## Метод плоского заметания

Для исследования свойств геометрической конфигурации можно применить метод *плоского заметания*. Схема решения геометрической задачи методом *плоского заметания* *выглядит* следующим образом:

1. Произвольная, например, вертикальная прямая разбивает плоскость на две полуплоскости;

2. Решение, получаемое, скажем, для левой полуплоскости является окончательным, т. е. не изменяемым корректировками, предполагающимися в будущем;

3. Глобальное решение можно получить последовательным расширением текущего решения вправо. [Шеймос М Препарата Ф Вычислительная геометрия. Введение. – М.: Мир, 1989.]

Обобщением метода плоского заметания является метод *пространственного заметания*.

Рассматриваемый метод является основой алгоритма решения следующей **задачи:** у*становить, есть ли среди n данных отрезков на плоскости два пересекающихся*.

*Ограничения.* Среди рассматриваемых отрезков нет отрезков вертикальных, и никакие три отрезка не проходят через одну и ту же точку.

**Идея алгоритма** является конкретизацией описанного метода *плоского заметания*.

Воображаемая вертикальная прямая (параллельная оси ординат) движется слева направо мимо рассматриваемых геометрических объектов. При этом отрезки становятся движущимися по прямой точками. Их можно упорядочить (обычно они хранятся с помощью динамической структуры данных). Эта прямая является мгновенным срезом ситуации. Пересечение движущейся прямой с входными данными задачи содержит всю полезную для решения информацию. Информация обновляется по мере продвижения прямой.

Алгоритм просматривает концы всех отрезков слева направо и проверяет, нет ли пересечения на очередном участке оси абсцисс.

Для решения задачи введем *отношение порядка на отрезках.* Движущаяся вертикальная прямая в любой момент пересекает каждый отрезок максимум в одной точке. Упорядочиваем отрезки, пересекающие прямую, по ординате точки пересечения.

Если рассматриваемая в задаче прямая пересекается с отрезками s1 и s2,то будем считать, что отрезки *сравнимы* относительно абсциссы *x* прямой. При этом s1 выше s2 относительно *x* (s1 > s2), если отрезки s1 и s2 сравнимы относительно *x* и точка пересечения s1 с вертикальной прямой находится выше точки пересечения s2 с этой же прямой.

Например, на рис. 1. QB >r CD, EF >u CD, QB >t EF, EF >t CD, QB> t CD, отрезок GH не сравним ни с одним из отрезков.

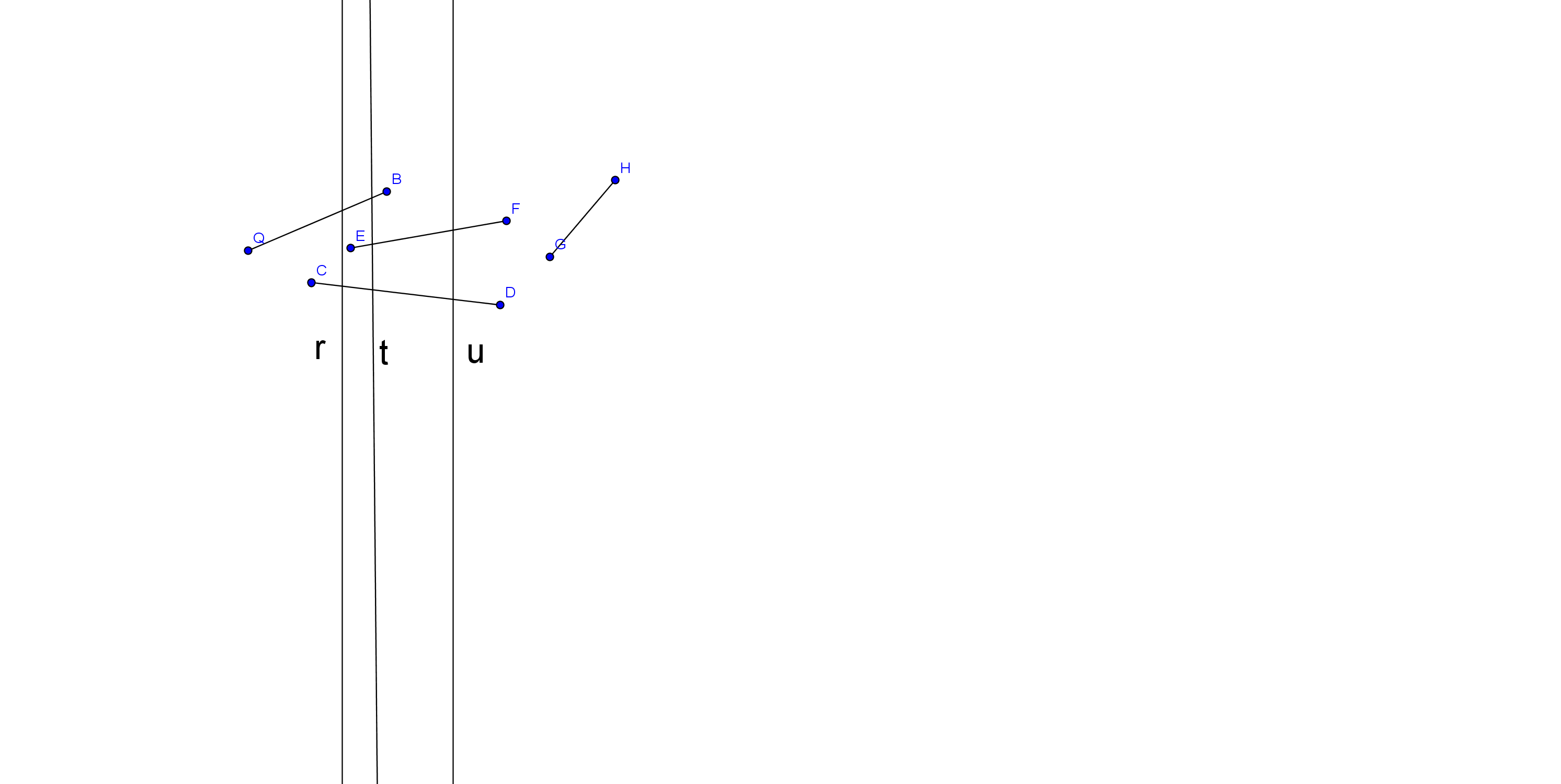


Рис. 1

Для любого фиксированного *x* отношение « >*x* » является отношением порядка на множестве отрезков, пересекающихся с вертикальной прямой, проходящей через *x*. При разных значениях *x* этот порядок (как и множество, на котором он определен) может быть различным. Отрезок попадает в это множество, когда вертикальная прямая проходит через его левый конец, и выбывает из него, когда она проходит через правый конец.

Когда прямая проходит через точку пересечения двух отрезков, отношение порядка между пересекающимися отрезками в точке пересечения меняется на противоположное. На рис. 2 AB > r CD, но CD > t AB.

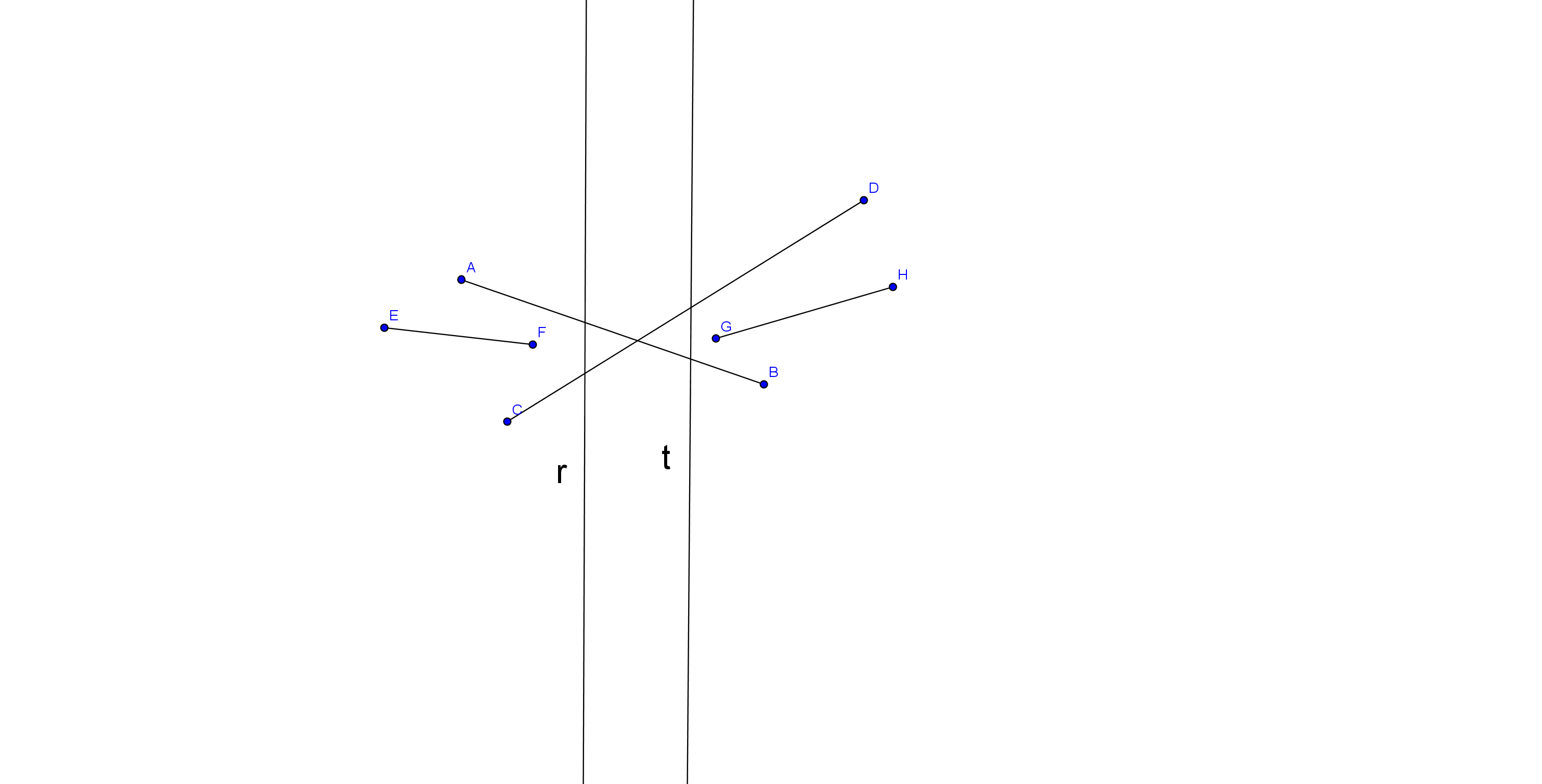


Рис. 2

По предположению никакие три отрезка не проходят через одну и ту же точку, поэтому в окрестности точки пересечения пересекающиеся отрезки непосредственно следуют один за другим (с точки зрения указанного порядка).

*Движение прямой.*

Используя метод *движущейся* прямой, мы храним следующую информацию.

1. *Состояние прямой* задается упорядоченным множеством объектов, пересекаемых движущейся прямой в текущий момент.
2. *Расписание* представляет собой последовательность моментов времени, в которых состояние может измениться (перечисленных в порядке возрастания времени). Такие моменты времени мы будем называть *критическими точками*, так что изменение состояния прямой возможно только в критической точке.

Для некоторых алгоритмом критические точки определяются постепенно по ходу работы алгоритма. В рассматриваемом алгоритме расписание определяется заранее. В частности, абсцисса конца любого отрезка является критической точкой. Упорядочим концы отрезков в порядке возрастания их абсцисс. Отрезок начинает влиять на состояние прямой с момента, когда прямая проходит через его левый конец, и перестает с момента, когда прямая проходит через его правый конец.

Состояние прямой можно хранить как упорядоченное множество отрезков, с которым выполняются следующие операции:

Insert (T, s): добавить отрезок s в T;

Delete (T, s): удалить отрезок s из T;

Above (T, s): указать отрезок, располагающийся непосредственно выше s в множестве T;

Below (T, s): возвращает отрезок, располагающийся непосредственно ниже s в множестве T.

Построим алгоритм, который по заданному множеству S, состоящему из отрезков, проверяет, есть ли среди них хотя бы два пересекающихся

Алгоритм

1 T пустое множество

2 сортируем концы отрезков в порядке возрастания абсцисс (точки с равными абсциссами идут в порядке возрастания ординат); проверяем, нет ли совпадающих точек среди концов (если есть – возвращаем TRUE)

3 for (для) каждой точки p из полученного списка

4 do if p – левый конец некоторого отрезка s

5 then Insert (T, s)

6 if (Above (T, s) существует и пересекает s)

или (Below (T, s) существует и пересекает s)

7 then return TRUE

8 if p – правый конец некоторого отрезка s

9 then if определены Above (T, s) и Below (T, s)

и Above (T, s) пересекает Below (T, s)

10 then return TRUE

11 Delete (T, s)

12 return FALSE

Изначально множество T пусто. В строке 2 строится расписание ожидаемых событий, соответствующих концам отрезков (критическими точками являются также моменты пересечения отрезков, но после обнаружения такого пересечения алгоритм заканчивает работу сразу же, так что эти моменты можно игнорировать).

Каждая итерация цикла for в строках 3-11 обрабатывает одну критическую точку. Если она соответствует левому концу некоторого отрезка s, то в строке 5 этот отрезок добавляется в упорядоченное множество, а в строках 6-7 проверяется, не пересекается ли он с одним из соседних отрезков (если да, то алгоритм кончает работу и дает ответ TRUE). (Возможно, что критическая точка соответствует концам нескольких отрезков – в этом случае они добавляются один за другим.)

Если обрабатываемая критическая точка является правым концом некоторого отрезка s, то этот отрезок удаляется из T (строка 11); предварительно проверяется, не пересекаются ли отрезки, которые разделял удаляемый отрезок и которые теперь становятся соседними (строки 9-10).

Так делается для всех концов всех отрезков; если при этом пересечение так и не обнаруживается, то алгоритм возвращает FALSE (строка 12).

**Корректность алгоритма.** *Процедура**Алгоритм возвращает* TRUE *в том и только том случае, когда среди отрезков данного множества есть пересекающиеся.*

*Доказательство*. Заметим, что значение TRUE возвращается лишь после того, как обнаружены два пересекающихся отрезка (это обеспечено структурой алгоритма). Поэтому надо лишь проверить, что если в данном множестве есть пара пересекающихся отрезков, то она будет обнаружена.

Предположим, что алгоритм не возвращает значение TRUE, а просматривает по очереди все концы отрезков, но не находит пересечения и возвращает значение FALSE. Покажем, что в таком случае отрезки не пересекаются.

Рассмотрим сначала случай, когда концы всех отрезков имеют различные абсциссы. Тогда на каждой итерации цикла рассматривается новое значение абсциссы. Докажем, что при этом остаются верными такие свойства:

А) множество T соответствует положению прямой непосредственно справа от последней обработанной абсциссы;

В) соседние в множестве T отрезки не пересекаются.

Вначале эти свойства выполнены (T пусто, движущаяся прямая не пересекает ни одного из отрезков, находясь слева от всех них). Проверим, что они остаются выполненными после прохода через один из концов (другими словами, после очередной итерации цикла).

Если очередная точка есть левый конец отрезка, то появляется новая точка пересечения с движущейся прямой. Начинающийся в ней отрезок не пересекается с соседними (в смысле текущего состояния множества T) отрезками, иначе это было бы обнаружено. Остальные пары соседних отрезков были соседними раньше, так что мы уже знаем, что они не пересекаются.

Раз соседние отрезки не пересекаются, то на участке до следующего значения абсциссы в списке относительный порядок сохраняется (точка пересечения может быть только у не соседних отрезков и только после конца того отрезка, который их разделял).

Пусть теперь очередная точка есть правый конец отрезка. Тогда из множества T удаляется элемент (как и полагается, если мы хотим, чтобы T соответствовало положению прямой справа от очередной точки). Появляется новая пара соседних отрезков – про которую алгоритм проверяет, что они не пересекаются. По тем же причинам, что и раньше, можно утверждать, что до следующего значения абсциссы относительный порядок сохраняется.

Итак, свойства А) и В) проверены, и заодно доказано, что никакие два отрезка не пересекаются. В самом деле, раз между рассматриваемыми абсциссами порядок сохраняется, между ними пересечение невозможно; а при прохождении движущейся прямой через конец отрезка пересечение также невозможно, так как некоторые пересекающиеся отрезки были бы соседними слева или справа от прямой.

Осталось разобраться, почему рассуждение применимо к случаю, когда несколько концов отрезков лежат на одной вертикали (но не совпадают – это проверяется при сортировке концов отрезков). Это можно объяснить так: представим себе, что движущаяся прямая чуть-чуть наклонена влево(против часовой стрелки). Тогда она будет пересекать концы отрезков в разные (хотя и очень близкие) моменты времени, и просматриваться они будут как раз в нужном порядке ( точки с одинаковой абсциссой – в порядке возрастания ординат). Тем самым можно свести дело к уже разобранному случаю (проверяться на пересечение будут те же пары отрезков, что и с воображаемой наклоненной прямой).