Ивановский Сергей Алексеевич,
Симончик Сергей Константинович

АЛГОРИТМЫ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ. ПЕРЕСЕЧЕНИЕ ОТРЕЗКОВ: МЕТОД ЗАМЕТАНИЯ ПЛОСКОСТИ

1. ВВЕДЕНИЕ

Тема этой статьи определяется не только желанием авторов рассказать об одной из *базовых задач* вычислительной геометрии – задаче о пересечении отрезков на плоскости, но и познакомить читателя с таким важным и эффективным приемом построения геометрических алгоритмов, как *метод заметания* плоскости прямой линией.

Задача о пересечении прямолинейных отрезков на плоскости кратко формулируется так: *дано* конечное множество отрезков; *требуется* найти их пересечения. Эта задача имеет разнообразные приложения.

В геоинформационных системах (и не только в них) используются разнообразные географические карты. На картах отображается множество объектов: населенные пункты, границы, реки и другие водоемы, овраги и горы, дороги разного типа, мосты, линии электропередач и другие промышленные объекты; на картах городов: улицы, скверы и площади, кварталы и отдельные дома, реки и каналы (см. рис. 1). Все эти объекты представляются определенными геометрическими образами, многие из которых можно считать так или иначе состоящими из прямолинейных отрезков. Поэтому определение пересечения географических объектов, например, дорог, рек, границ и т. п., сводится к нахождению пересечений множества отрезков.

Рис. 1. Примеры различных карт и планов: многие геометрические образы на них представлены (или могут быть аппроксимированы) отрезками прямых линий

В машинной графике требуются методы удаления невидимых линий и

поверхностей в трехмерных сценах, когда различные объекты сцены частично закрывают друг друга (см. рис. 2). Не имея оптимального алгоритма определения попарных пересечений отрезков из заданного набора, из которых, как правило, состоит большинство объектов сцены, трудно рассчитывать на эффективную реализацию удаления невидимых линий.

В микроэлектронике возникает необходимость проверки пересечения различных компонентов интегральных схем, состоящих из большого количества элементов (до миллиона и более), что невозможно осуществить без компьютерных методов, в том числе без алгоритмов пересечения геометрических объектов.

2. ПЕРЕСЕЧЕНИЕ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ НА ПЛОСКОСТИ. ПОСТАНОВКА ЗАДАЧИ

Уточним постановку задачи о пересечении отрезков и рассмотрим различные формы этой задачи, а также некоторые другие задачи, тесно связанные с ней.

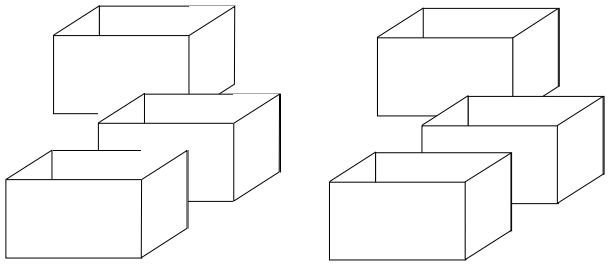


Рис. 2. Удаление невидимых линий

Задача 1. ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ (ППО). *Дано*: *n* прямолинейных отрезков на плоскости. *Требуется*: определить факт пересечения хотя бы двух из них (ответ задачи – «***да***», если пересекаются, или «***нет***» в противном случае).

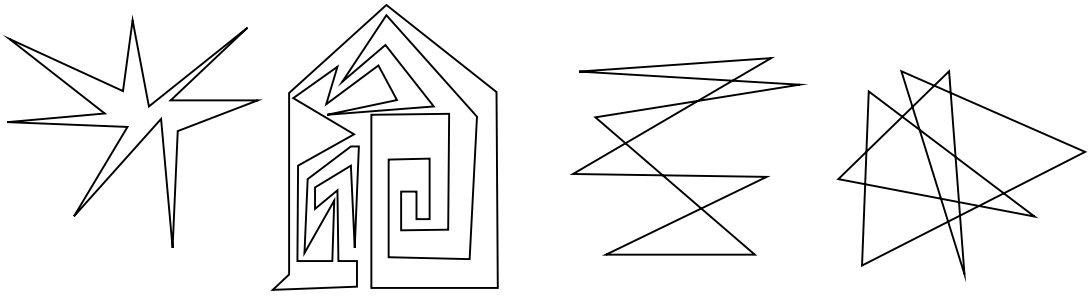
Задача 2. ВСЕ ПЕРЕСЕЧЕНИЯ ОТРЕЗКОВ (ВПО). *Дано*: *n* прямолинейных отрезков на плоскости. *Требуется*: найти все попарные пересечения отрезков.

Постановку этой задачи можно уточнить, рассматривая две её формы:

1. Задача 2.1. ВПО-П: Задача ВПО в форме *подсчета* (ответ: число попарных пересечений).

2. Задача 2.2. ВПО-О: Задача ВПО в форме *отчета* (ответ: перечень всех пар пересекающихся отрезков).

Очевидно, что задача ВПО-П не сложнее задачи ВПО-О, поскольку из ответа второй задачи можно легко (за линейное время) получить ответ первой.



а *б* *в* *г*

Рис. 3. Простые (*а* и *б*) и непростые (*в* и *г*) многоугольники

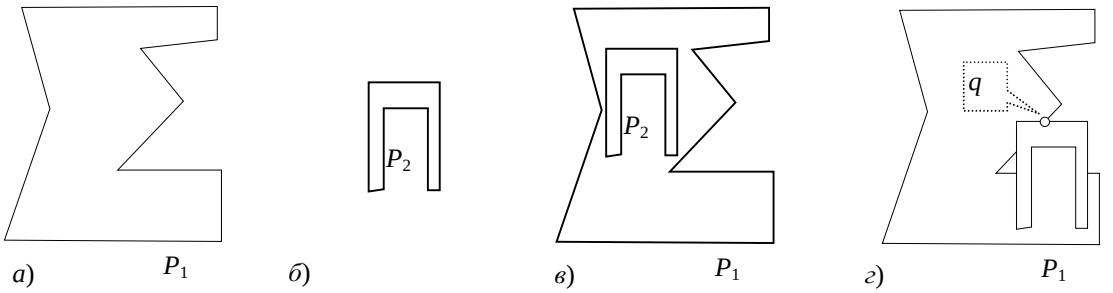


Рис. 4. Исходные многоугольники P_1 и P_2 (*а* и *б*).
Случай *в*) $P_2 \subset P_1$. Случай *г*) в точке q пересекаются ребра

Задача 3. ПРОВЕРКА ПЕРЕСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ (ППМ). *Дано*: два простых многоугольника P_1 (с n вершинами) и P_2 (с m вершинами). *Требуется*: определить, пересекаются ли они (ответ задачи – «***да***», если пересекаются, или «***нет***» в противном случае).

Напомним, что простым многоугольником называется такой многоугольник, у которого никакая пара его непоследовательных ребер не имеет общих точек. Примеры простых и непростых многоугольников приведены на рис. 3.

Так как в задаче 3 входные многоугольники P_1 и P_2 простые, то при любом пересечении их ребер это будут ребра разных многоугольников. Пусть $t(n)$ время, необходимое для решения задачи 1 (ПППО). Тогда факт пересечения многоугольников P_1 и P_2 можно определить за время $t(n + m)$. При этом если в задаче ПППО получен ответ «*да*», то это означает, что многоугольники P_1 и P_2 пересекаются (пересекаются их границы). Если в задаче ПППО получен ответ «*нет*», то необходимо проверить случаи $P_1 \subset P_2$ и $P_2 \subset P_1$ (см. рис. 4).

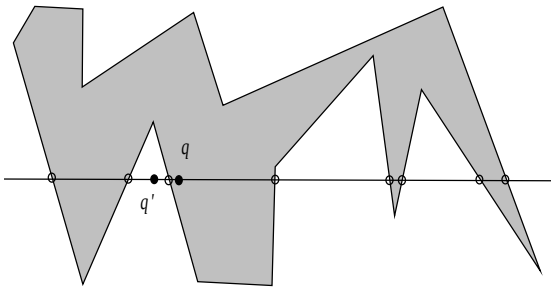


Рис. 5. Число пересечений справа от точки q равно 5, следовательно, точка q лежит внутри многоугольника. Число пересечений справа от точки q' равно 6, следовательно, точка q' лежит вне многоугольника

Проверку $P_1 \subset P_2$ можно выполнить, определив для любой вершины q многоугольника P_1 ее принадлежность многоугольнику P_2. Если вершина $q \notin P_2$, то тем же способом следует проверить $P_2 \subset P_1$. Проверка принадлежности точки q простому многоугольнику P может быть выполнена за время $O(n)$ (см., например [1]). Идея алгоритма такова (см. рис. 5): следует подсчитать число пересечений k линией $y = q_y$ сторон многоугольника, например справа от точки $q = (q_x, q_y)$, тогда если k нечетно, то точка q лежит внутри многоугольника.

В статье [2] мы уже использовали понятие *преобразования задач*. Напомним, что задача A *преобразуется* в задачу B за время $O(T(n))$ (будем обозначать это, как $A \xrightarrow{T(n)} B$), если задачу A можно решить следующим образом:

1. Из исходных данных к задаче A сконструировать соответствующие исходные данные к задаче B за время $O(T(n))$.
2. Решить задачу B.
3. Из результата решения задачи B получить результат решения задачи A за время $O(T(n))$.

Можно подвести итог нашего обсуждения связи задачи 3 о проверке пересечения многоугольников (ППМ) и задачи 1 о проверке пересечения прямолинейных отрезков (ППО):

$$\text{ППМ} \xrightarrow{n} \text{ППО}.$$

Отметим, что сложность алгоритмов, работающих с многоугольниками, может зависеть от того, известно ли, что многоугольник

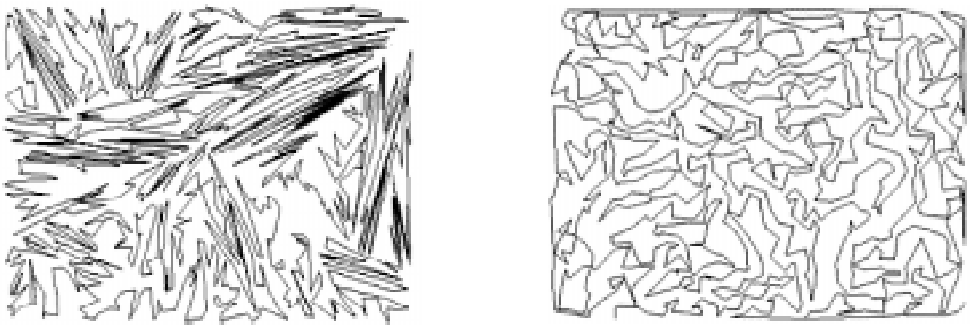


Рис. 6. Примеры простых многоугольников при $n = 1000$

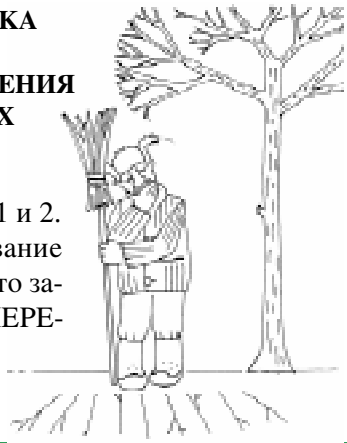
ник простой. Проверка простоты многоугольника не является простой задачей. Например, при *тестировании* программ, порождающих простые n-угольники, в случае больших значений n визуальный анализ, как правило, неосуществим. На рис. 6 в качестве примера приведены некоторые простые многоугольники при $n = 1000$. Уже здесь при данном разрешении трудно оценить простоту многоугольников, а во многих задачах, речь идет о многоугольниках размера около $n = 1\ 000\ 000$.

Задача 4. ТЕСТ ПРОСТОТЫ МНОГОУГОЛЬНИКА (ТПМ). *Дано*: многоугольник P (с n вершинами). *Требуется*: определить, прост ли он (ответ задачи – «***да***», если многоугольник простой, или «***нет***» в противном случае).

Многоугольник прост тогда и только тогда, когда никакая пара его ребер не пересекается. Следовательно, тест пересечения многоугольников (ТПМ) преобразуется в задачу проверки пересечения прямолинейных отрезков (ППО):

$$\text{ТПМ} \xrightarrow{n} \text{ППО}.$$

3. НИЖНЯЯ ОЦЕНКА СЛОЖНОСТИ ПРОВЕРКИ ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ



Рассмотрим задачи 1 и 2. Используя преобразование задач, легко показать, что задача 1 – ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ преобразуется в задачу 2.1 – ВСЕ ПЕРЕСЕЧЕНИЯ ОТРЕЗКОВ (в форме *подсчета*), которая, в свою очередь, преобразуется в задачу 2.2 – ВСЕ ПЕРЕСЕЧЕНИЯ ОТРЕЗКОВ (в форме *отчета*):

$$\text{ППО} \xrightarrow{1} \text{ВПО-П} \xrightarrow{n} \text{ВПО-О}.$$

Действительно, если иметь ответ задачи ВПО-О в виде списка пар пересекающихся отрезков, то прямым подсчетом можно получить ответ задачи ВПО-П: число пересечений k ($k \geq 0$). Далее, если $k > 0$, то ответом задачи ППО является «***да***», если же $k = 0$, то ответом будет «***нет***».

Задача ВПО-О решается очевидным алгоритмом: перебираются все $\frac{n(n-1)}{2}$ пары отрезков и каждая пара проверяется на пересечение. Этот «лобовой» алгоритм имеет сложность $O(n^2)$. Этот же алгоритм с учетов описанных преобразований задач 1 и 2 решает и задачи ВПО-П и ППО. Естественно задать вопрос: можно ли решить все эти задачи каким-либо более эффективным алгоритмом? Для ответа на этот вопрос целесообразно сначала выявить нижнюю оценку сложности задачи ППО, то есть выяснить, каково необходимое время гарантированного решения задачи при произвольных входных данных. Имеется в виду, что всегда можно предъявить для алгоритма, решающего задачу, такие исходные данные, что время работы алгоритма (или количество операций алгоритма) будет не менее некоторой определенной величины, зависящей от размера входных данных задачи (в нашем случае от количества отрезков n).

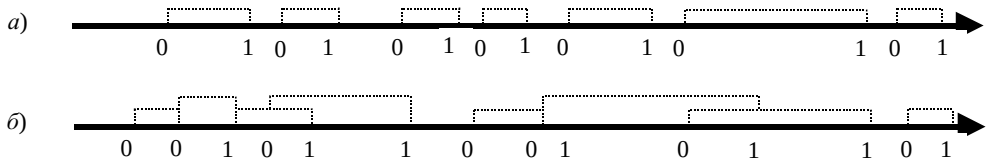


Рис. 7. Проверка наложения интервалов:
а) последовательность «01010101010101» чередующаяся – нет наложений;
б) последовательность «00101100101101» не чередующаяся – есть наложения интервалов

Для выявления нижней оценки рассмотрим задачу ППП0 в одномерном случае: все заданные отрезки расположены на одной прямой, и необходимо проверить, пересекаются ли хотя бы два из них. Ясно, что задача на плоскости не проще, чем задача на прямой. В одномерном случае задачу можно переформулировать так: заданы n интервалов на вещественной оси и необходимо узнать, не перекрываются ли какие-нибудь два из них. Конечно, наш «лобовой» алгоритм решает эту задачу за время $O(n^2)$. Можно поступить иначе. Пометим все левые концы интервалов знаком «0», а все правые – знаком «1» и упорядочим по возрастанию $2n$ концевых точек заданных интервалов за время $O(n \log n)$. Интервалы не перекрываются тогда и только тогда, когда после сортировки последовательность пометок является чередующейся – «0101...0101» (см. рис. 7).

Проверку чередования можно провести за время $O(n)$ и, следовательно, решить всю задачу за время $O(n \log n)$.

Фактически этот алгоритм является преобразованием задачи ППП0 в задачу сортировки и дает верхнюю оценку сложности. Для получения нижней оценки требуется преобразовать некоторую задачу с известной нижней оценкой (вычислительный прототип) в задачу ППП0. Задача сортировки на эту роль не подходит. Рассмотрим задачу ПРОВЕРКИ ЕДИНСТВЕННОСТИ ЭЛЕМЕНТОВ (ПЕЭ): даны n вещественных чисел, требуется проверить, все ли они различны. Очевидно, задача ПЕЭ преобразуется за время $O(n)$ в задачу сортировки. Однако обратного преобразования нет. Оказывается, что нижняя оценка задачи ПЕЭ есть $\Omega(n \log n)$ и этот факт устанавливается прямым использованием формальной вычислительной модели, основанной на деревьях решений (см. [1], с. 232). Покажем, что задача ППП0 преобразуется в задачу ПЕЭ. Действительно, за время $O(n)$ заданный набор из n вещественных чисел $\{x_i\}$ можно преобразовать в набор из n интервалов $\{[x_i, x_i]\}$. Эти (вырожденные) интервалы перекрываются тогда и только тогда, когда среди исходных чисел $\{x_i\}$ есть совпадающие. Следовательно, ПЕЭ $\xrightarrow{n}$ ППП0, и нижняя оценка задачи ППП0 есть $\Omega(n \log n)$, так как решив задачу ППП0 за меньшее, чем $\Omega(n \log n)$, время, мы могли бы и задачу ПЕЭ решить за меньшее время, что невозможно.

Итак, нижняя оценка сложности задач о пересечении отрезков на плоскости (задач ППП0, ВП0-П и ВП0-О) есть $\Omega(n \log n)$. Достижима ли она, то есть существуют ли алгоритмы, решающие перечисленные задачи за время $O(n \log n)$? Ответ на этот вопрос будет дан в разделе 5.

4. БАЗОВАЯ ОПЕРАЦИЯ «ПЕРЕСЕЧЕНИЕ ДВУХ ОТРЕЗКОВ»

Определяя нижнюю оценку сложности задач о пересечении отрезков, мы считали, что базовые операции, используемые в этих задачах, выполняются за время $O(1)$. Можно предположить, что основной базовой операцией будет проверка пересечения пары отрезков.

Определять факт пересечения пары отрезков можно в два этапа. Сначала определяется факт пересечения *ограничивающих прямоугольников* каждого из отрезков (этот

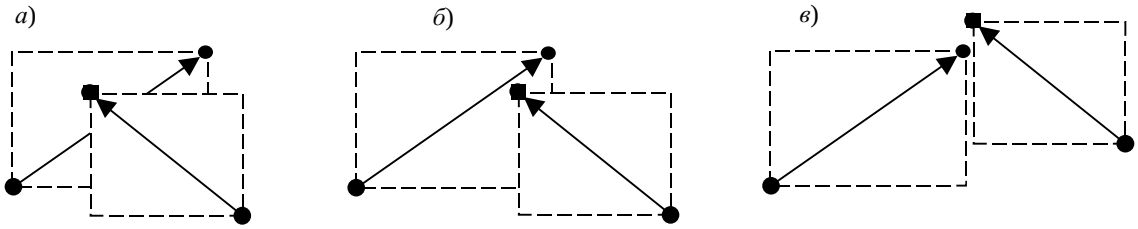


Рис. 8. Ограничивающие прямоугольники: *а*) из пересечения отрезков следует пересечение прямоугольников; *б*) из пересечения прямоугольников не следует пересечение отрезков; *в*) из факта непересечения прямоугольников следует факт непересечения отрезков

грубый тест выполняется быстро). Ограничивающий прямоугольник отрезка является наименьшим из прямоугольников, содержащих отрезок и имеющих стороны, параллельные осям координат. На рис. 8 показаны ограничивающие прямоугольники ориентированных отрезков $[p_a, p_b]$ и $[p_c, p_d]$ в различных ситуациях. Если ограничивающие прямоугольники не пересекаются, то и отрезки не пересекаются, если же установлен факт пересечения ограничивающих прямоугольников, то следует перейти ко второму этапу и проверить собственно пересечение отрезков (точный тест).

Перед проверкой пересечения прямоугольников полезно сначала определить ограничивающие прямоугольники отрезков, используя следующий вспомогательный алгоритм (листинг 1).

Два прямоугольника со сторонами, параллельными координатным осям, пересекаются тогда и только тогда, когда пересекаются и их проекции на ось x, и их проекции на ось y. Алгоритм проверки пересечения прямоугольников можно представить в виде алгоритма (см. листинг 2).

Условное выражение в строке 1 алгоритма иллюстрируется на рис. 9 (на примере пересечения x-проекций, то есть части конъюнкции $(x_b \geq x_c)$ **and** $(x_d \geq x_a)$.

Следует отметить, что в случаях, когда хотя бы один из исходных отрезков $[p_a, p_b]$ и $[p_c, p_d]$ вертикален или горизонтален, алгоритмы BOUNDINGRECTANGLE и RECTANGLESINTERSECT дают правильный результат.

Итак, если на первом этапе отсутствие пересечения двух отрезков не установлено тестом на пересечение ограничивающих прямоугольников, то необходимо применить точный тест. В основе теста лежит следующее утверждение: два отрезка пересекаются тогда и только тогда, когда каждый из отрезков пересекается с прямой, содержащей другой отрезок. Различные граничные случаи требуют аккуратного рассмотрения (см. далее). Отрезок $[p_a, p_b]$ пересекается с прямой, если концы отрезка лежат по разные стороны от прямой или на прямой (хотя бы один конец). Как мы знаем [3, стр. 9], определить, лежит ли точка $q = (x_q, y_q)$ по левую или по правую сторону от прямой, содержащей ориентированный отрезок

Листинг 1. Algorithm BOUNDINGRECTANGLE $(p_a, p_b) \rightarrow (p_1, p_2)$

Вход: $p_a = (x_a, y_a)$, $p_b = (x_b, y_b)$

Выход: ограничивающий прямоугольник для отрезка $[p_a, p_b]$, заданный парой вершин (p_1, p_2), где
$p_1 = (x_1, y_1)$ – левый нижний угол прямоугольника;
$p_2 = (x_2, y_2)$ – правый верхний угол прямоугольника;

1 | $x_1 \leftarrow min(x_a, x_b)$
2 | $y_1 \leftarrow min(y_a, y_b)$
3 | $x_2 \leftarrow max(x_a, x_b)$
4 | $y_2 \leftarrow max(y_a, y_b)$
5 | Вернуть в качестве результата $p_1 = (x_1, y_1)$ и $p_2 = (x_2, y_2)$

Листинг 2. Algorithm RECTANGLESINTERSECT $(p_a, p_b, p_c, p_d) \rightarrow$ Boolean
Вход: $p_a = (x_a, y_a)$, $p_b = (x_b, y_b)$, $p_c = (x_c, y_c)$, $p_d = (x_d, y_d)$
p_a – левый нижний угол первого прямоугольника;
p_b – правый верхний угол первого прямоугольника;
p_c – левый нижний угол второго прямоугольника;
p_d – правый верхний угол второго прямоугольника
Выход: True, если прямоугольники пересекаются, иначе False

```
1 | if (x_b ≥ x_c) and (x_d ≥ x_a) and (y_b ≥ y_c) and (y_d ≥ y_a)
2 | then Вернуть в качестве результата True
3 | else Вернуть в качестве результата False
```

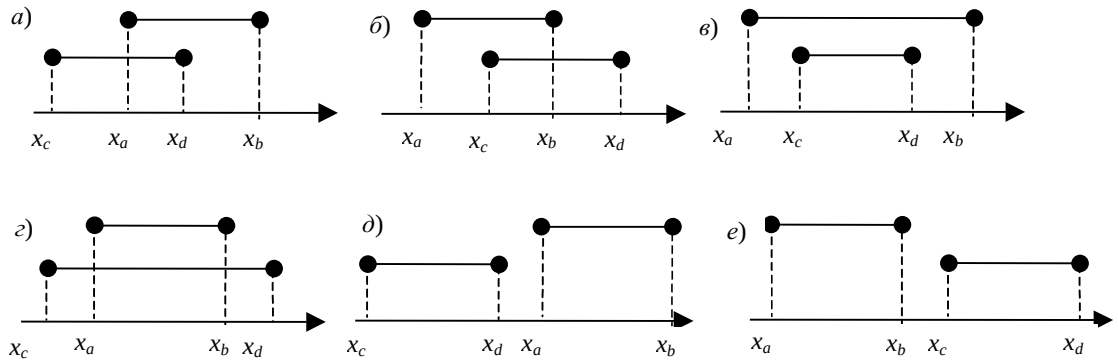


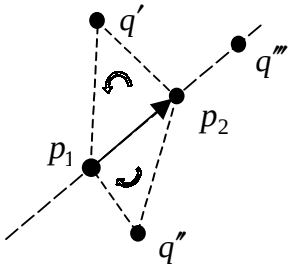
Рис. 9. В вариантах *а*)–*г*) справедливо $(x_b \geq x_c)$ **and** $(x_d \geq x_a)$, в варианте *д*) – $(x_b \geq x_c)$ **and** $(x_d < x_a)$, в варианте *е*) – $(x_b < x_c)$ **and** $(x_d \geq x_a)$

$[p_1, p_2]$, или на этой прямой, можно, анализируя знак ориентированной площади $S(p_1, p_2, q) = \frac{1}{2}(x_1 - x_2)(y_q - y_1) - (y_2 - y_1)(x_q - x_1)$ треугольника $\Delta\, p_1 p_2 q$ (или, что то же, векторного произведения $[\overrightarrow{p_1 p_2}, \overrightarrow{p_1 q}]$, см. [4] и приложение в [3]). На рис. 10 рассмотрены различные варианты расположения точки относительно отрезка.

Введем вспомогательный алгоритм вычисления ориентированной площади (см. листинг 3).

На рис. 11 представлены основные варианты расположения отрезков и их характеристики, определяющие факт пересечения отрезков.

На рис. 12 представлены некоторые из граничных случаев (остальные легко дополнить), когда хотя бы один из концов одного отрезка лежит на прямой, содержащей другой отрезок, и, следовательно, хотя бы одна из ориентированных площадей равна нулю. Как показывают, например, варианты *а*) и *б*), решение о пересечении отрезков нельзя принять только на основе анализа ориенти-



1. $S(p_1, p_2, q') > 0$, точка q' – слева от $[p_1, p_2]$, $\Delta\, p_1 p_2 q'$ ориентирован (обходится) против часовой стрелки;
2. $S(p_1, p_2, q'') < 0$, точка q'' – справа от $[p_1, p_2]$, $\Delta\, p_1 p_2 q''$ ориентирован (обходится) по часовой стрелке;
3. $S(p_1, p_2, q''') = 0$, точка q''' – на прямой, содержащей отрезок $[p_1, p_2]$, $\Delta\, p_1 p_2 q'''$ вырожден, точки p_1, p_2, q''' – коллинеарные.

Рис. 10. Различные варианты расположения точки q относительно отрезка $[p_1, p_2]$

Листинг 3. Algorithm AREA $(p_a, p_b, p_c) \to 2S(p_a, p_b, p_c)$

Вход: $p_a = (x_a, y_a)$, $p_b = (x_b, y_b)$, $p_c = (x_c, y_c)$

Выход: удвоенная ориентированная площадь треугольника $\Delta\, p_a p_b p_c$

1 Вернуть в качестве результата $(x_a - x_b)(y_c - y_a) - (y_b - y_a)(x_c - x_a)$
{если результат > 0, то точка p_c лежит *слева* от ориентированного отрезка $[p_a, p_b]$,
если результат < 0, то точка p_c лежит *справа* от ориентированного отрезка $[p_a, p_b]$,
если результат = 0, то точка p_c лежит *на* прямой, содержащей отрезок $[p_a, p_b]$,
то есть точки p_a, p_b, p_c коллинеарные}

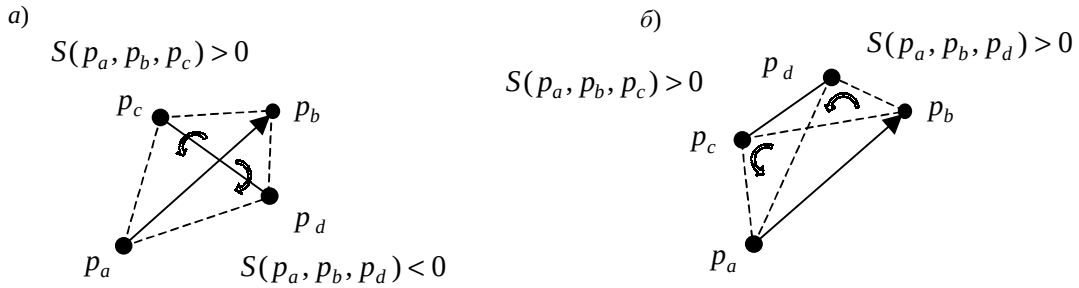


Рис. 11. Анализ пересечения отрезков $[p_a, p_b]$ и $[p_c, p_d]$:
а) отрезки пересекаются, и $S(p_a, p_b, p_c)$ и $S(p_a, p_b, p_d)$ имеют разные знаки;
б) отрезки не пересекаются, и $S(p_a, p_b, p_c)$ и $S(p_a, p_b, p_d)$ имеют одинаковые знаки

рованных площадей. Отметим, что в нашем подходе вариант *г*) не может появиться на втором этапе, так как ограничивающие прямоугольники здесь не пересекаются.

Итак, граничные случаи требуют дополнительного анализа, который может быть проведен с использованием следующего алгоритма (см. листинг 4).

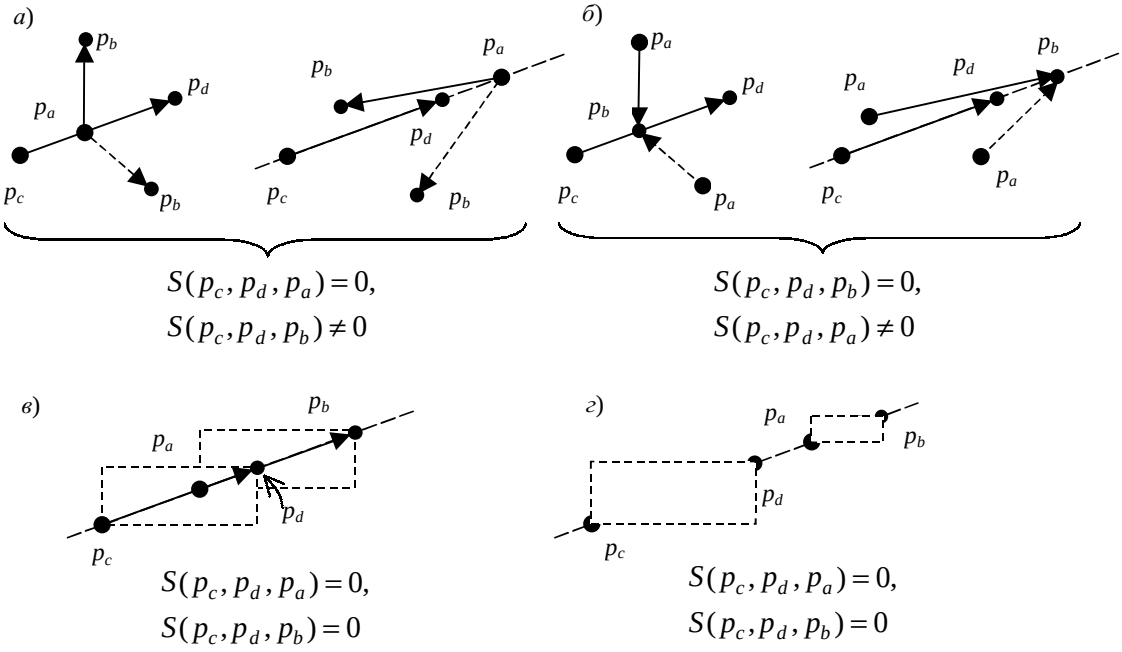


Рис. 12. Граничные случаи, когда ориентированная площадь равна нулю

Листинг 4. Algorithm BETWEEN $(p_a, p_b, p_c) \rightarrow$ Boolean

Вход: $p_a = (x_a, y_a)$, $p_b = (x_b, y_b)$, $p_c = (x_c, y_c)$
Предусловие: точка p_c лежит *на* прямой, содержащей отрезок $[p_a, p_b]$
Выход: True, если точка p_c является точкой ориентированного отрезка $[p_a, p_b]$, иначе False

```
1 | if min(x_a, x_b) ≤ x_c ≤ max(x_a, x_b) and min(y_a, y_b) ≤ y_c ≤ max(x_a, x_b)
2 | then Вернуть в качестве результата True
3 | else Вернуть в качестве результата False
```

Подытожим наши соображения в алгоритме SEGMENTSINTERSECT (листинг 5).

В тех случаях, когда кроме факта пересечения пары отрезков требуется определить точку их пересечения, алгоритм SEGMENTSINTERSECT следует дополнить. Рассмотрим основные необходимые для этого соотношения, оставляя читателю модификацию алгоритма SegmentsIntersect в качестве упражнения.

Для представления отрезков $[p_a, p_b]$ и $[p_c, p_d]$, заданных концевыми точками, удобно использовать параметрические уравнения [4]. Например, все точки отрезка $[p_a, p_b]$ описываются следующим уравнением:

$$p_{ab}(t) = p_a + (p_b - p_a)t,$$

где вещественный параметр $t \in [0,1]$. Очевидно, $p_{ab}(0) = p_a$ и $p_{ab}(1) = p_b$, а внут-

Листинг 5. Algorithm SEGMENTSINTERSECT $(p_a, p_b, p_c, p_d) \rightarrow$ Boolean

Вход: $p_a = (x_a, y_a)$, $p_b = (x_b, y_b)$ – концы ориентированного отрезка $[p_a, p_b]$, $p_c = (x_c, y_c)$, $p_b = (x_b, y_b)$ – концы ориентированного отрезка $[p_c, p_d]$,
Выход: True, если отрезки пересекаются, иначе False

```
1  | (p1, p2) ← BoundingRectangle (p_a, p_b)
2  | (p3, p4) ← BoundingRectangle (p_c, p_d)
3  | if not RectanglesIntersect (p1, p2, p3, p4)
4  | then Вернуть в качестве результата False
   |                {ограничивающие прямоугольники не пересекаются}
5  | else {ограничивающие прямоугольники пересекаются}
6  |   s1 ← Area (p_c, p_d, p_a)
7  |   s2 ← Area (p_c, p_d, p_b)
8  |   s3 ← Area (p_a, p_b, p_c)
9  |   s4 ← Area (p_a, p_b, p_d)
10 |   if ((s1 > 0 and s2 < 0) or (s1 < 0 and s2 > 0)) and
   |          ((s3 > 0 and s4 < 0) or (s3 < 0 and s4 > 0))
11 |   then Вернуть в качестве результата True
12 |   else if (s1 = 0) and Between (p_c, p_d, p_a)
13 |        then Вернуть в качестве результата True
14 |   else if (s2 = 0) and Between (p_c, p_d, p_b)
15 |        then Вернуть в качестве результата True
16 |   else if (s3 = 0) and Between (p_a, p_b, p_c)
17 |        then Вернуть в качестве результата True
18 |   else if (s4 = 0) and Between (p_a, p_b, p_d)
19 |        then Вернуть в качестве результата True
20 |   else Вернуть в качестве результата False
```

ренним точкам $p_{ab}(t)$ отрезка $[p_a, p_b]$ соответствуют значения параметра $[p_a, p_b]$. Аналогичное уравнение запишем для отрезка $[p_c, p_d]$:

$$p_{cd}(t) = p_c + (p_d - p_c)s,$$

где параметр $s \in [0,1]$.

Поскольку алгоритм SEGMENTSINTERSECT при определении факта пересечения пары отрезков разбирает все необходимые ситуации, то будем интересоваться точкой пересечения отрезков только тогда, когда установлено, что отрезки пересекаются и при этом не лежат на одной прямой.

Для вычисления точки пересечения запишем уравнение $p_{ab}(t) = p_{cd}(s)$:

$$p_a + (p_b - p_a)t = p_c + (p_d - p_c)s,$$

которое определяет значения параметров t и s, соответствующие точке пересечения q. Покоординатная запись этого уравнения

$$x_a + (x_b - x_a)t = x_c + (x_d - x_c)s,$$
$$x_a + (x_b - x_a)t = x_c + (x_d - x_c)s,$$

является системой двух линейных уравнений для определения неизвестных t и s. Нетрудно убедиться, что решением этой системы будут

$$t = [(x_a - x_c)(y_d - y_c) + (x_d - x_c)(y_c - y_a)] / D,$$
$$s = [(x_b - x_a)(y_c - y_a) + (x_a - x_c)(y_b - y_a)] / D,$$

где D есть определитель системы

$$D = (x_a - x_b)(y_d - y_c) + (x_d - x_c)(y_b - y_a).$$

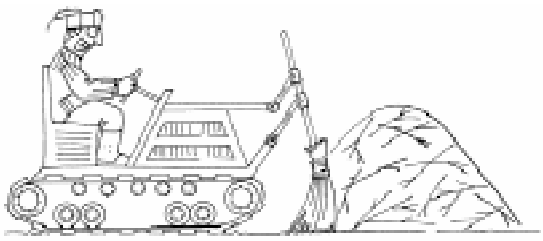
В нашем случае $D \neq 0$, поскольку мы заранее знаем, что отрезки пересекаются (то есть не параллельны) и заведомо не лежат на одной прямой.

Подставив вычисленное значение t или s в уравнение $p_{ab}(t) = p_a + (p_b - p_a)t$ или $p_{cd}(t) = p_c + (p_d - p_c)s$ соответственно, получим точку пересечения, например,

$$q = p_a + (p_b - p_a)t.$$

Разумеется, в вырожденных случаях, рассмотренных в алгоритме SEGMENTSINTERSECT, а также в случае вертикальных или горизонтальных отрезков, система уравнений для определения t и s, а, следовательно, и формулы вычисления значений t или s могут быть при необходимости упрощены.

5. АЛГОРИТМ НАХОЖДЕНИЯ ПЕРЕСЕЧЕНИЯ ОТРЕЗКОВ



Более эффективным, чем «лобовой» алгоритм, имеющий сложность $O(n^2)$, является алгоритм, основанный на применении *метода заметания плоскости*. Метод подсказан геометрической природой задач. В нашем случае речь идет о *плоском заметании*, поскольку рассматриваются задачи на плоскости.

Опишем идею метода плоского заметания применительно к задаче нахождения всех пересечений отрезков, а затем перейдем к описанию алгоритма. Возьмем *вертикальную прямую* L, которая разбивает плоскость на левую и правую полуплоскости (см. рис. 13). Пусть каждая из этих полуплоскостей содержит концы заданных отрезков. Решение нашей задачи может быть

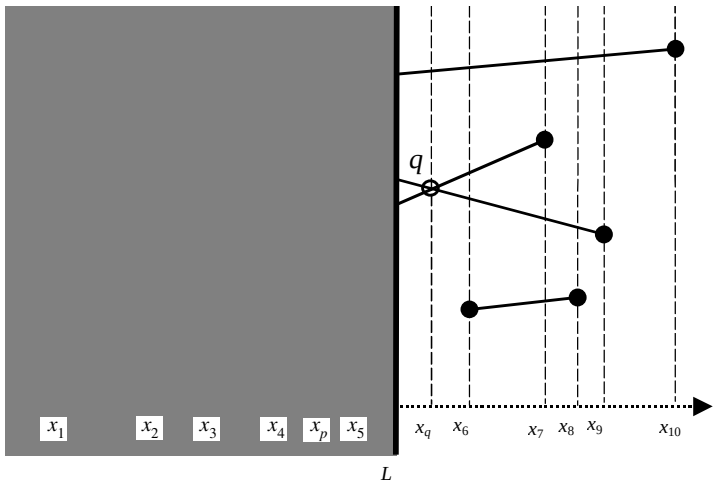


Рис. 13. Заметание плоскости вертикальной прямой L

получено объединением решений для каждой из двух полуплоскостей. Если предположить, что у нас уже есть множество точек пересечения слева от L, то на это множество не будут влиять отрезки, лежащие справа от L. Заметим, что искомое пересечение может произойти только между такими двумя отрезками, чьи пересечения с некоторой вертикалью смежны.

Теперь представим себе, что вертикальная прямая движется слева направо. По ходу движения будут пройдены все вертикальные сечения и, следовательно, могут быть выявлены все пары смежных в указанном смысле отрезков, а, значит, все пересечения будут обнаружены. Можно избежать построения всего бесконечного (континуального) множества секущих, если заметить (см. рис. 13), что плоскость разбивается на вертикальные полосы, ограниченные или концами отрезков, или точками их пересечения. При этом в пределах полосы вертикальный порядок точек пересечения отрезков с прямой L постоянен. Таким образом, вместо непрерывного движения заметающей прямой L мы можем рассмотреть *скачки* этой прямой по границам вертикальных полос. Каждый такой скачок связан с необходимостью обновления порядка точек пересечения вертикали и поиском пересечений среди соседних (смежных) в этом порядке отрезков.

Итак, подчеркнем, что непрерывное заметание плоскости является метафорой, в то время как алгоритм, основанный на этой идее, будет обрабатывать конечное множество «скачков». Точки, по которым заметающая прямая «скачет», называют *точками событий*. В этих точках пересечение заметающей прямой с входными отрезками содержит всю полезную информацию, которая позволит продолжить поиск. В связи с этим в методе заметания используют две основные структуры данных:

1. Список точек событий – последовательность абсцисс, упорядоченных слева направо и определяемых позициями «остановок» заметающей прямой (этот список может динамически обновляться в процессе заметания).

2. Статус заметающей прямой – подходящее описание пересечения этой прямой с набором заданных отрезков. Статус заметающей прямой должен обновляться в точках событий, и представляющая статус структура данных должна выбираться, исходя из требуемых операций анализа и обновления статуса.

Можно интерпретировать движение заметающей прямой по плоскости в терминах *пространства-времени* следующим образом. Представим ось абсцисс как ось времени, а заметающую прямую как мгновенный «снимок» ситуации. Тогда отрезки на плоскости становятся движущимися по вертикальной прямой («пространственной» оси) точками, которые в каждом временном сечении могут быть упорядочены (см. рис. 14). Иногда пространственно-временное представление способно «подтолкнуть» интуицию в процессе разработки геометрических алгоритмов.

Алгоритм использует два упрощающих предположения: среди рассматриваемых

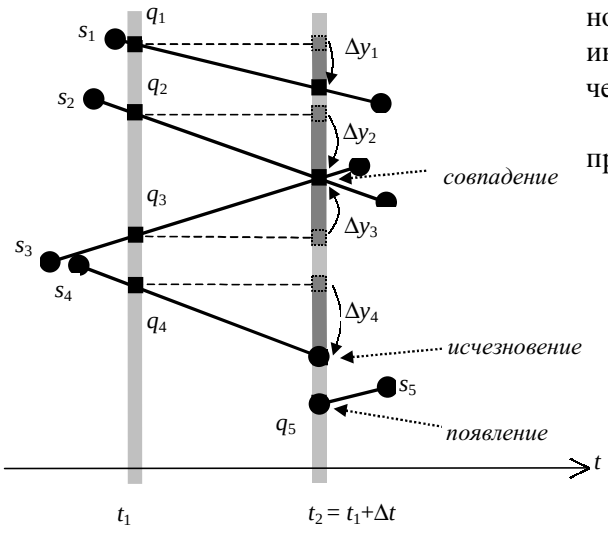


Рис. 14. Перемещение по вертикальной прямой точек q_i ($i \in 1...4$) пересечения отрезков s_i с этой прямой за время Δt на величину Δy_i. В момент времени $t_2 = t_1 + \Delta t$ точка q_5 появляется на заметающей прямой, точка q_4 должна исчезнуть, а точки q_2 и q_3 сливаются в месте пересечения отрезков s_2 и s_3, чтобы далее продолжить свое движение, пройдя сквозь друг друга

отрезков нет вертикальных, и никакие три отрезка не имеют общих точек. При нарушении предположений потребовалась бы модификация алгоритма, не меняющая его сути, но включающая дополнительную корректную обработку указанных вырожденных ситуаций.

Определим отношение порядка на отрезках следующим образом. Рассмотрим пару непересекающихся отрезков s_1 и s_2. Будем считать, что s_1 и s_2 сравнимы в абсциссе x, если существует такая вертикаль, проходящая через x, которая пересекает оба отрезка. Введем отношение *выше* в x следующим образом: s_1 выше s_2 в x ($s_1 >_x s_2$), если s_1 и s_2 сравнимы в x, а точка пересечения отрезка s_1 с вертикалью x лежит выше точки пересечения отрезка s_2 с ней же. На рис. 15 показаны следующие отношения между отрезками s_1, s_2, s_3 и s_4:

$$s_2 >_u s_4,\ s_1 >_v s_2,\ s_2 >_v s_4,\ s_1 >_v s_4.$$

Заметим, что отношение $>_x$ задает полное упорядочение, которое изменяется по мере того, как вертикаль скользит слева направо. Отрезки входят в это упорядочение и покидают его, но оно всегда остается полным. Упорядочение должно быть представлено структурой статуса заметающей прямой и может изменяться только в трех случаях:

– встретился левый конец отрезка s (добавление s к структуре данных);

– встретился правый конец отрезка s (удаление s из структуры данных, поскольку он больше не сравним с оставшимися отрезками);

– обнаружена точка пересечения отрезков s_1 и s_2 (s_1 и s_2 меняются местами в структуре данных).

Заметим, что необходимое условие пересечения двух отрезков s_1 и s_2 состоит в том, что существует такая абсцисса x, в которой s_1 и s_2 смежны в упорядочении $>_x$. Отсюда следует, что последовательность пересечений множества отрезков с вертикалью в x (то есть отношение $>_x$) содержит всю необходимую информацию для поиска пересечений отрезков.

Статус заметающей прямой является описанием отношения $>_x$; значит, он представляет собой последовательность элементов (отрезков). Так как отношение $>_x$ изменяется на конечном множестве абсцисс при плоском заметании, то станет очевидно, что в структуре данных *SL* (Sweep Line), реализующей статус заметающей прямой, должны быть предусмотрены следующие операции:

- ВСТАВИТЬ(s, SL) – вставить отрезок s в полное упорядочение, представленное SL;
- УДАЛИТЬ(s, SL) – удалить отрезок s из SL;
- НАД(s, SL) – найти имя отрезка, расположенного непосредственно над s в SL;
- ПОД(s, SL) – найти имя отрезка, расположенного непосредственно под s в SL.

Подобная структура данных известна как словарь [1, 5], а все вышеописанные операции можно реализовать за время, логарифмически связанное с его размером, например, с помощью подходящего типа бинарных деревьев поиска. На практике использование прошитого словаря (с прямым употреблением указателей, если адрес s известен) позволяет реализовать функции НАД(s) и ПОД(s) за константное время.

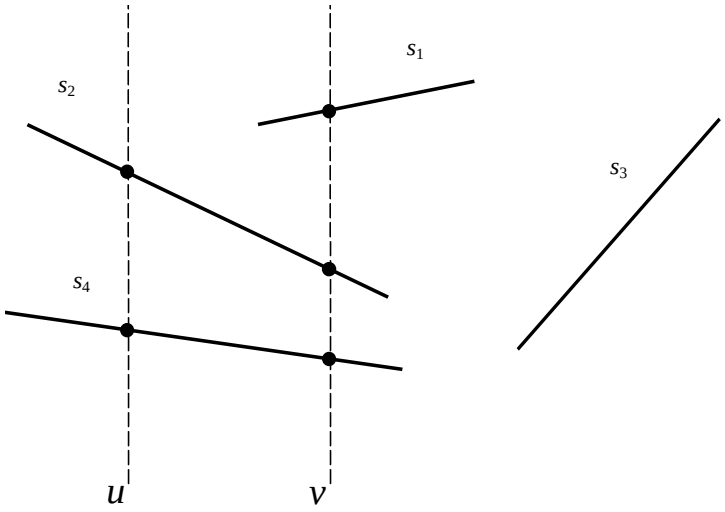


Рис. 15. Отношение порядка между отрезками

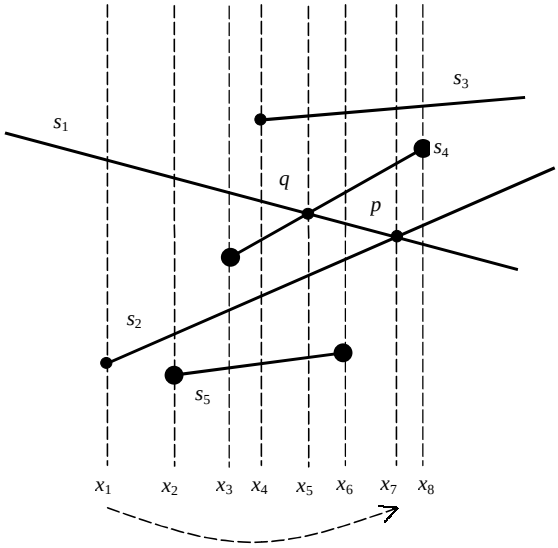


Рис. 16. Порождение нового события x_7 при обработке события x_1

Для регистрации всех пересечений в списке точек событий нужно уметь поддерживать отношение $>_x$ в течение всего процесса заметания плоскости. Как указано ранее, упорядочение $>_x$ изменяется только в некоторых абсциссах, которые являются концами отрезков или точками их пересечения. В то время как все концы отрезков заданы заранее, точка пересечения, найденная плоским заметанием, динамически порождает событие, которое, вообще говоря, должно запоминаться и обрабатываться алгоритмом в нужный момент. Заметим, что в действительности, возможно, придется обработать несколько других событий за время, прошедшее с момента обнаружения какой-нибудь точки пересечения до момента ее обработки.

На рис. 16 дан пример подобной ситуации: точка пересечения p обнаружена в момент x_1 (когда отрезки s_1 и s_2 стали смежными); однако необходимо обработать абсциссы x_2, x_3, x_4 и x_6 перед обработкой соответствующего p события x_7. Более того, при обработке события x_3 возникнет точка пересечения q, и связанное с q событие x_3 должно обрабатываться раньше событий x_6 и x_7. Отметим также, что после обработки события x_3 отрезки s_1 и s_2 перестанут быть смежными, и вновь станут таковыми после обработки события x_5. Тогда пересечение отрезков s_1 и s_2 в точке p будет обнаружено повторно, но его включение в список точек событий уже не потребуется.

Итак, для отчета обо всех пересечениях структура данных Q, создаваемая для работы со списком точек событий, должна поддерживать следующие операции (в Q хранится полное упорядочение точек событий):

- MIN(Q) – определить наименьший элемент в Q и удалить его;
- ВСТАВИТЬ(x, Q) – вставить абсциссу x в полное упорядочение, хранимое в Q;
- ПРИНАДЛЕЖИТ(x, Q) – узнать принадлежит ли абсцисса x списку точек событий Q.

Указанная структура данных представляет собой *очередь с приоритетом*, и, как хорошо известно [5], при соответствующей реализации она может поддерживать все выше описанные операции за время, логарифмически связанное с ее размером.

Теперь можно описать алгоритм: при заметании плоскости вертикалью в каждой точке события структура данных *SL* корректируется и все пары отрезков, которые становятся смежными при этой корректировке, проверяются на пересечение. Если какое-нибудь пересечение обнаружено впервые, о нем дается отчет, и его абсцисса вставляется в список точек событий Q. Более формально приведенные рассуждения выражены в алгоритме ALL-SEGMENT-INTERSECT (листинг 6). Отметим, что в алгоритме используется локальный объект (рабочий параметр) – очередь A для временного (до занесения в очередь событий) хранения обнаруженных пересечений. При записи алгоритма на псевдокоде мы не будем детализировать тип операции пересечения пары отрезков, считая, что читатель при необходимости сделает это самостоятельно на основе изложенного в разделе 4 материала.

Корректность алгоритма ALL-SEGMENT-INTERSECT основана на том, что в процессе его работы не будет пропущено ни одного из пересечений, поскольку могут пересечься только смежные отрезки, причем все смежные пары корректно проверяются, по крайней мере, однажды. Более того, каждое пересечение регистрируется ровно один

Листинг 6. Algorithm ALL-SEGMENT-INTERSECT $(S) \to W$

Вход: множество S отрезков $\{s_1, s_2, ..., s_n\}$, заданных своими концами $s = [p_a, p_b]$
Выход: W – последовательность пар пересекающихся отрезков (s, s')

```
1   Упорядочить 2n концов отрезков лексикографически по x и y и поместить их
    в приоритетную очередь Q
2   A ← ∅
3   while Q ≠ ∅ do
4       p ← MIN(Q)
5       if (p – левый конец) then
6           s ← отрезок, концом которого служит p
7           ВСТАВИТЬ(s, SL)
8           s1 ← НАД(s, SL)
9           s2 ← ПОД(s, SL)
10          if (s1 пересекает s) then A ← A ∪ (s1, s)
11          if (s2 пересекает s) then A ← A ∪ (s2, s)
12      else if (p – правый конец) then
13          s ← отрезок, концом которого служит p
14          s1 ← НАД(s, SL)
15          s2 ← ПОД(s, SL)
16          if (s1 пересекает s2 справа от p) then A ← A ∪ (s1, s2)
17          УДАЛИТЬ(s, SL)
18      else {p – точка пересечения}
19          (s1, s2) ← отрезки, пересекающиеся в p {причем s1 = НАД(s2) слева от p}
20          s3 ← НАД(s1, SL)
21          s4 ← ПОД(s2, SL)
22          if (s3 пересекает s2 справа от p) then A ← A ∪ (s3, s2)
23          if (s1 пересекает s4 справа от p) then A ← A ∪ (s1, s4)
24          поменять местами s1 и s2 в SL
25      end-if
        end-if
        {обнаруженные пересечения следует занести в очередь событий Q}
26      while A ≠ ∅ do
27          (s, s') ← A
28          x ← общая абсцисса s и s'
29          if not ПРИНАДЛЕЖИТ(x, Q) then
30                  Добавить (s, s') в выходную последовательность W
31                  ВСТАВИТЬ(x, Q)
32          end-if
33      end-do
34  end-do
```

раз, поскольку, когда его абсцисса вставляется в Q, проверка в строке 29 предупреждает нежелательные повторы.

Оценим сложность алгоритма. Заметим, что операция строки 1 (начальная сортировка) выполняется за время $O(n \log n)$. Блоки 6–11, 13–17 и 19–24 выполняются каждый за время $O(\log n)$, поскольку каждая операция на структуре данных SL укладывается в такую временную оценку в худшем случае, а проверка попарного пересечения требует константного времени. Для каждого события, то есть для каждого исполнения главного цикла **while** (строка 3), эти три блока являются взаимоисключающими. Если через k обозначить число пересечений, встреченных алгоритмом, то главный цикл **while** выполняется ровно

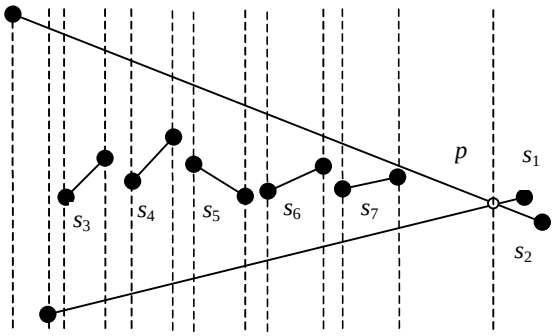


Рис. 17. Точка пересечения p отрезков s_1 и s_2 будет повторно обнаруживаться при обработке событий, связанных с правыми концами отрезков s_3, s_4, s_5, s_6, s_7, то есть всякий раз, когда отрезки s_1 и s_2 вновь становятся смежными в статусе заметающей прямой

$2n + k$ раз. На каждом шаге цикла обнаруживается и заносится в рабочую очередь A не более двух пересечений. Заметим, что какое-нибудь одно пересечение можно повторно обнаруживать (на разных шагах цикла) много раз (см. рис. 17).

Однако, если соотнести каждую пересекающуюся пару с тем выполнением главного цикла **while**, на котором она обнаружена, то станет ясно, что общее число обнаруженных пересечений равно $O(n + k)$. Поэтому строка 29 (проверка перед переносом пересечения из рабочей очереди A в очередь событий Q) выполняется $O(n + k)$ раз, а каждое выполнение требует $O(\log(n + k)) = O(\log n)$ времени, поскольку $k \le \frac{n(n-1)}{2} = O(n^2)$. Следовательно, общая оценка времени главного цикла **while**, равна $O((n + k)\log n)$, что, очевидно, превосходит оценку шага начальной сортировки. Отсюда имеем, что на множестве из n отрезков можно обнаружить k пересечений за время $O((n + k)\log n)$. В итоге можно сформулировать следующее утверждение.

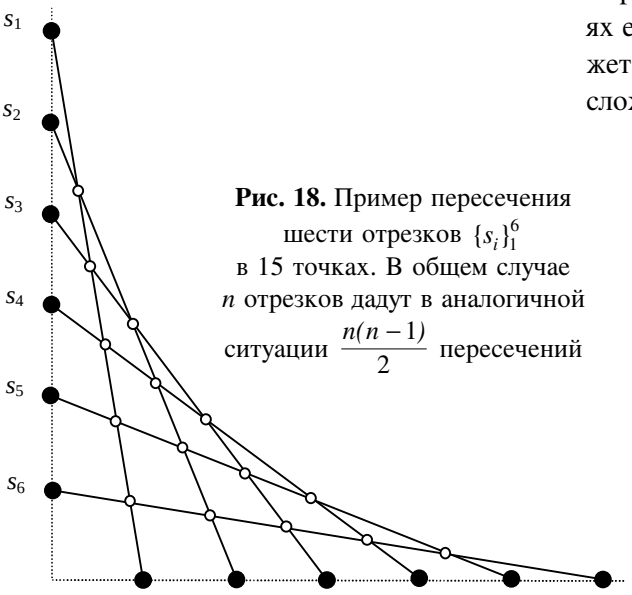


Рис. 18. Пример пересечения шести отрезков $\{s_i\}_1^6$ в 15 точках. В общем случае n отрезков дадут в аналогичной ситуации $\frac{n(n-1)}{2}$ пересечений

Утверждение 1. На множестве из n отрезков можно решить Задачу 2 (ВПО) и найти все k пересечений за время $O((n + k)\log n)$.

Алгоритм ALL-SEGMENT-INTERSECT значительно лучше, чем «лобовой» метод, если число пересечений достаточно мало, например, $k = O(n)$. Однако при $k = \Omega(n^2)$ (максимальное значение k есть $\frac{n(n-1)}{2}$, см. рис. 18) время работы алгоритма составит $O(n^2 \log n)$, что хуже, чем у «лобового» алгоритма. Дело в том, что алгоритм ALL-SEGMENT-INTERSECT не достигает нижней границы. В действительности можно надеяться на получение алгоритма сложности $O(k + n \log n)$, поскольку $\Omega(n \log n)$ есть нижняя оценка, полученная по задаче ППО, а время формирования отчета о пересечениях есть $O(k)$. Не вдаваясь в подробности сюжета о разработке оптимального алгоритма сложности $O(k + n \log n)$ [1], отметим, что такой алгоритм существует (см., например, [6]), но его описание выходит за рамки нашего изложения.

В том случае, когда нет необходимости находить все пересечения, а речь идет лишь о проверке наличия пересечений (задача 1 – ППО), применение схемы с заметанием значительно упрощается. Главное отличие связано со структурой данных для списка точек событий. Теперь не требуется обработка событий, связанных с пересечениями, так как ал-

горитм закончится при первом же обнаружении пересечения. Поэтому список событий представляется упорядоченным массивом, содержащим $2n$ исходных концевых точек. Соответствующую упрощенную версию алгоритма, работающего за время $O(n \log n)$, можно найти в [1] или [5]. Отсюда следует следующее утверждение.

Утверждение 2. Факт пересечения какой-либо пары из n отрезков на плоскости можно установить за оптимальное время $\Theta(n \log n)$.

Из утверждения 2 дополнительно вытекает

Следствие. За время $O(n \log n)$ в худшем случае решаются следующие задачи:

Задача 3. ПРОВЕРКА ПЕРЕСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ (ППМ).

Задача 4. ТЕСТ ПРОСТОТЫ МНОГОУГОЛЬНИКА (ТПМ).

6. ЗАКЛЮЧЕНИЕ

Примененный в задаче пересечения отрезков метод заметания является эффективным средством решения многих геометрических задач. В качестве примера перечислим следующие задачи:

– геометрический поиск: метод полос, метод цепей (предобработка) [1];

– разбиение простого многоугольника на монотонные многоугольники [1];

– пересечение, объединение и другие характеристики множества прямоугольников [1];

– построение диаграммы Вороного и триангуляции Делоне [7].

Литература

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение. М.: Мир, 1989. 478 с.

2. *Ивановский С.А., Преображенский А.С., Симончик С.К.* Алгоритмы вычислительной геометрии. Выпуклые оболочки: связь с задачей сортировки и оптимальные алгоритмы // Компьютерные инструменты в образовании, 2007, №2. С. 6-18.

3. *Ивановский С.А., Преображенский А.С., Симончик С.К.* Алгоритмы вычислительной геометрии. Выпуклые оболочки: простые алгоритмы // Компьютерные инструменты в образовании, 2007, №1. С. 4-19.

4. *Ильин В. А., Позняк Э. Г.* Аналитическая геометрия. М.: ФИЗМАТЛИТ, 2002. 240 с.

5. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦМНО, 2000. 960 с.

6. *Balaban I.J.* An Optimal Algorithm for Finding Segment Intersections, Proc. 11-th Ann. ACM Sympos. Comp. Geom., 211-219, 1995.

7. *de Berg M., van Kreveld M., Overmars M., Schwarzkopf O.* Computational Geometry: Algorithms and Applications. (Second edition). Springer-Verlag, Heidelberg, 2000

Ивановский Сергей Алексеевич,
кандидат технических наук,
доцент кафедры Математического обеспечения и применения ЭВМ СПбГЭТУ «ЛЭТИ»,

Симончик Сергей Константинович,
аспирант СПбГЭТУ «ЛЭТИ»
магистр прикладной математики и информатики.