МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Швед Юлія Василівна

Програмний додаток побудови та візуалізації блок-схем алгоритмів засобами мови програмування Delphi / Software application for constructing and visualizing for block diagrams algorithms on Delphi

напрям підготовки: 6.050102 - Комп’ютерна інженерія
фахове спрямування - Комп’ютерні системи та мережі
Бакалаврська робота

Виконав студент групи КСМз-41/2
Ю.В. Швед

Науковий керівник:
Батько Ю.М.

Тернопіль - 2018

РЕЗЮМЕ

Дипломний проект містить 59 сторінок пояснюючої записки, 12 рисунків, 7 таблиць, 2 додатки. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою дипломної роботи є розробка програмного додатку для створення та візуалізації блок-схем алгоритмів.

Методи досліджень базуються на теорії алгоритмів (для аналізу розроблених методів та алгоритмів), алгоритмах теорії графів (для обробки млок-схем у вигляді зв'язних графів), технологій об'єктно-орієнтованого програмування (для програмної реалізації спроектованої структури програмного додатку).

В дипломній роботі на основі аналізу існуючих алгоритмів обрбоки графів та вимог до створення конструкторської документації розроблений програмний додаток створення, редагування та візуалізації блок-схем алгоритмів довільної складності.

Проведено тестування розробленого програмного додатку на мові програмування Delphi, що підтвердило доцільність використання цифрової бібліотеки SimpleGraph при проектуванні та реалізації програмних додатків обробки інформації.

Розроблений програмний продукт є ефективним засобом з простим інтерфейсом, що дозволяє вирішувати проблему створення, редагування та візуалізації блок-схем алгоритмів довільної складності та може бути використаний при побудові програмних систем.

Ключові слова: ГРАФ, АЛГОРИТМ, БЛОК-СХЕМА, ДИНАМІЧНА БІБЛІОТЕКА.

RESUME

Diploma project contains 59 pages of explanatory notes, 12 figures, 7 tables 2 applications. Volume of graphic material 2 leaves of format A3.

The meta of the thesis is develop software application for creating and visualizing algorithms block diagrams.

Research methods are based on the theory of algorithms (for the analysis of developed methods and algorithms), graph theory theory algorithms (for processing graphic circuits in the form of connected graphs), object-oriented programming technologies (for software implementation of the designed software application structure).

In the dissertation work on the basis of analysis of existing algorithms of graphs and requirements for creation of design documentation, a software application for creation, editing and visualization of block diagrams of algorithms of arbitrary complexity has been developed.

Testing of the developed software application in the programming language Delphi has been tested, which confirmed the expediency of using the digital library SimpleGraph when designing and implementing software applications for information processing.

The developed software product is an effective tool with a simple interface, which allows you to solve the problem of creating, editing and visualizing block diagrams of arbitrary complexity algorithms and can be used in the construction of software systems.

Keywords: GRAPH, ALGORITHM, BLOCK-SCHEME, DYNAMIC LIBRARY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ОС	–	Операційна система
БД	–	База даних
ПЗ	–	Програмне забезпечення
GDI	–	Graphics Device Interface

ВСТУП

Теорія алгоритмів виникла з внутрішньої потреби математики. Математична логіка, метаматематика, алгебра, геометрія та аналіз залишаються і сьогодні однією з основних областей застосування теорії алгоритмів. Інша область її застосування появилася в 40-х роках XX-го століття — при створенні ефективних (перш за все швидкодіючих) електронних обчислювальних машин (ЕОМ). Поява ЕОМ особливо сприяла розвиткові тих розділів теорії алгоритмів, що мали яскраво виражену прикладну спрямованість. Сюди насамперед належали алгоритмічні системи та алгоритмічні мови (основа теорії програмування) універсальних ЕОМ, цифрові автомати. Саме поняття алгоритму тісно пов'язане з лінгвістикою, економікою, фізіологією мозку, психологією, філософією та природознавством тощо. Прикладом може послужити розумова чи практична діяльність людини у будь- якій сфері. Підвищення ефективності технологій виробництва, транспорту, зв'язку, енергетики, медицини тощо, а також відбору, зберігання, поширення, створення відповідних знань потребували удосконалення та розвитку інформаційних (вимірювальних) та управляючих (керуючих) алгоритмів. Це успішно здійснюється на базі математичних моделей величин, сигналів, фізичних полів. Виникла теорія таких алгоритмів. Для інженера теорія алгоритмів – це означення поняття алгоритму, алгоритмічні системи (рекурсивні функції, машини Тюрінга, нормальні алгоритми Маркова тощо), формальні перетворення і оцінки алгоритмів, теорія автоматів і універсальні ЕОМ, формальна побудова і аналіз мов програмування тощо. Проте, проблеми життєдіяльності з кожним роком висувають задачі, розв'язання яких універсальними обчислювачами (процесорами, контролерами) неефективне. Основною проблемою постає не традиційні вибір чи створення мови програмування, написання програм, а "наскрізне" (від математичного моделювання прикладної проблеми до розрахунку геометрії та топологій інтегральної схеми спеціалізованого пристрою) проектування. Це означає, що

інженерові необхідно вміти також й інше — оцінювати складність, часові параметри алгоритмів, будувати еквівалентні алгоритми, вміти робити їх тотожні перетворення задля пошуку кращого за певним критерієм алгоритму при заданих обмеженнях на технічні ресурси для його реалізації. А для швидкої та якісної оцінки алгоритмів необхідні зрозумілі та універсальні підходи до їх опису. На сьогоднішній день найбільш поширений є опис на основі групи графічних примітивів, що дозволяє шкидко та універсально описати алгоритм, що в подальшому, дозволяє провести його оцінку людьми, що можуть не розуміти один одного на рівні людської мови. Для пришвидшення процесу опису необідно розробляти програмні системи спрощення та автоматизації відображення алгоритмів.

Тому задача створення програмного додатку опису алгоритмів на основі блок-схем є актуальною.

1 ПРИНЦИПИ ТА ПРОГРАМНІ ЗАСОБИ ОПИСУ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ

1.1 Основні поняття теорії алгоритмів

Алгоритмом називається точний, однозначний та зрозумілий опис послідовності дій, що необхідно виконати виконавцю для вирішення поставленого завдання. Слово «алгоритм» походить від імені математика Аль Хорезмі, який сформулював правила виконання арифметичних дій. Спочатку під алгоритмом розуміли тільки правила виконання чотирьох арифметичних дій над числами. В подальшому це поняття стали використовувати в загальному, для позначення послідовності наперед визначених дій, що призводять до вирішення всіх поставлених завдань. Якщо розглядати алгоритми обчислювальних процесів, як окрему групу алгоритмів, то а такому випадку об'єктами, до яких застосовувався алгоритм, є дані в цифровому форматі.

Алгоритм рішення обчислювальної задачі представляє собою сукупність правил перетворення вихідних цифрових даних в деякий результ.

Основними властивостями алгоритму є:

– детермінованість (визначеність). Результати обчислювального процесу, при заданих тих самих вихідних даних завжди будуть однозначні. Завдяки цій властивості процес виконання алгоритму носить механічний характер;

– результативність. Дана властивість вказує на наявність таких вихідних даних, для яких виконаний за заданим алгоритмом обчислювальний процес повинен через скінченне, зліченне число кроків завершитись та видати шуканий результат;

– масовість. Ця властивість передбачає, що алгоритм повинен бути використаний для вирішення всіх завдань відповідного типу;

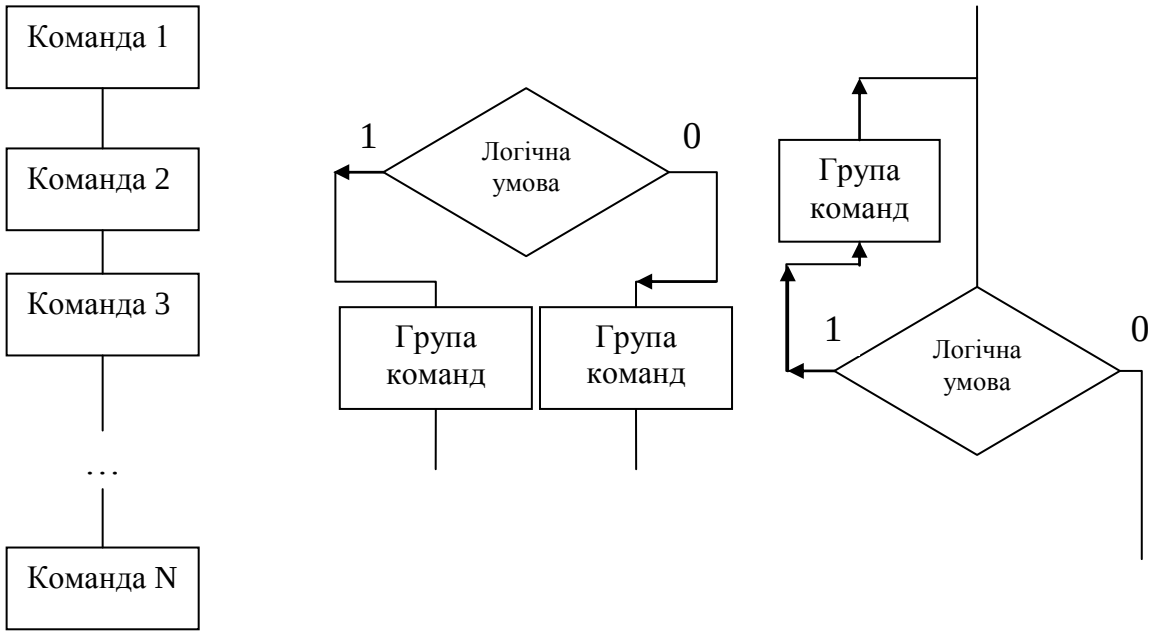
– дискретність. Це означає те, що весь алгоритмом обчислювального процесу повинен бути поділений на окремі етапи, можливість виконання яких виконавцем (комп'ютером) не викликає сумнівів.

При всьому різноманітті алгоритмів розв'язання задач в них можна виділити три основні види обчислювальних процесів:

– лінійний (рисунок 1.1,а);

– розгалужених (рисунок 1.1,б);

– циклічний (рисунок 1.1,в).



а) б) в)

Рисунок 1.1 – Приклади алгоритмів

Лінійним називається такий обчислювальний процес, при якому всі етапи рішення задачі виконуються в природному порядку проходження записи цих етапів.

Розгалуженим називається такий обчислювальний процес, в якому вибір напрямку обробки інформації залежить від вихідних або проміжних даних (від результатів перевірки виконання будь-якого логічного умови).

Циклом називається багаторазово повторювана дсукупність обчислень. Обчислювальний процес, що містить один або кілька циклів, називається циклічним. За кількістю виконання цикли поділяються на цикли з певним (заздалегідь заданим) числом повторень та цикли з невизначеним числом повторень. Кількість повторень останніх залежить від дотримання деякої

умови, що задає необхідність виконання циклу. При цьому умова може перевірятися на початку циклу - тоді мова йде про цикл з передумовою, або в кінці - тоді це цикл з умовою поста.

Окрім загально визначених існує ще декілька типів алгоритмів:

Імовірнісний (стохастичний) алгоритм дає напрямок вирішення задачі декількома шляхами або способами, що приводять до ймовірного досягнення результату.

Евристичний алгоритм - це такий тип алгоритмів, при якому досягнення кінцевого результату програми дій однозначно не визначено, так само як не позначена вся послідовність дій, так само які і не виділені всі дії виконавця. До евристичних алгоритмах відносять, наприклад, інструкції та розпорядження. У цих алгоритмах використовуються універсальні логічні процедури та способи прийняття рішень, засновані на аналогії, асоціаціях або минулому досвіді виконавця по вирішенню подібних завдань.

Допоміжний (підлеглий) алгоритм (процедура) - алгоритм, раніше і цілком використовується при алгоритмізації конкретного завдання. У деяких випадках при наявності однакових послідовностей вказівок (команд) для різних даних з метою скорочення запису також виділяють допоміжний алгоритм.

Для більшої зрозумілості та уникнення подвійних трактувань окремих кроків алгоритму опис алгоритм повинен бути формалізований за деякими правилами на основі конкретних образотворчих засобів. Серед існуючих способів задання алгоритмів належать такі:

- словесний;
- формульно-словесний;
- графічний;
- мова операторних схем;
- алгоритмічна мову.

Найбільшого поширення завдяки своїй наочності отримав графічний спосіб запису алгоритмів на основі блок-схем.

Блок-схемою називається графічне зображення логічної структури алгоритму, в якому кожен крок процесу обробки інформації представлений у

вигляді геометричних символів (блоків), що мають певну конфігурацію в залежності від характеру виконуваних операцій. Перелік символів, їх найменування, які відображаються ними функції, форма та розміри визначаються відповідними стандартами.

Таким чином, алгоритм виникає при вирішенні проблеми (інформування, управління). Для цього будується математична модель проблеми і на її базі ставиться задача та будуються методи її розв’язування. Під теорією алгоритмів розумітимемо систему фактів про властивості відповідних математичних об’єктів, якими подають алгоритм, у тому числі про взаємозв’язки між ними (їх алгебру, структури тощо).

1.2 Основні визначення та класифікація графів

Оскільки, будь який алгоритм є послідовністю зв'язаних кроків, тому його його можна розглядати як напрямлений граф, а для його графічного відображення оптимально використати елементи теорії графів.

Неформально граф можна розглядати як множину точок множину ліній (зі стрілками або без них), що зєднують ці точки.

Першою роботою теорії графів як математичної дисципліни вважають статтю Ейлера. Після того цікавітьсть до теорії графів періодично то зменшувалась, то збільшувалась, в залежності від розвитку суміжних дисциплін.

Граф – це пара $G=(V, E)$, де V - множина об'єктів довільної природи, називаних вершинами, та Е - множина пар $e_i = (v_{il}, v_{i2})$, що називаються ребрами.

В загальному випадку множина V і(або) сімейство E можуть містити нескінченне число елементів. Якщо порядок елементів, що входять в e_i, має значення, то граф називається орієнтованим, скорочено - орграф, в іншому випадку - неорієнтованим. Ребра орграфа називаються дугами.

Ступінь вершини графа - це число ребер, інцидентних даній вершині, причому петлі враховуються двічі. Оскільки кожне ребро інцидентне двом вершинам, сума ступенів усіх вершин графа дорівнює подвоєній кількості ребер.

Шляхом в графі (або маршрутом в орграфі) називається послідовність вершин і ребер (або дуг - в орграфі) виду v_0, (v_0, v_1), ..., (v_{n-1}, v_n), v_n. Число n називається довжиною шляху. Шлях без повторюваних ребер називається ланцюгом, без повторюваних вершин - простий ланцюгом. Шлях може бути замкнутим (v_0=v_n). Замкнутий шлях без повторюваних ребер називається циклом (або контуром в орграфі); без повторюваних вершин (крім першої та останньої) - простим циклом.

Графи служать зручним засобом опису зв'язків між об'єктами, прикладами можуть бути транспортні розвязки, архітектурні рішення тощо. Окрім того, теорія графів надає потужний інструментарій для проведення операцій над графами. Методи теорії графів широко застосовуються в дискретної математики. Без них неможливо обійтися при аналізі і синтезі різних дискретних перетворювачів: функціональних блоків комп'ютерів, комплексів програм тощо.

В даний час теорія графів охоплює великий спектр завдань та активно розвивається. Графи, як уже зазначалося в прикладах, є спосіб "візуалізації" зв'язків між певними об'єктами. Зв'язки ці можуть бути "спрямованими", як, наприклад, в генеологічному древі, або "ненаправленими" (мережа доріг з двостороннім рухом). Відповідно до цього в теорії графів виділяють два основних типи графів: орієнтовані (або спрямовані) і неорієнтовані.

Побудова математичного визначення графа здійснюється шляхом формалізації та "об'єктів", і "зв'язків" як елементів деяких (як правило, кінцевих) множин. Графи можна задавати наступними способами:

Можна задати граф як пару множин, слідуючи визначенню, однак цей спосіб досить громіздкий, та має більший інтерес для теоретиків. Розвиток алгоритмічних підходів до аналізу властивостей графів вимагає інших способів

опису графів, більш придатних для практичних обчислень, в тому числі з використанням ЕОМ.

Матриця інцидентності (рисунок 1.2,а). Припустимо, що всі вершини та всі ребра неорієнтованого графа або всі вершини та всі дуги (включаючи петлі) орієнтованого графа пронумеровані починаючи з одиниці. Граф (неорієнтований або орієнтований) може бути представлений у вигляді матриці nxm, де *n* - число вершин, а *m*- число ребер (або дуг). Значення елементів матриці визначаються наступним чином: якщо ребро xi і вершина vj інцидентні, то значення соотвествующего елемента матриці дорівнює одиниці, в іншому випадку значення дорівнює нулю. Для орієнтованих графів матриця інцидентності будується за наступним принципом: значення елемента рівне -1, якщо ребро x_i виходить з вершини v_j; дорівнює 1, якщо ребро x_i заходить в вершину v_j, і дорівнює 0 в іншому випадку.

Для неорієнтованого графа елементи цієї матриці задаються на основі формули:

$$a_{ij} = \begin{cases} 1, \text{ якщо для } i \text{ вершини } j \text{ ребро інцеденти} \\ 0, \text{інакше} \end{cases}.$$

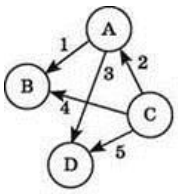
Для орієнтованого графа елементи матриці задаються так:

$$a_{ij} = \begin{cases} 1, \text{ якщо для } i \text{ вершини } j \text{ дуга вихідна} \\ -1, \text{якщо для } i \text{ вершини } j \text{ дуга вхідна}. \\ 0, \text{інакше} \end{cases}$$

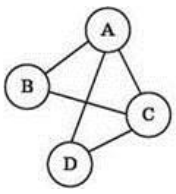
Проте представлення алгоритму у вигляді матриці інцеденцій не є репрезентативним, тому даний підхід не є доцільним

Матриця суміжності (рисунок 1.2,б). Це квадратна матриця розмірності *n*x*n*, де *n* - кількість вершин. Якщо вершини v_i і v_j суміжні, тобто якщо існує ребро, що їх з'єднує, то відповідний елемент матриці дорівнює одиниці, в іншому випадку він дорівнює нулю. Правила побудови даної матриці для

орієнтованого та неорієнтованого графів не відрізняються. Матриця суміжності більш компактна, ніж матриця інцидентності. Слід зауважити, що ця матриця також сильно розріджена, проте в разі неорієнтованого графа вона є симетричною відносно головної діагоналі, тому можна зберігати не всю матрицю, а тільки її половину (трикутну матрицю).



	A	B	C	D
1	-1	1	0	0
2	1	0	-1	0
3	-1	0	0	1
4	0	1	-1	0
5	0	0	-1	1



	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

а) б)

Рисунок 1.2 –Задання графів на основі матриці інцеденцій (а) та матриці суміжностей (б)

Список суміжності (інцидентності) (рисунок 1.3). Являє собою структуру даних, яка для кожної вершини графа зберігає список суміжних з нею вершин. Список являє собою масив вказівників, *i*-ий елемент якого містить вказівник на список вершин, суміжних з *i*-ою вершиною.

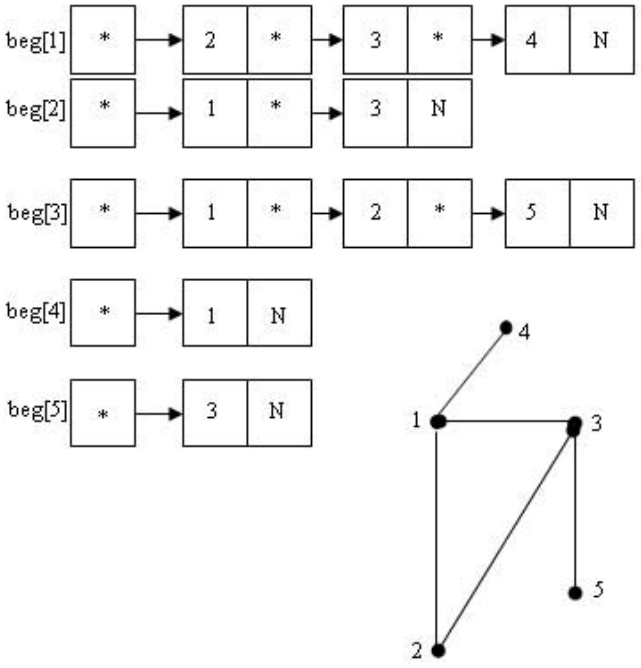


Рисунок 1.3 – Приклад опису графа на основі списку вказівників

Список суміжності більш ефективний у порівнянні з матрицею суміжності, так як виключає зберігання нульових елементів.

Список списків. Граф відображається у вигляді деревоподібної структуру даних, в якій одна гілка містить списки вершин, суміжних для кожної з вершин графа, а друга гілка вказує на чергову вершину графа. Такий спосіб представлення графа є найбільш оптимальним для представлення алгоритмі. Перевагами даного підходу є те, що таке представлення є найбільш наочне, та дозволяє в значній мірі відтворити структуру програмних додатків. Описані способи подання графа не вичерпують всіх можливих варіантів.

1.3 Програмні засоби створення та опису алгоритмів в графічній формі

Вивчення мов програмування включає в себе побудову логічних схем різних алгоритмів. Для цього існує цілий спектр програм-редакторів:

Diagram Designer - популярна програма, яка не потребує багато місця на жорсткому диску, проте достатньо функціональна. Підтримується практично усіма версіями операційної системи Windows. За допомогою неї можна створити як просту блок-схему, так і складний електричний ланцюг.

Інтерфейс програми простий і схожий на текстовий редактор. Включає в себе вікно для побудови, панель інструментів і список поточних елементів. Різноманітна колірна гамма дозволяє виділяти важливі фрагменти, створювати яскраві, унікальні блок-схеми алгоритмів. Розібратися в програмі зможе навіть користувач без досвіду роботи з даним типом програмних продуктів. Процес встановлення та використання стандартний для програмних додатків розроблених для Windows орієнтованих інтерфейсах.

FCEditor - дозволяє створювати схеми, управляти ними, переводити в графічний формат. Присутня функція, що дозволяє автоматично визначати розмір блоків, положення стрілок тощо. В комплекті передбачена велика

кількість готових шаблонів та наробок. Розроблена функція конвертації програмного коду в графічне представлення у вигляді блок-схем.

Autoflowchart - має просте оформлення та зручний інтерфейс. Робоче вікно включає в себе:

– текст коду на мові програмування;

– деревоподібне представлення структури проекту;

– блок схему.

Додаткові функції дозволяють виділяти частину тексту, згортати вкладені структури. Зроблені роботи можна зберігати як в графічному форматі так і в текстовому документі.

Flying Logic - призначена для швидкої та якісної побудови блок-схем алгоритмів. Інтерфейс програми містить найнеобхідніші функції, виключаючи складні налаштування функціонування програми. Додавання блоку відбувається в один клік миші, до того ж на ньому можна відобразити додаткову інформацію. В подальшому отримані проміжні результати, можна об'єднати в єдину систему.

Інша група програмних продуктів розрахована на он-лайн роботу в глобальній мережі інтернет:

Draw.io – відмінний додаток для створення діаграм і блок-схем алгоритмів. Сервіс має великий інструментарій, який дозволяє вибирати готові замальовки шаблонів блоків, створювати власні, використовувати зображення та малюнки з пам'яті комп'ютера.

Працювати в цій програмі можуть одночасно кілька людей, так як вона дозволяє прив'язувати один документ до різних комп'ютерів. У порівнянні з іншими редакторами Draw.io можна використовувати без реєстрації, проте в даному режимі роботи в програми є ряд функціональних обмежень.

Lucichart.com - Один з кращих онлайн сервісів. В результаті роботи з даним сервісом алгоритми виходять яскраві та інтерактивні. Даний варіант ідеально підійде для презентації проектів. Користуватися додатком можна після проходження реєстрації, яка не займе більше 3 хвилин. Інтерфейс простий, зручний, зрозумілий.

Google Drawing - зручний додаток для запису алгоритмів у графічному форматі. Для роботи необхідна реєстрація, після якої створюється обліковий запису та відкриється широкий спектр можливостей для створення проектів. Зберегти роботу можна в графічних форматах. Додаток розповсюджується за безкоштовною ліцензією.

Microsoft Visio - програма для побудови креслень та діаграм, що дозволяє візуалізувати, досліджувати та поширювати складну інформацію. Так, наприклад, важкий для розуміння текст і таблиці, описи процесів, плани поверхів та багато іншого можна представити у вигляді простих і наочних схем.

Visio служить робочим інструментом для широкого кола користувачів: від офіс-менеджера до бізнес-аналітика, від програміста до інженера-будівельника, причому користувачі, які не вміють малювати і не мають навичок дизайнерської роботи, можуть швидко та зручно створювати професійно оформлені схеми.

Visio – програмне рішення для побудови діаграм від Microsoft. За словами розробників, Visio допомагає перетворити технічні та бізнес-концепції в візуальну форму. І дійсно, цей пакет з сімейства Microsoft Office призначений виключно для малювання діаграм. Visio має деякі додаткові можливості, проте - це тільки засіб для ілюстрування документів MS Office. Візуалізуючі можливості Visio дуже широкі:

Використовуючи зумовлені фігури Visio, drag-and-drop та майстри, можна швидко та просто створювати зрозумілі і інформативні діаграми.

Можливості Visio можна легко розширювати, використовуючи нові шаблони бізнес-діаграм.

В Visio можна прототипувати інтерфейс додатків за допомогою вбудованих шаблонів призначеного для користувача інтерфейсу Microsoft Windows, що дозволяє створювати модель користувальницького інтерфейсу в стандартному Windows стилі.

Можна легко малювати діаграми мережевих ресурсів, що ілюструють розгортання нового програмного забезпечення на існуючі мережеві ресурси.

Visio Professional також тісно інтегрується з Microsoft Office Project, що дозволяє, наприклад, імпортувати звідти завдання для членів команди.

За допомогою шаблонів UML ви можете створювати UML-діаграми статичної структури ПЗ або проводити зворотне проектування.

Результати порівняння різних програмних засобів наведено в таблиці 1.1:

Таблиця 1.1 – Порівняння програмних засобів створення блок-схем

Назва програми	Підтримка стандартів	Наявність шаблонів	Конвертація програмного коду	Підтримка UML	Інші типи схем
Diagram Designer	+	+	-	+	+
FCEditor	+	+	-	-	+
Autoflowchart	+	+	-	-	+
Flying Logic	+	+	+-	-	+
Draw.io	+	-	-	-	+
Lucichart.com	+	-	+-	-	+
Google Drawing	+	+	+-	+	+
MS Visio	+	+	+-	+	+

Проведений аналіз показав, що на сьогоднішній день існує досить велика кількість програмних засобів з великою кількістю функціональних можливостей для створення блок-схем алгоритмів як в он-лайн так і в офф-лайн режимах. Серед основних функцій, що підтримуються програмами даного типу слід відмітити: графічний редактор з великою кількістю геометричних примітивів, функції конвертації блок-схем в програмний код на найбільш розповсюджені мови програмування, можливість створення інтерактивних динамічних презентацій своїх проектів. Серед основних недоліків велика кількість надлишкових функцій винесені на головні вікна програмних додатків.

1.4 Висновки та постановка задачі

В даному розділі проаналізовано задачі, що виникають у розробників програмних засобів побудови та візуалізації блок-схем обчислювальних алгоритмів. Проведено аналіз та класифікацію існуючих груп алгоритмів та досліджено можливості використання елементів теорії графів для проектування програмного додатку створення блок-схем. Розглянуто сучасні програми-аналоги побудови та візуалізації блок-схем як для роботи в офф-лайн так і он-лайн режимах.

Для досягнення поставленої мети необхідно розв’язати наступні задачі.

1. Провести аналіз та класифікацію існуючих типів алгоритмів.
2. Провести аналіз елементів теорії графів для побудови дерев.
3. Проаналізувати існуючі програмні засоби побудови та візуалізації блок-схем.
4. Дослідити цифрову бібліотеку для роботи з графами TSimpleGraph.
5. Спроектувати структуру програмного засобу побудови та візуалізації блок-схем алгоритмів.
6. Реалізувати програмний додаток побудови та візуалізації блок-схем, провести його тестування та порівняльний аналіз з програмами аналогами.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ МОНІТОРИНГУ МЕРЕЖЕВИХ ІНТЕРФЕЙСІВ

2.1 Технологійї створення та запису алгоритмів

Щоб алгоритм виконав своє призначення, його необхідно будувати за певними правилами. Тому потрібно говорити все ж не про властивості алгоритму, а про правила побудови алгоритму, або про вимоги, що пред'являються до алгоритму.

Перше правило - при побудові алгоритму насамперед необхідно задати множину об'єктів, з якими буде працювати алгоритм. Формалізоване (закодоване) представлення цих об'єктів носить назву даних. Алгоритм приступає до роботи з певним набором даних, які називаються вхідними, і в результаті своєї роботи видає дані, які називаються вихідними. Таким чином, алгоритм перетворює вхідні дані у вихідні. Поки відсутні формалізовані вхідні дані, не можливо побудувати алгоритм.

Друге правило - для роботи алгоритму потрібна пам'ять. У пам'яті розміщуються вхідні дані, з якими алгоритм починає працювати, проміжні дані та вихідні дані, які є результатом роботи алгоритму. Пам'ять є дискретною, тобто складається з окремих комірок. Поіменована комірка пам'яті має назву змінної. В теорії алгоритмів розміри пам'яті не обмежуються, тобто вважається, що ми можемо надати алгоритму будь-який необхідний для роботи обсяг пам'яті.

Третє правило - дискретність. Алгоритм будується з окремих послідовних кроків (дій, операцій, команд). Множина кроків, з яких складено алгоритм, повинна бути визначеною і зрозумілою виконавцю.

Четверте правило - детермінованість. Після кожного кроку необхідно вказувати, який крок виконується наступним, або давати команду зупинки.

П'яте правило - збіжність (результативність). Алгоритм повинен завершувати роботу після деякого числа зліченних кроків. При цьому необхідно вказати, що вважати результатом роботи алгоритму.

При проектуванні алгоритмів використовуються два основних підходи.

Метод послідовної деталізації завдання («зверху-вниз») полягає в тому, що вихідне складне завдання розбивається на підзадачі. Кожна з підзадач розглядається та вирішується окремо. Якщо будь-які з підзадач складні, вони також розбиваються на підзадачі. Процес триває до тих пір, поки підзадачі не зведуть до елементарних. Рішення окремих підзадач потім збираються в єдиний алгоритм рішення вихідної задачі. Метод широко використовується, так як дозволяє вести розробку загального алгоритму одночасно кільком програмістам, вирішальним локальні підзадачі. Це необхідна умова швидкої розробки програмних продуктів.

Складальний метод («знизу-вгору») полягає у створенні великої кількості програмних модулів, що реалізують рішення типових задач. При вирішенні складного завдання програміст може використовувати розроблені модулі в якості допоміжних алгоритмів (процедур). У багатьох системах програмування вже існують подібні набори модулів, що істотно спрощує та прискорює проектування складного алгоритму.

Для швидкого читання алгоритму слід його записувати у зрозумілій для відповідного кола виконавців формі, серед найбільш розповсюджених форм задання алгоритмів є такі як.

Словесний опис. Даний підхід представляє структуру алгоритму природною мовою. Наприклад, будь-який прилад побутової техніки (праска, електропила тощо) має інструкцію по експлуатації (словесний опис алгоритму, відповідно до якого даний прилад повинен використовуватися). Ніяких правил складання словесного опису не існує. Запис алгоритму здійснюється в довільній формі на природній мові. Даний спосіб опису не має широкого розповсюдження, так як строго не формалізуємо (під «формальним» розуміється то, що опис абсолютно повне і враховує всі можливі ситуації, які можуть виникнути в ході вирішення); допускає неоднозначність тлумачення при описі деяких дій; даний опис може бути перенасичений описовим характером. Наприклад алгоритм вибору одягу відносно температури

навколишнього середовища:"визначити температуру повітря; якщо температура нижче 0, то надіти шубу, інакше надіти куртку".

Псевдокод - опис структури алгоритму на природній, частково формалізованій мові, що дозволяє виділити основні етапи вирішення задачі, перед точним його записом на мові програмування. У псевдокоді використовуються деякі формальні конструкції та загальноприйнята математична символіка. Строгих синтаксичних правил для запису псевдокоду не існує. Це полегшує запис алгоритму при проектуванні та дозволяє описати алгоритм, використовуючи будь-який набір команд. Проте в псевдокоді зазвичай використовуються деякі конструкції, властиві формальним мовам, що полегшує перехід від псевдокоду до запису алгоритму на мові програмування. Єдиного або формального визначення псевдокоду не існує, тому можливі різні варіанти псевдокоду, що відрізняються набором використовуваних слів і конструкцій.

Задання алгоритму за допомогою графів. Використовуючи матриці інциденцій і суміжності, інші властивості графу обчислюють характеристики графу чи його частин — шляхів, дерев тощо. Ці характеристики мають свої інтерпретації та аналогії на характеристиках алгоритму (складність, швидкодія тощо). Наприклад, знаючи час виконання операцій, які відображені вузлами та гілками графу досліджуваного алгоритму, та використавши поняття шляху можна побудувати алгоритм обчислення часу роботи досліджуваного алгоритму; автоматичним підрахунком числа вузлів, гілок можна встановити складність алгоритму (надаючи певним операціям відповідну "вагу"). Приклад задання алгоритму за допомогою графу наведено на рисунку 2.1:

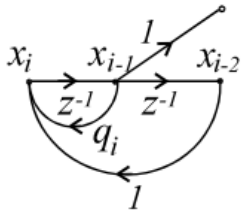


Рисунок 2.1 - Подання алгоритму у вигляді графу (цифрового кола)

Блок-схема - опис структури алгоритму за допомогою геометричних фігур з лініями-зв'язками, які показують порядок виконання окремих інструкцій. Цей спосіб має ряд переваг. Завдяки наочності, він забезпечує «читабельність» алгоритму та явно відображає порядок виконання окремих команд. У блок-схемі кожній формальній конструкції відповідає певна геометрична фігура або пов'язана лініями сукупність фігур. Основні графічні елементи наведені в таблиці 2.1:

Таблиця 2.1 – Графічні позначення елементів згідно ДСТУ ISO 5807:2016

Науменування	Позначення	Функція
Термінатор		Елемент відображає вхід із зовнішнньо середовище або вихід в нього.
Процес		Елемент відображає одну або кількох операцій, обробку даних будь-якого виду (зміна значення даних, форми подання, розташування).
Рішення		Елемент відображає оброку умови, рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.
Зумовлений процес		Елемент відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі).
Дані		Елемент відображає перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

Продовження Таблиці 2.1

Цикл з параметром		Елемент відображає заголовок циклу з параметром. У ньому через крапку з комою вказуються ім'я змінної (параметра) з початковим значенням, граничне значення параметра (або умова виконання циклу), крок зміни параметра.
Межа циклу		Елемент складається з двох частин - відповідно, початок і кінець циклу - операції, що виконуються всередині циклу, розміщуються між ними. Умови циклу і збільшення записуються всередині символу початку або кінця циклу - в залежності від типу організації циклу. Часто для зображення на блок-схемі циклу замість цього символу використовують символ рішення, вказуючи в ньому умову, а одну з ліній виходу замикають вище в блок-схемі.
З'єднувач		Елемент відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.
Коментар		Елемент використовується для детальнішої інформації про кроки, процесу або групи процесів. Опис поміщається з боку квадратної дужки і охоплюється нею по всій висоті. Пунктирна лінія йде до описуваного елементу, або групи елементів. Також символ коментаря слід використовувати в тих випадках, коли обсяг тексту в будь-якому іншому символі перевищує його обсяг.

На практиці в якості виконавців алгоритмів використовуються спеціальні автомати - комп'ютери. Тому алгоритм, призначений для виконання на комп'ютері, повинен бути записаний на зрозумілій йому мові. І тут на перший план висувається необхідність точного запису команд, яка не залишає місця для довільного тлумачення їх виконавцем. Отже, мова для запису алгоритмів повинен бути формалізований. Така мова прийнято називати мовою програмування, а запис алгоритму на цій мові - програмою для комп'ютера. Програма, створювана людиною - програмістом, являє собою текст, що складається із знаків, як правило букв, цифр і спеціальних знаків. Знаки в тексті програми часто об'єднані в послідовності - ключові слова, слова об'єднані в пропозиції мови програмування - оператори. Кожен оператор, як правило, записується в окремий рядок тексту програми. Таким чином текстове програмування являє собою ієрархічну послідовність знаків, слів, операторів, що записуються і читаються послідовно, як звичайний текст людської писемності.

2.2 Цифрова бібліотека опису та опрацювання графів TSimpleGraph

Для реалізації роботи програми створення та візуалізації блок-схем обчислювальних алгоритмів було обрано об'єктно-орієнтований підхід з використанням елементів теорії графів для отримання взаємозв'язних послідовностей графічних примітивів, що віповідають елементам блок-схеми. Для роботи з графами та їх елементами використано цифрову бібліотеку TSimpleGraph.

TSimpleGraph - це візуальний компонент, який забезпечує простий користувальницький інтерфейс для малювання графів. Дуги та вузли графа повністю настроюються таким чином, що дозволяють створювати практично будь-які графи довільної складності.

В даний час у бібліотеці є сім типів графічних об'єктів для базових геометричних форм: трикутник, ромб, прямокутник, круглий прямокутник, п'ятигранник, шестигранник та еліпс. Крім цього, взявши за основу окремий клас та використавши одним або двома методи, користувач можете створити власні графічні примітиви для візуалізації окремих елементів графів. З'єднувальні лінії (створені як об'єкти) - це об'єкт управління з дуже гнучкими параметрами, та може бути використаний для пов'язування окремих елементів (частини схеми) до інших об'єктів або з'єднати їх разом.

Проаналізуемо окремі класи, що входять до даної бібліотеки:

TSimpleGraph = class (TCustomControl). TSimpleGraph - це вікно управління, яке забезпечує полотно для нанесення графа. Він також управляє співвідношенням між об'єктами графа та додатком.

Параметри:

TGraphCommandMode = (cmViewOnly, cmEdit, cmInsertNode, cmLinkNodes) Визначає, які типи команд доступні користувачеві:

– cmViewOnly - користувач не може змінити граф;

– cmEdit користувач може переміщувати, змінювати розмір та видаляти вузли;

– cmInsertNode користувач може вставити новий вузол;

– cmLinkNodes користувач може пов'язати два вузли.

DefaultLinkClass: TGraphLinkClas) - визначає клас, який слід використовувати для нового об'єкта посилання.

DefaultKeyMap - вказує, чи контролюється внутрішній контроль комбінацій клавіш, введених користувачем.

DefaultNodeClass: TGraphNodeClass - визначає клас, який слід використовувати для об'єкта нового вузла.

FreezeTopLeft - вказує, чи можуть об'єкти переміщатися за верхній/лівий край на екрані.

GraphBounds: TRect - вказує найменший обмежувальний прямокутник всіх видимих об'єктів графа у блоці графів.

GridColor: TColor - визначає колір сітки.

GridSize: TGridSize (TGridSize = 4..128) - визначає відстань сітки в блоці графів вздовж вісі x і y.

HideSelection - визначає, чи залишається візуальне зображення вибраних об'єктів grpah, коли фокус переходить до іншого елемента керування.

Події на які реагує даний клас:

OnCanLinkNodes: TCanLinkNodesEvent – Виникає при спробі об'єднати два вузли.

OnCanMoveResizeNode: TCanMoveResizeNodeEvent - виникає при спробі змінити розмір або перемістити вузол.

OnCommandModeChange: TNotifyEvent - виникає при зміні CommandMode.

OnGraphChange: TNotifyEvent - виникає при зміні графіка.

OnObjectContextPopup: TGraphContextPopupEvent - виникає, коли користувач клацнув правою клавішею на графічному об'єкті.

OnObjectDblClick: TGraphNotifyEvent - виникає, коли користувач подвійним клацанням на об'єкті графіка.

OnObjectInsert: TGraphNotifyEvent – виникає, коли новий граф об'єкта додається до графіка.

OnObjectRemove: TGraphNotifyEvent – виникає, коли об'єкт графіка видаляється з графіка.

OnObjectSelect: TGraphNotifyEvent - виникає, коли граф об'єкта вибирається або знімається.

OnInfoTip: TGraphInfoTipEvent- виникає, коли користувач призупиняє мишу над об'єктом на графіку.

OnMoveResizeNode = TGraphNodeResizeEvent- виникає, коли вузол переміщується або змінюється.

OnObjectAfterDraw: TGraphObjectDrawEvent- виникає, після нанесення об'єкта графа на полотно.

OnObjectBeforeDraw: TGraphObjectDrawEvent- виникає, безпосередньо перед тим, як об'єкт графа намальований на полотні.

OnObjectBeginDrag: TGraphBeginDragEvent - виникає, відразу після того, як об'єкт графа ініціюється для перетягування. Причому, інший виділений об'єкт може слідувати операції перетягування, однак ця подія відбувається лише для джерела перетягування.

OnObjectEndDrag: TGraphEndDragEvent - виникає, відразу після того, як об'єкт графа буде перетягнено.

OnObjectContextPopup: TGraphContextPopupEvent - виникає, коли користувач клацнув правою клавішею на графічному об'єкті.

OnObjectClick: TGraphNotifyEvent - виникає, коли користувач натискає на об'єкт графа.

OnObjectDblClick: TGraphNotifyEvent - виникає, коли користувач подвійним клацанням на об'єкті графіка.

OnObjectHook: TGraphHookEvent - виникає, коли посилання підключено до об'єкта графіка.

OnObjectUnhook: TGraphHookEvent - виникає, коли посилання від'єднується від об'єкта графа.

OnObjectInitInstance: TGraphNotifyEvent - виникає, при створенні нового об'єкта графа. На цьому етапі об'єкт графа ще не вставлений на графік.

OnObjectInsert: TGraphNotifyEvent - виникає, при додаванні нового графа до графа.

OnObjectRemove: TGraphNotifyEvent - виникає, коли об'єкт графіка видаляється з графіка.

OnObjectChange: TGraphNotifyEvent - виникає, при зміні об'єкта графа.

OnObjectSelect: TGraphNotifyEvent - виникає, коли граф об'єкта вибирається або знімається. Використовуйте виділену властивість об'єкта, щоб визначити, чи він обраний або знятий.

OnObjectRead: TGraphStreamEvent - виникає, при зчитуванні параметрів властивості об'єкта графа, а об'єкт графа має деякі власні дані. Обробник події може читати власні дані з потоку, переданого як параметр.

OnObjectWrite: TGraphStreamEvent - виникає, при написанні властивостей об'єкта графа, а властивість HasCustomData об'єкта графа - True. Обробник подій може записувати власні дані в потоці, переданому як параметр.

OnObjectMouseEnter: TGraphNotifyEvent - виникає, коли покажчик миші рухається над об'єктом графіка.

OnObjectMouseLeave: TGraphNotifyEvent - виникає, коли покажчик миші рухається з об'єкта графіка.

OnZoomChange: TNotifyEvent - виникає, при зміні властивостей масштабу.

Окрам головного класу бібліотеки існують ще ряд допоміжних:

TPolygonalNode - це базовий клас для всіх вузлів фігур полігону.

TRectangularNode вигляд вузла форми прямокутника.

TRoundRectangularNode вигляд навколо вузла прямокутної форми.

TEllipticNode вигляд вузла форми еліпсу.

TTriangularNode вигляд вузла форми трикутника.

TRhomboidalNode стиль ромбовидної форми.

TPentagonalNode вигляд вузла п'ятикутної форми.

TGraphScrollBar відображає горизонтальну або вертикальну смугу прокрутки на графічному дизайнері. Властивості TGraphScrollBar схожі з TControlScrollBar Delphi.

TMemoryHandleStream - це потік, який зберігає свої дані в динамічній пам'яті, виділеній у глобальній купі Windows. Вона має всі властивості та методи класу предків, тобто TMemoryStream. Крім того, інтерфейс цього потоку забезпечує глобальну можливість виділення пам'яті, яку можна використовувати в функціях Windows API, для яких потрібен хендл до цього файлу.

Оскільки дана бібліотека реалізована на базисі мови Delphi та містить усі необхідні класи та методи для вирішення поставлених задач, то вона цілком підходить для реалізації програмного додатку створення та візуалізації блок-схем обчислювальних алгоритмів.

Для проведення попереднього моделювання розроблювального програмного додатку, використовуються ряд UML діаграм. Вибір UML пов'язаний з тим, що:

UML об'єктно-орієнтована, в результаті чого методи опису результатів аналізу та проектування семантично близькі до методів програмування на сучасних ОО-мовами;

UML дозволяє описати систему практично з усіх можливих точок зору та різні аспекти поведінки системи;

Діаграми UML порівняно прості для читання після досить швидкого ознайомлення з його синтаксисом.

В першому варіанті розгдатиметься розроблювальна система як сукупність акторів та прецедентів з якими вони можуть взаємодіяти під час роботи з програмою (рисунок 2.2):

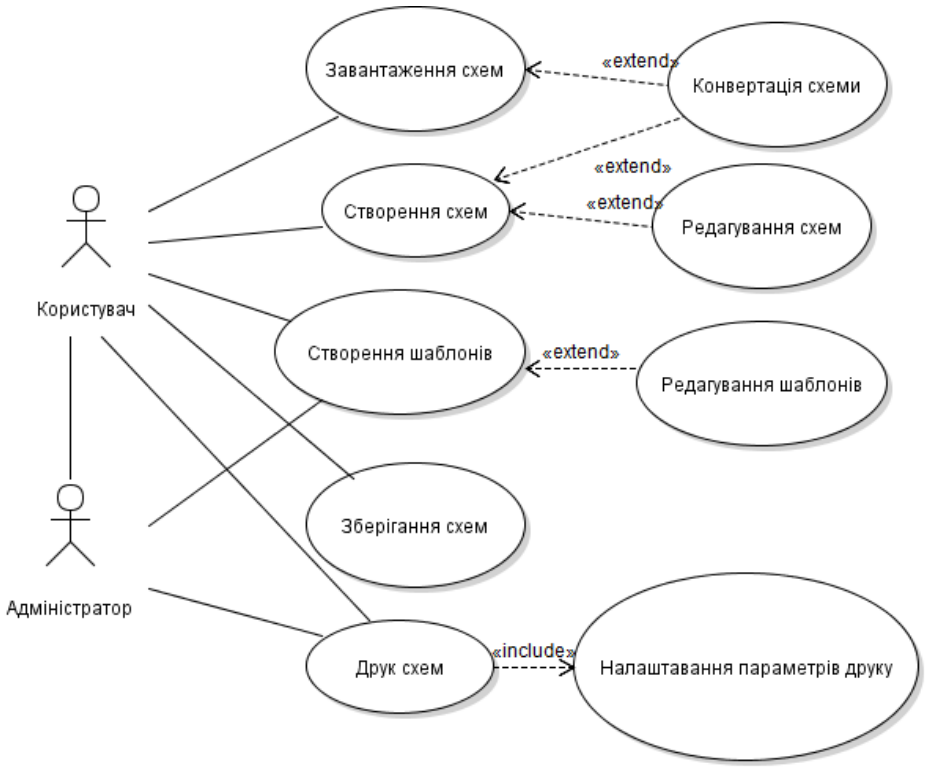


Рисунок 2.2 – Діаграма прецедентів програмного додатку опису алгоритмів за допомогою блок-схем

При моделюванні було виділено два типа акторів, що будуть взаємодіяти з програмною системою. Актори типу "Користувач" мають змогу створювати, редагувати, зберігати графічні схеми алгоритмів, для подальшого їх друку або модифікації. "Користувач" повинен володіти основними принципами розробки алгоритмів та знання основ та вимог до відображення алгоритмів в графічному форматі за допомогою блок-схем. Група "Адмінастратор" взаємодіють з системою з метою внесення змін в її роботу або для створення/додавання нових графічних шаблонів геометричних примітивів.

Рисунок 2.3 – Діаграма послідовності

Аналіз послідовності кроків роботи з програмним додатком показав, що:

– кількість операцій, які повинен виконати користувач є незначною;

– відсутні повторні дії, що не несуть корисного навантаження;

– використання програмного додатку не вимагає глобоких початкових знань.

Виділення двох типів акторів, що будуть взаємодіяти з розробленим програмним додатком є цілком достатнім для коректної роботи системи та виконання усіх поставлених перед програмним додатком завдань.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМИ МОНІТОРИНГУ МЕРЕЖЕВИХ ІНТЕРФЕЙСІВ

3.1 Опис структури програмного додатку

Вивід графіки в Windows здійснюється за допомогою функцій GDI (Graphics Device Interface - інтерфейс графічних пристроїв). Функції GDI використовують як сама операційна система, так і програми, що функціонують під управлінням Windows. Функції, реалізовані в GDI, є апаратно-незалежними. Це означає, що створення графічних зображень не залежить від відеокарти, яка встановлена на комп'ютері. Недоліки GDI:

- не використовує прискорення;
- занадто повільний;
- підтримує тільки двовимірну графіку.

Тому GDI не призначений, наприклад, для створення складних програм для відтворення візуальної інформації. Для побудови зображень в Delphi була введена ієрархія класів, пов'язаних з графічними об'єктами (рисунок 3.1), а також розроблені відповідні компоненти.

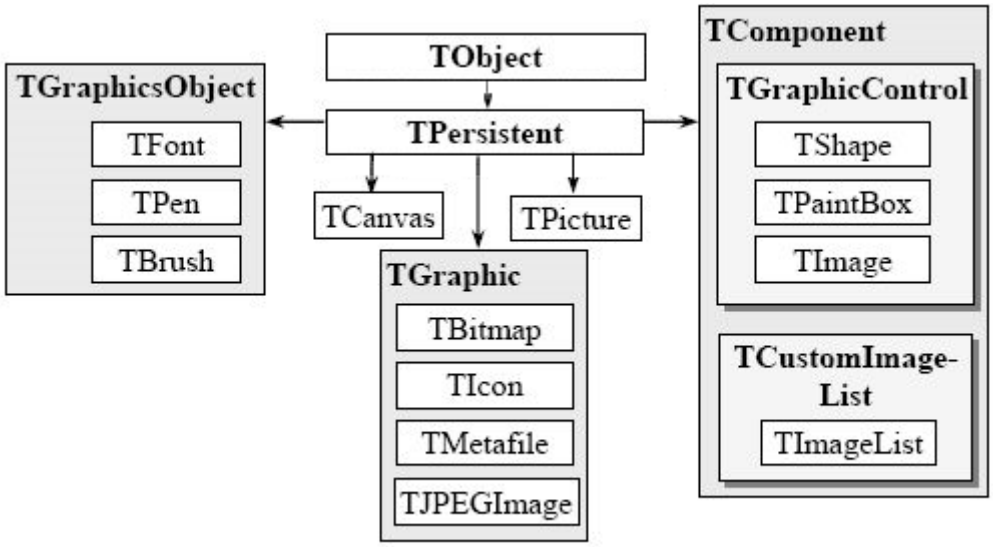


Рисунок 3.1 – Ієрархія класів для обробки візуальної статичної інформації

Середовище візуального програмування Delphi в повній мірі надає користувачеві можливість розробляти програми, за допомогою яких можна отримати графічні зображення: схеми, креслення, текст та ілюстрації.

У Delphi існує ряд спеціальних компонентів для виведення готових зображень (малюнків або фотографій) з графічних файлів (компонент Image), деяких геометричних фігур (компонент Shape), графіків і діаграм (компонент Chart) тощо. Наприклад, за допомогою компонента Shape можна зобразити на формі коло, еліпс,квадрат, прямокутник або прямокутник із закругленими кутами. Для цього досить помістити на форму компонент Shape і налаштувати його за допомогою відповідних властивостей (властивість Shape дозволяє вибрати фігуру, властивість Brush відповідає за її фоновий колір, а властивість Pen - за товщину і колір її межі).

Основним класом, що дозволяє використовувати великі графічні можливості Delphi є клас TCanvas. Цей клас має безліч властивостей і методів. Об'єкт Canvas (канва, полотно) цього класу є властивістю форми і багатьох графічних компонентів (Image, PaintBox, BitMap і ін.). Він представляї собою область компонента, на якій можна малювати або відображати готові зображення. Канва містить властивості та методи, що істотно спрощують графіку Delphi.

Для реалізації програмного додатку створення та редагування блок-схем спроектована архітектура програмного додатку. Під архітектурою розуміється сукупність компонентів програми, а також зв'язки та способи організації інформаційного обміну між ними. Першим завданням, яке необхідно вирішити при розробці структурної схеми системи, є завдання визначення складових її компонентів.

Виходячи з аналізу вимог, що пред'являються до системи, визначається набір всіх функцій, виконання яких програма повинна підтримувати. Далі отримані функції об'єднуються в логічно пов'язані між собою групи. Кожна з таких груп може стати одним з компонентів програмної системи.

Архітектура програми включає загальний опис системи. Без такого опису досить важко скласти узгоджену картину з множини дрібних деталей або хоча б

десятка окремих класів. Запропонована архітектура ілюструє те, що при її розробці були розглянуті альтернативні варіанти, і обґрунтовувано вибір остаточної організації системи.

Узагальнена архітектура програмного додатку наведена у вигляді структури на рисунку 3.1.

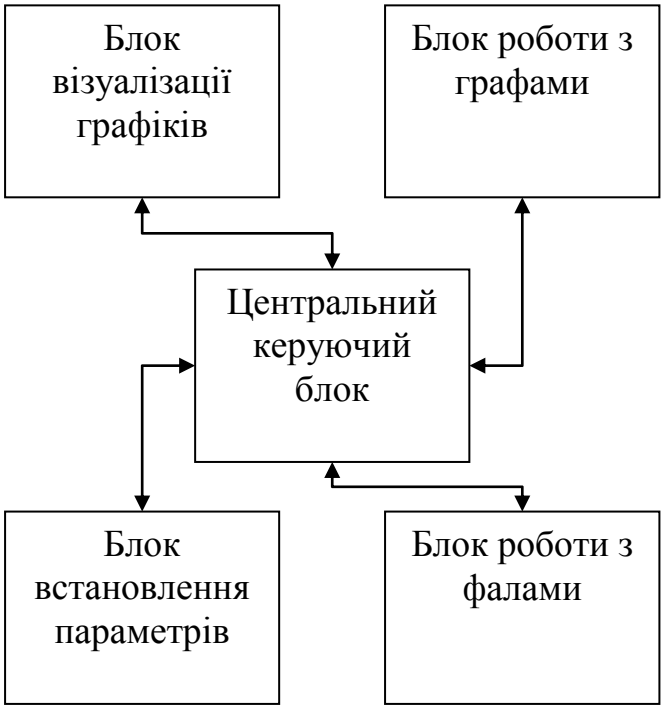


Рисунок 3.1 – узагальнена структура програмного додатку візуалізації блок-схем алгоритмів

Серед основних структурних блоків програмного додатку можна відзначити такі:

- “Блок встановлення параметрів” – набір процедур та функцій, що дозволяють користувачеві встановити параметри роботи програми або змінити характеристики зовнішній вигляд графічного інтерфейсу користувача. Серед додаткових можливостей присутні функції налаштування параметрів візуалізації блок-схем (колір, товщина ліній, параметри шрифтів тощо). Даний модуль носить допоміжний характер.
- “Блок роботи з фалами” – містить набір функцій для роботи з файловою системою робочої станції. Серед основних можливостей набір функцій для пошуку, відкриття, зберігання, друку файлів, що вмістять

розроблені блок-схеми алгоритмів. В основному прireалізації даного блоку використовувались windows-подібні діалогові вікна, що значно спрощує процес опанування програмним додатком.

- “Блок візуалізації графіків” – один з головний структурних блоків програмного додатку, що поєднує в собі множину процедур та функцій для виводу отриманого графа на екран робочої станції.
- “Блок роботи з графами” – набір функцій та інтерфейсів для організації взаємозв’язку між бібліотекую роботи з графами та програмним додатком. Також в даному блоці передбачені функції кодування графічних елементів блок-схеми у зрозумілий для користувача вигляд.
- “Центральний керуючий блок” – головний елемента архітектури програмного додатку, що забезпечує можливість взаємодії окремих елементів програмного додатку. Серед додаткових функцій, що реалізовані в даному структурному елементі, є можливість перевірки цілісновті та коректності роботи програмного додатку.

З запропонованої архітектури чітко видно відповідальність кожного компонента. Компонент має одну область відповідальності, а взаємодія між окремими компонентами зведена до мінімуму. Архітектура показує правила комунікації між компонентами програми та описує, які інші компоненти даний компонент може використовувати безпосередньо, які побічно, а які взагалі не повинен використовувати.

Графічний інтерфейс спроектовано на етапі аналізу вимог. В архітектурі описано головні елементи формату графічного інтерфейсу (GUI). Зручність графічного інтерфейсу користувача може в підсумку визначити популярність або провал програмного продукту. При проектуванні графічного інтерфейсу було прийнято рішення проподіл головного вікна програмного додатку на окремі функціональні області. Такий поділ дозволив зменшити візуальне навантаження на користувача та дозвлив зменшити час на опанування програмним додатком користувачів з невеликим досвідом та навиками роботи з персональними комп’ютерами та операційними системами з графічним

інтерфейсом. Розроблений графічний інтерфейс користувача наведено на наступному рисунку (рисунок 3.2).

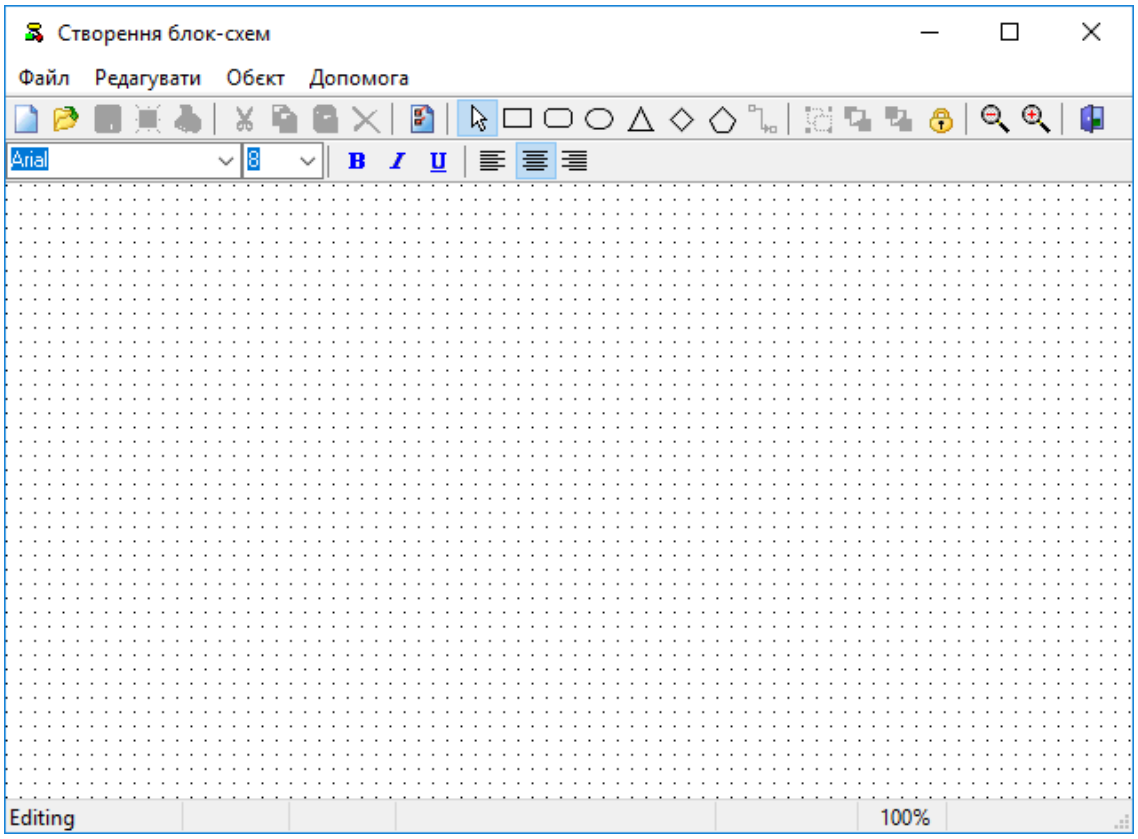


Рисунок 3.2 – Загальний вигляд головного вікна програмного додатку

Серед основних структурних елементів графічного інтерфейсу слід відмітити:

- основна рбоча область програми- на ній відбувається процес побудови та візуалізації блок-схеми алгоритмів. Дана область займає 90% від хагальної кількості головного вікна програмного додатку, оскільки на неї покладено основіні функції по взаємодії між програмою та користувачем.
- Блок встановлення параметрів роботи програми та характеристик виводу блок-схеми на екран. Серед параметрів, які може редагувати користувач є : встановлення типу, розміру, кольору шрифтів, місце розташування набписів на елементах блок-схеми, параметри виводу графічних елементів тощо.
- Головне випадаюче меню програмного додатку – надає користувачеві можливості роботи з фаловою системою (створення/відкриття/збереження файлів), функції для редагування блок-схем,

встановлення параметрів об’єктів на графі, дозволяє отримати системні підказки для підшвидшення роботи з програмним додатком.

Архітектура програми є модульною, щоб графічний інтерфейс можна було змінити, не зачіпаючи основну логіку програми.

3.2 Реалізація функції створення блок-схем алгоритмів

Під час реалізації програмного додатку була використана бібліотека SimpleGraph, що надає модливості роботи з зв’язними графами. Оскільки блок схеми та своїй суті є графами, то набір функцій з бібліотеки SimpleGraph оптимально підходить для реалізації програмного додатку. Одним з важливих етапів обрбки блок-схем є її редагування. Даний процес включає виділення потрібного графічного елемента блок-схеми. Функція для реалізаці даного алгоритму наведена нище:

```
if (SimpleGraph.SelectedObjects.Count > 0) then
begin
GraphObject := SimpleGraph.SelectedObjects[0];
cbxFontName.Text := GraphObject.Font.Name;
cbxFontSize.Text := IntToStr(GraphObject.Font.Size);
FormatBold.Checked := (fsBold in GraphObject.Font.Style);
FormatItalic.Checked := (fsItalic in GraphObject.Font.Style);
FormatUnderline.Checked := (fsUnderline in GraphObject.Font.Style);
end;
if (SimpleGraph.SelectedObjects.Count = 1) and
 (SimpleGraph.SelectedObjects[0] is TGraphNode) then
begin
Node := TGraphNode(SimpleGraph.SelectedObjects[0]);
case Node.Alignment of
```

```
taCenter: FormatCenter.Checked := True;
taLeftJustify: FormatAlignLeft.Checked := True;
taRightJustify: FormatAlignRight.Checked := True;
end;
StatusBar.Panels[1].Text := Format('(%d, %d)', [Node.Left, Node.Top]);
StatusBar.Panels[2].Text := Format('%d x %d', [Node.Width, Node.Height]);
PosFirstLine := Pos(#$D#$A, Node.Text);
if PosFirstLine <> 0 then
StatusBar.Panels[3].Text := Copy(Node.Text, 1, PosFirstLine)
else
StatusBar.Panels[3].Text := Node.Text;
```

Особливістю даної реалізації є вибір не самого графічного елемента, а вузла графа, який описує даний елемент. Такий підхід дозволяє більш точно опрацьовувати сусідні елементи блок-схеми, що значно спрощує сам процес створення схем алгоритмів.

Для полекшення процесу реалізації програми боло розроблено структуру для збереження ключових праметрів роботи з рограмним додатком. Вид даної структури наведено нище:

```
SSaveChanges   = 'Зберегти граф?';
SViewOnly      = 'Тільки перегляд';
SEditing       = 'Редагування';
SLinkingNodes  = 'З'єднати вузли';
SInsertingNode = 'Додати вузол';
SModified      = 'редагувати';
SNotModified   = '';
SUntitled      = 'Без підпису';
```

Серед полів даної структури задокументовано основіні функціональні параметри роботи програмного додатку.

3.3 Тестування програмного додатку створення та редагування блок-схем алгоритмів

Для проведення тестування розробленого програмного додатку проектування блок-схем було обрано робочу станцію з наступними технічними характеристиками:

- корпус Zalmanблок живлення 550B;
- вінчестер WD Caviar 500GB;
- відеокарта Asus PH-GTX1060-3G;
- оперативна пам'ять, 2Gb 2400GHz ;
- процесор AMD Ryzen 3 2200G BOX 120;
- материнська плата Asus B350M-E 90.

Характеристики монітора:

- діагональ дисплея 17";
- максимальна роздільна здатність дисплея 1920 x 1080;
- тип матриці TN;
- частота 70 Гц;
- доступні інтерфейси VGA;
- відношення сторін 16: 9.

Технічні характеристики робочої станції є достатніми для проведення тестування розробленого програмного додатку та отримання достовірних даних про коректність роботи, швидкість опрацювання зображень та можливість подальшого впровадження запропонованої програмної розробки.

Процес тестування проходив вдекілька етапів, на першому етапі проводилось тестування програмного додатку постворенню простих блок схем алгоритмів, що включають невелику кількість структурних елементів. Результат рестування наведено на рисуноку 3.3.

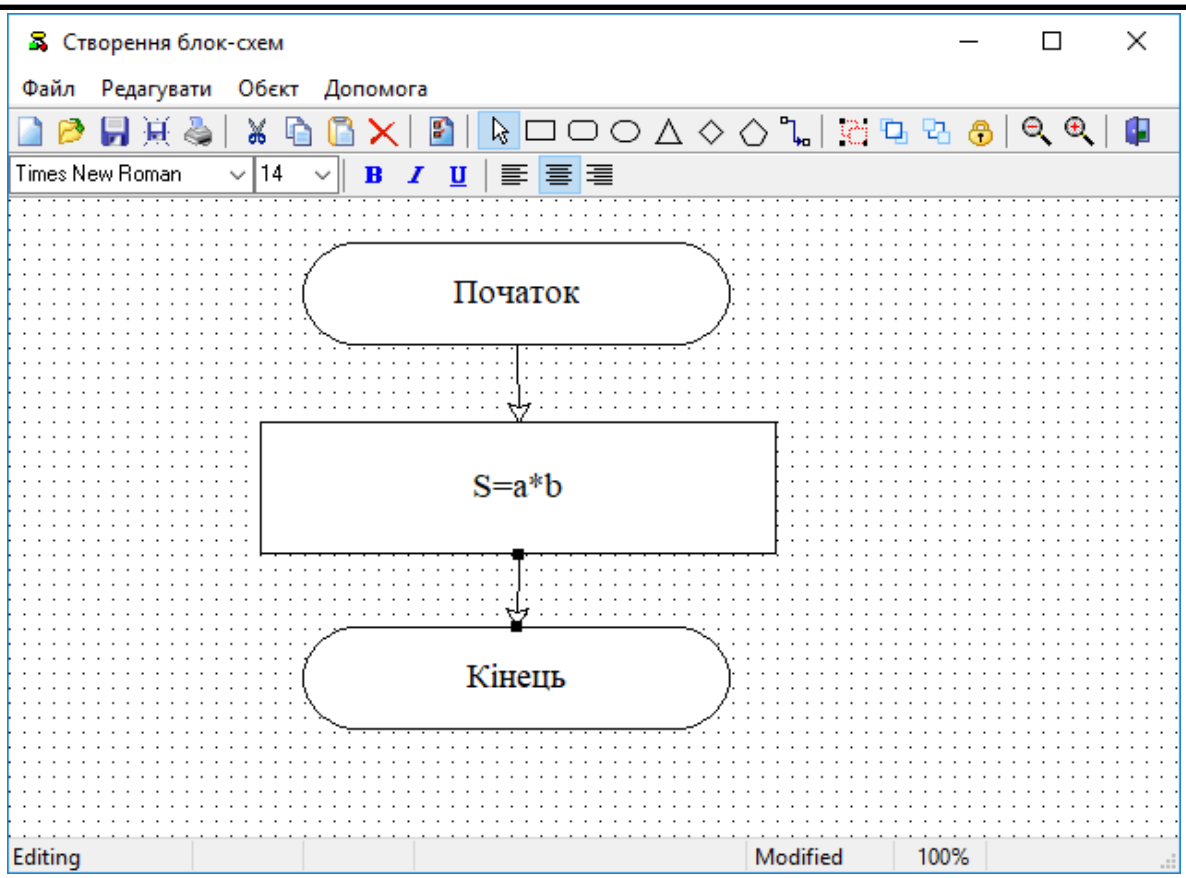


Рисунок 3.3 – Приклад створення простої блок-схеми

Як показали результати першогоетапу тестування розроблений програмний добаток успішно зумів побудувати просту блок-схему, а також виконати ряд другорядних завдань, а саме:

- успішно здійснив редагування введеної інформації в описі фігур (проведена зміна типу шрифта, розміру та кольору, проведно центрування внутрішнього надпису в структурному блоці);
- успішно проведено редагування елементів блок схеми, проведено зміну розмірів та розташування окремих елементів;
- зв’язуючі лінії зберегли прив’язку до виділених елементів при їх переміщенні та надладанні один на одного;
- успішно було видалено графічний елемент з поля редагування блок-схеми алгоритму.

На другому етапі тестування було проведено перевірку на максимальну кількість геометричних елементів, які можуть одночасно знаходиться на

робочому полотні програмного додатку. Результати тестування наведено на рисунку 3.4.

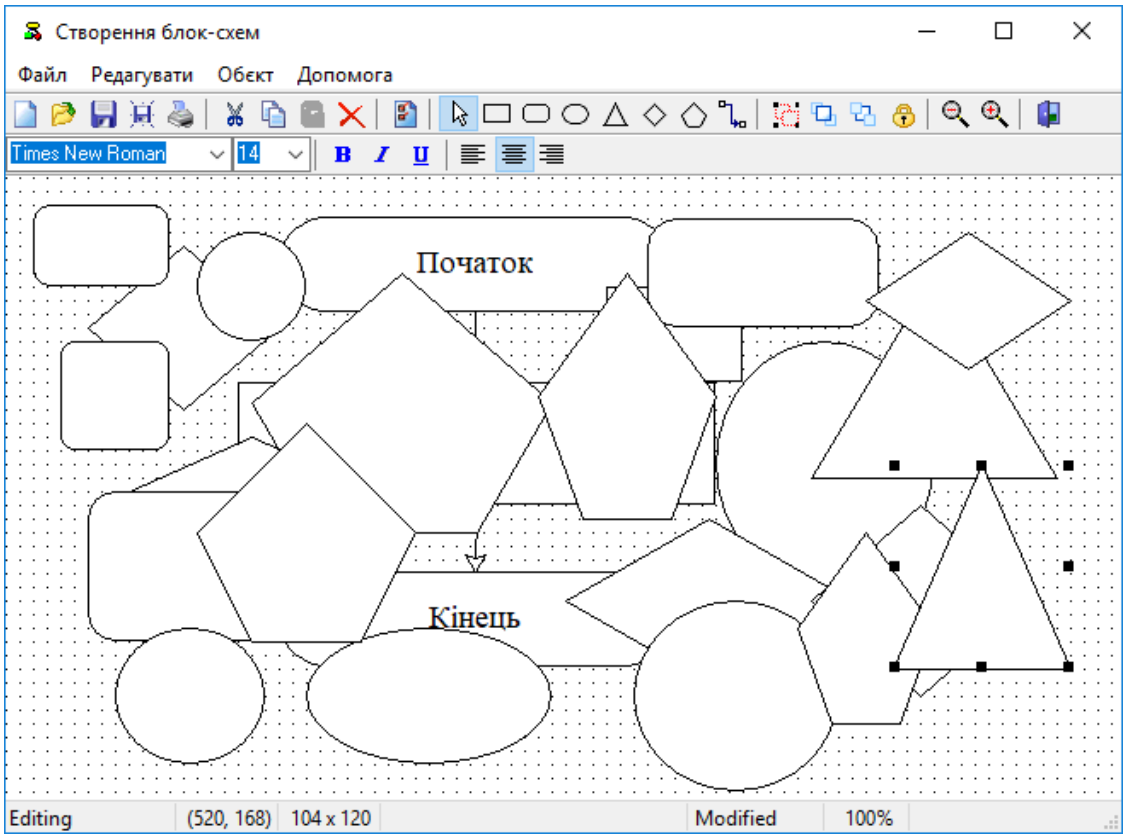


Рисунок 3.4 – Результати другого етапу тестування програмного додатку створення та редагування блок-схем алгоритмів

На основі аналізу отриманих результатів можна зробити наступні висновки:

- програма дозволяє одночасно працювати з великою кількістю графічних елементів (під час тестування було використано більше 100 елементів), при цьому помітні сповільнення роботи самої програми відсутні;

- при використанні великої кількості графічних елементів, що накладаються один на одного, не призводить до видимих сповільнень, швидкість виділення потрібного елемента відбувається без відчутних труднощів. Курсор позиціонується з необхідною точністю, а сусідні графічні елементи розрізняються без додаткових зусиль;

- Розмір вихідного файлу зростає пропорційно кількості задіяних графічних елементів, проте сумарний розмір є незначним.

Під час останнього етапу тестування було перевірено додаткові функції програмного додатку, а саме:

- можливість зберігання/завантаження блок-схем алгоримів;
- можливості масштабування робочого проекту;
- функції автоматичного вибору групи графічних елементів;
- редагування шарів розташування графічних елементів з можливістю переносу на передні/задній план;
- функції блокування графічного елементу, для уникнення випадкової зміни його параметрів;
- перевірка коректногофункціонування контекстного та випадаючого меню.

.

4 ТЕХНІКО-ЕКОНОМІЧНИЙ РОЗДІЛ

Метою техніко – економічного розділу дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності програмного візуалзації блок-схем алггоритмів довільної складності та прийняття рішення про його подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки. Для проведення даного дослідження необхідно провести ряд розразунків.

4.1 Розрахунок витрат на розробку програмного додатку

Витрати на розробку і впровадження програмного модуля візуалізації блок-схем алгоритмів (K) включають:

$$K = K_1 + K_2,$$

де K_1 - витрати на розробку апаратного та програмного забезпечення грн.;

K_2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп'ютері, грн.

Витрати на розробку апаратних та програмних засобів включають:

- витрати на оплату праці розробників ($B_{ОП}$);
- витрати на відрахування у спеціальні державні фонди ($B_ф$);
- витрати на матеріали та комплектуючі ($П_в$);
- накладні витрати (H);
- інші витрати ($I_в$)
- витрати на використання комп'ютерної техніки($B_{КТ}$) .

Розрахунок витрат на оплату праці.

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудоємності відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти - розробники, а саме: керівник проекту; студент-дипломант; консультант техніко-економічного розділу (таблиця 4.1).

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

№п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник ДП, викладач	5286
2	Консультант техніко-економічного розділу, доцент	6026
3	Студент	1100

Витрати на оплату праці розробників проекту визначаються за наступною формулою (4.1):

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \ , \qquad (4.1)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн.,

Середньо годинна ставка працівника може бути розрахована за такою формулою (4.2):

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_i}, \quad (4.2)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ - місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год. (приймаємо 168 год.).

Коефіцієнт *h*, який визначає розмір додаткової заробітної плати, для керівника та консультанта техніко-економічного розділу дорівнює 0,47.

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 - Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, год	Погодинна заробітна плата, грн/год.	Витрати на розробку, грн
1	Керівник ДП, старший викладач	16	46,25	740
2	Консультант техніко-економічного розділу, доцент	2	88,6	177,2
3	Студент	144	6,55	943,2
Разом				1860,4

Відрахування на соціальні заходи. Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5% від суми заробітної плати:

$$B_Ф = \frac{20,5}{100} \cdot 1860,4 = 381,38 \text{ грн.}$$

Розрахунок витрат на матеріали та комплектуючі.

Загальна сума витрат на матеріальні ресурси (B_M) визначається за формулою (4.3):

$$B_M = \sum_{i=1}^{n} K_i \cdot Ц_i, \qquad (4.3)$$

де K_i - витрата i-го типу матеріалу, натуральні одиниці вимірювання;

$Ц_i$ - ціна за одиницю i-го типу матеріалу, грн.;

i - тип матеріального ресурсу;

n - кількість типів матеріальних ресурсів.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна за одиницю, грн.	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
	Допоміжна література	шт	1	600	600	60	660
	Папір (формат А4)	уп	2	80	160	16	176
	Ручка кулькова	шт	2	10	20	2	22
	Олівець простий	шт	2	10	20	2	22
	Диски CD-R	шт	2	15	30	3	33
	Зошит, 96 арк	шт	1	50	50	5	55
	Тонер для принтера	уп	1	90	90	9	99
	Канцелярскі маркери (червоний, зелений)	шт	2	20	40	4	44
	Р а з о м						1111,00

Витрати на використання комп’ютерної техніки.

Витрати на використання комп’ютерної техніки($В_{КТ}$) включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. За даними обчислювального центру ТНЕУ для комп’ютера типу IBM PC/ATX вартість години роботи становить 6 грн. Середній щоденний час роботи на комп’ютері – 2 години. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4- Розрахунок витрат на використання комп’ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп’ютер	Час використання комп’ютера, год.	Витрати на використання комп’ютера грн.
1	Проведення досліджень та оформлення їх результатів	60	360
2	Оформлення техніко-економічного розділу	8	48
3	Оформлення ДП	12	72
Разом		80	480

Накладні витрати.

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$H = 1{,}5 \cdot 1860{,}4 = 2790{,}6 \text{ (грн)}.$$

Інші витрати.

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$I_B = 1860{,}4 \cdot 0{,}1 = 186{,}04 \text{ (грн).}$$

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_M + H + I_B + B_{КТ},$$

$$K_1 = 1860{,}4+381{,}38+1111{,}00+2790{,}6+186{,}04+480{,}00=6809{,}42 \text{ (грн)} .$$

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою (4.4):

$$K_2 = S_{м.г.} \cdot t_{від} \qquad (4.4)$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн./год;

$t_{від}$ - комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 30 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 6 грн., тому $K_2 = 6 \cdot 60 = 360$ грн.

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблювальної системи моніторингу слід порівняти її з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_{\text{П}} = E_{1\text{П}} + E_{2\text{П}},$$

де $E_{П}$ - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1П}$ - вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

$E_{2П}$ - вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати $B_{ЕП}$ визначаються за формулою:

$$B_{E\text{П}} = E_{\text{П}} * N_{\text{П}},$$

де $N_{П}$ - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1\text{П}} = \sum_{l=1}^{n} n_i t_i c_i ,$$

де i - категорії працівників, які приймають участь у підготовці даних (i=1,2,...n);

n_i - кількість працівників i-ої категорії, осіб.;

t_i - трудомісткість роботи співробітників i-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника i-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0 (1+b)}{m},$$

де c_i^0 - основна місячна заробітна плата працівника *i*-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57);

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: *c* = 3723 грн. Тоді:

$$c_1 = \frac{3723(1+0{,}57)}{22*8} = 33{,}21 \text{грн/год}$$

Трудомісткість підготовки даних для проектного рішення складає 1 год., для аналога 1,5 год.

Таблиця 4.5- Розрахунок витрат на підготовку даних та реалізацію проектного рішення на комп'ютері

№	Час роботи співробітників, год.	Середньогодинна заробітна плата, грн./год.	Витрати , грн.
	Проектне рішення		
1	1	33,21	33,21
	Аналог		
1	1,5	33,21	66,42

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2\text{П}} = t * S_{МГ}$$

де *t* - витрати машинного часу для реалізації рішення (аналогу), год.;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн./год.

$E_{2П}$ = 1* 6 = 6 грн.; $E_{2А}$=1,5*6 = 9 грн.

$E_{П}$=33,21+6=39,21 грн. ; $E_{А}$=66,42+9 = 75,42 грн.

$B_{ЕП}$= 39,21*252=9880,92 грн. ; $B_{ЕА}$=75,42 *252=19005,84 грн.

Обчислення накладних витрат.

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 60–100 % від суми основної та додаткової заробітної плати працівників.

$$H_B = 0{,}7 * B_{ОП}, \tag{4.7}$$

де H_B – накладні витрати.

$$H_B = 0{,}7 * 5845{,}11 = 4091{,}58 \text{ грн.}$$

Складання кошторису витрат та визначення собівартості.

Результати проведених розрахунків зведемо у таблицю 4.6.

Таблиця 4.6 - Кошторис витрат

№ п/п	Найменування витрат	Сума витрат, грн.
1	Витрати на оплату праці	1860,4
2	Відрахування у спеціальні державні фонди	381,38
3	Витрати на матеріали та комплектуючі	1111,00
4	Накладні витрати на розробку	2790,6
5	Інші витрати	186,04
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту	360
7	Накладні витрати експлуатацію	4091,58
8	Річні експлуатаційні витрати	19005,84
Разом		29786,09

Розрахунок ціни проекту.

Договірна ціна ($Ц_Д$) для проектних рішень розраховується за формулою (4.8):

$$Ц_Д = B_{КС} \cdot \left(1 + \frac{p}{100}\right), \quad (4.8)$$

де $B_{КС}$ – кошторисна вартість, грн.;

p - середній рівень рентабельності, % (приймаємо 26% за погодженням з керівником).

$$Ц_Д = 29786,09 \cdot (1+0,26) = 37530,47 \text{ грн.}$$

4.3 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічна ефективність ($E_ф$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_ф = \frac{П}{B_{КС}}, \quad (4.9)$$

де $П$ – прибуток, грн.;

$B_{КС}$ – кошторисна вартість, грн..

$$E_ф = 7812,27 \text{ грн.} / 29786,09 \text{ грн.} = 0,25.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Тр$):

$$T_P = \frac{1}{E_P} \ . \qquad (4.10)$$

Тобто: T_P =1/0,25 = 4р.

Прийнятним вважається термін окупності близький до 7 років.

Розраховані економічні показники проекту занесемо до таблиці 4.7.

Таблиця 4.7 - Економічні показники розробки

№ п/п	Показник	Значення
1.	Собівартість, грн.	29786,09
2.	Плановий прибуток, грн.	7744,38в
3.	Ціна, грн.	37530,47
4.	Економічна ефективність	0,25
5.	Термін окупності, рік	4

Враховуючи основі економічні показники з таблиці 4.7, можна зробити висновок, що при економічній ефективності 0,25 та терміні окупності – 4 роки проводити роботи по впровадженню даного програмного модуля є доцільним та економічно вигідним.

ВИСНОВКИ

На основі аналізу сучасних цифрових бібліотек для розробки програмних засобів по створенню та обробці цифрових зображень можна зробити наступні висновки:

1. Проведено аналіз та класифікацію існуючих типів алгоритмів, на основі дослідження стандартів проектуваня та креслення блок-схем алгоритмів, що дозволило визначити множину необхідних графічних елементів для побулови блок-схем алгоритмів.

2. Проведено аналіз елементів теорії графів для побудови дерев, що дозволило розробити алгоритми візуалізації блок-схем алгоритмів у вигляді зв’язних графів.

3. Проведено аналіз сучасних програмних засобів побудови та візуалізації блок-схем, що дозволило визначити основні вимоги які ставляться перед розробниками.

4. Досліджено цифрову бібліотеку для роботи з графами TSimpleGraph, що дозволило виділити множину функцій які необхідні для реалізації програмного додатку згідно поставленого технічного завдання.

5. Спроектовано структуру програмного засобу побудови та візуалізації блок-схем, розроблено множину класів на основі яких реалізовано програмний додаток;

6. Реалізовано програмний засобів побудови та візуалізації блок-схем алгоритмів та проведено тестування розробленого програмного засобу на основі побудови блок-схем алгоритмів різної складності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Баженова И.Ю. Язык программирования Delphi// И.Ю. Баженова// АО "Диалог-МИФИ", 1997. – 366с.

2. Бартлетт Н. Программирование на Delphi Путеводитель // Н. Бартлетт А. Лесли, С. Симкин / Издательство НИПФ "ДиаСофт Лтд.",1996. – 116с.

3. Вебер Дж. Технология С++в подлиннике // Дж. Вебер //QUE Corporation, 1996. – 256с.

4. Волш А. И. Основы программирования на С++для World Wide Web // А. И. Волш // Издательство "Диалектика",1996. – 458с.

5. Марков А. С. «Базы данных. Введение в теорию и методологию // А. С. Марков, К.Ю. Лисовский // Финансы и статистика»-2006,-Р. 24-35.

6. Абрамов С. А. Задачи по программированию // С. А. Абрамов, Г. Г. Гнездилова, Е. Н. Капустина, М. И. Селюн// М.: Наука, 1988. – 256с.

7. Березин Б.И., Начальный курс Delphi // Б.И.Березин, С.Б.Березин // М.: ДИАЛОГ-МИФИ, 1996. – 331с.

8. Бондарев В.М. Основы программирования // В.М. Бондарев, В.И. Рублинецкий, Е.Г. Качко // Харьков: Фолио, Ростов н/Д: Феникс, 1997. – 446с.

9. Вирт Н. Алгоритмы и структуры данных. // Н. Вирт // М.: Мир, 1989. - 345с.

10. Гладков В. П. Задачи по информатике на вступительном экзамене в вуз и их решения: Учебное пособие // В. П. Гладков // Пермь: Перм. техн. ун-т, 1994. – 516с.

11. Грогоно П. Программирование на языке Delphi // П. Грогоно // М.: Мир, 1982. – 216с.

12. Дагене В.А. 100 задач по программированию // В.А.Дагене, Г.К. Григас, К.Ф. Аугутис // М.: Просвещение, 1993. – 106с.

13. Джамса К. Библиотека программиста Java // К.Джамса // ООО "Попурри", 1996. – 656с.

14. Марков А. С. «Базы данных. Введение в теорию и методологію // А. С. Марков, К.Ю. Лисовский // Финансы и статистика»-2006, - Р. 24-35.

15. Заварыкин В.М. Основы информатики и вычислительной техники // В.М. Заварыкин, В.Г. Житомирский, М.П. Лапчик // М.: Просвещение, 1989. – 556с.

16. Касаткин В. Н. Информация. Алгоритмы. ЭВМ // В. Н. Касаткин // М.: Просвещение, 1991. – 219с.

17. Кен А. Язык программирования Delphi // А. Кен, Дж. Гослинг // Издательство "Питер-Пресс", 1997. – 378с.

18. Керниган Б. Язык программирования Delphi // Б. Керниган, Д. Ритчи // Пер. с англ. — М.: Финансы и статистика, 1992. – 391с.

19. Ляхович В.Ф. Руководство к решению задач по основам информатики и вычислительной техники // В.Ф. Ляхович// М.: Высшая школа, 1994. – 127с.

20. Мейнджер Дж. Delphi Основы программирования // Дж. Мейнджер // Издательская группа BHV, Киев,1997. – 346с.

21. Миков А. И. Информатика. Введение в компьютерные науки // А. И. Миков // Пермь: Изд-во ПГУ, 1998. – 442с.

22. Могилев А. В. Информатика: Учеб. пособие для студ. пед. вузов // А. В. Могилев, Е. К. Хеннера.// М.: Изд. центр «Академия», 1999. – 629с.

23. Нотон П. JAVA:Справ.руководство// П.Нотон, А.Тихонова.-М.:БИНОМ:Восточ.Кн.Компания,1996:Восточ.Кн.Компания. - 447с.

24. Нотон П. Полный справочник по Java // П. Нотон, Г. Шилдт// McGraw-Hill,1997, Издательство "Диалектика",1997. – 556с.

25. Ренеган Э.Дж. 1001 адрес WEB для программистов :Новейший путеводитель программиста по ресурсам World Wide Web:Пер.// Э.Дж. Ренеган /с англ..-Минск:Попурри,1997.-512с.ил.

26. Родли Дж. Создание Java-апплетов // Дж. Родли // The Coriolis Group,Inc.,1996, Издательство НИПФ "ДиаСофт Лтд.",1996. – 466с.

27. Томас М. Секреты программирования для Internet на Java // М.Томас, П. Пратик, А. Хадсон, Д. Болл// Ventana Press, Ventana Communications Group, U.S.A., 1997. – 396с.

28. Семакина И. Г. Информатика. Задачник-практикум: В 2 т. // И. Г. Семакина, Е.К.Хеннера // М.: Лаборатория Базовых Знаний, 1999. – 476с.

29. Сокольский М.В. Все об Intranet и Internet // М.В. Сокольский // М.:Элиот,1998. - 254с.ил.

30. Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ // Д. Тассел // М.: Мир, 1981. – 56с.

31. Тюрин Ю.Н. Анализ данных на компьютере. // Ю.Н. Тюрин, А.А. Макаров, В.Э.Фигурнова // М.: ИНФРА-М, Финансы и статистика, 1995. - 384с.

32. Флэнэген Д. Java in a Nutshell // Д. Флэнэген // O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 1998. – 473с.

33. Чен М.С. Программирование на С++:1001 совет:Наиболее полное руководство по Java и Visual J++ :Пер.с англ.// М.С.Чен, С.В. Грифис, Э.Ф. Изи./ -Минск:Попурри,1997.- 640с.ил.+ Прил.(1диск.).

34. Эферган М. С++: справочник // М. Эферган // QUE Corporation, 1997, Издательство "Питер Ком", 1998. – 256с.

35. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» фахового спрямування «Комп’ютерні системи та мережі» / О.М. Березький, Л.О.Дубчак, Р.Б. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв // Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2014.–65 с.

36. Методичні вказівки до написання техніко-економічного розділу дипломних проектів освітньо-кваліфікаційного рівня «ба калавр» напряму підготовки 6.050102 комп’ютерна інженерія/ І.Р. Паздрій // Тернопіль: ТНЕУ, 2014. – 37 с.

ДОДАТОК А

Вихідний програмний код побудови блок-схем

```
unit Main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  SimpleGraph {$IFDEF DELPHI7_UP}, XPMan {$ENDIF}, Dialogs, ExtDlgs,
  Menus, ActnList, ImgList, StdCtrls, ComCtrls, ToolWin, JPEG;
type
  TMainForm = class(TForm)
    SimpleGraph: TSimpleGraph;
    ToolBar: TToolBar;
    StatusBar: TStatusBar;
    ImageList: TImageList;
    ActionList: TActionList;
    FileNew: TAction;
    FileOpen: TAction;
    FileSave: TAction;
    MainMenu: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    FileExit: TAction;
    N1: TMenuItem;
    Exit1: TMenuItem;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    EditCut: TAction;
    EditCopy: TAction;
    EditPaste: TAction;
    EditDelete: TAction;
    EditSelectAll: TAction;
    EditLockNodes: TAction;
    Edit1: TMenuItem;
    EditCut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    Delete1: TMenuItem;
    N2: TMenuItem;
    SelectAll1: TMenuItem;
    N3: TMenuItem;
    LockNodes1: TMenuItem;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
```

```
ToolButton7: TToolButton;
ToolButton8: TToolButton;
ToolButton9: TToolButton;
ToolButton10: TToolButton;
ToolButton11: TToolButton;
ObjectsNone: TAction;
ObjectsRectangle: TAction;
ObjectsRoundRect: TAction;
ObjectsEllipse: TAction;
ObjectsLink: TAction;
Opjects1: TMenuItem;
ToolButton12: TToolButton;
ToolButton13: TToolButton;
ToolButton14: TToolButton;
ToolButton15: TToolButton;
ToolButton16: TToolButton;
EditBringToFront: TAction;
EditSendToBack: TAction;
N6: TMenuItem;
BringToFront1: TMenuItem;
SendToBack1: TMenuItem;
EditProperties: TAction;
N7: TMenuItem;
Properties1: TMenuItem;
DesignerPopup: TPopupMenu;
ObjectsPopup: TPopupMenu;
Properties2: TMenuItem;
Cut1: TMenuItem;
Copy2: TMenuItem;
Paste2: TMenuItem;
Delete2: TMenuItem;
N8: TMenuItem;
Properties3: TMenuItem;
N10: TMenuItem;
N12: TMenuItem;
SelectAllNodes1: TMenuItem;
ToolButton18: TToolButton;
Paste5: TMenuItem;
EditMode1: TMenuItem;
N4: TMenuItem;
InsertRectangle1: TMenuItem;
InsertRoundRectangle1: TMenuItem;
InsertEllipse1: TMenuItem;
N5: TMenuItem;
LinkObjects1: TMenuItem;
N9: TMenuItem;
InsertRectangle2: TMenuItem;
InsertRoundRectangle2: TMenuItem;
InsertEllipse2: TMenuItem;
N14: TMenuItem;
LinkObjects2: TMenuItem;
N15: TMenuItem;
BringToFront2: TMenuItem;
SendToBack2: TMenuItem;
```

```
FilePrint: TAction;
N16: TMenuItem;
Print1: TMenuItem;
PrinterSetupDialog: TPrinterSetupDialog;
ToolButton19: TToolButton;
ToolButton20: TToolButton;
ToolButton21: TToolButton;
ToolButton22: TToolButton;
ToolButton23: TToolButton;
ToolButton24: TToolButton;
btnSaveAs: TToolButton;
FileSaveAs: TAction;
FileSaveAs1: TMenuItem;
HelpAbout: TAction;
Help1: TMenuItem;
About2: TMenuItem;
FormatToolBar: TToolBar;
cbxFontName: TComboBox;
cbxFontSize: TComboBox;
btnBoldface: TToolButton;
btnItalic: TToolButton;
btnUnderline: TToolButton;
FormatBold: TAction;
FormatItalic: TAction;
FormatUnderline: TAction;
FormatAlignLeft: TAction;
FormatCenter: TAction;
FormatAlignRight: TAction;
ToolButton27: TToolButton;
ToolButton28: TToolButton;
ToolButton29: TToolButton;
FileExport: TAction;
SavePictureDialog: TSavePictureDialog;
ToolButton30: TToolButton;
Export1: TMenuItem;
ToolButton17: TToolButton;
ToolButton25: TToolButton;
ViewZoomIn: TAction;
ViewZoomOut: TAction;
ToolButton26: TToolButton;
ToolButton31: TToolButton;
ToolButton32: TToolButton;
ObjectsTriangle: TAction;
ToolButton33: TToolButton;
ObjectsRhomboid: TAction;
ToolButton34: TToolButton;
ObjectsPentagon: TAction;
ToolButton35: TToolButton;
procedure FileNewExecute(Sender: TObject);
procedure FileOpenExecute(Sender: TObject);
procedure FileSaveExecute(Sender: TObject);
procedure FileSaveAsExecute(Sender: TObject);
procedure FileExportExecute(Sender: TObject);
procedure FilePrintExecute(Sender: TObject);
```

```
    procedure FileExitExecute(Sender: TObject);
    procedure EditCutExecute(Sender: TObject);
    procedure EditCopyExecute(Sender: TObject);
    procedure EditPasteExecute(Sender: TObject);
    procedure EditDeleteExecute(Sender: TObject);
    procedure EditSelectAllExecute(Sender: TObject);
    procedure EditSendToBackExecute(Sender: TObject);
    procedure EditBringToFrontExecute(Sender: TObject);
    procedure EditLockNodesExecute(Sender: TObject);
    procedure EditPropertiesExecute(Sender: TObject);
    procedure FormatBoldExecute(Sender: TObject);
    procedure FormatItalicExecute(Sender: TObject);
    procedure FormatUnderlineExecute(Sender: TObject);
    procedure FormatAlignLeftExecute(Sender: TObject);
    procedure FormatCenterExecute(Sender: TObject);
    procedure FormatAlignRightExecute(Sender: TObject);
    procedure HelpAboutExecute(Sender: TObject);
    procedure ObjectsNoneExecute(Sender: TObject);
    procedure ObjectsRectangleExecute(Sender: TObject);
    procedure ObjectsRoundRectExecute(Sender: TObject);
    procedure ObjectsEllipseExecute(Sender: TObject);
    procedure ObjectsTriangleExecute(Sender: TObject);
    procedure ObjectsLinkExecute(Sender: TObject);
    procedure ObjectsRhomboidExecute(Sender: TObject);
    procedure ObjectsPentagonExecute(Sender: TObject);
    procedure ViewZoomInExecute(Sender: TObject);
    procedure ViewZoomOutExecute(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject;var CanClose: Boolean);
    procedure ActionListUpdate(Action: TBasicAction;var Handled: Boolean);
    procedure cbxFontSizeChange(Sender: TObject);
    procedure cbxFontNameChange(Sender: TObject);
    procedure SimpleGraphDblClick(Sender: TObject);
    procedure SimpleGraphNodeDblClick(Graph: TSimpleGraph;
      Node: TGraphNode);
    procedure SimpleGraphLinkDblClick(Graph: TSimpleGraph;
      Link: TGraphLink);
    procedure FormCreate(Sender: TObject);
    procedure SimpleGraphCommandModeChange(Sender: TObject);
    procedure SimpleGraphCanMoveResizeNode(Graph: TSimpleGraph;
      Node: TGraphNode;var NewLeft, NewTop, NewWidth, NewHeight: Integer;
      var CanMove, CanResize: Boolean);
    procedure SimpleGraphObjectSelect(Graph: TSimpleGraph;
      GraphObject: TGraphObject);
    procedure SimpleGraphObjectDblClick(Graph: TSimpleGraph;
      GraphObject: TGraphObject);
    procedure SimpleGraphObjectInsert(Graph: TSimpleGraph;
      GraphObject: TGraphObject);
  private
    function IsGraphSaved: Boolean;
    procedure ShowHint(Sender: TObject);
  end;

var
  MainForm: TMainForm;
```

```
implementation

{$R *.dfm}
uses
  Clipbrd, Printers, DesignProp, NodeProp, LinkProp, ObjectProp,
  AboutDelphiArea;
resourcestring
  SSaveChanges   = 'Зберегти граф?';
  SViewOnly      = 'Тільки перегляд';
  SEditing       = 'Редагування';
  SLinkingNodes  = 'З'єднати вузли';
  SInsertingNode = 'Додати вузол';
  SModified      = 'редагувати';
  SNotModified   = '';
  SUntitled      = 'Без підпису';
function TMainForm.IsGraphSaved: Boolean;
begin
  Result := True;
  if SimpleGraph.Modified then
    case MessageDlg(SSaveChanges, mtConfirmation, [mbYes, mbNo, mbCancel], 0) of
      mrYes:
        begin
          FileSave.Execute;
          Result := not SimpleGraph.Modified;
        end;
      mrCancel:
        Result := False;
    end;
end;
procedure TMainForm.ShowHint(Sender: TObject);
begin
  StatusBar.Panels[6].Text := Application.Hint;
end;
procedure TMainForm.FileNewExecute(Sender: TObject);
begin
  if IsGraphSaved then
  begin
    SimpleGraph.Clear;
    SaveDialog.FileName := SUntitled;
    Caption := SaveDialog.FileName + ' - ' + Application.Title;
  end;
end;
procedure TMainForm.FileOpenExecute(Sender: TObject);
begin
  if IsGraphSaved and OpenDialog.Execute then
  begin
    SimpleGraph.LoadFromFile(OpenDialog.FileName);
    SaveDialog.FileName := OpenDialog.FileName;
    Caption := SaveDialog.FileName + ' - ' + Application.Title;
  end;
end;

procedure TMainForm.FileSaveExecute(Sender: TObject);
```

```
begin
  if SaveDialog.FileName <> SUntitled then
  begin
    SimpleGraph.SaveToFile(SaveDialog.FileName);
    Caption := SaveDialog.FileName + ' - ' + Application.Title;
  end
  else
  begin
    if SaveDialog.Execute then
      SimpleGraph.SaveToFile(SaveDialog.FileName);
  end;
  Caption := SaveDialog.FileName + ' - ' + Application.Title;
end;

procedure TMainForm.FileSaveAsExecute(Sender: TObject);
begin
  if SaveDialog.Execute then
  begin
    SimpleGraph.SaveToFile(SaveDialog.FileName);
    Caption := SaveDialog.FileName + ' - ' + Application.Title;
  end;
end;

procedure TMainForm.FileExportExecute(Sender: TObject);
begin
  SavePictureDialog.FileName := ChangeFileExt(SaveDialog.FileName, '.' +
SavePictureDialog.DefaultExt);
  if SavePictureDialog.Execute then
    SimpleGraph.SaveAsMetafile(SavePictureDialog.FileName);
end;

procedure TMainForm.FilePrintExecute(Sender: TObject);
var
  Rect: TRect;
begin
  if PrinterSetupDialog.Execute then
  begin
    SetRect(Rect, 0, 0, Printer.PageWidth, Printer.PageHeight);
    InflateRect(Rect, -50, -50);
    Printer.BeginDoc;
    SimpleGraph.Print(Printer.Canvas, Rect);
    Printer.EndDoc;
  end;
end;

procedure TMainForm.FileExitExecute(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.EditCutExecute(Sender: TObject);
begin
  EditCopy.Execute;
  EditDelete.Execute;
```

```
end;

procedure TMainForm.EditCopyExecute(Sender: TObject);
begin
  SimpleGraph.CopyToClipboard(True);
end;

procedure TMainForm.EditPasteExecute(Sender: TObject);
begin
  SimpleGraph.PasteFromClipboard;
end;

procedure TMainForm.EditDeleteExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        SelectedObjects[I].Free;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.EditSelectAllExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := 0 to Objects.Count - 1 do
        Objects[I].Selected := True;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.EditSendToBackExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        SelectedObjects[I].SendToBack;
```

```
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.EditBringToFrontExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        SelectedObjects[I].BringToFront;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.EditLockNodesExecute(Sender: TObject);
begin
  SimpleGraph.LockNodes := not SimpleGraph.LockNodes;
end;

procedure TMainForm.EditPropertiesExecute(Sender: TObject);
var
  LinkCount: Integer;
begin
  if SimpleGraph.SelectedObjects.Count = 0 then
    TDesignerProperties.Execute(SimpleGraph)
  else
  begin
    LinkCount := SimpleGraph.SelectedObjectsCount(TGraphLink);
    if LinkCount = 0 then
      TNodeProperties.Execute(SimpleGraph.SelectedObjects)
    else if LinkCount = SimpleGraph.SelectedObjects.Count then
      TLinkProperties.Execute(SimpleGraph.SelectedObjects)
    else
      TObjectProperties.Execute(SimpleGraph.SelectedObjects);
  end;
end;

procedure TMainForm.FormatBoldExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
```

```
        with SelectedObjects[I].Font do
          if FormatBold.Checked then
            Style := Style + [fsBold]
          else
            Style := Style - [fsBold];
      finally
        EndUpdate;
      end;
    end;
  end;

procedure TMainForm.FormatItalicExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        with SelectedObjects[I].Font do
          if FormatItalic.Checked then
            Style := Style + [fsItalic]
          else
            Style := Style - [fsItalic];
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.FormatUnderlineExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        with SelectedObjects[I].Font do
          if FormatUnderline.Checked then
            Style := Style + [fsUnderLine]
          else
            Style := Style - [fsUnderline];
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.FormatAlignLeftExecute(Sender: TObject);
var
  I: Integer;
```

```
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        if SelectedObjects[I] is TGraphNode then
          TGraphNode(SelectedObjects[I]).Alignment := taLeftJustify;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.FormatCenterExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        if SelectedObjects[I] is TGraphNode then
          TGraphNode(SelectedObjects[I]).Alignment := taCenter;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.FormatAlignRightExecute(Sender: TObject);
var
  I: Integer;
begin
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        if SelectedObjects[I] is TGraphNode then
          TGraphNode(SelectedObjects[I]).Alignment := taRightJustify;
    finally
      EndUpdate;
    end;
  end;
end;

procedure TMainForm.HelpAboutExecute(Sender: TObject);
begin
  with TAbout.Create(Application) do
    try
      ShowModal;
    finally
```

```
      Free;
    end;
end;

procedure TMainForm.ObjectsNoneExecute(Sender: TObject);
begin
  SimpleGraph.CommandMode := cmEdit;
end;

procedure TMainForm.ObjectsRectangleExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TRectangularNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;

procedure TMainForm.ObjectsRoundRectExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TRoundRectangularNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;

procedure TMainForm.ObjectsEllipseExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TEllipticNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;

procedure TMainForm.ObjectsTriangleExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TTriangularNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;

procedure TMainForm.ObjectsRhomboidExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TRhomboidalNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;

procedure TMainForm.ObjectsPentagonExecute(Sender: TObject);
begin
  SimpleGraph.DefaultNodeClass := TPentagonalNode;
  SimpleGraph.CommandMode := cmInsertNode;
end;
procedure TMainForm.ObjectsLinkExecute(Sender: TObject);
begin
  SimpleGraph.CommandMode := cmLinkNodes;
end;
procedure TMainForm.ViewZoomInExecute(Sender: TObject);
begin
  SimpleGraph.Zoom := SimpleGraph.Zoom + SimpleGraph.ZoomStep;
end;
procedure TMainForm.ViewZoomOutExecute(Sender: TObject);
begin
```

```
  SimpleGraph.Zoom := SimpleGraph.Zoom - SimpleGraph.ZoomStep;
end;
procedure TMainForm.FormCloseQuery(Sender: TObject;var CanClose: Boolean);
begin
  CanClose := IsGraphSaved;
end;
procedure TMainForm.ActionListUpdate(Action: TBasicAction;
  var Handled: Boolean);
begin
  Handled := True;
  FileSave.Enabled := SimpleGraph.Modified;
  FileSaveAs.Enabled := (SimpleGraph.Objects.Count > 0);
  FileExport.Enabled := (SimpleGraph.Objects.Count > 0);
  FilePrint.Enabled :=(Printer.Printers.Count > 0) and
   (SimpleGraph.Objects.Count > 0);
  EditCut.Enabled :=(SimpleGraph.SelectedObjects.Count > 0);
  EditCopy.Enabled :=(SimpleGraph.SelectedObjects.Count > 0);
  EditPaste.Enabled := Clipboard.HasFormat(CF_SIMPLEGRAPH);
  EditDelete.Enabled :=(SimpleGraph.SelectedObjects.Count > 0);
  EditBringToFront.Enabled :=(SimpleGraph.SelectedObjects.Count > 0);
  EditSendToBack.Enabled :=(SimpleGraph.SelectedObjects.Count > 0);
  EditSelectAll.Enabled :=
   (SimpleGraph.Objects.Count > SimpleGraph.SelectedObjects.Count);
  EditLockNodes.Checked := SimpleGraph.LockNodes;
  ObjectsNone.Checked :=(SimpleGraph.CommandMode = cmEdit);
  ObjectsRectangle.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TRectangularNode);
  ObjectsRoundRect.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TRoundRectangularNode);
  ObjectsEllipse.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TEllipticNode);
  ObjectsTriangle.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TTriangularNode);
  ObjectsRhomboid.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TRhomboidalNode);
  ObjectsPentagon.Checked :=(SimpleGraph.CommandMode = cmInsertNode) and
   (SimpleGraph.DefaultNodeClass = TPentagonalNode);
  ObjectsLink.Enabled :=(SimpleGraph.ObjectsCount(TGraphNode) >= 2);
  ObjectsLink.Checked :=(SimpleGraph.CommandMode = cmLinkNodes);
  ViewZoomOut.Enabled := (SimpleGraph.Zoom > SimpleGraph.ZoomMin);
  ViewZoomIn.Enabled := (SimpleGraph.Zoom < SimpleGraph.ZoomMax);
  if SimpleGraph.Modified then
    StatusBar.Panels[4].Text := SModified
  else
    StatusBar.Panels[4].Text := SNotModified;
  StatusBar.Panels[5].Text := Format('%d%%', [SimpleGraph.Zoom]);
end;
procedure TMainForm.cbxFontSizeChange(Sender: TObject);
var
  I: Integer;
  FontSize: Integer;
begin
  try
    FontSize := StrToInt(cbxFontSize.Text);
```

```
  except
    Exit;
  end;
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        SelectedObjects[I].Font.Size := FontSize;
    finally
      EndUpdate;
    end;
  end;
end;
procedure TMainForm.cbxFontNameChange(Sender: TObject);
var
  I: Integer;
  FontName: String;
begin
  FontName := cbxFontName.Items[cbxFontName.ItemIndex];
  with SimpleGraph do
  begin
    BeginUpdate;
    try
      for I := SelectedObjects.Count - 1 downto 0 do
        SelectedObjects[I].Font.Name := FontName;
    finally
      EndUpdate;
    end;
  end;
end;
procedure TMainForm.SimpleGraphDblClick(Sender: TObject);
begin
  EditProperties.Execute;
end;
procedure TMainForm.SimpleGraphNodeDblClick(Graph: TSimpleGraph;
  Node: TGraphNode);
begin
  EditProperties.Execute;
end;
procedure TMainForm.SimpleGraphLinkDblClick(Graph: TSimpleGraph;
  Link: TGraphLink);
begin
  EditProperties.Execute;
end;
procedure TMainForm.FormCreate(Sender: TObject);
begin
  {$IFDEF DELPHI7_UP}
  TXPManifest.Create(Self);
  {$ENDIF}
  Application.OnHint := ShowHint;
  SimpleGraphCommandModeChange(nil);
  cbxFontName.Items := Screen.Fonts;
  if ParamCount > 0 then
```

```
  begin
    SimpleGraph.LoadFromFile(ParamStr(1));
    SaveDialog.FileName := ExpandFileName(ParamStr(1));
    Caption := SaveDialog.FileName + ' - ' + Application.Title;
  end;
end;
procedure TMainForm.SimpleGraphCommandModeChange(Sender: TObject);
begin
  case SimpleGraph.CommandMode of
    cmViewOnly:
      StatusBar.Panels[0].Text := SViewOnly;
    cmEdit:
      StatusBar.Panels[0].Text := SEditing;
    cmLinkNodes:
      StatusBar.Panels[0].Text := SLinkingNodes;
    cmInsertNode:
      StatusBar.Panels[0].Text := SInsertingNode;
  end;
end;
procedure TMainForm.SimpleGraphCanMoveResizeNode(Graph: TSimpleGraph;
  Node: TGraphNode;var NewLeft, NewTop, NewWidth, NewHeight: Integer;
  var CanMove, CanResize: Boolean);
begin
  if SimpleGraph.SelectedObjects.Count = 1 then
  begin
    StatusBar.Panels[1].Text := Format('(%d, %d)', [NewLeft, NewTop]);
    StatusBar.Panels[2].Text := Format('%d x %d', [NewWidth, NewHeight]);
  end;
end;
procedure TMainForm.SimpleGraphObjectSelect(Graph: TSimpleGraph;
  GraphObject: TGraphObject);
var
  Node: TGraphNode;
  PosFirstLine: integer;
begin
  if (SimpleGraph.SelectedObjects.Count > 0) then
  begin
    GraphObject := SimpleGraph.SelectedObjects[0];
    cbxFontName.Text := GraphObject.Font.Name;
    cbxFontSize.Text := IntToStr(GraphObject.Font.Size);
    FormatBold.Checked := (fsBold in GraphObject.Font.Style);
    FormatItalic.Checked := (fsItalic in GraphObject.Font.Style);
    FormatUnderline.Checked := (fsUnderline in GraphObject.Font.Style);
  end;
  if (SimpleGraph.SelectedObjects.Count = 1) and
     (SimpleGraph.SelectedObjects[0] is TGraphNode) then
  begin
    Node := TGraphNode(SimpleGraph.SelectedObjects[0]);
    case Node.Alignment of
      taCenter: FormatCenter.Checked := True;
      taLeftJustify: FormatAlignLeft.Checked := True;
      taRightJustify: FormatAlignRight.Checked := True;
    end;
    StatusBar.Panels[1].Text := Format('(%d, %d)', [Node.Left, Node.Top]);
```

```
    StatusBar.Panels[2].Text := Format('%d x %d', [Node.Width, Node.Height]);
    PosFirstLine := Pos(#$D#$A, Node.Text);
    if PosFirstLine <> 0 then
      StatusBar.Panels[3].Text := Copy(Node.Text, 1, PosFirstLine)
    else
      StatusBar.Panels[3].Text := Node.Text;
  end
  else
  begin
    StatusBar.Panels[1].Text := '';
    StatusBar.Panels[2].Text := '';
    StatusBar.Panels[3].Text := '';
  end;
end;
procedure TMainForm.SimpleGraphObjectDblClick(Graph: TSimpleGraph;
  GraphObject: TGraphObject);
begin
  EditProperties.Execute;
end;
procedure TMainForm.SimpleGraphObjectInsert(Graph: TSimpleGraph;
  GraphObject: TGraphObject);
var
  FontStyle: TFontStyles;
begin
  FontStyle := [];
  if FormatBold.Checked then
    Include(FontStyle, fsBold);
  if FormatItalic.Checked then
    Include(FontStyle, fsItalic);
  if FormatUnderline.Checked then
    Include(FontStyle, fsUnderline);
  with GraphObject.Font do
  begin
    Size := StrToIntDef(cbxFontSize.Text, Size);
    Name := cbxFontName.Text;
    Style := FontStyle;
  end;
  if GraphObject is TGraphNode then
  begin
    if FormatAlignLeft.Checked then
      TGraphNode(GraphObject).Alignment := taLeftJustify
    else if FormatAlignRight.Checked then
      TGraphNode(GraphObject).Alignment := taRightJustify
    else
      TGraphNode(GraphObject).Alignment := taCenter;
  end;
end;
end.
```

ДОДАТОК Б

Довідка про використання результатів дипломного проекту