Утилита SHIFT

Сёмкин Кирилл, Б05–007 Морозов Кирилл, Б05–007

21 мая 2022 г.

Краткое описание и перспектива

Цель утилиты — облегчить процесс перевода между преподавателями на физтехе, автоматизировав рассчёт перевода для студента и составление нужных документов. Основная трудность, которая может возникнуть при переводах: это возникновение новых конфликтов в расписании из-за изначально желаемой смены, в результате чего нужно искать нового преподавателя по уже другому предмету. Наша программа рассчитана как раз на комфортное и ясное разрешение таких ситуаций. Опишем, как происходит работа с программой:

1. При запуске, сначала студент вводит свою группу
2. По номеру группы программа загружает его расписание и отображает его. Первым делом студент выбирает уроки, которые он не желает менять, если вдруг возникнут конфликты.
3. Далее, выбираем предметы, по которым хотим совершить перевод, и преподавателей, к которым переводимся. Все шаги пользователя отображаются в отображаемом расписании.
4. Далее, происходит рассчёт всех возможных вариантов переводов между преподавателями. После, для пользователя открываются n треков(n — количество изначальных переводов), т.е. n окон, в которых он последовательно выбирает варианты перевода (т.е. новых преподавателей), допустимые его предпочтениями. При выборе нового преподавателя можно посмотреть его характеристики (с известных источников). Все шаги пользователя, опять, имеют своё отображение на расписании.
5. После выбора переводов, отображается новое расписание, а также возможность скачать все документы, нужные для совершения перевода на кафедрах.

Логическое разбиение проекта

Разработка программы естественно разбивается на две части: визуальный интерфейс (**GUI**), и внутреннее взаимодействие программы с базами данных для получение информации, разработка основого алгоритма программы — поиск правильных путей перевода, а также составление pdf-документов (**Kernel**). Опишем задачи каждой части.

Kernel

- взаимодействие с БД и реализация функций, достающих из БД нужную информацию

- реализация основного алгоритма поиска путей перевода
- выгрузка данных о преподавателях из источников
- создание корректных pdf-файлов на перевод
- загрузка расписания студента по его группе

GUI

- графическое представление расписания, и служебных виджетов для работы пользователя с программой
- составление входа для основного алгоритма, и затем использование его выхода для графического отображения путей перевода. Удобное сопровождение пользователя при последовательном выборе преподавателей на перевод (откаты на предыдущие шаги, отображение информации о преподавателе и т.д.)
- предоставить созданные pdf-файлы пользователю

Описание основного алгоритма

Вся сущность программы заключается в алгоритме поиска путей перевода (всё остальное — просто технические детали :)). Считаем, что преподаватели являются вершинами графа и между преподавателями p_1 и p_2 есть ребро, если в расписании возникла коллизия: p_1 занимает место предмета в расписании, который ведёт p_2, т.о. p_2 — новый преподаватель, который может вести этот предмет в новое время.

Вход: расписание, т.е. отображение (day of the week, lesson) $\to$ (subject, teacher); список из n пар (subject, new teacher) — список изначальных желаемых переводов; список из k пар (subject, neverchanging teacher) — предметы, которые мы не хотим менять в расписании.

Выход: на выходе имеем квадратную матрицу смежности (матрица цветов): каждая ячейка соответствует ребру между учителями. В каждой ячейке храним множество цветов, в которые покрашено данное ребро. Каждый цвет — это номер корректного пути, найденного в графе.

Алгоритм: начинаем с первого конфликта, по id учителя узнаём, какое время он занимает. Если эта ячейка расписания свободна, то переходим к следующему конфликту. Иначе, смотрим предмет, по которому происходим коллизия, и находим всех преподавателей, которые ведут этот предмет. Переходим рекурсивно во всех этих преподавателей (если преподаватель работет в уже заблокированную ячейку расписания, то выходим; то же, если текущий преподаватель — наш текущий, т.к. как раз его мы и вынуждены менять). Т.о. имеем поиск в глубину по графу с поиском всех путей.

По ходу поиска в глубину храним лес деревьев поиска в глубину для каждого из n конфликтов. Как только все n конфликтов оказались разрешены, то проходимся по этому лесу и красим все ребра в текущий цвет (цвет — это просто число), т.о. пометили в графе один из возможных корректных путей. Покраска — это добавление в матрицу цветов в нужные ячейки цвет. Далее, алгоритм рекурсивно ищет все пути и каждый найдённый красит в свой цвет.