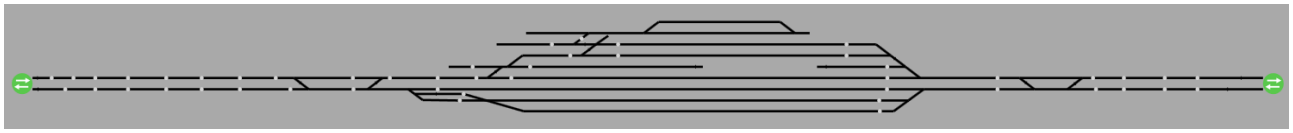
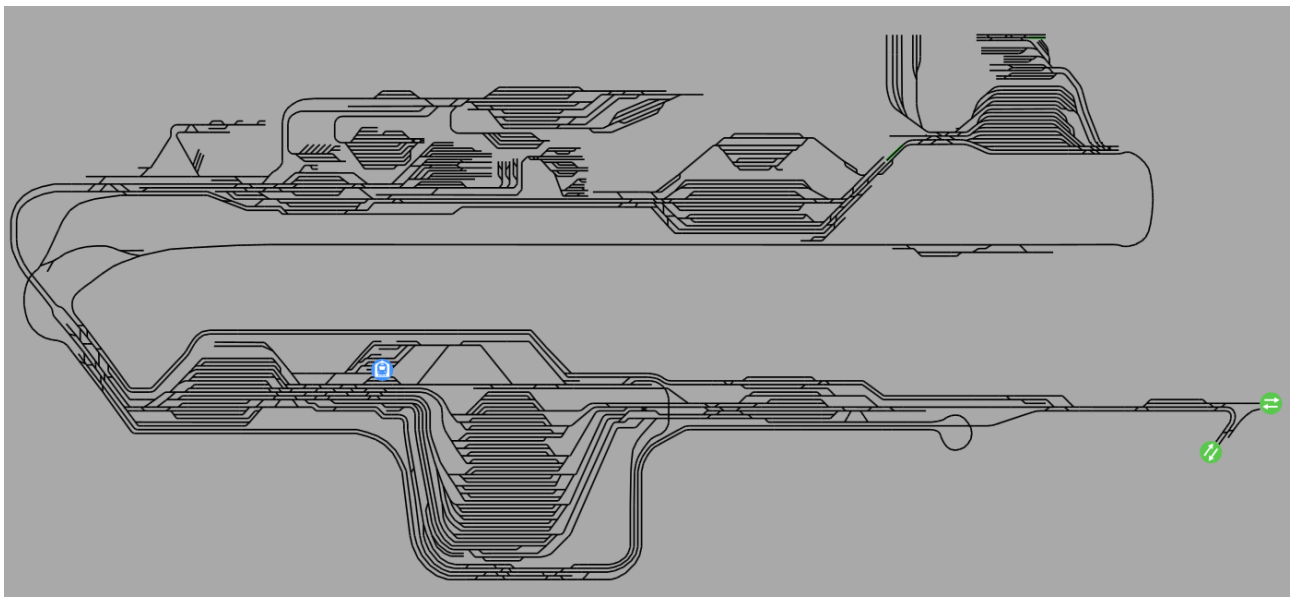
Тестовое задание

Мы занимаемся разработкой ПО для моделирования работы ЖД станции.

Упрощенно, схема станции - это набор участков. Участки могут быть объединены в пути, а пути, в свою очередь, могут принадлежать определенному парку.

Примеры станций:





Упрощая еще раз.

Участки пути - это *отрезки*. Отрезки, имеющие общие точки, могут быть объединены в *ломанные* (пути). Ломанная может быть ассоциирована с некоторой *сущностью* (парк).

Что требуется сделать.

1. Придумать и реализовать структуру данных, которая будет содержать в себе отрезки и точки (схема станции).

2. Захардкодить эти данные в программу.

3. Реализовать алгоритм "заливки" парка.

3.1. Каждый вариант заливки включает в себя набор путей (не участков!), принадлежащих заданному парку и имя парка.

К примеру, “Парк 1”: [Путь А, Путь В], “Парк 2”: [Путь Б, Путь Д], “Парк 3”: [Путь Г, Путь Е]

3.2. Эти варианты также можно захардкодить.

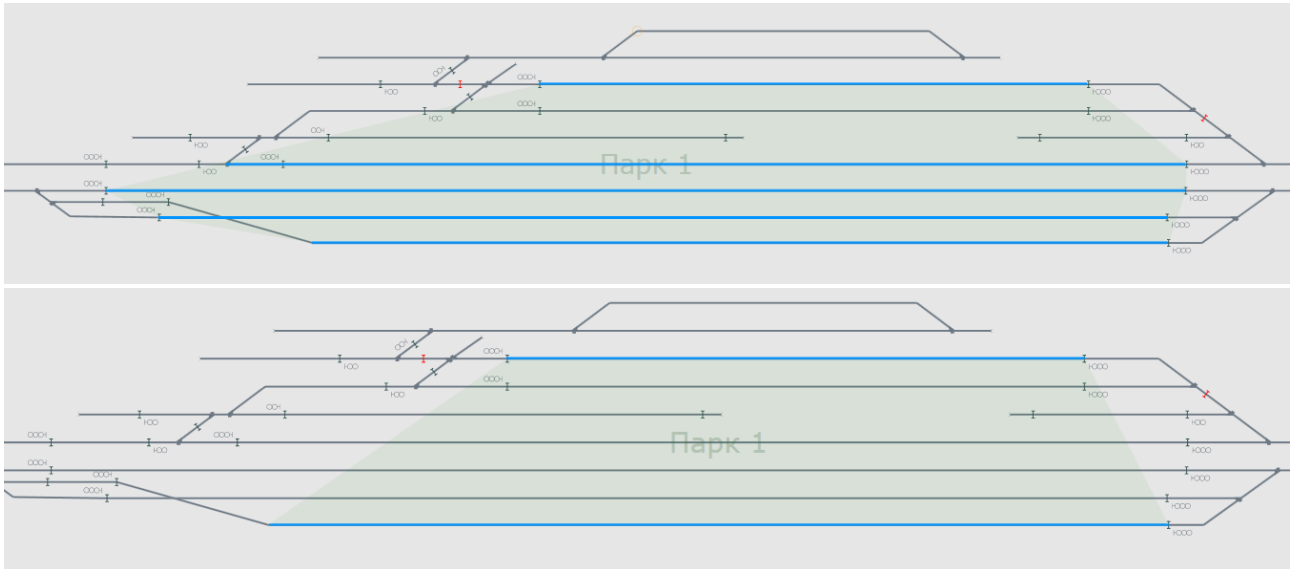
4. Создать консольное приложение, которое выводит в виде списка доступные парки и список вершин, описывающих парк.

Обратите внимание

1. Пути, входящие в парк, могут не иметь общих точек. Путей может быть несколько, а может быть только 1.

2. Результат работы алгоритма заливки – фигура, представляющая собой парк станции.

Примеры заливки (пути парка выделены синим цветом, залитый парк – зеленым):



5. Поиск кратчайшего пути между участками

5.1. Создать консольное приложение, которое выводит в виде списка все участки схемы станции: [Порядковый Номер]. [Имя].

5.2. Добавить возможность выбора номера начального и конечного участка пути.

5.3. Вывести в виде списка участки станции, входящие в кратчайший путь между указанными участками (или сообщить, что пути не существует).

6. Написать базовые unit-тесты для схемы станции, алгоритма заливки и поиска.

7. Залить всё это на github/gitlab или в любой другой публичный репозиторий и прислать нам ссылку.

Важно!

1. Не тратьте время на создание текстового файла с описанием структуры станции, чтобы потом писать отдельный сервис для парсинга. Отдельный класс/сервис/репозиторий (назовите, как хотите), который возвращает захардкоженную схему станции, более чем достаточен.

2. Это не олимпиадное программирование. Результат здесь вторичен. Для нас важно, как вы до него дошли.

3. "Это тестовое задание, но в реальном проекте так делать не буду/нельзя/нужно делать вот так, `todo// сделать хорошо`".

Нам интересно, как вы бы сделали в реальном проекте. Какие бы *технологии* использовали, какие *паттерны* применили, какое *имя* дали переменным/методам/классам. Знаете ли про полиморфизм, инкапсуляцию и пр.

4. “Я сделал, как понял”.

Нет смысла делать задание, которое вы не поняли. Пишите и задавайте вопросы, если считаете, что данные документ не в полной степени описывает поставленные задачи.

5. Кандидатов много и у каждого свой стиль написания кода. Чтобы нам проще было проверить ваш код, **необходимо** скачать готовый `.editorconfig` отсюда: https://github.com/niias-team/editorconfig (обновлен 14.03.2023)

За некоторыми исключениями мы используем общепринятую naming convention для C#. PascalCase - для классов, public методов/свойств и пр, camelCase - для private полей.

8. Используйте TargetFramework .NET 5.0+, не нужно legacy.

9. DI framework – на ваш вкус.