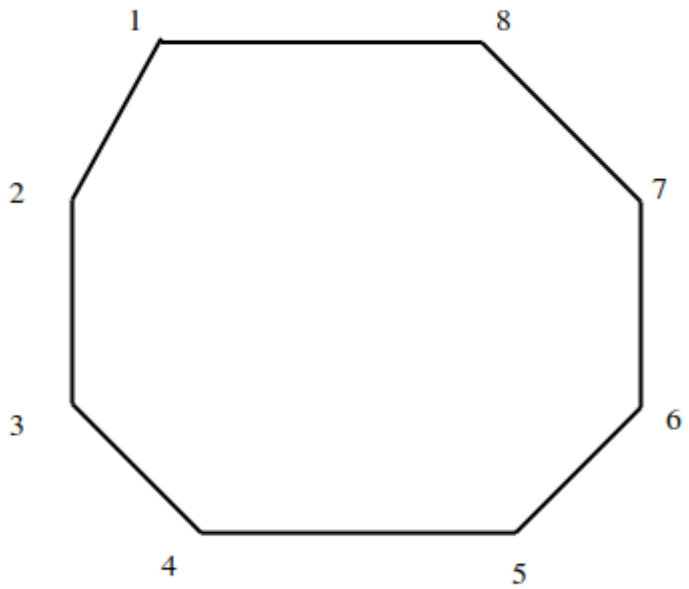
Octagon

Здесь описаны нетривиальные алгоритмы, которые я применил при реализации класса ***Octagon***

Для начала определим нормальную систему неравенств, задающую многоугольник:
Система, задающая многоугольник, называется нормальной, если она задает непустое множество точек и при уменьшении хотя бы одного из лимитов (лимиты - целые числа) мы получим другое множество точек (То есть уменьшать лимиты мы не можем).
Иногда при доказательствах я буду пользоваться прямыми, которые задаются неравенствами при замене нестрого знака на знак равенства, так как это удобно (неравенство задает одну из полуплоскостей, на которые прямая делит плоскость).

Метод Point vertex(int) const:



На самом деле от нас требуется найти такие точки по их номеру.
Заметим, что любая такая точка является точкой пересечения прямой, параллельной одной из оси координат, и диагональной прямой. Из первой прямой мы явно получаем одну из координат этой точки, из второй прямой получаем вторую координату, так как эта прямая имеет вид

Ax + By = C

Метод int isPointInside(const Point&) const:

*Точка принадлежит многоугольнику **<=>** ее координаты удовлетворяют всем неравенствам системы.*

Точка находится на границе многоугольника **<=>** *точка принадлежит многоугольнику и хотя бы в одном из неравенств системы мы получаем равенство левой и правой части при подстановке координат точки.*

Точка не принадлежит многоугольнику **<=>** *ее координаты не удовлетворяют хотя бы одному неравенству в системе.*

Метод void coverPoint(const Point&):

В этом методе мы модифицируем систему неравенств таким образом, чтобы все точки исходного многоугольника и новая точка удовлетворяли ей. Для этого достаточно увеличить каждый лимит таким образом, чтобы новая точка удовлетворяла соответствующему неравенству. Если исходная система была нормальной, то новая тоже будет нормальной: каждая прямая соприкасается с точкой множества, если мы ее не двигали (двигать прямую = изменять лимит, соответствующий этой прямой), так как исходное множество не уменьшается, а если двигали, то соответствующая прямая соприкасается с новой точкой.

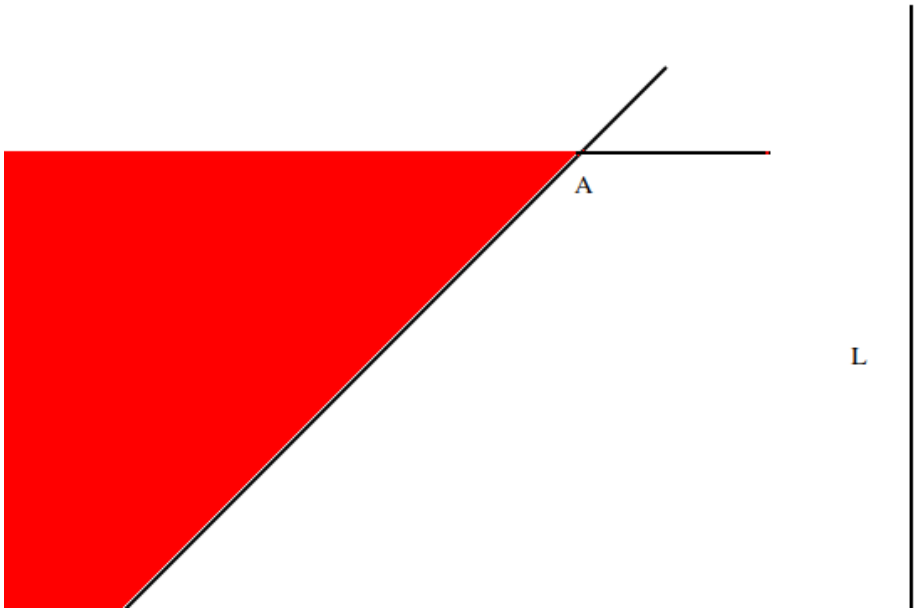
Метод bool isEqual(const Octagon&) const:

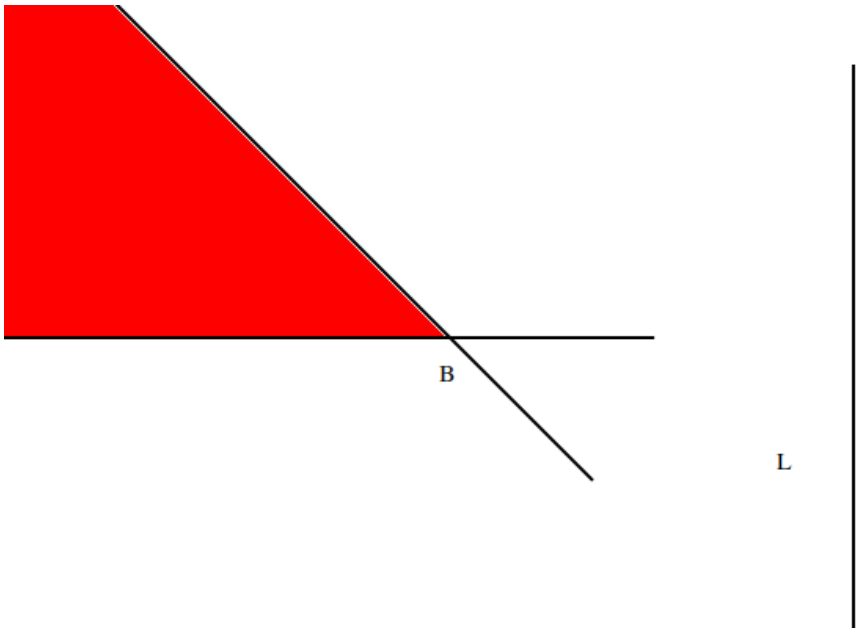
Очевидно, существует биекция между непустым многоугольником и нормальной системой неравенств. Докажем это:

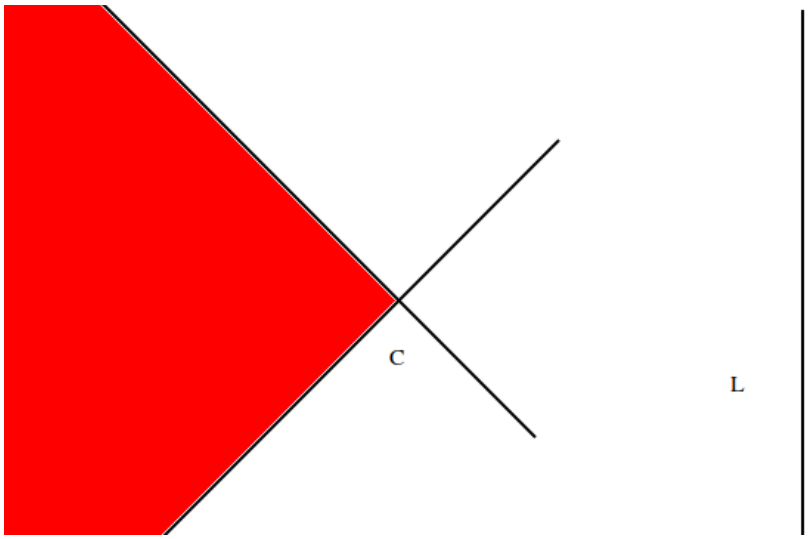
Предположим, что это не так, и существует две разные нормальные системы, задающие многоугольник. Тогда хотя бы один лимит у них отличается. В таком случае возьмем меньший из двух лимитов: мы получили противоречие с тем, что каждая система нормальная, так как в другой системе мы можем уменьшить лимит до другого.

Для сравнения двух многоугольников достаточно сравнить их нормальные системы неравенств: если они совпадают, то многоугольники равны.

Для сравнения многоугольников на равенство нам нужно научиться приводить их системы к нормальную виду. Но сначала нужно понять, когда мы может получить ненормальную систему неравенств. В конструкторе от множества точек мы получаем нормальную систему неравенств. Проблема возникает в конструкторе от набора лимитов и пересечении многоугольников: здесь мы можем получить произвольную систему неравенств. Реализуем метод void normalize(), который нормирует систему неравенств. Чтобы понять, как нам ее нормировать, рассмотрим следующие случаи (и рисунки):


A
L


B
L


C
L

Рассмотрим такой случай: прямая L не соприкасается со множеством точек, удовлетворяющих системе неравенств. Тогда нам нужно рассмотреть 3 точки: A, B и C. Нам достаточно подвинуть прямую в самую левую из этих трёх точек (“левость” этих точек определяется значением, полученным при подстановке координат точек в уравнение прямой L: берем точки с наименьшим значением). Каждая из этих точек определяется пересечением некоторых прямых, которые мы получаем в результаты несложных рассуждений.

В случае, когда прямая соприкасается со множеством точек, она находится левее любой из этих трех точек (либо проходит через них), тогда ее двигать не надо.

Проделаем такие рассуждения для каждой из прямых, этим мы нормируем всю систему неравенств. Теперь нужно это доказать.

Доказательство:

Каждая прямая встанет на нужное ей место, потому что самая “левая” точка Q, в которую мы сдвинем прямую, всегда является самой правой точкой множества в этом направлении. Предположим, что это не так, тогда одна из оставшихся двух точек (точка M) является самой правой точкой множества в этом направлении, тут мы получаем противоречие, так как M не будет принадлежать множеству вообще.

В этом доказательстве мы не рассматриваем пустые многоугольники, так как нормализация пустого многоугольника - это UB.

UPD*:*
Здесь возникает проблема: диагональные прямые могут пересекаться в точке с нецелыми координатами. Реализация нормализации системы была упрощена, для поиска точки пересечения прямых я складываю их уравнения. Кроме этого решена проблема с нецелыми точками пересечения диагональных прямых (например, x + y = 1 и y – x = 0 пересекаются в нецелой точке).

Метод void inflate(int inflateParam):

*В этом методе достаточно сдвинуть все прямые, параллельные осям координат, в соответствующем направлении на inflateParam, а диагональные прямые на inflateParam *
sqrt(2), *округленное вверх, если inflateParam > 0, и вниз в обратном случае.*

При inflateParam = 0 ничего делать не надо.

UPD: здесь ошибку так и не понял 😕

*UPD2: Ошибка была в том, что я неправильно осознал раздутие многоугольника и диагональные стороны двигал строго на вектор (enum(dir).x * d / sqrt(2), enum(dir).y * d / sqrt(2)) (знаки при координатах вектора зависят от направления, в которое я двигаю прямую). Так прямую я должен двигать только в случае, когда она проходит через сторону октагона. Иначе она проходит через точку, либо вообще не касается октагона. В таком случае после раздутия она также должна быть вне октагона, поэтому я двигаю ее дальше (на вектор (enum(dir).x * d, enum(dir).y * d)). Аналогично со сдутием октагона, только там я*

не трогаю прямые, которые находятся вне многоугольника: они встанут на свое место посредством вызова метода normalize().

Аналогично я двигаю горизонтальные прямые.

Методы, связанные с пересечением многоугольников:

```
bool hasIntersection(const Octagon& first, const Octagon& second)
```

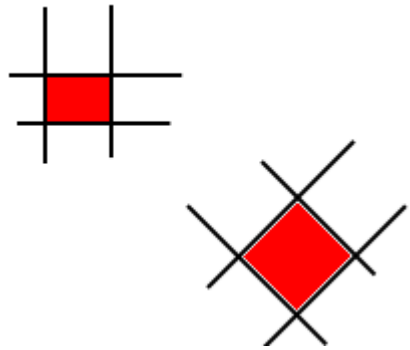
Чтобы проверить наличие пересечения многоугольников, достаточно проверить, существует ли точка, принадлежащая как первому, так и второму многоугольнику. Утверждается, что любой непустой многоугольник содержит хотя бы одну из точек пересечения непараллельных прямых (прямые мы берем как из первого, так и из второго многоугольников), докажем этот факт:
Предположим, что многоугольник не содержит точек пересечения прямых, тогда он пустой (не содержит точек вообще), так как его вершины - это точки пересечения прямых, а многоугольника без вершин не существует.

Параллельные прямые мы не рассматриваем. Если две параллельные прямые пересекаются в какой-то точке, то многоугольник вырождается в отрезок, концы которого являются точками пересечения этих прямых с другими непараллельными им. Вот эти точки в алгоритме мы и будем рассматривать.

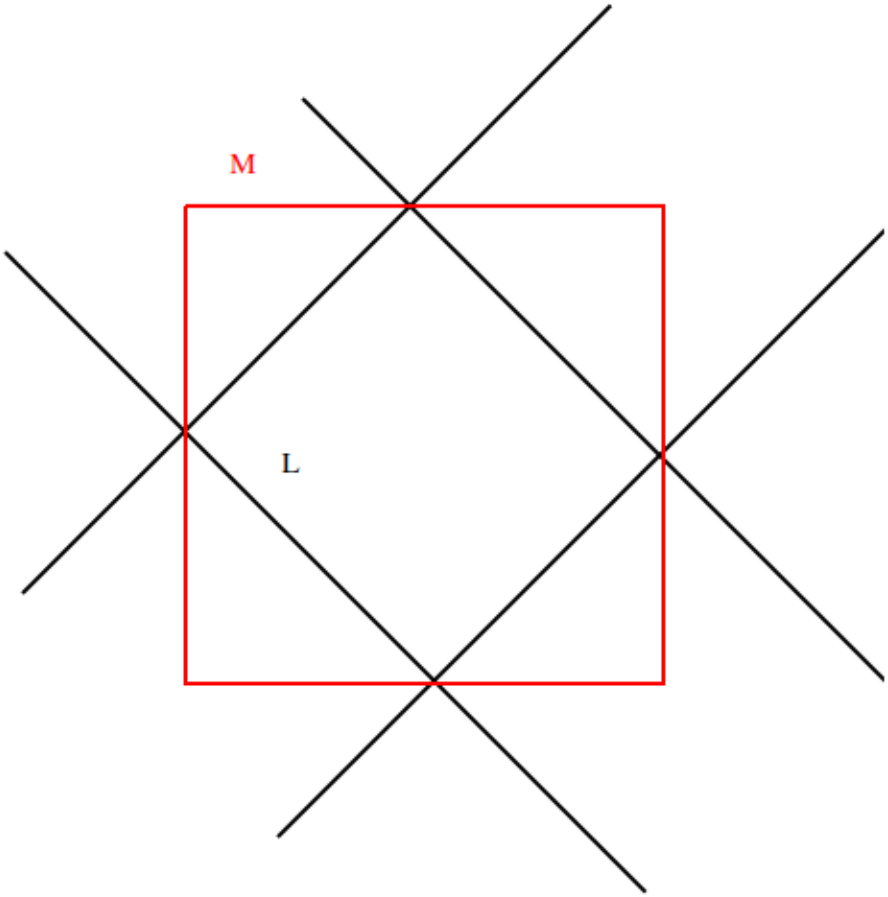
Алгоритм состоит в том, что мы ищем точки пересечения непараллельных прямых и проверяем, удовлетворяет ли хотя бы одна из них системе неравенств. Если удовлетворяет, то пересечение многоугольников не пусто.

UPD: Чтобы понять, пересекаются ли многоугольники, попробуем найти их пересечение: если оно пусто, то пересечения нет. Алгоритм проверки многоугольника на пустоту реализован в методе empty():

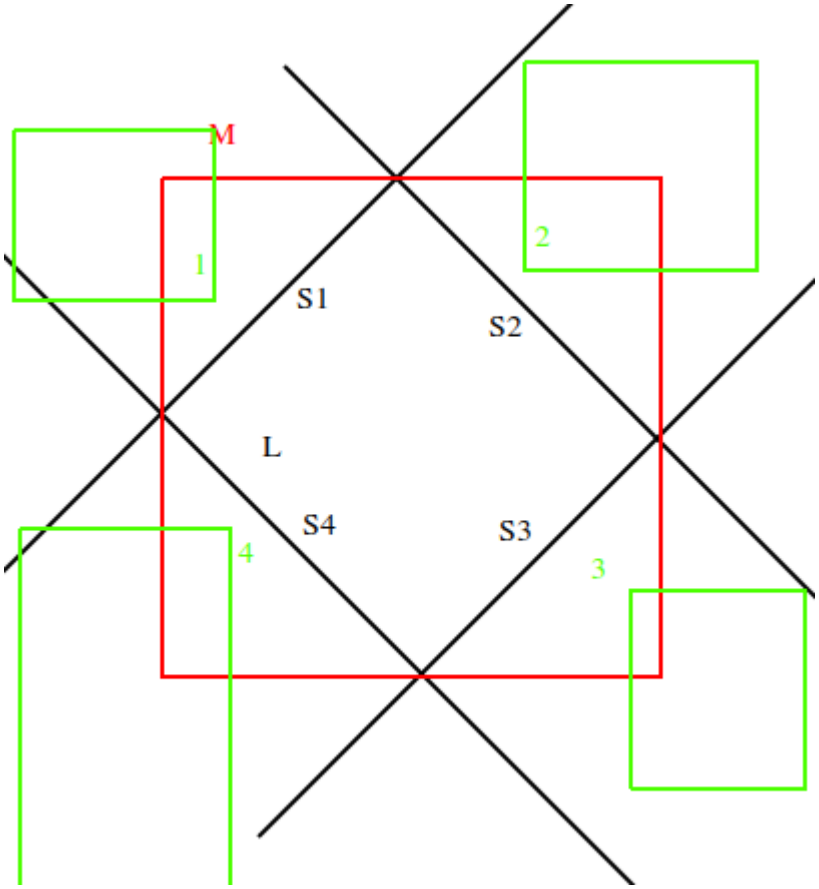
Для начала сделаем тривиальные действия: проверим, что каждая пара прямых корректна, т.е. верхняя прямая находится выше нижней и т.д. (например, x <= L1, -x <= L5, тогда -L5 <= L1). Теперь задача свелась к проверке пересечения двух прямоугольников, образованных диагональными прямыми и прямыми, параллельными осями координат (назовем их L и N соответственно).



Выделим прямоугольник M, в который вписан прямоугольник L:


M
L

Если N пересекается с L, то он пересекается и с M. Несложным образом проверим это. Если N пересекается с M, но не пересекается с L, то он имеет одно из 4 положений:



Каждый случай устанавливается одним из следующих условий: нижняя правая точка N лежит выше прямой S1, левая нижняя - выше прямой S2, левая верхняя - ниже прямой S3, правая верхняя - ниже прямой S4. Если прямоугольник N не подходит ни под один из этих случаев, то он пересекается с прямоугольником L => многоугольник не пустой.

```
Octagon* intersection(const Octagon& first, const Octagon& second)
```

Чтобы найти пересечение многоугольников, достаточно составить систему из 16 неравенств. Она очевидно упрощается до системы из 8 неравенств (по каждой паре соответствующих лимитов мы берем наименьший из них). Новую систему нужно нормировать, поскольку мы получаем произвольную систему.

Кроме всех вышеперечисленных алгоритмов, я пересекаю прямые, решая систему из двух неизвестных и двух равенств. Воспользовавшись методом Крамера, мы найдем решение этой системы (оно единственно, так как система невырожденная, непараллельные прямые пересекаются). В этом методе достаточно посчитать три определителя трёх матриц. Координаты точек пересечения всегда будут целыми.

UPD: Метод Крамера больше не используется.