# Анотація

Магістерська кваліфікаційна робота присвячена дослідженню та реалізації алгоритму Apriori, та введенню можливих модифікацій для кращої роботи алгоритму.

Проект включає програмну реалізацію алгоритму Apriori та проведення дослідження для цього алгоритму.

Проведено дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму для різних значень коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

Основна мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил для бази даних із дорожньо-транспортними пригодами. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

Загальний обсяг роботи 79 сторінок.

# ABSTRACT

The master's qualification work is devoted to the research and development of Apriori algorithm and provide some modifications for better performance.

The project includes software implementation of the Apriori algorithm and conducting experiments of the developed tool.

The Research was conducted for basic algorithm without modifications, the study of the modified algorithm by the first method, the research of the modified algorithm by the second method, research of the algorithm's performance for different values ​​of the coefficients and a comparative analysis of the above-mentioned studies.

The main purpose of master's qualification work is to compare the work of the basic Apriori algorithm and its modifications on large volumes of data. It is necessary to process the data with the specified algorithm and estimate the speed of the algorithm, the load on the resources, the number of found associative rules for the database with road accidentsts. To work out on the basis of computational experiments a proposal for modification of the algorithm.

The object of the research is the apriori algorithm and its modifications.

Subject of research is the time and accuracy of the search for associative rules and the search for possible modifications.

The total workload is 79 pages.

**ЗМІСТ**

[Анотація 4](#_Toc531903709)

[ABSTRACT 5](#_Toc531903710)

[ВСТУП 8](#_Toc531903711)

[РОЗДІЛ 1. АНАЛІЗ ЗАСТОСУВАННЯ АЛГОРИТМУ APRIORI 10](#_Toc531903713)

[1.1 Великі дані, визначення та їх особливості 10](#_Toc531903714)

[1.2 Поняття асоціативних правил 11](#_Toc531903715)

[1.3 Аналіз існуючих досліджень 17](#_Toc531903716)

[1.4 Мета, об’єкт та предмет дослідження 18](#_Toc531903717)

[1.5 Висновок до розділу 1 19](#_Toc531903718)

[РОЗДІЛ 2. Опис алгоритму apriori 20](#_Toc531903719)

[2.1 Алгоритм apriori 20](#_Toc531903721)

[2.2 Видобування асоціативних правил 26](#_Toc531903722)

[2.3 Опис відомих модифікацій 27](#_Toc531903723)

[2.4 Опис можливих модифікацій 30](#_Toc531903724)

[2.5 Висновок до розділу 2 32](#_Toc531903725)

[Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori 33](#_Toc531903726)

[3.1. Загальна постановка задачі 33](#_Toc531903728)

[3.2. Специфікація вимог 33](#_Toc531903729)

[3.3. Використані технології 36](#_Toc531903730)

[3.4. Реалізація інтерфейсу користувача програмного засобу для роботи алгоритму apriori 40](#_Toc531903731)

[3.5. Розробка архітектури системи для роботи алгоритму apriori 43](#_Toc531903732)

[3.6. Реалізація алгоритму apriori для аналізу ДТП 45](#_Toc531903733)

[3.7. Висновки до розділу 3 47](#_Toc531903734)

[Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ 48](#_Toc531903735)

[4.1. Опис процесу дослідження 48](#_Toc531903736)

[4.2. Аналіз ароботи алгоритму без модифікацій 50](#_Toc531903737)

[4.3. Аналіз роботи алгоритму із модифікаціями 52](#_Toc531903738)

[4.4. Висновки до розділу 4 58](#_Toc531903739)

[Розділ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ПРОДУКТУ 59](#_Toc531903740)

[5.1. Економічна характеристика проектного рішення 59](#_Toc531903741)

[5.2. Розрахунок витрат на розробку та впровадження проектного рішення 60](#_Toc531903742)

[5.3. Визначення комплексного показника якості 65](#_Toc531903743)

[5.4. Визначення експлуатаційних витрат 66](#_Toc531903744)

[5.5. Розрахунок ціни споживання проектного рішення 69](#_Toc531903745)

[5.6. Визначення показників економічної ефективності 70](#_Toc531903746)

[5.7. Висновки до розділу 5 73](#_Toc531903747)

[Висновки 74](#_Toc531903748)

[Перелік використаної літератури 76](#_Toc531903749)

[ДОДАТКИ 79](#_Toc531903750)

[Додаток А. План забезпечення якості програмного продукту 79](#_Toc531903751)

[Додаток Б. Блок-схема роботи алгоритму 90](#_Toc531903752)

[Додаток В. Приклад даних для проведення досліджень 90](#_Toc531903753)

[Додаток Г. Фрагменти коду 91](#_Toc531903754)

# ВСТУП

Зараз ІТ є однією з динамічніших сфер, де активно розробляються нові технології, створюються сотні програмних продуктів та накопичуються приблизно декілька сотень гігабайт інформації за годину. Ця інформація надходить з різних джерел, та містить багато даних.

Big Data відноситься до величезних масивів даних, які важко аналізуються традиційними методами. Найбільш відомим та широко застосовуваним зараз є алгоритм Apriori.

Алгоритм Apriori призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює ітераційно, використовуючи пошук в ширину і та принцип зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для знаходження асоціативних зв’язків.

Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

У першому розділі даної роботи проведено аналіз предметної області. Після аналізу існуючих систем, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

У другому розділі цієї роботи подано опис алгоритму та можливі модифікації. Для першої модифікації потрібно розбити множину всіх можливих транспортних пригод на дві підмножини, одна з них містить корисну інформацію для аналізу, а інша ні, отже та що не містить корисної інформації потрібно відкидати на кожному кроці виконання алгоритму. Для другої модифікації потрібно додати характеристику асоціативних правил Ліфт, та обчислювати його на кожному кроці алгоритму.

Третій розділ роботи містить специфікацію вимог до програмної реалізації алгоритму представленого у розділі 2, інформацію про використані технології, графічний інтерфейс програмного засобу та певні особливості програмної реалізації засобу для проведення аналізу даних алгоритмом Apriori.

У четвертому розділі показано дослідження розробленого засобу. Дослідження складається з таких частин: дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму при різними значеннями коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

У останньому розділі даної роботи показано економічну характеристику програмного продукту, розробленого у розділі 3.

# РОЗДІЛ 1. АНАЛІЗ ЗАСТОСУВАННЯ АЛГОРИТМУ APRIORI

* 1. **Великі дані, визначення та їх особливості**

Великі дані - це набори даних, настільки об'ємні та складні, що традиційне прикладне програмне забезпечення для обробки даних є недостатнім для їх вирішення. Великі задачі даних охоплюють захоплення даних, зберігання даних, аналіз даних, пошук, обмін, передачу, візуалізацію, запити, оновлення, конфіденційність інформації та джерела даних. Існує ряд понять, пов'язаних з великими даними: спочатку було 3 поняття обсягу, різноманітності, швидкості[1]. Іншими поняттями, які потім пояснюються великими даними, є вірогідність (тобто, скільки шуму в даних) [2] та значення. Останнім часом термін "великі дані" має на увазі використання прогнозної аналітики, аналізу поведінки користувачів або деяких інших розширених методів аналізу даних, які витягують значення з даних, а рідко - до певного розміру набору даних. "Існує мало сумнівів у тому, що кількість даних, які зараз доступні, дійсно велика, але це не є найважливішою характеристикою цієї нової екосистеми даних". Аналіз наборів даних може знайти нові зв'язки з "точковими діловими тенденціями, запобігання захворюванням, боротьба зі злочинністю та ін. "[3]

Вчені, керівники підприємств, практики медицини, реклами та уряди так само регулярно зустрічаються з труднощами з великими наборами даних у таких областях, як пошук в Інтернеті, технології, міська інформатика та бізнес-інформатика. Вчені стикаються з обмеженнями в роботі електронної науки, включаючи метеорологію, складні фізичні симуляції, біологію та екологічні дослідження [4].

Цей термін використовується з 1990-х років, при цьому деякі приносять кредит Джону Маші за виготовлення або, принаймні, популярністю. [5] Великі дані зазвичай включають в себе набори даних, розміри яких не залежать від можливостей широко використовуваних програмних інструментів для захоплення, копіювання, керування та обробки даних протягом допустимого минулого часу [6]. Велика філософія даних охоплює неструктуровані, напівструктуровані та структуровані дані, однак основна увага приділяється неструктурованим даним.

Визначення 2016 р. Свідчить, що "великі дані являють собою інформаційні активи, які характеризуються такими великими обсягами, швидкістю та різноманітністю, що вимагають спеціальних технологій та аналітичних методів перетворення їх у вартість" [7].

Великі дані можна описати за такими характеристиками: [8]

**Обсяг** (англ. Volume) **–** кількість створених та збережених даних. Розмір даних визначає цінність та потенційне розуміння, і чи можна вважати великі дані чи ні.

**Різноманітність** (англ. Variety) **–** тип та характер даних. Це допомагає людям, які аналізують його, ефективно використовувати отримане розуміння. Великі дані витягуються з тексту, зображень, аудіо, відео; плюс він завершує відсутність частин через злиття даних.

**Швидкість** (англ. Velocity) **–** у цьому контексті швидкість, з якою дані створюються та обробляються, щоб задовольнити вимоги та виклики, які лежать на шляху зростання та розвитку. Великі дані часто доступні в режимі реального часу.

Дані треба обробляти за допомогою розширених інструментів (аналітики та алгоритмів) для виявлення значущої інформації. Наприклад, щоб керувати фабрикою, слід розглядати як видимі, так і невидимі проблеми з різними компонентами. Алгоритми формування інформації повинні виявляти та розв'язувати невидимі проблеми, такі як деградація машини, знос компонентів тощо на заводі. [9]

* 1. **Поняття асоціативних правил**

Афінітивний аналіз — один з найбільш відомих методів Data Mining. Назва його походить від слова affinity, що перекладається як «близькість», «подібність». Мета даного методу — взаємне дослідження зв'язку між подіями, які відбуваються спільно. Різновидом афінітивного аналізу є аналіз ринкового кошика (market basket analysis), ціль якого - виявити асоціації між різними подіями, тобто знайти правила для кількісного опису взаємного зв'язку між двома або більше подіями. Такі правила називаються асоціативними правилами.[10]

Основним поняттям в теорії асоціативних правил є транзакція - сукупність подій, що відбуваються разом. Типова транзакція - покупки клієнта в супермаркеті. У переважній більшості випадків клієнт купує більше одного продукту, а набір товарів, є ринковою корзиною. Це ставить питання: чи є покупка одного продукту в кошику як наслідок чи причина для покупки іншого товару, тобто пов'язані з цим події? Це з'єднання встановлює асоціативні правила. Наприклад, можна знайти асоціативне правило, в якому зазначається, що замовник, який купив молоко, з імовірністю на 75% купує хліб.

Наступним важливим поняттям є предметний набір. Це непустий набір елементів, які з'являються в одній транзакції.

Аналіз ринку корзин - це аналіз наборів даних для певної комбінації суміжних продуктів. Іншими словами, здійснюється пошук товарів, наявність яких у транзакції впливає на вірогідність наявності інших товарів або комбінацій товарів.

Сучасні касові апарати в супермаркетах дозволяють збирати інформацію про покупки, які можна зберігати в базі даних. Потім накопичені дані можуть бути використані для побудови пошукових систем асоціативних правил.

У табл. 1.1 наведено простий приклад ринкової корзини. Кожен рядок містить комбінацію продуктів, придбаних для однієї покупки. Хоча на практиці він має справу з мільйонами транзакцій із десятками та сотнями різних продуктів, приклад обмежується 10 операціями, що містять 13 видів продукції: це достатньо для ілюстрації методології визначення асоціативних правил.

Таблиця 1.1.

Приклад набору транзакцій

|  |  |
| --- | --- |
| **№** | **Транзакція** |
| 1 | Сливи, салат, помідори |
| 2 | Селера, цукерки |
| 3 | Цукерки |
| 4 | Яблука, морква, помідори, картопля, цукерки |
| 5 | Яблука, апельсини, салат, цукерки, помідори |
| 6 | Персики, апельсини, селера, помідори |
| 7 | Квасоля, салат, помідори |
| 8 | Апельсини, салат, морква, помідори, цукерки |
| 9 | Яблука, банани, сливи, морква, помідори, цибуля, цукерки |
| 10 | Яблука, картопля |

Візуальний аналіз цього прикладу показує, що всі чотири транзакції, що включають салат, також включають помідори, а в чотири з семи помідорів містять також салат. Салат і помідори в більшості випадків купуються разом. Асоціативні правила дозволяють виявити та кількісно визначити такі збіги.

Асоціативне правило складається з двох наборів об'єктів, що називаються попередніми та наступними, і записані у формі. Таким чином, асоціативне правило формулюється у формі: "Якщо стан, то наслідок".

Умова може бути обмежена лише одним предметом. Правила зазвичай відображаються за допомогою стрілок, спрямованих від умов до ефекту, наприклад, помідори → салат.

Візуальний аналіз цього прикладу показує, що всі чотири транзакції, що включають цибулю, також включають картоплю, і чотири з семи картоплі, що містять картоплю, також містять цибулю. У більшості випадків цибулю та картоплю купуються разом. Асоціативні правила дозволяють виявити та кількісно визначити такі збіги. Також можна комбінувати правила (Цибуля, Картопля) => {Бургер}

Умова може бути обмежена лише одним предметом. Правила зазвичай відображаються за допомогою стрілок, спрямованих від умови до ефекту, наприклад, цибуля → картопля.

Основними характеристиками, які описують асоціативне правило, є підтримка та надійність. [11-12].

Позначимо базу даних транзакцій через , а число транзакцій у цій базі , то кожна транзакція  являє собою певний набір предметів. Позначимо підтримку правила через , а вірогідність – через .

Підтримка асоціативного правила — це число транзакцій, які містять як умову, так і наслідок. Наприклад, для асоціації  можна записати[30]

 (1.1)

Імовірність асоціативного правила  є мірою точності правил і визначається як відношення кількості транзакцій, що містять стан і наслідок до кількості транзакцій, що містять лише умову:

 (1.2)

Якщо підтримка та ймовірність досить високі, ви, швидше за все, можете стверджувати, що будь-яка майбутня транзакція, включаючи умови, також матиме наслідок.

Наприклад розглянемо правило салат → помідори, яке випливає з попередніх спостережень. Для нього

;

.

Це правило, яке відбувається в 40% транзакцій, полягає в тому, що вихідні дані є абсолютно правильними - у всіх випадках, коли клієнт купує салат, він також купує помідори. Це легко пояснити логічно - обидва продукти використовуються для приготування овочевих страв і часто купуються разом.

Тепер давайте розглянемо асоціацію цукерки → помідори, яка містить продукти, які погано сумісні з гастрономією.

Підтримка цієї асоціації S = 4/10 = 0,4 - така ж, як у попередньому правилі, а ймовірність C = 4/6 = 0,67. Отже, відносно низька ймовірність цієї асоціації викликає сумніви в тому, що це правило.

Число 0.67, здається, не настільки мале. Чому ми говоримо про "незначну ймовірність" цієї асоціації? Справа в тому, що помідори знаходяться в 7 перевірках з 10 (Р(В)=0.7). Прийнято вважати, що всі правила з ймовірністю, меншими за просту ймовірність наслідків, не повинні розглядатися, оскільки вони фактично є випадковими. Щоб прийняти асоціативне правило, його ймовірність повинна бути не меншою, ніж ймовірність наслідку.

Останнє зауваження в досить великому діапазоні продуктів призводить до необхідності подвоїти кількість розрахунків. На практиці аналітики можуть віддати перевагу правилам, які мають високий рівень підтримки (вище певного рівня, наприклад 0,3) або високої ймовірності (щонайменше 0,8-0,85). Висування вимог щодо одночасної підтримки та правдоподібності може значно пом'якшити критерії (підтримка до 0.1-0.15, ймовірність - до 0.67-0.75). Правила, для яких підтримка та значення надійності перевищують певні пороги, встановлені користувачем, називаються сильними правилами. Всі вищезгадані числові значення є емпіричними. Наприклад, в задачі виявлення шахрайських операцій вартість підтримки може зменшитися до 1%, оскільки кількість операцій, пов'язаних із шахрайством, є порівняно невеликою.

Окрім об'єктивних оцінок (підтримки та ймовірності) кожного порожнього правила, для різних джерел рекомендується використовувати деякі суб'єктивні оцінки. Всі вони, так чи інакше, базуються на об'єктивних.

Ліфт (від interest lift – підвищення інтересу) підраховується наступним чином

 (1.3)

Ліфт - це співвідношення частоти виникнення умови у транзакціях, які містять як умову, так і наслідок до частоти виникнення ефекту в цілому. Чим більше значення ліфта, тим частіше наслідок визначається умовою у порівнянні з випадками, коли відсутня умова. Якщо ліфт дорівнює 1, зв'язок відсутній, значення, близькі до нуля, вказують на сильні взаємні відносини [12].

Для нашого прикладу в таблиці 1.1. з однаковою вірогідністю приймають два правила: 4/10 = 0,4. ,4/7 = 0,57.

*салат) =*

*C(помідори*



 6/10 = 0,6. 4/7 = 0,57.

*цукерки) =*

*C(помідори*



Здавалося б, правила однаково достовірні. Після розрахунку ліфта все стає на місця:

0,57/0,4 = 1,425;-

*салат) =*

*L(помідори*



0,57/0,6 = 0,95.

*цукерки) =*

*L(помідори*



Не вважайте ліфт універсальною мірою адекватності. Справа в тому, що правило з меншою підтримкою та збільшенням ліфта може бути менш важливим, ніж альтернативне правило з більшою підтримкою та меншим підйомом. Це пов'язано з тим, що останній застосовується для більшої кількості покупців.

Левередж (leverage – важіль, плече) – це різниця між частотою спостереження, з якої складаються умови та наслідки, а також результат періодичності появи умов і ефектів окремо

 (1.4)

Левередж дозволяє впоратися із ситуаціями, коли й підтримка, й ліфт у правил ідентичні, але їх важливість явно відрізняється. Наприклад, у нашому овочевому магазині в правила салат → помідори

;  0,7;  1/0,7 = 1,43.

Морква, як показує таблиця 1.1, також продається тільки з помідорами, і також зустрічається чотири рази, тому й у правила морква → помідори

;  0,7;  1/0,7 = 1,43.

А от левередж у цих правил відрізняється на 30%:

 0,3 – 0,3 \* 0,7 = 0,09.

 0,4 – 0,4 \* 0,7 = 0,12.

Таким чином, важливість другої асоціації більша, ніж першої.

Альтернативою для левередж є поліпшення.

Поліпшення (improvement) – це відношення частоти спостережуваних виконань правила до добутку частот появи умови й наслідку окремо.

 (1.5)

фактично, поліпшення показує, у скільки разів розглянуте правило забезпечує правильний прогноз краще, ніж випадкове вгадування. Всі правила  не є важливими.

Такі міри, як ліфт, левередж і поліпшення, можуть використовуватися для обмеження набору розглянутих асоціацій шляхом встановлення граничних значень важливості, нижче яких асоціації відкидаються.

* 1. **Аналіз існуючих досліджень**

У [13] проаналізовано алгоритм Apriori як метод пошуку асоціативних правил у структурованих та неструктурованих даних з погляду кількості знайдених правил, швидкодії та потреб в обчислювальних ресурсах. Неструктуровані дані тісно пов’язані з терміном Big Data. Актуальним завданням інженерії даних є виявлення ефективних засобів опрацювання неструктурованої інформації. Для проведення обчислювальних експериментів розроблено програмну систему, що опрацьовує дані алгоритмом Apriori, предметною областю якої вибрано торгівлю. Така система може бути прототипом реальної рекомендаційної системи.

Отже, було досліджено що,

1. Втрати у швидкодії для неструктурованої інформації порівняно зі структурованою інформацією незначні зі зростанням обсягів інформації, і зокрема, для випадку обсягу даних 2,5 Гб зростання часу роботи становить близько 12 %. А на порівняно невеликих обсягах даних для структурованих даних алгоритм працює утричі швидше порівняно із неструктурованими.

2. Кількість знайдених правил для неструктурованої інформації на близько 25 % менша, ніж для структурованої інформації.

3. Для коефіцієнтів підтримки та достовірності, вищих від середнього, для неструктурованих даних не знайдено жодного правила, а для структурованих результат – від 2 до 5 правил.

У [14] виконувався аналіз надзвичайних ситуацій у залізничному транспорті для пошуку асоціативних правил із використанням алгоритму Apriori. У цьому дослідженні запропоновано модифікацію алгоритму: розбиття множини всіх можливих характеристик на дві підмножини:

1. A – характеристики, сукупність яких могла стати причиною виникнення надзвичайних ситуацій.
2. B – характеристики, скупність яких описує надзвичайну ситуацію що виникла.

В результаті застосування цієї модифікації збільшується швидкість роботи

алгоритму.

З отриманих результатів дослідження видно що алгоритм є доволі дослідженим та схильним до модифікацій проте, його можна досліджувати для навіструктурованих даних або ввести нові модифікації для більш точної обробки інформації.

* 1. **Мета, об’єкт та предмет дослідження**

Мета роботи полягає в порівняльному аналізі роботи алгоритму Apriori. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил для бази даних із дорожньо-транспортними пригодами. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

* 1. **Висновок до розділу 1**

Накопичення інформації є актуальною темою сьогодення. Для аналізу цієї інформації використовують різноманітні технології та алгоритму. Після огляду декількох із них було вирішено провести дослідження для алгоритму Apriori.

Отже, у магістерській кваліфікаційній роботі необхідно розглянути алгоритм Apriori з точки зору пошуку асоціативних правил у великій кількості даних та проаналізувати затрачені обчислювальні ресурси. Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних.

Після аналізу існуючих досліджень, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

# РОЗДІЛ 2. Опис алгоритму apriori

1. 1. **Алгоритм apriori**

На даний момент існує декілька основних алгоритмів пошуку асоціативних правил в загальному вигляді, що не зав’язані на конкретизовану предметну область. Розглянемо найбільш поширені з них.

Алгоритм Apriori [15] призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює ітеративно, використовуючи пошук в ширину і принцип зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для вирішення задачі пошуку асоціативних правил.

Алгоритм Eclat [16] побудований на основі пошуку в глибину використовуючи перетин наборів елементів для знаходження частих наборів елементів. Це дуже простий алгоритм пошуку частого набору елементів. Цей алгоритм використовує вертикальну базу даних. Він не може використовувати горизонтальну базу даних. Якщо є горизонтальна база даних, то нам потрібно перетворити її у вертикальну базу даних. Не потрібно сканувати базу даних знову і знову. Алгоритм Eclat сканує базу даних лише один раз. Підтримка враховується в цьому алгоритмі. В цьому алгоритмі не враховується достовірність.

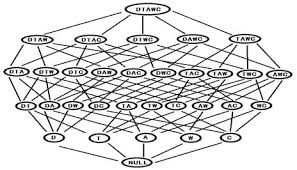


Рис. 2.1. Приклад роботи алгоритму Eclat

В загальному випадку алгоритм працює повільніше ніж алгоритм Apriori, однак в ситуації коли кількість різних елементів транзакцій є невеликою, він працює швидше.

Алгоритм FP-growth [17] (frequent pattern growth, зростання частих патернів) використовує розширене префіксне дерево для збереження бази даних у стислому вигляді. Цей алгоритм застосовує метод «розділяй і володарюй» для декомпозиції і видобутку знань із бази даних. В процесі роботи алгоритм використовує метод зростаючих патернів для уникнення ресурсоємкого процесу генерації частих кандидатів і їх тестування, що використовується в алгоритмі Apriori. Однак через це ускладнюється процес оптимізації алгоритму для використання з специфічними наборами даних, де покращення швидкості роботи можна досягти іншими методами.

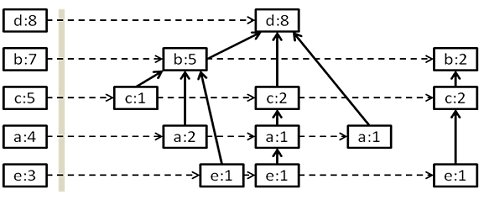


Рис. 2.2. Приклад роботи алгоритму FP-growth

Отже, враховуючи переваги та недоліки розглянутих алгоритмів пошуку асоціативних правил, було обрано алгоритм Apriori так як він є простим у розумінні та можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних.[23-29]

Таблиця 2.1.

Приклад набору транзакцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction ID** | **Цибуля** | **Картопля** | **Бургер** | **Молоко** | **Пиво** |
|  | * 1 | 1 | 1 | 0 | 0 |
| t_2 | 0 | 1 | 1 | 1 | 0 |
| t_3 | 0 | 0 | 0 | 1 | 1 |
| t_4 | 1 | 1 | 0 | 1 | 0 |
| t_5 | 1 | 1 | 1 | 0 | 1 |
| t_6 | 1 | 1 | 1 | 1 | 1 |

Ключовим поняттям в алгоритмі Апріорі є антимонотонність міри підтримки. Це передбачає що:

* Всі підмножини частої Набір повинні бути частими
* Подібним чином, для будь-якого незвичного набору, всі його суперсеті також повинні бути нечастими

Давайте розглянемо інтуїтивно зрозуміле пояснення алгоритму за допомогою наведеного вище прикладу(табл. 2). Перед початком процесу, встановіть порогової підтримки до 50%, тобто лише ті елементи значні, для яких підтримка більше 50%.

**Крок 1:** Створити таблицю частот для всіх елементів, що виникають у всіх транзакціях. Для нашого випадку:

Таблиця 2.2.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |
| Пиво(Be) | 2 |

**Крок 2:** Ми знаємо, що важливі лише ті елементи, для яких підтримка більше або дорівнює пороговій підтримці. Тут границя підтримки становить 50%, отже, лише ті елементи значущі, що трапляються у більш ніж трьох транзакціях, а такими є - Цибуля (O), Картопля (P), Бургер (B) і Молоко (M). Тому ми отримуємо наступну таблицю:

Таблиця 2.3.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |

**Крок 3:** Наступним кроком є зробити всі можливі пари значущих елементів, маючи на увазі, що порядок не має значення, тобто АB такий самий, як BА. Щоб зробити це, візьміть перший предмет і з'єднайте його з усіма іншими, такими як OP, OB, OM. Аналогічним чином розглянемо другий елемент і об'єднає його з попередніми пунктами, тобто PB, PM. Ми розглядаємо лише попередні пункти, оскільки PO (як OP) вже існує. Отже, всі пари в нашому прикладі - OP, OB, OM, PB, PM, BM.

**Крок 4:** Тепер ми розглянемо випадки кожної пари у всіх транзакціях.

Таблиця 2.4.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OP | 4 |
| OB | 3 |
| OM | 2 |
| PB | 4 |
| PM | 3 |
| BM | 2 |

**Крок 5.** Знову тільки ті набори наборів є значними, які перетинають порогову підтримку, а такі - OP, OB, PB та PM.

**Крок 6.** Тепер скажімо, ми хотіли б шукати набір з трьох елементів, які купуються разом. Ми будемо використовувати пункти, знайдені на кроці 5, і створимо набір з 3 елементів.

Щоб створити набір з 3 елементів, потрібно інше правило, яке називається самоз'єднання. Він говорить, що з пар OP, OB, PB та PM ми шукаємо дві пари з однаковою першою буквою, і об’єднуємо їх:

* OP та OB, це дає OPB
* PB і PM, це дає PBM

Далі ми знайдемо частоту для цих двох наборів.

Таблиця 2.5.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OPB | 4 |
| PBM | 3 |

Знову застосувавши правило порогу, ми виявили, що OPB є єдиним значним набором елементів.

Отже, набір з 3 предметів, який був придбаний найчастіше, - OPB.

Приклад, який ми розглянули, був досить простим, і видобуток частих предметів у товаристві зупинено на 3 пунктах, але на практиці існує кілька десятків предметів, і цей процес може тривати багато предметів. Припустимо, що ми отримали значні набори з трьома елементами як OPQ, OPR, OQR, OQS та PQR, і тепер ми хочемо створити набір з 4 елементів. Для цього ми розглянемо набори, які мають перші два алфавіти, тобто загалом

OPQ та OPR дають OPQR

OQR та OQS дають OQRS

Взагалі, ми повинні шукати набори, які тільки відрізняються в їхній останній букві / предметі.

Тепер, коли ми розглянули приклад функціональності алгоритму Апріорі, давайте сформулювати загальний процес.

Весь алгоритм можна розділити на два етапи:

**Крок 1:** застосуйте мінімальну підтримку, щоб знайти всі часто використовувані набори з елементами k у базі даних.

**Крок 2.** Використовуйте правило самопідключення, щоб знайти часті множини з елементами k + 1 за допомогою часто використовуваних k-елементів. Повторіть цей процес з k = 1 до моменту, коли нам не вдається застосувати правило самопідключення.

Такий підхід до розповсюдження частих елементів, що належить до одного, називається підходом "знизу вверх".

Cкладність цього алгоритму залежить від мінімального значення підтримки. Припустимо, що кількість вхідних транзакцій N, а кількість унікальних елементів R. Складність для генерації набору розміру i – О(Ri), де і – час для обчислення підтримки для кожного набору може бути O(N), у разі використання HashMap. Блок-схема алгоритму подано у Додатку Б.

* 1. **Видобування асоціативних правил**

До цих пір ми розглянули алгоритм Апріорі по відношенню до генеруванні частих наборів. Існує ще одне завдання, за допомогою якого ми можемо використовувати цей алгоритм, тобто ефективно знаходити асоціативні правила. [26-27]

Щоб знайти асоціативні правила, нам потрібно знайти всі правила, що мають підтримку, більшу за порогову підтримку та довіру, більшу за порогову довіру.

Але як ми знаходимо це? Одним із можливих шляхів – це перебір всіх можливих асоціативних правил та обчислення підтримки та впевненості для кожного правила. Потім просканувати правила, які не підтримують порогову підтримку та довіру. Але це обчислювально дуже складний і неефективний процес, оскільки кількість всіх можливих правил асоціації зростає експоненціально з кількістю предметів.

Враховуючи, що в наборі I є n пунктів, загальна кількість можливих правил асоціації становить 3 ^ n - 2 ^ (n +1) + 1.

Ми також можемо використовувати інший спосіб, який називається двоетапним підходом, для пошуку ефективних правил асоціації.

Двоступеневий підхід:

Крок 1: генерація частотних наборів елементів: знайдіть всі набори предметів, для яких підтримка перевищує порогову підтримку відповідно до процесу, який ми вже бачили раніше в цій статті.

Крок 2: генерація правил: створюйте правила з кожної частинки, використовуючи бінарний розділ часто використовуваних предметів і шукайте тих, що мають високу впевненість. Ці правила називаються правилами кандидатів.

Давайте розглянемо наш попередній приклад, щоб отримати ефективне правило об'єднання. Ми виявили, що OPB є частим комплектом. Тому для цієї проблеми, крок 1 вже зроблено. Отже, давайте розглянемо крок 2. Усі можливі правила, що використовують OPB, є:

OP\longrightarrowB, OB\longrightarrowP, PB\longrightarrowO, B\longrightarrow OP, P\longrightarrowOB, O\longrightarrowPB

Якщо X являє собою частий набір елементів з k-елементами, то існують правила асоціації кандидатів 2 ^ k-2.

Плюси алгоритму:

Легкий у реалізації та розумінні.

Може бути використаний для величезних даних.

Мінуси алгоритму:

Іноді, потрібно знайти величезну кількість правил кандидатів, що є обчислювально затратно.

Підрахунок підтримки також затратний по ресурсах, адже потрібно пройтись по всій базі.

* 1. **Опис відомих модифікацій**

Проаналізувавши суміжні дослідження було виявлено що вже існує ряд модифікацій для алгоритму apriori, які покращують його роботу.

PredictiveApriori - це алгоритм, який використовує більшу підтримку та обмінюється з більшою впевненістю, а також обчислює очікувану точність в байєсівських рамках. Результат цього алгоритму максимізує очікувану точність майбутніх даних правил асоціації. Цей алгоритм генерує правила асоціації як очікуване число правил користувача. Схеффер [13] визначає цей алгоритм за допомогою:

Нехай D - це база даних, індивідуальні записи r яких генеруються статичним процесом P, нехай X → Y є правилом асоціації. Точність прогнозування c (X → Y) = Pr (r, що задовольняє Y | r, задовольняє X) є умовною ймовірністю Y ⊆ r, якщо X ⊆ r, коли розподіл r регулюється P.

Правило додається, якщо: очікувана прогностична точність цього правила є одним з 'N' найкращих і не поглинена правилом, принаймні тієї ж очікуваної точності прогнозу. Якщо включені асоціативні правила видобуваються замість (загальних) асоціативних правил.

Якщо встановлено значення -1, останній атрибут буде взятий в якості атрибута класу та існує кількість правил для знаходження. Можливі пропущені значення, бінарні атрибути, порожні номінальні атрибути та номінальний клас.

Алгоритм AprioriTid є різновидом алгоритму Apriori. Цікава особливість полягає в тому, що база даних D не використовується для підрахунку підтримка після першого проходу. Кожен член комплекту | Ck | має форму <TID; {Xk}>, де кожний Xk потенційно великі k-елементи присутні в транзакції за ідентифікатором TID. Для k = 1, C1 відповідає базі даних D, хоча концептуально кожен елемент замінюється itemset {i}. При k> 1, | Ck | згенерований алгоритмом. Член Ck, що відповідає транзакції t, є <t.TID, {c∈ Ck | c міститься в t}>. Якщо транзакція не містить жодного кандидата k-елементів, а потім | Ck | не матиме запису для цього транзакція Таким чином, кількість записів у | Ck | може бути менше числа операції в базі даних, особливо для великих значень k. Також, для великих значення k, кожен запис може бути меншим, ніж відповідна транзакція, тому що, дуже мало кандидатів можуть бути включені в транзакцію. Однак для невеликих цінностей для k кожен запис може бути більшим, ніж відповідна транзакція, тому що запис у Ck включає всі k-множини кандидатів, що містяться в транзакції.

Кожен член безлічі Ck є парою виду , де кожен Fk є потенційно частим k - елементним набором, представленим в транзакції з ідентифікатором TID. Безліч C1 = D відповідає безлічі транзакцій, хоча кожен об'єкт в транзакції відповідає однооб'єктному набору в безлічі C1, що містить цей об'єкт. Для k> 1 безліч Ck генерується відповідно до алгоритму, описаним нижче.

Член безлічі Ck, відповідний транзакції Т, є парою такого вигляду:

 (2.1)

Підмножина наборів в Ck з однаковими TID (містяться в одній і тій же транзакції) називається записом. Якщо транзакція не містить ні одного k - елементного кандидата, то Ck не матиме записи для цієї транзакції Тобто кількість записів в Ck може бути менше, ніж в D, особливо для великих значень k. Крім того, для великих значень k кожна запис може бути менше, ніж відповідна їй транзакція, т. К. В транзакції буде міститися мало кандидатів. Однак для малих знань k кожен запис може бути більше, ніж відповідна транзакція, тобто Ck включає всіх кандидатів k-елементних наборів, що містяться в транзакції [13].

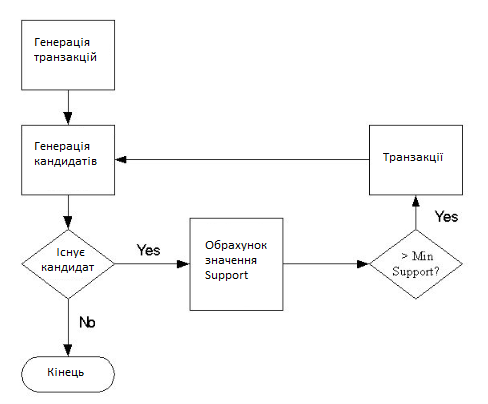


Рис. 2.1. Блок-схема роботи алгоритму AprioriTID

Інший вид алгоритму Apriori - алгоритм MSAP (Mining Sequential Alarm Patterns), спеціально розроблений для проведення послідовного аналізу мережних телекомунікаційних збоїв. Він підтримує такі функції: для будь-якої послідовності Lk його підтримка буде меншою, ніж підтримуючі послідовності з набору Lk1. Алгоритм MSAP використовує концепцію "Urgent Window" для пошуку подій, які слідують один за одним. Це дозволяє виявити не тільки однакові послідовності подій, але й наступні послідовності подій. В іншому випадку цей алгоритм працює на тому ж принципі, що й Апріорі.

* 1. **Опис можливих модифікацій**

Отже, переглянувши декілька відомих модифікацій, проаналізувавши предметну область та зрозумівши принцип роботи алгоритму та пошуку асоціативних правил було запропоновано модифікації для алгоритму.

Відомо, що алгоритм на кожному кроці визначає:

* підтримку (англ. support) - скільки разів у всьому масиві використано елементи даних, що складаються з X та Y
* достовірність (англ. confidence) - який відсоток від всіх одиниць, що містять X, містить також і Y

Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсіканання абсурдних або випадкових груп правил, оттже швидкість алгоритму повинна зрости адже не потрібно здійснюватинаступні кроки по вже відкинутих правилах.

Цей підхід можна спробувати для вище перелічених методів, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Ще одним із можливих варіантів модифікації розбиття множини всіх можливих транспортних пригод I = {Is, Id} на дві підмножини:

* Is – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами (стан алкогольного сп’яніння, технічний стан автомобіля, вік водія, вік авто тощо).
* Id – множина характеристик, дані якої для нашого випадку будуть цікавими (місце проживання – село/місто, освіта, колір машини, кількість пасажирів, порядок власника авто тощо).

Асоціативні правила, що містять лише підмножину факторів із множини Si , не містять\* корисної інформації для виконання подальшого аналізу, завдяки чому можна прискорити алгоритм пошуку асоціативних правил. Для цього необхідно змінити процес генерації наборів кандидатів Ck , в якому відсікати набори, що не містять факторів із множини Di. В результаті значно зменшиться кількість наборів кандидатів Ck , що генеруватиметься на кожній ітерації алгоритму, а також це впливатиме на генерацію частих наборів Lk. Час обробки множини наборів Ck та генерації Lk лінійно залежить від потужності даних множин і загальної потужності множини І, яка є сталою в процесі роботи алгоритму. Тому завдяки зменшенню кількості елементів множини Ck зменшиться загальний час роботи алгоритму.

* 1. **Висновок до розділу 2**

Здійснено обгрунтування та порівняльний аналіз алгоритму, визначено що алгоритм є простий у розумінні та схильний до модифікацій.

Запропоновано покращення базового алгоритму, за допомогою застосування характиристики Ліфту на кожному кроці алгоритму. Даний підхід зменшить витрати часу при знаходженні асоціативних правил. Також є можливим впровадження даного підходу що уже відомих модифікацій, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Також запропоновано ще один варіант у якому вхідна множина розділяється на дві підмножини, перша множина це множина із факторами які мали місце до виникнення асоціацій і друга – множина із факторами що стали результатом виникнення асоціацій, на кожному кроці відсікається перша множина, адже вона не містить корисної інформаці.

# Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori



## Загальна постановка задачі

Реалізувати програмний засіб, за допомогою якого можна здійснювати інтелектуальний аналіз даних, а саме пошук асоціативних правил. Даний програмний засіб повинен надавати можливість для проведення аналізу вхідних даних та виведення результатів аналізу (знайдені асоціативні правила, їх кількість та швидкодію). Після виконання програма повинна формувати звітність на основі результатів аналізу представлених даних. Результати представлятимуться безпосередньо на екрані із можливістю збереження їх у файл.

Провести дослідження ефективності алгоритму. На основі отриманих даних із експериментів визначити можливість видозміни алгоритму для збільшення ефективності його роботи. При позитивних результатах виконати модифікацію алгоритму.

## Специфікація вимог

**3.2.1. Вступ**

Призначення даного продукту – автоматичний аналіз великих наборів даних та пошук в них асоціативних правил, формування звіту.

Мета – реалізувати класичний алгоритм apriori та модифіковані його версії та порівняти результати роботи у вигляді програмного продуку.

**3.2.2. Загальний опис**

**3.2.2.1. Характеристики продукту**

Функції, що будуть реалізовані у програмі:

* Зчитування даних з файлу;
* Аналіз даних;
* Виведення результатів у вигляді таблиць та діаграм;
* Формування звіту про результати аналізу;

**3.2.2.2. Класи користувачів та їх характеристики**

В даній програмі передбачено лише один клас користувачів, тому функціонал не буде мати обмеження.

Програмний засіб призначений для бізнес аналітиків, щоб ефективно провести інтелектуальний аналіз даних та сформувати звіт замовнику.

**3.2.2.3. Середовище функціонування**

Апаратні вимоги:

* Частота процесора 1 GHz;
* Оперативна пам’ять 512 Mb;
* 10 Mb вільного простору на диску.

Системні вимоги:

* Операційна система Microsoft Windows XP або вище;
* .Net Framework версії 4.0 і вище;

**3.2.3. Характеристики системи**

**3.2.3.1. Пошук асоціативних правил для введених даних**

3.2.3.1.1. Опис і пріоритет

Аналіз введених даних, пошук залежностей. Виведення результатів тестування у зручному для користувача вигляді, у вигляді таблиць та діаграм, із можливістю збереження на локальний диск.

Пріоритет – високий.

3.2.3.1.2. Послідовності дія/відгук

Користувач вибирає текстовий документ із даними, аналіз яких потрібно провести. Далі завантажує список характеристик написнувши на кнопку «Завантажити I». Потім задає коефіцієнти підтримки, достовірності та ліфта, при потребі завантажує характеристики що описують ДТП. Натискає на кнопку «Розпочати» і у низу вікна з’являється анімація завантаження.

3.2.3.1.3. Функціональні вимоги

REQ-1.1: Зчитування даних із .txt формата;

REQ-1.2: Проведення аналізу;

REQ-1.3: Пошук асоціативних правил;

REQ-1.4: Виведення повідомлення про закінчення проведення аналізу;

REQ-1.5: Виведення результатів;

**3.2.3.2. Формування звіту результату**

3.2.3.2.1. Опис і пріоритет

Створення звіту, який буде відображати інформацію про результат пошуку залежностей. Звіт можна зберегти на диск у форматі txt.

3.2.3.2.2. Послідовності дія/відгук

Для збереження звіту потрібно натиснути кнопку «Зберегти звіт», після чого появиться діалогове вікно для вибору місця збереження файлу.

3.2.3.2.3. Функціональні вимоги

REQ-2.1: Формування звіту;

REQ-2.2: Збереження на диску у форматі txt;

**3.2.3.3. Перегляд результатів виконання**

3.2.3.3.1. Опис і пріоритет

Перегляд результатів виконання програми у вигляді діаграм та таблиці

3.2.3.3.2. Послідовності дія/відгук

Для перегляду результатів потрібно натиснути кнопку «Результати», після чого появиться нове вікно із результатами виконання.

3.2.3.3.3. Функціональні вимоги

REQ-3.1: Формування звіту;

REQ-3.2: Збереження на диску у форматі txt;

**3.2.4. Вимоги зовнішніх інтерфейсів**

**3.2.4.1. Користувацькі інтерфейси**

Інтерфейс програми повинен бути простим для розуміння, забезпечувати швидкодію відповідно до вимог та дозволяти зручно виконувати необхідні операції.

**3.2.4.2. Програмні інтерфейси**

* Бібліотека Microsoft.Office.Interop.Excel;
* Бібліотека Xml.Serializer;

**3.2.5. Інші нефункційні вимоги**

**3.2.5.1. Вимоги продуктивності**

Швидкість проведення аналізу буде прямо залежати від кількості завантажених даних, коефіцієнтів роботи алгоритму та від потужності процесора.

**3.2.5.2. Вимоги безпеки**

Немає вимог безпеки.

**3.2.5.3. Атрибути якості програмного продукту**

Програма реалізована із мінімальним основним набором функції, саме тому вона зрозуміла та зручна для використання користувачам, які з нею ніколи не працювали;

Ніяким чином не впливає на роботу інших програм.

**3.2.5.4. Інші вимоги**

Необхідна наявність .Net Framework версії 4.0 і вище.

## Використані технології

Оскільки основним завданням проекту буде аналіз роботи алгоритму то було обрано технологію .Net, середовище розробки Microsoft Visual Studio, мова C# та WPF[18].

Для зберігання поточного стану розробленої системи використано систему контролю версій Git - це система керування версіями для відстеження змін у комп'ютерних файлах та координаціїіроботи над цими файлами серед декількох людей. Він використовується, перш за все, для управління вихідним кодом у розробці програмного забезпечення [18], але йогоіможнаівикористовувати для відстеження змініу будь-якомуінаборі файлів. В якості розподіленої системи перегляду-контролю вонаіспрямованаіна швидкість, цілісністьіданих та підтримку розподілених, нелінійних робочих процесів.

.NET Framework є основою програмного забезпечення, розроблена корпорацією≥Майкрософт, яка працює≥в основному на Microsoft Windows. Вона містить велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує мовну сумісність (кожна мова може використовувати код, написаний на інших мовах) на кілька≥мов програмування. Програми, написані для .NET Framework виконуються в програмному≥середовищі (на відміну від апаратної середовища) під≥назвою Common≥Language Runtime (CLR), віртуальна≥машина додаток, яке надає≥такі послуги, як безпека, управління≥пам'яттю і обробки винятків.

Microsoft Visual Studio є інтегрованим середовищем розробки Microsoft (IDE). Він використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи для розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Він може створювати як власний код, так і керований код.[18]

Visual Studio включає редактор коду, що підтримує IntelliSense, а також рефакторинг коду. Вбудований відладчик працює як відладчик на рівні вихідного рівня та налагоджувача на рівні машини. Інші вбудовані інструменти включають програму для кодування, дизайнеру форми для побудови графічних інтерфейсів, веб-дизайнера, дизайнера класу та дизайнера схеми баз даних. Він приймає плагіни, які покращують функціональність практично на всіх рівнях, включаючи підтримку систем управління ресурсами (наприклад, Subversion та Gіt) та додавання нових наборів інструментів, таких як редактори та візуальні розробники для мов або наборів інструментів для інших аспектів розробки програмного забезпечення життєвий цикл.[18-19]

C # — є об’єктно-орієнтованою мовою≥програмування≥яка охоплює строгу типізацію, імперативність, декларативність, функціональні, загальні, об'єктно-орієнтовані (на основі класів), компонент-орієнтоване≥програмування дисциплін. C # є одним з мов програмування, призначених для Common Language Infrastructure.

Мова призначена як проста, сучасна, загальноприйнята, об'єктно-орієнтована мова програмування.

Мова та її реалізація повинні забезпечувати підтримку принципів розробки програмного забезпечення, таких як перевірка сильного типу, перевірка кордону, виявлення спроб використання неініціалізованих змінних та автоматичного збирання сміття. Важливою є надійність програмного забезпечення, надійність і продуктивність програміста.[19]

Мова призначений для розробки програмних компонентів, які підходять для розгортання в розподілених середовищах.

Переносимість дуже важлива для вихідного коду та програмістів, особливо тих, хто вже знайомий з C і C ++.

Інтернаціоналізаційна підтримка дуже важлива.

C # призначений для написання програм для хостингу та вбудованих систем, починаючи від дуже великих, використовуючи складні операційні системи, до дуже малих спеціальних функцій.

Хоча додатки C # є економічними з точки зору вимог до пам'яті та керування енергією, мова не мав на меті прямо конкурувати з продуктивністю чи розміром з C або асемблером.

На відміну від C ++, C # не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було рішення дизайнера провідного архітектора мови, щоб уникнути ускладнень та спростити архітектурні вимоги в рамках CLI. При виконанні декількох інтерфейсів, що містять метод з тим самим підписом, C # дозволяє реалізувати кожен метод залежно від того, який інтерфейс цей метод називається, або, наприклад, Java, дозволяє реалізувати цей метод один раз, і це єдиний виклик до дзвінка через будь-який інтерфейс класу.

C #, покажчики адрес пам'яті можуть використовуватися лише в блоках, спеціально позначених як небезпечні, а програми з небезпечним кодом потребують відповідних дозволів для запуску. Більшість доступ до об'єкта здійснюється через безпечні посилання на об'єкти, які завжди вказують на "живий" об'єкт або мають чітко визначене нульове значення; неможливо отримати посилання на "мертвий" об'єкт (той, який був зібраний сміттям) або випадковий блок пам'яті. Небезпечний покажчик може вказувати на екземпляр типу-значення, масиву, рядка або блоку пам'яті, виділеного стеку. Код, який не позначено як «небезпечний», може зберігати і керувати покажчиками через тип System.IntPtr, але він не може їх вилучити.

Неможливо явним чином звільнити керовану пам'ять; замість цього автоматично збирають сміття. Мусорна колекція вирішує проблему витоків пам'яті, звільняючи програміста від відповідальності за випуск пам'яті, яка більше не потрібна.

Для малювання діаграм використано бібліотеку DataVisualization, яка дозволяє легко та прозоро малювати різного роду діаграми.

Windows Presentation Foundation (WPF) - це Microsoft графічна підсистема для відображення користувацьких інтерфейсів у додатках Windows. WPF, раніше відомий як "Avalon", спочатку був випущений в рамках .NET Framework 3.0 у 2006 році. WPF використовує DirectX і намагається забезпечити послідовну модель програмування для створення додатків. Він відокремлює інтерфейс користувача від бізнес-логіки і нагадує подібні XML-орієнтовані об'єктні моделі, такі як ті, що реалізовані в XUL та SVG.[19]

WPF використовує XAML, мову на базі XML, для визначення та поєднання різних елементів інтерфейсу. Програми WPF можуть бути розгорнуті як окремі настільні програми або розміщені як вбудований об'єкт на веб-сайті. WPF має на меті об'єднати ряд загальних елементів користувальницького інтерфейсу, таких як 2D / 3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка, анімація часу виконання та попередньо відтворені носії. Потім ці елементи можуть бути пов'язаними та маніпулювати на основі різних подій, взаємодії користувачів та прив'язування даних.

Бібліотеки бібліотеки WPF входять до всіх версій Microsoft Windows, починаючи з Windows Vista та Windows Server 2008. Користувачі Windows XP SP2 / SP3 та Windows Server 2003 можуть додатково встановлювати необхідні бібліотеки.

## Реалізація інтерфейсу користувача програмного засобу для роботи алгоритму apriori

Розроблений інтерфейс користувача складається із мінімального набору компонентів, що робить його дуже зручним та простим у користуванні, в той самий час, даний інтерфейстдаєтзмогу користувачу в повній мірітзастосувати весь функціоналтрозробленого засобу.

Інтерфейс користувача складається з основного вікна та додаткового (див. Рис. 3.1. та Рис. 3.2.). Основне вікно застосунку дозволяє користувачу завантажити список транзакцій(список із ДТП), завантажити характеристики ДТП, завантажити характеристики що описують виникнення ДТП, а також встановити коефіцієнти підтримки достовірності та ліфта. Також, на головному вікні користувач має можливість почати процес аналізу великих даних алгоритмом apriori, та побачити результати виконання алгоритму результати (кількість знайдених правил чис виконання, кількість транзакій тощо.).

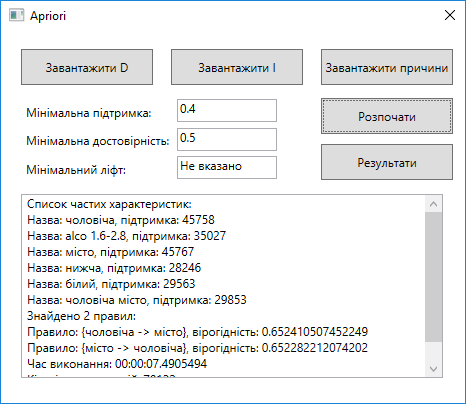


Рис. 3.1. Користувацький інтерфейс основного вікна програми

Вікно «Результати» відповідає за відображення результатів програми та порівняння їх із попередніми значеннями. Дане вікно складається дане вікно складається із трьох діаграм, таблиці та кнопки «Зберегти»(Рис. 3.2.). Таблиця представляє список із результатами виконання програми та такими колонками як: «Спроба №» - показує номер спроби, «Час виконання» - час виконання анлізу для спроби, «Кількість транзакцій» - кількість транзакцій у початковому списку, «Кількість правил» - кількість знайдених асоціативних правил.

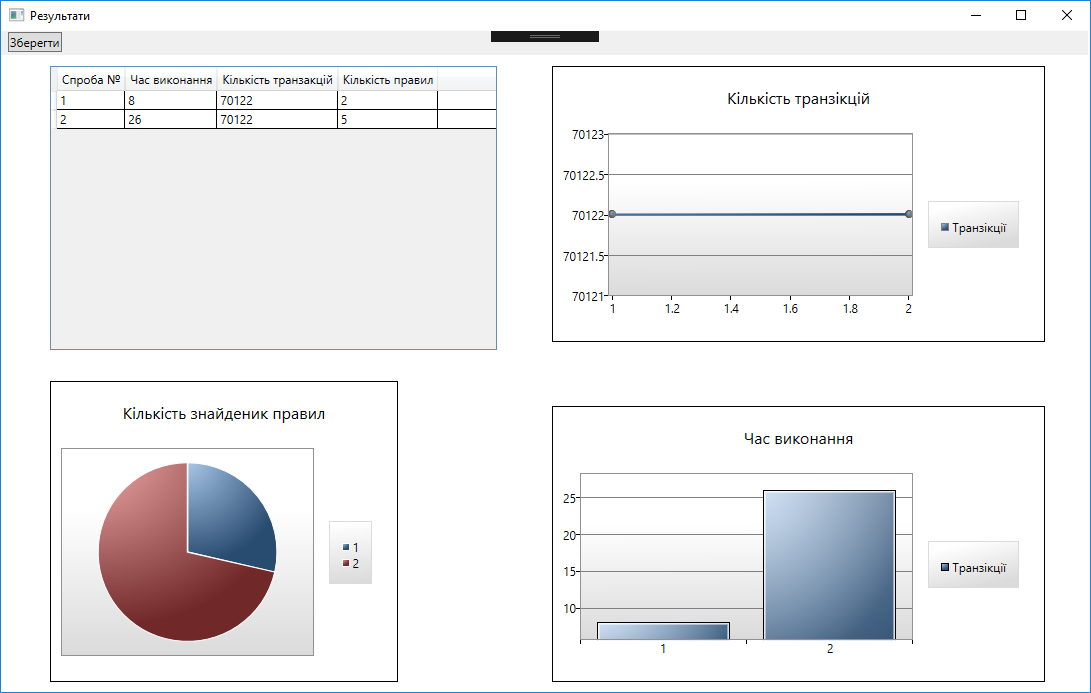


Рис. 3.2. Користувацький інтерфейс із результатами виконання програми

Таблиця «Кількість знайдених правил» показує відношення у відсотках скільки правил було знайдено для кожної спроби.

Таблиця «Кількість транзакцій» показує скільки транзакцій було в початковій базі даних.

Таблиця «Час виконання» показує час виконання для кожної спроби.

Якщо користувачцвводить неправильне значення змінної (наприклад, для змінної типу int – стрічку), програмацнецдасть змоги зберегтиезміни і попередить проецеекористувача(Рис. 3.3.).

Отже розглянувши всі ці результати можна легко зробити висновки про результати роботи алгоритму як для мофікованих режимів так і для стандартного вигляду.

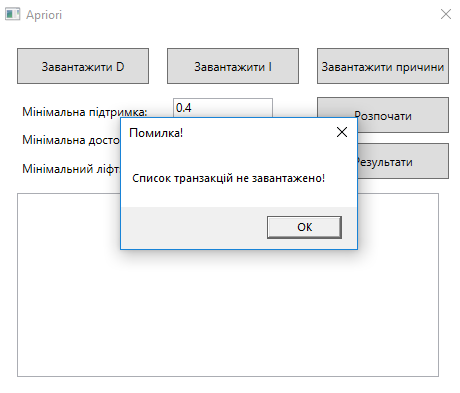


Рис. 3.3. Інтерфейс користувача, при неправильному введені даних.

## Розробка архітектури системи для роботи алгоритму apriori

Створення програми відбувається згідно з розробленою діаграмою класів програми. Розглянемо складові діаграми, щоб детальніше зрозуміти роботу програми:

* FileReader відповідає за зчитування даних із файлів та конвертання їх у масив. Після відкриття файлу з’являється діалогове вікно для вибору .txt файлу.
* Клас Item містить інформацію про кожен предмет який відповідає вравилам.
* Rule містить інформацію про правило X – лівий операнд, Y правий операнд, та Confidence – вірогідність.
* Клас Output – містить інформацію про результат роботи алгоритму.
* Основна логіка виконується а класі Apriori – клас який асимілює всі перелічені вище класи та має публічний метод ProcessTransaction, який на вхід отримує вхідні дані: масив «items», масив «transactions», масив «itemsD» мінімальне значення вірогідності, підтримки та ліфта. Та після обробки повертає результат типу Output.



Рис. 3.4. UML діаграма класів програми

* AprioriInThread – клас який забезпечує виконання алгоритму в окремому потоці щоб не блокувати користувацький інтерфейс, містить один метод «ProcessInThread» який повертає результат Task<Output>.
* Клас Result містить інформацію яка необхідна для побудови графіків та списку у вікні з результатами.

Також було розроблено UML діаграму прецедентів(Рис. 3.5) на якій зображено відношення акторів та прецедентів в системі.



Рис. 3.5. UML діаграма прецедентів

## Реалізація алгоритму apriori для аналізу ДТП

Алгоритм apriori було реалізовано згідно наведеної блок-схеми (Див. Додаток Б)

Список транзакцій завантажується із .txt файла та обробляється до потрібного вигляду, це двовимірний масив стрічок, за допомогою класу FileReader. Далі за таким самими принципом завантажується список характеристик та наслідків.

Список транзакцій разом із коефіцієнтами подається на вхід функції ProcessTransactions. Всередині цієї функції відбувається покрокова генерація кандидатів та частих наборів.

Під час генерації кандидатів обраховується їхня кількість відносно всіх транзакцій, та якщо використовується модифікована версія із завантаженими причинами то відсікаються кандидати у яких немає цих причин, завдяки такому підходу на кожній ітерації вдається відсікнути багато зайвих кандидатів і тому виконання алгоритму стає набагато швидшим.

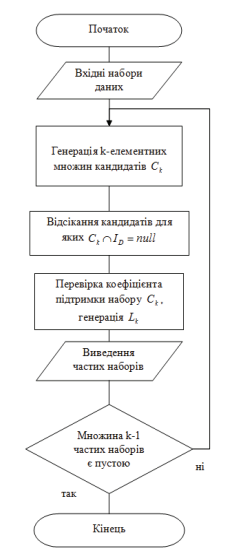


Рис. 3.6. Блок-схема модифікованого алгоритму apriori

Під час генерації частих наборів обраховується «підтримка» та порівнюється із мінімально допустимою, якщо ця умова виконується кандидат додається до частих наборів.

Усе це повторяється поки множина кандидатів не буде порожньою.

Після цього відбувається генерація асоціативних правил на основі частих наборів. Далі відбувається генерація «міцних» правил в методі GetStrongRules, де відбувається обрахунок вірогідності, та якщо використовується модифікована версія обраховується значення ліфта, такий підхід дозволяє відсікати абсурдні або випадкові правила та дозволяє збільшити точність виконання алгоритму.

Далі метод ProcessTransactions повертає список Output об’єктів з яких в подальшому можна згенерувати звіт та порівняльні діаграми.

## Висновки до розділу 3

Ключовим завданням постає розробка програмного засобу для проведення аналізу даних алгоритмом apriori. По суті програма буде працювати у певній логічній послідовності, отже варіантів дій користувача не так і багато, що робить користування просте та зрозуміле. Користувачу представляється лише вибір даних, який слід аналізувати і результати для подальшого використання.

Так як ціль проекту включає в собі інтелектуальний аналіз даних то технологія Microsoft .NET чудово підходить для написання прикладної програми. Для побудови інтерфейсу використовуватиметься технологія WPF, яка дозволить без проблем створити цікавий вигляд програми.

У цьому розділі детально описано специфікацію вимог, реалізовано інтерфейс, архітектуру системи та реалізовано алгоритм.

# Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ

## 4.1. Опис процесу дослідження

Поданий алгоритм apriori використовується для аналізу дорожньо-транспортних пригод, а саме пошуку асоціативних зв’зків між наборами даних які стали причинами ДТП.

При застосуванні пошуку асоціативних правил для аналізу ДТП виникають проблеми:

1. Збільшення потужності множини всіх можливих ДТП.
2. Збільшення часу для аналізу даних.
3. Низька інформаційність отриманого результату.

Запропоновано наступні модифікації для розв'язання цих проблем:

1. Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини:[3]

* *Is* – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами (стан алкогольного сп’яніння, технічний стан автомобіля, вік водія, вік авто тощо).
* *Id* – множина характеристик, дані якої для нашого випадку будуть цікавими (місце проживання – село/місто, освіта, колір машини, кількість пасажирів, порядок власника авто тощо).

Асоціативні правила, що містять лише підмножину із множини Is не містять корисної інформації, тому на кожному кроці після генерації кандидатів можна відсікати кандидатів, для яких Ck ∩ Id = null.

У результаті застосування такого підходу збільшується швидкість роботи алгоритму та ступінь інформаційності отриманих асоціативних правил.

1. Додати ще одну характеристику асоціативних правил Ліфт (англ. lift) - це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому.[4] Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил, отже швидкість алгоритму повинна зрости адже не потрібно здійснювати наступні кроки по вже відкинутих правилах.

Отже, для якісної оцінки запропонованих модифікацій потрібно

згенерувати базу даних із причинами які спровокували виникнення дорожньо-транспортних пригод.

Перш за все, потрібно здійснити аналіз стандартного алгоритму без модифікацій.

База даних складається із таких характеристик:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Порядок власності (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);
* Стать (чоловіча, жіноча);

Другим етапом дослідження є порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Третім етапом буде порівняння роботи стандартного алгоритму із запропонованою другою модифікацією.

На наступному етапі можна порівняти між собою три попередні варіанти.

На завершення потрібно підібрати найкращі варіанти коефіцієнтів алгоритму, які б показували найкращі результати в швидкодії та точності пошуку асоціативних правил.

## 4.2. Аналіз ароботи алгоритму без модифікацій

Після генерації даних було здійснено аналіз роботи алгоритму без модифікацій.

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori без додаткових модифікацій для дорожньо-транспортних пригод наведені в Таблиці 4.1.

*Таблиця 4.1.*

Результати виконання розробленого засобу

|  |  |
| --- | --- |
| Номер досліду | Час виконання, мс |
| 1 | 40 409 |
| 2 | 40 407 |
| 3 | 42 354 |
| 4 | 45 373 |
| 5 | 41 295 |
| 6 | 45 573 |
| 7 | 43 401 |
| 8 | 42 563 |
| 9 | 45 765 |
| 10 | 43 642 |

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 42 878.2мс.

Також, аплікація згенерувала графік на якому видно, що дані підтверджено. Результат представлений на рис. 4.1.



Рис. 4.1. Графік із зображенням відношення часу виконання до кожної транзакції

Таким чином, дані які представлено вище можна вважати еталоном для нашого дослідження, так як це результати роботи алгоритму без будь яких додаткових втручань.

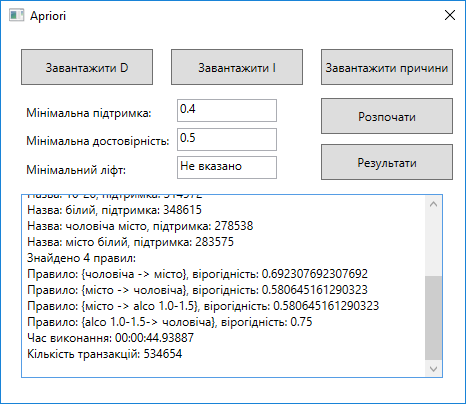


Рис. 4.2. Знімок результатів виконання програми

Отже, середнє значенням часу виконання алгоритму дорівнює 42 878.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила таких як:

{чоловіча –> місто}, вірогідність: 0.692307692307692

{місто –> чоловіча}, вірогідність: 0.580645161290323

{місто –> alco 1.0-1.5}, вірогідність: 0.580645161290323

{alco 1.0-1.5 –> чоловіча}, вірогідність: 0.75

Також знайдено 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

## 4.3. Аналіз роботи алгоритму із модифікаціями

Для початку дослідимо роботу алгоритму для модифікації №1 – Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Для множини Id – множина із характеристиками що описують ДТП, було обрано такі категорії:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Кількість власників (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Стать (чоловіча, жіноча);

Для множини Is – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами

* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 38 178.2мс.

Також, результати можна переглянути на графіку. Результат представлений на рис. 4.3.

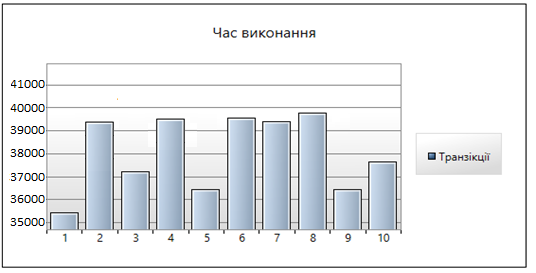


Рис. 4.3. Графік із зображенням відношення часу виконання до кожної транзакції

Середнє значенням часу виконання модифікованого алгоритму №1 дорівнює 38 178.2мс, для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

Отже, можна побачити, що час роботи алгоритму скоротився на 11% при цьому кількість знайдених частих наборів і «міцних» правил не зменшилась.

Приступимо до аналізу роботи алгоритму для модифікації №2 – додавання ще однієї характеристику асоціативних правил Ліфта.Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил. Проте перш ніж здійснити аналіз потрібно уточнити що застосувати Ліфт на етапі генерації кандидатів не вийшло так як логіка алгоритму не дозволяє цього зробити, ліфт можна застосувати при генерації асоціативних правил.

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 43 132.2мс.

Як видно із результатів виконання алгоритму дані не сильно відрізняються від еталонних(результати які було отримано із базового алгоритму) в деяких випадках результати навіть гірші. На практиці додавання характеристики Ліфта до роботи алгоритму нічого не або навіть погіршило результати. Проте це тільки для нашого конкретного випадку і бази даних із дорожнього-транспортними пригодами можливо приінших випадках результатит були б інакшими.

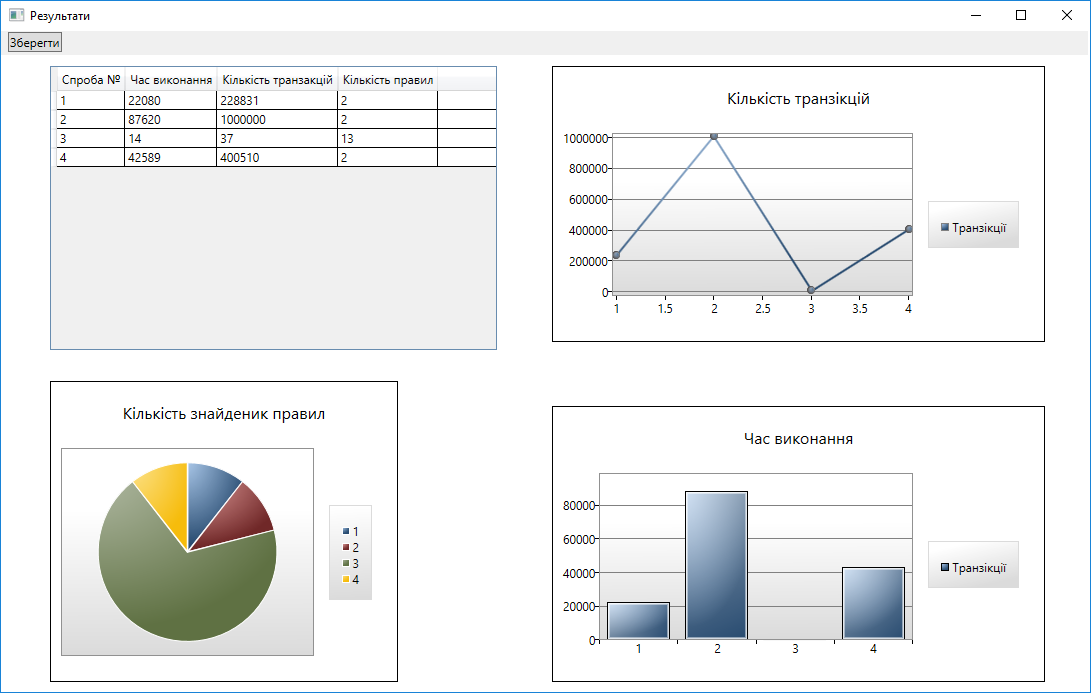


Рис. 4.5. Результати виконання програми

Отже, середнє значенням часу виконання алгоритму дорівнює 43 132.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 мінімальне значення вірогідності 0.5 та мінімальне значення Ліфта дорівнює 0.9.

Після генерації іншої бази даних було проведено додакове дослідження для впливу Ліфта на результати аналізу.

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori із модифікацією №2 для дорожньо-транспортних пригод наведені в Таблиці 4.2.

*Таблиця 4.2.*

Результати виконання розробленого засобу

|  |  |  |
| --- | --- | --- |
| Номер досліду | Кількість знайдених правил | Значення Ліфта |
| 1 | 75 | Не вказано |
| 2 | 64 | 0.95 |
| 3 | 56 | 1.05 |
| 4 | 45 | 1.2 |
| 5 | 35 | 1.4 |

Як видно із результатів виконання алгоритму зі збільшенням значення Ліфта деякі збільшується відсікання знайдених правил із цього виходить підтвердження теоретичних міркувань, що при додаванні значення Ліфта багато абсурдних або випадкових правил будуть відкинуті.

Так як, дослідження проводилось на невеликій базі із кількістю транзакцій 70 одиниць, час не було враховано в цьому дослідженні.

Було здійсненно порівняльний аналіз для різних значень коефіцієнтів

Результати роботи алгоритму apriori для різних значень коефіцієнтів наведені в Таблиці 4.3.

*Таблиця 4.3.*

Результати виконання розробленого засобу

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер досліду | Час виконання, мс | Кількість знайдених правил | Значення підтримки | Значення достовірності | Значення Ліфта | Причини завантажено |
| 1 | 38784 | 2 | 0,4 | 0,5 | Не вказано | Ні |
| 2 | 25751 | 2 | 0.4 | 0,5 | Не вказано | Так |
| 3 | 36118 | 15 | 0,3 | 0,5 | Не вказано | Так |
| 4 | 38118 | 18 | 0,3 | 0,5 | Не вказано | Ні |
| 5 | 26680 | 4 | 0,35 | 0,5 | Не вказано | Так |
| 6 | 26778 | 2 | 0,35 | 0,5 | 1 | Так |
| 7 | 30564 | 6 | 0,3 | 0,6 | Не вказано | Так |

Усі вище перелічені досліди було проведено для однієї і тієї ж бази розміром 530000 транзакцій. Отже, проаналізувавши результати можна зробити висновок, що найкращим результатом по швидкодії є варіанті із застосування методу модифікації №2 та із коефіцієнтами: достовірність 0,5 , підтримки 0,4. Найточніший варіант, коли було знайдено найбільше правил це варіант базового алгоритму та коефіцієнтами: достовірність 0,5 , підтримки 0,3.

## 4.4. Висновки до розділу 4

У цьому розділі було проведено дослідження алгоритму Apriori та його модифікацій. Дослідження здійснено в такій послідовності: перш за все, було проведено аналіз базового алгоритму без модифікацій, далі було проведено порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини, третім етапом було порівняння роботи стандартного алгоритму із запропонованою другою модифікацією. Останнім етапом було порівняння між собою трьох попередніх варіантів.

Отже, після здійснення досліджень можна сказати що запропонований варіанти модифікації №1 є найкращим у нашому випадку, за цим методом далось скоротити час виконання алгоритму на 11%, при цьому не втративши жодної важливої інформації.

Варіант модифікації №2 не вдалось реалізувати як було описано в теорії, тому що це руйнує логіку алгоритму, проте можна застосувати при генерації асоціативних зв’язків але результати для нашого випадку, аналіз ДТП, були гіршими за еталон.

# Розділ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ПРОДУКТУ

## 5.1. Економічна характеристика проектного рішення

Метою дипломного проекту є розробка засобів для аналізу велики за обсягом даних. Кінцевим продуктом даного проекту, є алгоритм apriori його модифікації та програмна реалізація. Призначенням цієї системи є допомога бізнес аналітиками швидко та коректно аналізувати дані та знаходити якусь корисну інформацію.

Розроблена система створена як застосунок для комп’ютера. Дана система володіє наступним рядом переваг:

* зменшення часу для проведення аналізу даних алгоритом apriori;
* збільшення точності пошуку асоціативних правил;

Новизною даного проекту є те, що розроблений засіб дозволяє проводити аналіз великих даних за допомогою алгоритму apriori набагато швидше і знаходити більше асоціативних правил. Такий підхід робить дану систему особливою, аналогів якій на ринку немає.

Існуючи засоби для проведення інтелектуального аналізу даних:

1. Cluvio - це сучасна платформа для аналізу даних, яка дозволяє виконувати SQL запити до вашої бази даних, обробляти дані в R, візуалізувати результати та створювати красиві, інтерактивні інформаційні панелі за лічені хвилини. Cluvio також підтримує потужне вставлення, яке дозволяє вам легко додавати аналітичні функції на будь-який веб-сайт або веб-додаток.
2. Qlik Sense Desktop - це безкоштовний продукт, який дозволяє створювати інтерактивні звіти та інформаційні панелі з приголомшливими діаграмами та діаграмами. Програмне забезпечення для візуалізації спрощує аналіз даних і допомагає приймати інформовані бізнес-рішення швидше, ніж будь-коли раніше. Перетворення електронних таблиць у чіткіші візуалізації, що робить процес аналізу простішим та швидшим для перегляду всіх користувачів.
3. Програмне забезпечення IBM SPSS Statistics може допомогти вам знайти нові відносини в даних і прогнозувати, що, швидше за все, відбудеться далі. Перегляньте демо-версію безкоштовної статистики від IBM, щоб дізнатися, як легко отримати доступ до даних, керувати ними та аналізувати їх - без попередньої статистики; практично ліквідувати довгострокову підготовку даних; і швидко створювати, маніпулювати та розповсюджувати статистику для прийняття рішень.

Розроблена система є актуальною, оскільки володіє рядом переваг відносно існуючих засобів аналізу даних, що дозволяє провести аналіз даних більш якісно.

В даному розділі дипломного проекту проводиться економічне обґрунтування проектування та розробки засобу для проведення аналізу великих даних. Зокрема розраховується комплексний показник якості проектного рішення, котрий відображає його переваги та недоліки.

В економічній частині дипломного проекту відображено доцільність проектного рішення через відповідні економічні розрахунки, проведено техніко-економічне обґрунтування розробки.

## 5.2. Розрахунок витрат на розробку та впровадження проектного рішення

1) Витрати на розробку і впровадження програмного засобу (К) визначаються за формулою 5.1:

 (5.1)

де *К1*– витрати на розробку програмного засобу, грн.;

*К2* – витрати на відлагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу охоплюють:

1. витрати на оплату праці розробників (*Воп*);
2. єдиний соціальний внесок (*Вєсв*);
3. вартість додаткових виробів (матеріалів), що закуповуються (*Вд*);
4. транспортно-заготівельні витрати (*Втр*);
5. витрати на придбання спецобладнання (*Всо*);
6. накладні витрати (*Вн*);
7. інші витрати (*Він*).

Середньоденна заробітна плата і-го розробника (*ЗПДі*) обчислюється за формулою (5.2):

 (5.2)

де *ЗПі* – основна місячна заробітна плата розробника і-ої спеціальності, грн.;

*Фм* – місячний фонд робочого часу, днів.

Розрахунок витрат на оплату праці усіх розробників проекту обчислюємо за формулою (5.3):

 (5.3)

де *ni* – чисельність розробників проекту і-ої спеціальності, осіб;

*ti* – час, витрачений на розробку проекту працівником і-ої спеціальності, дні;

*ЗПДі*– денна заробітна плата розробника і-ої спеціальності, грн.;

Для розробки програмного забезпечення потрібна команда з чотирьох осіб:

* тестувальник;
* два програміста;
* проектний менеджер.

Період виконання проектного рішення з 18.09.2018 року до 11.12.2018 року, що становить 59 робочих днів.

Витрати на їхню заробітну плату наведені в табл.5.1.

*Таблиця 5.1*

**Розрахунок витрат на оплату праці**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №п/п | Спеціальність розробника | Кількість розробників, осіб | Час роботи, дні | Денна заробітна плата розробника, грн. | Витрати на оплату праці, грн. |
| 1 | Проектний менеджер | 1 | 59 днів | 1500 | 88 500 |
| 2 | Програміст | 2 | 59 днів | 1350 | 159 300 |
| 3 | Тестувальник | 1 | 38 днів | 1000 | 38 000 |
|  | Разом: | | | | 285 800 |

2) Витрати на оплату праці працівникам тягнуть за собою сплату єдиного соціального внеску Вєсв (22%).

Максимальну величину бази нарахування єдиного внеску встановлено на рівні 25 розмірів прожиткового мінімуму для працездатних осіб (п. 4 ч. 1 ст. 1 Закону про ЄСВ). Отже, з 01.01.2018р. максимальний розмір заробітної плати, з якої сплачують єдиний соціальний внесок - 55 845,00 грн.

Місячна заробітна плата всіх розробників проекту не перевищує максимальну величину бази нарахування єдиного внеску, тому нарахування розраховується, як 22% від розміру заробітної плати.

Вєсв = (285 800,00) \* 0,22 = 62 876,00 грн.

3) Витрати на додаткові вироби, що закуповуються (*Вд*) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків оформляються у табл. 5.2. Транспортно-заготівельні витрати (*Втр)* становлять 15% суми витрат на додаткові вироби, що закуповуються.

*Таблиця 5.2*

**Розрахунок витрат на куповані вироби**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Найменування матеріалів | Марка, тип | Кількість на розробку, шт. | Ціна за одиницю, грн. | Сума витрат, грн. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| 1 | Флешка | Kingston DataTraveler SE9 32GB | 5 | 300,00 | 1500,00 | 1725,00 |
| 2 | Папір | ZOOM 543234 | 10 | 100,00 | 1000,00 | 1150,00 |
| 3 | Канцтовари | Dendi | 20 | 100,00 | 2000,00 | 2300,00 |
| 4 | Картридж | HP 17A LJ Pro | 2 | 2000,00 | 4000,00 | 4600,00 |
| 5 | Навушники | Sennheiser CX 3 | 4 | 1000,00 | 4000,00 | 4600,00 |
| Всього | | | | | | 14375,00 |

4) Для ведення основної діяльності необхідним є спецобладнання, його перелік і ціни наведені у табл. 5.3.

*Таблиця 5.3*

**Бюджет витрат на придбання спецобладнання**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Найменування матеріалів | Марка, тип | Кількість на розробку, шт. | Ціна за одиницю, грн. | Сума витрат, грн. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| 1 | Ноутбук | HP 250 G5 | 4 | 18 000,00 | 72 000,00 | 82 800,00 |
| Всього | | | | | | 82 800,00 |

5) Накладні витрати (*Bн)* включають витрати на управління, загальногосподарські, невиробничі витрати. Вони становлять 20 % витрат на оплату праці.

=\*0,2 = 285 800,00\*0,20 = 57 160,00 грн.

6) Інші витрати (*Bін)* – це витрати, які не враховані в попередніх статтях витрат. Вони розраховуються за встановленими відсотками (5-12 %) до витрат на оплату праці.

=\*0,12 = 285 800,00\*0,12 = 34 296,00 грн

7) Витрати на розробку проектного рішення обчислюємо за формулою:

 (5.4)

*К1=*182 100,00+40 062,00+14 375,00+36 420,00+9 105,00 = 282 062,00 грн*.*

Витрати на відлагодження і дослідну експлуатацію визначаємо за формулою:

 (5.5)

де *Sм.г* – вартість однієї години роботи ПК, грн./год.

*tвід* – кількість годин роботи ПК на відлагодження програми, год.

*К2*=150 год\*75,00 грн/год=11 250,00 грн.

Результати розрахунків зводяться у табл. 5.4.

*Таблиця 5.4*

**Кошторис витрат на розробку проектного рішення**

|  |  |
| --- | --- |
| Найменування елементів витрат | Сума витрат, грн. |
| Витрати на розробку проектного рішення, у т.ч.: |  |
| витрати на оплату праці | 285 800,00 |
| єдиний соціальний внесок | 62 876,00 |
| витрати на додаткові вироби, що закуповуються | 14 375,00 |
| витрати на придбання спецобладнання | 82 800,00 |
| накладні витрати | 57 160,00 |
| інші витрати | 34 296,00 |
| Витрати на відлагодження і дослідну експлуатацію системи | 11 250,00 |
| Всього | 548 557,00 |

Отже, загальний кошторис витрат на проектне рішення по аукціону становить 548 557, 00 грн., найбільшу частку цих витрат займають витрати на оплату праці – 285 800,00 грн. та витрати на придбання спецобладнання 82 800,00 грн.

## 5.3. Визначення комплексного показника якості

Зростаюча складність ПЗ призводить до збільшення кількості помилок у ньому, а одночасне зростання кількості й критичності виконуваних ним функцій тягне за собою збільшення збитків від цих помилок. Тому завдання оцінювання його якості є актуальним. Під якістю ПЗ розуміють сукупність властивостей ПЗ, які обумовлюють його придатність задовольняти деякі потреби відповідно до призначення. Метою оцінки якості ПЗ може бути прийняття рішення про ступінь відповідності якості ПЗ досягнутого при розробці відповідно до заданого рівня.

Комплексний показник якості (*Пя*) визначається шляхом порівняння показників якості проектованої системи і вибраного аналогу.

Як аналог використаємо Cluvio – платформа для аналізу даних, яка дозволяє виконувати SQL запити до вашої бази даних, обробляти дані в R, візуалізувати результати та створювати красиві, інтерактивні інформаційні панелі за лічені хвилини.

Визначення комплексного показника якості наведено в табл.5.5.

*Таблиця 5.5*

**Визначення комплексного показника якості проектованої системи (аналога)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показники | Числове значення показників, бали | | Відносний показник якості, Сі | Коефіцієнт вагомості  qi | Ci ×qi |
| Аналог | Проект. прогр. продукт |
| Функціональна повнота | 7 | 10 | 1,429 | 0,08 | 0,114 |
| Правильність | 7 | 9 | 1,286 | 0,06 | 0,077 |
| Сумісність | 6 | 9 | 1,500 | 0,07 | 0,105 |
| Захищеність | 8 | 10 | 1,250 | 0,09 | 0,113 |
| Узгодженість | 7 | 10 | 1,429 | 0,05 | 0,071 |
| Безвідмовність | 7 | 8 | 1,143 | 0,07 | 0,080 |
| Стійкість до помилок | 6 | 8 | 1,333 | 0,06 | 0,080 |
| Відновлюваність | 8 | 9 | 1,125 | 0,07 | 0,079 |
| Зрозумілість | 7 | 8 | 1,143 | 0,05 | 0,057 |
| Легкість навчання | 9 | 10 | 1,111 | 0,06 | 0,067 |
| Оперативність | 7 | 9 | 1,286 | 0,05 | 0,064 |
| Аналізованість | 7 | 10 | 1,429 | 0,08 | 0,114 |
| Змінюваність | 8 | 10 | 1,250 | 0,05 | 0,063 |
| Стабільність | 7 | 9 | 1,286 | 0,07 | 0,090 |
| Тестованість | 8 | 9 | 1,125 | 0,09 | 0,101 |
| Всього | 109 | 138 | 19,123 | 1 | 1,275 |

Отже, комплексний показник якості проектованої системи становить 1,275, найбільшу вагомість в якому мають такі показники як захищеність, тестованість, аналізованість, функціональна повнота, легкість навчання тощо.

## 5.4. Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні витрати включають вартість підготовки даних (*Е1*) і вартість годин роботи ПК (*Е2*). Одноразові експлуатаційні витрати визначаються за формулою:

 (5.6)

де *ЕП(А)* - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

*Е1П(А)* - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

*Е2П(А)* - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

*ЕП* = 10 000 + 12\*280 = 13 360 грн.

*ЕА* = 25 000 + 25\*400 = 35 000 грн.

Річні експлуатаційні витрати визначаються за формулою:

 (5.7)

де *B(e)П(А)* – експлуатаційні річні витрати проектного рішення, грн.;

*NП(А)* - періодичність експлуатації проектного рішення (аналогу), разів/рік.

*В(е)П* =13 360 \* 14 = 187 040 грн/рік

*В(е)А* = 35 000 \*17 =595 000 грн/рік

Вартість підготовки даних для експлуатації проектного рішення (аналогу) () визначаються за формулою:

 (5.8)

де *i* – номери категорій персоналу, які беруть участь у підготовці даних;

*ni*– кількість співробітників і-ї категорії, ос.;

*ti*– трудомісткість роботи співробітників і-ї категорії, ос.;

*ЗПгі* – середньогодинна ставка робітника і-ї категорії з врахуванням сплати єдиного соціального внеску, грн./год.

Середньогодинна ставка оператора визначається за формулою:

 (5.9)

де *ЗПг0i* – основна місячна зарплата працівника і-ї категорії, грн.;

*b* – коефіцієнт, який враховує сплату єдиного соціального внеску;

*Фг* – місячний фонд робочого часу, год.

Вихідні дані та результати розрахунків витрат на підготовку даних для експлуатації проектного рішення наведені у табл. 5.6.

*Таблиця 5.6*

**Розрахунок витрат на підготовку даних для роботи на ЕОМ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія персоналу | Чисельність співробітників і-ої категорії, осіб. | Час роботи співробітників  і-ої категорії, год. | Середньогодинна ЗП співробітника і-ої категорії, грн. | Витрати на підготовку даних, грн. |
| Проектне рішення | | | | |
| Проектний менеджер | 1 | 10 | 125,00 | 1 250,00 |
| Тестувальник | 1 | 21 | 75,00 | 1 575,00 |
| Всього | 2 | 27 | 200,00 | 2 825,00 |
| Аналог | | | | |
| Проектний менеджер | 2 | 15 | 95,00 | 1 425,00 |
| Тестувальник | 6 | 41 | 55,00 | 2 255,00 |
| Всього | 8 | 56 | 150,00 | 3 680,00 |

У випадку ручного виконання функцій одноразові витрати на них (*E*) визначаються за формулою:

 (5.10)

де *C0* - трудоємність обробки інформації ручним способом при одноразовому виконанні функцій, люд./год.;

*a* - коефіцієнт, який враховує додаткову трудомісткість обробки інформації на додаткових операціях (від 2 до 3);

*C0* – середньогодинна ставка персоналу певної категорії, грн./год.

Трудомісткість обробки інформації ручним способом наведена в табл. 5.7.

*Таблиця 5.7*

**Розрахунок трудомісткість виконання автоматизованих функцій традиційним методом**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операції по виконуваних фукціях | Спеціальність і розряд виконавців | Чисельність виконавців по операціях, осіб | Час, що витрачається виконавцем на операцію, год. | Трудоємність виконання операцій Dоі, люд./год. |
| Перевірка даних покупця при створенні аккаунту | Тестувальник | 1 | 1 | 1 |

У випадку ручного виконання функцій одноразові витрати становлять:

*Е* = 1 \* 2 \*75 = 150,00 грн.

Отже, річні експлуатаційні витрати проектного продукту становлять 187 040 грн/рік, а той же час як річні експлуатаційні витрати для аналогу становлять 595 000 грн/рік. А у випадку ручного виконання функцій одноразові витрати для проекту становлять всього 150,00 грн., що означає що більшість усіх робіт по проекту є автоматизованими і не потребують ручного втручання.

## 5.5. Розрахунок ціни споживання проектного рішення

Ціна споживання (*Цc*) – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

 (5.1)

де *Цп* – ціна придбання проектного рішення, грн.;

*B(E)NPV*– теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

, (5.11)

де *Пр* – норматив рентабельності;

*Ко*– витрати на прив'язку та освоєння проектного рішення на конкретному об’єкті, грн.;

*Кк* – витрати на доукомплектування технічних засобів на об'єкті, грн.

Ц*П* = 548 557,00\*(1 + 72%/100%) + 5 500 + 8 500 = 1 001 758,04 грн.

Ц*П А* = 408 885,55\*(1 + 49%/100%) + 2 300 + 4 000 = 615 539,47 грн.

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

 (5.12)

де  - річні експлуатаційні витрати в t-ому році, грн.;

*T* - строк служби проектного рішення, років;

*R* - річна ставка проценту банків.

В*(е)NPV* = Ʃ 87 040/(1+11%)5 = 691 280,58 грн.

ЦС = 1 001 758,04 + 691 280,58 = 1 693 478,21 грн.

Аналогічно визначається ціна споживання для аналогу. Якщо аналог – програмний засіб, ціна придбання встановлюється за середньою ринковою ціною; якщо - ручне виконання функцій, .

В*(е)NPV* = Ʃ595 000/(1+11%)3 = 1 454 010,26 грн.

ЦС = 615 539,47 + 1 454 010,26 = 2 069 549,73 грн.

Отже, так як наш програмний продукт планується використовувати вродовж п’яти років, то його теперішня вартість витрат на експлуатацію проектного рішення становить 691 280,58 грн., а ціна споживання – 1 693 478,21 грн., в той же час аналог ще планують використовувати ще три роки і його ціна споживання становить 2 069 549,73 грн.

## 5.6. Визначення показників економічної ефективності

Якщо базою для порівняння обрані відповідні програмні засоби, в даному розділі розраховуються такі показники:

1) Показник конкурентоспроможності:

 (5.13)

*Ккс* = (1 693 478,21 \* 1,275)/ 2 069 549,73 = 1,056

2) Економічний ефект в сфері експлуатації (грн.):

. (5.14)

*Еекс* = 595 000,00 – 187 040,00 = 407 960,00 грн

3) Економічний ефект в сфері проектування (грн.):

. (5.15)

*Епр* = 902 539,47 – 1 001 758,04 = < 0 грн.

1. Додатковий економічний ефект в сфері експлуатації (грн.):

 (5.16)

*Еекс Д* = Ʃ 407 960,00\*(1+11%)5-1 = 2 540 693,86 грн.

1. Додатковий економічний ефект в сфері проектування (грн.):

. (5.17)

Додаткового економічного ефекту в сфері проектування немає, бо *Епр*< 0.

6) Термін окупності витрат на проектування рішення (років):

. (5.18)

Т*ок* = 586 737,00 /407960,00 = 1,5

Результуючі показники економічної ефективності наведені у табл. 5.8.

*Таблиця 5.8*

**Показники економічної ефективності проектного рішення**

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування показників | Одиниці вимірювання | Значення показників | |
| Аналог | Проектне рішення |
| 1. Капітальні вкладення | грн. | 289 745,00 | 82 800,00 |
| 2. Ціна придбання | грн. | 615 539,47 | 1 001 758,04 |
| 3. Річні експлуатаційні витрати | грн. | 1 454 010,26 | 691 280,58 |
| 4. Ціна споживання | грн. | 2 069 549,73 | 1 693 478,21 |
| 5. Економічний ефект в сфері експлуатації | грн. | 407 960,00 | |
| 6. Додатковий економічний ефект в сфері експлуатації | грн. | 2 540 693,86 | |
| 7. Економічний ефект в сфері проектування | грн. | <0 | |
| 8. Додатковий економічний ефект в сфері проектування | грн. | <0 | |
| 9. Термін окупності витрат на проектування рішення | роки | 1,5 | |
| 10. Коефіцієнт конкурентоспроможності |  | 1,056 | |

Отже, розрахувавши показники економічної ефективності проектного рішення, ми бачимо, що ціна придбання проектного рішення – 1 001 758,04 грн. при ціні споживання – 1 693 478,21 грн. І терміном окупності витрат на проектування рішення є 1,5 року, що є хорошим показником для комерційного проекту.

## 5.7. Висновки до розділу 5

Загальний кошторис витрат на проектне рішення для аналізубеликих даних алгоритмом apriori становить 586 737, 00 грн., найбільшу частку цих витрат займають витрати на оплату праці – 285 800,00 грн. та витрати на придбання спецобладнання 120 980,00 грн.

Комплексний показник якості проектованої системи становить 1,275, найбільшу вагомість в якому мають такі показники як захищеність, тестованість, аналізованість, функціональна повнота, легкість навчання тощо.

Річні експлуатаційні витрати проектного продукту становлять 187 040 грн/рік, в той же час як річні експлуатаційні витрати для аналогу становлять 595 000 грн/рік. А у випадку ручного виконання функцій одноразові витрати для проекту становлять всього 150,00 грн., що означає що більшість усіх робіт по проекту є автоматизованими і не потребують ручного втручання.

Досягнення результатів проектів розвитку значною мірою залежать від якості менеджменту проекту, рівня кваліфікації та досвіду персоналу, залученого до проекту, а також управлінської структури організації – виконавця проекту.

Так як наш програмний продукт планується використовувати вродовж п’яти років, то його теперішня вартість витрат на експлуатацію проектного рішення становить 691 280,58 грн., а ціна споживання – 1 714 468,22 грн., в той же час аналог ще планують використовувати ще три роки і його ціна споживання становить 2 069 549,73 грн.

Отже, розрахувавши показники економічної ефективності проектного рішення, ми бачимо, що ціна придбання проектного рішення – 1 023 187,64 грн. при ціні споживання – 1 714 468,22 грн. І терміном окупності витрат на проектування рішення є 1,5 року, що є хорошим показником для комерційного проекту.

# Висновки

Магістерська кваліфікаційна робота присвячена дослідженню та реалізації алгоритму Apriori, та введенню можливих модифікацій для кращої роботи алгоритму.

Було здійснено порівняльний аналіз роботи класичного алгоритму Apriori та його модифікацій. Опрацьовано дані вказаним алгоритмом та оцінено швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил із значенням підтримки та вірогідності до кожного правила, для база даних із дорожньо-транспортними пригодами. Вироблено на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Для програмної реалізації створеного алгоритму використано технології .Net, мову C#, технологію WPF для створення інтерфейсу користувача та інтеграційне середовище розробки Visual Studio 2015.

Було проведено дослідження розробленого засобу. Дослідження складається з таких частин: дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом - розбиття множини всіх можливих транспортних пригод на дві підмножини, дослідження модифікованого алгоритму за другим способом – додавання ще однієї характеристики асоціативних правил та порівняльний аналіз вище перелічених досліджень.

Отже, після здійснення досліджень можна сказати що запропонований варіанти модифікації №1 є найкращим у нашому випадку, за цим методом далось скоротити час виконання алгоритму на 11%, при цьому не втративши жодної важливої інформації.

Варіант модифікації №2 не вдалось реалізувати як було описано в теорії, тому що це руйнує логіку алгоритму, проте можна застосувати при генерації асоціативних зв’язків але результати для нашого випадку, аналіз ДТП, були гіршими за еталон

Магістерська робота містить дослідницький характер та містить дослідження в якому розроблено модифікацію для базового алгориту, яка дозволяє проводити аналіз бази даних швидше не втративши при цьому точності.

Практична цінність отриманих результатів полягає у тому, що реалізоване програмне забезпечення дозволяє проводити аналіз даних, а для нашого випадку для попередження виникнення нових ДТП.

За результатами дослідження опубліковано матеріали доповіді на Міжнародній науковій інтернет-конференції "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення"(випуск 33) з доповіддю на тему «Модифікація алгоритму Apriori для аналізу дорожньо-транспортних пригод».

Проведено економічну характеристику розробленого програмного продукту. Згідно проведених розрахунків можна стверджувати, що розроблювані засоби для аналізу роботи алгоритму Apriori є кращою від її аналогу за економічними показниками. Термін окупності витрат на проектування становить всього 1 рік та 5 місяців.

# Перелік використаної літератури

1. Laney, Doug. "3D data management: Controlling data volume, velocity and variety". META Group Research Note. (2001) 45–60.
2. Marr, Bernard "Big Data: The 5 Vs Everyone Must Know". (6 March 2014). 98–103
3. "Data, data everywhere". The Economist. 25 February 2010. Retrieved 9 December 2012. 30–35
4. Reichman, O.J.; Jones, M.B.; Schildhauer, M.P. (2011). "Challenges and Opportunities of Open Data in Ecology". Science. 331 (6018): 703–5. doi:10.1126/science.1197962. PMID 21311007.
5. John R. Mashey. "Big Data ... and the Next Wave of InfraStress" (PDF). Slides from invited talk. Usenix. Retrieved 28 September 2016. 20–21
6. Snijders, C.; Matzat, U.; Reips, U.-D. (2012). "'Big Data': Big gaps of knowledge in the field of Internet". International Journal of Internet Science. 7: 1–5.
7. De Mauro, Andrea; Greco, Marco; Grimaldi, Michele (2016). "A Formal definition of Big Data based on its essential Features". Library Review. 65: 122–135. doi:10.1108/LR-06-2015-0061.
8. Hilbert, Martin. "Big Data for Development: A Review of Promises and Challenges. Development Policy Review". martinhilbert.net. Retrieved 7 October 2015.
9. Lee, Jay; Bagheri, Behrad; Kao, Hung-An (2014). "Recent Advances and Trends of Cyber-Physical Systems and Big Data Analytics in Industrial Informatics". IEEE Int. Conference on Industrial Informatics (INDIN) 2014.
10. Опис асоціативних правил studfiles.net [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://studfiles.net/preview/4494757/page:2/ (дата звернення: 09.05.2018).
11. Tufféry, Stéphane (2011); *Data Mining and Statistics for Decision Making*, Chichester, GB: John Wiley & Sons, translated from the French *Data Mining et statistique décisionnelle* (Éditions Technip, 2008)
12. Brin S. Dynamic itemset counting and implication rules for market basket data / Brin S., Motwani R., Ullman J., Tsur S // SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, pages 255-264, Tucson, Arizona, USA, May 1997.
13. Аналіз алгоритму apripri для структурованих та неструктурованих даних [Веб-сайт]. – Електронні дані. – Режим доступу:

http://oleg-ism.ua-warez.com/storage/articles/872/8\_%D0%8.pdf (дата звернення: 09.05.2018)

1. Пошук асоціативних правил для аналізу надзвичайних ситуацій на залізничному транспорті [Веб-сайт]. – Електронні дані. – Режим доступу:

https://itce.vntu.edu.ua/index.php/itce/article/view/27/28 (дата звернення: 09.05.2018)

1. Data Mining [Електронний ресурс] ZeroR Режим доступу: <http://www.saedsayad.com/zeror.htm>.
2. Arcuri A. Black-Box System Testing of Real-Time Embedded Systems Using Random and Search-Based Testing / A. Arcuri, M. Z. Iqbal, L. Briand. // Proc. 22nd IFIP Int. Conf. Testing Software and Systems. – Natal, 2010. – С. 95–110.
3. Mohammed J. Zaki. Scalable algorithms for association mining / Mohammed J. Zaki // IEEE Transactions on Knowledge and Data Engineering, 12 (3): 372-390, May/June 2000.
4. WPF [Електронний ресурс] : – Режим доступу: <https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/aa970268(v=vs.100)> (дата звернення: 19.07.2018).
5. Документація по мові C# [Електронний ресурс] : – <https://docs.microsoft.com/en-us/dotnet/articles/csharp/csharp>
6. Документація по середовишу розробки Visual Studio [Електронний ресурс] : – <https://www.visualstudio.com/en-us/docs/vs/overview>
7. Документація по JavaScript [Електронний ресурс] : – Режим доступу : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
8. Документація по HTML5 [Електронний ресурс] : – Режим доступу : <https://www.w3schools.com/html/default.asp>
9. GreenPlum. [Електронний ресурс] //: сайт. - Режим доступу <http://www.emc.com/campaign/global/greenplumdca/index.htm>
10. Big Data. [Електронний ресурс] // Вікіпедія: сайт. - Режим доступу <https://en.wikipedia.org/wiki/Big_data>
11. Дюк В., Самойленко А. Data Mining: Учебный курс. – СПб: Питер, 2001. – 368 с.
12. А. Шахиди. Data Mining – добыча данных [Електронний ресурс] // Big Data: сайт. - Режим доступу // <http://www.basegroup.ru>
13. Karen Montgomery. Big Data Now: 2014 Edition. O'Reilly Media.- Junuary, 2015 – 165p.
14. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. John Wiley & Sons. 2014-12-19. 300p.
15. Tong Qiang, Zhou Yuanchun, Wu Kaichao, Yan Baoping. A quantitative association rules mining algorithm[J]. Computer engineering. 2007
16. Hahsler, Michael (2005). ["Introduction to arules – A computational environment for mining association rules and frequent item sets"](https://mran.revolutionanalytics.com/web/packages/arules/vignettes/arules.pdf) (PDF). *Journal of Statistical Software*
17. Hipp, J.; Güntzer, U.; Nakhaeizadeh, G. (2000). "Algorithms for association rule mining --- a general survey and comparison". ACM SIGKDD Explorations Newsletter.
18. Гаврилюк А. Модифікація алгоритму apriori для аналізу дорожньо-транспортних пригод [Електронний ресурс]: сайт. - Режим доступу // http://www.konferenciaonline.org.ua

# ДОДАТКИ

## Додаток А. План забезпечення якості програмного продукту

1. **Мета**

Цей план забезпечення якості встановлює процеси, методи, стандарти і процедури які будуть використовуватись для встановлення та забезпечення якості програмного продукту - «Додаток для аналізу даних за допомогою алгоритму Apriori».

Для розробки проекту буде використана швидка (RAD) модель розробки програмного забезпечення. Особливістю такої моделі є розробка, яка здійснюється ітераційним шляхом її еволюційного розвитку з отриманням деякого варіанта системи–прототипу, на якому перевіряється реалізація вимог. Дана модель використовується для розробки систем, де головною вимогою є реалізація функцій системи. В разі використання даної моделі система послідовно розробляється з блоків конструкцій. В основу швидкої моделі розробки програмного забезпечення покладена спіральна модель.

Недоліки цієї моделі такі:

* Використання моделі може виявитися невдалим у випадку, якщо відсутні придатні для повторного використання компоненти;
* Виникає потреба в системі, яка може бути змодельована коректним чином;
* Існує ризик, що роботу над проектом не буде завершено;

Мета плану забезпечення якості полягає в забезпеченні відповідності процесу розробки вимогам до програмного продукту.

1. **Управління**
2. 1. **Організація**

Кожен учасник команди відповідає за якість виконання своєї роботи. Для виконання проекту призначається відповідальний за якість програмного проекту, в даному випадку це студент магістр. Відповідальний за якість керує всіма питаннями, пов'язаними із забезпеченням якості в проекті. Зокрема за вчасне виконання поставлених завдань, дотримування вимог до проекту, відповідність стандартам, ефективність процесу розробки та ефективність реалізації вимог до програмного забезпечення, відповідність вимогам та стандартам якості.

* 1. **Завдання**

Завдання з контролю якості включають в себе:

* Документування;
* Оглядові наради (експертизи, інспекції);
* Верифікацію;
* Валідацію;
* Види діяльності, спрямовані на поліпшення самого процесу забезпечення якості.

Ці завдання деталізовані в даному документі.

* 1. **Ролі та обов'язки**

Оскільки команда розробників складається з однієї людини, то роль інженера по якості програмного забезпечення буде виконувати розробник. Дана особа повинна мати чітке уявлення як продукт має бути реалізований. У зв’язку з невеликою кількість людей у команді, було вирішено поділити ролі таким чином, щоб не довелося залучати в команду додаткову особу, яка б виконувала роль тільки інженера з якості програмного забезпечення.

Також на проекті присутня роль власника продукту, який відповідає за формулювання вимог до програмного продукту та керівника проекту. Всі питання, що виникають власник продукту обговорює з розробником, чия робота напряму спрямована на результат та відповідність плану якості.

До обов’язків інженера з забезпечення якості належать:

* створення та підтримка плану забезпечення якості;
* виконання положень плану забезпечення якості, визначених документом, для перевірки дотримання якості та відповідності вимогам програмного продукту;
* надання необхідних пояснень, щодо цього документу, процесу забезпечення якості на проекті;
* перевірка дотримування плану якості;
* при необхідності оновлювання плану якості програмного забезпечення.

До обов’язків керівника проекту належать:

* затвердження положень та змін до процесу забезпечення якості;
* контроль за дотриманням виконання обов’язків розробника.

1. **Документація**
2. 1. **Мета**

У даному розділі визначається документація, що описує забезпечення якості і буде використана в рамка виконання проекту для аналізу даних за допомогою алгоритму Apriori.

* 1. **Мінімальні вимоги до документації**

Повинні бути створені такі документи:

* SQAP: План забезпечення якості (даний документ).
* SRD: Специфікація вимог до програмного забезпечення.
* SDD: Дизайн програмного забезпечення.
* STD: Документація з тестування програмного забезпечення.
* Плани проведення верифікації та валідації.
* План по виявленню та класифікації ризиків.
* План супроводу.
  + 1. **Опис вимог до програмного забезпечення (SRD)**

SRD визначає вимоги для програмного продукту, які виконують певні функції в конкретному середовищі. Опис вимог повинен визначати вимоги для конкретного програмного продукту. У ньому документі слід розглянути основні питання функціональності, зовнішні інтерфейси, продуктивність, атрибутів та обмежень на реалізацію. Кожна вимога повинна бути однозначно ідентифікована та визначена так, що її задоволення здатне бути об'єктивно перевіреним і підтвердженим.

Опис вимог складається з метою формування спільного бачення роботи системи всім учасниками проекту та надає вичерпний перелік вимог, розділених за їх класами. Документ формується на початкових етапах проекту та може доповнюватись по мірі необхідності.

Кроки для підтвердження відповідності опису вимог:

* При внесенні зміни в опис вимог документ провіряється на відповідність:
  + Внутрішні практикам та узгодження;
  + Міжнародним стандартам;
* Кожна нова\змінена вимоги повинні верифікується керівником продукту;
* Критерії відповіднсоті для опису вимог:
* Відповідність установленим практикам та домовленостям;
* Відповідність стандартам;
* Затвердження власником продукту.
  + 1. **Опис дизайну програмного забезпечення (SDD)**

SDD описує, як програмне забезпечення буде будуватися відповідно до вимог SRD. Документ повинен описувати компоненти й підкомпоненти розробки програмного забезпечення, включаючи бази даних і внутрішні інтерфейси. Він може бути підготовлений як архітектура продукту.

Кроки для підтвердження опису дизайну ПЗ:

* При внесенні змін в опис дизайну ПЗ документ перевіряється на відповідність:
  + функціональним та нефункціональним вимогам до продукту згідно опису вимог;
  + установленим практикам та стандартам.
* Потім документ презентується керівнику продукту (керівнику магістерської кваліфікаційної роботи);
* У випадку несхвалення документ потрібно переопрацювати з урахування зауважень керівника;
* При необхідності збереження поточних змін в документі опису дизайну ініціюється зміна до опису вимог, що обговорюється з власником продукту.

Критерії для підтвердження опису дизайну ПЗ:

* Інформованість керівника продукту щодо документу;
* Відповідність стандартам та визначеним практикам;
* Задоволення опису вимог до ПЗ.
  + 1. **Плани верифікації та валідації**

Верифікація та валідація використовується для визначення того, чи розроблений програмний продукт відповідає вимогам, і чи є програмне забезпечення відповідним до призначення і очікуванням користувачів. План включає в себе: аналіз, контроль, оцінку і тестування програмного продукту і процесів, які стосуються випуску продукту. Крім того, тестування програмного забезпечення, атестація і перевірка процесів застосовуються при інтеграції зовнішніх частин розроблюваного продукту.

План визначає завдання верифікації та валідації, входів і виходів, необхідних для підтримки потрібного рівня цілісності програмного забезпечення. Він також надає засоби тестування (інтеграційного та системного рівня) та розробки вимог SRD, щодо розробки, як описано в SDD, і щодо тестування, як описано в проектній документації.

Кроки для підтвердження плану верифікації та валідації:

* При внесенні змін в план верифікації та валідації документ провіряється на відповідність установленим практикам та стандартам;

Далі документ презентується власника продукту;

* У випадку несхвалення документ потрібно переопрацювати з урахування зауважень керівника;
* Критерії для підтвердження плану варифікації та валідації:
* Інформованість керівника продукту щодо документу;
* Відповідність стандартам та визначеним практикам.
  + 1. **Користувацька документація**

Користувацька документація містить опис встановлення, експлуатації, управління та обслуговування програмного продукту.

Документація описує контроль вихідних даних, вхідних послідовностей, параметри програми, обмеження, і всі інші необхідні дані для програмного продукту. Всі повідомлення про помилки визначені і описані. Описані всі коригувальні дії, направлені на виправлення помилок, описаних в повідомленнях про помилку. Документація поширюється на будь-яку частину вбудованого програмного забезпечення, з якими користувач взаємодіє безпосередньо.

Кроки для підтвердження користувацької документації:

* При внесенні змін в користувацьку документацію потрібно інформувати власника продукту;
* У випадку несхвалення потрібно врахувати зауваження та внести потрібні поправки.
* Критерії для підтвердження користувацької документації:
* Достовірність;
* Повнота;
* Несуперечливість;
* Чіткість та лаконічність.
  + 1. **Інші документи**

Керівник проекту, керівник продукту, чи інженер з забезпечення якості можуть включати в документ нову документацію, відносно якої відбуватиметься контроль забезпечення якості.

1. **Стандарти, практики, конвенцій, і метрики**

Даний розділ описує стандарти, практики, угоди і метрики, які використовуються в проекті. Ці матеріали покликані не тільки забезпечити якість проекту, але й отримати кількісні дані про сам процес контролю якості. Отримані дані повинні бути використані розробником для підвищення рівня самого програмного продукту.

*Стандарти.*

Для ведення документації використовуються стандарти IEEE з відповідними модифікаціями.

*Практики.*

* Оскільки відкладати забезпечення якості недоцільно, розробник (студент магістр) застосовує процедури забезпечення якості безпосередньо під час роботи.
* Для зменшення кількості помилок перед початком написання коду розробляються необхідні для нього тести;
* Всі артефакти проекту підлягають інспектуванню і всі вони доступні після випуску. Це досягається шляхом переміщенням артефактів в систему управління конфігураціями, що забезпечує доступ до вмісту в будь-який час.
* Для всіх процесів має бути проведений огляд можливості поліпшення хоча б раз, і результати цього огляду документуються окремо у письмовій формі.

*Угоди.*

Правила та стиль написання документів повинні відповідати стандартам.

Для полегшення супроводу при написанні коду застосовуються стандарти мови програмування C, а також дотримання стандарту написання коду MISRA C: 2004.

*Метрики.*

* Для кожного процесу і кожного документа повинні вимірюватися принаймні три показника.
* Час, витрачений розробником, на виконання кожної під задачі.
* Інтегральний показник якості за шкалою від 1 до 10 на основі самооцінки та оцінки керівника.
* Кількість дефектів на одиницю об'єму (наприклад, на тисячу рядків коду).
* Кількість виправлених дефектів.

1. **Тестування**

В процесі реалізації програмного продукту здійснюватиметься його тестування. Передбачається використання наступних видів тестування:

* **Функціональне тестування –** перевіряє чи реалізовані функціональні вимоги, тобто можливості ПЗ в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить продукт, які завдання вирішує.
* **Навантажувальне тестування –** проводиться для того, щоб оцінити поведінку програми із заданим очікуваним навантаженням.
* **Стресове тестування –** застосовується для встановлення границь пропускної здатності програми. Цей тип тестування проводиться для визначення надійності системи під час екстремальних або непропорційних навантаженнях і відповідає на питання про достатню продуктивність системи у випадку, якщо поточне навантаження значно перевищить очікуваний максимум.
* **Тестування швидкодії –** Проводиться з метою встановлення, як швидко працює система або її частина, під певним навантаженням. Оскільки буде опрацьовуватись графові дані великого обсягу.
* **Тестування зручності використання –** Виконується з метою визначення зручності використання ПЗ для його подальшого застосування

Проектування та опис тестових випадків здійснюється згідно стандарту ISO 839. Модульні тести не документуються. Процес тестування, повинен відбуватись за планом функціонального тестування. Цей план тестування створюється керівником проекту та узгоджується з замовником. Модульне тестування повинно проводитися в повному обсязі після внесення в проект будь-яких змін.

План функціонального тестування не повинен включати тестові випадки для перевірки функцій, тестування яких було здійснене за допомогою модульних тестів.

1. **Звіти про помилки та коригувальні дії**

Для звітності буде використовуватись система відслідковування дефектів, що буде оформлена як дошка на веб-сервісі kanbanize. Опис програмних дефектів здійснюється згідно стандарту ISO 839.

Керівник проекту повинен визначити процедуру звітності про проблеми в забезпеченні якості. Вимоги до процедури звітності:

* При виявленні дефекту необхідно описати його відповідно до вимог та повідомити про нього керівника проекту;
* Всі дефекти вносяться в електронну систему відслідковування дефектів.
* Звіт про проблему повинен містити:
* Відомості про частину системи, в якій виник дефект;
* Час винекнення ефекту;
* Дії, що призвели до винекнення дефекту, в разі наявності такої інформації.

1. **Засоби, методи і метрики**

Керівник проекту повинен вимірювати здатність команди передбачати та долати ризики, здавати проект у заплановані сроки та вести статистику порушень якості продукту відносно порушених вимог, програмних модулів, порушників тощо.

Метрикою продуктивності є відношення виконаних завдань до запланованих завдань.

Метриками якості є кількість відкритих дефектів та кількість закритих дефектів.

1. **Контроль середовища**

Для визначення середовища для кожного проміжного результату та документації, у тому числі процесу копіювання і відновлення буде застосовуватись додаткове програмне забезпечення, а саме: система контролю версій GIT та веб-сервіс TortoiseGIT для управління нею, який надає можливість резервного копіювання проекту.

Проект буде знаходитись на сервері з обмеженим доступом, що забезпечить його від пошкодження на всіх етапах життєвого циклу програмного забезпечення.

1. **Контроль розробника**

Для забезпечення контролю якості після кожної внесеної зміни буде проводитися ручне модульне тестування для перевірки наявності дефектів в роботі програмного забезпечення.

1. **Збір коментарів, їх підтримка і збереження**

В процесі забезпечення якості продукту керівник проекту може збирати таку осовну інформацію про стан якості на проекті:

* Звіти про порушення якості
* Вимоги до системи
* Стандарти
* Плани тестувань

Ця інформація повинна бути надана замовнику по його вимозі.

1. **Навчання**

Для навчання і покращення програмного коду буде використовуватися методика Code Review, що дозволить покращити якість програмного продукту.

1. **Управління ризиками**

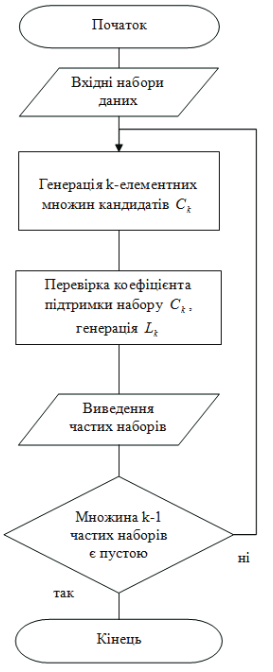
Для кожного ризику визначається його тип та вплив. Ці параметри використовуються для подальшої його оцінки керівником проекту. Ризики та їх параметри можуть змінюватися впродовж виконання проекту. Управління ризиками здійснюють керівник проекту та замовник.

Основними ризиками є:

1. Технологічні:
   1. Несумісність програмного забезпечення з апаратним (середній пріорітет);
   2. Непередбачувані технологічні складності (середній пріоритет);
   3. Наявність дефектів у підключених бібліотеках (низький пріоритет);
   4. Несумісність версій використовуваних технологій і фреймворків (середній пріорітет);
2. Організаційні:
   1. Зміни в вимогах (середній пріоритет);
   2. Не правильне планування (середній пріорітет);
   3. Недостатньо часу для реалізації програмного продукту;
3. Зовнішні ризики:
   1. Зміни в бізнес-середовищі (високий пріоритет);
   2. Паралельний розвиток конкурентоспроможних продуктів (високий пріоритет);

Зміни законодавчої бази держави (високий пріоритет);

## Додаток Б. Блок-схема роботи алгоритму



## Додаток В. Приклад даних для проведення досліджень

"чоловіча""31-35""alco 1.6-2.8""ніч""чорний""нижча""пасажири 4+""власник 1""16-20""місто""осінь"

"жіноча""15-17""alco 1.6-2.8""ранок""білий""середня""пасажир 0""власник 4+""11-15""село""зима"

"чоловіча""41-45""alco 0""ніч""білий""нижча""пасажири 2""власник 1""1-5""місто""зима"

"чоловіча""41-45""alco 1.6-2.8""вечір""чорний""вища""пасажири 4+""власник 1""1-5""місто""осінь"

"жіноча""46-50""alco 1.6-2.8""вечір""чорний""нижча""пасажир 0""власник 3""11-15""місто""весна"

"чоловіча""15-17""alco 1.0-1.5""ранок""чорний""вища""пасажир 0""власник 3""1-5""село""літо"

"жіноча""22-25""alco 0""ніч""білий""нижча""пасажири 1""власник 4+""16-20""місто""літо"

"жіноча""26-30""alco 1.6-2.8""ранок""білий""середня""пасажир 0""власник 4+""16-20""село""зима"

"чоловіча""46-50""alco 1.6-2.8""обід""чорний""вища""пасажири 2""власник 3""11-15""місто""осінь"

"чоловіча""22-25""alco 1.6-2.8""вечір""чорний""середня""пасажир 0""власник 3""1-5""село""літо"

"чоловіча""18-21""alco 1.0-1.5""обід""білий""середня""пасажири 4+""власник 4+""6-10""село""зима"

"чоловіча""31-35""alco 0""ранок""білий""нижча""пасажир 0""власник 2""11-15""місто""зима"

"жіноча""41-45""alco 1.0-1.5""ніч""чорний""нижча""пасажири 1""власник 3""1-5""село""зима"

"чоловіча""41-45""alco 1.6-2.8""ніч""білий""середня""пасажири 3""власник 2""11-15""село""осінь"

"жіноча""36-40""alco 1.6-2.8""ранок""білий""середня""пасажир 0""власник 3""16-20""село""осінь"

"жіноча""46-50""alco 1.6-2.8""обід""білий""нижча""пасажири 1""власник 4+""11-15""місто""зима"

"чоловіча""41-45""alco 1.0-1.5""вечір""білий""нижча""пасажири 4+""власник 4+""16-20""село""весна"

"жіноча""46-50""alco 0.6-0.9""вечір""білий""вища""пасажири 1""власник 3""6-10""село""осінь"

"чоловіча""31-35""alco 1.6-2.8""вечір""чорний""нижча""пасажири 3""власник 4+""11-15""місто""весна"

"чоловіча""22-25""alco 0""вечір""чорний""середня""пасажири 2""власник 4+""11-15""село""осінь"

"жіноча""46-50""alco 1.6-2.8""обід""жовтий""нижча""пасажир 0""власник 2""11-15""село""осінь"

"чоловіча""31-35""alco 1.0-1.5""ранок""чорний""середня""пасажири 2""власник 3""16-20""село""зима"

"чоловіча""46-50""alco 0.6-0.9""ранок""білий""вища""пасажир 0""власник 3""16-20""село""весна"

"жіноча""41-45""alco 0.6-0.9""обід""зелений""вища""пасажир 0""власник 4+""11-15""місто""весна"

"чоловіча""46-50""alco 1.0-1.5""обід""зелений""нижча""пасажири 4+""власник 2""16-20""місто""зима"

"жіноча""22-25""alco 1.0-1.5""ранок""білий""нижча""пасажири 1""власник 4+""6-10""місто""зима"

"чоловіча""41-45""alco 0""ніч""чорний""середня""пасажири 3""власник 4+""11-15""село""зима"

"чоловіча""46-50""alco 0.6-0.9""обід""чорний""вища""пасажир 0""власник 4+""11-15""місто""літо"

"чоловіча""46-50""alco 0""ранок""білий""середня""пасажири 4+""власник 1""11-15""місто""зима"

"жіноча""26-30""alco 0""обід""червоний""середня""пасажир 0""власник 1""11-15""місто""зима"

"чоловіча""31-35""alco 1.6-2.8""ранок""білий""нижча""пасажир 0""власник 3""11-15""місто""весна"

"чоловіча""46-50""alco 1.0-1.5""ранок""зелений""середня""пасажири 2""власник 4+""16-20""місто""зима"

"жіноча""18-21""alco 0""ніч""білий""середня""пасажири 4+""власник 4+""11-15""місто""літо"

"жіноча""18-21""alco 1.6-2.8""вечір""зелений""нижча""пасажир 0""власник 1""16-20""місто""літо"

## Додаток Г. Фрагменти коду

namespace MasterApriori.Implementation

{

public class Apriori : IApriori

{

Output IApriori.ProcessTransaction(float minSupport, float minConfidence, float minLift, IEnumerable<string> items, string[][] transactions, string[] itemsD = null)

{

IList<Item> frequentItems = GetL1FrequentItems(items, transactions);

ItemsDictionary allFrequentItems = new ItemsDictionary();

allFrequentItems.ConcatItems(frequentItems);

var candidates = new Dictionary<string[], double>();

double transactionsCount = transactions.Count();

do

{

candidates = GenerateCandidates(frequentItems, transactions, itemsD);

frequentItems = GetFrequentItems(candidates, transactionsCount);

allFrequentItems.ConcatItems(frequentItems);

} while (candidates.Count != 0);

HashSet<Rule> rules = GenerateRules(allFrequentItems);

IList<Rule> strongRules = GetStrongRules(minConfidence, rules, allFrequentItems);

return new Output

{

StrongRules = strongRules,

FrequentItems = allFrequentItems

};

}

private List<Item> GetL1FrequentItems(IEnumerable<string> items, string[][] transactions)

{

var frequentItemsL1 = new List<Item>();

double transactionsCount = transactions.Count();

foreach (var item in items)

{

double support = GetSupport(new [] {item}, transactions);

if (support / transactionsCount >= minSupport)

{

frequentItemsL1.Add(new Item { Names = new [] { item }, Support = support });

}

}

return frequentItemsL1;

}

private double GetSupport(string[] generatedCandidates, IEnumerable<IEnumerable<string>> transactionsList)

{

double support = 0;

foreach (var transactions in transactionsList)

{

if (!generatedCandidates.Except(transactions).Any())

{

support++;

}

}

return support;

}

private Dictionary<string[], double> GenerateCandidates(IList<Item> frequentItems, IEnumerable<IEnumerable<string>> transactions, string[] itemsD)

{

var candidates = new Dictionary<string[], double>();

for (var i = 0; i < frequentItems.Count - 1; i++)

{

var firstItems = frequentItems[i].Names.OrderBy(x => x).ToArray();

for (var j = i + 1; j < frequentItems.Count; j++)

{

var secondItems = frequentItems[j].Names.OrderBy(x => x).ToArray();

var generatedCandidate = GenerateCandidate(firstItems, secondItems);

if (generatedCandidate == null)

{

continue;

}

double support;

if (itemsD == null)

{

support = GetSupport(generatedCandidate, transactions);

candidates.Add(generatedCandidate, support);

}

else

{

if (itemsD.Any(itemD => generatedCandidate.Contains(itemD)))

{

support = GetSupport(generatedCandidate, transactions);

candidates.Add(generatedCandidate, support);

}

}

}

}

return candidates;

}

private string[] GenerateCandidate(string[] firstItems, string[] secondItems)

{

if (firstItems.Length == 1)

{

return firstItems.Concat(secondItems).Distinct().ToArray();

}

else

{

var firstSubArray = firstItems.Take(firstItems.Length - 1);

var secondSubArray = secondItems.Take(secondItems.Length - 1);

if (string.Join("", firstSubArray) == string.Join("", secondSubArray))

{

return firstItems.Concat(new [] { secondItems[secondItems.Length - 1] }).ToArray();

}

return null;

}

}}