Министерство образования Российской Федерации
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ДОМАШНЯЯ РАБОТА:

Полимино

Преподаватель:
Чесноков В. О.
Студент:
Петропавлов Р. И.
ИУ8-53

Москва 2017

Оглавление:

Практическая часть:

Описание классов.

1. Data
2. Coordinates
3. Figures
4. Condition
5. Substring
6. FiguresVariant

1.Data:

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

2.Coordinates:

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

3. Figures:

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

4.Condition:

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.

5. 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.

6. FiguresVariant:

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 Figures.

Алгоритм:

1. На вход программе подается два пути :
    a. к файлу в котором находятся фигуры.
    b. к файлу, в который запишем ответ.
2. В файле фигуры записаны в следующем виде:
    a. Первое число в файле – количество фигур.
    b. Далее для каждой фигуры:
    - Порядковый идентификатор.
    - Размер минимального квадрата, в который можно вписать фигуру.
    - Количество фигур данного вида.
    - Двумерный массив из нулей и единиц, показывающий конфигурацию клеток в фигуре.
3. Создаем объект класса Data, конструктору на вход подаем argc и argv для проверки корректности ввода в конструкторе.
4. В конструкторе заполняются следующие поля класса Field:
    a. vector<Figures> valid_data – заносим информацию о всех фигурах поданных на вход.
    b. vector<Coordinates> valid_sizes – все возможные размеры исходного прямоугольника.
5. Вызываем метод bool Data::tryLocalOptimum() – для построения иерархии состояний в поле класса vector<Condition> data_tree;
6. Далее для каждого состояние вызываем метод Data Data::countStep(int new_state_id), который порождает новое состояние.
    a. 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 последовательностью букв(R – Right, D – Down, L – Left, U - Up

Пример:

Если на вход программе подать данную посследовательность:

```
2
0
4
2
1 0 0 0
1 0 0 0
1 1 0 0
1 1 0 0
1
3
2
1 1 1
1 1 1
0 0 0
```

То на выходе мы должны получить следующий ответ:

```
0.
DDLLLLURRURR
0.
RRRRDDDLLLLUUU
1.
RRRRRRDDDLLLLLLUUU
1.
RRRRRRRRDDDLLLLLLLLUUU
```

Тесты программы:

Тесты программы лежат в репозитории на Github: https://github.com/BMSTU732/Polymino/tree/master/Tests

Там же лежит файл – **Tests.md**, описывающий содержащиеся там тесты.

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).

1. Файлы 00x.dat - это входные файлы, данные туда заносятся в соответсвии с правилами, описанными в пункте **Алгоритм.**

2.1 Файлы 001.ans-003.ans – это выходные файлы, данные хранятся заносятся в соответсвии с правилами, описанными в пункте **Алгоритм.**

2.2 Файл 004.ans и 005.ans – это выходные файлы, которые проверяют основные алгоритмы программы на корректность (данные алгоритмы описанные в пункте **«Сложность используемых алгоритмов»**).

Сложность используемых алгоритмов

	Алгоритмы	*Сложность по времени*	*Сложность по памяти*
1	*Проверка массива* *а)на допустимые числа* *б) на наличие нескольких фигур в массиве*	*а) O(n)* *б) O(n^2log(n))*	*а)O(n)* *б) O(n)*
2	*Перевод в групповое представление*	*O(n)*	*O(n)*
3	*Найти наибольшую общую подстроку для циклических строк*	*O(n*m), где m и n – длины строк.*	*O(n*m), где m и n – длины строк.*
4	*Проверить групповое представление*	*O(nlog(n))*	*O(n)*
5	*Посчитать площадь и все пары сторон*	*O(n)*	*O(n)*
6	*Основной алгоритм перебора*	*O(e^n)*	*O(e^n)*

Пояснение сложности алгоритмов

1. Проверка массива
    a) На допустимые числа:

    По времени и памяти - обычный обход каждого числа (которых у нас n)

    b) На наличие нескольких фигур в массиве:

    По памяти - проверка проводится с использованием массива и стека (в массив заносятся только соседи единиц)

    x x x

    x 1 x

    x x x

    По времени – используется бинарный поиск, который требует предобработку (сложность которой Не менее O(nlogn))
2. Перевод в групповое представление:
    a) **По памяти** – строится вспомогательный массив
    b) **По времени** – Обходим вспомогательный массив (n раз)
3. Найти наибольшую общую подстроку для циклических строк:
    a) **По памяти** – m и n – длины строк
    b) **По времени** – используется метод динамического программирования, то есть разбиение на несколько подстрок
4. Проверить групповое представление:
    a) **По памяти** – каждая координата заносится в массив
    b) **По времени** – аналогично для проверки на наличие нескольких фигур в массиве
5. Посчитать площадь и все пары сторон:
    a) **По памяти** – обход массива с n элементами.
6. Основной алгоритм перебора:
    a) **По памяти** – храним данные в иерархической структуре состояний

b) Проходимся по структуре состояний (для малых значений n, данная сложность оптимальнее полиномиальной)

Заключение:

В результате домашнего задания, мною была реализована программа на языке C++ Я на практике применил знания, полученные на курсе “Алгоритмы и структуры данных”, реализовав алгоритм, способный найти решение NP-сложной задачи. Все данные, касающиеся работы были выложены в публично доступном репозитории на Github: https://github.com/BMSTU732/Polymino