Туториал по использованию редактора графов

Привет! Попробую рассказать, как пользоваться программой и дополнять её своими алгоритмами.

Программа написана на JavaScript с использованием библиотеки Konva.js. В проекте есть несколько файлов, которые трогать не рекомендую, а то всё может поломаться. Эти файлы: main.js, main.css, файл библиотеки Konva – konva.js. В файл main.html нужно будет добавлять несколько строк, а весь вами написанный код алгоритмов будет в файле tasks.js, но об этом поговорим позже.

При возникновении работы программы прошу написать об этом мне - возможно, получится исправить. Думаю, это будет лучший сценарий, потому что мало представляю, что кто-то разберётся в написанном, ибо оно было написано в спешке. К концу написания кода я уже понимал, где архитектурно можно улучшить, но времени не было заново переписывать.

Оглавление

Части программы

В самом верху стоит меню, ниже написано про это подробнее.

Основную часть программы занимает собственно канвас, на котором все графы рисуются.

Справа расположена матрица графа, про это тоже подробнее будет ниже.

Пункты меню

Файл

В Файле есть следующие подпункты:

- открыть/сохранить матрицу смежности
- открыть/сохранить матрицу инцидентности
- открыть/сохранить лист рёбер
- сохранить изображение графа
- выйти из программы

Важно правильно составленные файлы загружать в программу.

При открытии файла матрицы смежности в случае его невалидности сообщается об ошибке. В проекте лежат правильные примеры файлов. Подробнее о них можно узнать в ТЗ к лабам, что тоже прикреплю.

Думаю, остальные моменты и так понятны, двинемся дальше.

Show

Нужен для отображения актуального состояния графа в памяти программы (для дебага). Нужно нажать F12, чтобы открылся консоль.

Кнопки Undo и Redo

Как и следует из названия, кнопки отвечают за отмену последних 10 действий и отмены отмен. Состояния реализованы в виде команд, хранящихся на стэке. Хранятся последние 10 изменений, более старые удаляются.

Отмена работает для всех действий в программе.

Создание вершины

Доя создания вершины по клику нужно нажать на кнопку, после чего клики по канвасу будут создавать вершины графа.

Создание ребра

Для создания ребра, которое соединяет две вершины, нужно нажать на кнопку, после кликнуть на первую и вторую вершину – это вызовет окно, в которое нужно ввести вес ребра.

Алгоритмы

Все алгоритмы будут в этом списке. По нажатию на алгоритм будут окна, в которых выбираются вершины, а после алгоритм выполняется и выводится информация на экране и в отдельном окне. Далее можно в отдельном окне выбрать сохранение.

Об авторе

Здесь хранится информация об авторе программы.

Туториал

Собственно, по нему вы и переходите к документации.

ТЗ

Приложил на всякий случай задания по лабораторной работе.

Матрица смежности

Все действия с графом на экране отображаются на матрице смежности.

Изначальна она пуста, но по ходу добавления ребер и вершин обновляется/расширяется.

Граф можно изменять, обновляя матрицу смежности – поля в ней редактируемы. Эти изменения также отображаются на графе.

Как добавлять алгоритмы

И наконец к вопросу о добавлении алгоритмов в программу.

Как упоминалось ранее единственные файлы, в которых нужно что-то писать – это main.html (добавить кнопку) и tasks.js (добавить алгоритм).

Представление графа в программе

Сейчас граф представляется в памяти программы как:

[вершина1 : [вершина2 : { вес1, … }]]

где [] – это Map, {} – вектор.

Такое представление выбрано было из-за моего удобства. Добавлю по мере необходимости функции, которые из этого представление будут получать необходимые.

tasks.js

В этот файл нужно добавить алгоритм, который принимает два *string* – название начальной вершины и конечной и возвращает вектор пути*.

*Я пока сделал не все алгоритмы, и, возможно, в других алгоритмах вывод другой – это добавится позже.

Собственно, рассмотрим функцию *bfs*:

```
function bfs(start, end) {
    let queue = [], visited = [], d = [Array(nodes.size).fill( value: 0)], p = [Array(nodes.size).fill( value: 0)]
    queue.push(start)
    visited.push(start)
    p[start] = -1

    while(queue.length > 0) {...}

    if (!visited.includes(end)) {
        alert(`От вершины ${start} нет пути до вершины ${end}`)
    } else {
        let path = []

        for (let i = end; i !== -1; i = p[i])
            path.push(i)

        path = path.reverse()
        return path
    }
}
```

Как видно, функция принимает название начальной и конечной вершин и возвращает путь или вызывает alert (окно с ошибкой) при отсутствии пути.

Дальше перейдем к функции *addTask*:

```
function addTask(procedure) {
    let start = parseInt(prompt( message: "Введите начальную вершину",  _default: '0'))
    let end = parseInt(prompt( message: "Введите конечную вершину",  _default: '0'))
    let path = procedure(start, end)

    addHistoryCommand( command: {
        undo: () => {
            returnDefaultPath(path)
        },
        redo: () => {
            paintPath(path)
        }
    }).redo()
}
```

Это как раз функция, которая принимает и вызывает ваш алгоритм поиска. Внутри она вызывает окна, требующие ввести вершины, а потом вызывает ваш алгоритм, после чего рисует на графе путь и предлагает сохранить путь в файл.

main.html

О теперь о том, как добавить кнопку в программу.

В файле main.html есть *div*:

```
38          <div onmouseover="hoverTasks()" class="dropdown">
39              <button class="dropbtn2" onclick="dropDownTasks()">Алгоритмы</button>
40              <div class="dropdown-content" id="dropdownTasks">
41                  <a onclick="addTask(bfs)">2. Поиск в ширину</a>
42              </div>
43          </div>
```

Именно в нем и надо внести изменения:

- создать тэг *a*
- назвать алгоритм (используйте номер лабораторной работы)
- повесить событие *onclick*, которые вызывает функцию *addTask*, принимающий функцию вашего алгоритма (в данном случае *bfs*).

Поздравляю, вы молодцы!!!