Описание алгоритмов:

NFDH(Next Fit Decreasing High).

Самый простой алгоритм , если не считать сортировку ,выполняется за O(n).Ставит прямоугольники подряд , когда закончилось место , переходит на новый уровень и закрывает старый.

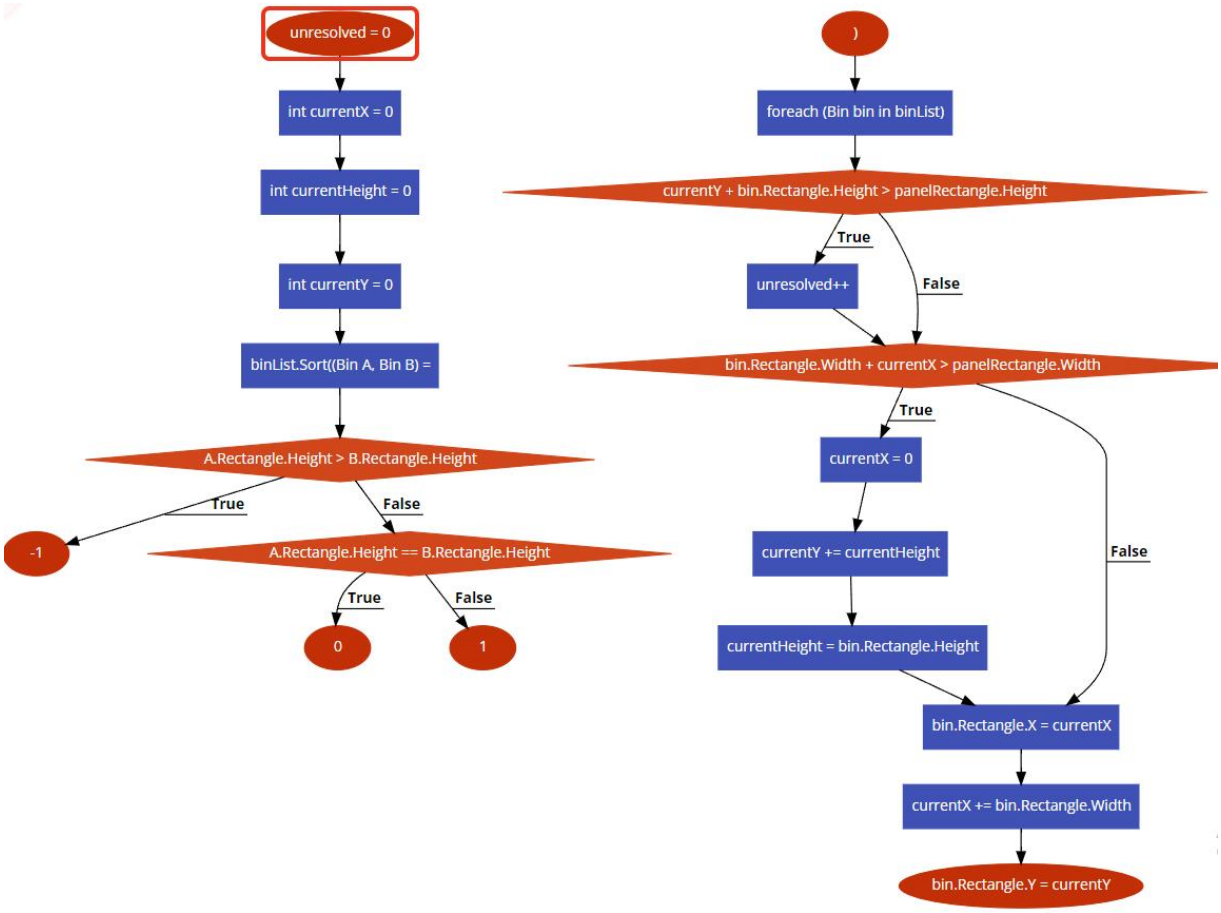
Сначала инициализируем переменную “unresolved” нулём , и создаём (определяем и инициализируем) локальные переменные:

- “currentX” которая хранит горизонтальную составляющую координаты следующего прямоугольника
- “currentHeight” которая хранит длинну первого прямоугольника в данном уровне
- “currentY” которая хранит вертикальную составляющую координаты следующего прямоугольника

Далее , с помощью лямбда-функции и метода Sort() , сортируется по высоте binlist.

Затем создаеём цикл foreach , где временная перерменная bin проходиться по листу binList , что происходит в цикле :

- Проверям прямоугольник по ширине (влезет ли в большой прямоугольник panelRectangle)
- Условный оператор , если прям. не влезает в этот уровень , то “создаём новый уровень” :
    - currentX присваиваем ноль , так как на следующем уровне прямоугольник должен быть слева
    - Увеличиваем currentY на высоту первого прямоугольника на прошлом уровне , то есть currentHeight
    - currentHeight меняем на высоту данного прямоугольника , так как он является первым на новом уровне.
- Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения current.X и current.Y.
- Увеличиваем currentX на ширину прямоугольнка.



FFDH(First Fit Decreasing Hight).

Следующий по сложности алгоритм , во время работы которого храниться и обрабатывается информация о каждом уровне. То есть внутри цикла прохода по листу bin , также осуществляется проход по всем предыдущим уровням и если прямоугольник смог влезть , то производиться выход из цикла и становление его на то самое место. Поэтому и O(n^2).

Сначала инициализируем переменную “unresolved” нулём , и создаём (определяем и инициализируем) локальные переменные и определяем массивы:

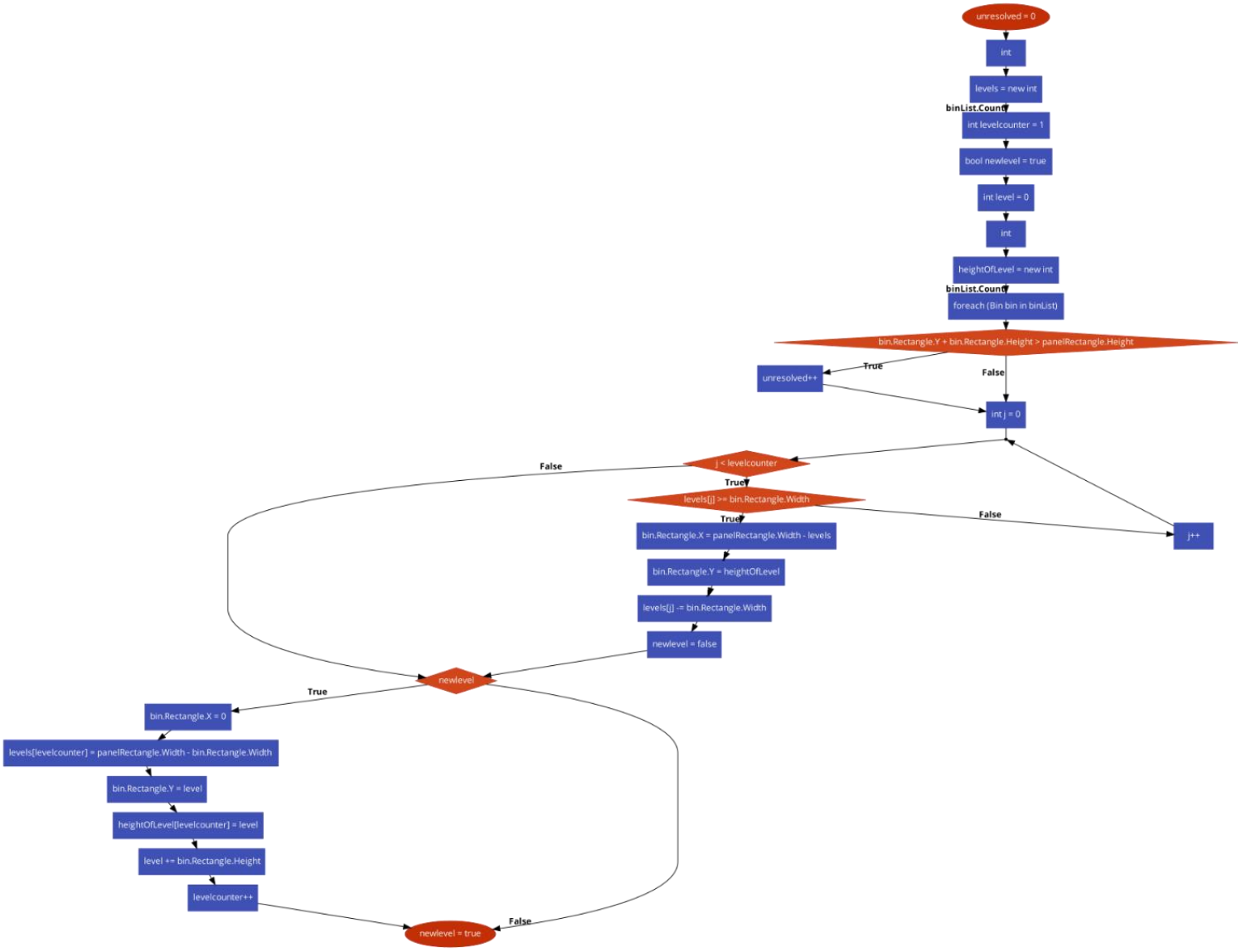
- “levels” массив int, в котором и будет храниться оставшаяся высота каждого уровня
- Счётчик “levelcounter” , считающий уровни
- Флаг “newlevel” , если true , то создаём новый уровень , следовательно если false ,то нет.
- “heightOfLevel” массив int хронящий высоту каждого уровня , для постановки его на нужную высоту.

Перед этим с помощью лямбда-функции и метода Sort() , сортируется по высоте binlist. Более подробно алгоритм продемонстрирован в NDFH.

Затем создаеём цикл foreach , где временная перерменная bin проходиться по листу binList , что происходит в цикле :

- Проверям прямоугольник по ширине (влезет ли в большой прямоугольник panelRectangle)
- Также создаём цикл и проходимся по уже созданным уровням :
    - Если на уровне есть место(ширина) ,то мы сразу же занимаем это место :
        - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .

        - Стоит заметить что в “levels” изначально храниться ширина контейнера (panelRectangle) и каждый раз при помещении нового прям. в уровень мы уменьшаем его на ширину этого прям.
        - “newlevel” делаем false ,так как новый уровнь нам уже не нужен.
        - Выходим из цикла.
- Если “newlevel” true , то :
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .
    - Инициализируем новый элемент массива вычитанием из ширины контейнера ширину прям. который мы ставим.
    - Создаём новый элемент массива “heighOfLevel ” инициализировав его высотой контейнера.
    - Увеличиваем высоту контейнера , то есть “level” , на высоту прям.
    - Увеличиваем на один количество уровней.
- Возвращаем в флаг “newlevel” значение true.



BFDH(Best Fit Decreasing Hight).

Фактически это тот же самый алгоритм FFDH , с одним лишь изменением в условном операторе выбора уровня и проходу по уровням. Вместо того чтобы найти уровень в который влезает

прямоугольник и выти из цикла нахождения подходящего уровня. В BFDH ищется наиболее подходящий уровень , то есть с наименьшим остатком после вставки соответствующего прямоугольника.

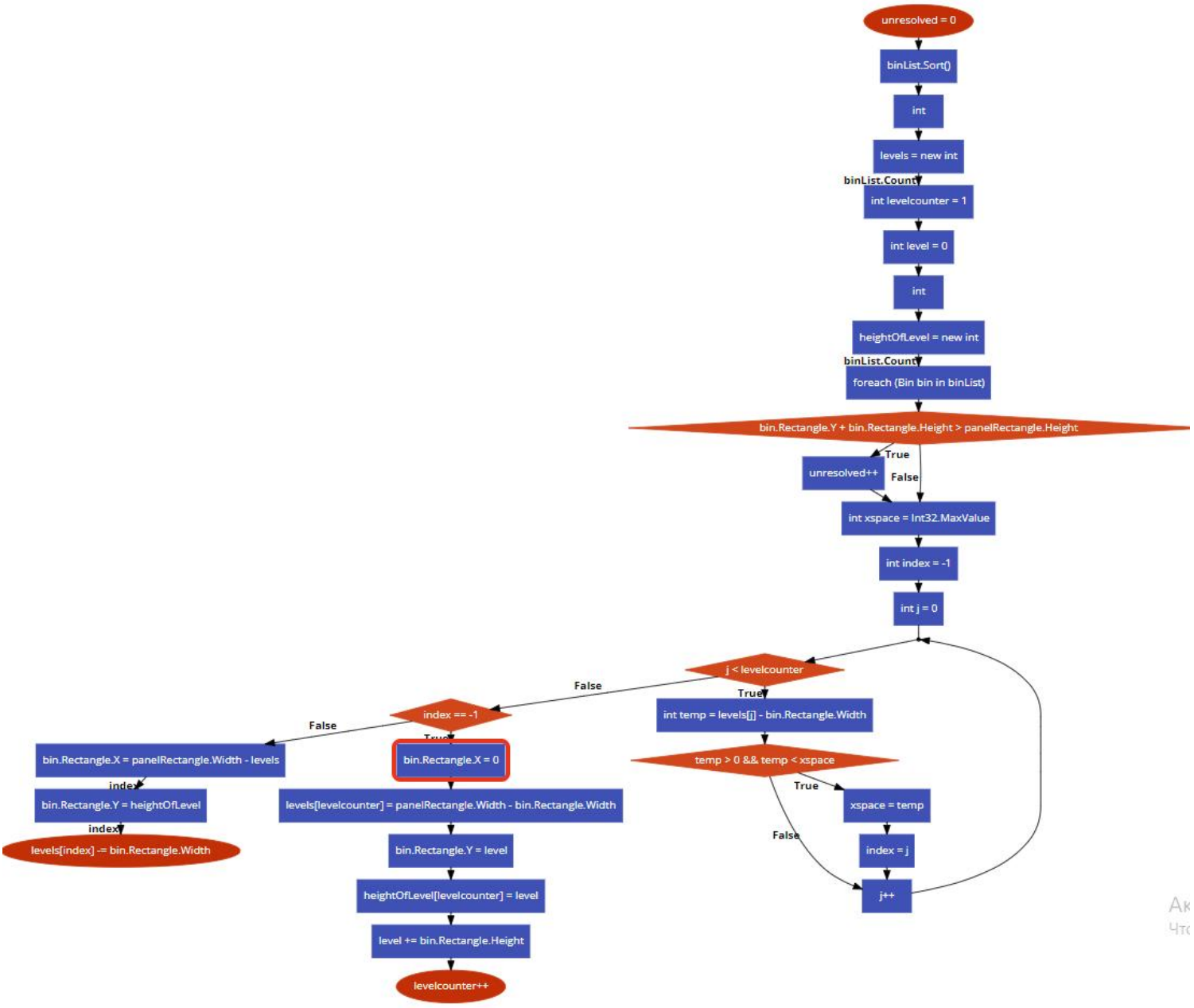
Сначала инициализируем переменную “unresolved” нулём , и создаём (определяем и инициализируем) локальные переменные и определяем массивы:

- “levels” массив int, в котором и будет храниться оставшаяся высота каждого уровня
- Счётчик “levelcounter” , считающий уровни
- “heightOfLevel” массив int хронящий высоту каждого уровня , для постановки его на нужную высоту.

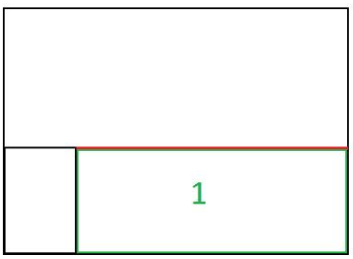
Перед этим с помощью лямбда-функции и метода Sort() , сортируется по высоте binlist. Более подробно алгоритм продемонстрирован в NDFH.

Затем создаеём цикл foreach , где временная перерменная bin проходиться по листу binList , что происходит в цикле :

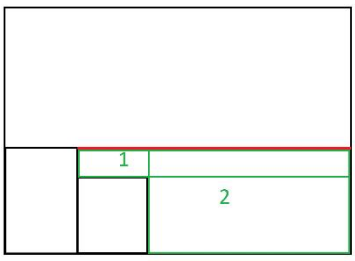
- Проверям прямоугольник по ширине (влезет ли в большой прямоугольник panelRectangle)
- Создаём локальную переменную “xspace” для нахождения уровня с наименьшей шириной , но при этом влезающим в него прямоугольником.Инициализируем её максимальным возможным занчением INT32.А также переменную для хранения индекса уровня в который должен будет поставлен прямоугольник и одновременно флагом создания нового уровня.В начале цикла поиска инициализируем её занением “-1” , следовательно если хотя бы раз прямоугольник влез в уровень , то эта перменная переопределяется и уже хронит индекс уровня.
- Также создаём цикл и проходимся по уже созданным уровням :
    - Создаём временную переменную для хранения дополнительной ширины , то есть той котораяостанется после постановки прямоуголтника в это место.
    - Если на уровне есть место(ширина) и дополнительная ширина(“temp”) меньше наименьшей дополнительной походящей ширины (“xspace”) ,то мы переопределяем “xspace” и index .”xspace” становится равен “temp” ,а “index ” индексу уровня.
- Если “index” равен “-1” , то :
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .
    - Инициализируем новый элемент массива вычитанием из ширины контейнера ширину прям. который мы ставим.
    - Создаём новый элемент массива “heighOfLevel ” инициализировав его высотой контейнера.
    - Увеличиваем высоту контейнера , то есть “level” , на высоту прям.
    - Увеличиваем на один количество уровней.
- Иначе :
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения и уменьшаем ширину полосы в “levels[index]”.

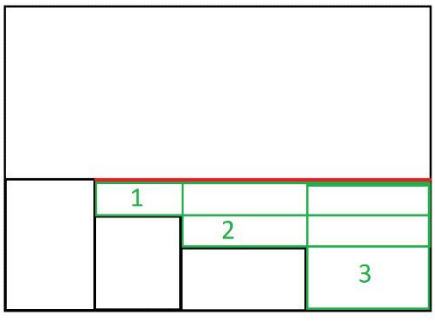


FCNR(Floor Celling No Rotation).



У алгоритмов описанных выше есть проблема которая заключается в том , что довольно много места остаётся между уровнями , а точнее чем длинне ширина контейнера тем больше этого пустого место которое ничем не занимается. Смысл данного алгоритма и нашей реализации состоит в том чтобы сохранять в специальном списке полости в которых потом могут размещаться прямоугольники .В данной реализации этот список называется “Possible” , формируется и меняется он на каждой итерации основного цикла.В самом начале , при постановке первого прямоугольника в новый уровень он состоит из одного прямоугольника размером с весь оставшийся уровень.В последующих итерациях меняется только последнний





элемент списка , то есть нижний прямоугольник. Пример показан на рисунках.Так же стоит более подробно разобрать каким образом изменяется список при постановке прямоугольника в потолок.Во первых как только это происходит ставить прямоугольник на пол запрещается , для этого нужен массив “PossFlag” , в котором и хранится булевое значение обозначающее начало постановки прямоугольников на потолок.Естественно когда деталь уже поставлена , необходимо удалить их списка “Possible” место которое она занимает , также очевидно что все детали которые были до неё должны быть уменьшены по ширине , а те что после также удалены.

Сначала инициализируем переменную “unresolved” нулём , и создаём (определяем и инициализируем) локальные переменные и определяем массивы на рисунке 1:

- “levels” массив int, в котором и будет храниться оставшаяся высота каждого уровня
- Счётчик “levelcounter” , считающий уровни
- “heightOfLevel” массив int хронящий высоту каждого уровня , для постановки его на нужную высоту.
- “reverselevel” массив для постановки прямоугольника в нужные координаты.
- “Possible” и “PossFlag” описаны выше.

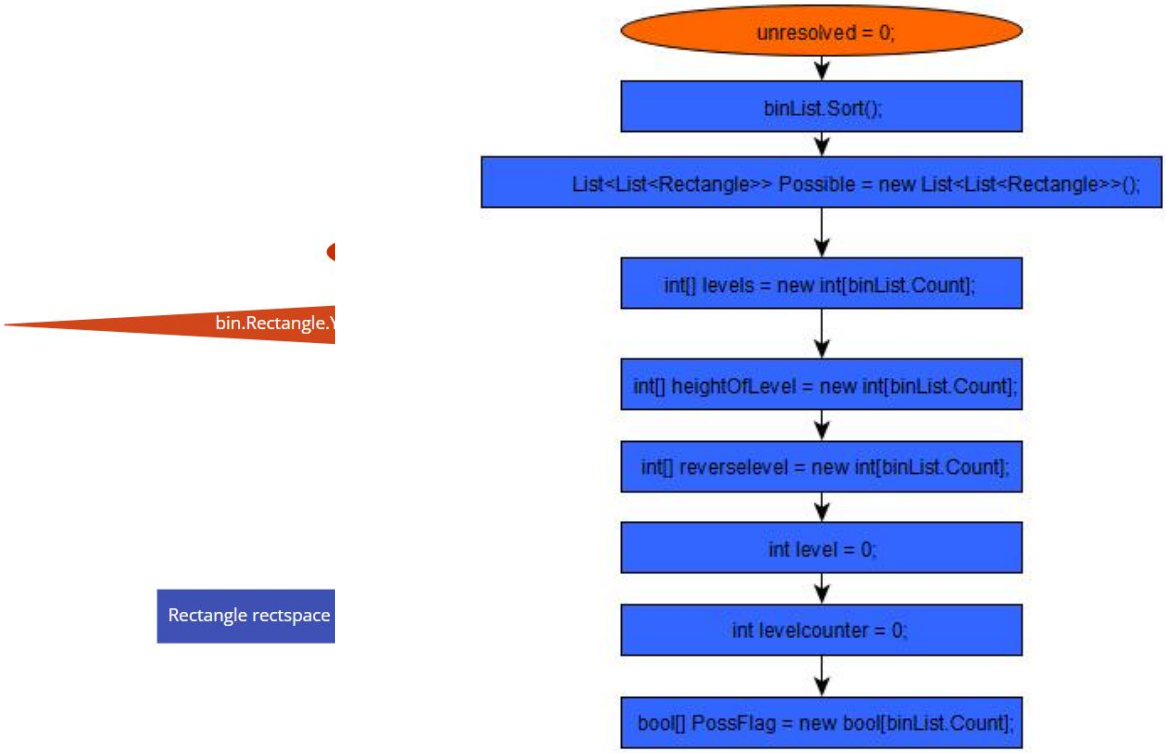
Перед этим с помощью лямбда-функции и метода Sort() , сортируется по высоте binlist. Более подробно алгоритм продемонстрирован в NDFH.

Затем создаеём цикл foreach , где временная перерменная bin проходиться по листу binList , что происходит в цикле :

- Проверям прямоугольник по ширине (влезет ли в большой прямоугольник panelRectangle)
- Создаём локальную переменную “xspace” для нахождения уровня с наименьшей шириной , но при этом влезающим в него прямоугольником.Инициализируем её максимальным возможным занчением INT32.А также переменную для хранения индекса уровня в который должен будет поставлен прямоугольник и одновременно флагом создания нового уровня.В начале цикла поиска инициализируем её занением “-1” , следовательно если хотя бы раз прямоугольник влез в уровень , то эта перменная переопределяется и уже хронит индекс уровня.
- Создаём локальную переменную “rectspace” для нахождения в “Possible” прямоугольника с наименьшей шириной и высотой , но при этом влезающим в него.Инициализируем поля прямоугольника максимальными возможными занчениями INT32.А также переменную для хранения индекса уровня и индекс самого элемента в этом уровне и одновременно флагом создания нового уровня.В начале цикла поиска инициализируем её занением “-1” , следовательно если хотя бы раз прямоугольник влез в уровень , то эта перменная переопределяется и уже хронит индекс уровня.
- Также создаём цикл и проходимся по уже созданным уровням :
    - Создаём временную переменную для хранения дополнительной ширины , то есть той которая останется после постановки прямоуголтника в это место.
    - Если на уровне есть место(ширина) и дополнительная ширина(“temp”) меньше наименьшей дополнительной походящей ширины (“xspace”) ,то мы переопределяем “xspace” и index .”xspace” становится равен “temp” ,а “index ” индексу уровня.
    - Для потолка создаётся ещё один цикл который проходиться уже по каждому элементу в списке “Possible” и находит наболее подходящий.
- Если “index” равен “-1” , то :
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .

- Инициализируем новый элемент массива вычитанием из ширины контейнера ширину прям. который мы ставим.
- Создаём новый элемент массива “heighOfLevel ” инициализировав его высотой контейнера.
- Увеличиваем высоту контейнера , то есть “level” , на высоту прям.
- Увеличиваем на один количество уровней.

- Иначе :
  - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения и уменьшаем ширину полосы в “levels[index]”.
-
- Также создаём цикл и проходимся по уже созданным уровням :
  - Создаём временную переменную для хранения дополнительной ширины , то есть той котораяостанется после постановки прямоуголтника в это место.
  - Если на уровне есть место(ширина) и дополнительная ширина(“temp”) меньше наименьшей дополнительной походящей ширины (“xspace”) ,то мы переопределяем “xspace” и index .”xspace” становится равен “temp” ,а “index ” индексу уровня.
- Если “index” равен “-1” и “indexlevel” равен “-1” , то есть создаётся новый уровень :
  - Инииализируем “reverslevel” длинной контейнера.
  - Создаём уровень в Possible , то есть ещё один лист .Определяем и инициалиазируем первый элемент.
  - Инициализируем новый элемент массива “ levels” вычитанием из ширины контейнера ширину прям. который мы ставим.
  - Создаём новый элемент массива “heighOfLevel ” инициализировав его высотой контейнера.
  - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .
  - Увеличиваем высоту контейнера , то есть “level” , на высоту прям.
  - Увеличиваем на один количество уровней.
- Иначе :
  - Если “xspace” меньше ширины “rectspase” , то есть упаковка на полу будет плотнее , и мы ещё не упаковывали на потолке , то ставим прямоугольника на пол:
    - Как и было описано выше в массив “Possible” специальным образом добавляется ещё один элемент.
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .
    - Уменьшаем значение массива “levels”.
  - Иначе , упаковываем на потолок:
    - Уменьшаем “reverselevel”.
    - Присваиваем bin.Rectangle.X и bin.Rectangle.Y соответсвующие значения .
    - Как и описывалось выше удаляются и уменьшаются элементы списка Possible.



исунок 2

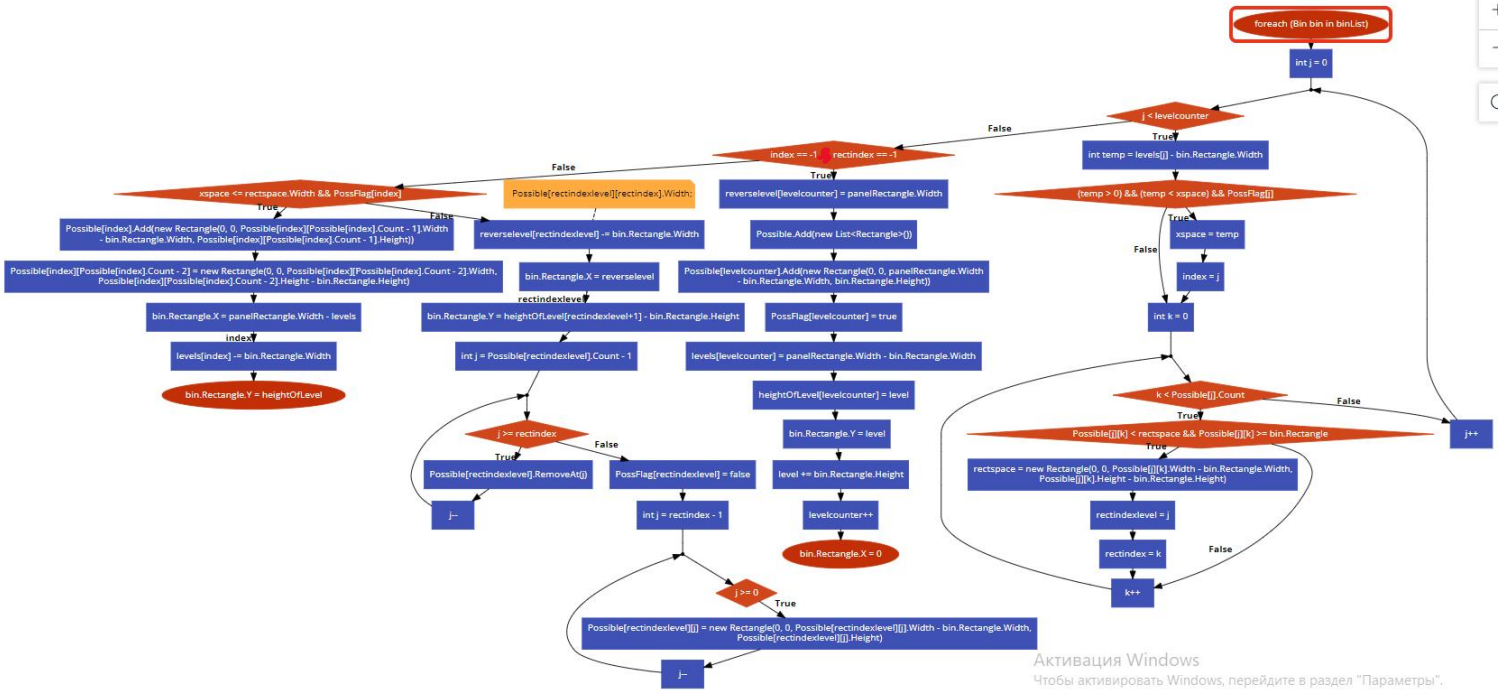


Рисунок 3