Закраска односвязного контура

Условие задачи. На плоскости дан произвольный односвязный контур. Требуется его закрасить.

Контуром будем называть ломаную линию, конец которой совпадает с началом. Контур имеет внутреннюю область. Внутренняя область отличается от внешней тем, что любые две точки, одна из которых принадлежит внутренней области, а другая — внешней, разделяются линией контура и пройти из одной точки в другую, не пересекая линии контура, нельзя. Односвязный контур — это такой контур, что любые две точки его внутренней области можно соединить линией (не обязательно прямой), не пересекая линии контура.

Задача достаточно сложная. Есть в ней трудности и идейного характера и технического. Приступая к такой задаче, полезно решить похожую, но более простую. Это, во-первых, поможет в выработке идеи и уж во всяком случае даст полезный опыт в отработке необходимых технических навыков. Конечно, может получиться и так, что решение простого аналога окажется бесполезным, но опыт говорит, что похожие задачи часто имеют похожие решения.

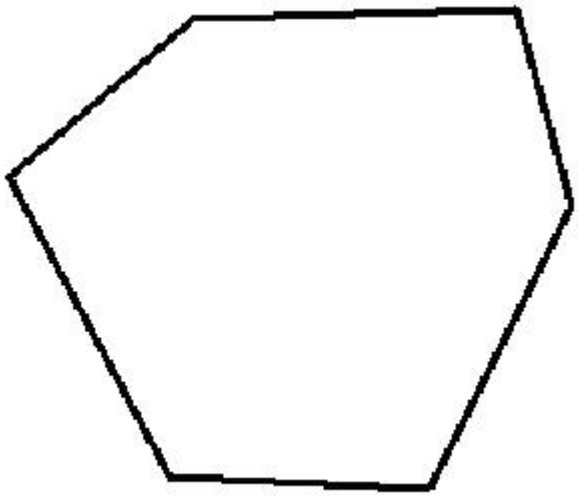
Можно, конечно, возразить и так: решая похожую задачу, мы тратим время, которое можно было бы потратить на решение исходной, это потраченное время нам никто не вернет, а решать большую задачу все равно придется с нуля. Это правильное возражение, если у вас уже есть хорошая идея, и вы четко представляете, как ее реализовать, но ведь у нас пока нет никаких идей и тем более ясного представления что делать!

Поэтому подумаем об упрощении. В условии используется три содержательных термина: "контур", "произвольный", "односвязный". Упрощение условия должно быть как-то связано с изменением смысла этих терминов. Термин "контур", конечно, изменять нельзя, он заключает в себе главный смысл задачи. "Односвязный" можно заменить, изменив связность контура, но это не облегчит, а усложнит задачу. Следовательно, остается только одна возможность — это сузить понятие "произвольный". И сразу запишем новое правило.

Правило объема терминологии

Если упрощение задачи — это изменение области применения искомого алгоритма, то следует в условии выделить существенные термины, определить область их значений и подумать, как можно уменьшить области значений без критического искажения смысла задачи.

Ограничение задачи контурами определенной формы, например, прямоугольниками, будет слишком сильным сужением, попробуем поработать с произвольными выпуклыми контурами. Выпуклый контур можно определить, как контур, любые две внутренние точки которого можно соединить прямой линией, не пересекающейся с линией контура. Хорошим примером выпуклого контура может быть любой параллелограмм или треугольник. Более сложный пример на следующей картинке.



Обсудить некоторые алгоритмические идеи.

Идея 1. Заливка. Представьте себе, как растекается вода, если ее вылить на ровную поверхность. Если поверхность будет совершенно ровной, то граница водной поверхности будет очевидно окружностью (так как все направления равноправны) и остановится вода, только достигнув контура (а контур представьте себе как некий бордюрчик). Для реализации такого процесса достаточно одной точки гарантированно находящейся во внутренней области. Затем запускаем процесс, в котором для каждой уже закрашенной точки просматриваются все соседи и для каждой соседней, не закрашенной точки выполняется закраска, после чего она рассматривается в качестве следующего источника закраски.

Описанный процесс явно рекуррентный, и если у вас появилась идея решить задачу рекурсивно, то эта мысль вполне естественна. Но против идеи рекурсивного решения есть сильное возражение. Рекурсия требует серьезных ресурсов памяти, а наш процесс будет очень сильно ветвиться, и мы рискуем сорвать закраску из-за переполнения стека. Но это будет хороший пример необходимости учитывать возможности машины и системы программирования. Иначе говоря, необходимо помнить, что алгоритм не может висеть в воздухе, он выполняется на конкретном компьютере, располагающем конкретными возможностями.

Впрочем, можно поступить иначе. Если точки, являющиеся источником закраски заносить в специальный массив записей координат, то без рекурсивных процедур можно обойтись. При очень длинном фронте разлива краски этот массив может растянуться на тысячи точек, что не слишком страшно. Если несколько тысяч одновременно работающих процедур — это много, то несколько тысяч элементов массива — это не очень много. Но тогда возникнет существенная логическая сложность. Каждая конкретная точка может являться точкой разлива краски только один шаг цикла закраски. Это означает, что массив необходимо постоянно перестраивать. Одни точки в него будут записываться, другие уходить, кроме того, длина массива не остается постоянной, и это все существенно усложнит логику алгоритма.

Может быть упомянутые логические сложности и не так страшны, мы сейчас просто обозначили их существование. Кроме того, худа без добра не бывает, и если удастся преодолеть все преграды, мы будет существенно вознаграждены.

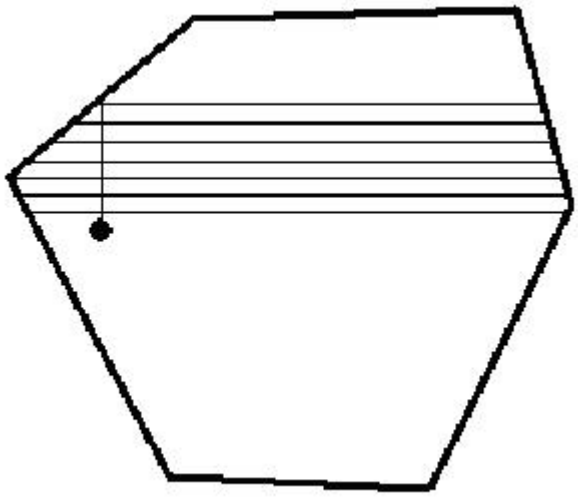
Продолжим. Каждая точка-источник видит вокруг себя только на одну точку, т. е. каждому источнику совершенно безразлична форма закрашиваемого контура. Это

означает, что мы попали на идею, которая поможет сразу решить исходную задачу. И решать вспомогательную задачу нет никакой необходимости.

Но у этого подхода есть существенный минус. Если закрашиваемая фигура велика, то фронт разливаемой краски может быть длинным, следовательно, массив, предназначенный для хранения координат точек-источников, также может оказаться большим. Это влечет за собой два недостатка: расход памяти и потерю скорости, которая будет неизбежна при перестройке массива.

Поэтому мы попробуем решить задачу двумя способами — изложенным только что и новым методом, не требующим хранения больших массивов. Такой метод нам пока неизвестен, его мы еще поищем. И для успешности поиска нового метода все же решим упрощенную задачу (с закраской выпуклого контура).

Идея 2. Закраска параллельных линий. Идея решения несложна. Мы имеем одну точку, гарантированно находящуюся внутри закрашиваемого контура. Пройдем по горизонтали влево от нее и вправо с целью найти соответственно левую и правую точки контура. Определив координаты искомых точек, проведем между ними линию. Затем поднимемся на один пиксел вверх и повторим описанные операции. Ясно, что при подъеме вверх внутренняя часть контура будет закрашиваться. Также поступим и для части контура лежащей ниже исходной точки. В построенном способе осталась одна неточность. Говорится о подъеме вверх и о спуске вниз и не говорится о том, до каких пор? Каково условие завершения подъема и соответственно спуска? Ответ на этот вопрос не так прост. Предположим, мы будем выполнять подъем до тех пор пока не встретим точку контура. Тогда возникает проблема, проиллюстрированная на следующем рисунке:



Здесь вертикальный отрезок — это линия движения от изначально известной точки до касания с контуром, а горизонтальные линии показывают закраску. Видно, что вертикальный отрезок в своем движении вверх встретил линию контура до того, как контур был закрашен в части расположенной выше исходной точки.

Проблема впрочем несложная. Ясно, что исходную точку надо просто смещать от границ контура. Делать это можно по-разному, мы используем красивый прием, позволяющий не выяснять, с какой стороны находится ближайшая линия границы.

На каждом шаге смещения вверх, для определения координат закрашивающей линии необходимо вычислять координаты X левой и правой границы контура. Обозначим их через rx и lx. Необходимо обеспечить выполнение неравенства lx<=x<=rx. Среднее арифметическое величин lx и rx всегда находится между ними. И если организовать пересчет координаты X на каждом шагу подъема, (спуска) как среднее арифметическое

крайних точек, то точка всегда будет стремиться при подъеме к самой верхней вершине и при спуске к самой нижней. Отсюда сразу видно и условие завершения подъема (спуска). Подъем (спуск) возможен только до тех пор, пока rx-lx>0, то есть до тех пор, пока левая и правая точки не совпадут.

Для разработки программы заметим, что спуск и подъем отличаются только направлением, следовательно, эти два процесса возможно реализовать одной процедурой, в которую в качестве параметров передаются координаты исходной точки и направление смещения. Операции поиска левой и правой точки отличаются только направлением, поэтому их также можно оформить одной функцией, получающей в качестве аргументов координаты точки, исходной для поиска и возвращающей координату X найденной точки.

Текст программы, записанный в листинге выглядит совершенно прозрачно, поэтому описывать алгоритм не будем. Заметим только, что часть программы, в которой выполняется ввод, закомментирована. А данные для хорошего примера вводятся посредством группы операторов присваивания.

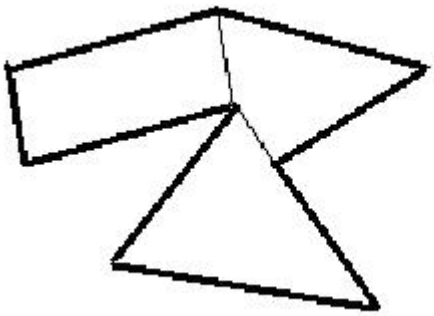
Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  point:array[1..20] of record
                         x,y:integer;
                        end;
  x,y,n,i:integer;
procedure paint(x,y,k:integer);
var
 lx,rx:integer;
function search(x,y,k:integer):integer;
begin
 while getpixel(x,y)<>15 do x:=x+k;
 search:=x;
end;
begin
  repeat
   y:=y+k;
   rx:=search(x,y,1);
   lx:=search(x,y,-1);
   line(lx,y,rx,y);
   delay(20000);
   x:=(lx+rx) div 2;
  until rx-lx=0;
end;
begin
 {read(n);
 for i:=1 to n do
   read(point[i].x,point[i].y);
 read(x,y);}
 n:=5;
 x:=160;y:=130;
 point[1].x:=150;point[1].y:=90;
 point[2].x:=260;point[2].y:=150;
 point[3].x:=240;point[3].y:=180;
 point[4].x:=200;point[4].y:=200;
 point[5].x:=100;point[5].y:=150;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to n-1 do
   line(point[i].x,point[i].y,point[i+1].x,point[i+1].y);
  line(point[n].x,point[n].y,point[1].x,point[1].y);
 paint(x,y,1);
 paint(x,y+1,-1);
```

```
 readkey;
end.
```

Если вы не сочтете за труд как следует протестировать программу, то попробуйте залить контуры, в которых хотя бы одна сторона сильно наклонена к горизонтали. Вполне возможно, вам удастся найти пример, в котором краска выльется за пределы контура. Если такой пример встретится, посмотрите внимательно на сильно наклоненную сторону. Хорошо видно, что фактически это не линия, а лесенка, между ступеньками которой могут быть пустые точки. Это небольшая техническая проблема, не имеющая принципиального значения, решать ее мы не будем, а вы попробуйте.

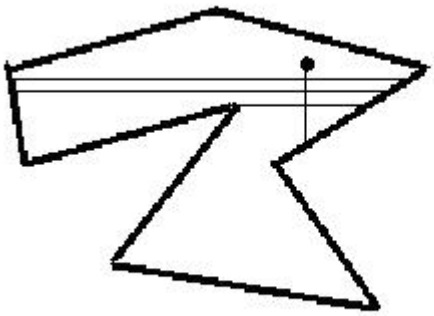
Мы же двинемся далее. Имея средство заливки краской выпуклых фигур, можно подумать о произвольном контуре, так как любой контур можно представить как объединение нескольких выпуклых. Пример такого объединения приведен на следующем рисунке.



На рисунке тонкими линиями показано деление сложной фигуры на несколько выпуклых. Таким образом, задача закраски произвольного односвязного контура свелась к задаче разбиения произвольной фигуры на несколько выпуклых. Такое разбиение, наверное, можно выполнить если знать координаты вершин контура. Но в этом случае метод закраски теряет универсальность, намного интереснее было бы найти метод, выполняющий свою работу и ничего не знающий о контуре, окружающем исходную точку. Кроме того, использование координат вершин, скорее всего, потребует достаточно трудоемких расчетов.

Конечно, было бы замечательно, если бы произвольный контур удалось закрасить, располагая только координатами одной внутренней точки. Однако логика и здравый смысл подсказывают, что резкое усложнение условия требует дополнительной информации. Это как золотое правило механики, выигрываем в расстоянии, проигрываем в силе и наоборот. Так же и здесь, выигрываем в условии (делаем его более сложным), проигрываем в известной информации (больше требований к известному). Однако золотое правило можно направить в другую сторону. Можно попробовать не требовать дополнительных условий, а попробовать разработать более сложный алгоритм.

Более сложный алгоритм, используя только текущую информацию, должен каким-то образом определять появление новой выпуклой фигуры. Если в качестве основы для рассуждений используем алгоритм закраски выпуклой фигуры, то текущая информация — это координаты левой и правой точек контура относительно точки спуска (подъема). Рассмотрим более внимательно процесс спуска на фигуре разобранной ранее.

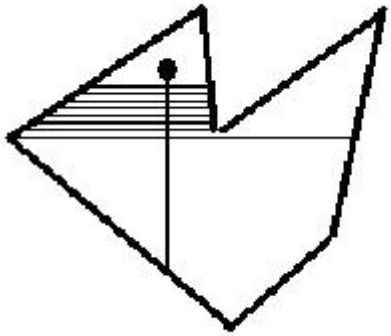


Исходная точка обозначена черным кружочком. Вертикальный отрезок показывает направление спуска. Направление показано без пересчета координаты точки спуска через среднее арифметическое левой и правой точек контура. Мы опустили пересчет для упрощения анализа. Итак, видим, что в некоторый момент горизонтальная линия резко изменяет свою длину, в данном случае она становится существенно короче, одновременно с этим событием слева от точки спуска появляется треугольник выпадающий из процесса заливки. Следовательно, можно резкое изменение длины связать с появлением новых фигур (точнее с выпадением части фигуры из процесса заливки).

Нас впрочем интересует не только факт появления новой фигуры, а и то, где она находится. Поэтому рассмотрим не просто изменение длины горизонтальной линии, а изменение координат X левой и правой точек контура. Это действительно даст больше информации. Применительно к рассмотренному примеру можно заметить, что:

- выполнялось движение вниз;
- сильно изменилось значение левой координаты, она стала меньше.

Из этого делаем вывод, что координату *X* новой точки закраски можно определить как среднее арифметическое координаты *X* предыдущей левой точки и координаты *X* текущей левой точки. При этом новая точка закраски будет точкой спуска. Для лучшего понимания рассмотрим немного иной пример:



Здесь точка закраски также спускается вниз, но изменения происходят не с левой, а с правой точкой, и ее координата не уменьшается, а увеличивается, при этом новая фигура расположена вверху, поэтому новая точка закраски будет уже точкой подъема. Двух приведенных примеров, наверное, достаточно, чтобы понять, как принципиально решается задача.

- Мы отслеживаем изменения координат левой и правой точек.

- Если координата X одной из них претерпела резкое изменение, а слишком резким изменением будем считать изменение на 2 точки (по причине различного наклона линий контура), то формируем новую точку закраски, определяем новое направление и запускаем новый процесс закраски относительно определенной точки.

Единственно, что осталось сделать, это определить возможные ситуации и для каждой определить координаты новой точки закраски и направление закраски. Возможно несколько ситуаций.

- Левая граница стала больше при движении вниз.
  - Координата X новой точки закраски есть среднее арифметическое координат X предыдущей и текущей левой точки. Новое направление закраски — вверх.

- Левая граница стала больше при движении вверх.
  - Координата X новой точки закраски есть среднее арифметическое координат X предыдущей и текущей левой точки. Новое направление закраски — вниз

- Левая граница стала меньше при движении вверх.
  - Координата X новой точки закраски есть среднее арифметическое координат X предыдущей и текущей левой точки. Новое направление закраски — вверх.

- Левая граница стала меньше при движении вниз.
  - Координата X новой точки закраски есть среднее арифметическое координат X предыдущей и текущей левой точки. Новое направление закраски — вниз.

Четыре аналогичных случая определяются и для возможных изменений правой точки. Мы их рассматривать не будем, так как аналогия практически полная. Программное решение оформим в виде рекурсивной процедуры закраски. Каждый новый вызов процедуры закраски выполняется при обнаружении одной из восьми возможных ситуаций в левой и правой точках. Текст программы приведен в листинге. Он достаточно прозрачен, поэтому расписывать алгоритм не будем.

Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  point:array[1..20] of record
                         x,y:integer;
                        end;
  x,y,n,i:integer;
procedure paint(x,y,k:integer);
var
 lx,rx,lx1,rx1:integer;
function search(x,y,k:integer):integer;
begin
 while getpixel(x,y)<>15 do x:=x+k;
 search:=x;
```

```
end;
procedure ris;
begin
   y:=y+k;
   rx:=search(x,y,1);
   lx:=search(x,y,-1);
   line(lx,y,rx,y);
   delay(2000);
   x:=(lx+rx) div 2;
end;
begin
 ris;
 lx1:=lx;rx1:=rx;
 while rx-lx>=1 do
  begin
   ris;
   if (lx-lx1>2) and (k=-1) then paint((lx+lx1) div 2,y,1);
   if (lx-lx1>2) and (k=1) then paint((lx+lx1) div 2,y-1,1);
   if (lx1-lx>2) and (k=-1) then paint((lx+lx1) div 2,y-1,-1);
   if (lx1-lx>2) and (k=1) then paint((lx+lx1) div 2,y,-1);
   if (rx-rx1>2) and (k=-1) then paint((rx+rx1) div 2,y,1);
   if (rx-rx1>2) and (k=1) then paint((rx+rx1) div 2,y,-1);
   if (rx1-rx>2) and (k=-1) then paint((rx+rx1) div 2,y,-1);
   if (rx1-rx>2) and (k=1) then paint((rx+rx1) div 2,y-1,1);
   lx1:=lx;rx1:=rx;
  end;
end;
begin
 {read(n);
 for i:=1 to n do
   read(point[i].x,point[i].y);
 read(x,y);}
 n:=9;
 x:=100;y:=110;
 point[1].x:=100;point[1].y:=100;
 point[2].x:=190;point[2].y:=170;
 point[3].x:=200;point[3].y:=200;
 point[4].x:=250;point[4].y:=150;
 point[5].x:=270;point[5].y:=280;
 point[6].x:=135;point[6].y:=180;
 point[7].x:=90;point[7].y:=280;
 point[8].x:=30;point[8].y:=170;
 point[9].x:=60;point[9].y:=200;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to n-1 do
   line(point[i].x,point[i].y,point[i+1].x,point[i+1].y);
 line(point[n].x,point[n].y,point[1].x,point[1].y);
 {putpixel(x,y,15);}
 paint(x,y,1);
 readkey;
 paint(x,y+1,-1);
 readkey;
end.
```

Полученное решение обладает теми же недостатками, которые были указаны для программы закраски выпуклого контура. Достоинством решения является малая требовательность памяти, необходимой для хранения промежуточных данных. Достаточно трудно прорисовать контур настолько сложный, что потребуется работа большого количества рекурсивных процедур закраски. Наверное, это решение можно назвать оптимальным, если исправить имеющиеся недоработки, но тем не менее мы поищем второе решение.

Второе решение основано на идее, обсуждавшейся в самом начале главы: назовем исходную точку закраски источником. Смысл этого названия в том, что точка является источником краски для четырех соседних точек. Тогда весь процесс закраски выглядит так: мы создаем массив точек источников. На каждом шагу закраски просматриваем весь массив и для каждой точки источника выполняем проверку, какие соседи еще не закрашены. Найденный не закрашенный сосед закрашивается и становится новым источником закраски. Точка, соседи которой проверялись, перестает быть источником и убирается из массива. Самый первый источник — это известная точка. Таков принцип. Но конечно же принцип — это далеко не все, еще придется сражаться с техническими проблемами реализации.

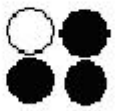
Процесс закраски на первый взгляд совершенно понятен. У каждой точки только четыре соседа (соседей по диагонали рассматривать не будем). Проверить их цвет сложности не составляет. Интуитивно ясно, что наиболее сложное место в программе — это массив точек-источников. Источники в массив должны заноситься и удаляться, соответственно длина массива постоянно меняется, а характер этих изменений зависим от формы контура.

Конечно, мы можем построить алгоритм таким образом, чтобы точки-источники на обработку выбирались не хаотично, а в определенном порядке, например с начала массива или с конца, такой порядок даже будет наиболее естественен, но предсказать сколько новых источников породит данный практически невозможно. Известно только то, что как минимум их может вообще не быть, а как максимум — четыре. Правда четыре соседа у источника может быть только на первом шаге, когда внутри контура есть только один источник — исходный.

Предположим, что точки-источники, кандидаты на обработку выбираются с начала массива. Это означает, что, взяв хотя бы один источник, мы создали в массиве одну дырку, которую можно заполнить. Если взятый источник породит только один новый источник, то мы его запишем на место использованного, а если он породит несколько источников? Или если он не породит ни одного, и дырка в начале массива начнет разрастаться на следующих шагах? Из сказанного видно, что если мы и можем брать источники на обработку только с начала, то записывать новые только в начало уже не получится. Возникшую проблему можно обойти, если источники на обработку брать не сначала, а с конца. Тогда разрывов в массиве источников не будет. Но мы пока выберем другую стратегию поведения. Введем для каждого элемента массива дополнительное описательное поле — флаг, который сыграет роль метки. Если элемент массива использован, то флаг имеет значение "истина", иначе флаг имеет значение "ложь". Тогда для выбора из массива источников кандидата на обработку достаточно найти в массиве источник с флагом, равным "истина". Данный прием сразу дает критерий завершения процесса — процесс завершается тогда, когда не нашлось ни одного источника с истинным флагом.

В нашей программе правда используется другой косвенный критерий. Ясно, что если с каждым новым источником длину массива источников увеличивать, а с каждым использованным — уменьшать, то к концу процесса длина массива окажется равной нулю. Данный факт мы и используем для завершения процесса.

Это были очевидные трудности, а есть одна неочевидная. Рассмотрим следующий рисунок:



Черные точки — это источники уже занесенные в массив источников, а белая — нет. В этой ситуации две черные точки могут объявить белую точку следующим источником, что даст удвоение потребности в размере массива в данной локальной ситуации, а фактически размер массива начнет расти почти неограниченно, так как ситуации такого типа не редкость. В нашей стратегии записи (исключения) источников, исключить источник можно только один раз, следовательно, если он окажется записанным несколько раз, балласт из лишних записей будет накапливаться быстрее, чем сбрасываться. Опытная программа, написанная без учета озвученной проблемы, показала, что удается закрасить только незначительную поверхность и существенное увеличение длинны массива источников не приводит к значительному увеличению закрашенной области.

Решение проблемы заключается в ответе на вопрос, в каком порядке выполнять действия: записать точку как источник, а потом ее закрасить (когда она будет обнаружена в поиске источника) или наоборот.

Дело в том, что после занесения в массив источников точка может достаточно долго не попасть на обработку, если она была записана в конец массива, а это как раз и означает появление реальной возможности ее повторной и даже многократной записи. Чтобы этого избежать необходимо после того, как точка попадет в массив, ее закрасить, тогда для всех прочих ее соседей она уже не сможет быть источником. Поэтому ответ на поставленный вопрос такой — точка должна быть закрашена тогда же, когда ее координаты записываются в массив источников.

Проблема с закраской/занесением делает программу не совсем тривиальной, поэтому запишем ее алгоритм:

```
Длина массива источников = 1
Записываем в массив источников исходную точку
Пока длина массива источников больше нуля, делать
  Найти первый неиспользованный источник и для него
    Объявить его использованным
    Длину массива увеличить на 1
    Провести анализ соседей следующим образом
      Если сосед незакрашенная точка, то
        Найти неиспользованный элемент массива источников.
        Сохранить в нем информацию о точке-источнике.
        Закрасить точку
        Уменьшить длину массива источников на 1
    Перейти в начало массива.
```

Обратим еще раз внимание на логику изменения длины массива источников. Уменьшение длины выполняется вполне логично. Если мы закрасили точку, то она полностью выпадает из дальнейшего анализа, ее нужно убрать из массива, в результате чего длина массива, конечно, уменьшится, но тут появляется парадокс, посмотрите внимательно, мы заносим новый источник в массив и в тот же момент уменьшаем длину массива, и наоборот, убираем точку из массива, а длину увеличиваем.

Причина уже объяснена ранее. Точка-источник закрашивается в момент занесения в массив источников. Но, конечно, такая последовательность действий в алгоритме выглядит странно, чем осложняет понимание.

Текст программы приведен в листинге.

Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  point:array[1..20] of record
                         x,y:integer;
                        end;
  front:array[1..10000] of record
                            x,y:integer;
                            flag:boolean;
                           end;
  max,x,y,n,i:integer;
procedure paint(x,y:integer);
var
 i:integer;
begin
 if getpixel(x,y)=0 then
  begin
   i:=1;
 { В цикле используется дополнительная проверка для параметра I, на тот
случай, если фронт заливки окажется слишком велик}
   while (front[i].flag) and (i<10000) do i:=i+1;
   front[i].flag:=true;
   front[i].x:=x;
   front[i].y:=y;
   putpixel(front[i].x,front[i].y,15);
   max:=max-1;
  end;
end;
begin
 {read(n);
 for i:=1 to n do
   read(point[i].x,point[i].y);
 read(x,y);}
{ n:=5;
 x:=160;y:=130;
 point[1].x:=150;point[1].y:=90;
 point[2].x:=260;point[2].y:=150;
 point[3].x:=240;point[3].y:=180;
 point[4].x:=200;point[4].y:=200;
 point[5].x:=100;point[5].y:=150;}
 n:=9;
 x:=100;y:=110;
 point[1].x:=100;point[1].y:=100;
 point[2].x:=240;point[2].y:=120;
 point[3].x:=200;point[3].y:=200;
 point[4].x:=250;point[4].y:=150;
 point[5].x:=270;point[5].y:=280;
 point[6].x:=135;point[6].y:=180;
 point[7].x:=90;point[7].y:=280;
 point[8].x:=30;point[8].y:=170;
 point[9].x:=60;point[9].y:=200;

 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 10000 do
```

```
   front[i].flag:=false;
 for i:=1 to n-1 do
   line(point[i].x,point[i].y,point[i+1].x,point[i+1].y);
 line(point[n].x,point[n].y,point[1].x,point[1].y);
 putpixel(x,y,15);
 readkey;
 max:=1;
 front[1].x:=x;front[1].y:=y;front[1].flag:=true;
 i:=0;
 repeat
  i:=i+1;
  if front[i].flag then
    begin
      front[i].flag:=false;
      delay(100);
      max:=max+1;
      paint(front[i].x,front[i].y-1);
      paint(front[i].x+1,front[i].y);
      paint(front[i].x,front[i].y+1);
      paint(front[i].x-1,front[i].y);
      i:=0;
    end;
 until max=0;
 readkey;
end.
```

Обратите внимание на последовательность, в которой программа закрашивает точки. Она действительно трудно поддается пониманию. Минусы программы — сложность логики и затраты памяти. Однако есть и компенсация. Если первый подход нуждается в доработке, напомним, что первая программа в том варианте, в котором она приведена, не может корректно закрасить контуры, содержащие линии, сильно наклоненные к горизонтали. Данная программа избавлена от такого недостатка.

А сейчас третий вариант решения. Попробуем сделать логику работы с массивом источников более естественной. Для этого будем брать на обработку источники с конца массива и выполнять запись новых также в конец.

Логика изменения массива резко упрощается. Он начинает работать, как стек. Источник, последним попавший в массив, первый попадает на обработку. В массиве нет дырок и нет балласта. Логика программы, реализующая идею, настолько прозрачна, что мы не будем записывать алгоритм, а ограничимся программой.

Запустите эту программу с первым примером. Логика выбора точек для закраски видна сразу, и можно подумать, что мы нашли идеальный алгоритм, если не удастся для первого подхода найти легкое решение сильно наклоненных линий, то, пожалуй, так оно и будет.

Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  point:array[1..20] of record
                          x,y:integer;
                        end;
  front:array[1..10000] of record
                             x,y:integer;
                           end;
  max,x,y,n,i:integer;
procedure paint(x,y:integer);
begin
 if getpixel(x,y)=0 then
```

```
  begin
   putpixel(x,y,15);
   max:=max+1;
   front[max].x:=x;
   front[max].y:=y;
  end
end;
begin
 {read(n);
 for i:=1 to n do
   read(point[i].x,point[i].y);
 read(x,y);}
 {n:=5;
 x:=160;y:=130;
 point[1].x:=150;point[1].y:=90;
 point[2].x:=260;point[2].y:=150;
 point[3].x:=240;point[3].y:=180;
 point[4].x:=200;point[4].y:=200;
 point[5].x:=100;point[5].y:=150;}
 n:=9;
 x:=100;y:=110;
 point[1].x:=100;point[1].y:=100;
 point[2].x:=240;point[2].y:=120;
 point[3].x:=200;point[3].y:=200;
 point[4].x:=250;point[4].y:=150;
 point[5].x:=270;point[5].y:=280;
 point[6].x:=135;point[6].y:=180;
 point[7].x:=90;point[7].y:=280;
 point[8].x:=30;point[8].y:=170;
 point[9].x:=60;point[9].y:=200;

 dr:=detect;initgraph(dr,md,'');
 for i:=1 to n-1 do
   line(point[i].x,point[i].y,point[i+1].x,point[i+1].y);
 line(point[n].x,point[n].y,point[1].x,point[1].y);
 readkey;
 max:=1;
 front[1].x:=x;front[1].y:=y;
 repeat
  delay(100);
  x:=front[max].x;
  y:=front[max].y;
  max:=max-1;
  paint(x,y-1);
  paint(x+1,y);
  paint(x,y+1);
  paint(x-1,y);
 until max=0;
 readkey;
end.
```