**Министерство образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

**Алгоритмы и структуры данных**

**Домашняя работа:**

Замещение прямоугольника полимино

**Преподаватель:**

Чесноков В. О.

**Студент**:

Валитов Р. Ш.

ИУ8-54

Москва 2016

Оглавление:

[Задача: 2](#_Toc468903217)

[Теоритическая часть: 2](#_Toc468903218)

[Описание известной задачи: 3](#_Toc468903219)

[Теорема: 3](#_Toc468903220)

[Доказательство: 3](#_Toc468903221)

[Математическая интерпретация: 6](#_Toc468903222)

[Основные подходы к решению: 6](#_Toc468903223)

[Описание метода решения задачи: 6](#_Toc468903224)

[Описание структур: 7](#_Toc468903225)

[Практическая часть: 7](#_Toc468903226)

[Тесты ввода вывода: 7](#_Toc468903227)

[Основная программа: 8](#_Toc468903228)

[Coordinates: 8](#_Toc468903229)

[Substring: 9](#_Toc468903230)

[PolyminoComparative: 9](#_Toc468903231)

[Polymino: 9](#_Toc468903232)

[State: 9](#_Toc468903233)

[Field: 9](#_Toc468903234)

[Алгоритм: 9](#_Toc468903235)

[Пример: 12](#_Toc468903236)

[Сложности основных алгоритмов: 13](#_Toc468903237)

[Заключение: 13](#_Toc468903238)

# Задача:

Пусть у вас есть несколько плоских фигур различной формы. Каждая фигура вырезана из листочка в клеточку строго по линиям и не содержит отверстий. Напишите алгоритм укладки этих фигур в прямоугольник. Фигуры можно вращать на угол, кратный 90º и отражать по горизонтали или вертикали.

# Теоритическая часть:

Полимино (полиомино, polyomino) - плоская геометрическая фигура, состоящая из N одноклеточных квадратов, соединенных по сторонам.

Замощение плоскости (tiling) - представление плоскости в виде множества непересекающихся полимино.

Паркет или замощение — разбиение [плоскости](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%BE%D1%81%D0%BA%D0%BE%D1%81%D1%82%D1%8C_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)) [многоугольниками](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA) (или пространства многогранниками) без пробелов и перекрытий.

## Описание известной задачи:

Задача замещения конкретного прямоугольника заданным набором полимино является частным случаем задачи об упаковке в контейнеры (и ее варианта offline-2DSP)

Различия задач в том, что на вход подаются не прямоугольники, а полимино и укладываются они не в непрерывную полуограниченную полосу, а в дискретный прямоугольник заданных размеров. Но если в задаче об упаковке допустить, что стороны прямоугольников будут кратны какому-либо числу, и конфигурация некоторого множества прямоугольников постоянна, то можно говорить о схожести заданий.

Также существует задача о замещении плоскости набором полимино. Доказано, что она алгебраически неразрешима.

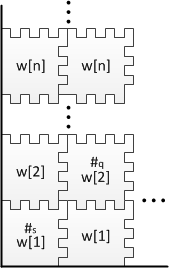
### Теорема:

Задача о замощении четверти плоскости полимино неразрешима.

### Доказательство:

[Сведём](http://neerc.ifmo.ru/wiki/index.php?title=M-%D1%81%D0%B2%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) задачу останова к данной задаче. Пусть дана [машина Тьюринга](http://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0)  =\langle \Sigma, Q, \Pi, B  \in \Pi, s,\delta: Q \times \Pi \rightarrow Q \t и слово  \in \Sigma^*. Требуется определить, остановится ли данная МТ на входе http://neerc.ifmo.ru/wiki/images/math/1/d/6/1d69a7408de15f155cd6b3d17c3bed5e.png.

Будем эмулировать процесс выполнения МТ путем построения вертикальных рядов, каждый из которых эквивалентен конфигурации МТ на определенном этапе выполнения. Первый ряд заполняется начальной конфигурацией, а каждый следующий ряд соответствует следующей конфигурации.

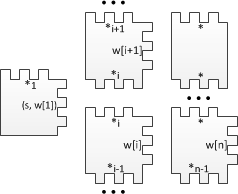
[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_init.png)

Теперь на основе заданной МТ будем строить набор полимино, которые будут иметь следующий вид:

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_gen.png)

На каждой стороне такого полимино находится определенное число выступов/впадин. Каждому символу из алфавита, состоянию и паре из состояния и символа сопоставим некоторое уникальное число (можно ограничить  \le |\Pi| + |Q| + |\Pi \times Q| + 1) – это и будет количество выступов/впадин находящихся на одной стороне полимино.

Сначала построим набор полимино, который задаёт начальную конфигурацию:

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_start.png)

где http://neerc.ifmo.ru/wiki/images/math/6/c/8/6c8b7754aa40f5503dea2645a547fe28.png – уникальные числа для каждых соседних двух полимино из начальной конфигурации. Первое полимино характеризует начальное состояние, последующие за ним кодируют входное слово, и завершающее полимино требуется для корректного замощения оставшейся части ряда.

Далее строим полимино для всех элементов алфавита  \in \Pi:

[olyomino alph.png](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_alph.png)

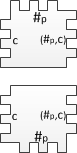
В нем количество впадин слева равно количеству выступов справа. Такой тип полимино передает содержимое ленты МТ следующему ряду.

Теперь построим полимино для функции перехода delta (a, c) = \langle p, d, D \rangle, где  \in Q, p \in Q, c \in \Pi, d \in \Pi, D\in \{\leftarrow, \downarrow, \rightarrow \}:

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_delta.png)

На рисунке изображены (сверху вниз) полимино соответствующие значениям  = \{\leftarrow, \downarrow, \rightarrow \}. Вместе со следующим типом они эмулируют перемещение головки МТ.

Далее построим следующий тип полимино:

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_delta2.png)

Эти полимино получают на вход символ алфавита http://neerc.ifmo.ru/wiki/images/math/c/e/f/cef160c9876cc23aa59313fa78238223.png от предыдущего ряда и состояние http://neerc.ifmo.ru/wiki/images/math/5/4/1/541365d1bab99f1a2a885ee6bfa42451.png от соседнего полимино, а затем передает следующему ряду пару из состояния и символа.

Построим последний тип полимино, характеризующие состояния #_Y и #_N:

[olyomino halt.png](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Polyomino_halt.png)

Такое полимино имеет уникальное число выступов справа. Ни одно другое полимино из полученного набора не сможет к нему присоединиться, и процесс дальнейшего замощения будет невозможен.

Полученный алгоритм сведения получает на вход МТ и слово, а на выход выдает соответствующий им набор полимино.

Таким образом, четверть плоскости можно замостить тогда и только тогда, когда закодированная МТ не останавливается на данном входе. Иными словами есть бесконечное количество конфигураций, не переходящих в конечное состояние. Это значит, что мы сможем замощать плоскость ряд за рядом бесконечное количество раз, что в результате замостит плоскость.

Если же МТ остановится, то и замостить четверть плоскости мы не сможем из-за того, что конечное полимино не имеет продолжения. Значит задача о замощении полимино не разрешима.

## Математическая интерпретация:

Обозначим S – прямоугольник размера m x n,

Изобразим на S решетку состоящую из единичных квадратов.

Пусть ХУ ориентированная ломанная на S, с началом в точке Х и концом в точке У, образованная линиями решетки.

Возьмем группу G, в ней два элемента А и В, движению ломанной вверх сопоставим А, вправо – В, соответственно А^(-1) – вниз, А^(-1) – влево. Операцией определим соответствующее движение в заданном направлении. Таким образом можно описать наше полимино, с обязательным условием совпадения начала и конца. Обозначим полимино через Р.

Наша задача, составить функцию:

, где G – функция, присваивающая каждому из полимино координату начала так, что S полностью замощена элементами Р.

## Основные подходы к решению:

Для того, чтобы определить, можно ли соединить полимино будет проводиться анализ описание полимино через элементы множества выбранной группы.

Для фигур имеющих общие элементы с несколькими другими фигурами будет применен метод ветвей и границ. Основное достоинство метода в том, что конфигурации фигур после склейки, не оптимальные для данной задачи, будут отброшены.

## Описание метода решения задачи:

Для начала будет проведена проверка входных данных, для того чтобы отсеять условия с неправильными полимино, с прямоугольниками не позволяющими замостить их без наложения и прямоугольниками не позволяющими вместить самый широкий и самый высокий элементы.

После начальных проверок будет проведена проверка на возможность размещение исходных полимино по периметру прямоугольника. Если входные данные корректны, то возможны случаи нескольких вариантов размещения полимино по периметру. На этот случай нам поможет анализ описания полимино. Благодаря его структуре, анализ поможет нам сразу проверить все возможные ориентации фигуры в пространстве используя сериальное кодирование.

На случай вариативности расположения очередной фигуры мы используем метод ветвей и границ, который позволит нам убрать не корректные и не оптимальные случаи.

## Описание структур:

Прямоугольник будет представлен в виде массива двумерного массива, координаты узлов будут задаваться нумерацией вертикальных и горизонтальных линий решетки.

Фигуры полимино будут задаваться последовательностями А и В, а также точкой начала координат и номером ориентации.

Структура возможных вариаций будет представлена в виде бинарного дерева, в узлах которой поместятся координаты фигуры поставленной с допущением.

# Практическая часть:

## Тесты ввода вывода:

Прототип программы содержится в папке Polymino и представляет собой несколько модулей:

1. main

Здесь происходит взаимодействие с пользователем, считывание из файла и запись в структуру данных. Заглушка тоже тут.

1. Polymino

В классе Polymino организована структура данных, включающих в себя двумерное

представление фигуры в виде массива, его площадь, количество идентичных фигур, размерность, строку группового представления, и проверку на корректность.

1. print, functions, errors, Coodinates

Вспомогательные структуры и методы

1. input\_check

Данный файл содержит основные методы проверки входных данных.

Контракт таков, что в первой строке содержится количество фигур. Далее идет primary\_key фигуры (для упрощения тестирования, в релизе не будет), размер массива и сам массив. Основные проверки входных данных связаны с тем, что в фигурах могут содержаться отверстия или в одном представлении могут содержаться несколько фигур. Также присутствуют проверки на некорректные данные в самом массиве (элементы отличные от 1 или 0).

В папках IOTest и OITest находятся тесты для входных и выходных данных соответственно.

Для IOTest в папке TestGenerator лежит программа генерирующая входные файлы, содержащие до 10000 объектов шириной до 15 единиц. Если подать такой файл на вход IOTest, то в выходном файле будет описан каждый из обьектов, и соотвественно содержащиеся в нем ошибки.

Для OITest в папке Tests лежат разные выходные файлы, которые можно подать на вход программе и на выходе получить данные о корректности файла.

## Основная программа:

Для решения задачи, мною были написаны следующие классы:

* Field
* Coordinates
* Polymino
* State
* Substring
* PolyminoComparative

Далее я проведу обзорный анализ данных классов.

## Coordinates:

Класс координат в двумерном пространстве, с перегруженными операторами присваивания и сравнения.

## Substring:

Класс, предоставляющий интерфейс для работы с наибольшей общей подстрокой группового представления поля и фигуры. Хранит строку, длину, индексы начала подстрок в поле и в фигуры и номер вращения фигуры.

## PolyminoComparative:

Класс, хранящий в себе все возможные лучшие варианты общих подстрок для конкретной фигуры в отдельный момент работы программы. Также содержит все подстроки максимальной длины, все возможные новые групповые представления поля, и объект класса Polymino

## Polymino:

Класс, объекты которого представляют из себя фигуры полимино. Построение фигуры делается с помощью двумерного массива из нулей и единиц. В классе определены методы проверки полимино на наличие нескольких фигур в матрице, проверки на отверстия, функция генерации группового представления, и всех вращений фигуры.

## State:

Класс, хранящий в себе конкретное состояние поля в текущий момент. Является элементом иерархичной структуры, описывающей все переходы поля, подчиняющихся жадной эвристике метода ветвей и границ.

## Field:

Основной класс программы, единственный объект которого хранит всю информацию о фигурах, возможных размерах прямоугольника, дереве состояний и площади поля.

Для ознакомления, определения классов прописаны в файлах с расширением .hpp.

# Алгоритм:

1. На вход программе подается два пути:
   1. к файлу в котором находятся фигуры
   2. к файлу, в который запишем ответ
2. В файле фигуры записаны в следующем виде:
   1. Первое число в файле – количество фигур
   2. Далее для каждой фигуры:
      1. Порядковый идентификатор
      2. Размер минимального квадрата, в который можно вписать фигуру
      3. Количество фигур данного вида
      4. Двумерный массив из нулей и единиц, показывающий конфигурацию клеток в фигуре.
3. Создаем объект класса Field, конструктору на вход подаем argc и argv для проверки корректности ввода в конструкторе.
4. В конструкторе заполняются следующие поля класса Field:
   1. vector<Polymino> valid\_data – заносим информацию о всех фигурах поданных на вход
   2. vector<Coordinates> valid\_sizes – все возможные размеры исходного прямоугольника
5. Вызываем метод bool Field::tryLocalOptimum() – для построения иерархии состояний в поле класса vector<State> state\_tree;
6. Далее для каждого состояние вызываем метод State State::countStep(int new\_state\_id), который порождает новое состояние.
   1. Внутри данного метода анализируются исходные фигуры, и та фигура, чья наибольшая общая подстрока с групповым представлением поля максимальная, занимает свое место внутри поля, групповое представление поля пересчитывается, возвращается новое состояние.(если новое состояние не проходит проверку, прежде чем перейти к фигурам с меньшими общими подстроками проверяются все вращения исходной фигуры)
7. Состояние порождаются до тех пор, пока новое не станет финальным, то есть наибольшая общая подстрока будет полностью совпадать с фигурой и групповым представлением поля.
8. Если замечено такое состояние, выполнение программы прекращается, выводится ответ.
9. Если мы перебрали все состояния, а финального так и не нашли, говорим о том, что решение не найдено.
10. Ответ выводится в формате:

A - Right

B - Down

C - Left

D - Up

Answer:

3.

BBCDDA

2.

BBCCDCDAAA

1.

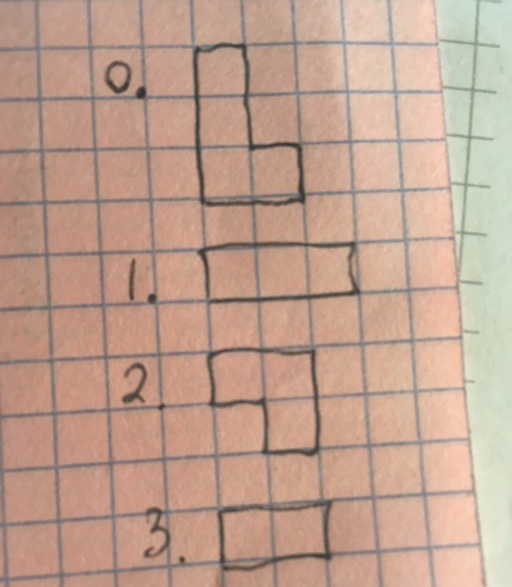
AAABBBCCDCDD

0.

AAAABBBCCCCDDD

, где первые пять строчек – подсказка

следующее число – идентификатор фигуры и далее его групповое представление.

Данный ответ выведен для следующих фигур

## Пример:

1. 3.

BBCDDA

1. 2.

BBCCDCDAAA

1. 1.

AAABBBCCDCDD

1. 0.

AAAABBBCCCCDDD

Что совпадает с исходными данными

# Сложности основных алгоритмов:

1. Проверка массива:
   * Состоит из трех проверок:

Проверка на допустимые числа О(n),

где n – количество элементов массива

Проверка на наличие нескольких фигур в массиве:

Проверка с использованием буфера(массив) и стека:

* По памяти – О(n)

(Т.к. в буфер заносятся только соседи единиц , которых не более 8.)

* По времени – О(n^2log(n))

(Т.к. производится поиск значений в буфере)

Аналогично для проверки отверстий

1. Перевод в групповое представление:
   * По памяти – О(n) – т.к. строится вспомогательный массив
   * По времени – О(n) – несколько раз обходим вспомогательный массив
2. Найти наибольшую общую подстроку для циклических строк
   * По памяти - О(n\*m), где m и n – длины строк.
   * По времени - О(n\*m), метод динамического программирования
3. Проверить групповое представление
   * По памяти – О(n) , координаты заносятся в буфер
   * По времени – О(n^2log(n)) , т.к. производится поиск значений в буфере
4. Посчитать площадь и все пары сторон:
   * По памяти – O(n)
5. Основной алгоритм перебора:
   * По памяти – O(e^n) – т.к. храним данные в иерархической структуре состояний
   * По времени – O(e^n) – проходимся по структуре состояний

# Заключение:

Таким образом, я на практике применил знания, полученые на курсе “Алгоритмы и структуры данных”, реализовав алгоритм, способный найти локальное решение NP-hard задачи.