Статические выпуклые оболочки: Джарвис, Грэхем, Эндрю, Чен, QuickHull

Конспект готов к прочтению.

Определение:

Выпуклой оболочкой множества точек называется пересечение всех выпуклых множеств, содержащих все заданные точки.

Ниже приводятся основные алгоритмы построения выпуклых оболочек статического множества.
Используются обозначения: n - размер входных данных, k - размер оболочки.

Содержание

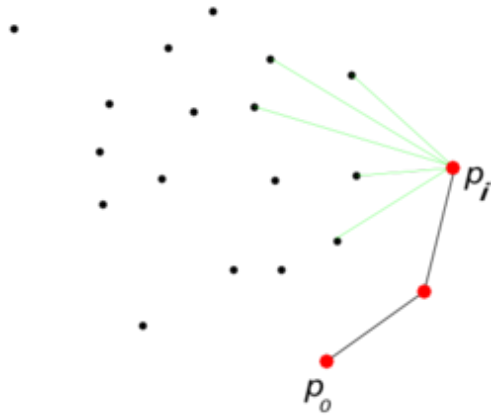
- 1 Алгоритм Джарвиса
  - 1.1 Описание Алгоритма
  - 1.2 Корректность
  - 1.3 Псевдокод
  - 1.4 Сложность
  - 1.5 Ссылки
- 2 Алгоритм Грэхема
  - 2.1 Описание Алгоритма
  - 2.2 Корректность
  - 2.3 Псевдокод
  - 2.4 Сложность
  - 2.5 Ссылки
- 3 Алгоритм Эндрю
  - 3.1 Описание Алгоритма
  - 3.2 Корректность
  - 3.3 Псевдокод
  - 3.4 Сложность
  - 3.5 Ссылки
- 4 Алгоритм Чена
  - 4.1 Описание Алгоритма
  - 4.2 Сложность
  - 4.3 Поиск m
  - 4.4 Ссылки
- 5 Алгоритм QuickHull
  - 5.1 Описание Алгоритма
  - 5.2 Корректность
  - 5.3 Реализация
  - 5.4 Псевдокод
  - 5.5 Сложность
  - 5.6 Ссылки

Алгоритм Джарвиса

По-другому "Gift wrapping algorithm" (Заворачивание подарка). Он заключается в том, что мы ищем выпуклую оболочку последовательно, против часовой стрелки, начиная с определенной точки.

Описание Алгоритма

1. Возьмем точку p_0 нашего множества с самой маленькой y-координатой (если таких несколько, берем самую правую из них). Добавляем ее в ответ.
2. На каждом следующем шаге для последнего добавленного p_i ищем p_{i+1} среди всех недобавленных точек и p_0 с максимальным полярным углом относительно p_i (Если углы равны, надо сравнивать по расстоянию). Добавляем p_{i+1} в ответ. Если $p_{i+1} == p_0$, заканчиваем алгоритм.



Промежуточный шаг алгоритма. Для точки p_i ищем следующую перебором.

Корректность

Точка p_0, очевидно, принадлежит оболочке. На каждом последующем шаге алгоритма мы получаем прямую $p_{i-1}p_i$, по построению которой все точки множества лежат слева от нее. Значит, выпуклая оболочка состоит из p_i-ых и только из них.

Псевдокод

Inplace-реализация алгоритма. $S[1..n]$ - исходное множество, $n > 2$

```
Jarvis(S)
  find i such that S[i] has the lowest y-coordinate and highest x-coordinate
  p0 = S[i]
  pi = p0
  k = 0
  do
    k++
    for i = k..n
      if S[i] has lower angle and higher distance than S[k] in relation to pi
        swap(S[i], S[k])
    pi = S[k]
  while pi != p0
  return k
```

Сложность

Добавление каждой точки в ответ занимает $O(n)$ времени, всего точек будет k, поэтому итоговая сложность $O(nk)$. В худшем случае, когда оболочка состоит из всех точек сложность $O(n^2)$.

Ссылки

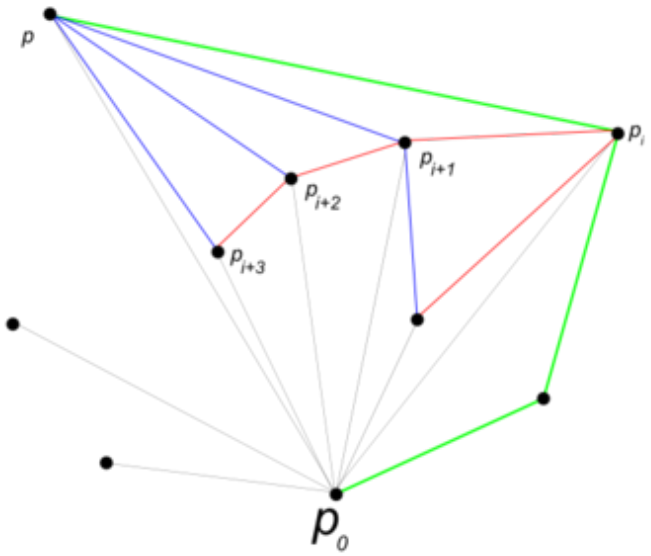
- Английская статья — Wikipedia (http://en.wikipedia.org/wiki/Gift_wrapping_algorithm)
- Русская статья — Wikipedia (http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B6%D0%B0%D1%80%D0%B2%D0%B8%D1%81%D0%B0)

Алгоритм Грэхема

Алгоритм заключается в том, что мы ищем точки оболочки последовательно, используя стек.

Описание Алгоритма

1. Находим точку p_0 нашего множества с самой маленькой y-координатой (если таких несколько, берем самую правую из них), добавляем в ответ.
2. Сортируем все остальные точки по полярному углу относительно p_0.
3. Добавляем в ответ p_1 - самую первую из отсортированных точек.
4. Берем следующую по счету точку t. Пока t и две последних точки в текущей оболочке p_i и p_{i-1} образуют неправый поворот (вектора $p_i t$ и $p_{i-1} p_i$), удаляем из оболочки p_i.
5. Добавляем в оболочку t.
6. Делаем п.5, пока не закончатся точки.



Промежуточный шаг алгоритма. Зелеными линиями обозначена текущая выпуклая оболочка, синими - промежуточные соединения точек, красными - те отрезки, которые раньше входили в оболочку, а сейчас нет. На текущем шаге при добавлении точки p последовательно убираем из оболочки точки с $i+3$-ей до $i+1$-ой

Корректность

Докажем, что на каждом шаге множество p_i-тых является выпуклой оболочкой всех уже рассмотренных точек. Доказательство проведем по индукции.

- База. Для трех первых точек утверждение, очевидно, выполняется.
- Переход. Пусть для $i-1$ точек оболочки совпадают. Докажем, что и для i точек они совпадут.

Рассмотрим истинную оболочку $ch(S \cup i) = ch(S) \cup i \setminus P$, где P - множество всех точек из $ch(S)$, видимых из i. Так как мы добавляли точки в нашу оболочку против часовой стрелки и так как i-тая точка лежит в $ch(S \cup i)$, то P состоит из нескольких подряд идущих последних добавленных в оболочку точек, и именно их мы удаляем на текущем шаге. Поэтому наша оболочка и истинная для i точек совпадают.

Тогда по индукции оболочки совпадают и для $i = n$.

Псевдокод

Inplace-реализация алгоритма. $S[1..n]$ - исходное множество, $n > 2$

```
Graham(S)
  find i such that S[i] has the lowest y-coordinate and highest x-coordinate
  swap(S[i], S[1])
  sort S[2..n] by angle in relation to S[1]
  k = 2
  for p = 3..n
    while S[k - 1], S[k], S[p] has non-right orientation and k > 1
      k--
    swap(S[p], S[k + 1])
    k++
  return k + 1
```

Сложность

Сортировка точек занимает $O(n \log n)$ времени. При обходе каждая точка добавляется в ответ не более одного раза, поэтому сложность этой части - $O(n)$. Суммарное время — $O(n \log n)$.

Ссылки

- Английская статья — Wikipedia (http://en.wikipedia.org/wiki/Graham_scan)
- Русская статья — Wikipedia (http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%93%D1%80%D1%8D%D1%85%D0%B5%D0%BC%D0%B0)

Алгоритм Эндрю

Алгоритм, очень похожий на алгоритм Грехема. Он заключается в том, что мы находим самую левую и самую правую точки, ищем для точек над и под этой прямой выпуклую оболочку Грехемом - для них начальные точки будут лежать на $\pm inf$, а сортировка по углу относительно далекой точки аналогична сортировке по координате; после этого объединяем две оболочки в одну.

Описание Алгоритма

1. Находим самую левую и самую правую точки множества - p_0 и p_1.
2. Делим множество на две части: точки над и под прямой.
3. Для каждой половины ищем выпуклую оболочку Грехемом с условием, что сортируем не по полярному углу, а по координате.
4. Сливаем получившиеся оболочки.

Корректность

См. доказательство алгоритма Грехема.

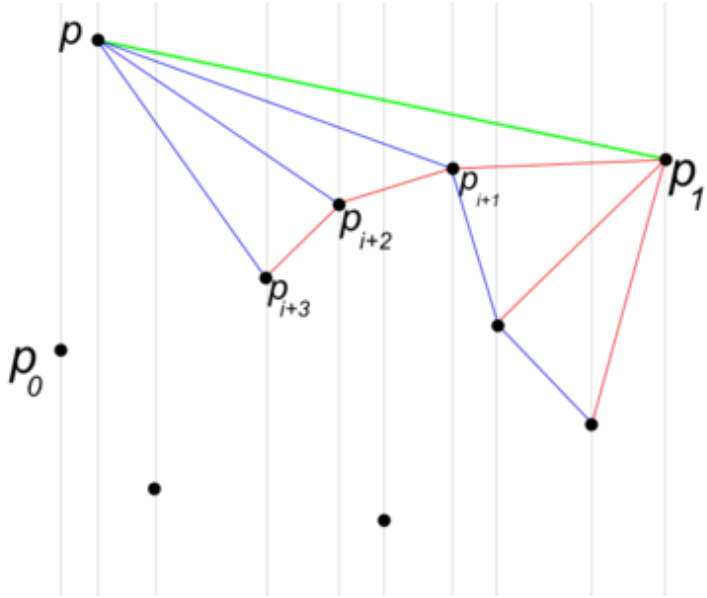
Псевдокод

Inplace-реализация алгоритма. $S[1..n]$ - исходное множество, $n > 2$

```
Andrew(S)
  sort S[1..n] by x-coordinate backward(than by y backward)
  k = 2
  for p = 3..n
   while S[k - 1], S[k], S[p] has non-right orientation
     k--
   swap(S[p], S[k + 1])
  k++
  sort S[k + 1..n] by x-coordinate (than by y)
  for p = k + 1..n
    while S[k - 1], S[k], S[p] has non-right orientation
      k--
    swap(S[p], S[k + 1])
  return k + 1
```

Сложность

Сортировка точек занимает $O(n \log n)$ времени. При обходе каждая точка добавляется в ответ не более одного раза, поэтому сложность двух обходов - $2 \cdot O(n)$. Суммарное время - $O(n \log n)$. Также можно отметить тот факт, что Эндрю в целом работает быстрее чем Грэхем, так как использует всего $O(n)$ поворотов, в то время как Грэхем использует $O(n \log n)$ поворотов.



Промежуточный шаг алгоритма. Зелеными линиями обозначена текущая выпуклая оболочка, синими - промежуточные соединения точек, красными - те отрезки, которые раньше входили в оболочку, а сейчас нет. На текущем шаге при добавлении точки p последовательно убираем из оболочки точки с $i + 3$-ей до $i + 1$-ой

Ссылки

- Одно предложение о нем (http://en.wikipedia.org/wiki/Graham_scan#Notes)
- Имплементация на 13 языках (https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain)

Алгоритм Чена

Является комбинацией двух алгоритмов - Джарвиса и Грехема. Недостатком Грэхема является необходимость сортировки всех точек по полярному углу, что занимает достаточно много времени $O(n \log n)$. Джарвис требует перебора всех точек для каждой из k точек оболочки, что в худшем случае занимает $O(n^2)$.

Описание Алгоритма

1. Разобьем все множество на произвольные группы по m штук в каждой. Будем считать, что m нам известно. Тогда всего групп окажется $r = n/m$.

2. Для каждой группы запускаем Грехема.
3. Начиная с самой нижней точки ищем саму выпуклую оболочку Джарвисом, но перебираем не все точки, а по одной из каждой группы.

Сложность

На втором шаге алгоритма в каждой группе оболочка ищется за $O(m \log m)$, общее время - $O(rm \log m) = O(n \log m)$. На третьем шаге поиск каждой следующей точки в каждой группе занимает $O(\log m)$, так как точки уже отсортированы, и мы можем найти нужную бинпоиском. Тогда поиск по всем группам займет $O(r \log m) = O(\frac{n}{m} \log m)$. Всего таких шагов будет k, значит общее время - $O(\frac{kn}{m} \log m)$. Итоговое время - $O(n(1 + \frac{k}{m}) \log m)$. Несложно видеть, что минимум достигается при $m = k$. В таком случае сложность равна $O(n \log k)$.

Поиск m

Как заранее узнать k? Воспользуемся следующим методом. Положим $m = 2^{2^t}$. Начиная с маленьких m будем запускать наш алгоритм, причем если на третьем шаге Джарвис уже сделал m шагов, то мы выбрали наше m слишком маленьким, будем увеличивать, пока не станет $m \geq k$. Тогда общее время алгоритма - $\sum_{t=0}^{O(\log \log k)} O\left(n \log(2^{2^t})\right) = O(n) \sum_{t=0}^{O(\log \log k)} O(2^t) = O\left(n \cdot 2^{1+O(\log \log k)}\right) = O(n \log k).$

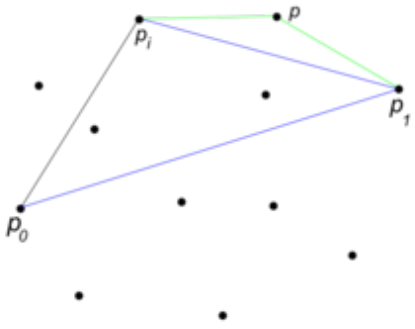
Ссылки

- Английская статья — Wikipedia (http://en.wikipedia.org/wiki/Chan%27s_algorithm)
- Русская статья — Wikipedia (http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A7%D0%B0%D0%BD%D0%B0)

Алгоритм QuickHull

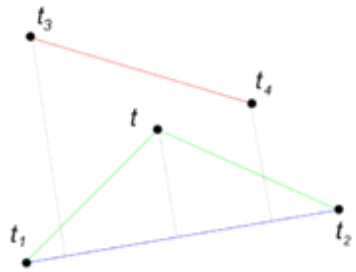
Описание Алгоритма

1. Найдем самую левую точку p_0 и самую правую точку p_1 (Если таких несколько, выберем среди таких нижнюю и верхнюю соответственно).
2. Возьмем все точки выше прямой p_0p_1 .
3. Найдем среди этого множества точку p_i , наиболее отдаленную от прямой (если таких несколько, взять самую правую).
4. Рекурсивно повторить шаги 2-3 для прямых p_0p_i и p_ip_1 , пока есть точки.
5. Добавить в ответ точки $p_0 . . p_i . . p_1$, полученные в п. 3.
6. Повторить пункты 2-5 для p_1p_0 (то есть для "нижней" половины).
7. Ответ - объединение списков из п. 5 для верхней и нижней половины.



Промежуточный шаг алгоритма. Для прямой p_ip_1 нашли точку p. Над прямыми p_ip и pp_1 точек нет, поэтому переходим к следующей прямой p_0p_i .

Корректность



Очевидно, что выпуклая оболочка всего множества является объединением выпуклых оболочек для верхнего и нижнего множества. Докажем, что алгоритм верно строит оболочку для верхнего множества, для нижнего рассуждения аналогичны. Точки p_0 и p_1 принадлежат оболочке.

- Пусть какая-то точка входит в нашу оболочку, но не должна.

Назовем эту точку t. По алгоритму эта точка появилась как самая удаленная от некой прямой $t_1 t_2$. Так как t не входит в оболочку, то существует прямая $t_3 t_4$ из настоящей выпуклой оболочки, что t лежит снизу от прямой. Тогда какая-то из t_3 и t_4 удалена от прямой дальше t, что противоречит алгоритму.

- Наоборот, пусть какой-то точки t в нашей оболочке нет, а должна быть.

Пойдем вниз рекурсии в те ветки, где есть t. В какой-то момент t окажется внутри некоторого треугольника. Но тогда возникает противоречие с тем, что t принадлежит выпуклой оболочке.

Таким образом, наша оболочка совпадает с истинной, а значит алгоритм корректен.

Реализация

Заметим, что длина высоты, опущенная из точки t на отрезок ab, пропорциональна векторному произведению $[bt \cdot ba]$, поэтому для сравнения можно использовать именно это. Векторное произведение эквивалентно предикату поворота, поэтому можно так же использовать и его.

Псевдокод

Inplace-реализация алгоритма. $S[1..n]$ - исходное множество. $quick\_hull()$ - рекурсивная функция, находящая оболочку подмножества S. В реализации в конце каждого подмножества находятся эл-ты, точно не принадлежащие оболочке.

```
QuickHull(S)
  find i such that S[i] has the highest x-coordinate and lowest y-coordinate
  swap(S[1], S[i])
  find i such that S[i] has the lowest x-coordinate and lowest y-coordinate
  swap(S[n], S[i])
  k = partition1(S) // разбиваем на те эл-ты, которые лежат над прямой и на остальные
  a = quick_hull(S, 1, k)
  b = quick_hull(S, k + 1, n);
  swap(S[a..k], S[k + 1, b])
  return start + (a - 1) + (b - k - 1)

quick_hull(S, start, end)
  find i such that S[i], S[start], S[end] has maximum value
  (a, b) = partition2(S, start, end, S[i]) //свапаем эл-ты S так, чтобы сначала были все эл-ты над прямой S[start]
S[i], потом S[i]S[end], потом все остальное
  c = quick_hull(S, start, a)
  d = quick_hull(S, a + 1, b)
  swap(S[c..a], S[a + 1..d])
  return start + (a - c) + (d - b)
```

Сложность

Пусть время, необходимое для нахождения оболочки над некой прямой и множеством точек S есть $T(S)$
Тогда $T(S) = O(\|S\|) + T(A \in S) + T(B \in S)$, где A, B - множества над полученными прямыми.
Отсюда видно, что в худшем случае, алгоритм тратит $O(n^2)$. На рандомных же данных это число равно $O(n \log n)$

Ссылки

- Английская статья — Wikipedia (http://en.wikipedia.org/wiki/QuickHull)

Источник — «http://neerc.ifmo.ru/wiki/index.php?title=Статические_выпуклые_оболочки:_Джарвис,_Грэхем,_Эндрю,_Чен,_QuickHull&oldid=64865»

- Эта страница последний раз была отредактирована 8 апреля 2018 в 18:07.