МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

"Нижегородский государственный университет им. Н.И. Лобачевского"

Институт информационных технологий, математики и механики

Кафедра информатики и автоматизации научных исследований

Направление подготовки: «Прикладная информатика»

КУРСОВАЯ РАБОТА

Тема:

«Система построения расписания с User Interface

на базе фронтального алгоритма»

**Выполнил:** студент 3 курса, гр. 381507-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А. Суворов

(подпись)

**Научный руководитель:**

Старший научный сотрудник НИЛ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Филимонов

(подпись)

Нижний Новгород

2018

**Содержание:**

1. Введение
2. Формальная постановка задачи
   1. Исходные параметры
   2. Математическая модель
   3. Критерий задачи
   4. Комбинаторная и математическая сложность
   5. Индивидуальная постановка задачи
3. Алгоритм решения задачи
   1. Общая идея фронтального алгоритма
   2. Перестановочные процедуры преобразования фронта работ в вектор.
   3. Процедуры «построители»
4. Программная реализация
   1. Функциональные возможности программы
   2. Основные формы, классы и функции программы
   3. Эксперимент
5. Заключение
6. Список литературы
7. Приложение
8. **Введение**

Во все времена, управление бизнес проектами или командами людей являлось очень сложным и ответственным занятием. Руководители вынуждены держать в голове большой объем информации, делегировать обязанности и контролировать их выполнение. Как известно, время является одним из главных ресурсов в информационную эпоху, и потому очень важно грамотно его использовать. Для этого необходимо выбирать наиболее приоритетные задачи и оптимально планировать их выполнение. Однако, с учетом большого количества постоянно изменяемой информации, становится практически невозможным ручное планирование и потому возникает необходимость в автоматизировании этого процесса.

Перед каждым руководителем рано или поздно встает вопрос как лучше распределить обязанности внутри команды, чтобы общая эффективность организации была максимальна? Иногда проектов настолько много, что одному человеку очень сложно грамотно построить работу. Нужно держать в голове очень большое количество информации, строить громадные таблицы расписаний, следить за выполнением всех этапов проекта, а главное, изменять их и их порядок в течение работы, подстраиваясь под внешние факторы. Именно поэтому очень удобным инструментом было бы приложение с дружественным пользовательским интерфейсом, которое могло бы делать это вместо человека.

Рассмотрим план работы студенческого отряда, в котором состоит определенное количество людей, и каждый из них может выполнять поставленные перед ним задачи. Известны все проекты, планируемые на расчётный период, каждое из них тщательно продуманно и разделено на малые подзадачи. Все полученные работы взаимозависимы, причем ни одна работа не может быть выполнена пока не закончены все предшествующие ей этапы. Таким образом подготовку всех проектов можно описать каноническими сетевыми моделями, которые представляют собой взвешенный ориентированный граф без петель и контуров. Вершинами этого графа являются задачи, которым в соответствие ставят определенный набор характеристик: для всех работ известно время выполнения каждым сотрудником и время максимально раннего начала выполнения (по умолчанию 0 – начало проектируемого отрезка времени), а для некоторых, может указываться крайнее время окончания выполнения. В случае задержки какой-либо работы отряд несет потери в рейтинге, в зависимости от важности конкретного проекта.

Таким образом перед нами встает задача сетевого планирования. Необходимо написать программу, которая сможет построить такое расписание, при котором отряд минимизирует свои потери в рейтинге. При этом приложение должно содержать удобный для пользователей интерфейс, который будет включать себя возможность редактирования списка проектов и включенных в них работ, возможность просмотра расписание для отдельных исполнителей на выбранное время, а также общего представления занятости исполнителей.

1. **Формальная постановка задачи**
   1. **Исходные параметры:**

* 1. **Математическая модель**

**Варьируемые параметры:**

**Ограничения математической модели:**

Естественные ограничения на варьируемые параметры:

Ограничение на время выполнения работ, исходя из канонической сетевой модели:

Ограничение на время выполнения исполнителем работ без перерывов:

Ограничение, связанное с выполнением исполнителем не более одной работы в каждый момент времени:

Ограничение, связанное с выполнением работ не более чем одним исполнителем:

Ограничение на начало выполнения работы:

* 1. **Критерий задачи:**
  2. **Комбинаторная и математическая сложность**

Поставленная задача относится к классу NP-трудных задач, т.е. для ее решения не существует эффективных алгоритмов определения оптимального решения. В случае полного перебора каждой работе нужно выбрать исполнителя. Поэтому число всевозможных вариантов будет равно размещению с повторением из m исполнителей по n работ ().

* 1. **Индивидуальные постановки задач**

T={1, …, 20}

P={1,2}

J(1) = {1,2,3,8)

J(2)={1,4,5,6,7,8}

J={1, …, 8}

I={1,2}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | A(j) |  |  |  |
| 1 | Ø | 2 | 1 |  |
| 2 | {1} | 2 | 1 |  |
| 3 | {2} | 2 | 1 | 8 |
| 4 | {1} | 2 | 1 |  |
| 5 | {4} | 2 | 8 |  |
| 6 | {5} | 2 | 1 |  |
| 7 | {4} | 2 | 12 |  |
| 8 | {3,6,7} | 2 | 1 | 20 |

C=

|  |  |  |
| --- | --- | --- |
|  | 1 | 2 |
| 1 | 3 | 3 |
| 2 | 2 | 3 |
| 3 | 3 | 2 |
| 4 | 4 | 3 |
| 5 | 4 | 2 |
| 6 | 1 | 3 |
| 7 | 5 | 6 |
| 8 | 3 | 3 |

X=(0,3,5,3,8,10,12,17)

Y=(3,5,8,6,10,11,17,20)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

1. **Алгоритм решения задачи**
   1. **Общая идея алгоритма**

Для решения поставленной задачи был выбран фронтальный подход, являющийся представителем идеологии «жадных» алгоритмов. Под жадными алгоритмами будем понимать алгоритмы, в которых включенная в строящееся расписание работа не может быть исключена из него на последующих шагах построения. Жадные алгоритмы применительно к рассматриваемой задаче являются эффективными алгоритмами (по вычислительной сложности), которые не гарантируют получения точного решения, но находят допустимое решения, являющиеся приближенным к оптимальному.

Выбранный подход решения основывается на построении в каждый такт времени «фронта работ». Пусть t – произвольный такт планирования, . Назовем «фронтом работ» множество F(t) – множество операций, любая из которых может выполняться, начиная с такта t, . Для выбора очередной операции из фронта работ перейдем от множества F(t) к вектору , компоненты которого и будут определять порядок выполнения операций. После установления порядка выполнения выбранная из фронта работ очередная работа включается в строящееся расписание с помощью определенной процедуры «построитель», причем для ее выполнения используются все доступные к данному моменту времени ресурсы в максимально возможном объеме. Для установления порядка могут быть применены специальные перестановочные процедуры.

Таким образом схема фронтального алгоритма состоит из следующих шагов:

* в заданный такт времени t задается фронт работ - множество операций, любая из которых может начать выполняться (нет невыполненных предшествующих работ и самое раннее время возможного старта «пройдено» )
* фронт работ преобразуется в вектор, используя перестановочные процедуры;
* операции в установленном порядке из вектора работ включаются в строящееся расписание, с помощью определенной процедуры «построитель»
* переход к следующему такту планирования
  1. **Перестановочные процедуры преобразования фронта работ в вектор.**

Для формирования вектора операций из фронта работ используем стратегию сортировки по убыванию штрафа на единицу времени выполнения. Рассмотрим алгоритм выбранного метода:

1. Сформируем V(t) - множество исполнителей, свободных в рассматриваемый момент времени t, а так же создадим копию фронта работ F`(t)=F(t) .
2. Если свободных исполнителей нет, то есть , переходим на шаг 6.
3. Найдем - минимальное значение временных затрат, необходимых на выполнения произвольной работы из F`(t) любым исполнителем из V(t).
4. Добавим операцию с номером h в формируемый вектор работ, и исключим ее из используемого множества работ F`(t), а также исключим исполнителя с номером g из множества свободных исполнителей V(t).
5. Если еще остались доступные работы и исполнители, т.е. , то вернемся к шагу 3.
6. Если еще остались работы, готовые к исполнению (), добавляем их по очереди к строящемуся вектору. (Эти работы не начнут свое выполнение в рассматриваемый момент времени t, так как нет свободных исполнителей (), поэтому их порядок не важен. Однако, добавим их для сохранения размерности вектора)
   1. **Процедуры «построители»**

Данная процедура необходима для формирования расписания из упорядоченного вектора работ. В рассматриваемой задаче имеется m различных исполнителей, поэтому, в общем случае, соотношение между первой работой из вектора и множеством возможных исполнителей не однозначно. Поэтому при построении расписания будем нагружать такого исполнителя, у которого временные затраты на выполнение этой работы минимальны.

1. **Программная реализация**
   1. **Функциональные возможности программы**
2. Программа предоставляет пользователю возможность создать проект, включающий в себя работы, исполнителей, матрицу временных затрат на выполнение и другие характеристики.
3. Созданный проект можно сохранить как Excel-файл, а также возможно загружать или изменить ранее сохраненные проекты.
4. Проверка корректности вводимых данных.
5. Построение допустимого расписания и вывод его в виде графика загруженности исполнителей
6. Подсчет величины штрафа за просрочку выполнения работ.
   1. **Основные формы, классы и функции программы**
      1. **Общая информация о программе**

Для решения поставленной задачи был выбран язык объектно-ориентированного программирования C# на платформе .Net Framework 4.5.2.

* + 1. **Формы программы**

Для работы пользователя с программой было создано 4 формы. Ознакомиться с их видом можно в Приложении №1.

1 форма является основной. Ее задача заключается в общем представлении выбранного проекта и его составляющих: список работ, множество исполнителей и их время выполнения работ. Кроме того, пользователю предлагается открыть существующий, создать новый или изменить текущий проект. Так же здесь выводится построенное для этого проекта расписание.

2 форма позволяет пользователю создать новый проект, заполнив необходимые поля, а также предоставляется полная информация по имеющимся работам и исполнителям.

3 и 4 форма позволяют пользователю создавать новые или изменять текущие работы и исполнителей соответственно.

* + 1. **Структура данных**

Были сконструированы 3 класса объектов: Project, Job и Worker. Данные классы содержат свойства реальных объектов и временные характеристике, необходимые для решения поставленной задачи. Так, например, работа в нашей задаче включает такие характеристики как, название, время раннего начала и позднего окончания выполнения, количество и множество предшествующих работ, штраф за опоздание и другие.

Более подробно ознакомиться со структурой классов можно в Приложении№2.

* + 1. **Основные функции**

1. Функция формирования фронта работ в момент времени time

|  |
| --- |
| public Job[] ReadyJobs(int time) |

1. Функция построителя расписания проекта за период time.

|  |
| --- |
| public static int[,] CreateSchedule(Project project, int time) |

1. Функция чтения сохраненного проекта их файла Excel

|  |
| --- |
| public static Project OpenProject(string FileName) |

* + 1. **Хранение входной информации**

Для хранения созданных проектов были выбраны Excel-таблицы. Это обусловлено тем, что Excel имеет богатый инструмент для работы с данными, в том числе построение различных видов диаграмм. Кроме того, при просмотре созданного файла проекта информация достаточна понятна и удобна для пользователей. Для использования таблиц, в программу добавлена работа с COM(Component Object Model) - объектами, используя библиотеку Microsoft.Office.Interop.Excel.

* 1. **Эксперимент**

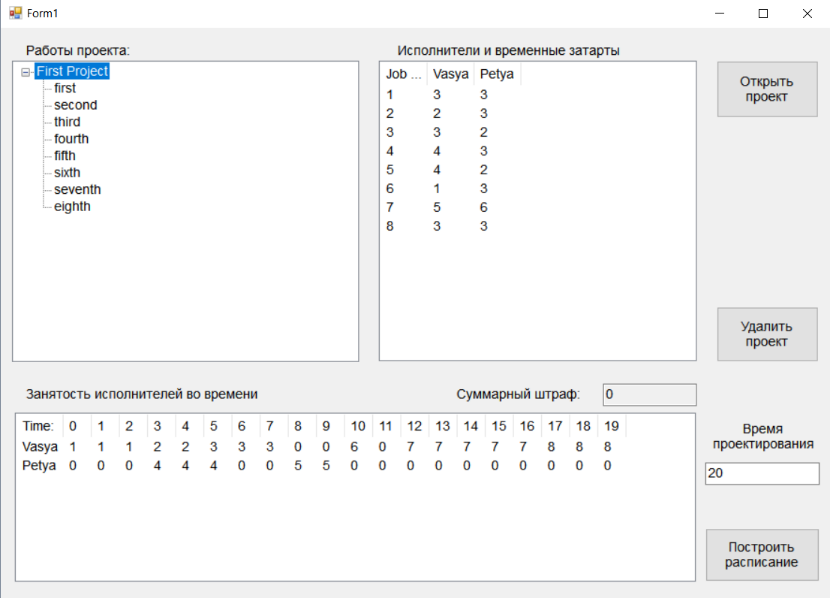
Было проведено ряд различных экспериментов. Рассмотрим 2 элементарных из них.

1. Входные данные: в проект входят 8 работ и 2 исполнителей

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер работы | Название работы | время раннего начала | время позднего окончания | Штраф на 1 такт за позднее выполнение | Количество предшественников | Предшеству-ющие работы | | |
| 1 | first |  |  | 2 | 0 |  | | |
| 2 | second |  |  | 2 | 1 | 1 | | |
| 3 | third |  | 8 | 2 | 1 | 2 | | |
| 4 | fourth |  |  | 2 | 1 | 1 | | |
| 5 | fifth | 8 |  | 2 | 1 | 4 | | |
| 6 | sixth |  |  | 2 | 1 | 5 | | |
| 7 | seventh | 12 |  | 2 | 1 | 4 | | |
| 8 | eighth |  | 20 | 2 | 3 | 3 | 6 | 7 |

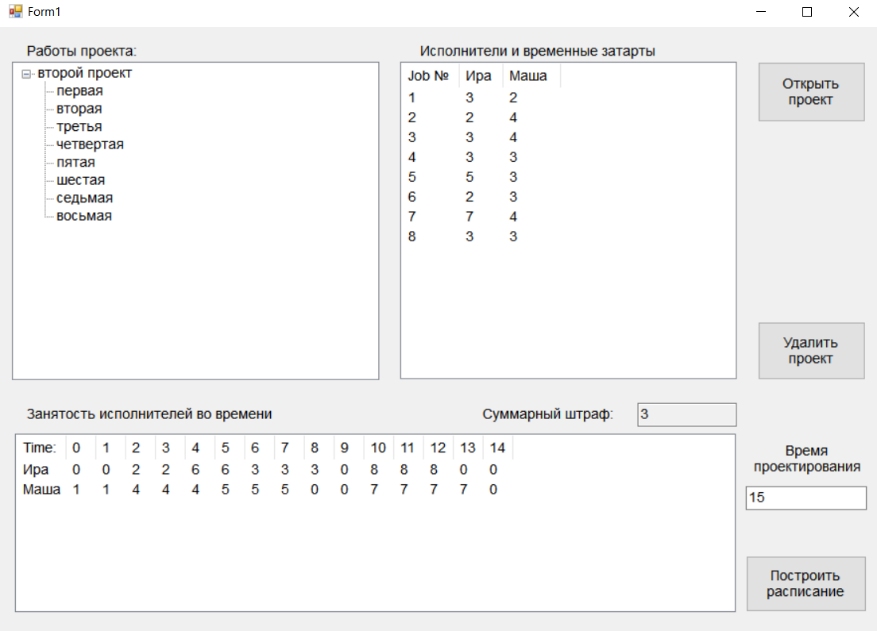
Время выполнения работ исполнителями:

|  |  |  |
| --- | --- | --- |
| Номер работы | Вася | Петя |
| 1 | 3 | 3 |
| 2 | 2 | 3 |
| 3 | 3 | 2 |
| 4 | 4 | 3 |
| 5 | 4 | 2 |
| 6 | 1 | 3 |
| 7 | 5 | 6 |
| 8 | 3 | 3 |

Полученные результаты:

1. Входные данные: 8 работ, 2 исполнителей

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер работы | Название работы | время раннего начала | время позднего окончания | Штраф на 1 такт за позднее выполнение | Количество предшественников | Предшествующие работы | | | |
| 1 | первая | 0 | 3 | 2 | 0 |  | | | |
| 2 | вторая | 0 | 5 | 8 | 1 | 1 | | | |
| 3 | третья | 0 | 5 | 1 | 1 | 1 | | | |
| 4 | четвертая | 0 | 5 | 10 | 1 | 1 | | | |
| 5 | пятая | 0 | 8 | 3 | 1 | 1 | | | |
| 6 | шестая | 0 | 10 | 7 | 1 | 1 | | | |
| 7 | седьмая | 10 | 15 | 2 | 4 | 2 | 3 | 4 | 5 |
| 8 | восьмая | 10 | 20 | 2 | 2 | 3 | 6 | 7 | |

Время выполнения работ исполнителями:

|  |  |  |
| --- | --- | --- |
| Номер работы | Ира | Маша |
| 1 | 3 | 2 |
| 2 | 2 | 4 |
| 3 | 3 | 4 |
| 4 | 3 | 3 |
| 5 | 5 | 3 |
| 6 | 2 | 3 |
| 7 | 7 | 4 |
| 8 | 3 | 3 |

Полученные результаты:

По полученным результатам мы можем сделать некоторые выводы о работе программы:

1. Программа работает, на выход выдает допустимое решение. В некоторых случаях оно является оптимальным, как в 1ом эксперименте.
2. Но во 2 эксперименте, не трудно видеть, что можно получить более хорошее решение поменяв порядок выполнения 3 и 6 работы, что приведет к уменьшению суммарного штрафа.

Таким образом, мы можем сделать вывод, что программа работает. Однако, есть потенциал для улучшения получаемого расписания. Для этого стоит уделить внимание перестановочным процедурам преобразования фронта работ в упорядоченное множество.

1. **Заключение**

В рамках данной работы получилось реализовать систему построения расписания с User Interface для OC Windows на основе фронтального алгоритма. Данное приложение способно получить допустимое решение и предоставить график занятости каждого исполнителя.

В течение данного проекта было получено большое количество опыта и знаний в различных областях программирования, проектирования и алгоритмизации. Были изучены метод фронтального построения расписания их виды и модификации, получен опыт в работе с COM-объектами на примере работы с таблицами Excel.

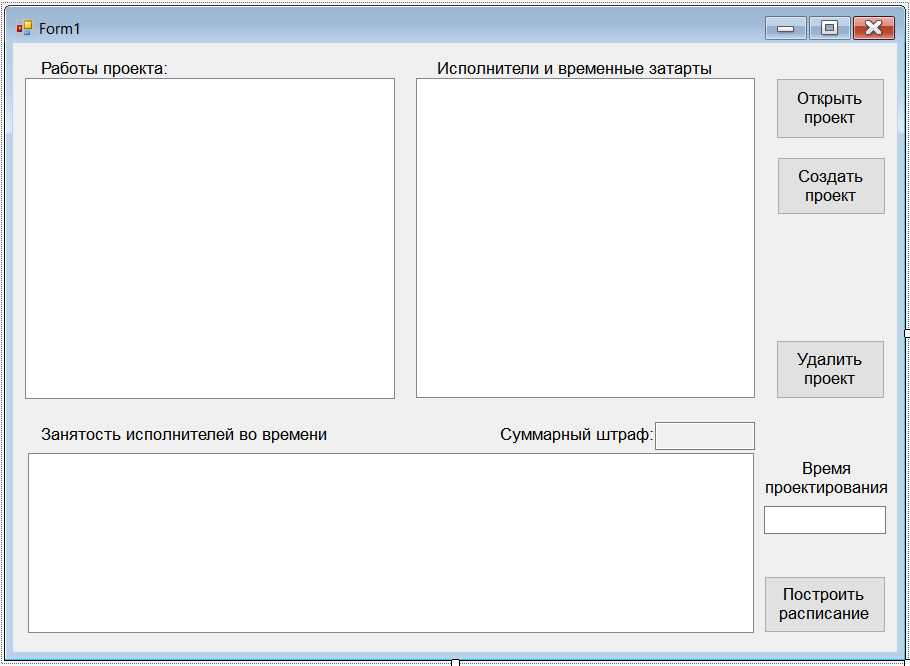
Однако на этом данная тема не исчерпана. Есть много возможностей улучшить полученный результат и усовершенствовать полученное приложение. Уже сейчас есть представление ориентировочных шагов дальнейшей работы, рассмотрим несколько из них:

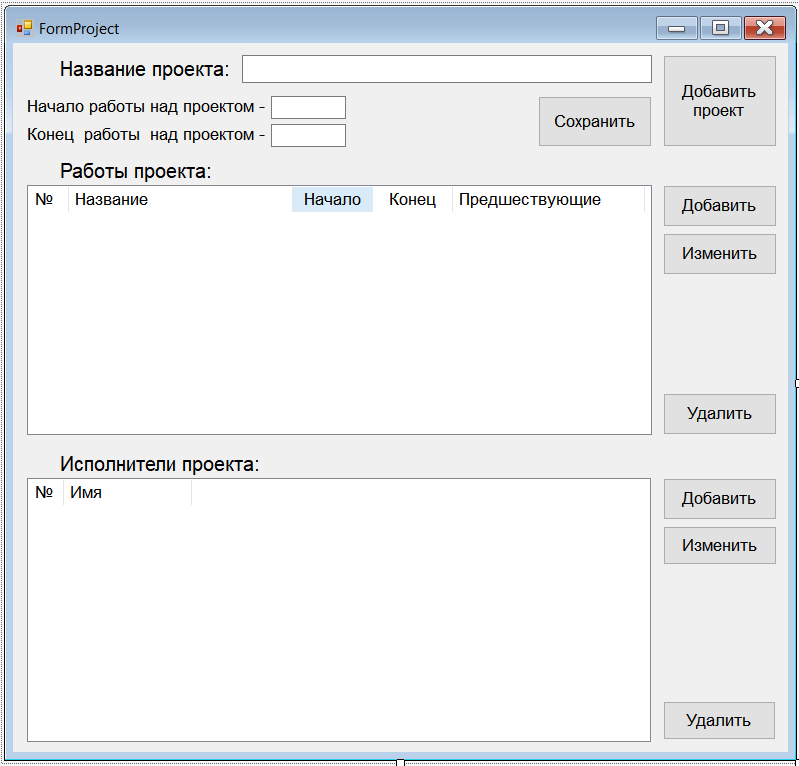
* В первую очередь необходимо уделить внимание получаемому расписанию: усовершенствовать алгоритм путем добавления новых перестановочных процедур и временных характеристик для множества работ.
* Во-вторых, добавить возможность построения расписания для нескольких проектов. При этом множество исполнителей придется выносить за рамки одного проекта, что вызовет некоторые трудности при заполнении матрицы временных затрат.
* В-третьих, использовать более удобные и надёжные методы хранения входной информации, например, базы данных.
* В-четвертых, для большей автоматизации и удобства добавить разделение работ на разные типы со своей сложностью. В свою очередь исполнители смогут выполнять определенный тип работ с определенной эффективностью. Эти новые характеристики будут формировать матрицу временных ресурсов, что избавит пользователя от большого объема работы.
* Ну и конечно, улучшить интерфейс программы, сделать его более дружелюбным к пользователю, сведя заполнение входных данных к минимуму.

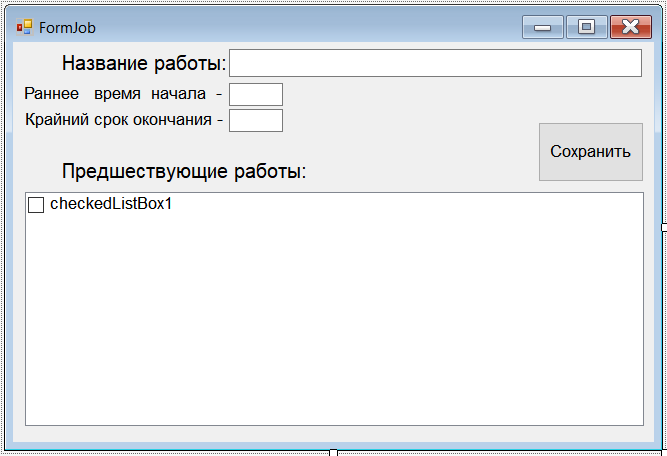
1. **Список литературы**
2. Прилуцкий М.Х., Кумагина Е.А. Задачи распределения разнородных ресурсов в сетевых канонических структурах // Перспективные информационные технологии и интеллектуальные системы. 2000. № 4. С. 46–52.
3. Молчанов В.В. Интернет книга «Практика программирования на С# для Windows и Web в Microsoft Visual Studio» http://wladm.narod.ru/C\_Sharp/
4. М.Х.Прилуцкий, С.Е.Власов. Многостадийные задачи теории расписаний с альтернативными вариантами выполнения работ
5. Кожухов Н.И. Алгоритм решения задачи оптимального распределения работ в сетевых канонических структурах // Лесотехнический журнал 4/2014 УДК 004.02
6. **Приложение**

**Приложение№1. Формы**

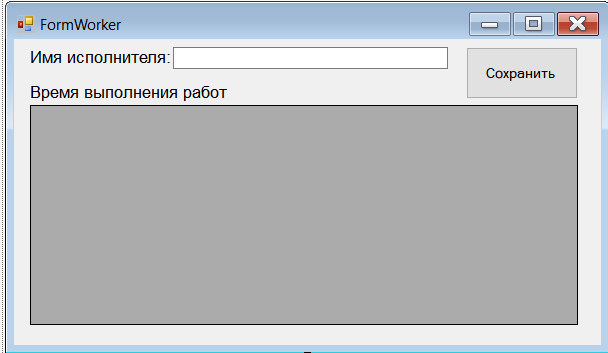
1. Главная форма



1. Форма создания проекта
2. Форма создания работы



1. Форма создания исполнителей.



**Приложение №2. Пользовательские классы программы.**

1. class Job

{

private int id; // ID работы

private string name; // Название работы

private int early, late; // Ограничения на самое раннее начало и самое позднее окончание работы соотвественно

private int num\_Prev; // Количество предшествующих работ

private Job[] previos; // Множество предшествующих работ

private bool finish, inwork; // Маркер завершение работы

private int mulct; // Штраф за опоздание при выполнении

private int penalty; // Общий штраф полученный по завершению выполнения

public int start, end; // Начало и конец выполнения работы

//Методы

public Job(string \_name, int \_ID, int \_early, int \_late, int \_mulct) //Конструктор с заданным именем, временем раннего начала и познего окнчания выполнения, а так же штрафом

{

//Инициализируем поля

id = \_ID;

name = \_name;

early = \_early;

late = \_late;

mulct = \_mulct;

num\_Prev = 0;

finish = false;

inwork = false;

start = -1;

end = -1;

}

public void AddPrevios(Job NewPrevios) // Добавление предшествующей работы

{

//Увеличиваем размерность множества предшественников

if (num\_Prev == 0) previos = new Job[1];

else Array.Resize(ref previos, num\_Prev + 1);

previos[num\_Prev] = NewPrevios; // Добавляем предшествующую работу

num\_Prev++; //Увеличиваем счетчик предшественников

}

public string Name() //Возвращает название работы

{

return name;

}

public int ID() //Возвращает индивидуальный номер работы

{

return id;

}

public int EarlyTime() //Возвращает самое раннее время начала

{

return early;

}

public int Mulct() //Возвращает штраф за опоздание

{

return mulct;

}

public bool Ready() // Работа готова к выполнению

{

if ((finish == true) || (inwork == true)) return false; //Если работа завершена или в работе - не готова к выполнению

//Проверяем завершение выполнения предшественников

for (int i = 0; i < num\_Prev; i++)

if (previos[i].finish == false) return false;

return true;

}

public void Execut() //Возвращает true, если в момент time рыбота находится в процессе выполнения, иначе false

{

inwork = true;

}

public void Finish(int time) //Работа выполнена

{

inwork = false; // Снимаем стаус "в работе"

finish = true; // Устанваливаем статус "завершен"

// Рассчитываем полученный штраф за работу

if (time > late) penalty = mulct \* (time - late);

else penalty = 0;

Console.WriteLine("!!Penalty by" + id + " "+time+" "+late+" - " + penalty);

}

public int Penalty(int time) //Возвращает общий штраф на момент времени time

{

if (finish == true) // Для заверешенных уже посчитано

{

return penalty;

}

else if (time > late) //Если просрочилось - добавляем штраф

{

return mulct \* (time - late);

}

else return 0;

}

}

1. class Project

{

private string name;

private int early, late; //Время начала и окончания рыботы над проектом

private int num\_Job; //Количество работ в проекте

private int id\_jobs; //последнее id работы

private int num\_Workers; //Количество исполнителей

public Job[] project\_Job; //Множество работ в проекте

public Worker[] workers; //Множество исполнителей в проекте

public Project(string \_name, int \_early, int \_late, int \_num\_Job, int \_num\_Workers, string[] \_name\_Workers, int[,] \_workers\_time) //Конструктор

{

//Инициализируем поля класса

name = \_name;

num\_Job = 0;

id\_jobs = 0;

num\_Workers = \_num\_Workers;

early = \_early;

late = \_late;

//создаем массив исполнителей

workers = new Worker[num\_Workers];

//создаем исполнителей проекта

for (int i = 0; i < num\_Workers; i++)

{

int[] time\_of\_Job = new int[\_num\_Job];

//Выделяем массив времен выбранного работника

for (int j = 0; j < \_num\_Job; j++)

{

time\_of\_Job[j] = \_workers\_time[i, j];

}

//создаем работника

workers[i] = new Worker(\_name\_Workers[i], \_num\_Job, time\_of\_Job);

}

}

public void AddJob(string \_name, int \_early, int \_late, int \_mulct, int \_num\_Previos, int[] \_previos) //Добавляем множество работ в проект

{

//Расширяем множество работ

if (num\_Job == 0) project\_Job = new Job[1];

else Array.Resize(ref project\_Job, num\_Job + 1);

// Устанваливаем временные ограничения для работы, если отсутсвует ограничения или они выходят за рамки ограничений проекта - заменяются значениями проекта

if ((\_early == -1) || (\_early < early)) \_early = this.early;

if ((\_late == -1) || (\_late > late)) \_late = this.late;

int \_id = ++id\_jobs; // увеличиваем счетчик id работ проекта

// Создаем новую работу в массиве работ проекта

project\_Job[num\_Job] = new Job(\_name, \_id, \_early, \_late, \_mulct);

//Добавляем для новой работы множество предшестующих

for (int i = 0; i < \_num\_Previos; i++)

{

project\_Job[num\_Job].AddPrevios(project\_Job[\_previos[i] - 1]);

}

num\_Job++;

}

public Job[] ReadyJobs(int time) //Возвращает фронт работ проекта в момент времени time

{

//Определяем множество работ, готовых к выполнению - фронт работ

Job[] Front = new Job[1];

int num = 0; //Количество работ во фронте

//Проверяем условия готовности работ, подходящие добавляются во фронт

for (int i = 0; i < num\_Job; i++)

{

if ((project\_Job[i].EarlyTime() <= time) && (project\_Job[i].Ready())) //ранее время старта работы прошло и работа готова к выполнению

{

if (num != 0) Array.Resize(ref Front, num + 1); //Изменяем размерность массива фронта

Front[num] = project\_Job[i]; //Добавляем работу

num++; //Увеличиваем счетчик работ

}

}

//Возвращаем сформированный фронт работ

return Front;

}

public void SetStatusJob(int time) //Устанавливает статус работ проекта на момент времни time

{

for (int i = 0; i < num\_Job; i++)

{

if (project\_Job[i].start == time) project\_Job[i].Execut(); //Если момент времени равен началу выполнения - выполняется

if (project\_Job[i].end == time) project\_Job[i].Finish(time); //Если момент времени равен концу выполнения - завершен

}

}

public string Name() //Возвращает имя проекта

{

return name;

}

public int NumJobs() //Возвращает количество работа проекта

{

return num\_Job;

}

public int NumWorkers() //Возвращает количество исполнителей

{

return num\_Workers;

}

public int PenaltyProject(int time) //Общая сумма штрафа работ проекта на момент времени time

{

int penalty=0;

for (int i=0; i<num\_Job; i++)

{

penalty += project\_Job[i].Penalty(time); //Прибавляем штраф очередной работы

}

return penalty;

}

1. class Worker

{

private string name; // Имя исполнителя

private int num\_Job; // Количество работ

private int[] time\_of\_work; // Временые затраты на выполнения работ

public Worker(string \_name, int \_num\_Job, int[] \_time\_of\_work) // Конструктор

{

name = \_name;

num\_Job = \_num\_Job;

time\_of\_work = new int[num\_Job];

for (int i = 0; i < num\_Job; i++)

{

time\_of\_work = \_time\_of\_work;

}

}

public string Name() // Возвращает имя исполнителя

{

return name;

}

public int Time\_of\_work(int i) //Возвращает временыt затраты на выполнение i работы

{

return time\_of\_work[i];

}

**Приложение №3. Главный класс программы**

static class Program

{

public static Form1 form1;

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

form1 = new Form1();

Application.Run(form1);

}

public static Project OpenProject(string FileName) //Открываем проект, сохраненный как файл Excel

{

string proj\_name; //Имя проекта

int proj\_start; //Раннее время старта работы над проектом

int proj\_end; //Позднее время окончания работы над проектом

int NumJobs; //Количество работ

int NumWorkers; //Количество исполнителей

string[] Name\_Workers; //Именя исполнителей

int[,] Workers; //Временные затраты на выполнение работ

Excel.Application app = new Excel.Application(); // Приложение Excel

Excel.Workbook workbook; // Книга данныъх

Excel.Worksheet worksheet; // Страница данных

Excel.Range range; // Ячейки данных

//Открываем книгу

workbook = app.Workbooks.Open(FileName, Missing.Value, Missing.Value, Missing.Value,

Missing.Value, Missing.Value, Missing.Value, Missing.Value, Missing.Value, Missing.Value,

Missing.Value, Missing.Value, Missing.Value, Missing.Value, Missing.Value);

//Выбираем страницу с характеристиками проекта

worksheet = (Excel.Worksheet)workbook.Sheets.get\_Item(1);

range = worksheet.get\_Range("B1", Type.Missing); //Считываем название проекта

proj\_name = Convert.ToString(range.Value2);

range = worksheet.get\_Range("B2", Type.Missing); //Считываем раннее время старта работы над проектом

proj\_start = (int)range.Value2;

range = worksheet.get\_Range("B3", Type.Missing); //Считываем позднее время окончания работы над проектом

proj\_end = (int)range.Value2;

range = worksheet.get\_Range("B4", Type.Missing); //Считываем количество работ, включенных в проект

NumJobs = (int)range.Value2;

range = worksheet.get\_Range("B5", Type.Missing); //Считываем количество работ, включенных в проект

NumWorkers = (int)range.Value2;

//Создаем множества временных затрат и имен исполнителей

Workers = new int[NumWorkers, NumJobs];

Name\_Workers = new string[NumWorkers];

//Добавляем исполнителй

worksheet = (Excel.Worksheet)workbook.Sheets.get\_Item(3); //Открываем страницу исполнителей проекта

range = worksheet.UsedRange;

for (int i = 0; i < NumWorkers; i++)

{

Name\_Workers[i] = (range.Cells[2, 2 + i] as Excel.Range).Value2.ToString(); //Считываем название исполниетля

for (int j = 0; j < NumJobs; j++)

{

Workers[i, j] = (int)(range.Cells[4 + j, 2 + i] as Excel.Range).Value2; //Считаываем время выполнения работ

}

}

//Создаем проект

Project proj = new Project(proj\_name, proj\_start, proj\_end, NumJobs, NumWorkers, Name\_Workers, Workers);

worksheet = (Excel.Worksheet)workbook.Sheets.get\_Item(2); //Открываем страницу работ проекта

range = worksheet.UsedRange; //Задаем множество ячеек для работы - изменных ранее

//Добавляем работу проекта

for (int Rnum = 2; Rnum < NumJobs + 2; Rnum++)

{

//считываем данные из файла

string job\_name = (range.Cells[Rnum, 2] as Excel.Range).Value2.ToString();

int job\_start = (int)(range.Cells[Rnum, 3] as Excel.Range).Value2;

int job\_end = (int)(range.Cells[Rnum, 4] as Excel.Range).Value2;

int mulct = (int)(range.Cells[Rnum, 5] as Excel.Range).Value2;

int NumPrevios = (int)(range.Cells[Rnum, 6] as Excel.Range).Value2;

int[] Previos = new int[NumPrevios];

for (int p = 0; p < NumPrevios; p++)

{

Previos[p] = (int)(range.Cells[Rnum, 7 + p] as Excel.Range).Value2;

}

//Добавляем работы в проект

proj.AddJob(job\_name, job\_start, job\_end, mulct, NumPrevios, Previos);

}

//Закрываем Excel файл

app.Workbooks.Close();

app.Quit();

return proj;

}

public static int[,] CreateSchedule(Project project, int time) //Создатель расписания выполнения проекта за промежуток времени длиною time

{

//Определяем расписание как график Ганта

int[,] Plan = new int[project.NumWorkers(), time];

//Проходим все такты планирования

for (int i = 0; i < time; i++)

{

//Создаем фронт работ в данный момент времени

Job[] Front = project.ReadyJobs(i);

//Если фронт не пустой

if (Front[0] != null)

{

int[] waiting\_workers = new int[1]; //Массив свободных исполнителей

int waiting\_num = 0; //Количество свободных исполнителей

// формируем множетсво свободных исполнителей

for (int p = 0; p < project.NumWorkers(); p++) //Начиная с текущего и до последнего исполнителя

if (Plan[p, i] == 0) //Если исполнитель свободен

{

if (waiting\_num != 0) Array.Resize(ref waiting\_workers, waiting\_num + 1); //Увеличиваем размерность массива свободных исполнителей

waiting\_workers[waiting\_num] = p; //Добавляем свободного исполнителя

waiting\_num++; //Увеличиваем количество свободных исполнителей

}

int k = 0;

while ((waiting\_num > 1) && (k < Front.Length))

{

int min\_time = Int32.MaxValue; //Минимальное время выполнения работы

int num\_min = -1; //Номер исполнителя с наименьшим временем выполнения

int waiting\_min = -1; //Номер исполнителя с наименьшим временем выполнения в массиве свободных исполнителей

for (int p = 0; p < waiting\_num; p++) //Начиная с первого до последнего свободного исполнителя

{

if (project.workers[waiting\_workers[p]].Time\_of\_work(Front[k].ID() - 1) < min\_time) //Если время исполнителя меньше предыдущих

{

min\_time = project.workers[waiting\_workers[p]].Time\_of\_work(Front[k].ID() - 1); //запоминаем новое минимальное время

num\_min = waiting\_workers[p]; //запоминаем номер исполнителя с новым минимальным временм

waiting\_min = p; //запоминаем номер исполнителя с новым минимальным временем в массиве свободных

}

}

if (i + min\_time <= time)

{

for (int l = 0; l < min\_time; l++) //Заполняем план для исполнителя с минимальным временм исполнения

Plan[num\_min, i + l] = Front[k].ID();

Front[k].start = i; //Задаем начало выполнения работы

Front[k].end = i + min\_time - 1; //Задаем конец выполнения работы

Front[k].Execut(); //Задаем статус "выполняется"

Front = project.ReadyJobs(i);

}

int[] copy = waiting\_workers; //Убираем выбранного исполнителя из массива свободных

Array.Resize(ref waiting\_workers, waiting\_num - 1);

for (int p = 0; p < waiting\_min; p++)

waiting\_workers[p] = copy[p];

for (int p = waiting\_min; p < waiting\_num - 1; p++)

waiting\_workers[p] = copy[p + 1];

waiting\_num--;

k++;

}

}

for (int j = 0; j < project.NumWorkers(); j++) //Цикл по всем исполнителям

{

if ((Plan[j, i] == 0) && (Front[0] != null)) //Если не используется и есть ожидающие работы - планируем

{

int min\_time = Int32.MaxValue;

int num\_min = 0;

for (int p = 0; p < Front.Length; p++)

{

if (project.workers[j].Time\_of\_work(Front[p].ID() - 1) < min\_time)

{

min\_time = project.workers[j].Time\_of\_work(Front[p].ID() - 1);

num\_min = p;

}

}

int longWork = min\_time;

if (i + min\_time <= time)

{

for (int l = 0; l < longWork; l++)

Plan[j, i + l] = Front[num\_min].ID();

Front[num\_min].start = i;

Front[num\_min].end = i + longWork - 1;

Front[num\_min].Execut();

Front = project.ReadyJobs(i);

}

}

Console.Write(Plan[j, i] + " ");

}

project.SetStatusJob(i);

Console.WriteLine();

}

return Plan;

}

}