Имеем составную фигуру, расположенную по частям в указанном порядке в массиве: 

Цель: корректно и оптимально определить слои в порядке расположения в массиве и количество фигур в каждом слое. Наложением одной фигуры на другую считаем пересечение обрамляющих прямоугольников этих фигур.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ФИГУРЫ** | | |
| **С**  **Л**  **О**  **И** | *1* | № 1 | № 2 |
| *2* | № 4 | № 3 |
| *3* | № 5 |  |

Для приведенной в качестве примера составной фигуры результат разбиения на слои будет следующим:

Задача: определить способ такого разбиения.

## «Копилка» | n2

Создаем временный массив, равный по размеру исходному (не забыть после выполнения вычислений освободить память из-под массива), состоящий из двух полей: беззнаковый целый тип и список из беззнаковых целых. Инициализируем каждую ячейку в качестве 1 для первого поля и null для второго.

Далее пишем внешний цикл: от нулевого индекса до декрементированного размера массива (не включая). Внутренний же цикл: от инкрементированного текущего индекса внешнего цикла до размера массива (не включая).

Во внутреннем цикле делаем проверку на то, что фигура с индексом внешнего цикла пресекается/не пересекается с фигурой с индексом внутреннего цикла. В случае пересечения проверяем список ячейки временного массива с индексом внутреннего цикла: если фигура внешнего цикла пересекается хотя бы с одной фигурой из списка или список вовсе равен null, тогда инкрементируем поле-счетчик этой ячейки; иначе ничего не делаем и переходим к следующей итерации.

В итоге имеем массив, где первое поле указывает на расположение фигуры на определенном слое. Еще раз обойдя массив, получаем число слоев и количество фигур для каждого слоя.

## Оптимизированная реализация

Хранение только номера слоя фигуры и в случае пересечения сравнивать у кого слой больше.

|  |
| --- |
| pylaev::Matrix pylaev::CompositeShape::split() const  {  Matrix matrix;  if (size\_ == 0) {  return matrix;  }  std::size\_t shapesLayers[size\_];  for (std::size\_t i = 0; i < size\_; i++) {  shapesLayers[i] = 0;  }  for (std::size\_t i = 0; i < size\_; i++) {  matrix.add(shapesArray\_[i], shapesLayers[i]);  rectangle\_t shapeFrameRect = shapesArray\_[i]->getFrameRect();  for (std::size\_t j = i + 1; j < size\_; j++) {  if (intersection(shapeFrameRect, shapesArray\_[j]->getFrameRect())) {  shapesLayers[j] = std::max(shapesLayers[j], shapesLayers[i] + 1);  }  }  }  return matrix;  } |