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 Измайлова Юлия Андреевна
фамилия, имя, отчество

Группа ФН2-31Б

Тип практики: Ознакомительная практика

Название предприятия: НУК ФН МГТУ им. Н.Э. Баумана

Студент _____ Измайлова Ю.А.
подпись, дата *фамилия и.о.*

Руководитель практики _____ Попов А.Ю.
подпись, дата *фамилия и.о.*

Оценка _____

2019 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 НУК ФН МГТУ им. Н.Э. Баумана

Студент Измайлова Юлия Андреевна
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 решения задачи идентификации положения точек в системе ячеек.
3. Реализовать алгоритм простого перебора, а также метода Киркпатрика («метода уточнения триангуляции»).

Дата выдачи задания «22» сентября 2019 г.

Руководитель практики _____ Попов А.Ю.
подпись, дата *фамилия и.о.*

Студент _____ Измайлова Ю.А.
подпись, дата *фамилия и.о.*

Содержание

Задание

Плоская многоугольная сетка задана следующим образом: дан список точек, каждая из которых задана двумя своими координатами, и список ячеек, для каждой из которых записаны последовательно против часовой стрелки номера вершин из первого списка точек. При этом гарантируется, что любые две ячейки либо не имеют общих точек, либо имеют одну общую вершину, либо примыкают к одному общему отрезку (т.е. имеют общую сторону и две общих вершины).

Требуется идентифицировать положение, т.е. установить принадлежность конкретной ячейке, системы точек, про которые известно лишь то, что они лежат строго внутри ячеек.

а) решить задачу «перебором»;

б) решить задачу эффективно, используя «метод уточнения триангуляции», называемый также методом Киркпатрика (Kirkpatrick point location method).

Структура исходного файла данных:

n	<< количество узлов сетки
x1 y1	<< координаты первого узла сетки
…	
xn yn	<< координаты n-го узла сетки
p	<< количество ячеек сетки
m1	<< количество углов в первой ячейке
k11 k12 … k1(m1)	<< номера узлов, образующих первую ячейку
…	
mp	<< количество углов в p-й ячейке
kp1 kp2 …	<< номера узлов, образующих p-ю ячейку
kp(mp) q	<< количество точек для идентификации
x1 y1	<< координаты первой точки
…	
xq yq	<< координаты q-й точки

Структура файла результата:

q	<< количество точек для идентификации
c1	<< ячейка, в которую попадает первая точка
…	
cq	<< ячейка, в которую попадает q точка

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. История рассматриваемой задачи

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 г.г. Но их метод оказался настолько 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. В 1983г. Киркпатриком был предложен оптимальный метод, дающий ответ на ожидания Липтона и Тарьяна, — детализация триангуляции [3].

2. Обзор методов решения задачи

2.1. Общие соображения. Простые случаи

Задачи локализации точки можно также с полным основанием назвать задачами о принадлежности точки области. Трудоемкость решения этой задачи, безусловно, будет существенно зависеть от природы пространства и от способа его разбиения. Даже в самом простом случае плоскости тип разбиения, на котором ведется поиск, определяет для этой задачи сложность, оцениваемую тремя основными мерами (запрос, память, предобработка).

Теорема 2.1.1. Принадлежность точки *z* внутренней области простого *N*-угольника *P* можно установить за время *O(N)* без предобработки [1] .

Метод 1. «Трассировка лучей».

Алгоритм можно описать следующим образом:

1. Из тестируемой точки выпускается луч либо в заранее заданном либо в произвольном направлении.
2. Считается количество пересечений с многоугольником.
3. Если количество пересечений четное, то точка находится снаружи. Если количество пересечений нечетное, то – внутри [4].

Теорема 2.1.2. Время ответа на запрос о принадлежности точки выпуклому N-угольнику равно $O(\log N)$ при затрате $O(N)$ памяти и $O(N)$ времени на предварительную обработку [1].

Действительно, алгоритмическая сложность данного алгоритма $O(\log n)$, где n – количество ребер в многоугольнике. Однако данный метод не всегда помогает получить верный ответ, например выпускаемый луч может пересечь вершину внутри многоугольника, либо же вообще, совпасть с одной или несколькими сторонами многоугольника.

Метод 2. «Angle weighted pseudo normal».

Алгоритм можно описать следующим образом:

1. Для тестируемой точки ищем ближайшую точку на многоугольнике. При этом помним, что ближайшая точка может быть не только на отрезке, но и в вершине.
2. Ищем нормаль ближайшей точки. Если ближняя точка лежит на ребре, то нормаль является вектор, перпендикулярный ребру и смотрящий наружу многоугольника. Если ближняя точка – одна из вершин, то нормалью является усредненный вектор ребер, прилежащих к вершине.
3. Вычисляем угол между нормалью ближайшей точки и вектором от тестируемой точки до ближайшей. Если угол меньше 90 градусов, то мы – внутри, иначе – снаружи.

Алгоритмическая сложность данного алгоритма $O(\log n)$.

Метод 3. «Индекс точки относительно многоугольника».

Рассматривается в п3 [4].

2.2 Локализация точки на планарном подразбиении

Определение 2.2.1. Планарный граф — граф, который может быть изображен на плоскости без пересечения ребер. Иначе говоря, граф планарен, если он изоморфен некоторому плоскому графу, то есть графу, изображенному на плоскости так, что его вершины — это точки плоскости, а ребра — непересекающиеся кривые на ней.

Метод 1. «Метод полос»

Локализацию точки в N-вершинном планарном подразбиении можно реализовать за время $O(\log N)$ с использованием $O(N^2)$ памяти, если $O(N^2)$ времени ушло на предобработку.

Метод 2. «Метод цепей»

Метод 3. «Метод детализации триангуляции»

Метод 4. «Метод трапеций»

3. Метод перебора, описание алгоритма

В качестве метода перебора был выбран метод «индекс точки относительно многоугольника». Возьмем единичную окружность с центром в тестируемой точке; каждую вершину многоугольника спроецируем на эту окружность лучами, которые проходят через вершину и тестируемую точку (рис. 1).

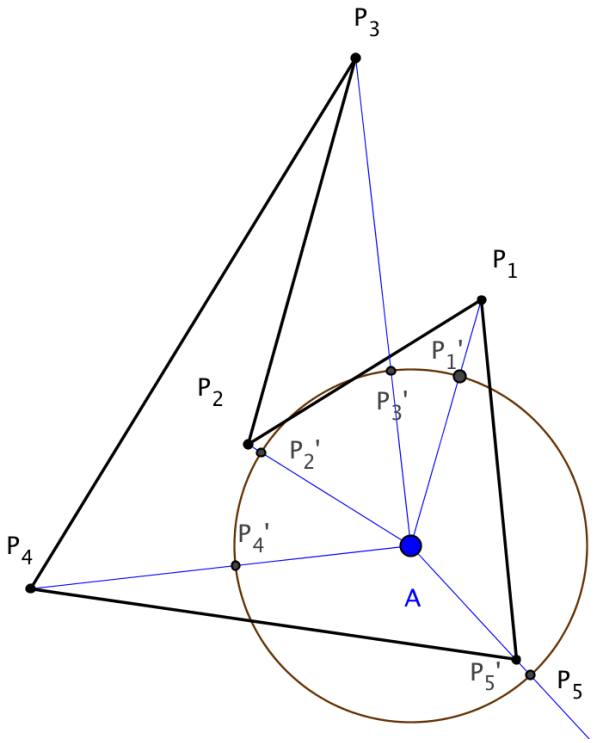

Рис. 1. Многоугольник и единичная окружность.

На рисунке точки *P1, P2* и так далее – вершины многоугольника, а точки *P1', P2'* и так далее – их проекции на окружность. Теперь, когда рассматривается ребро многоугольника, по проекциям можно определить, происходит ли враще-

ние против часовой стрелке или по часовой стрелке при переходе от одной вершины к другой. Вращение против часовой стрелки будем считать положительным поворотом, а вращение по часовой стрелке – отрицательным. Угол, который соответствует каждому ребру – это угол между сегментами окружности через проекции вершин этого ребра. Так как поворот может быть положительный или отрицательный, то и угол может быть положительный или отрицательный.

Если после этого сложить все эти углы, то сумма должна быть +360 или -360 градусов для точки находящейся внутри и 0 для точки снаружи. Если при задании порядка обхода многоугольник обходится против часовой стрелки, должно получиться +360. Если же порядок обхода ребер в многоугольнике против часовой стрелки, то получается -360. Во избежание числовых ошибок как правило делают так: делят получившуюся сумму на 360 и приводят к ближайшему целому. Получившееся число и является индексом точки. Результат может быть один из трех: -1 означает что мы внутри и многоугольник обходим по часовой стрелке, 0 означает что мы снаружи, +1 означает что мы снаружи. Все остальное означает что мы где-то ошиблись, или что входные данные некорректны. Алгоритм в этом случае следующий:

1. Устанавливаем сумму углов в 0.
2. Для всех ребер многоугольника:

2.1. С помощью скалярного произведения вычисляем угол, образованный векторами начинающимся в тестируемой точке и заканчивающимися в концах ребра.

2.2. Вычисляем векторное произведение этих же векторов. Если его z-компонента положительна – прибавляем угол к сумме углов, а иначе – вычитаем из той же суммы.

Делим сумму на 2 если считаем в радианах или на 360 если считаем в градусах; округляем результат до ближайшего целого. +1 или -1 значит, что точка находится внутри. 0 означает, что точка снаружи.

Метод прост математически, но не эффективен с точки зрения производительности. Во-первых, его алгоритмическая сложность O(n). Во-вторых, для вычисления угла придется воспользоваться операцией арккосинуса и двумя операциями взятия корня (формула скалярного произведения и связь его с углом). Эти операции очень недешевы с точки зрения скорости, и, к тому же, погрешности связанные с ними могут быть существенны. И в третьих, алгоритм напрямую не определяет точку, лежащую на ребре. Впрочем, последний недостаток легко определяется: если хотя бы один из углов равен (или почти равен) 180 градусам, это автоматически означает ребро.

Недостатки метода в чем-то компенсируются его достоинствами. Во-первых, это самый стабильный метод. Если многоугольник на вход подан корректный, то результат получается корректный для всех точек, за исключением разве что точек на ребрах [4].

4. Метод детализации триангуляции

4.1 Описание алгоритма

Определение 4.1.1. Триангуляция — это разбиение геометрического объекта на симплексы (на плоскости — треугольники).

Пусть планарный N-вершинный граф задает триангуляцию нашего многоугольника (если это не так, то воспользуемся методом триангуляции многоугольника (рис. 2, рис. 3). Для удобства описания алгоритма поместим нашу триангуляцию в охватывающий треугольник и построим триангуляцию области между нашими объектами (рис. 4).

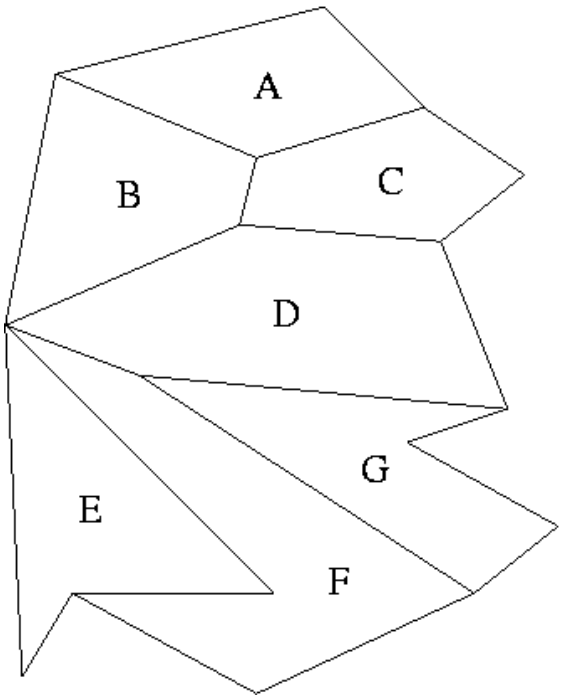

Рис. 2. Исходный многоугольник.

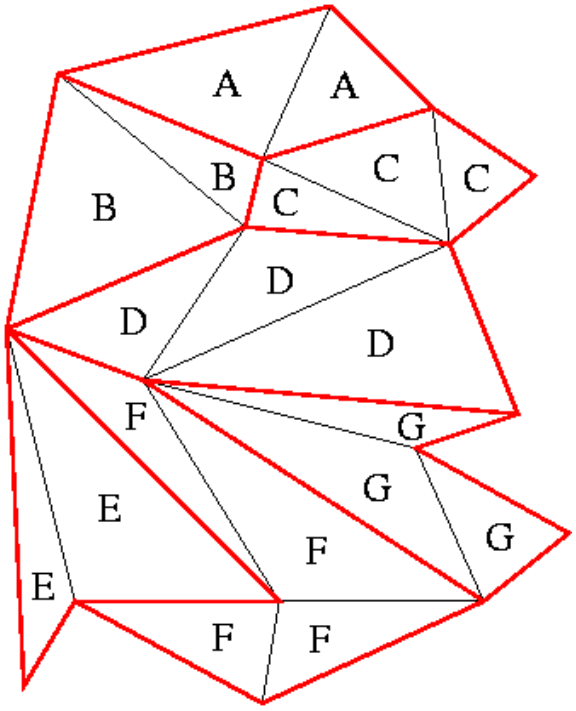

Рис. 3. Триангуляция ячеек.

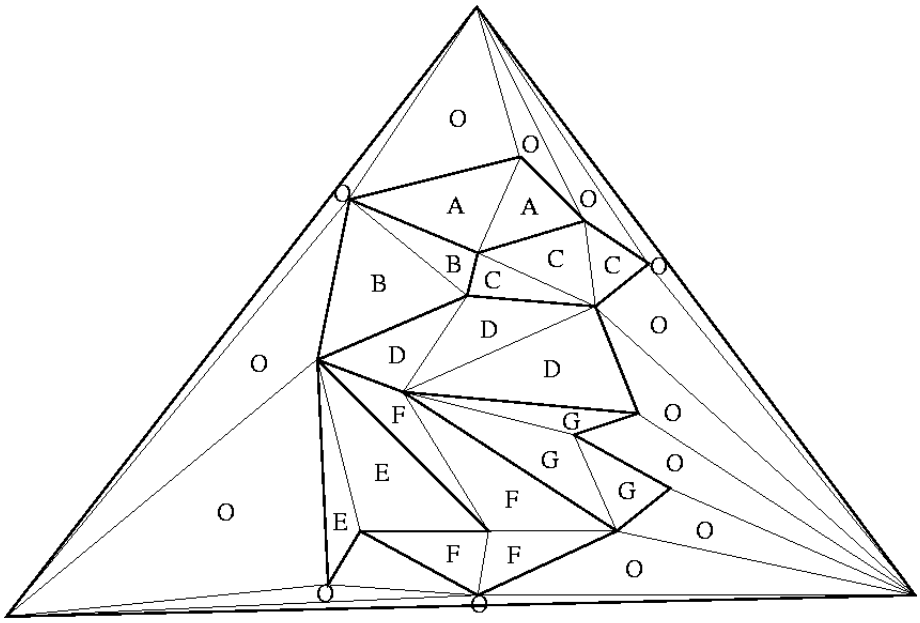

Рис. 4. Окаймляющий треугольник и триангуляция внутренних областей.

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_i получается из S_{i-1} по следующим правилам:

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) (рис. 5, рис. 6).

2. Построить триангуляцию получившихся в результате шага 1 многоугольников (рис. 7).

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.

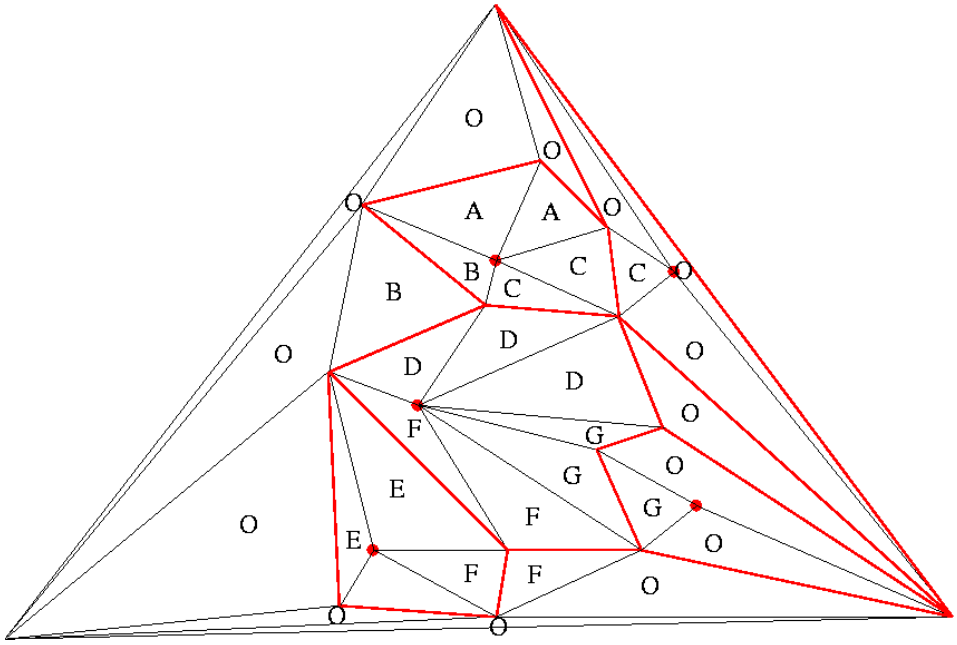

Рис. 5. Выбор независимых вершин.

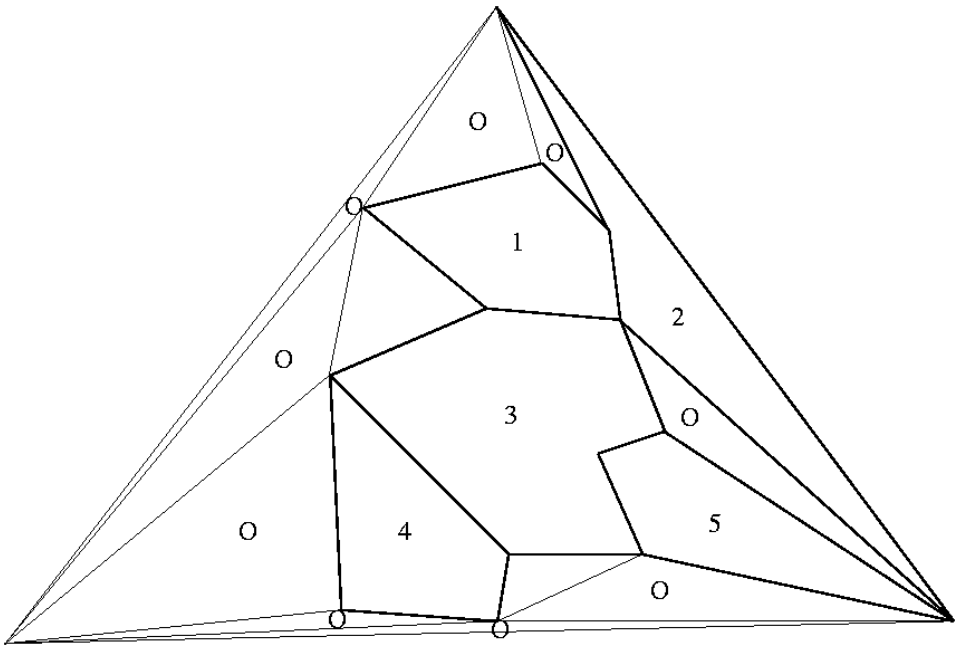

Рис. 6. Удаление вершин и соответствующих ребер.

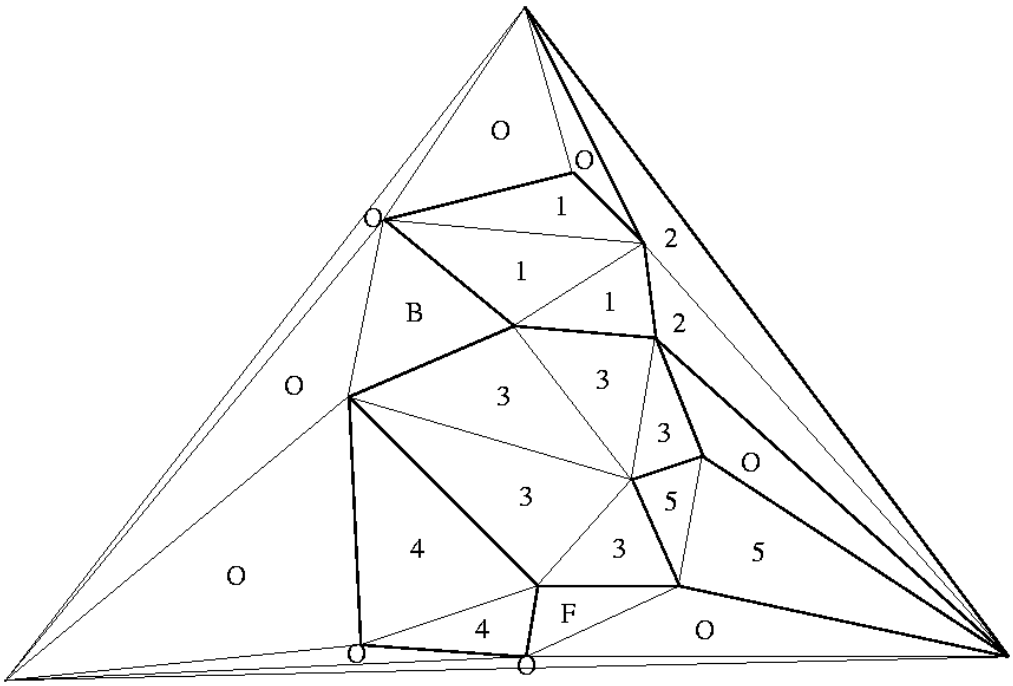

Рис. 7. Триангуляция получившихся «дырок».

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)
- $R_j \cap R_k \neq \emptyset$

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

Как уже упоминалось, от выбора множества вершин 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. $N_i = \alpha_i N_{i-1}$, где $\alpha_i \leq \alpha < 1$ для $i=2,\ldots,h(N)$.

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 $h(N) \leq [log_{1/a} N] = O(logN)$, поскольку при переходе от S_{i-1} к S_i удаляется по меньшей мере фиксированная доля вершин.

Также из этих свойств следует, что память для *T* равна *O*(*N*).

Теорема 4.1.1. (Критерий выбора множества удаляемых вершин):

Если на шаге (1) построения последовательности триангуляции удалять несмежные вершины со степенью меньше некоторого целого числа *K*, то свойства, описанные выше, будут выполнены.

Доказательство:

1. Для проверки первого свойства воспользуемся некоторыми особенностями плоских графов. Из формулы Эйлера для плоских графов, в частном случае триангуляции, ограниченной тремя ребрами, следует, что число вершин *N* и число ребер ее связаны соотношением $e = 3N - 6$. Пока в триангнуляции есть внутренние вершины (в противном случае задача тривиальна), степень каждой из трех граничных вершин не меньше трех. Поскольку существует $3N - 6$ ребер, а каждое ребро инцидентно двум вершинам, то сумма степеней всех вершин меньше 6*N*. 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 $v \geq [\frac{1}{12} (\frac{N}{2} - 3)]$. 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.

После построения структуры (рис. 8) 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, 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 [3].

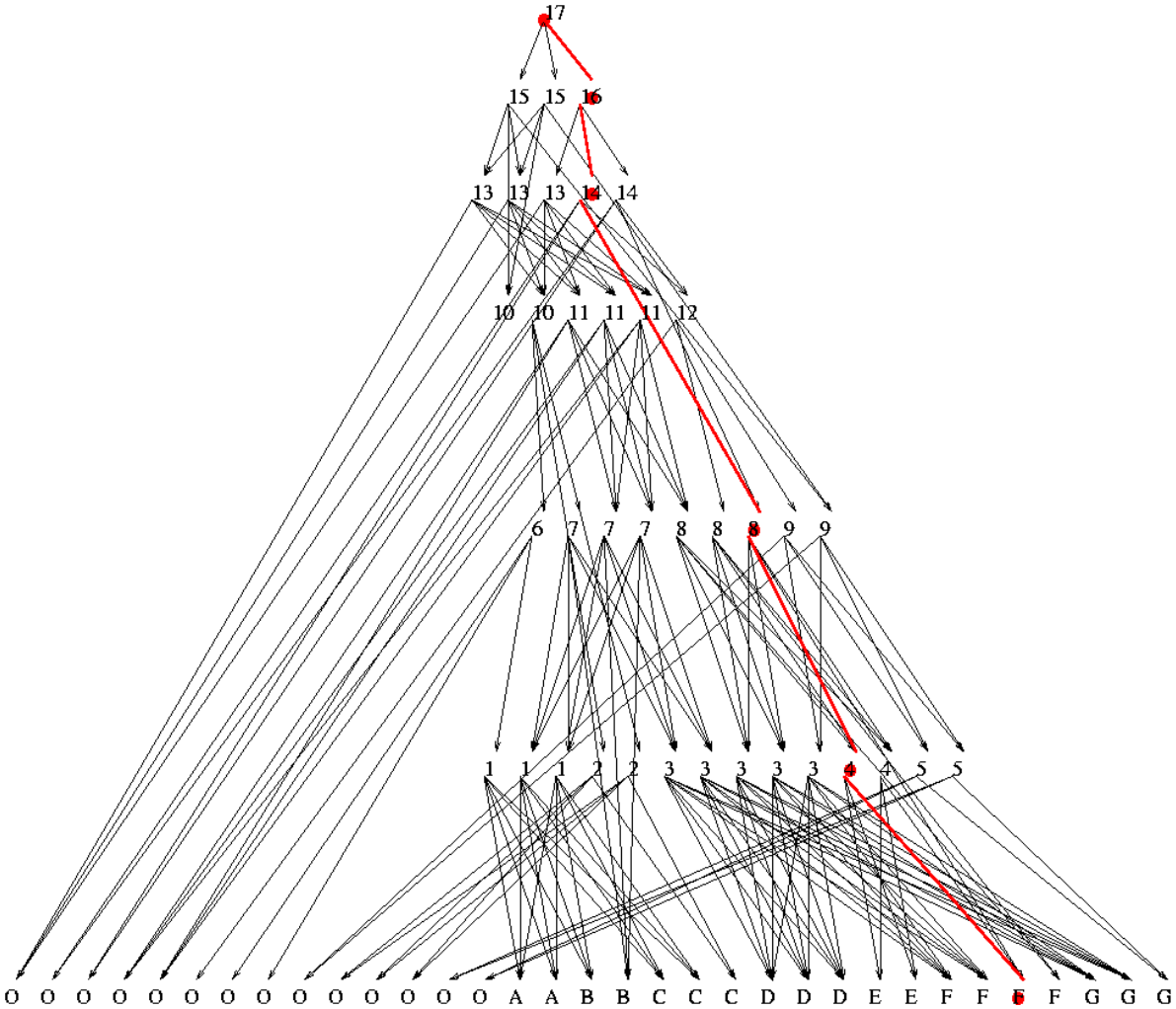

Рис. 8. Структура данных [2].

4.2. Триангуляция многоугольника. Метод «Earcut»

Определение 4.2.1. Вершина v_i называется ухом, если диагональ $v_{i-1}v_{i+1}$ лежит строго во внутренней области многоугольника P (рис. 9).

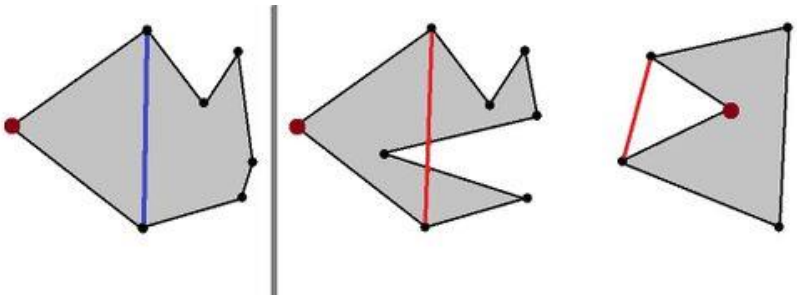

Рис. 9. В первом случае выделенная вершина является ухом, в остальных нет.

При проверке каждой вершину следует для начала проверить, является ли она выпуклой, в противном случае ее просто нет надобности рассматривать в качестве уха. Это несложно сделать, воспользовавшись левым поворотом. "Ушную" проверку вершины будем осуществлять алгоритмом принадлежности точки n-угольнику (в нашем случае треугольнику).

Рассмотрим все вершины многоугольника P, и где возможно, будем отрезать уши до тех пор, пока P не станет треугольником.

Будем рассматривать вершины многоугольника в порядке обхода. Индексирование вершин для удобства будем вести по модулю n, т.е. $v_{-1}=v_{n-1}$ и $v_0=v_n$. Если вершина v_i является ухом, построим диагональ $v_{i+1}v_{i-1}$ и отрежем треугольник $\Delta v_{i+1}v_iv_{i-1}$ от P. В противном случае переходим к следующей вершине v_{i+1} в порядке обхода.

Изначально в многоугольнике содержится O(n) ушей. Нетрудно понять, что в процессе отрезания ушей, смежные точки могут тоже становиться ушами. В результате триангуляции образуется n−3 диагонали, соответственно максимальное количество вершин, которые в процессе могут становиться ушами $2n-6$. Итого общее количество ушей будет $O(n)$. Определить, является ли вершина ухом можно за $O(n)$, поскольку используется алгоритм определения принадлежности точки треугольнику — это $O(1)$. Таким образом общий процесс отрезания ушей займет $O(n^2)$. Невыпуклых вершин всего $O(n)$, каждая из них обрабатывается за константу, поэтому общее время для их обработки $O(n)$. Списки ребер и вершин строятся за линейное время, добавление ребра и удаление вершины в каждом из них работает за константу. Общее время $O(n^2)$. Поскольку храним только два списка — память линейная [5].

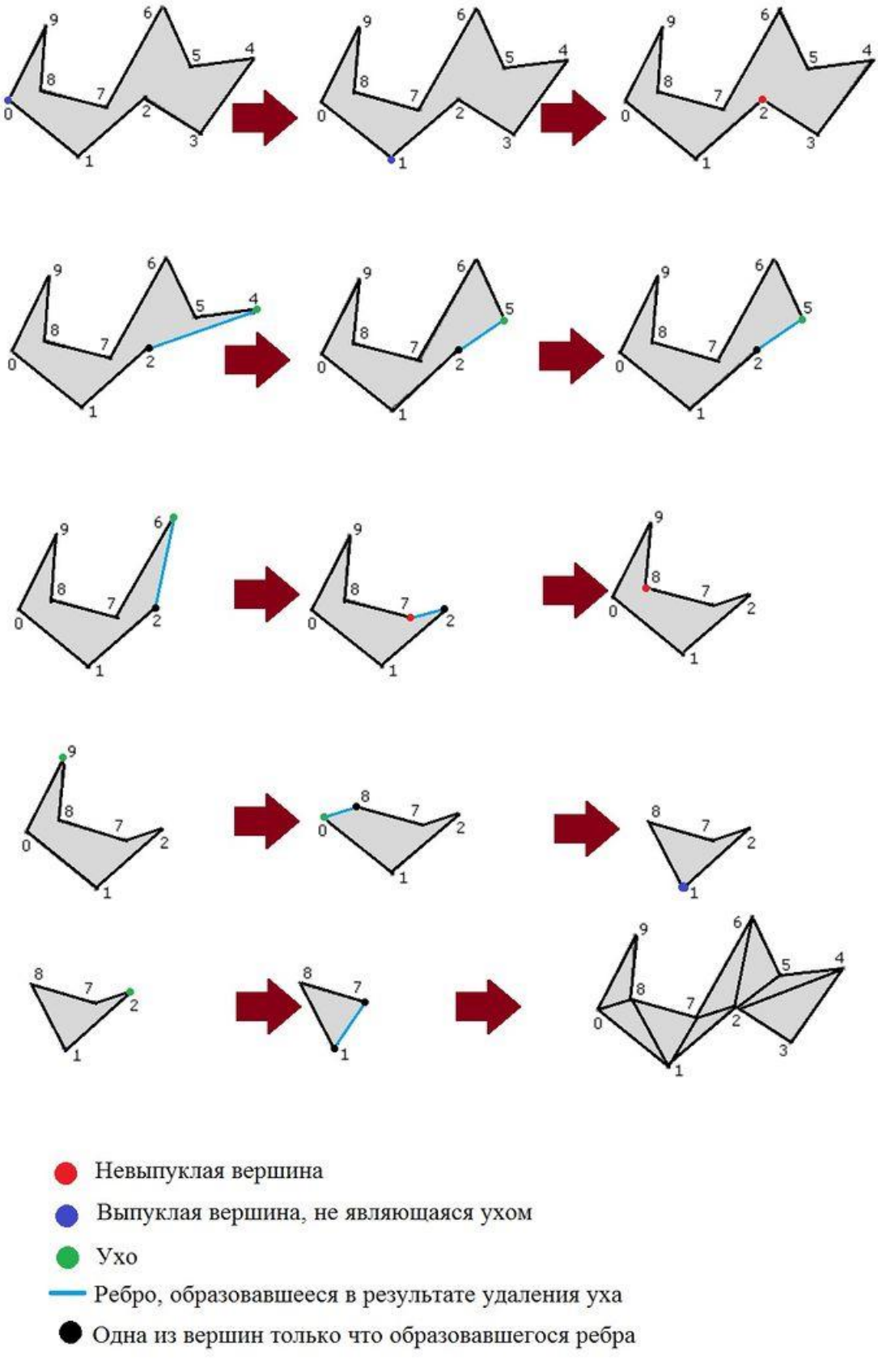

Рис. 10. Пример работы алгоритма.

4.3. Блок-схема алгоритма Киркпатрика

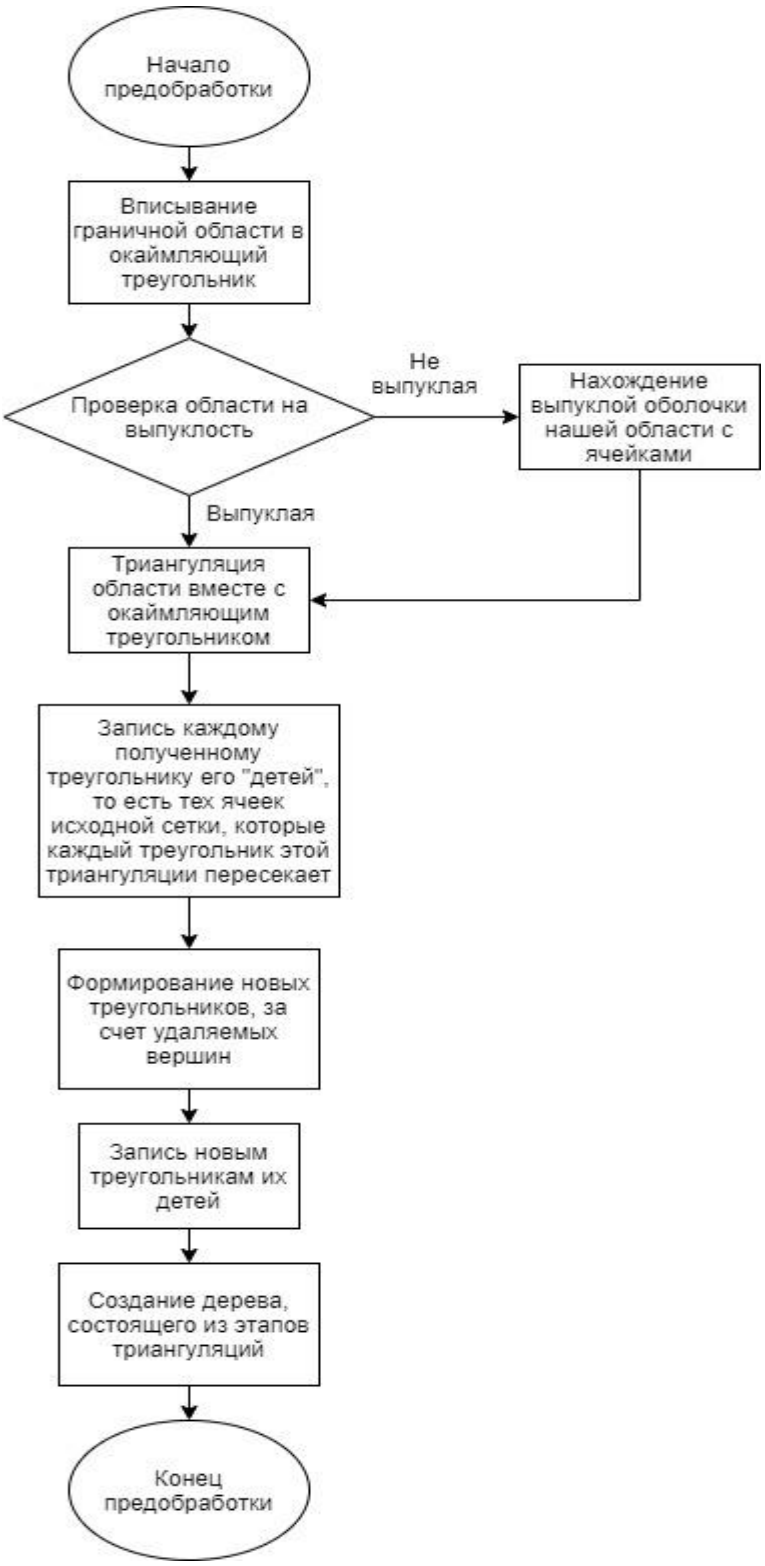

Рис. 11. Блок-схема предобработки и построения структуры данных.

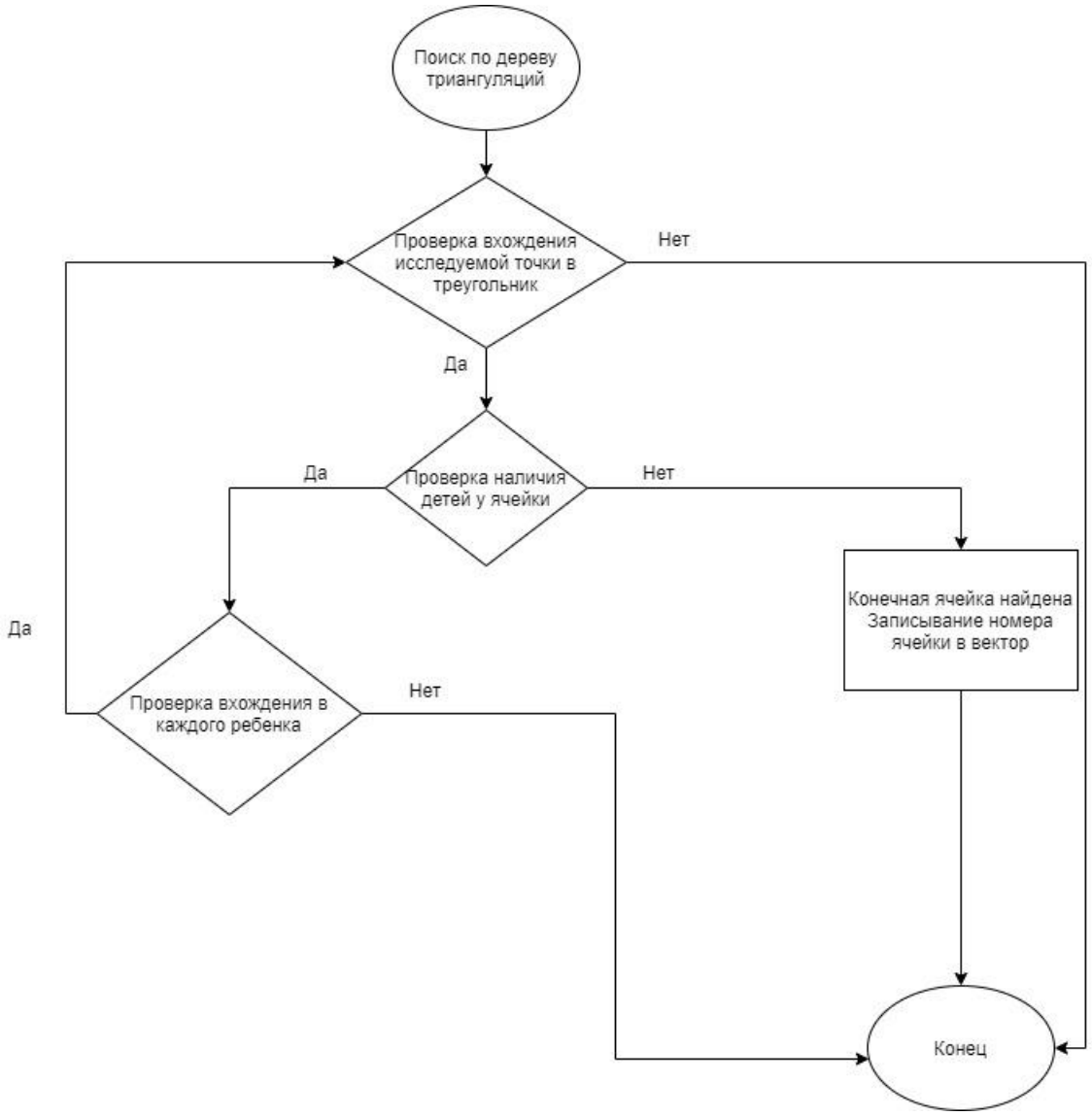

Рис. 12. Поиск точки по дереву.

5. Реализация метода перебора

5.1. Особенности реализации на языке C++

Для хранения полученных из исходного файла данных (узлы, ячейки, точки для идентификации) мы реализовали класс *loadmesh*, в котором описаны 3 метода: конструктор, принимающий строку с названием файла (*loadmesh*), поиск перебором (*points_location*), поиска с помощь алгоритма Киркпатрика (*kirkpatrick_points_location*) и который содержит 4 информационных поля – ячейки; точки для идентификации; точки, образующие ячейки, дерево триангуляций (для метода Киркпатрика). Для реализации метода перебора нам не понадобилось преобразовывать исходные данные. Мы написали функцию-член

points_location, которая является воплощением метода «индекса точки относительно многоугольника». Функция выполняет поставленную задачу и выводит результат в файл.

5.2. Особенности реализации в системе компьютерной алгебры «Wolfram Mathematica»

Для реализации метода перебора в среде *Wolfram Mathematica* нами были использованы базовые структуры — списки. Соответственно, мы создали 3 списка:

1. Список точек для идентификации (*points*);
2. Список ячеек согласно исходному файлу (*cells*);
3. Список узлов, составляющих сетку (*nodes*).

Далее использовали базовые функции для графической интерпретации исходных данных — *Graphics*, *ListPlot*, *Show*.

Реализация метода перебора в *Wolfram Mathematica* оказалась очень проста. Мы создали цикл, где прошлись по всем точкам, затем по всем ячейкам, где каждую точку проверяли на вхождение в выбранную ячейку с помощью функции *RegionMember*. Результатом работы программы является переменная *result*, которую мы записываем в файл «*Output.txt*».

6. Реализация алгоритма Киркпатрика

6.1. Особенности реализации на языке C++

Реализация алгоритма Киркпатрика начинается с создания графа (реализован класс *graph*) из узлов окаймляющего треугольника, который формируется функцией *find_outer_triangle*, принимающим как параметр выпуклую оболочку исходной сетки, а она, в свою очередь, использует функцию *convex_hull*. В последствии происходит добавление узлов оболочки в граф методом *graph::add_poly* и создание триангулированной области, состоящей из выпуклой оболочки и окаймляющего треугольника, функцией *initial_triangulation*. Затем мы удаляем инцидентные вершины и соответствующие им ребра и формируем

треугольники из соседних узлов (далее — «соседи») удаляемой вершины, функцией refinement. Процесс выбора инцидентных узлов происходит с помощью метода *graph::independent_set*. Эта функция, в зависимости от количества соседей выбранной точки, либо добавляет в вектор удаляемых точек *iset*, либо добавляет в вектор точек, которые остаются для последующего удаления *masked*.

Условия выбора точек для удаления:

1. Количество соседей выбранной точки меньше 8 (теорема, рассмотренная в п. 4).

2. Данная точка не содержится в векторе удаляемых точек.

3. Данная точка не является инцидентной точкам последующего удаления.

Помимо этого, на каждом этапе триангуляции используется функция *retriangulate*. Она формирует новые треугольники из соседей удаляемой вершины таким образом, что новообразованные треугольники являются выпуклыми и не пересекаются между собой.

После этого происходит добавление ячеек, которые пересекаются с новообразованными треугольниками (далее - детьми) и конечное удаление удаляемой точки из всех контейнеров. После прохождения цикла по всем инцидентным вершинам, мы вновь ищем новые инцидентные вершины. Выполняем эти действия до тех пор, пока не останется единственного треугольника, который являлся окаймляющим для нашей выпуклой оболочки. Во время этого процесса мы параллельно создаем дерево триангуляций (*m_top_triangle*), которое содержит все этапы трангулирования областей. Это сделано для того, чтобы осуществить очень быстрый поиск локации точки, что является целью алгоритма. Метод поиска называется *cells::query*.

6.2. Особенности реализации в системе компьютерной алгебры

Для реализации алгоритма Киркпатрика в среде *Wolfram Mathematica* мы использовали целый ряд функций. Первым делом мы находили выпуклую обо-

лочку нашей области, состоящей из узлов, с помощью функций *MeshCoordinates[RegionBoundary[ConvexHullMesh[nodes]]]*. Затем мы сортировали координаты полученной выпуклой оболочки против часовой стрелки (*sortedhull*). В целом, реализация данного алгоритма в среде *Wolfram Mathematica* одновременно очень упрощается за счет встроенных функций, и усложняется из-за работы со списками.

Алгоритм действий в *Wolfram Mathematica* аналогичен алгоритму на C++.

В данной программе мы использовали структуру *Association*, в которой хранили узлы и всех их соседей. Мы тем же способом, что и в пункте 6.1., выполняли самую первую триангуляцию области. Отдельно выполняется формирование треугольников вне оболочки, пересечение которых с оболочкой и друг другом проверяется за счет набора функций *RegionDimension[RegionIntersection[Polygon[a]]]*. Алгоритм последующих триангуляций, выбора точек для удаления тоже аналогичен алгоритму на C++, и никаких дополнительных функций нами не использовалось. Для графической интерпретации использовались те же функции *Graphics*, *ListPlot*, *Show*.

7. Примеры решения модельных задач

Пример №1

Входные данные:

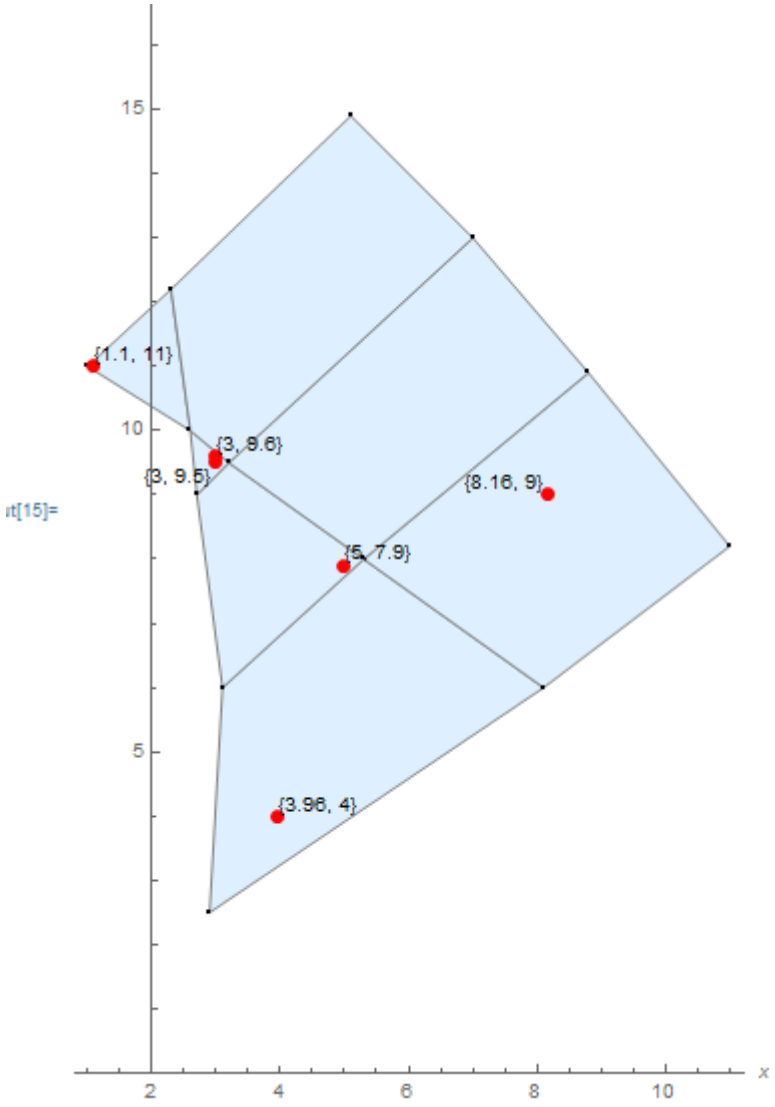

Рис. 13. Сетка для примера №1.

Результаты метода перебора на C++: за 0.001 секунды находит верно все точки и записывает в файл верный результат.

Результаты метода Киркпатрика на C++: за 0.001 секунды находит верно все точки и записывает в файл верный результат.

Результаты метода перебора в Wolfram Mathematica: за 0.1875 секунд находит верно все точки и записывает в файл верный результат.

Пример №2

Входные данные:

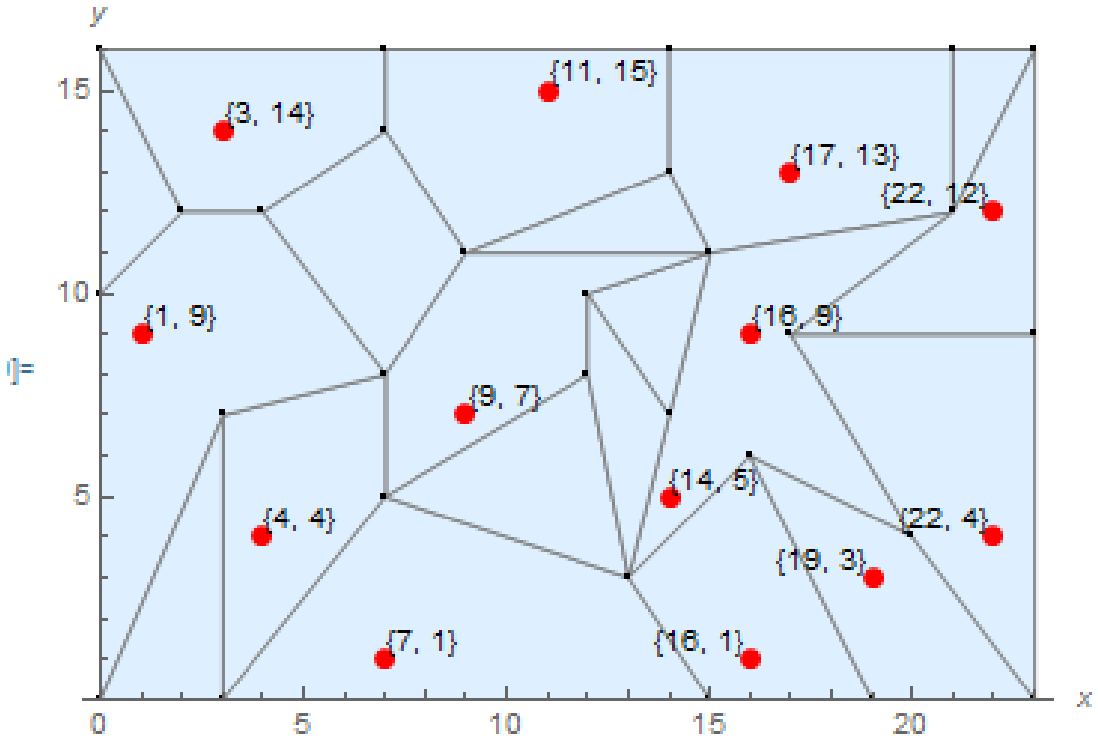

Рис. 14. Сетка для примера №2.

Результаты метода перебора на C++: за 0.002 секунды находит верно все точки и записывает в файл верный результат.

Результаты метода Киркпатрика на C++: за 0.002 секунды находит верно все точки и записывает в файл верный результат.

Результаты метода перебора в Wolfram Mathematica: за 0.1753 секунд находит верно все точки и записывает в файл верный результат.

Заключение

В ходе практики нам удалось изучить основные возможности языка программирования C++ и системы компьютерной алгебры «Wolfram Mathematica», закрепить знания и умения, полученные в курсах «Введение в информационные технологии», «Информационные технологии профессиональной деятельности».

Изучить реализации методов решения задачи идентификации положения точек в системе ячеек.

Список литературы

1. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение: Пер. с англ. — М.: Мир, 1989. — 478 с. — ISBN 5-03-001041-6.

2. The Computational Geometry Lab at McGill.

URL:

http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2002/PaulSandulescu/index.html (дата обращения: 26.12.2019).

3. Университет ИТМО // Вики-конспекты.

URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Киркпатрика_детализации_триангуляции (дата обращения: 26.12.2019).

4. Сообщество IT специалистов // Хабр.

URL https://habr.com/ru/post/301102/ (дата обращения: 26.12.2019).

5. Университет ИТМО // Вики-конспекты.

URL: https://neerc.ifmo.ru/wiki/index.php?title=Триангуляция_полигонов_(ушная_%2B_монотонная) (дата обращения: 26.12.2019).