Автономная некоммерческая образовательная организация
“Физтех - лицей” имени П.Л. Капицы

XX научно-практическая конференция
«Старт в инновации»

Приложение для автоматизации равномерного распределения образовательной нагрузки ученика

Авторы работы:
Иван Белецкий,
Александр Сушин
9 “И” класс

Руководитель:
Мерзляков Алексей Владимирович

Московская область, г. Долгопрудный, 2021 г.

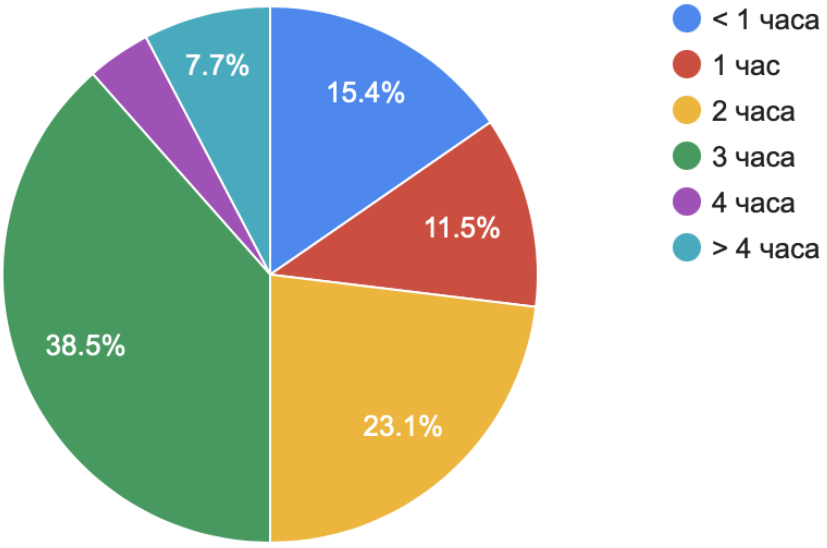
Оглавление

Введение

Нередко ученики самых различных учебных заведений могут столкнуться с неравномерным распределением домашней работы в различные дни. Несмотря на старание преподавателей, учебные задолженности, прокрастинация и иные дополнительные занятия могут спровоцировать формирование большого количества заданий в один день. Нашу команду заинтересовала данная проблема и мы решили найти ее решение.

Социологический опрос учеников лицея

Для понимания важности данной проблемы нужно задаться вопросом: “Сколько в среднем ученики ФТЛ выполняют домашнюю работу?”
Проанализировав данные, полученные в результате социологического опроса, мы выяснили, что в среднем на выполнение домашнего задания обучающиеся тратят:



Легко заметить, что значительная часть учеников в среднем выполняет домашнее задание более трех часов (50 %). Это может говорить о высокой загруженности учеников, из чего следует, что даже единичный случай неравномерного распределения может привести к излишним затратам времени и половины учеников.
Исходя из полученных данных, можно прийти к выводу, что большая часть учеников потенциально могут испытывать проблемы со сдачей домашней работы вовремя, из-за чего могут появиться задолженности и неудовлетворительные оценки, увеличивающие вероятность повторения подобных событий.

Актуальность работы

Озвучив проблему выше, легко сделать вывод, что куда правильнее выполнить часть работы заранее, однако, проще предоставить выбор по распределению дел специальной программе, нежели делать это единолично. Нам не удалось найти реализацию нашей идеи в сети Интернет, именно поэтому мы решили воплотить ее в жизнь самостоятельно. Безусловно, существуют лицеисты, которые по разным причинам либо игнорируют задания, данные учителями, либо доверяют собственным суждениям и импровизации больше, чем придуманному нами алгоритму, тем не менее, нельзя отрицать, что все - таки человеческая лень стала мотивацией к созданию данной работы. Как сказала Н.А. Тэффи: “Лень - двигатель прогресса”. В XXI веке, когда технологии надежнее чем люди в выполнении монотонной работы, данная программа способна помочь школьникам с оптимизированным распределением нагрузки по дням для получения наилучшего результата. Будучи увлеченными своими хобби, мы способны напрочь забыть о домашней работе, что постепенно отражается на успеваемости, а также на нашей самооценке. Наше приложение рассчитано на то, чтобы облегчить управление тайм - менеджментом и, возможно, уберечь от ненужного стресса.

Принцип работы программы прост: она составляет комфортный учебный график и у пользователя появляется время на важные для него занятия: подготовка к олимпиаде, помощь близким, необходимый отдых. Развиваясь в спокойном темпе, есть высокая вероятность раскрыть свой потенциал и потратить оставшееся силы продуктивно.

Цели и задачи

1. Придумать алгоритм распределения домашней работы по дням.
2. Написать на C++ внутреннюю часть программы.
3. Написать внешнюю оболочку программы с помощью библиотеки SFML.
4. Провести тестирование программы, с помощью различных методов.
5. После доработки части интерфейсов и оптимизации алгоритма, выпустить приложение для свободного пользования

Описание алгоритма

В начале работы программы пользователь указывает приоритет 4 параметров домашнего задания: *TIME* - прогнозируемое время, затрачиваемое на выполнение, *INTEREST* - интерес работы, *DEADLINE* - предельный срок сдачи работы, *COMPLEXITY* - оцениваемая пользователем сложность работы, т.е. сколько часов в день он готов трудиться. Создается структура Prior - предмет с названием работы и параметрами, указанными выше. Предметы добавляются в *multiset*, находящийся в стандартной библиотеке в *C++*. На основе приоритетов создается компаратор, после происходит упорядочивание элементов с данным компаратором. Набирается набор предметов на i-й день следующим образом в функции *GetHomeWork()*:

```
std::vector<std::pair<std::string, std::string>> GetHomeWork() {
    if(mas.size() == 0) {
        return {};
    }
    int timer = 0;
    std::vector<std::pair<std::string, std::string>> task;
    while(timer < timePerDay) {
        task.push_back({mas.begin()->type, mas.begin()->name});
        timer += mas.begin()->time;
        mas.erase(mas.begin());
    }
    return task;
}
```

Пока счетчик времени работы не превысит заданный параметр, в массив, отвечающий за список работы, добавляется название задания и удаляется из исходного *multiset'a*.
При добавлении работы:

```
void AddEvent(Prior elem) {
    mas.insert(elem);
}
```

В *multiset* добавляется новый предмет.
N - количество элементов в массиве.
Итоговая асимптотика работы алгоритма:

1. Добавление элемента: $O(logN)$, добавление элемента в сет.
2. Получение домашнего задания на день: $O(NlogN)$. В худшем случае будет удалены все предметы (N). Удаление одного предмета $O(logN)$.
3. M - количество запросов получить домашнее задание. T - количество запросов добавить домашнее задание.

Суммарная асимптотика: $O(TlogN + M \times NlogN)$.

Доказательство корректности работы

Предположим, что число запросов на добавление предметов равняется нулю, следовательно, мощность множества домашней работы может только уменьшаться. При изменении приоритета и времени работы доказательство аналогично, поэтому все значения константы.

Докажем, что алгоритм распределит домашнее задание оптимально по дням с помощью метода математической индукции.
Индукционное предположение: *в i-й день будет выбран оптимальный набор.*

База: *рассмотрим первый день.*

Лемма 1: *набор оптимален по времени.*
Действительно, если убрать наименее приоритетный предмет из нашего набора, где суммарное время работы < времени, которое было задано пользователем (по алгоритму набора домашнего задания на день), остаток времени стоит заполнить работой, нежели оставить его пустым.

Лемма 2: *набор оптимален по важности работы*.
Так как рассматривается первый день, то не существует предметов в множестве с большим приоритетом, чем в наборе (вследствие, что массив отсортирован); ∀объект ∉набор, имеет меньший приоритет, чем ∀ входящий. Следовательно, при добавлении любого элемента день будет перегружен по времени, а замена любого элемента в наборе на элемент вне набора уменьшит приоритет, значит, набор оптимален по важности работы.
Исходя из данных двух лемм, набор будет оптимален.

Индукционный переход:
Рассмотрим i-й день: исходя из индукционного предположения, в *1...i-1* день были выбраны оптимальные наборы, следовательно, из множества были удалены все объекты, входящие в данные наборы. В новый набор, очевидно, не могут войти удаленные элементы, соответственно, данная ситуация аналогична базе. Необходимо повторить рассуждения, приведенные выше.
Пусть число добавлений предметов > 0; пусть в i-й день было добавлено новое задание, тогда ясно, что мы чисто физически не могли его выполнить раньше, чем срок добавления, поэтому все наборы, выбранные ранее, не влияют на новые наборы, поэтому можно считать, что после добавления множество снова может только уменьшаться, повторив рассуждения выше, будут выбраны оптимальные наборы домашнего задания.

Следствие:

При малом времени на выполнение работы, существуют случаи, когда ученик будет не успевать все делать в срок, но по доказанному распределение будет оптимально, а потому выполнить задание будет невозможно.

Описание визуализации

Первоначальной задачей была разработка удобного и эстетичного дизайна, способного доступно передать суть приложения. Основной каркас визуализации построен на выделении главных частей приложения, используя основы минимализма. С помощью приложения *Photoshop* были созданы ранние прототипы оформления, позднее переработанные и воплощенные в жизнь с помощью библиотеки *SFML*.

На начальном этапе разработки визуализации наша команда задалась вопросом о выборе средства для исполнения пользовательской части проекта. После недолгих рассуждений нами было принято решение использовать библиотеку *SFML* из-за ее гибкости в настройках написанного интерфейса. Далее мы создали средства взаимодействия пользователя с приложением в виде различных кнопок и полей. С помощью создания дополнительного окна в приложение была добавлена возможность создания пользователем заданий, которые впоследствии и распределяет программа. Меню, находящееся в упомянутом окне, содержит в себе всю информацию, которая влияет на дальнейшую упорядочивание элементов в *multiset'e*. Затем был реализован вывод данных получаемых от программной части приложения, тем самым было закончено создание основной части проекта.

После окончания работ со "скелетом" приложения, было решено расширить набор возможностей пользователя, добавив кастомизацию и дополнительные возможности взаимодействия с интерфейсом. Идея кастомизации приложения помогает клиенту самому выбрать удобные для себя дизайнерские решения, тем самым программа может удовлетворить большую часть потребителей. Пример кастомизации - возможность изменить основную цветовую гамму со светлой, на темную.

Также часть дополнений была связана с удобством взаимодействия программы и пользователя. Так при нажатии на колонку, обозначающую одно из заданий, эта колонка расширяется, позволяя просмотреть подробности домашнего задания и сроков его выполнения. Также в этом разделе есть функция более раннего завершения конкретного задания. После выполнения цель переходит в раздел выполненных, и помечается серым цветом.

С внешним видом приложения можно ознакомиться в приложении 1 данной работы.

В дальнейшем целью нашей команды будет усовершенствование данного интерфейса, посредством добавления продвинутых анимаций, всплывающих окон, уведомлений и других возможностей, которые не были реализованы в данной версии приложения.

Тестирование проекта

Было решено разделить процесс тестирования на две части:

1. Юнит - тестирование
2. Стресс - тестирование

Юнит - тестирование

***Модульное тестирование**, или **юнит-тестирование** — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.*

1. При написании кода возникла проблема с выбором структуры данных. При использовании, например, массива, сложность алгоритма становилась $O(N^2)$ на распределение домашнего задания в функции *GetHomeWork().* Поэтому было решено использовать встроенную структуру данных *multiset.* Из - за специфики структур домашнего задания появилась необходимость писать компаратор для *multiset*. Для тестирования было выбрано домашнее задание, выданное нам на неделю в период: с 15 февраля по 20 февраля. После дебага было решено следовать результатам программы и делать домашнее задание именно в этом порядке.
2. После было проверена работа следующих функций: GetPriority(), AddTimePerDay(), AddEvent().
3. Тестирование пользовательской части программы.
4. Удостоверившись, что все функции работают верно, было решено перейти к завершающей части.

Стресс - тестирование

***Стресс-тестирование** — один из видов тестирования программного обеспечения, которое оценивает надежность и устойчивость системы в условиях превышения пределов нормального функционирования. Стресс-тестирование особенно необходимо для «критически важного» ПО, однако также используется и для остального ПО. Обычно стресс-тестирование лучше обнаруживает устойчивость, доступность и обработку исключений системой под большой нагрузкой, чем то, что считается корректным поведением в нормальных условиях.*

Для тестирования использовалась библиотека *random* и генератор псевдослучайных чисел *Вихрь Мерсенна.* Генерировался случайный список домашних заданий с случайным распределением коэффициентов. Ученику выдавался список заданий на дом, который проверялся функцией *Check().* Данная функция осуществляла проверку, успевал ли ученик

выполнять их вовремя, была ли нагрузка равномерной, а после чего выводила информацию об этом в консоль.

Вывод

Перед нами стояла цель придумать алгоритм ранжирования домашнего задания и реализовать его с помощью библиотеки *C++ SFML*. После аналитики работы алгоритма и написания кода был получен желаемый результат в виде программы и жадного алгоритма, способных автоматизировать равномерное распределение образовательной нагрузки ученика. При создании мы прошли долгий, но интересный путь, наполненный проблемами и их решениями.

Дальнейшая поддержка проекта

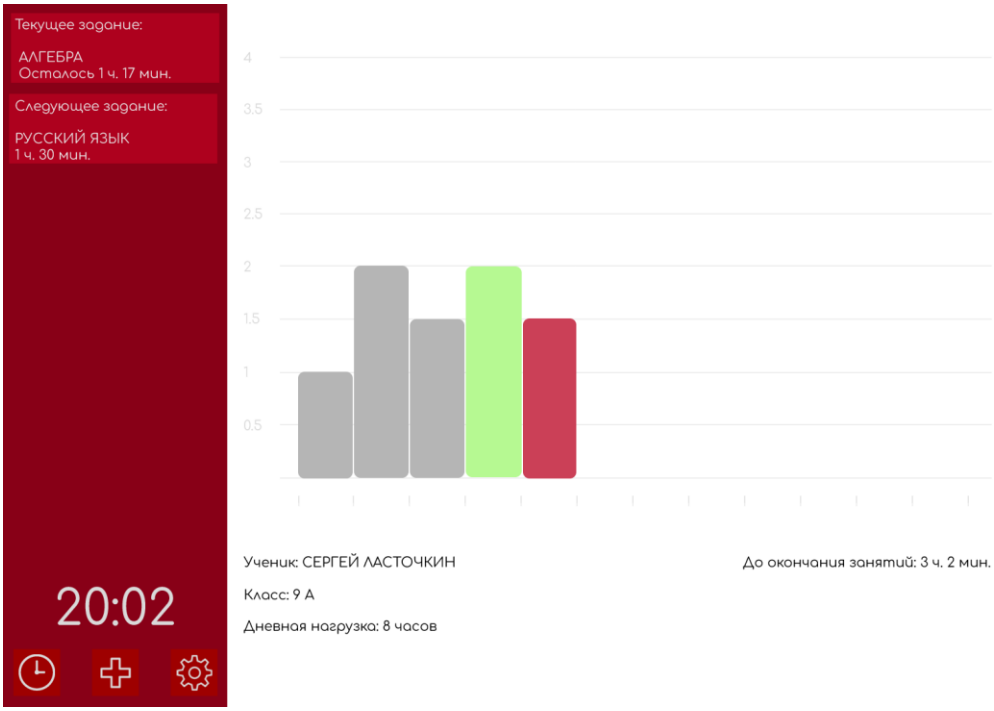
В будущем планируется расширить возможности программы: добавить возможность синхронизации с системой *ЭлЖур*, ускорить скорость работы алгоритма с помощью сложных структур данных, например, *приоритетной кучи Бродала-Окасаки,* создать интерфейс включающий в себя анимации, всплывающие окна, уведомления и иные методы продвинутого взаимодействия с интерфейсом.

Литература

1. https://habr.com/ru/post/169381/
2. https://ru.wikipedia.org/wiki/Стресс-тестирование_программного_обеспечения

Приложения

Приложение 1:


Текущее задание:
АЛГЕБРА
Осталось 1 ч. 17 мин.
Следующее задание:
РУССКИЙ ЯЗЫК
1 ч. 30 мин.
20:02
4
3.5
3
2.5
2
1.5
1
0.5
Ученик: СЕРГЕЙ ЛАСТОЧКИН
Класс: 9 А
Дневная нагрузка: 8 часов
До окончания занятий: 3 ч. 2 мин.