его в конец отсортированной части последовательности (см. рис. 1.2). Сложность простого алгоритма сортировки выбором для n элементов составляет $O(n^2)$ Усовершенствованная сортировка выбором, известная как пирамидальная сортировка *(Heap sort)*, имеет сложность $O(n\log n)$.

Обменная сортировка многократно использует сравнения и, если требуется, перестановку выбранной пары элементов. Вопрос в том, какие пары и в каком порядке выбираются для сравнения и перестановок. Простая обменная сортировка («пузырьковая») последовательно рассматривает соседние пары элементов. За один проход от начала к концу последовательности больший элемент пары продвигается в направлении конца последовательности, пока, в свою очередь, не наткнется на ещё больший, который продолжит продвигаться дальше (подобно всплывающему пузырьку). После нескольких таких проходов последовательность упорядочивается (см. рис. 1.3). Сложность пузырьковой сортировки есть $O(n^2)$.

Усовершенствованная обменная сортировка, известная как «быстрая сортировка» *(Quick sort)*, имеет в худшем случае сложность $O(n^2)$, однако в среднем – лишь $O(n\log n)$, при этом на практике быстрая сортировка работает значительно быстрее, чем другие алгоритмы с оценкой $O(n\log n)$. Идея алгоритма заключается в разделении исходного набора данных на две прилегающие части, так что любой элемент первой части упорядочен относительно любого элемента второй части; затем алгоритм применяется рекурсивно к каждой из частей.

Ещё два важных метода сортировки можно рассматривать как обобщение метода вставок. Во-первых, это сортировка слиянием (*Merge sort*). Основная идея этого алгоритма заключается в разбиении множества элементов на два подмножества примерно равной мощности, рекурсивной сортировке каждого из них, а затем слиянии результатов (слияние подобно вставке элементов одного упорядоченного множества в другое). Общее время работы сортировки слиянием для n элементов составляет $O(n\log n)$. Во-вторых, для сортировки можно использовать бинарное дерево поиска, подобная сортировка называется сортировкой двоичным деревом (*Binary tree sort*). Для этого в исходно пустое дерево поочередно добавляются все элементы множества, а затем производится *симметричный* обход [2] дерева поиска с выписыванием всех элементов множества в соответствии с отношением порядка. Время работы сортировки зависит от реализации бинарного дерева поиска и в случае использования сбалансированного дерева поиска [2] составляет $O(n\log n)$.

Поскольку все перечисленные алгоритмы сортировки универсальны, то есть ис-

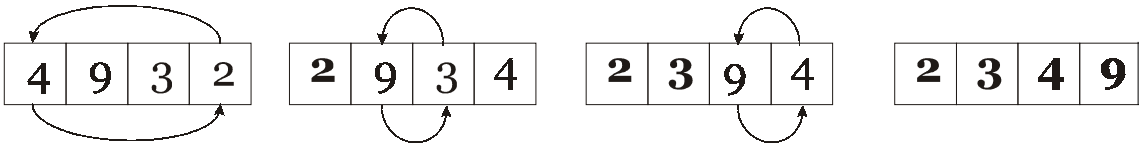


Рис. 1.2. Сортировка выбором

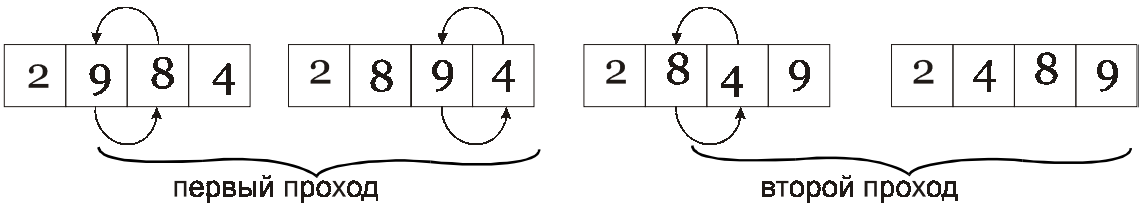


Рис. 1.3. «Пузырьковая» сортировка

пользуют только попарные сравнения элементов, не учитывая особенности их внутренней структуры, то, введя подходящую формальную модель (деревья решений), можно показать [2], что любая такая сортировка в худшем случае требует времени $\Omega(n \log n)$.

Возвратимся к связи между задачей построения выпуклой оболочки множества точек и задачей сортировки. Уместно заметить, что аналогия будет более строгой и полной, если все точки исходного множества являются вершинами выпуклой оболочки. Это вполне понятно, так как в задаче сортировки нет аналога для внутренних точек выпуклой оболочки, которые в итоге не включаются в результат.

Итак, вспомним основные шаги обхода Джарвиса [1] для точек $\{p_i\}_1^n$:

1. Найти заведомо крайнюю точку p_{start}, положить $p_{current} \leftarrow p_{start}$.

2. Найти точку p_{next}, следующую непосредственно за точкой $p_{current}$ в обходе выпуклой оболочки.

3. Аналогичным образом найти вершины выпуклой оболочки, полагая на каждом шаге $p_{current} \leftarrow p_{next}$.

Ясно, что алгоритм Джарвиса соответствует сортировке методом выбора: на каждом шаге выбирается точка, в некотором отношении наименьшая, и добавляется к выпуклой оболочке.

Аналогом пошагового алгоритма построения выпуклой оболочки в режиме «online» является сортировка методом вставок. Действительно, при вставке на i-ом шаге точки p_i для нее подбирается подходящая позиция (в результате нахождения опорных точек).

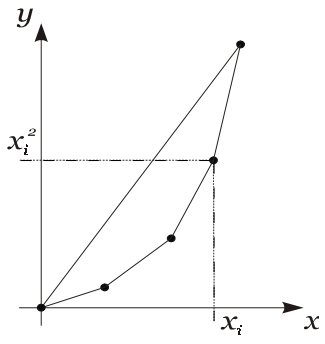


Рис. 1.4. Связь задачи сортировки с задачей построения выпуклой оболочки.

Как было указано ранее, для сортировки n элементов, вообще говоря, требуется не менее $\Omega(n \log n)$ действий или, иными словами, для любого алгоритма сортировки можно предъявить такие входные данные, что его применение потребует не менее $\Omega(n \log n)$ операций. В связи с этим говорят, что нижняя граница задачи сортировки равна $\Omega(n \log n)$. Можно ли установить нижнюю границу задачи построения выпуклой оболочки множества точек? Оказывается, что можно, и при этом существенно используется связь этой задачи с задачей сортировки. Для этого воспользуемся так называемым *преобразованием* задачи сортировки в задачу построения выпуклой оболочки.

Определим процедуру преобразования задач. Пусть имеются две задачи *A* и *B*, которые связаны так, что задачу *A* можно решить следующим образом:

1. Из исходных данных к задаче *A* сконструировать соответствующие исходные данные к задаче *B*.

2. Решить задачу *B*.

3. Из результата решения задачи *B* получить результат решения задачи *A*.

В этом случае будем говорить, что задача *A* преобразуема в задачу *B*.

Получим нижнюю границу времени построения выпуклой оболочки множества точек, основываясь на преобразовании задачи сортировки в задачу построения выпуклой оболочки. Для этого рассмотрим задачу сортировки последовательности из n действительных чисел $\{x_i\}_1^n$. Известно, что эту задачу нельзя решить асимптотически быстрее, чем за время $\Omega(n \log n)$. Рассмотрим взаимнооднозначное преобразование последовательности (входных данных задачи сортировки) во множество точек $\{x_i\}_1^n$ на плоскости $Q = \{(x_i, y_i)\}_1^n$ (во входные данные задачи построения выпуклой оболочки). Преобразуем каждый элемент x_i в точку $\{(x_i, x_i^2)\}$ (см. рис. 1.4).

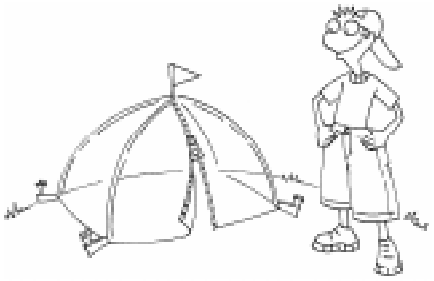
Затем решим задачу построения выпуклой оболочки множества Q за некоторое время $T(n)$. В результате, в силу выпуклости параболы $y = x^2$, построенная выпуклая оболочка будет состоять из всех точек множества Q. Один просмотр списка вершин выпуклой оболочки позволяет прочитать в нужном порядке значения x_i (для этого необходимо найти точку с минимальной абсциссой, а затем, начиная с нее, перечислить все вершины выпуклой оболочки против часовой стрелки). Таким образом, в построенном преобразовании задач общее время преобразования данных между задачами равно $O(n)$. С другой стороны мы знаем, что задача сортировки не может быть решена быстрее, чем за $\Omega(n \log n)$ шагов. Связав все воедино, получим, что $\Omega(n \log n) \le O(n) + T(n)$. Отсюда следует, что нижняя оценка времени построения выпуклой оболочки множества точек есть $\Omega(n \log n)$. Как мы видим, алгоритм Грэхема достигает нижней границы, то есть является асимптотически оптимальным.

С учетом вышеизложенного можно вернуться к рассмотрению пошагового алгоритма построения выпуклой оболочки [1] со сложностью $O(n^2)$ Возникает вопрос, можно ли улучшить этот алгоритм до достижения нижней границы? Разумно предположить, что такая цель может быть достигнута, если время решения задачи сортировки в режиме «online», в свою очередь, также будет достигать нижней границы задачи сортировки. Сортировка двоичным деревом полностью удовлетворяет поставленным условиям. Действительно, представляя последовательность вершин выпуклой оболочки с помощью сбалансированного дерева поиска [2], опорные точки можно найти за время $O(n \log n)$. Таким образом, задача построения выпуклой оболочки в режиме «online» может быть решена за время $O(n \log n)$.

Полученный результат позволяет говорить, что на самом деле любой алгоритм, строящий упорядоченную последовательность вершин выпуклой оболочки, независимо от реализуемого им метода, является замаскированным алгоритмом сортировки. Отметим также, что задача построения выпуклой оболочки преобразуется в задачу сортировки (фактически это преобразование осуществляется алгоритмом Грэхема).

Напомним, что алгоритм Джарвиса имеет сложность $O(nh)$, где h – число вершин выпуклой оболочки. Уместно поставить вопрос: каково среднее время работы алгоритма? В свою очередь, среднее время работы алгоритма зависит от математического ожидания величины h, различного при различных распределениях координат точек исходного множества. Оказывается [3], что при нормальном распределении точек алгоритм Джарвиса потребует в среднем $O(n\sqrt{\log n})$ времени, что меньше времени работы алгоритма Грэхема.

2. ДРУГИЕ АЛГОРИТМЫ ПОСТРОЕНИЯ ВЫПУКЛОЙ ОБОЛОЧКИ



До сих пор рассматривались алгоритмы построения выпуклой оболочки множества точек, а затем к ним находились аналоги из алгоритмов сортировки. Будем действовать обратным образом.

Обобщением известного алгоритма сортировки слиянием является алгоритм построения выпуклой оболочки, использующий метод «разделяй и властвуй». Суть алгоритма такова.

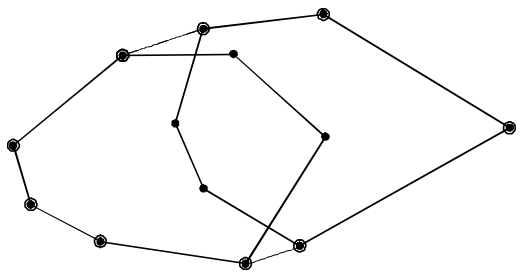


Рис. 2.1. Слияние выпуклых оболочек. Вершины результирующей оболочки выделены.

Если число заданных точек n меньше некоторой константы n_0, то оболочка вычисляется некоторым прямым методом (например, алгоритмом Джарвиса). Если же n больше n_0, то сначала множество из n точек произвольным образом разбивается на два подмножества, содержащие примерно по $n/2$ точек каждое. Затем рекурсивно ищутся выпуклые оболочки для двух этих подмножеств. Для построения выпуклой оболочки множества из n точек необходимо осуществить шаг слияния, то есть объединить уже построенные оболочки P_1 и P_2 подмножеств (см. рис. 2.1) за время $O(n)$.

Пользуясь упорядоченностью вершин сливаемых оболочек, такую процедуру можно преобразовать к обходу Грэхема. Опишем шаг построения оболочки объединения выпуклых многоугольников P_1 и P_2 более детально:

1. Найти некоторую внутреннюю точку p многоугольника P_1. В качестве такой вершины можно взять, например, центроид трех любых вершин P_1. Выбранная таким образом точка p будет внутренней точкой $CH(P_1 \cup P_2)$ (см. рис. 2.2).

2. Определить является ли точка p внутренней точкой P_2. Это можно сделать за линейное время от количества вершин в P_2 (точка p – внутренняя точка выпуклого многоугольника P, если при обходе P против часовой стрелки, p все время лежит слева). Если p не является внутренней точкой P_2, перейти к шагу 4.

3. p является внутренней точкой P_2 (см. рисунок 2.2). Вершины P_1 и P_2 оказываются упорядоченными по полярному углу относительно точки p. За время $O(n)$ можно получить упорядоченный список вершин как P_1, так и P_2, путем слияния списков вершин этих многоугольников (аналогично как в сортировке *MergeSort* при слиянии двух отсортированных массивов в один). Перейти к шагу 5.

4. p не является внутренней точкой P_2 (см. рис. 2.3). Так же как и в пошаговом алгоритме представим, что в точке p находится точечный источник света. Тогда граница P_2 распадется на освещенную и затенённую цепи. Очевидно, что вершины освещенной цепи могут быть удалены, так как они будут внутренними точками $CH(P_1 \cup P_2)$. Оставшаяся (затенённая) цепь P_2 (вместе с граничными точками) и граница P_1 представляют два упорядоченных списка, содержащих в сумме не более n вершин. За время $O(n)$ их можно слить в один список вершин $P_1 \cup P_2$, упорядоченный по углу относительно точки p.

5. Теперь к полученному списку можно применить обход Грэхема, требующий лишь линейное время, и получить выпуклую оболочку $CH(P_1 \cup P_2)$.

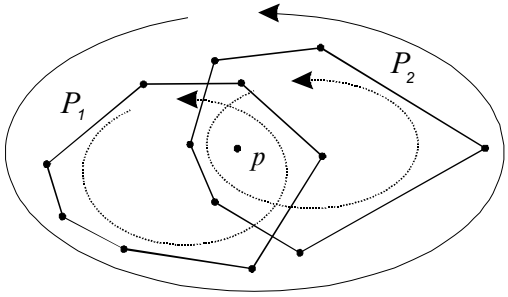


Рис. 2.2. Точка p находится внутри многоугольника P_2. Так как точка p находится одновременно внутри обоих многоугольников P_1 и P_2, то их вершины упорядочены по значению полярного угла относительно точки p. Слияние двух упорядоченных множеств вершин можно выполнить за линейное время.

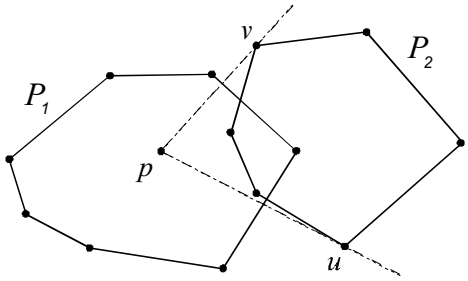


Рис. 2.3. Точка p находится вне многоугольника P_2. Многоугольник P_2 разбивается на освещенную и затененную цепи. Освещенную цепь можно удалить, а слияние вершин второй цепи с упорядоченным множеством вершин многоугольника P_1 можно выполнить за линейное время.

Обозначив время работы этого алгоритма на n точках как $T(n)$, получим:

$$T(n)=\begin{cases} c_1, & n<n_0, \\ c_2 n+2T(n/2), & n\ge n_0. \end{cases}$$

Так как c_1 и c_2 – константы, то $T(n)=O(n\log n)$.

Аналогичное рекуррентное соотношение возникает в анализе алгоритма сортировки слиянием. Заметим, что в случае линейной разделимости (например, по оси абсцисс) объединяемых выпуклых оболочек процедура объединения выпуклых оболочек состояла бы в поиске верхних и нижних «мостиков» (см. рис. 2.4).

Перейдем к следующему алгоритму, который можно рассматривать как обобщение алгоритма быстрой сортировки. Такой алгоритм называется *QuickHull*.

Суть алгоритма состоит в том, что сначала исходное множество из n точек разбивается на два подмножества, выпуклая оболочка каждого из них будет содержать одну из двух ломаных, которые при соединении образуют выпуклую оболочку. Для разбиения исходного множества найдем две различные вершины (крайние точки) выпуклой оболочки A и B, тогда точки, лежащие слева от ориентированного отрезка $[A,B]$, сформируют верхнее подмножество, а справа – нижнее. В качестве точки A можно взять точку с наименьшей ординатой из всех точек, имеющих наименьшую абсциссу, а в качестве точки B – точку с наименьшей ординатой из всех точек, имеющих наибольшую абсциссу.

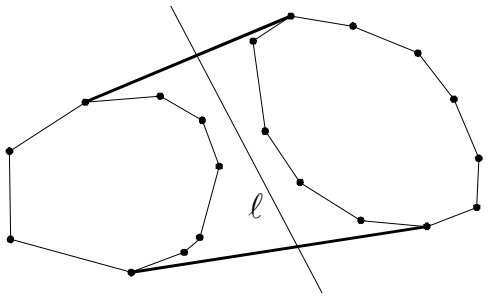


Рис. 2.4. Слияние двух выпуклых оболочек, линейно разделимых прямой l

Для построения выпуклой оболочки, к примеру, верхнего подмножества, найдем вершину C итоговой выпуклой оболочки из данного подмножества, разделяющую выпуклую оболочку на две части. Например, эту вершину можно выбрать так, чтобы площадь $\Delta\, ABC$ была максимальна, а в случае равенства площадей – угол $\angle BAC$ максимален. Точка C определяет разбиение остальных точек подмножества на три множества: точки, лежащие слева от $[A,C]$ (левое множество), справа от $[B,C]$ (правое множество) и внутри или на границе $\Delta\, ABC$ (точки последнего множества можно не рассматривать, так как они заведомо не входят в выпуклую оболочку) (см. рис. 2.5). Далее рекурсивно строятся части выпуклых оболочек левого и правого множеств, затем выпуклые цепи сцепляются в точке C.

В случае, когда разделение на левое и правое множества в среднем сбалансированное, сложность алгоритма, как и в быстрой сортировке $O(n\log n)$. Однако в худшем случае может потребоваться время $O(n^2)$. В отличие от ситуации с быстрой сортировкой невозможно управлять сбалансированным разделением множеств и гарантировать в среднем время работы $O(n\log n)$.

К примеру, худший случай может достигаться на следующем множестве точек (см. рис. 2.6).

Заметим, что, хотя $\Omega(n\log n)$ является нижней оценкой сложности в худшем случае, тем не менее, когда число крайних точек h существенно меньше n, сложность алгоритма может быть уменьшена. В 1986 году

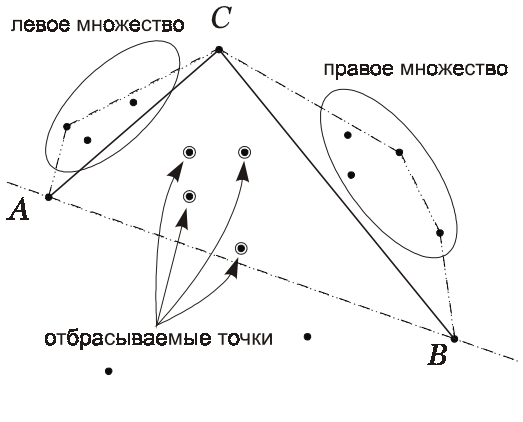
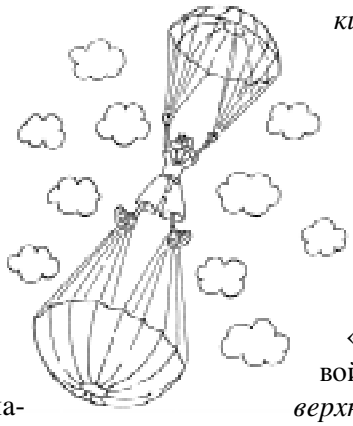


Рис. 2.5. Начало быстрого алгоритма.

это соображение с успехом использовали Киркпатрик и Зайдель, разработавшие асимптотически оптимальный алгоритм со сложностью $O(n \log h)$. Позже, в 1994 году, Тимоти Чен предложил более простой алгоритм также с трудоемкостью $O(n \log h)$.

3. АЛГОРИТМ КИРКПАТРИКА-ЗАЙДЕЛЯ ПОСТРОЕНИЯ ВЫПУКЛОЙ ОБОЛОЧКИ



Как ранее было отмечено, любой алгоритм построения выпуклой оболочки требует в худшем случае времени $\Omega(n \log n)$. Если количество точек в исходном множестве является единственным параметром задачи, то алгоритм со временем работы $O(n \log n)$ является также и оптимальным (например, алгоритм Грэхема). Однако при рассмотрении дополнительного параметра – размера выпуклой оболочки h, возникает вопрос: существует ли алгоритм, асимптотически более эффективный, чем и алгоритм Грэхема со временем работы $O(n \log n)$, и алгоритм Джарвиса со временем работы $O(nh)$ при всех возможных значениях h? Значение h (размер результата задачи) является априори не известным и может варьироваться от константы до n. Ниже будет описан алгоритм Киркпатрика-Зайделя [4] со временем работы $O(n \log h)$ (заметим, что при такой оценке нельзя даже отсортировать исходные точки).

Алгоритм Киркпатрика-Зайделя использует метод *отсечения и поиска*.

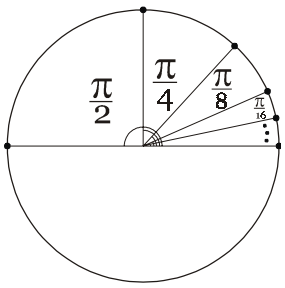


Рис. 2.6. Входные данные, требующие квадратичного времени работы алгоритма *QuickHull*.

Рассмотрим точки с минимальной и максимальной абсциссами из исходного множества Q (а в случае равенства абсцисс – с максимальной ординатой), обозначенные как $p_{\min}$ и $p_{\max}$. Выпуклая оболочка множества Q может быть представлена как пара выпуклых цепочек – *верхней оболочки* и *нижней оболочки* множества Q (см. рис. 3.1 *а*). Опишем алгоритм построения верхней оболочки (процедура построения нижней оболочки аналогична). Этот алгоритм (листинг 1) строит рис. 3.1.

Описанный алгоритм принадлежит к классу алгоритмов «разделяй и властвуй». Ключевой момент состоит в нахождении *верхнего моста* на шаге 6 (см. рис. 3.1 *б*). Верхний мост – это отрезок прямой, соединяющий точку из L и точку из R, такие что все точки из L и R лежат ниже этой прямой или на этом отрезке. Далее будет отдельно показано, как этот шаг может быть выполнен за $O(n)$ операций с использованием метода отсечения и поиска. Известно [2], что шаг 4 (нахождения медианы) также может быть выполнен за $O(n)$. Следовательно, шаги 3–8 могут быть суммарно выполнены за $O(n)$. Для анализа времени выпол-

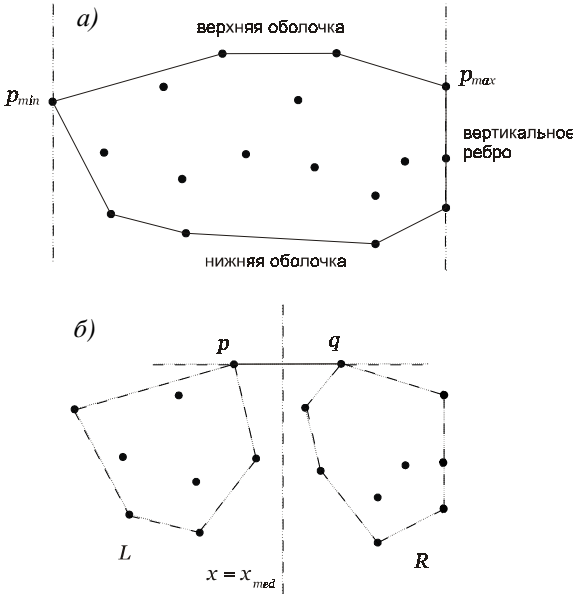


Рис. 3.1. *а)* Верхняя и нижняя оболочки, *б)* верхний мост $\overline{pq}$

Листинг 1. algorithm UPPERHULL (Q) → верхняя оболочка

Вход. Исходное множество точек $Q = \{p_i \mid p_i = (x_i, y_i), i \in [1..n]\}$, целочисленный параметр m $(1 \le m \le n)$.

Выход. Верхняя оболочка множества точек Q, заданная последовательностью вершин в порядке возрастания абсциссы

1 **if** $|Q| \le 2$ **then**
2 Вернуть упорядоченную по абсциссе последовательность вершин из Q
3 Найти точки $p_{\min}$ и $p_{\max}$ из Q с минимальной и максимальной абсциссами
4 Найти медиану x_{med} абсцисс точек из Q
5 Линейно разделить Q на множества L и R по вертикальной прямой $x = x_{med}$ (точки, принадлежащие прямой, разделить между L и R).
6 Найти *верхний мост* $\overline{pq}$ множеств L и R, где $p \in L$ и $q \in R$
7 $L' \leftarrow \{r \in L \mid (p_{\min}, r, p)$ образуют правый поворот$\}$
8 $R' \leftarrow \{r \in R \mid (p_{\max}, r, q)$ образуют левый поворот$\}$
9 $LUH \leftarrow$ **UPPERHULL** (L')
10 $RUH \leftarrow$ **UPPERHULL** (R')
11 Вернуть сцепление LUH, $\overline{pq}$ и RUH как верхнюю оболочку множества Q

нения UPPERHULL(Q) обозначим через h общее количество вершин в верхней оболочке, а через $T(n,h)$ – асимптотическую оценку алгоритма в худшем случае. Предположим, что LUH и RUH содержат по h_1 и h_2 вершин соответственно (заметим, что $h = h_1 + h_2$). Поскольку $|L'| \le |L|$ и $|R'| \le |R|$, то два рекурсивных вызова на шагах 9 и 10 выполнятся за время $T(n/2, h_1)$ и $T(n/2, h_2)$, соответственно. Рекуррентное соотношение, описывающее $T(n,h)$, имеет следующий вид:

$$T(n,h) = \begin{cases} O(n) + \max\limits_{h_1,h_2} \{T(n/2,h_1) + T(n/2,h_2)\}, & h > 2 \\ O(n), & h \le 2 \end{cases}$$

Если $h = 2$, то верхняя оболочка состоит из одного верхнего моста, соединяющего самую левую и самую правую точки из Q (найти такой верхний мост можно за время $O(n)$). Докажем по индукции, что $T(n,h) \le cn \log h$. Действительно, для любых h_1 и h_2, в том числе и для тех, что соответствуют худшему случаю, имеем

$$T(n,h) = T\left(\frac{n}{2}, h_1\right) + T\left(\frac{n}{2}, h_2\right) + O(n) \le$$

$$\le \frac{cn}{2} \log h_1 + \frac{cn}{2} \log h_2 + an \le$$

$$\le cn \log \left(\frac{h_1 + h_2}{2}\right) + an = cn \log h - cn + an =$$

$$= cn \log h - (c - a)n \le cn \log h$$

Здесь использовано неравенство $\sqrt{h_1 h_2} \le \frac{h_1 + h_2}{2}$ и тот факт, что всегда может быть сделано $(c - a) \ge 0$. Таким образом, $T(n,h) = O(n \log h)$.

НАХОЖДЕНИЕ ВЕРХНЕГО МОСТА ЗА ЛИНЕЙНОЕ ВРЕМЯ

Задача формулируется следующим образом: дано множество Q из n точек на плоскости, разделенное на два непустых подмножества L и R вертикальной прямой $x = a$ ($L = \{p \in Q \mid p_x \le a\}$ и $R = Q \setminus L$). Необходимо найти прямую t, проходящую через точку из L и точку из R, так чтобы точки из Q не лежали выше t. Если выпуклые оболочки L и R были бы известны, то общая опорная прямая могла быть найдена за линейное время, как это делается в алгоритме построения выпуклой оболочки методом «разделяй и властвуй» с линейным разделением. Но нахождение выпуклой оболочки n точек требует $\Omega(n \log n)$ времени в худшем случае.

Алгоритм Киркпатрика-Зайделя решения этой задачи основан на методе отсечения и поиска и позволяет за линейное время найти верхний мост, удаляя на каждом шаге часть заведомо ненужных точек.

Рассмотрим все прямые с фиксированным наклоном α, проходящие через точки из Q (наклон прямой определяется как тангенс угла между прямой и осью абсцисс). Определим опорную прямую к множеству Q с наклоном α как прямую с наклоном α, проходящую через точку из Q и имеющую максимальную ординату точки пересечения с осью OY. Очевидно, что такая опорная прямая к множеству Q может быть построена за $O(n)$. Можно сделать следующий вывод (см. рис. 3.2):

1. Если опорная прямая проходит через точки $p \in L$ и $q \in R$, то это мост.

2. Если опорная прямая проходит только через точки множества L, то это означает, что наклон α больше наклона верхнего моста.

3. Если опорная прямая проходит только через точки из R, то наклон α меньше наклона верхнего моста.

Рассмотрим первый случай, когда наклон моста меньше α. Обозначим за r и за s такие две точки из множества Q, что наклон rs больше α и $x_r \le x_s$.

Можно сделать вывод, что верхний мост не может проходить через r, потому что прямая, проходящая через r, с наклоном меньшим α будет проходить под точкой s. Следовательно, точка r может быть удалена из рассмотрения (см. рис. 3.2). Во втором случае, когда наклон моста больше α, ситуация симметрична, и точка s может быть удалена из рассмотрения, поскольку прямая с наклоном, большим α, будет проходить под точкой r.

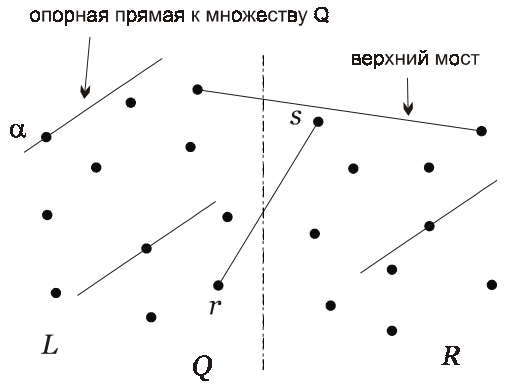


Рис. 3.2. Наклон моста меньше $\alpha \Rightarrow$ точка r может быть удалена из рассмотрения

Эти соображения позволяют сформулировать следующий метод «отсечения и поиска» для нахождения верхнего моста. Разобьем точки из множества Q в произвольном порядке на пары и найдем медиану α среди наклонов $\lfloor n/2 \rfloor$ отрезков, образованных парами точек. Затем построим опорную прямую l с наклоном α к множеству Q. Если l проходит только через точки множества L, то из каждой пары, образующей отрезок с наклоном, большим или равным α, одна из точек (левая) может быть удалена из рассмотрения. Аналогично, в случае, если l проходит только через точки множества R, то из каждой пары, образующей отрезок с наклоном, меньшим или равным α, одна из точек (правая) может быть удалена из рассмотрения. В обоих случаях из множества Q удаляется не менее $n/4$ точек. Если l проходит через точки $p \in L$ и $q \in R$, то мост найден (вершины, определяющие мост, могут быть найдены за $O(n)$ операций).

Найти медиану наклонов можно за линейное время [2]. Остальные операции требуют также линейного времени. Таким образом, за $O(n)$ операций происходит либо удаление $n/4$ точек, либо нахождение моста. Если последовательно применять такое удаление (или отсечение) на оставшихся точках, то можно гарантировать нахождение моста за $O(n + (3/4)n + (3/4)^2 n + \ldots) = O(n)$. Заметим, что удаление различного числа точек из L и R не влияет на анализ. Таким образом, приходим к выводу, что верхний мост может быть найден за линейное время, что завершает анализ алгоритма Киркпатрика-Зайделя.

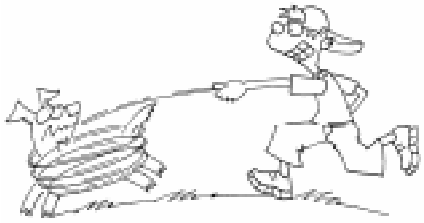
В [4] также показано, что оценка $\Omega(n \log h)$ является нижней границей задачи построения выпуклой оболочки в худшем случае (используя модель алгебраических деревьев решения). Таким образом, алгоритм Киркпатрика-Зайделя является оптимальным в худшем случае.

4. АЛГОРИТМ ЧЕНА

Идея алгоритма Киркпатрика-Зайделя достаточно сложна и опирается на возмож-

ность нахождения медианы последовательности чисел за линейное в худшем случае время.

В 1995 году Тимоти Ченом был предложен алгоритм, имеющий ту же сложность $O(n \log h)$, что и алгоритм Киркпатрика-Зайделя, но использующий более простую идею. Этот алгоритм принято называть по имени автора *алгоритмом Чена* [5]. Для достижения указанной асим-птотики он использует ту же идею *заворачивания*, что и алгоритм Джарвиса, с той лишь разницей, что нахождение очередной точки выпуклой оболочки осуществляется быстрее за счет использования простой предобработки, основанной на идее *группировки*.



Предлагается разбить исходное множество точек Q на несколько непересекающихся подмножеств Q_i. Затем следует найти их выпуклые оболочки $CH(Q_i)$ и построить результирующую выпуклую оболочку $CH(Q)$, используя метод заворачивания.

В процессе заворачивания для нахождения следующей вершины выпуклой оболочки будет использоваться информация об уже построенных выпуклых оболочках $CH(Q_i)$. Базовой операцией будем считать нахождение *правой опорной точки* выпуклого многоугольника S относительно текущей точки p. Правая опорная точка выпуклого многоугольника S относительно точки p – это такая точка $q \in S$, что любая точка многоугольника S лежит по левую сторону от ориентированного отрезка $[p,q]$ или на нем.

На каждом шаге заворачивания необходимо найти все правые опорные точки q_i выпуклых

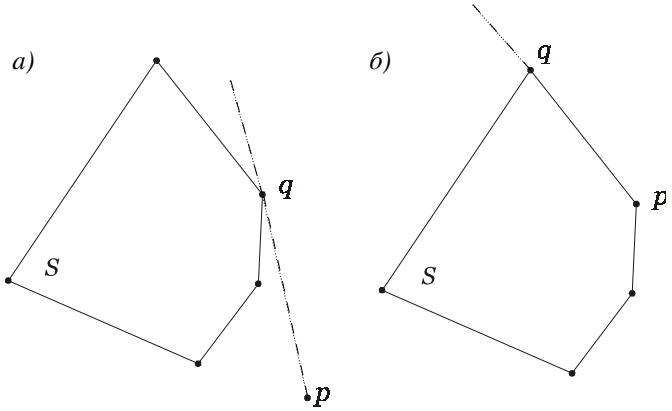


Рис. 4.1. Точка q – правая опорная точка многоугольника S относительно точки p. Пример (*а*) и вырожденный случай, когда точка p принадлежит границе многоугольника (*б*)

многоугольников $CH(Q_i)$ относительно точки $p_{current}$ и линейным поиском выбрать среди них такую точку p_{next}, чтобы любая точка q_i лежала по левую сторону от ориентированного отрезка $[p_{current}, p_{next}]$ или на нем. В таком случае точка p_{next} будет следующей вершиной выпуклой оболочки в порядке обхода против часовой стрелки (см. рис. 4.2).

Очевидно, что от количества групп, на которые мы разбиваем исходное множество, и от количества точек в каждой группе зависит сложность алгоритма.

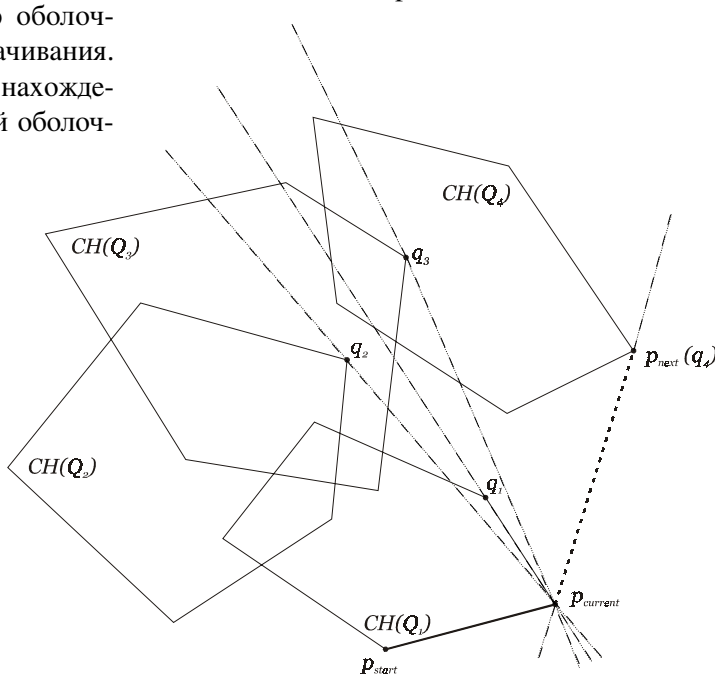


Рис. 4.2. Один из шагов алгоритма Чена

Пусть задан целочисленный параметр $1 \le m \le n$ (n – число точек в исходном множестве), такой что количество групп равно $\lceil n/m \rceil$ и в каждой группе содержится не более чем m точек. Заметим, что такое разбиение на группы всегда осуществимо.

Алгоритм в зависимости от параметра m будет выглядеть следующим образом (см. листинг 2).

Правую опорную точку выпуклого многоугольника S относительно точки p можно найти двоичным поиском за время $O(\log m)$, где m – количество вершин в многоугольнике. Тогда суммарная сложность заворачивания будет равна

$$O(\lceil n/m \rceil \cdot h \cdot \log m),$$

так как будет сделано всего h шагов заворачивания и на каждом шаге в каждой из $\lceil n/m \rceil$ групп за $O(\log m)$ будет найдена правая опорная точка из текущей точки к соответствующей группе.

Можно улучшить эту асимптотику до $O(n + \lceil n/m \rceil \cdot h)$, если заметить, что на каждом следующем шаге правая опорная точка q_i либо останется такой же, либо примет значение одной из следующих вершин $CH(Q_i)$ в сторону обхода против часовой стрелки, причем никакая вершина не может быть пройдена больше чем два раза.

Таким образом, каждый раз, когда нужно найти правую опорную точку q_i относительно новой точки $p_{current}$, берется старое

Листинг 2. algorithm CHAN-MARCH-WITH-PARAM $(Q, m) \to CH(Q)$

Вход. Исходное множество точек $Q = \{p_i \mid p_i = (x_i, y_i), i \in [1..n]\}$, целочисленный параметр m $(1 \le m \le n)$

Выход. Выпуклая оболочка множества точек $CH(Q)$, заданная последовательностью вершин (перечисленных в порядке обхода против часовой стрелки).

```
1   Создать пустой стек S
2   Разделить множество Q на ⌈n/m⌉ непересекающихся подмножеств Q_i так, чтобы
    в каждом подмножестве было не более чем m точек
3   Построить выпуклые оболочки CH(Q_i) для множеств Q_i любым из оптимальных
    в худшем случае алгоритмов (например, алгоритмом Грэхема)
4   Выбрать точку p_start, которая будет являться крайней точкой выпуклой оболочки
5   p_current ← p_start
6   Do
7       PUSH(S, p_current)
8       for ∀i ∈ [1..⌈n/m⌉] do
9           Найти правую опорную точку q_i выпуклого многоугольника CH(Q_i)
            относительно точки p_current
10      end-do
11      p_next ← q_1
12      for ∀i ∈ [2..⌈n/m⌉] do
13          if три точки (p_current, q_i, p_next) образуют левый поворот or
               p_next лежит на отрезке [p_current, q_i] then
14          p_next ← q_i
15          end-if
16      end-do
17      p_current ← p_next
18  while p_current ≠ p_start
19  Вернуть в качестве результата последовательность точек S, содержащуюся в стеке
    (дно стека будет первым элементом результирующей последовательности, а TOP(S) –
    последним)
```

значение q_i и проверяется следующая за q_i точка NEXT(q_i): если оказывается, что точка q_i лежит левее ориентированного отрезка [$p_{current}$,NEXT(q_i)] или строго на нем, то в качестве q_i берется NEXT(q_i) и операция повторяется, в противном случае опорной точкой считается точка q_i (см. рис. 4.3). На первом шаге для нахождения правых опорных точек q_i используется наивный алгоритм (или двоичный поиск).

Оценим суммарную сложность работы алгоритма:

1. Разбиение множества Q на подмножества Q_i выполняется за время $O(n)$.

2. Так как в каждом подмножестве Q_i не более чем m точек, построение выпуклых оболочек будет иметь суммарную сложность $O(\lceil n/m \rceil \cdot m \log m)$, то есть $O(n \log m)$.

3. Выбор начальной вершины выпуклой оболочки осуществляется за время $O(n)$.

4. Алгоритм сделает h шагов, при этом на каждом шаге он затратит время $O(\lceil n/m \rceil)$ на линейный поиск следующей точки выпуклой оболочки среди правых опорных точек (без учета времени, потраченного на их поиск).

5. Так как каждая точка исходного множества Q либо является вершиной в точности одного выпуклого многоугольника $CH(Q_i)$, либо не является вершиной ни одного из них и при этом каждая вершина при поиске опорных точек может быть пройдена не более чем два раза, то сложность подсчета всех опорных точек на всех шагах алгоритма составит $O(n)$.

Таким образом, получается, что суммарная сложность алгоритма составляет $O(n \log m + h \cdot \lceil n/m \rceil + n)$. Если в качестве параметра m мы бы выбрали h, то сложность алгоритма свелась бы к $O(n \log h)$. Проблема заключается в том, что мы не всегда можем точно знать h в момент начала выполнения алгоритма. Поэтому предлагается подход, позволяющий *подобрать* значение h без увеличения асимптотической сложности алгоритма.

Модифицируем алгоритм CHAN-MARCH-WITH-PARAM(Q, m) так, чтобы он делал не более чем m шагов и сигнализировал бы о том, что после последнего сделанного шага выпуклая оболочка $CH(Q)$ еще не построена. Тогда алгоритм построения выпуклой оболочки будет выглядеть следующим образом (см. листинг 3).

Очевидно, что алгоритм всегда завершится, поскольку для случая $m = n$, он представляет собой не что иное, как алгоритм Грэхема.

Также очевидно, что алгоритм завершится, как только выполнится условие $m \geq h$, то есть $t \geq \lceil \log \log h \rceil$. При фиксированном t время работы алгоритма составит $O(n \cdot 2^t)$. Таким образом, к тому моменту, как алгоритм завершится, он проработает время:

$$O\left(\sum_{i=1}^{\lceil \log \log h \rceil} n \cdot 2^i\right) = O(n \cdot 2^{\lceil \log \log h \rceil + 1}) = O(n \log h).$$

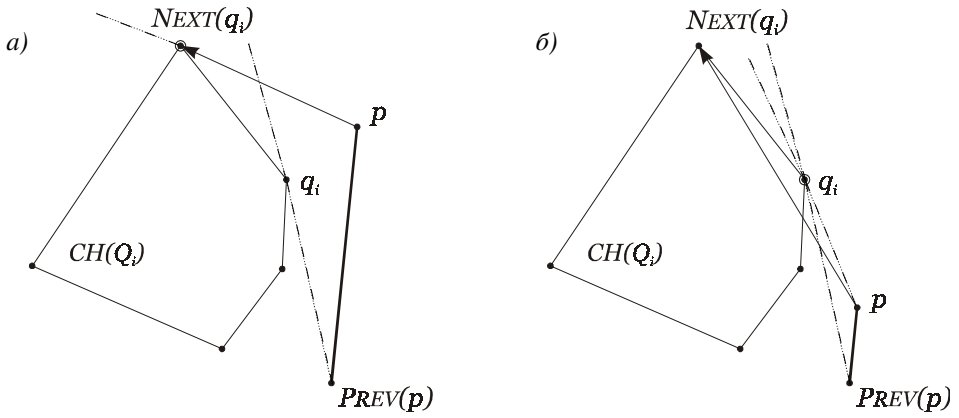


Рис. 4.3. Процесс получения из предыдущей опорной точки q_i следующей. Новой опорной точкой становится NEXT(q_i) (*а*), опорная точка не меняется (*б*).

Листинг 3. algorithm CHAN-MARCH $(Q) \to CH(Q)$

Вход. Исходное множество точек $Q = \{p_i \mid p_i = (x_i, y_i), i \in [1..n]\}$.

Выход. Выпуклая оболочка множества точек $CH(Q)$, заданная последовательностью вершин (перечисленных в порядке обхода против часовой стрелки).

```
1 | for t ← 1,2,3,… do
2 |    m ← min(n, 2^(2^t))
3 |    Вызвать модификацию CHAN-MARCH-WITH-PARAM(Q, m)
4 |    if Алгоритм построил выпуклую оболочку CH(Q) then
5 |       Вернуть в качестве результата CH(Q)
6 |    end-if
7 | end-do
```

ЗАКЛЮЧЕНИЕ

Рассмотренная связь задачи построения выпуклой оболочки на плоскости с задачей сортировки позволяет, во-первых, найти нижнюю оценку сложности задачи построения выпуклой оболочки, а во-вторых, открывает возможность конструировать новые алгоритмы построения выпуклой оболочки по аналогии с известными эффективными алгоритмами сортировки. За рамками этой аналогии остаются рассмотренные алгоритмы Киркпатрика-Зайделя и Чена, сложность которых зависит от размера построенной выпуклой оболочки. Однако и эти алгоритмы косвенно связаны с задачей частичной упорядоченности (например, с методом отсечения и поиска, впервые примененном в алгоритме нахождения медианы массива чисел).

Литература

1. *Ивановский С.А., Преображенский А.С., Симончик С.К. Алгоритмы вычислительной геометрии. Выпуклые оболочки: простые алгоритмы* // Компьютерные инструменты в образовании, 2007, № 1. С. 4–19.

2. *Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ*. М.: МЦНМО, 2000. 960 с.

3. *Препарата Ф., Шеймос М. Вычислительная геометрия: Введение*. М.: Мир, 480 с., 1989.

4. *D. G. Kirkpatrick and R. Seidel*. The Ultimate Planar Convex Hull Algorithm, SIAM Journal on Computing, 15, 1986. P. 286–294/

5. *Chan T.M.* Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions, Discrete and Computational Geometry, No. 16, pp. 361-368, Springer-Verlag New-York Inc., 1996.

Ивановский Сергей Алексеевич,
кандидат технических наук,
доцент кафедры Математического обеспечения и применения ЭВМ СПбГЭТУ «ЛЭТИ»,

Преображенский Алексей Семенович,
аспирант СПбГЭТУ “ЛЭТИ”,
магистр прикладной математики и информатики,

Симончик Сергей Константинович,
аспирант СПбГЭТУ «ЛЭТИ»
магистр прикладной математики и информатики.