Методы локализации точки

Воскобойник Алексей ПМ-41

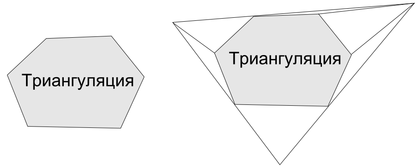
План

1. Метод детализации триагуляции Киркпатрика
   1. История
   2. Описание алгоритма
      1. Предобработка
      2. Структура данных
         1. Выбор множества удаляемых вершин
   3. Поиск
2. Локализация точки для многоугольника
   1. Для выпуклого многоугольника
3. Локализация точки в звездном многоугольнике
4. Локализация точки метод полос
5. **Метод детализации триагуляции Киркпатрика**
   1. **История**

Существует ли метод локализации со временем поиска за O(\log n), использующий менее чем квадратичную память? Эта задача оставалась не решенной довольно долго. Но все же была решена Липтоном и Тарьяном в 1977-1980 гг. Но их метод оказался на столько громоздким, а оценки времени его эффективности содержат слишком большую константу, что сами авторы не считали этот метод практичным, но его существование заставляет думать, что может найтись практичный алгоритм с временной оценкой O(\log n) и линейной памятью.

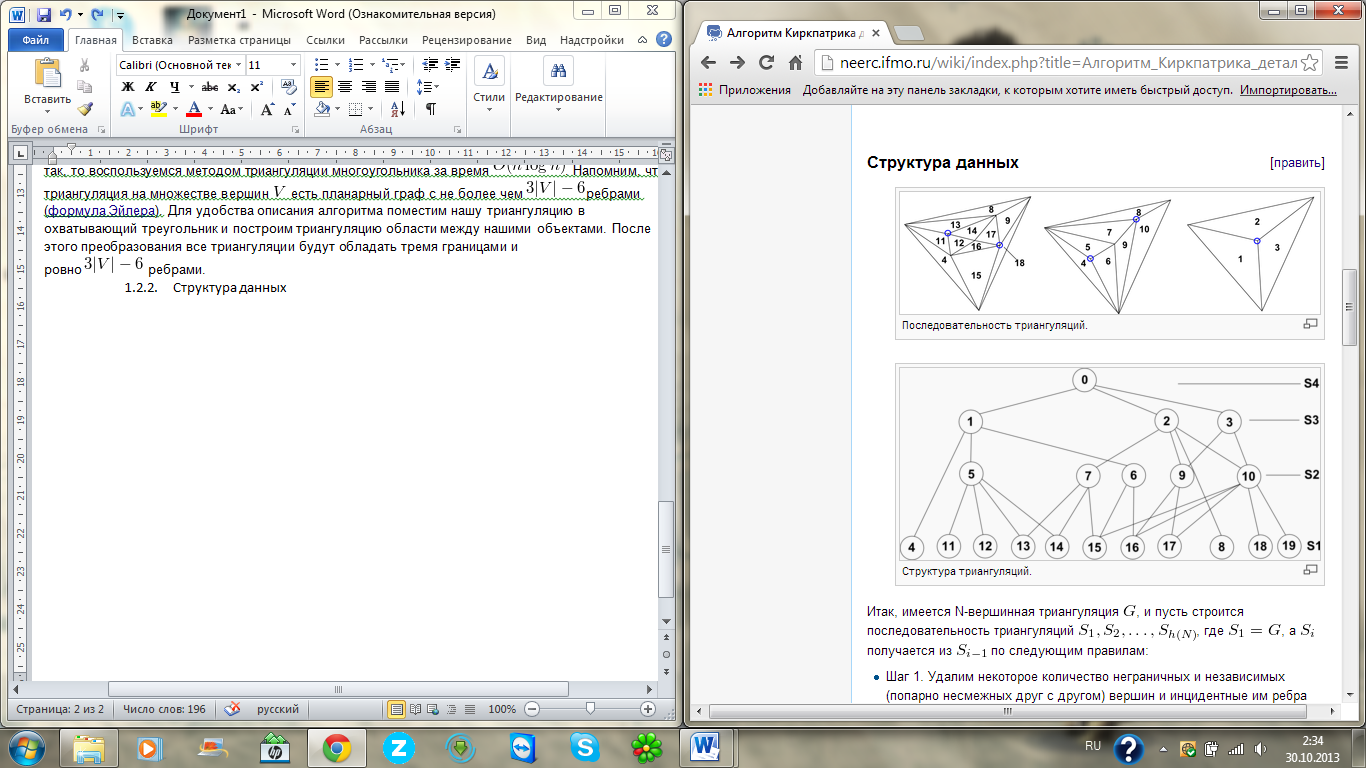
Недавно Киркпатриком был предложен оптимальный метод, дающий ответ на ожидания Липтона и Тарьяна, — детализация триангуляции.

* 1. Описание алгоритма
     1. Предобработка



Пусть планарный N-вершинный граф задает триангуляцию нашего многоугольника (если это не так, то воспользуемся методом триангуляции многоугольника за время O (n \log n). Напомним, что триангуляция на множестве вершин V есть планарный граф с не более чем 3 |V| - 6ребрами ([формула Эйлера](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0)). Для удобства описания алгоритма поместим нашу триангуляцию в охватывающий треугольник и построим триангуляцию области между нашими объектами. После этого преобразования все триангуляции будут обладать тремя границами и ровно 3 |V| - 6 ребрами.

* + 1. **Структура данных**



Итак, имеется N-вершинная триангуляция G, и пусть строится последовательность триангуляций S_1, S_2, \dots, S_{h(N)}, где S_1 = G, а S_iполучается из S_{i - 1} по следующим правилам:

* Шаг 1. Удалим некоторое количество неграничных и независимых (попарно несмежных друг с другом) вершин и инцидентные им ребра (от выбора этого множества напрямую зависит оптимальность алгоритма).
* Шаг 2. Построить триангуляцию получившихся в результате шага 1 многоугольников.

Таким образом S_{h(N)} состоит из одного треугольника. Заметим, что все триангуляции имеют одну общую границу, так как удаляются только внутренние узлы. Далее, будем обозначать все треугольники как R, а также будем говорить, что треугольник R_ij принадлежит триангуляции S_i, если он был создан на шаге (2) при построении этой триангуляции.

Теперь построим структуру данных T для поиска. Эта структура представляет собой направленный ацикличный граф, вершинами которого будут наши треугольники. Определим эту структуру следующим образом: из треугольника R_k будет вести ребро в треугольник R_j, если при построении S_i из S_{i-1} мы имеем

* R_j удалятся из S_{i - 1} на шаге (1)
* R_k создается в S_{i} на шаге (2)
* R_j \cap R_k \ne  \varnothing

Очевидно, что треугольники из S_1 (и только они) не имеют исходящих ребер.

Для ясности удобно изобразить T в рассмотренном виде, то есть помещая его узлы в горизонтальные строки, каждая из

которых соответствует какой-нибудь триангуляции. Последовательность триангуляций и соответствующая ей структура T показаны на рисунке. Треугольники пронумерованы в порядке их появления. Кружком обведены вершины, которые удалены на данном шаге.

* + - 1. **Выбор множества удаляемых вершин**

Как уже упоминалось, от выбора множества вершит триангуляции, которые будут удалены при построении S_i по S_{i-1} существенно зависит эффективность метода. Предположим, что можно выбрать это множество так, чтобы выполнялись следующие *свойства* (N_i обозначает число вершин в S_i):

**Свойство 1**. N_i = a_i N_{i-1}, где a_i \le a < 1 для i = 2,\dots , h(N).

**Свойство 2**. Каждый треугольник R_i \in S_i пересекается не более чем с H треугольниками из S_{i-1} и наоборот.

Первое свойство немедленно влечет за собой следствие, что h(N) \le \left \lceil \log_{1/a}N \right \rceil = O(log N), поскольку при переходе от S_{i-1} к S_iудаляется по меньшей мере фиксированная доля вершин.

Также из этих свойств следует, что память для T равна O(N). Действительно, заметим, что эта память используется для хранения узлов и указателей на их потомков. Из [теоремы Эйлера](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) о плоских графах следует, что S_i содержит F_i < 2N_i треугольников. Число узлов в T, представляющих треугольники из S_i, не превосходит F_i (только те треугольники, которые действительно принадлежат S_i, появляются на соответствующем «ярусе» T). Отсюда следует, что общее число узлов в T меньше, чем

2(N_1 + N_2 + \dots + N_{h(N)}) \le 2N_1(1 + a + a^2 + \dots + a^{h(N) - 1}) < \frac{2N}{1 - a}. Что касается памяти, используемой под указатели, то по свойству 2 каждый узел имеет не более H указателей, поэтому не более \frac{2NH}{1-a}указателей появится в T. Это доказывает последнее утверждение.

Покажем теперь, что критерий выбора множества удаляемых вершин, удовлетворяющий вышеописанным свойствам, существует.

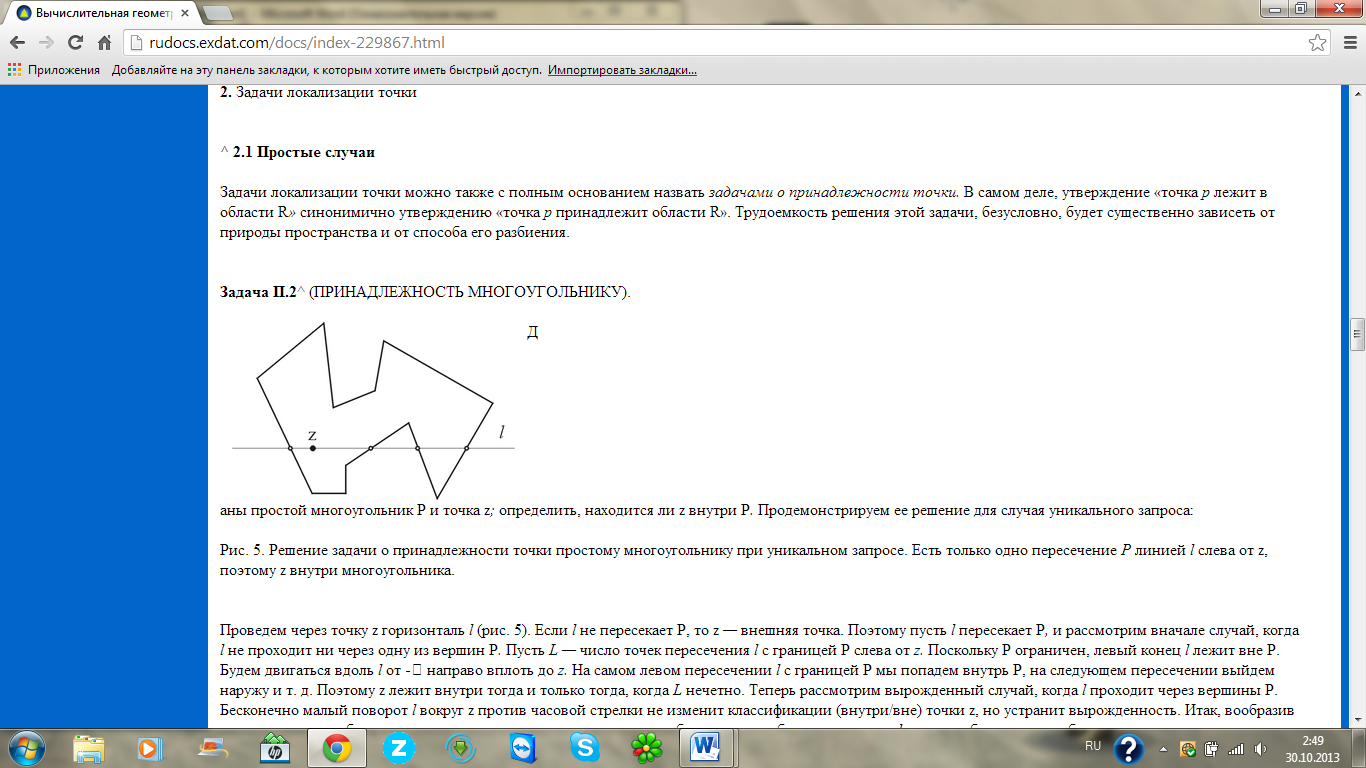
|  |
| --- |
| **Теорема (критерий выбора множества удаляемых вершин)**: |
| Если на шаге (1) построения последовательности триангуляции удалять несмежные вершины со степенью меньше некоторого целого (будет указано позже) числа K, то свойства, описанные выше, будут выполнены. |
| **Доказательство:** |
|  |
| **1.**Для проверки первого свойства воспользуемся некоторыми особенностями плоских графов. Из [формулы Эйлера](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) для плоских графов, в частном случае триангуляции, ограниченной тремя ребрами, следует, что число вершин N и число ребер e связаны соотношениемe = 3N - 6. Пока в триангнуляции есть внутренние вершины (в противном случае задача тривиальна), степень каждой из трех граничных вершин меньше трех. Поскольку существует 3N - 6 ребер, а каждое ребро инцидентно двум вершинам, то сумма степеней всех вершин меньше 6N. Отсюда сразу следует, что не менее \frac{N}{2} вершин имеет степень меньше 12. Следовательно, пусть K = 12. Пусть также v — число выбранных вершин. Поскольку каждой из них инцидентно не более K-1 = 11 ребер, а три граничные вершины не выбираются, то мы имеем v \ge \left \lfloor \frac{1}{12}(\frac{N}{2} - 3) \right \rfloor. Следовательно, a \cong 1 - \frac{1}{24} < 0,959 < 1, что доказывает справедливость свойства 1.  **2.**Выполнение второго свойства обеспечивается тривиально. Поскольку удаление вершины со степенью меньше K приводит к образованию многоугольника с числом ребер менее K, то каждый из удаленных треугольников пересекает не более K - 2 = H новых треугольников. |

* 1. **Поиск**

После построения структуры легко понять, как в ней происходит поиск. Элементарной операцией здесь является определение принадлежности треугольнику. Очевидно, что она выполняется константное время. Сначала мы локализуемся в треугольнике S_1. После этого мы строим путь от корневой вершины до листа следующим образом: находясь в какой-либо вершине z, просмотрим всех ее детей на принадлежность точки соответствующему треугольнику и, так как точка может находиться лишь в одном треугольнике конкретной триангуляции, перейдем в эту вершину, и продолжим поиск. Этот поиск также можно рассматривать как последовательную локализацию в триангуляциях S_1, \dots, S_{h(N)}, откуда и происходит название самого метода.

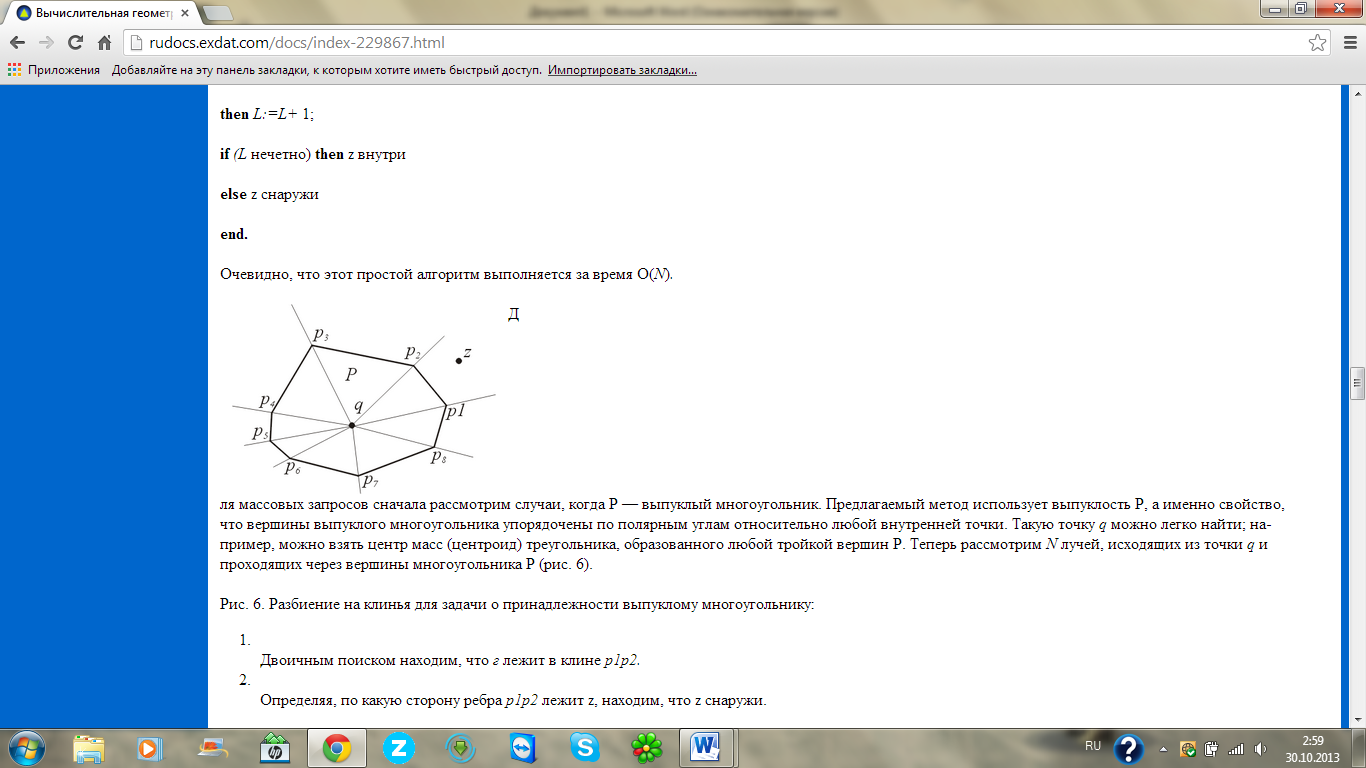
1. **Локализация точки для многоугольника**

Задачи локализации точки можно также с полным основанием назвать *задачами о принадлежности точки.* В самом деле, утверждение «точка *р* лежит в области R*»* синонимично утверждению «точка *р* принадлежит области R». Трудоемкость решения этой задачи, безусловно, будет существенно зависеть от природы пространства и от способа его разбиения.



Даны простой многоугольник Р и точка z*;* определить, находит­ся ли z внутри Р*.*Продемонстрируем ее решение для случая уникального запроса.  
 Решение задачи о принадлежности точки простому многоугольнику при уникальном запросе. Есть только одно пересечение *Р* линией *l* слева от z, поэтому z внутри многоугольника.  
Проведем через точку z горизонталь *l*. Если *l* не пересекает Р,тоz*—* внешняя точка. Поэтому пусть *l* пересекает Р*,* и рассмотрим вначале слу­чай, когда *l* не проходит ни через одну из вершин Р*.* Пусть *L —*число точек пересечения *l* с границей Р слева от *z.* Поскольку Рограничен, левый конец *l* лежит вне Р. Будем двигаться вдоль *l* от -∞ направо вплоть до *z.* На самом левом пересечении *l* с границей Р мы попадем внутрь Р, на следующем пересечении выйдем наружу и т. д. Поэтому z лежит внутри тогда и только тогда, когда *L* нечетно. Теперь рассмотрим вырожденный слу­чай, когда *l* проходит через вершины Р. Бесконечно малый по­ворот *l* вокруг z против часовой стрелки не изменит классифи­кации (внутри/вне) точки z, но устранит вырожденность. Итак, вообразив реализацию этого бесконечно малого поворота, мы увидим: если обе вершины ребра принадлежат *l,* то это ребро следует отбросить; если же ровно одна вершина ребра лежит на *l,* то пересечение будет учтено, когда эта вершина с большой ординатой, и игнорируется в противном случае. Очевидно, что этот простой алгоритм выполняется за время O(*N*)*.*

Для массовых запросов сначала рассмотрим случаи, когда Р — выпуклый многоугольник. Предлагаемый метод использует выпуклость Р, а именно свойство, что вершины выпуклого мно­гоугольника упорядочены по полярным углам относительно лю­бой внутренней точки. Такую точку *q* можно легко найти; на­пример, можно взять центр масс (центроид) треугольника, об­разованного любой тройкой вершин Р. Теперь рассмотрим *N* лу­чей, исходящих из точки *q* и проходящих через вершины мно­гоугольника Р.



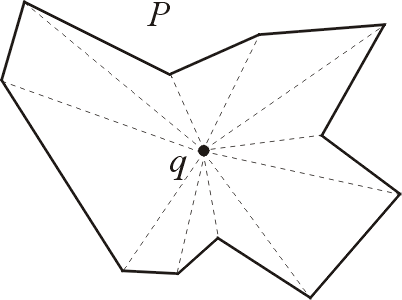
Разбиение на клинья для задачи о принадлежности выпуклому многоугольнику :  
  
 1) Двоичным поиском находим, что *z* лежит в клине *p1p2.*

2) Определяя, по какую сторону ребра *р1р2* лежит z, находим, что z сна­ружи.

Эти лучи разбивают плоскость на *N* клиньев. Каждый клин разбит на две части одним из ребер Р. Одна из этих частей ле­жит целиком внутри Р, другая—целиком снаружи. Считая *q*началом полярных координат, мы можем отыскать тот клин, где лежит точка z*,* проведя один раз двоичный поиск, поскольку лучи следуют в порядке возрастания их углов. После нахожде­ния клина остается только сравнить z с тем единственным реб­ром из Р, которое разрезает этот клин, и решить, лежит ли z внутри *Р.*  
  
 Предобработка в вышеописанном способе решения состоит в поиске *q* как центра масс трех вершин из Р и размещении вер­шин *р*1*, р*2*, ..., рN* в структуре данных, пригодной для двоич­ного поиска (например, в векторе). Очевидно, это можно про­делать за время *0(N)* для заданной последовательности *р*1*, р*2*, ...,рN*. Теперь сосредоточим внимание на процедуре поиска:

* 1. **ПРИНАДЛЕЖНОСТЬ ВЫПУКЛОМУ МНОГО­УГОЛЬНИКУ**

1. Дана пробная точка *z.* Определяем методом двоичного по­иска клин, в котором она лежит. Точка z лежит между луча­ми, определяемыми *p*i и *p*i+1*,* тогда и только тогда, когда угол (*zqp*i+1) положительный, а угол (*zqp*i) отрицательный.
2. Если *p*i и *p*i+1найдены, то z — внутренняя точка тогда и только тогда, когда угол (*p*i*pi*+1z) отрицательный.

Время ответа на запрос о принадлежности точ­ки выпуклому *^ N*-угольнику равно О(log *N*) при затрате O(*N*) памяти и O(*N*) времени на предварительную обработку.  
  
 Какое же свойство выпуклого многоугольника позволяет про­вести быстрый поиск? Чтобы можно было применить двоичный поиск, вершины должны быть упорядочены по углу относитель­но некоторой точки. Очевидно, что выпуклость является только достаточным условием обладания этим свойством. На самом же деле существует более обширный класс простых, многоугольни­ков, включающий в себя и выпуклые многоугольники, который обладает этим свойством: это класс *звезд­ных* многоугольников. Действительно, звездный многоугольник Р (рис. 7) содержит по мень­шей мере одну точку *q* та­кую, что отрезок *qp*iлежит   
целиком внутри многоуголь­ника Р для любой вершины *pi*из Р, *i*= 1, .... *N.*

1. **Локализация точки в звездном многоугольнике**

Для определения принад­лежности точки звездному многоугольнику можно непосредственно использовать предыду­щий алгоритм, если найден соответствующий центр *q,* служа­щий основой поиска. Множество подходящих центров внутри называется *ядром Р.* Построение ядра простого *N*-угольника лежит за рамками этой лекции; укажем только, что ядро может быть найдено за время О(*N*)*.*А сейчас предположим, что ядро *Р* известно (и непусто); по­этому можно выбрать центр, относительно которого строятся клинья, необходимые для поиска. Получаем аналогичные результаты, т.е. что время ответа на запрос о принадлежности точ­ки звёздному *N*-угольнику равно О(log *N*) при затрате O(*N*) памяти и O(*N*) времени на предварительную обработку.  
  
 Теперь можно обратить внимание на простые многоугольни­ки общего вида, которые будем называть обыкновенными. Су­ществует иерархия свойств, строго упорядоченная отношением «быть подмножеством»:

* Выпухлость
* Звездность
* Обыкновенность

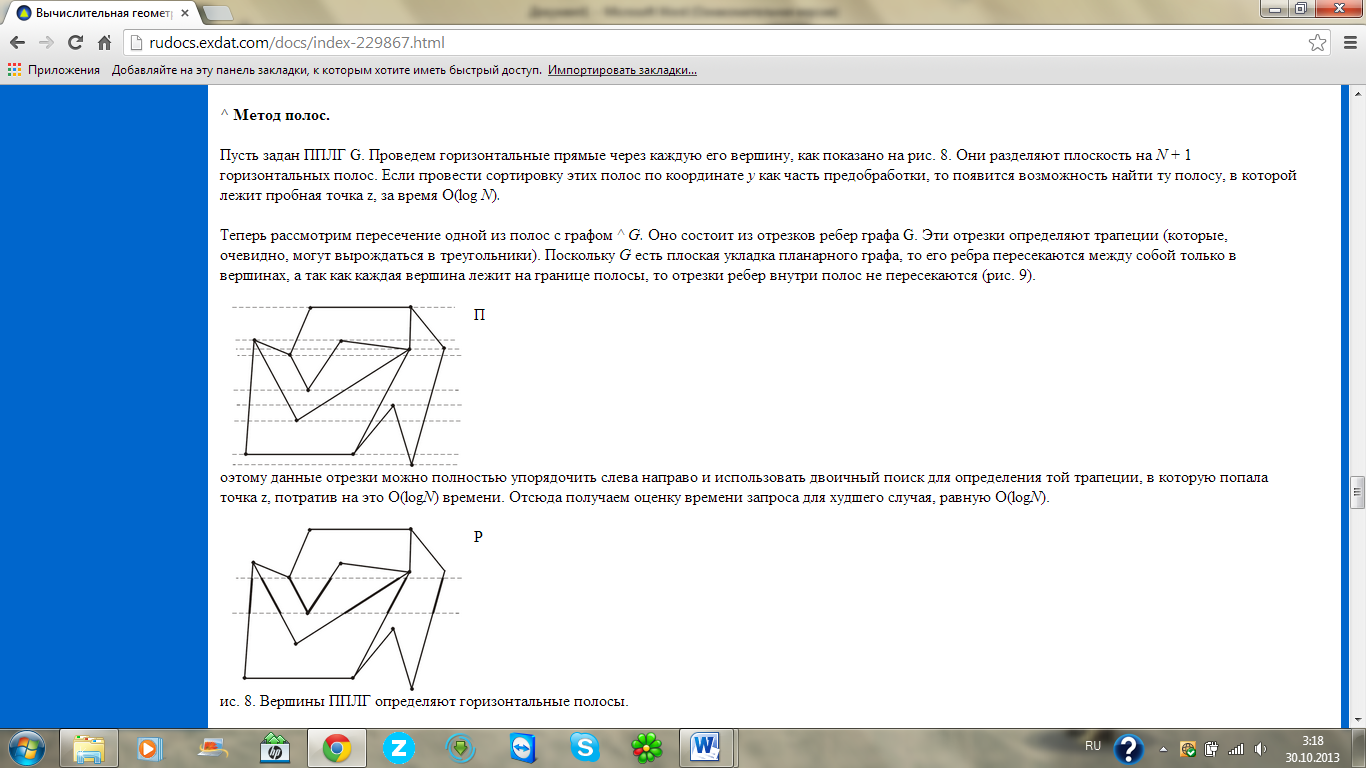
Мы только что видели, что задача о принадлежности звездному многоугольнику асимптотически ничуть не сложнее задачи о принадлежности выпуклому многоугольнику. Но что можно сказать об обыкновенном случае? Один из подходов к этой за­даче подсказан тем, что каждый простой многоугольник есть объединение некоторого числа многоугольников специального вида — таких, как звездные или выпуклые, или в конечном итоге треугольников. К сожалению, минимальная мощность А такой де­композиции может сама оказаться равной *O*(*N*). Поэтому дан­ный подход сводится к преобразованию простого *N*-угольника в *N-*вершинный плоский граф. В связи с этим кажется, что задача принадлежности обыкновенному простому многоугольнику не легче, чем, по-видимому, более общая задача о локализации точки в планарном подразбиении, хотя и неизвестно доказатель­ство их эквивалентности. Поэтому мы расскажем о по­следней из этих задач.

1. **Локализация точки метод полос**

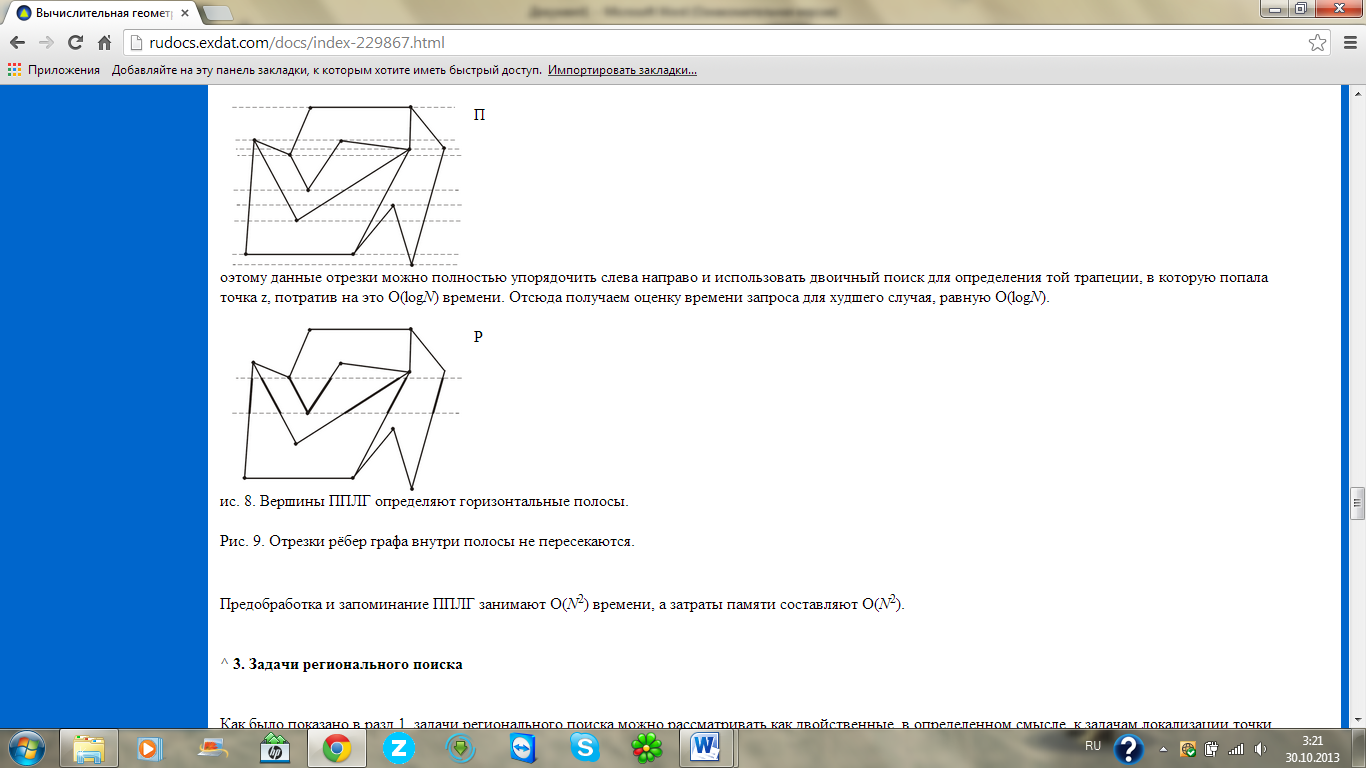
Планарный граф - это граф, который всегда может быть уложен на плоскости так, что его ребра перейдут в прямолинейные отрезки. Графы, уложенные подобным образом, будут называться *плоскими прямолинейными графами* (ППЛГ). Любой ППЛГ определяет, вообще говоря, подразбиение плоскости; если в ППЛГ нет вершин со степенью**<**2**,** то можно непосредственно убедиться в том, что все ограниченные области этого подразбиения — простые многоугольники. Без потери общ­ности здесь и далее граф предполагается связным. Рассмотрим один из методов поиска в такой структуре.

**Метод полос**

Пусть задан ППЛГ G. Проведем горизонтальные прямые через каждую его вершину, как показано на рис. 8. Они разделяют плоскость на *N* + 1 горизонтальных полос. Если провести сортировку этих полос по координате *у* как часть предобработки, то появится возможность найти ту полосу, в которой лежит пробная точка z, за время O(log *N*)*.*  
 Теперь рассмотрим пересечение одной из полос с графом *^ G.*Оно состоит из отрезков ребер графа G. Эти отрезки определяют трапеции (которые, очевидно, могут вырождаться в треугольники). Поскольку *G* есть плоская укладка планарного графа, то его ребра пересекаются между собой только в вершинах, а так как каждая вершина лежит на границе полосы, то отрезки ребер внутри полос не пересекаются. Поэтому данные отрезки можно полностью упорядочить сле­ва направо и использовать двоичный поиск для определения той трапеции, в которую попала точка z, потратив на это O(log*N*)времени. Отсюда получаем оценку времени запроса для худ­шего случая, равную O(log*N*). Вершины ППЛГ определяют горизонтальные полосы.



Отрезки рёбер графа внутри полосы не пересекаются.



Предобработка и запоминание ППЛГ занимают О(*N*2) времени, а затраты памяти составляют O(*N*2).