Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка інформаційної системи розкладу занять закладу вищої освіти»

Студента 4 курсу, 7 групи,
Напрям підготовки 6. 050103
«Програмна інженерія» _____ Хоанг Шон
підпис студента

Науковий керівник кандидат педагогічний наук, старший викладач _____ Пашорін Валерій Іванович
підпис керівника

Гарант освітньої програми
кандидат технічних наук, доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2019

ЗМІСТ

#

					КНТЕУ 6.050103 07-10.БР		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Зав. Каф.		*Криворучко О.В*			Розробка інформаційної системи розкладу занять закладу вищої освіти	*Стадія*	*Аркуш*	*Акрушів*
Керівник		*Пашорін В.І*				Зміст	*2*	*25*
Гарант		*Цензура М.О.*				*Факультет обліку, аудиту та інформаційних систем, 4 курс, 7 група*		
Розробив		*Хоанг Шон.*			*Зміст*			

ВСТУП

Актуальність теми. Розв'язки багатьох задач теорії розкладів можливо надати у вигляді перестановок чисел від 1 до *n*. Галузь їх практичного використання досить широка: календарне планування виробничих процесів, транспортні системи, побудова розкладів навчального процесу вищих навчальних закладів (ВНЗ).

Велика кількість задач теорії розкладів, на перший погляд, не є задачами, заданими на перестановках. У них для завдання розв'язку або недостатньо однієї перестановки, або необхідно враховувати деякі додаткові обмеження. У цьому випадку перестановку можна розглядати як деякий початковий порядок елементів (об'єктів), що використовується для отримання повного розв'язку, який задовольняє всім обмеженням. Тобто для одержання розв'язку слід побудувати процедуру, що при заданому початковому порядку одержувала б шуканий результат. Наприклад, такими задачами є найскладніші задачі теорії розкладів – конвеєрна і загальна задачі, задача побудови розкладу навчального закладу, задача комівояжера та інші.

Метою дослідження є розв'язання задач складання розкладів і створення програмного продукту розробка інформаційних систем розкладу занять закладу вищої освіти.

Для досягнення поставленої мети потрібно вирішити такі завдання:

– систематизувати та проаналізувати теоретичні та практичні досягнення в дослідженні задач теорії розкладів, заданих на перестановках;

- побудувати загальну математичну модель задач дослідження;
- розробити методи розв’язання поставлених задач, що дозволяють отримувати розв’язки із заданою точністю;
- на основі розроблених методів побудувати алгоритми складання розкладів та їх паралельні версії для реалізації на кластерних системах для збільшення швидкодії;

– програмно реалізувати розроблені алгоритми, організувати та провести обчислювальний експеримент для їх дослідження, проаналізувати результати обчислювального експерименту.

Об'єкт дослідження – освітній процес.

Предмет дослідження – розклад занять навчальних груп закладу вищої освіти.

РОЗДІЛ 1
ВІДОМІ МЕТОДИ І РІШЕННЯ ЗАДАЧІ ФОРМУВАННЯ РОЗКЛАДІВ

1.1 Огляд задачі формування розкладу

Більшість комбінаторних оптимізаційних задач теорії розкладів є NP-повними, і тому для знаходження їхніх точних розв'язків не можуть бути застосовані ефективні алгоритми.

Існують дві основні стратегії розв'язання NP-повних задач: пошук оптимуму за рахунок організації перебору в просторі перестановок і застосування наближених алгоритмів, включаючи евристичні, які побудовані виходячи з розумних, але не маючих строгого теоретичного обґрунтування міркувань.

Вибір першої стратегії потребує невиправданих затрат часу. Для розв'язання задач з обсягами вхідних даних, використовуваних на практиці, за обмеженого часу на обчислення кращою є друга стратегія. Але в цьому випадку відомі алгоритми для задач теорії розкладів дають велику похибку відносно оптимуму. Їх застосування виправдане, якщо точність побудованих розв'язків характеризується прийнятною граничною величиною ε щодо оптимуму.

У роботі сформульована задача проектування і дослідження методів побудови розв'язків, близьких до оптимальних, або розв'язків із заданою точністю для задач теорії розкладів, заданих на перестановках, і створення їх паралельних версій для використання на багатопроцесорних системах з метою збільшення швидкодії.

Нехай задана скінченна множина G об'єктів довільного виду: пункти в системі транспортних сполучень, послідовності операцій

технологічного процесу, перелік занять, які треба проводити у зазначений час, і т. п. Всі об’єкти пронумеровані числами від 1 до n, $|G| = n$.

Розв’язком задачі на перестановках у просторі розв’язків P є певна послідовність із n об'єктів. Їй відповідає перестановка $\pi = (i_1, i_2, ..., i_k, ..., i_n)$ номерів об’єктів, у якій об’єкт з номером i_k займає позицію k. Множина G характеризується набором параметрів X, що містить припустимі набори даних x, а також сукупністю умов q у системі обмежень Q.

Для обчислення значень функціоналу цілі необхідні достатньо великі ресурси часу. Те ж саме можна сказати і для більш складних задач теорії розкладів, заданих на перестановках (наприклад, для загальної задачі теорії розкладів). Але у багатьох випадках немає необхідності обчислювати такі складні вирази. Достатньо побудувати процедуру, яка б для заданого набору перестановок будувала розклад та обчислювала значення функціонала цілі.

Усі вимоги до розкладу можна розділити на основні і неосновні. Повне виконання всіх неосновних вимог до розкладу занять при обов’язковому виконанні основних найчастіше виявляється практично неможливим. У цьому випадку має місце дефект якості складеного розкладу.

Виходячи з аналізу вимог, видається природним при побудові математичної моделі розкладу записати основні вимоги у вигляді обмежень задачі, а неосновні – звести в цільову функцію. Таким чином, задача побудови розкладу занять полягає в мінімізації дефекту якості, викликаного порушенням ряду неосновних вимог при повному виконанні основних вимог. Як основні вимоги можна виділити такі:

проведення не більше одного заняття одним викладачем одночасно;

неможливість проведення в одному приміщенні більше одного заняття одночасно;

неможливість проведення занять для викладача в заборонені часові інтервали;

проведення занять у спеціалізованих аудиторіях;

неможливість проведення занять в аудиторіях у заборонені для них інтервали часу.

Виконання неосновних вимог бере на себе цільовий функціонал задачі, що дозволяє мінімізувати дефект якості розкладу. Кількісну оцінку дефекту якості розкладу будемо виражати сумою штрафів за кожен відступ від неосновних вимог.

До числа неосновних вимог варто віднести небажаність „вікон" у груп студентів, проміжки часу між проведенням занять одним викладачем, проведення лекцій на початку і в середині навчального дня та інше. Звичайно, перелік цих вимог не претендує на повноту і може змінюватися з урахуванням специфіки ВНЗ.

Штраф за будь-яке порушення з переліку неосновних вимог до розкладу передбачається брати за десятибальною шкалою оцінок. При цьому розмір штрафу в кожному конкретному навчальному закладі можна попередньо встановити за допомогою експертних оцінок.

За розробленою програмою було розраховано семестровий розклад Житомирського технологічного університету. Типовий приклад складався з наступного набору вихідних даних: кількість викладачів – 78, кількість предметів – 160, кількість груп – 102, кількість аудиторій – 74.

У результаті розподілу навантаження з урахуванням того, що лекційні заняття проводяться для декількох груп, отримано 6950 трійок (предмет, група, викладач).

Параметри генетичного алгоритму обрані такими: кількість хромосом у популяції – 200, кількість популяцій – 1000, коефіцієнт турнірного відбору – 3, штрафний коефіцієнт вікон для груп – 5, штрафний коефіцієнт вікон для викладачів – 1, штрафні коефіцієнти небажаності проведення занять (наприклад, лекцій) у певний час – від 1 до 5.

Початкова популяція формувалася випадковим способом. Для уточнення коефіцієнтів схрещування і мутації попередньо було розраховано близько 100 прикладів з невеликими наборами вихідних даних. У результаті отримано, що із зростанням коефіцієнта схрещування результати поліпшуються, а коефіцієнт мутації варто зменшувати. Найбільш сприятливими парами варто вважати коефіцієнт схрещування 0,96 – 0,99, а коефіцієнт мутації дорівнює 0,01 – 0,03.

Найменше значення штрафу в отриманому розкладі 277 одиниць. Час рахунку складає близько 6 годин, а в послідовному режимі більше 17. Причому обробка кожної популяції забирає не набагато більше 1 хв. Основний час при формуванні популяції забирає обчислення функцій придатності – близько 0,27 хв. для кожної хромосоми.

Результати експерименту підтвердили необхідність розробки паралельної версії алгоритму. Підтверджена також ідея здійснювати розпаралелювання при підрахунку функції придатності.

1.2 Розгляд можливих варіантів реалізації генерації розкладу і вибір варіанту для розробки

Розглянемо методи розв’язання задач, заданих на перестановках, які побудовані на базі генетичних алгоритмів.Перше, з чим стикається розробник при проектуванні власного генетичного алгоритму, це кодування або подання розв’язку у вигляді хромосоми.

Відомо, що при розв'язанні конвеєрної задачі теорії розкладів з не більш ніж трьома машинами для пошуку оптимального розв'язку досить обмежитись тільки перестановочним розкладом. Тоді кодування розв'язку у вигляді хромосоми цілком очевидне. Хромосома являє собою перестановку цілих чисел від 1 до n – масив номерів робіт у порядку їхнього виконання. Якщо в конвеєрній задачі не обмежуватися лише перестановочними розкладами, то для кодування розв'язку необхідно

зберігати порядок виконання робіт на кожній машині. Тоді для m машин довжина хромосоми збільшується до $n*(m-2)$. Хромосома являє собою перестановку m наборів чисел від 1 до n. Кожен ген такої хромосоми представляє номер роботи, а порядок їхнього проходження показує, на яку машину в розкладі буде встановлюватися чергова операція цієї роботи. Для загальної задачі теорії розкладів довжина хромосоми буде дорівнювати кількості операцій k по всіх роботах (не більш, ніж $n*m$), а кожен ген також являє собою номер роботи. Установка цієї роботи в розклад, який проектується, проводиться за допомогою деякої процедури, що враховує різні маршрути виконання робіт.

У задачі побудови розкладу вищого навчального закладу ми зіштовхуємося з деякими трійками (заняттями) виду (предмет, викладач,

група), яким треба знайти місце (аудиторію) і час виконання. Тоді такій трійці відповідає ген хромосоми. Таким чином, всі заняття нумеруються числами від 1 до N, і хромосома являє собою деяку перестановку цих чисел. Порядок чисел у хромосомі визначає чергу занять в розкладі. Для побудови самого розкладу та обчислення функції придатності хромосом існує процедура, що установлює заняття в розклад з урахуванням виконання всіх обов'язкових вимог і обчислює функцію придатності з урахуванням всіх необов'язкових вимог. Процедура

повертає велике число, якщо для даної хромосоми розклад не може бути побудовано, і суму штрафу розкладу, якщо його побудовано.

Такий підхід дозволяє використати для розв'язання цієї задачі схему класичного генетичного алгоритму з високою швидкодією і перенести всю складність розв'язання на процедуру побудови розкладу та обчислення функції придатності для кожної хромосоми. Це дає вагомий виграш при реалізації паралельної версії методу.

Для всіх задач побудови розкладів, заданих на перестановках, основні етапи методу, який пропонується, реалізовані у такий спосіб:

хромосоми в початковій популяції заповнюються випадковими перестановками чисел від 1 до N;

вибір хромосом для схрещування проводиться за методом "турнірного відбору";

застосовується одномістне схрещування;

операція мутації є перестановкою двох випадково обраних елементів у хромосомі;

реалізується регенераційний тип репродукції поколінь із переносом кращої хромосоми з попереднього покоління до наступного;

закінчення алгоритму проводиться або при розгляді заданої кількості поколінь, або при конвертуванні покоління.

У процесі роботи генетичного алгоритму найважливішою є постійна підтримка припустимості розв'язків, тобто підтримка хромосом такими, які не порушували б обов'язкових обмежень. На припустимість хромосоми може вплинути оператор схрещування. Тому після виконання цього оператора потрібно перевіряти новостворену хромосому на наявність порушень обов'язкових обмежень і з появою таких виконати процедуру виправлення хромосоми. У даній реалізації алгоритму пропонується така процедура з обчислювальною складністю не більш ніж

$O(n^2)$. У роботі також пропонується інший підхід до підтримки припустимості розв'язків – непряме кодування хромосом. Кожен ген у хромосомах з непрямим кодуванням є відносним порядковим номером ще не встановленого об'єкта. Прямому поданню хромосоми (1,3,2,5,4) буде відповідати непряме (1,2,1,2,1) у фіксованому порядку (1,2,3,4,5). Застосування непрямих подань хромосом дозволяє уникнути неприпустимих розв'язків, але збільшує час на розрахунок функції придатності.

Оскільки будь-якому припустимому розкладу можна поставити у відповідність деяку перестановку чисел від 1 до N (причому одному розкладу може відповідати кілька перестановок), то перебравши з деякою ймовірністю всі перестановки, ми переберемо всі припустимі розклади. Значення найкращої функції придатності при переході від одного покоління до іншого не погіршується, тобто алгоритм розроблений коректно, і тому при розгляді досить великої кількості поколінь ми з імовірністю, близькою до одиниці, одержимо оптимальний розв'язок.

РОЗДІЛ 2
МЕТОДИ СТВОРЕННЯ ТА ОПТИМІЗАЦІЇ РОЗКЛАДУ

2.1 Загальний опис алгоритму

Розглянемо наближений метод, розроблений на спільному використанні алгоритму типу гілок та меж і генетичного алгоритму. Генетичний алгоритм застосовується для знаходження верхніх оцінок поточних рішень і вибору напрямку в дереві перебору, що будується за схемою гілок та меж. Таким чином, у комбінації точного і наближеного методів оптимізації $F(\pi)$ побудований розв'язок пропонується разом з доказом того, що його відносна похибка не більше наперед заданого числа ε.

Опишемо модифікацію методу гілок та меж, що мінімізує $F(\pi)$. Для заданої підмножини $M = \{i_1, i_2, ..., i_m\}$ множини $N = \{1,2,...,n\}$ позначимо π^m

перестановку на M або часткову перестановку на N розмірності m. Якщо $M = N$, то π^m називається повною перестановкою на N, $\pi^\infty = \pi$. Для $N \setminus M \neq \varnothing$ визначимо $\{\pi^m, 0\}$ – множину всіх повних перестановок, що починаються з часткової перестановки π^m. Кожен елемент цієї множини складається з перестановки π^m та її продовження у вигляді перестановки на $N \setminus M$.

Множина часткових розв'язків задачі взаємно однозначно відповідає множині часткових перестановок на N, так що кожний частковий розв'язок може бути надано у вигляді відповідної часткової перестановки.

2.2 Розробка цільової функції

1.1 Найменування та область застосування

Найменування програми: «Додаток для перегляду розкладу занять»

Програма призначена для використання студентами та викладачами для перегляду особистого розкладу.

1.2 Призначення розробки

Функціональним призначенням програми є надання користувачу можливості зручного перегляду розкладу групи, викладача або аудиторії.

Експлуатаційне призначення

Програма повинна експлуатуватися на ПК та смартфонах.

Кінцевими користувачами програми повинні бути студенти та викладачі ВУЗу.

Вимоги до програмного забезпечення

Вхідні дані

- перелік предметів,
- розклад пар,
- перелік аудиторій,
- перелік викладачів.

Вихідна інформація

- інформація про розклад занять на сьогодні,
- нагадування про початок пари.

Опис функцій та обмежень

Функції:

- функція планування,
- функція оповіщення.

Обмеження:

- Оповіщення працює лише протягом навчального дня.

Часові характеристики

- програма працює цілодобово, окрім обмежень

Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

• технічне завдання;

• програму і методики випробувань;

• керівництво оператора;

• відомість експлуатаційних документів.

1.3 Стадії та етапи розробки

Стадії розробки

Розробка повинна бути проведена в 6 стадій:

1. розробка технічного завдання;

2. робоче проектування;

3. тестування.

2.3 Модель розкладу

Розробити базу даних вищого навчального закладу «Розклад занять».

Забезпечити збереження і накопичення інформації про:

- учбові групи;

- аудиторії;

- викладачів;

- дисципліни;

- часовий розклад.

Створити інформаційну систему для даної предметної області. Вона повинна включати в себе:

- зв’язані таблиці;

- набір вхідних форм для заповнення таблиць;

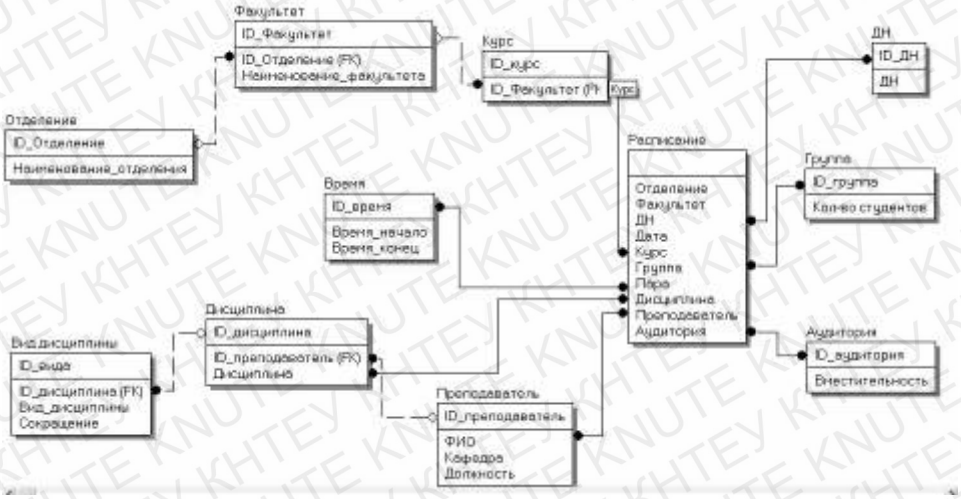

Рис.2.1. Логічна модель даних

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ МОДЕЛІ ТА ЇЇ ТЕСТУВАННЯ

3.1 Вибір середовища програмування

Головною особливістю мови C # є його орієнтованість на платформу Microsoft.NET - творці C # ставили собі за мету надання розробникам природних засобів доступу до всіх можливостей платформи .NET. Мабуть, це рішення можна вважати більш-менш вимушеним, так як платформа .NET спочатку пропонувала значно більшу функціональність, ніж будь-який з існуючих на той момент мов програмування.

Крім того, творці C # хотіли приховати від розробника якомога більше незначних технічних деталей, включаючи операції з пакування / розпакування типів, ініціалізації змінних і збірці сміття. Завдяки цьому програміст, що пише на C #, може краще сконцентруватися на змістовній частині завдання. У процесі вирішення цього завдання проектувальники C # намагалися врахувати уроки реалізації Visual Basic'a, який досить успішний в приховуванні деталей реалізації, але недостатньо ефективний для написання великих промислових систем: творці C # декларують, що нова мова володіє потужністю C ++ і в той же час простотою Visual Basic'a.

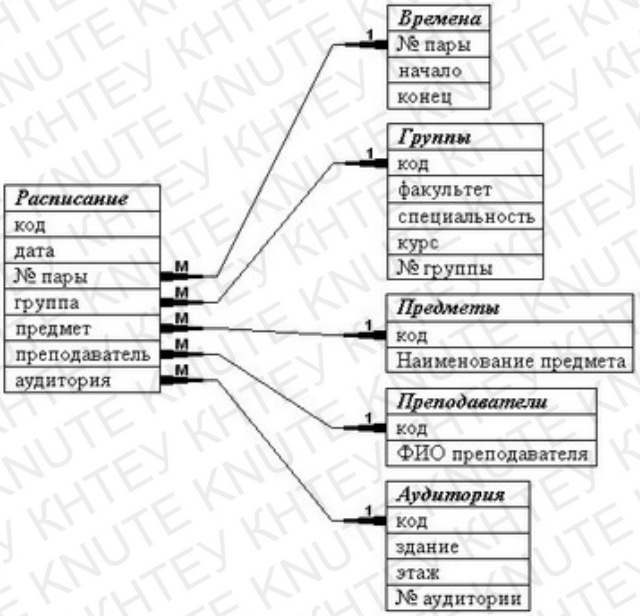

Рис. 3.2. Архітектура БД

Логічна структура інформації:

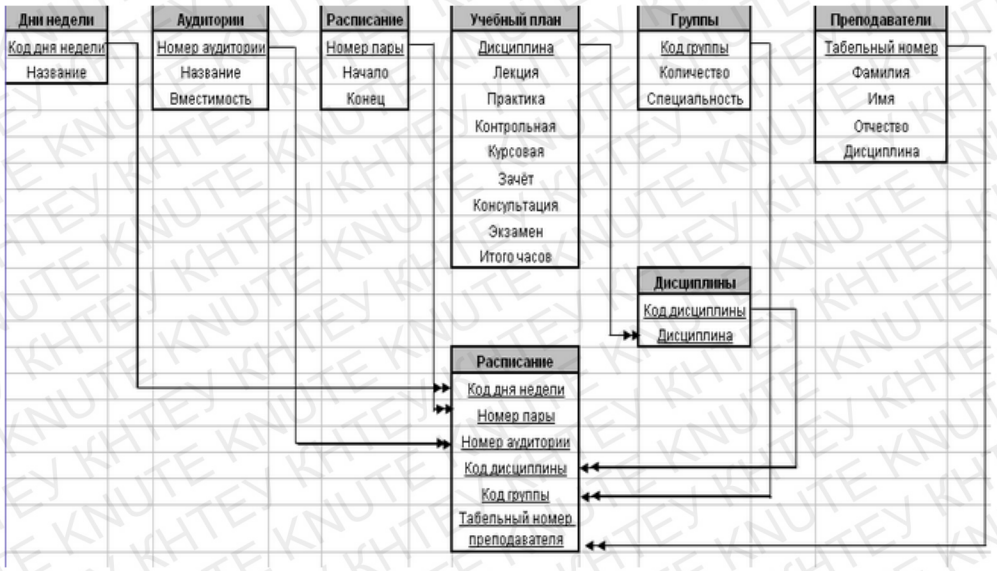

Рис. 3.2. Логічна структура інформації

Головне вікно програми матиме наступний вигляд:

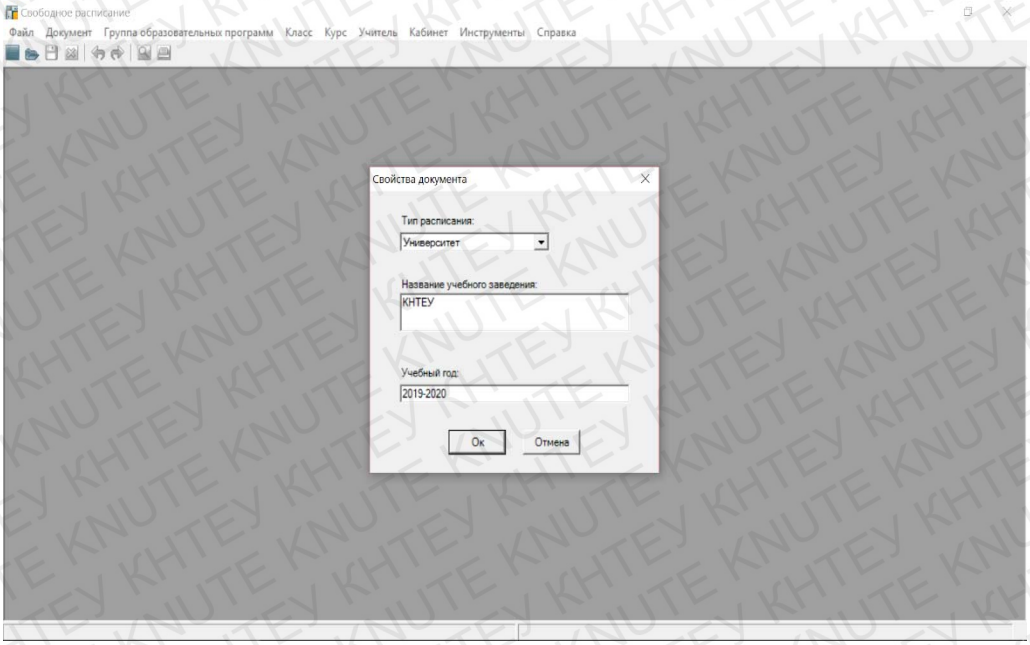

Рис.3.3. Головне вікно

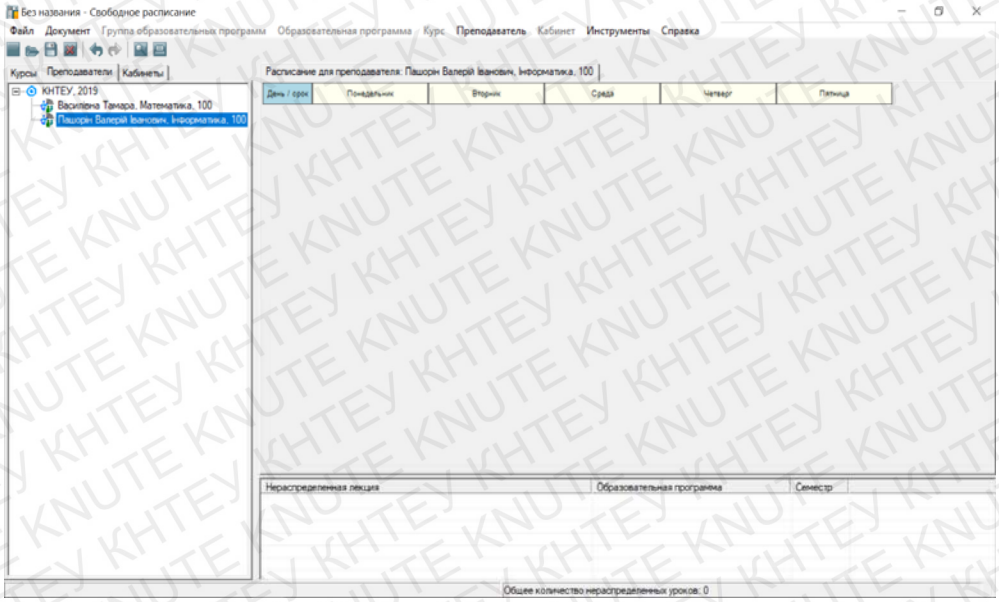

Рис.3.4. Вчителі

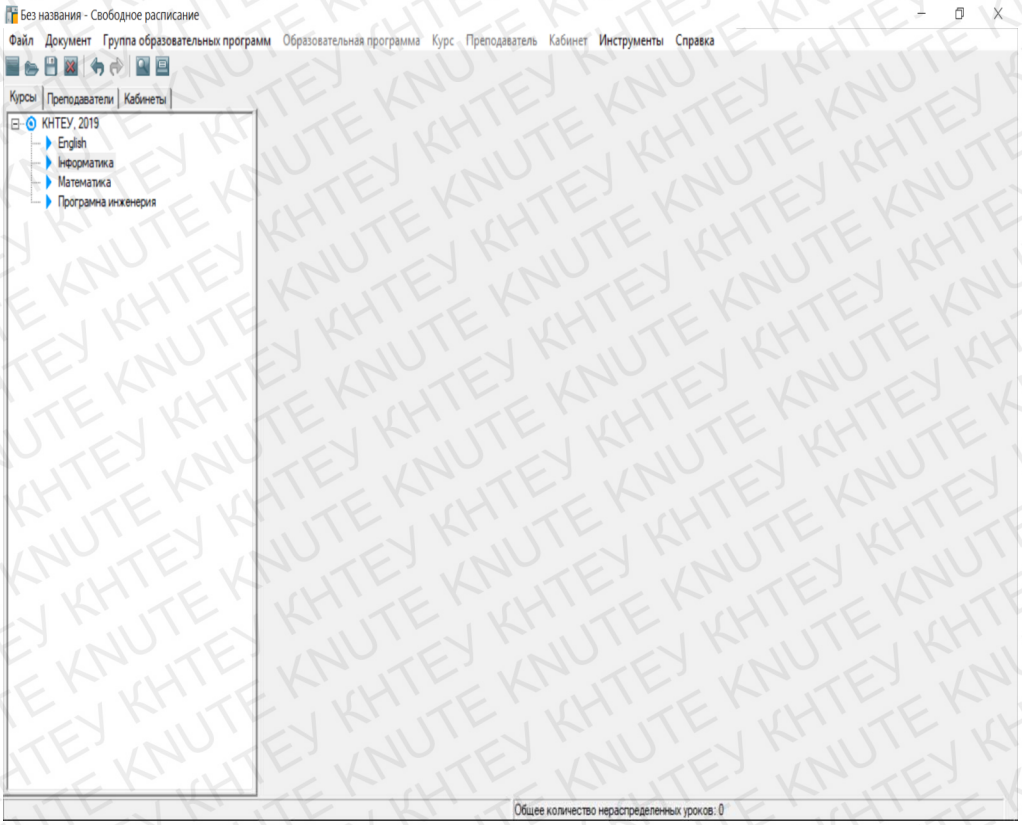

Рис.3.5. Предмети

Ще одна перевага створення нової мови програмування в порівнянні з розширенням існуючих полягає в тому, що при створенні нової мови немає необхідності піклуватися про проблеми сумісності, які зазвичай помітно ускладнюють виправлення застарілих проблем і навіть внесення нових властивостей в стандарт мови (докладний опис труднощів, що виникають при розширенні старого мови програмування, можна прочитати в книзі Б. Страуструпа "Дизайн і еволюція мови C ++", M .: ДМК, 2000).

Таким чином, C # є нову мову програмування, орієнтований на розробку для платформи .NET і придатний як для швидкого прототипування застосувань, так і для розробки великомасштабних додатків.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Загальновідомим є той факт, що якісна інформація дозволяє фахівцям різних областей здійснювати свою професійну діяльність цілеспрямовано і ефективно. Тому в умовах, що склалися зростання обсягу і ролі такої інформації виникає необхідність застосування інформаційних технологій, що дозволяють здійснювати її збір, структурування, збереження, пошук, обробку та видачу відповідно до вимог, що пред'являються користувачами.

Перша глава присвячена розгляду загальних уявлень про предметну область. Сформовано технічне завдання на проектування програми.

У другому розділі здійснюється опис процесів розробки. Також було розглянуто структурний підхід до проектування програми. Результатом якого є побудова моделі предметної області.

Наслідком цього розділу можна вважати розроблену програму "Розклад", яка вирішує всі поставлені завдання:

Введення, редагування та видалення інформації, що є основою для складання розкладу і званої "Дані до розкладу".

Введення, редагування та видалення об'єктів розкладу. До них відносяться: викладачі, дисципліни, аудиторії, факультети, курси, групи.

Висновок звітів: розклад занять, розклад для викладачів, розклад аудиторій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грекул В.І. "Проектування інформаційних систем". [Електронний ресурс]. - URL: http://www.intuit.ua
2. Джеффри Рихтер WIN32-додатків з урахуванням специфіки 64-розрядної версії Windows
3. Захаров, В.П. Інформаційні системи: навч. посібник / В.П. Захаров. - СПб., 2002. - 187 с.
4. Игумнов Е. "Специфікація каркаса інформаційної системи з розподіленою архітектурою". [Електронний ресурс]. - URL: http://www.citforum.ua
5. Мацяшек, Лешек А. Аналіз вимог і проектування систем. Розробка інформаційних систем с використанням UML .: Пер. з англ. - М .: Видавничий дім "Вільямс". 2002. - 432 с.
6. Н. А. Литвиненко - Технология програмування на C++. Win32 API
7. Проектування автоматизованих інформаційних систем. [Електронний ресурс]. - URL: http: // works. tarefer.ua
8. Рябишева І.В. "Порівняльний аналіз підходів до проектування ІС". [Електронний ресурс]. - URL: http: //www.ict. nsc.ua

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LessonPlanNS
{
   static class Program
   {
      static void Main()
      {
         int[] groups = new int[]{1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
         int[] teachers = new int[]{4,1,2,5,3,2,5,4,3,3,1,2,2,4,3,3,3,1,5,1,5,2,2,5,1,4,5,4,1,4};

         var list = new List<Lesson>();
         for (int i = 0; i < groups.Length; i++)
            list.Add(new Lesson(groups[i], teachers[i]));

         var solver = new Solver();//создаем решатель

         Plan.DaysPerWeek = 2;//
         Plan.HoursPerDay = 6;

         solver.FitnessFunctions.Add(FitnessFunctions.Windows);//
         solver.FitnessFunctions.Add(FitnessFunctions.LateLesson
```

```
    return res1 && res2;
}

public IEnumerable<Lesson> GetLessonsOfDay(byte day)
{
    for (byte hour = 0; hour < HoursPerDay; hour++)
        foreach (var p in HourPlans[day, hour].GroupToTeacher)
            yield return new Lesson(day, hour, p.Key, p.Value);
}

public IEnumerable<Lesson> GetLessons()
{
    for (byte day = 0; day < DaysPerWeek; day++)
        for (byte hour = 0; hour < HoursPerDay; hour++)
            foreach (var p in HourPlans[day, hour].GroupToTeacher)
                yield return new Lesson(day, hour, p.Key, p.Value);
}

public override string ToString()
{
    var sb = new StringBuilder();
    for (byte day = 0; day < Plan.DaysPerWeek; day++)
    {
        sb.AppendFormat("Day {0}\r\n", day);
        for (byte hour = 0; hour < Plan.HoursPerDay; hour++)
        {
            sb.AppendFormat("Hour {0}: ", hour);
            foreach (var p in HourPlans[day, hour].GroupToTeacher)
```

```
                sb.AppendFormat("Gr-Tch: {0}-{1} ", p.Key, p.Value);
            sb.AppendLine();
        }
    }

    sb.AppendFormat("Fitness: {0}\r\n", FitnessValue);

    return sb.ToString();
  }
}

/// <summary>
/// План на час
/// </summary>
class HourPlan
{

      return false;//
    GroupToTeacher[group] = teacher;
    TeacherToGroup[teacher] = group;

    return true;
  }

  public void RemoveLesson(int group, int teacher)
  {
    GroupToTeacher.Remove(group);
    TeacherToGroup.Remove(teacher);
      {
    var res = new HourPlan();
```

```
            res.GroupToTeacher = new Dictionary<int, int>(GroupToTeacher);
            res.TeacherToGroup = new Dictionary<int, int>(TeacherToGroup);

            return res;
        }
    }

    /// <summary>
    /// Пара
    /// </summary>
    class Lesson
    {
        public byte Day = 255;
        public byte Hour = 255;
        public int Group;
        public int Teacher;

        public Lesson(byte day, byte hour, int group, int teacher)
            : this(group, teacher)
        {
            Day = day;
            Hour = hour;
        }

        public Lesson(int group, int teacher)
        {
            Group = group;
            Teacher = teacher;
```