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

Тимчук Олена Юріївна

Програмний додаток для створення та візуалізації блок-схем алгоритмів на основі бібліотеки TSimpleGraph / Software application for creating and visualizing of block diagrams of algorithms based on the TSimpleGraph library

спеціальність: 123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ-41
Тимчук Олена Юріївна
_____

Науковий керівник
к.т.н., Батько Ю. М.

Кваліфікаційну роботу
Допущено до захисту
«___» _____ 20 ___ р.

Завідувач кафедри
_____ О.М. Березький

ТЕРНОПІЛЬ - 2021

РЕЗЮМЕ

Кваліфікаційна робота містить 78 сторінок пояснюючої записки, 14 рисунків, 10 таблиць, 2 додатки. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою кваліфікаційної роботи є розробка програмного додатку для створення та візуалізації блок-схем алгоримтів.

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 кваліфікаційній роботі на основі аналізу існуючих алгоритмів обробки 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

Qualification work contains 78 pages of explanatory note, 14 figures, 10 tables, 2 appendices. 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 qualification 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

КС	–	Комп’ютерна система
БД		База даних
GDI	–	Graphics Device Interface
ЦЗ	–	Цифрове зображення
ПЗ	–	Програмне забезпечення

ВСТУП

Алгоритми - це серце інформатики, і предмет має безліч практичних застосувань, а також інтелектуальну глибину. Під алгоритмом розуміється спосіб перетворення представлення інформації. Слово «algorithm» - походить від імені аль-Хорезмі - автора відомого арабського підручника з математики (від його імені відбулися також слова «алгебра» і «логарифм»).

Інтуїтивно кажучи, алгоритм - деякий формальний припис, діючи згідно з яким, можна отримати рішення задачі. Алгоритми типовим чином вирішують не тільки приватні задачі, але і класи задач. Що підлягають вирішенню приватні завдання, які виділяються при необхідності з розглянутого класу, визначаються за допомогою параметрів. Параметри грають роль вихідних даних для алгоритму. Приклади алгоритмів широко відомі: вивчаються в школі правила складання і множення десяткових чисел або, скажімо, алгоритми сортування масивів. Для алгоритмічно вирішуваної задачі завжди є багато різних способів її рішення тобто різних алгоритмів. На жаль, для вирішення деяких завдань не існує алгоритмів.

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. Поява комп’ютерів 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 комп’ютерів, цифрові автомати. 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 ДЛЯ СТВОРЕННЯ БЛОК-СХЕМ АЛГОРИТМІВ

1.1 Основні поняття теорії алгоритмів

Сьогодні весь світ оцифрований. Є відчуття інтелекту у кожному традиційному пристрої є відчуття спілкування, що робить наше життя таким легким і таким швидким. Всі ці технологічні досягнення просуваються вперед за допомогою програмного забезпечення, яке являє собою набір програм, призначених для вирішення проблеми. І кожна програма побудована на логіці/рішенні, яке називається алгоритмом. Алгоритм імені названий на честь розумної людини з Багдада Аль-Хорізмі. Він був першою людиною, яка впровадила у світ алгоритми, які були механічними, точними та однозначними.

Стандартним визначенням алгоритму - це чітко визначене поетапне рішення або серія вказівок для вирішення проблеми. Алгоритм може бути методом пошуку найменшого загального кратного двох чисел або рецептом приготування кулінарної страви.

Комп’ютер в основному робить багато математичних обчислень, а це означає, що у нього є багато проблем, які потрібно вирішити. Саме тому алгоритми становлять серце інформатики.

Комп’ютерний алгоритм - це обчислювальна процедура, яка бере набір кінцевих вхідних даних і перетворює їх у вихідні, застосовуючи деякі математичні та логічні методи.

Алгоритм програмування матиме кілька етапів наступним чином:

– «Визначення проблеми» - що необхідно робити.

– «Збір даних» - що потрібно для вирішення проблеми або вхідні дані.

– «Обробка даних» - розуміння того, що ми маємо, або перетворення їх у придатну для використання форму.

– «Логічний підхід» - використання зібраних та створених даних проти логіки для вирішення.

– «Рішення» - представте рішення таким чином, як хочете, у графічному інтерфейсі користувача або терміналі, на схемі чи діаграмі.

Якщо сказати коротко, з огляду на кінцеве вхідне значення для *x*, алгоритм перетворює його в ефективне вихідне значення *y*, де *y* дорівнює *f*(*x*) для деякої чітко визначеної функції *f*. Важливим аспектом, який слід знати, є те, що алгоритми не суворо прив'язані до жодної мови програмування. Вони є загальними рішеннями як такі.

Тематичне поле алгоритмів настільки розрослося та розширилося, що теорії та основи, що викладаються, допоможуть розробникам вирішити будь-яку обчислювальну проблему. Існує стільки ефективних алгоритмів, які вже опубліковані, як двійковий пошук, сортування за бульбашками, сортування за допомогою вставки, сортування за злиттям , швидке сортування , алгоритми Евкліда для пошуку GCM, для пошуку найкоротшого шляху на графіку тощо. Існує так багато різновидів алгоритмів, деякі групи алгоритмів наведено нижче:

Алгоритми грубої сили. Які є прямим підходом до вирішення проблем, наприклда повторіть додавання, щоб знайти результат задачі на множення.

Алгоритми поділу та завоювання. Це розбиває проблему на невеликі підзадачі, а потім поєднує кожен результат підзадачі, щоб отримати кінцевий результат. Так само, як спочатку розділяєте монети різних номіналів на різні відра, а потім підраховуєте кількість монет у кожному відрі, щоб дізнатися, скільки там монет окремих номіналів.

Жадібні алгоритми. Це слідує евристиці вирішення проблем, щоб досягти наступного найкращого стану, щоб знайти остаточний найкращий стан. Так само, як знайдете менш круту ділянку, по якій легко піднятись в гору.

Динамічне програмування. Підхід, який такий самий, як розділити і завоювати, але ділить проблему на підзадачі таким чином, що їх результати можуть бути повторно використані для інших підпроблем.

Такі методології допомагають створити хороший алгоритм, який має наступні визначальні характеристики:

Точність – він знає точні та правильні кроки для виконання.

Унікальність – дані для поточних інструкцій надходять лише з попередньої інструкції.

Кінцевість – Алгоритм закінчується наданням результату після виконання кінцевої кількості інструкцій.

Загальність – алгоритм містить хороший набір входів, а не строго один вхід.

Більше, ніж широкий кругозір застосувань у реальному світі, він виступає потужним об'єктивом для розгляду проблеми. Це допомагає нам вирішити, вирішувана проблема чи ні. Якщо так, то як, як швидко і наскільки точно? Якщо ні, тоді алгоритм знову допомагає вирішити, чи зможемо розв’язати його частину.

Алгоритми широко використовуються в програмуванні, тому що комп'ютерні програми приймають різні алгоритми, що працюють на комп'ютерному обладнанні, що має процесор та пам'ять, і ці компоненти мають обмеження. Процесор не безмежно швидкий, і пам’ять не вільна. Вони є обмеженими ресурсами. Їх потрібно використовувати з розумом, і хороший алгоритм, ефективний з точки зору часової та космічної складності, допоможе вам це зробити.

Як і будь-які інші технології, розробка алгоритмів у програмуванні також постійно розвивається, оскільки комп'ютерне обладнання постійно розвивається. Від традиційних машин x86 до суперкомп’ютерів до комп’ютерів Quantum відбулися революційні зміни у вирішенні проблем. Наявність міцних знань щодо проектування алгоритмів - це те, що відрізняє кваліфікованого програміста від решти. Сучасні ресурси насправді не зобов’язують вивчати алгоритми з такою кількістю розроблених програмних засобів та бібліотек, але глибоке розуміння того самого допоможе вам набагато більше.

Незважаючи на те, що коли-небудь у нас є неймовірно швидкий процесор та безперервна пам’ять, все одно доведеться вивчити алгоритм, розробити їх так, щоб перевірити, чи закінчується рішення, і робить це з правильним результатом. Нехай це будуть комерційні додатки, наукові обчислення, інженерія, оперативні дослідження або штучний інтелект, в кожній галузі формулювання проблем,

вироблення ефективних алгоритмів вирішення та структури даних, з якими потрібно мати справу, залишаться неминучими назавжди.

До переваг використання алгоритмів можна віднести:

– алгоритми дуже легкі для розуміння та можуть бути написані простою мовою, яку може зрозуміти кожен;

– алгоритми можна розбити на різні частини, що буде легко реалізувати практично;

– використовуючи алгоритми, можна легко зрозуміти послідовність, якої слід дотримуватися при обробці.

Нижче наведено кілька основних недоліків:

– перетворити складне завдання у належні алгоритми непросто;

– це трудомісткий процес, оскільки нам потрібно витратити належний час на написання алгоритму, а пізніше, потрібно реалізувати його мовою програмування;

– складно показати функціональні можливості для кожного кроку введення в алгоритми та важко зрозуміти кожен потік у терміні для циклу та гілки.

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;

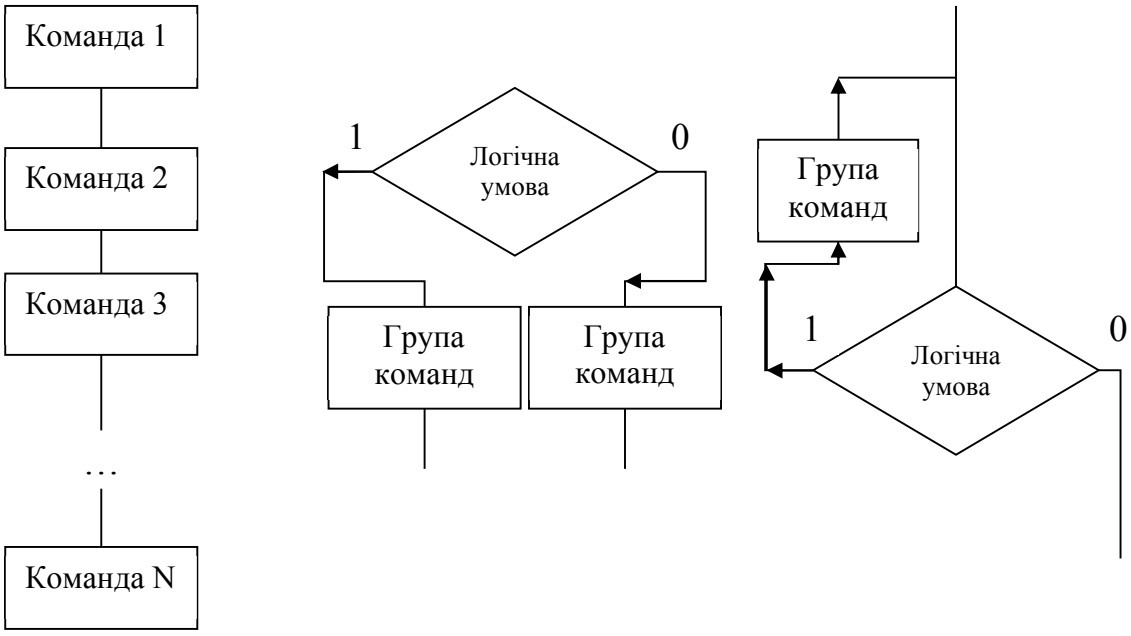
– дискретність. Це означає те, що весь алгоритмом обчислювального процесу повинен бути поділений на окремі етапи, можливість виконання яких виконавцем (комп'ютером) не викликає сумнівів.

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

Числовий процес називається лінійним, де кожен крок операції повинен визначатися в природному порядку кількості кроків.

Процес називається розгалуженим, якщо у будь-якому випадку отримується у формі обробки даних з прецедентних або галузевих даних (в результаті вивчення будь-якого питання, пов'язаного з логічною причиною).

Чисельний процес, що складається з повторення за один або кілька періодів, називається циклічним. Для декількох циклів цикл поділяється на цикл із кількістю (попередньо встановленими) ітераціями та цикл із невідомою

кількістю ітерацій. При цьому умова може перевірятися на початку циклу - тоді мова йде про цикл з передумовою, або в кінці - тоді це цикл з умовою поста.

При вторинній інтерпретації країв алгоритму, щоб досягти кращого інтелекту та унікальності, алгоритм має недоліки у формалізації правил на основі певних завдань зображення. Серед методів, доступних для інформаційних алгоритмів, є:

– 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.

Алгогритми є частиною нашого повсякденного, тому вони використовуються усюди:

– якщо ви будете шукати будь-який товар на будь-якому торговому сайті наступного разу, ви отримаєте подібний тип товару як пропозицію;

– якщо ви переглядаєте будь-яке відео на youtube наступного разу, ви отримаєте пропозиції, подібні до рекомендованого для вас;

– пошукова система Google за лічені секунди з’являються результати;

– таким же чином, якщо ми використовували Facebook, ми отримуватимемо пропозиції друга на основі деяких ключів, таких як назва школи, назва університету, місця тощо.

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

Багато реальних ситуацій можна зручно описати за допомогою адіагральні, що складається з набору точок разом з лініями, що з'єднують певні пари цих точок. Наприклад, точки могли представляти людей, лініями, що поєднують пари друзів; або пунктами можуть бути комунікаційні центри з лініями, що представляють лінії зв'язку. 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.

Графи названі так, тому що їх можна зобразити графічно, і саме це - графічне представлення допомагає нам зрозуміти багато їх властивостей. Кожна вершина позначена точкою, а кожне ребро - лінією, що з’єднує точки, що представляють її кінці. Не існує унікального способу малювання графіка; відносні положення точок, що представляють вершини і ліній, що представляють ребра, не мають значення. Інша діаграма просто зображує відношення випадковості, що утримується між його вершинами та ребрами. Однак часто будемо малювати схему графа та називати її самою графікою. Також будемо називати його точки «вершинами», а лінії - «ребрами». Зверніть увагу, що два ребра на діаграмі графіка можуть перетинатися в точці, яка не є вершиною. Ті графіки, які мають діаграму, ребра якої перетинаються лише на їх кінцях, називаються плоскими, оскільки такі графіки можуть бути представлені в

площині простим способом. Більшість визначень та понять у теорії графів пропонуються графічним поданням. Кажуть, що кінці ребра падають на ребро та навпаки. Дві вершини, які поєднані із загальним ребром, сусідні, як і два ребра, які межують із загальною вершиною. Ребро з однаковими кінцями називається петлею, а ребро з чітко вираженими кінцями - посиланням.

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 використовуються для зручного засобу опису зв'язків між різними об'єктами, прикладами можуть бути дорожні розв’язки, будувельні рішення тощо. Окрім того, теорія графів забезпечує для користувача важливий інструментарій для проведення операцій над графами. Методи та алгоритми теорії графів широко застосовуються в різних частинах математики. Без них неможливо обійтися при синтезі та аналізі дискретних перетворювачів сигналів: функціональних блоків комп'ютерів, комплексів програм тощо.

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:

Матриця інцидентності (рисунок 1.2,а). Візьмемо до уваги, що всі вершини графа та відповідно всі ребра неорієнтованого графа або якщо роглядати орієнтований граф то всі вершини та всі дуги (включаючи петлі) мають свій номер починаючи з одиниці. Граф (неорієнтований або орієнтований) може бути проілюстрований у вигляді матриці nxm, де n - число вершин, а m- число ребер (або дуг). Відповідно значення усіх елементів матриці будуть визначаються за

таким алгоритмом: якщо ребро x_i і вершина v_j інцидентні, то значення відповідного елемента матриці буде рівним 1, в іншому випадку значення рівне 0. Для орієнтованих графів така матриця інцидентності формується за таким принципом: значення елемента рівне мінус одиниці, якщо ребро x_i виходить з вершини v_j; рівне одиниці, якщо ребро x_i заходить в вершину v_j, а в усіх інших випадках рівне нулю.

Для неорієнтованого графа для визначення довільного елемента матриці використовують формули:

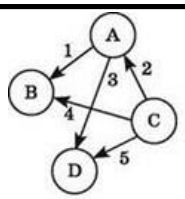
$$a_{ij} = \begin{cases} 1, \textit{ якщо для і вершини j ребро інцедентн} \\ 0, \textit{інакше} \end{cases}.$$

Для орієнтованого графа елементи матриці задаються так:

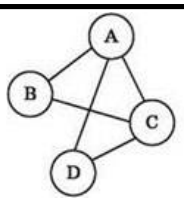
$$a_{ij} = \begin{cases} 1, \textit{ якщо для і вершини j дуга вихідна} \\ -1, \textit{якщо для і вершини j дуга вхідна} \\ 0, \textit{інакше} \end{cases}.$$

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 nxn, 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.

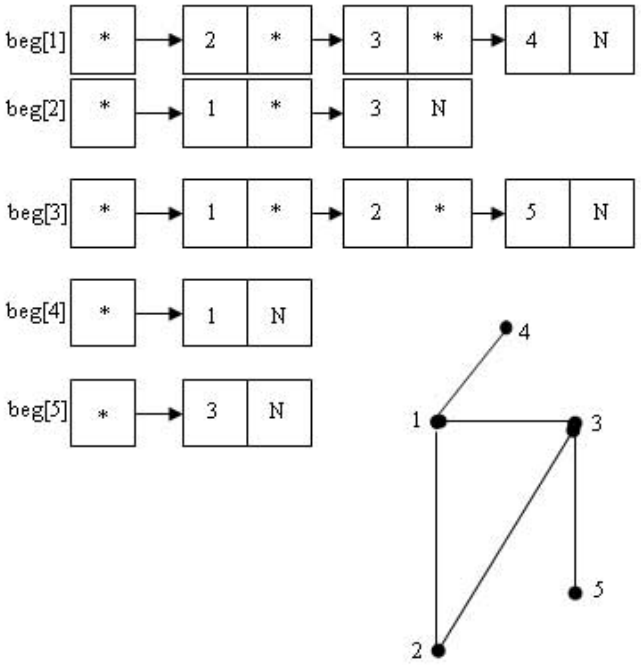


Рисунок 1.3 – Приклад опису графа на основі списку вказівників

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

Блок-схеми можуть допомогти представити складну концепцію легко зрозумілим способом. Ще однією причиною того, що блок-схеми використовуються настільки, є те, що вони допомагають легко відстежувати процес. Також багато інформації можна подати стисло, використовуючи блок-схеми. Не даремно блок-схеми є улюбленим засобом для презентацій інформації широким прошарком суспільства, починаючи від студентів і закінчуючи представниками галузі, діловим співтовариством тощо.

Насправді все стало ще простішим із появою програмного забезпечення блок-схем, оскільки це забезпечує миттєве створення навіть довгих блок-схем. Не потрібно мати такого художнього таланту, аби створити ідеальні форми, як ті, що вже намальовані. Вам потрібно лише вставити його у свій проект заготовки, крім того, звичайно, наповнивши їх своїми власними ідеями.

Adobe Illustrator - чудовий інструмент для створення діаграм, включаючи блок-схеми. Більшість користувачів покладаються на Illustrator для створення логотипів , векторних зображень, піктограм, ілюстрацій до книг тощо. Але програмне забезпечення надзвичайно універсальне і може використовуватися для інших цілей, таких як створення блок-схем та схем, шаблони маркетингу електронною поштою , дизайн текстилю чи дизайн взуття та багато іншого.

Visio – це одне з найпростіших програмних засобів для блок-схем, але воно, можливо, має найповніші функції. Він також постачається з навчальним посібником, якщо все одно знадобиться, тоді як на чудову програму допомоги та підтримки також можна покластися на всякий випадок, якщо у вас виникнуть проблеми. Крім того, причиною того, що Visio називають одним з найпростіших

програмних засобів для блок-схем, є те, що воно дозволяє вводити текст прямо в самі фігури. Це ще не все, редагувати теж просто, оскільки завжди можете замінити фігури без необхідності видаляти їх спочатку. Також доступна функція перевірки правопису, тоді як символи самостійно змінюють розмір, щоб відповідати навколишнім текстам. Додавання додаткових точок роз'єму - це також вітер. Visio пропонує передплатну модель власності.

Smartdraw. При знайомстві може спантеличити величезна кількість шаблонів та об’єктів, які пропонує програма. Тим не менш, початок роботи зі створення блок-схем за допомогою обраного вами шаблону - це абсолютний шлях. Варіанти інтеграції з програмою однаково помітні, так що все, що створено на Smartdraw, можна легко інтегрувати з програмами MS Office, такими як PowerPoint .

Creately. Вибір шаблонів у Creately не такий великий, як у багатьох із цього списку, але все ж має чудові можливості для шаблонів, які він має. Це робить його видатною платформою для створення професійних блок-схем з легкістю. Creately також дозволяє користувачам співпрацювати в режимі реального часу при створенні блок-схеми. Крім того, існують як веб-, так і настільні версії програми. Це ще не все, оскільки низька ціна робить Creately досить дешевою, а також порівняно з аналогами.

LucidChart. Набір шаблонів може бути не таким різноманітним, як багато серед однолітків, хоча LucidChart все ще має достатньо, щоб допомогти розпочати роботу з більшістю проектів. Його користувальницький інтерфейс також простий і привабливий, так що ви можете почати відразу. Він також може співпрацювати з багатьма іншими програмами та послугами. Це разом з безліччю інструментів для спільної роботи робить команди дуже простими для участі у створенні блок-схем. Це ще не все, оскільки члени команди можуть також вносити необхідні зміни в режимі реального часу. LucidChart також не з тих, хто випалює діру в кишенях, маючи пристойну ціну.

Gliffy - це ще одна програма для блок-схематизації, на яку ви можете покластися. Він також повністю заснований на хмарі та сумісний з Google Drive, Confluence та JIRA.

Це дуже полегшує користувачам співпрацю в режимі реального часу під час виконання складних проектів, у яких бере участь кілька команд. Додаток також доступний як у безкоштовній, так і в платній версії, причому перша має менше шаблонів та інших функцій, ніж пропонується у платній версії. Версія браузера є абсолютно безкоштовною і найкраще підходить для студентів чи інших, яким потрібні відносно прості блок-схеми. Інтерфейс також надзвичайно простий, оскільки вам просто потрібно перетягувати різні окремі фігури. Що стосується платних версій, Gliffy випускається у двох версіях, версіях Personal і Team, що виставляються щомісяця або щороку.

YED Works. Для редагування графіку yED, можливо, знадобиться трохи звикнути, але це, мабуть, невелика ціна, яку потрібно заплатити, щоб погодитись із сучасним програмним забезпеченням. В іншому випадку це привабливий інтерфейс, з яким вас вітають, і як тільки ви це зрозумієте, створення блок-схем в редакторі графіків yED буде легким. Ще одним плюсом редактора графіків yED є те, що він пропонує інтерфейс із вкладками, щоб ви могли одночасно відкривати декілька файлів. Інтерфейс також модульний, так що непотрібні властивості можна закрити, щоб звільнити більше області перегляду. Ви також можете зберегти фактичні схеми у декількох форматах, які включають навіть HTML та PDF.

LibreOffice Draw - одна з найкращих безкоштовних програм для діаграми блоків. Те, що він має багато спільного з інтерфейсом MS Office, є іншою його великою перевагою, оскільки у вас є занадто звичне налаштування для початку. Ще однією надзвичайно симпатичною особливістю LibreOffice Draw є спосіб, яким він підтримує групування. Draw також сумісний з декількома форматами, включаючи SWF-файли Flash, XML - формат за замовчуванням.

Dia - ще одна програма блочного діаграмування з відкритим кодом, яка, схоже, була натхненна Microsoft Visio. Це пояснює набір функцій, багато в чому схожий на MS Visio . Він також підтримує майже всі формати файлів, про які ви думаєте, які включають формат .vdx для Visio. Інтерфейс досить простий у використанні, зліва розміщення декількох елементів управління та фігур, тоді як решта дисплея складають основну область малювання.

Draw.io. Це, здається, одне з найпростіших програмних засобів блок-схеми, яке найкраще підходить для простих або менш інтенсивних проектів. Вам не потрібно встановлювати будь-яке конкретне програмне забезпечення для цього самого, а також Draw.io базується на браузері. І це також безкоштовно. Інтерфейс надзвичайно простий у використанні, оскільки доступні фігури та інші опції показані зліва, а решта дисплея знаходиться там, де фактично складається блок-схема.

Pencil Project - це безкоштовне програмне забезпечення з відкритим кодом, яке можна легко використовувати для створення блок-схем. Активна підтримка спільноти розробників також гарантує належне обслуговування програмного забезпечення завдяки новим функціям та іншим вдосконаленням, які регулярно надаються. Початок роботи з Pencil Project також дуже легкий, так що ви в найкоротші терміни впораєтеся з блок-схемами навіть для новачка. Ще один величезний плюс Pencil Project полягає в тому, що його також можна використовувати як розширення Firefox.

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
YED Works	+	+	-	+	+
FCEditor	+	+	-	-	+
Auto flowchart	+	+	-	-	+
Flying Logic	+	+	+-	-	+
Draw.io	+	-	-	-	+
Lucichart.com	+	-	+-	-	+
Draw.io	+	+	+-	+	+
MS Visio	+	+	+-	+	+

Результати огляду програмних додатків для побудови блок-схем показали, що на сьогоднішній день дний напрямок розробки програмного забезпечення широко використовується в різних галузях широким колом людей. При цьому існують як онлайн так і локальні програмні пакети, що значно спрощує використання даних програмних засобів в повсякденному житті.

1.4 Постановка задач кваліфікаційної роботи та шляхи її вирішення

В розділі було проаналізовано поняття алгоритму та виділено основні класи для подальшого аналізу. Розглянуто підходи до візуалізації алгоритмів у вигляді блок-схем, що у повній мірі підходить для використання у теорії графів. Досліджено онлайн та офлайн програмні системи для візуального відображення алгоритмів у вигляді блок-схем для виілення основних структурних модулів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1) дослідити та провести класифікацію існуючих видів алгоритмів;

2) проаналізувати можливості теорії графів для відображення блок-схем алгоритмів у вигляді дерев;

3) провести аналіз сучасних програмних додатків побудови та відображення блок-схем;

4) дослідити функції цифрової бібліотеки tsimplegraph;

5) спроектувати та провести моделювання структури програмного додатку створення та відображення блок-схем алгоритмів;

6) програмно реалізувати систему створення та відображення блок-схем алгоритмів, провести її тестування та порівняльний аналіз з програмами-аналогами.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ СТВОРЕННЯ БЛОК-СХЕМ АЛГОРИТМІВ

2.1 Технології створення та візуалізації алгоритмів

І блок-схеми, і алгоритми - це інструменти, які пояснюють, як програма працюватиме/поетапний процес. Між цими двома інструментами є деякі відмінності. Блок-схеми та алгоритми широко використовуються в області комп'ютерного програмування. Хоча алгоритми є поетапним і поетапним аналізом програми, блок-схеми наочно та графічно пояснюють ці етапи. Разом полегшити вирішення проблем, пояснивши всі етапи. Експерти рекомендують початківцям спочатку виписувати алгоритми, використовувати блок-схеми для представлення алгоритмів, а вже потім приступати до написання програми. Це значно полегшує процес написання високоякісної програми.

Як зазначалося, алгоритми - це логічна та поетапна методологія, що використовується для розв’язання проблеми/процедури розв’язання проблем. При спробі вирішити комп'ютерну або математичну задачу алгоритми є першим кроком і вони охоплюють обробку даних, міркування та обчислення. Блок-схема, з іншого боку – це графічне зображення/ілюстрація алгоритмів, що використовують різні символи, фігури та сполучники (стрілки), що означає процес програми. Головною метою блок-схем є аналіз кількох різних процесів із використанням стандартизованої графіки. Набагато простіше пояснити алгоритми, використовуючи блок-схеми, враховуючи той факт, що кожен крок і елемент можна виділити, а взаємозв'язок між кожним кроком можна зрозуміло пояснити за допомогою різних кольорових та розмірних вікон та стрілок.

Тепер, коли було визначено мету та значення як блок-схем, так і алгоритмів, пояснимо, як використовувати блок-схеми для представлення алгоритмів. Алгоритми використовуються в основному для опису математичних та комп'ютерних програм, тоді як блок-схеми є більш універсальними та можуть бути використані для опису множини процесів у різних сферах, що включає представлення алгоритмів. Отже, можна використовувати блок-схеми для представлення алгоритмів як інструмент планування, який допомагає

організувати етапи процесу програми наочно наочно. Блок-схеми можна розглядати як схему алгоритму, що графічно пояснює алгоритм та етапи. Універсальність блок-схеми допомагає представляти алгоритми, оскільки для цього не потрібно використовувати конкретні форми та розміри, скоріше, форми та розміри можуть бути розроблені для пояснення певного алгоритму. Різні форми та розміри будуть пов'язані між собою стрілками, забезпечуючи візуальне логічне пояснення алгоритму.

Давайте розглянемо ще деякі переваги використання блок-схем для представлення алгоритмів.

Оскільки блок-схеми є графічним зображенням кроків алгоритму, вони допомагають впорядкувати та полегшити розуміння логіки та наступних кроків/етапів. З огляду на спрощення навіть складного алгоритму, спілкування та пояснення того самого іншим стає набагато кращим та легшим. Оскільки більша кількість людей розуміє кроки програми, аналіз проблеми стає простішим, і більша кількість рішень може бути можливою завдяки ідеям, наданим зацікавленими особами.

Однією з інших переваг використання блок-схем для представлення алгоритмів є те, що вони підвищують ефективність процесу кодування - забезпечуючи чіткі інструкції, полегшуючи тим самим роботу програмістів. Крім того, оскільки кроки чітко і наочно проілюстровані, стає набагато простіше та ефективніше виявляти дефекти / помилки в програмі, допомагаючи в процесі тестування. Нарешті, оскільки будь-який процес вимагає належної документації, використання блок-схем для представлення алгоритмів гарантує, що складні тексти пояснюються на одній сторінці, зрозумілі інструкції, які можна використовувати для навчання нових співробітників та розуміння процесів для створення більшої кількості алгоритмів.

Форми, лінії та кольори на блок-схемі дозволяють точно візуалізувати алгоритм, який просто визначає послідовність дій, виконаних для завершення процесу. Залежно від складності, можна використовувати програмне забезпечення для створення блок-схем, що представляють алгоритми, або створити таку на папері. Алгоритми присутні скрізь - майже в усіх сферах життя.

Для тих, хто готує більш складні алгоритми комп'ютерних операційних систем, має сенс спочатку обговорити кроки, які будуть виконуватися для виконання процесу, а потім використати блок-схему для візуального фіксації всіх кроків.

Створення блок-схем для представлення алгоритмів настільки просте або настільки складне, наскільки це видає користувач. Форма прямокутника представляє завдання в блок-схемі, і ось як повинна починатися блок-схема - з написання першого завдання в прямокутнику, намальованому на крайній лівій стороні. Як уже згадувалось, блок-схеми рухаються зліва направо - і, отже, для опису наступного кроку слід намалювати ще один прямокутник із написаним завданням. Після того, як всі кроки виставлені, стає очевидним, що існує логічна послідовність кроків процесу, причому взаємопов'язані стрілки показують порядок виконання завдань.

Рішення, що приймаються в будь-якому процесі, повинні враховуватися в блок-схемі. Рішення в блок-схемі представлені ромбоподібною коробкою та пов'язані із завданням за допомогою стрілки. Ці форми рішення та стрілки допомагають тим, хто бере участь у проекті, вибрати будь-який із логічних шляхів, який був би найбільш доречним для відповіді на питання, яке вони виклали у першому прямокутнику. Цей тип логічного мислення покращує міркування та аналітичні здібності. Творцям та зацікавленим сторонам процесу було б легко дійти висновку, що означає заповнення блок-схеми. Після того, як блок-схема завершена, проект можна розпочати та успішно завершити.

Повсякденні завдання варіюються від простих до складних, і алгоритми виконання завдань відповідали б за своєю складністю. При використанні блок-схем для представлення алгоритмів, чим складніший алгоритм, тим більше кроків, символів завдань та символів точки прийняття буде присутній на блок-схемі. Звичайно, існує ще кілька символів, які існують для блок-схем, але базовий процес легко та візуально фіксується за допомогою форм завдання та рішення, а також взаємопов'язаних стрілок. При розробці блок-схеми цілком можливо, що зацікавлені сторони пропускають деякі кроки або забувають згадати деякі найдрібніші деталі. Гнучкість блок-схем дозволяє додавати (або

видаляти) етапи або переставляти ті, що вже є, щоб гарантувати, що алгоритм точно фіксується та представляється.

Можна впевнено сказати, що блок-схеми - це точна та чітка графічна візуалізація алгоритму, і що використання блок-схем для представлення алгоритмів робить алгоритми простими для вираження, використання та аналізу. Використання слів у алгоритмах замінено символами, фігурами, лініями та кольорами на блок-схемах – ефект сильніший і робить обтяжливе завдання створення алгоритмів цікавим, швидшим та ефективнішим.

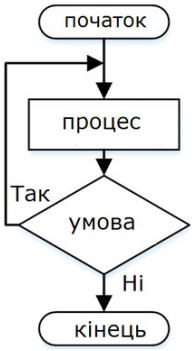
У блок-схемі кожній формальній конструкції відповідає певна геометрична фігура або пов'язана лініями сукупність фігур. Основні графічні елементи наведені в таблиці 2.1:

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

Найменування	Позначення	Функція
Термінатор		Елемент відображає вхід/вихід із/в зовнішнє середовище.
Процес		Елемент відображає обробку даних будь-якого виду (зміна форми подання, розташування даних, значення даних тощо).
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.
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).

Окрім представлення алгоритмів у вигляді блок-схем існує ще декілька варіантів. Приклади задання алгоритмів наведено на рисунку 2.1.

Крок 1: Почніть
Крок 2: Отримайте два числа як вхідні дані та збережіть їх у змінній як a та b
Крок 3: Додайте число a & b та збережіть у змінній c
Крок 4: Друк c
Крок 5: Зупиніть.




```
OUTPUT "Please enter the test result."
Score ← USERINPUT
IF score < 5 THEN
   OUTPUT "Try harder next time."
ELSE IF score >= 5 THEN
   OUTPUT " You scored 50%."
ELSE IF score > 7 THEN
   OUTPUT "Good result."
ELSE IF score > 8 THEN
   OUTPUT "Excellent result."
ENDIF
```

словесний графічний (блок-схеми) псевдокод

Рисунок 2.1 – Приклади задання алгоритмів

Кожних з підходів має свої преваги та недоліки, проте їх використання значно спрощує спідкування між членами команди на етапі проектування чи розв’язку деякої задачі.

2.2 Цифрова бібліотека обробки 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 набір компонентів для створення програм векторної графіки під Borland Delphi (CAD, GIS, SCADA, VISIO). Бібліотека TSimpleGraph може бути використана для створення сюжетів, креслень, електричних, концептуальних та мнемонічних карт, а також для створення інтерактивних планів з високим рівнем деталізації для різних об'єктів, будівель або їх частин.

Бібліотека TSimpleGraph також може використовуватися для візуалізації інформації, що надходить відрізних датчиків у режимі реального часу. SCADA-системи, GIS-системи та CAD-системи також можуть бути розроблені за допомогою цієї бібліотеки. TSimpleGraph для Borland Delphi Builder можна назвати міні-аналогом для Visio .

Більше того, розробник отримує повний контроль над усіма об'єктами під час виконання, включаючи можливість доступу до об'єктів як за ідентифікатором, так і за іменем. Користувацькі поля та значення, а також звичайні або багаторядкові підказки можна вказати для кожного об'єкта і зберігати в тілі документа.

Зразок програми редактора векторної графіки (разом із вихідним кодом на Borland Delphi) розповсюджується разом із бібліотекою. Цей зразок програми реалізує всю функціональність продукту (включаючи малювання об'єктів, зміну їх властивостей за допомогою інспектора об'єктів, групування об'єктів, вирівнювання об'єктів, роботу з растровими зображеннями, роботу з шарами, роботу з сіткою, створення бібліотек зображень, роботу з діаграмами в межах документ тощо)TSimpleGraph для Borland Delphi Builder дозволить вам створювати можливості векторної графіки у вашому програмному забезпеченні. За його допомогою ви зможете створити серйозні повнофункціональні системи обробки графіки. Реалізація бібліотеки базується на Windows GDI і підтримує такі графічні об'єкти:лінія, коробка (включаючи округлу коробку), коло (еліпс), полілінія, багатокутник, крива, текст (включаючи багаторядковий текст), растрове зображення растрового зображення (wmf, bmp, ico, jpg, jpeg, gif, emf та багато інших).

Існує один інструмент для об’єктів лінії, полілінії, багатокутника (утримання натиснутої кнопки модифікатора дозволить перетворити лінії в полілінії або багатокутники і навпаки).

Бібліотека TSimpleGraph має таку функціональність:

– один документ може містити кілька діаграм, на які можна робити перехресні посилання з різних об’єктів, що належать до різних сторінок (діаграм);

– фон сторінки та фонове зображення повністю підтримуються;

– підтримка декількох шарів (шар може бути редагованим, читабельним і прихованим);

– текстовий та двійковий формат документа;

– підтримуються групи об’єктів;

– підтримується скасування/повтор;

– гнучкі сполучні лінії;

– стилі лінійних шапок;

– підтримуються два типи сіток: фіксована сітка та точна сітка, що надзвичайно важливо в процесі проектування; зміщення сітки;

– підтримується вирівнювання об’єкта;

– підтримується дзеркало об'єкта;

– підтримується обертання об’єкта (лише 90);

– підтримка кутового тексту;

– градієнтне заповнення пов’язаних об’єктів;

– підтримується справжня прозорість;

– об’єкти та групи об’єктів мають такі атрибути: ідентифікатор, ім’я, підказка, тег, шар, поля користувача (зберігаються в тілі документа) та інші залежно від типу об’єкта;

– до документів може додаватися різноманітна інформація (як локальна, так і міститься в базі даних);

– об'єкти можна копіювати, дублювати, переміщувати, вирізати або копіювати в буфер обміну Windows або інші програми;

– об'єкти можуть зберігатися в базі даних для подальшої оптимізації проекту;

– повний контроль над усіма об'єктами під час виконання, включаючи можливість доступу до об'єктів як за ідентифікатором, так і за іменем;

– підтримується масштабування зображення;

– зображення можна роздрукувати;

– зображення можна експортувати у WMF;

– документи та бібліотеки зображень можна зберігати як локально, так і у внутрішній базі даних;

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.

2.3 Моделювання програмного додатку візуалізації блок-схем

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

Читається і гнучка. Найкраща перевага полягає в тому, що коди діаграми легко читаються будь-яким програмістом, який розуміє навіть незначну частину програми. За допомогою цього комп'ютерний програміст може запустити програму за допомогою кодів і змінювати або використовувати їх будь-коли, коли захоче. Ще одна річ, яка робить UML настільки необхідною програмою для розробки програмного забезпечення, - це те, наскільки гнучко налаштовувати її відповідно до використання компанії або технології.

Належне спілкування для архітектури програмного забезпечення. Концепцією системи є насправді архітектура програмного забезпечення, оскільки це структура, від якої залежить процес та ефективність. UML (уніфікована мова моделювання) - це розширена мова, яка використовується для моделювання програмного забезпечення. Це також допомагає оцінити ефективність роботи користувачів, а також відстеження, безпеку та дає відповідні вказівки щодо призначеної операції. Оскільки UML має властивості широко охоплювати, UML - ідеальна мова для передачі візуальної інформації про архітектуру програмного забезпечення великій кількості працівників, які її використовують.

Інструмент планування перед створенням програми. UML може допомогти вам спланувати програму, перш ніж наступить час її програмувати. Інструмент може допомогти у створенні кодів, які можуть створити модель UML. Діаграму порівняно легко змінити і може допомогти вам зменшити накладні витрати, коли ви нарешті впроваджуєте програму.

Легко налагоджувати будь-яку проблему. Більш обширна програма може зайняти до годин і годин пошуку, щоб знайти помилку в ній, а також може спричинити проблеми в системі пізніше. Отже, програма добре розроблена за допомогою UML, так що кожне із завдань містить свої власні коди, які програмісту легше налагоджувати.

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

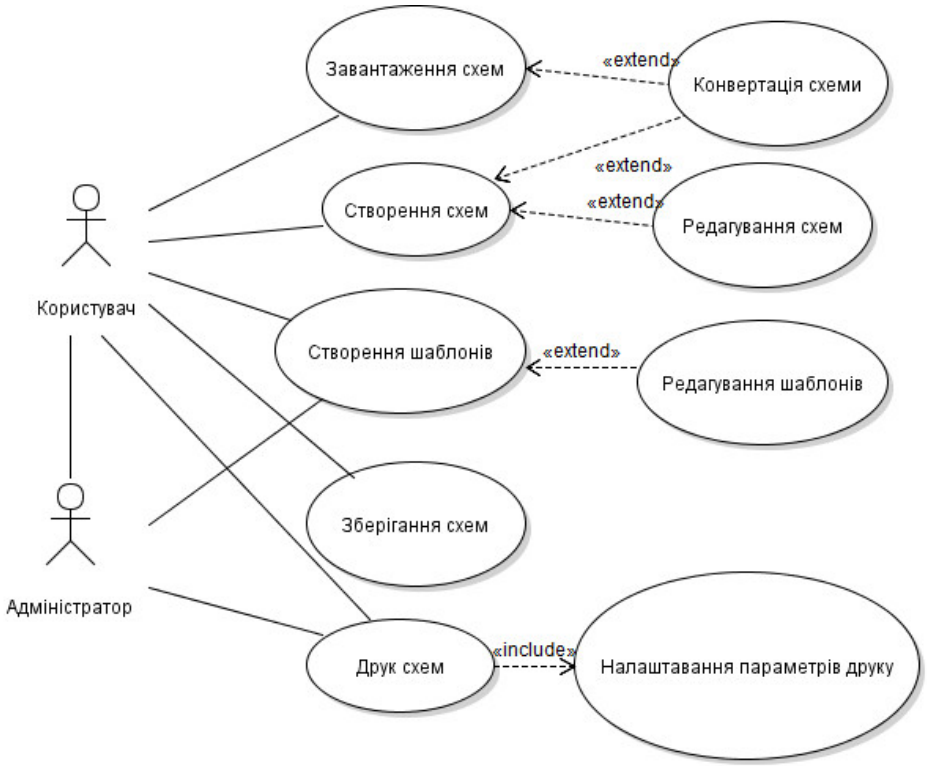


Рисунок 2.2 – Діаграма прецедентів програмного додатку опису алгоритмів за допомогою блок-схем

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

Моделювання послідовності кроків роботи з програмною системоюпроылюстрував, що в програмі:

– відсутні неоднозначні дії, що можуть сповільнити програму;

– для досягнення мети користувачу слід виконати мінмальну кількість дій;

– час на освоєння програми є мінімальний та не залежить від знань комп’ютерної грамотності.

Виділення двох типів акторів які нають певний набір функцій, при робті з програмною системою є цілком достатнім вирішення завдань користувачів при цьому робта програмної розробки буде проходити без залучення значних апаратних ресурсів. Час реакції на дії користувачів буде мінімальним, а результат своїх дій користувач бачитеме відразу.

Для реалізації програмного модуля розробки та візуалізації блок-схем був сттворений алгоритм для організації взаємодії користувача та програмного модуля. В йог ооснову був покладений принцип покрокового ввооду векторних примітивівта представлення їх у вигляді графа, що дозволить в подальшому отримувати зв'язку блок-схему та дозволить працювати з її окремими частинами. При цьому звязність окремих елементів блок-схеми не буде розриватись. Часові втрати при проведенні є відносно незначними та дозволяють значно зменшити час при подальшій обробці блок-схем.

Запропонований алгоритм містить наступні етапи роботи:

Етап 1. Початок роботи з програмою (вибір режиму роботи: створення нового полотна чи модифікація уже існуючого).

Етап 2. Проведення перевірки наявності та правильної роботи усіх копонентів програмної системи, при необхідності завантажити необхідні компоненти.

Етап 3. Якщо було обрано режим модифікації уже існуючої блок-схеми, то проводиться завантаження та відуалізація попередньо збереженого графа.

Етап 4. Формуємо структуру графа блок-схеми на основі встановлення відповідностей та взаємозв'язкуів між окремими структурними елементами візуального відображення алгоритму.

Етап 5. Перевірка чи додається новий елемент. Якщо новий елемент додається то встановлюються його взязки з сусідніми блоками та визначається набір внутрішніх параметрів. Якщо нових елементів не додано переходимо на крок 8.

Етап 6. Проводиться перевірка можливості встановлення зв'язків між новим блоком та елементами. Перевірка здійснюється за принципом наявності вільних слотів для створеня звязку. Якщо перевірка успішна то прееходимо на крок 7 в іншому випадку переходимо на крок 5.

Етап 7. Провдиться внесення внутрішніх параметрів візуального елемента (потрібні пояснювальні надписи) та візуальне розміщення компонента та лінійзвязку на робочій області блок-схеми.

Етап 8. Візуальна перевірка якості отриманої блок-схеми, якщо схема виконана вірно перходимо на етап 9 в іншому випадку повертаемось на етап 5.

Етап 9. Збереження отриманої блок-схеми.

Запропонований алгоритм містить всі етапи для проектування та візуального відображення структури алгоритму у вигляді блок-схеми та дозволяє користувачам швидко та інтуїтивно зрозумілій формі представити необхідний алгоритм. При цьому слід зазначити, що в алгоритмі не предбачено перевірку правильності обрання відповідного візуального компанента, цей етап повністю лежить в сфері відповідальності користувача. Графічно запропонований алгоритм можна представити у вигляді наступної блок-схеми (рисунок 2.3):

Рисунок 2.3 – Блок-схема алгоритму додавання нового елемента до блок-схеми алгоритму

До переваг розробленого підходу такі:

– простоту роботи та високу швидкодію;

– принципи встановлення звязків є простими та грунтуються на теорії графів;

– алгоритм не є ресурсозатратним тому для його реалізації та функціонування не потрібні значні апаратні ресурси;

– можливість створення алгоритмів довільної складності у реальному часі.

Серед недоліків є:

– функціонал буде залежати від бази даних візуальних компонентів та не пердбачає перевірку відповідності стандартам;

– для робти з алгоритмом користувач повинен мати знання про принципи побудови блок-схем алгоритмів.

3. ПРОГРАМНА СИСТЕМА СТВОРЕННЯ ТА ВІЗУАЛІЗАЦІЇ БЛОК-СХЕМ АЛГОРИТМІВ

3.1 Структура програмної системи роботи з блок-схемами алгоритмів

Вивід графіки в Windows здійснюється за допомогою функцій GDI (Graphics Device Interface - інтерфейс графічних пристроїв). GDI - це посередницька підтримка між графічним драйвером на базі Microsoft Windows NT та додатком. Додатки викликають функції Microsoft Win32 GDI для надсилання запитів на вихід графіки. Ці запити направляються в режим ядра GDI. Потім GDI в режимі ядра надсилає ці запити до відповідного графічного драйвера, такого як драйвер дисплея або драйвер принтера. GDI в режимі ядра - це модуль, що постачається системою, і який не можна замінити.

GDI взаємодіє з графічним драйвером через набір функцій графічного інтерфейсу драйвера пристрою (графічний DDI). Ці функції визначаються за допомогою префікса Drv . Інформація передається між GDI та драйвером через параметри введення / виводу цих точок входу. Драйвер повинен підтримувати певні функції DrvXxx для виклику GDI. Драйвер підтримує запити GDI, виконуючи відповідні операції з відповідним обладнанням перед поверненням до GDI.

GDI включає в себе багато можливостей графічного виводу, усуваючи необхідність драйверу підтримувати ці можливості і тим самим даючи можливість зменшити розмір драйвера. GDI також експортує сервісні функції, які може викликати драйвер, ще більше зменшуючи обсяг підтримки, яку повинен надавати драйвер. Функції служб GDI ідентифікуються за їх префіксом Eng , а функції, що забезпечують доступ до структур, що підтримуються GDI, мають імена у формі xxx obj * _ * xxx .

На рисунку 3.1 проілюстровано інтерфейс взіємодії між програмними додатками створеними для операціійних систем Windows та відповідно графічним ядром орпераційної системи.

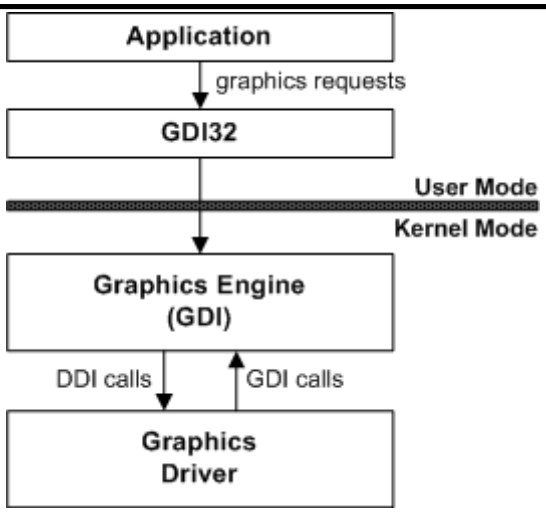


Рисунок 3.1 – Структура взаємодії програмної системи візуалізації блок-схем з графічною частиною ядра операційної системи

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 (рисунок 3.2). Цей клас має множину 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.

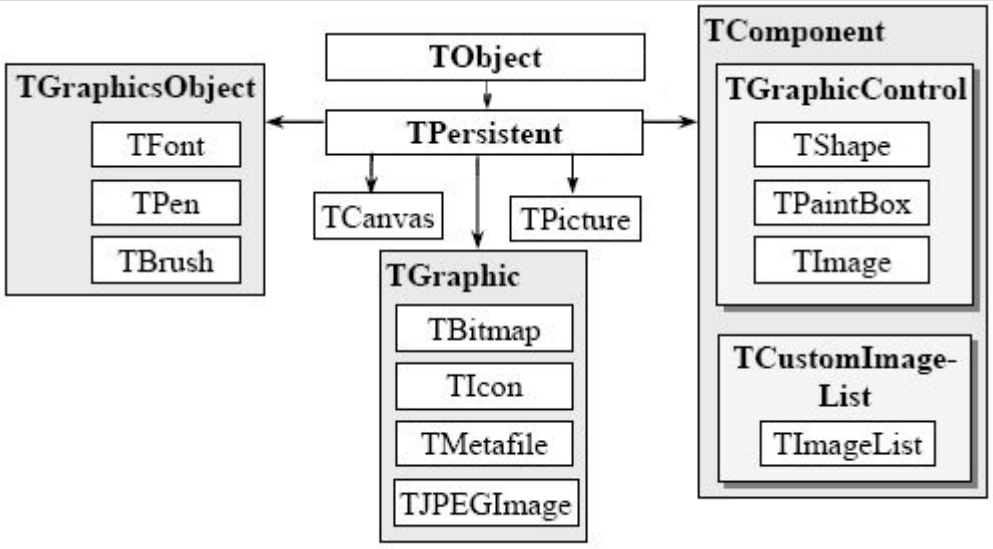


Рисунок 3.2 – Ієрархія класів для обробки візуальної статичної інформації

Архітектура програмного забезпечення призначена для реалізації програмних програм для створення та редагування блок-схем. Архітектура стосується набору програмних компонентів, а також зв’язків та способів організації обміну інформацією між ними. Першим завданням, яке необхідно виконати при створенні системної програми для блок-схеми, є ідентифікація її візуальних компонентів.

На основі аналізу системних вимог встановлено набір усіх функцій, які програма повинна підтримувати. Потім отримані функції об'єднують у логічно пов'язані групи. Кожна з цих груп може бути складовою частиною програмної системи.

Архітектура програми включає результати попереднього огляду систем даного класу. Без таких описів важко створити послідовну картину дуже тонких деталей або принаймні десятка окремих класів. Запропонована архітектура ілюструє, що при її розробці розглядаються альтернативи і робиться обґрунтований вибір остаточної організації системи.

Загальна архітектура програмного забезпечення представлена у вигляді структури, що ілюструє октері функйціональні блоки різних частин цілісної програмної системи. Стуруктура наведена на рисунку 3.3.

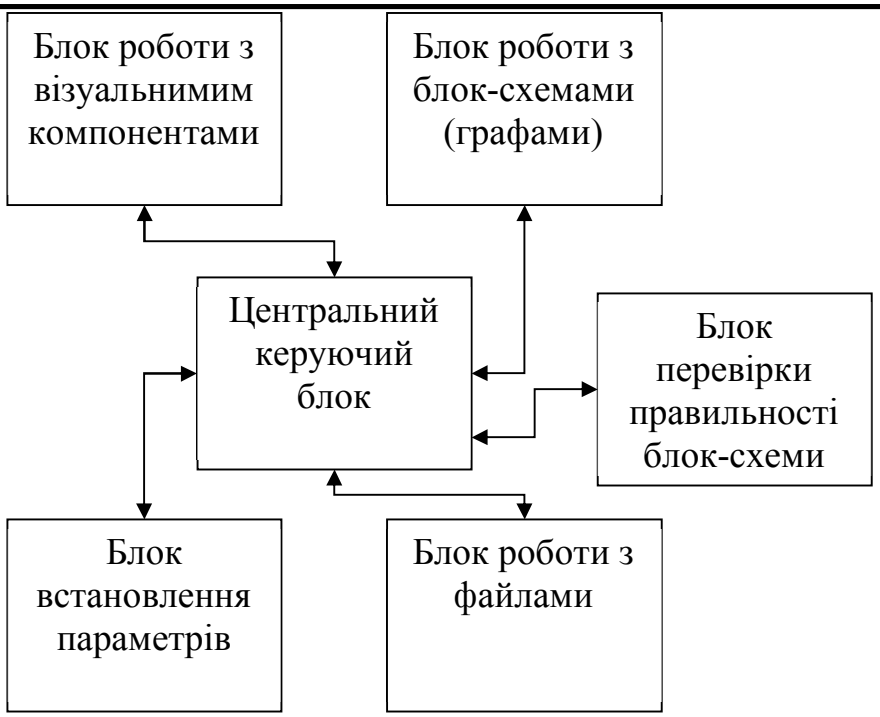


Рисунок 3.3 – Узагальнена структура програмного додатку візуалізації блок-схем алгоритмів

При аналізі структури програмного додатку детальніше зупинемось на основних програмних модулях:

- “Блок встановлення параметрів” – набір функцій та процедур, що надають користувачеві можливості регулювати налаштування робти програми або змінювати зовнішній вигляд графічного інтерфейсу користувача. Інші функції включають налаштування параметрів системи відображення інформації для користувача (колір, товщина лінії, налаштування шрифту тощо). Цей пристрій підтримує робту системи проте не є головним.
- “Блок роботи з фалами” – є частиною функцій для роботи з системою реєстрації робочої станції. Основні функції – це робота з вхідними множинами файлів із схемою алгоритму, набір функцій для пошуків, відкриття, збереження та публікація функцій. У більшості функцій у цьому полі використовується вікно, схоже на вікно ОС Windows, що полегшує освоєння програми.
- “Блок робти з блок-схемами алгоритмів” – в даній структурній одиниці програмної системі зосереджений набір функцій для безпосереднього створення, редагування блок-схем алгоритмів. В ній реалізовані основні

інтерфейсні можливості для користувача та саме він забезпечує пердачу команд від користувача до програмної системи.

- “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, означає основний компонент дизайну програми, який з'єднується з окремими елементами програми. Інші особливості, властиві цій збірці, включають можливість перевірки цілісності та дійсності програми. Окрім того в ньому закладені принципи для перевірки провильності та цілісності передачі даних між окремими функціональними блоками, що в рази піднімає стійкісні характеристики роботи програмної системи в цілому.

Відповідальність за кожен компонент походить від запропонованої архітектури. Компонент відповідальний, а зв’язок між окремими компонентами зведений до мінімуму. Архітектура визначає правила спілкування між пунктами порядку денного та описує, які компоненти можна використовувати безпосередньо, а які не слід використовувати безпосередньо та повністю. Ієрархічна модель програмної системи якісно відрізнає її від інших модклей, оскільки на її основі легко можна нарощувати функційональні моливості програмної розробки в цілому.

Графічний інтерфейс користувача був розроблений на етапі аналізу вимог. Архітектура описує основні аспекти графічного інтерфейсу користувача (GUI). Використання графічного інтерфейсу користувача може врешті-решт призвести до популярності або відмови програмного продукту. При створенні графічного інтерфейсу користувача було вирішено розділити головне вікно на окремі практичні області. Це розділення зменшило візуальне навантаження на користувача та скоротило час користувачів з меншим досвідом та навичками для оволодіння програмою з комп'ютерами та графічним інтерфейсом операційних систем. Результати проектування та створення графічного інтерфейсу користувача показано на рисунку 3.4.

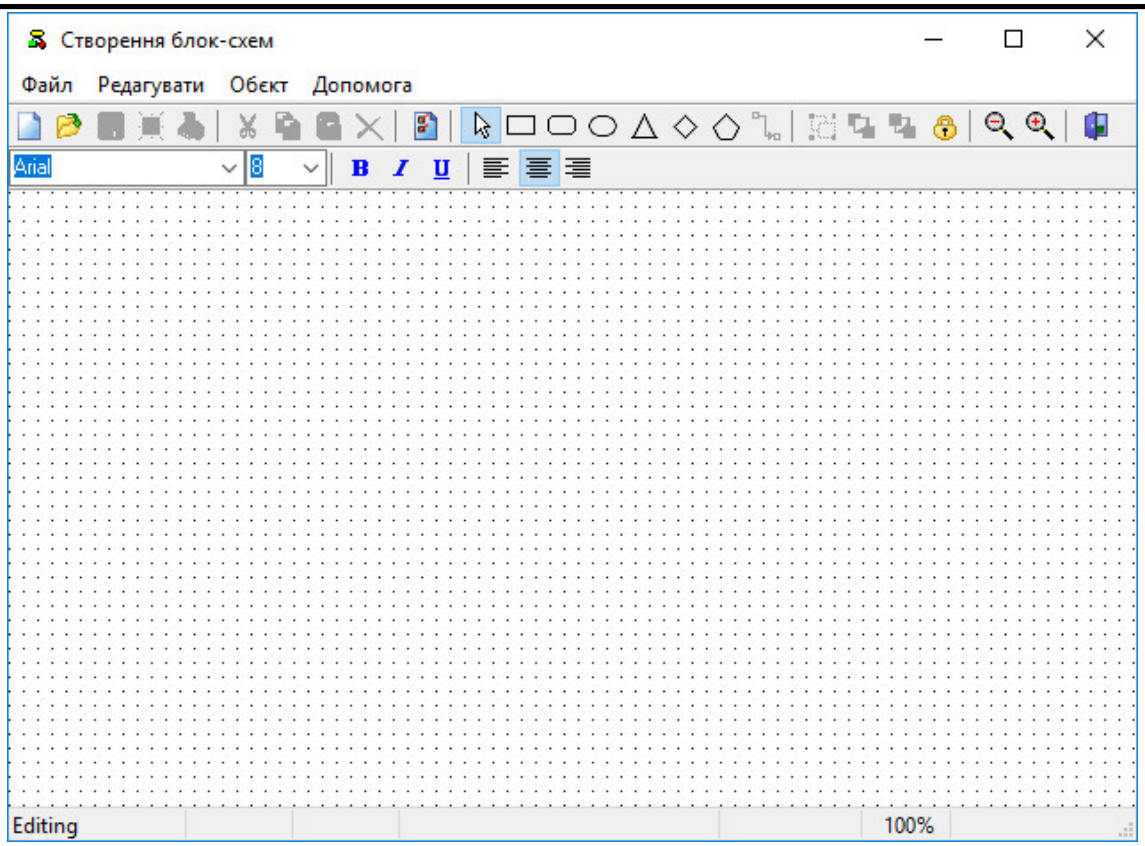


Рисунок 3.4 – Загальний вигляд головного вікна програмного додатку

Серед основних структурних елементів графічного інтерфейсу слід відмітити:

- основна робоча область програмного додатку. Даній області було надано максимальну корисну площу програмної ситеми, оскільки більшість часу користувач буду працювати саме на ній. За допогогою цієї області проводяться основні маніпуляції з блок-схемами та відбувається візуалізація цих робіт.
- Блок для роботи з візуальними компонентами призначений для встановлення параметрів компонентів блок-схем програми та характеристик виводу блок-схеми на екран. Серед параметрів, які може редагувати користувач є : встановлення кольору шрифтів, типу, розміру, місце розташування написів на елементах блок-схеми, параметри вводу/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. Використання даного принципу гарантує, що при необхідності внесення змін в програмний додаток, це відбудеться без значних часових затрат.

3.2 Підсистеми аналізу цифрових зображень

Під час реалізації програмного додатку була використана бібліотека SimpleGraph, що надає модливості роботи з зв'язними графами. Оскільки при реалізації програми блок-схеми представляються як направлені графи, то набір функцій з бібліотеки SimpleGraph є оптимально підходящим для програмної реалізації додатку. Одним з важливих етапів роботи з блок схемами є етап створення нового елемента. Для цього використовуємо процедури додавання нового вузла в структуру графа. Відповідних код функції наведено нижче:

```
function PrettyNodeClassName(const AClassName: string): string;
var
  I: Integer;
begin
  Result := '';
  for I := 2 to Length(AClassName) do
  begin
    if(UpCase(AClassName[I]) = AClassName[I]) and(Result <> '')
then
      Result := Result + ' ' + AClassName[I]
    else
      Result := Result + AClassName[I]
  end;
  Result := StringReplace(Result, ' Node', '', []);
end;
```

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.

Іншим не менш важливим етапом для перевірки цілісності та функціональності програмного додатку є етап завантаження попередньо збереженого представлення блок-схеми або поточна превірка коректності роботи програми. Для цього було реалізовано функції перевірки наявності та повноти заповнення кожного з вузлів грава, що характеризують окремі елементи візуалізації алгоритму за допомогою блок-схеми. Відповідний програмний код наведено нижче:

```
     class          function          TNodeProperties.Execute(Nodes:
TGraphObjectList): Boolean;
     begin
       Result := False;
       with Create(Application) do
       try
         N := Nodes;
         S := Nodes[0].Owner;
         ListRegistredNodeClasses;
         with TGraphNode(Nodes[0]) do
         begin
           case Alignment of
             taLeftJustify: rgAlignment.ItemIndex := 0;
             taCenter: rgAlignment.ItemIndex := 1;
             taRightJustify: rgAlignment.ItemIndex := 2;
           end;
```

В другій половині даної функції відувається додавання відповідних параметрів або їх корекція для покращення візуального відображення того чи іншого компоненту. Серед можливих параметрів є параметри кольору відображення, типу шрифтів тощо. Частина програмного коду наведена як:

```
           UpDownMargin.Position := Margin;
           NodeText.Lines.Text := Text;
           if Nodes.Count = 1 then
             NodeShape.ItemIndex                                    :=
NodeShape.Items.IndexOfObject(TObject(ClassType))
           else
             NodeShape.ItemIndex := -1;
           BodyColor.Color := Brush.Color;
           NodeBorderColor.Color := Pen.Color;
           FontDialog.Font := Font;
         end;
         if ShowModal = mrOK then
         begin
           ApplyChanges;
```

```
          Result := True;
        end;
      finally
        Free;
      end;
    end;
```

Іншою не менш важливою функцією роботи з програмою є модливість корекції та перегляду масиву вузлів графа. Для доступу до кожного з них та перевірки їх цілісності було реалізовано відповідну процедуру, код якої наведно нижче:

```
procedure TNodeProperties.ListRegistredNodeClasses;
var
  I: Integer;
  NodeClass: TGraphNodeClass;
begin
  for I := 0 to TSimpleGraph.NodeClassCount - 1 do
  begin
    NodeClass := TSimpleGraph.NodeClasses(I);

NodeShape.Items.AddObject(PrettyNodeClassName(NodeClass.ClassName)
       TObject(NodeClass));
  end;
end;
```

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 основні функціональні параметри роботи програмного додатку. А набір функцій, що пердбачено зберігати в даній структурі у повній мірі дозволяє забезпечити роботу програмного додатку.

3.3 Тестування модуля візуалізації блок-схем алгоритмів

Для проведення тестування розробленого програмного додатку використовувався персональний комп'ютер з наступними технічними параметрами (таблиця 3.1):

Таблиця 3.1 – Параметри персонального комп'ютера для тестування програмного додатку

Параметр	Значення
корпус	Zalman Z1 Black + Chieftec APS-550SB
HDD	1000GB
відеокарта	Asus PH-GTX1060-3G
ОЗУ	4Gb
процесор	AMD Ryzen 3 2200G BOX 120
материнська плата	Asus B350M-E 90
діагональ дисплея	21
роздільна здатність дисплея	1820 x 1180
тип матриці	IPS
частота оновлення	70 Гц
інтерфейси	HDMI
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.

Для коректної оцінки результатів тестування вець процес було вирішено проводити в декілька етапів та з використання алгоритмів різної стскладності. Причому як тип алгоритмів та і їхня кількість елементів значно відрізнялись, щоб зробити процес тестування максимально наближеним до робочих умов.

Результат робти з простими алгоритмами я яких присутня мінімальна кількість структурних елементів наведено на рисуноку 3.5.

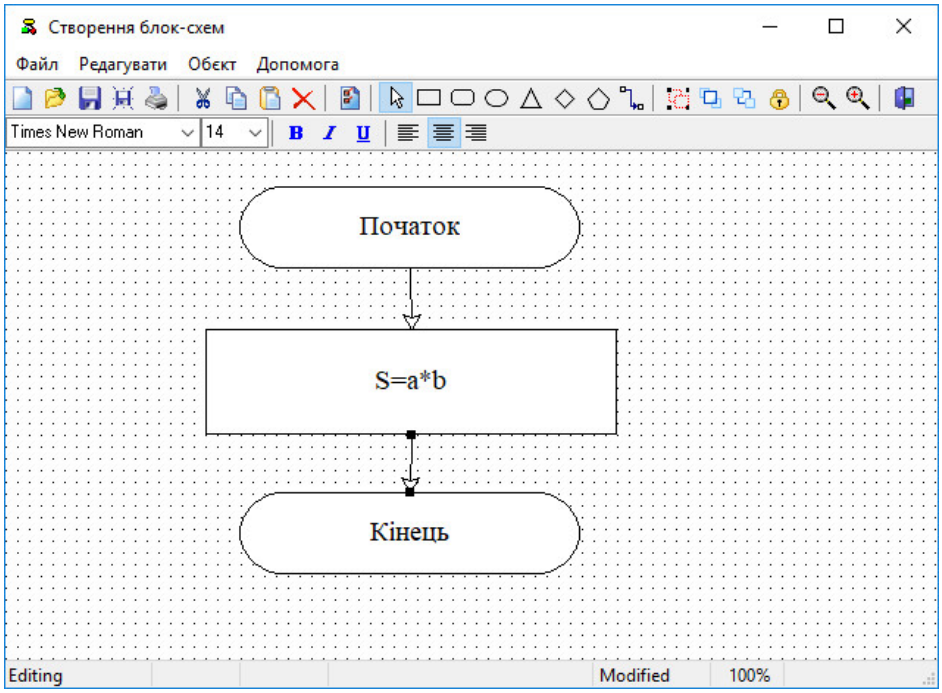


Рисунок 3.5 – Приклад створення простої блок-схеми

Як показали отримані результати обробки простих алгоритмів розроблений програмний додаток успішно виконав поставлене завдання з створення блок-схеми, 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.

Для наступного етапу тестування було обрано складніший підхід. В даному експерименті програмному додатку було запропоновано опрацювати велику кількість елементів, що були хаотично додані в блок-схему. При цьому

введення інформації відбувалосьз порушеннями правил формування та відображення блок-схем. А саме, елементам не додавались усі необхідні зв'язки або внутрішні параметри. Врезультаті проведеного експерименту було доведено, що розроблена програма дозволяє вносити в блок-схему хибну інформацію, проте завжди видає повідомлення про некоректність введених даних та надає довідкову інформацію, як це можна поправити. Результати тестування наведено на рисунку 3.6.

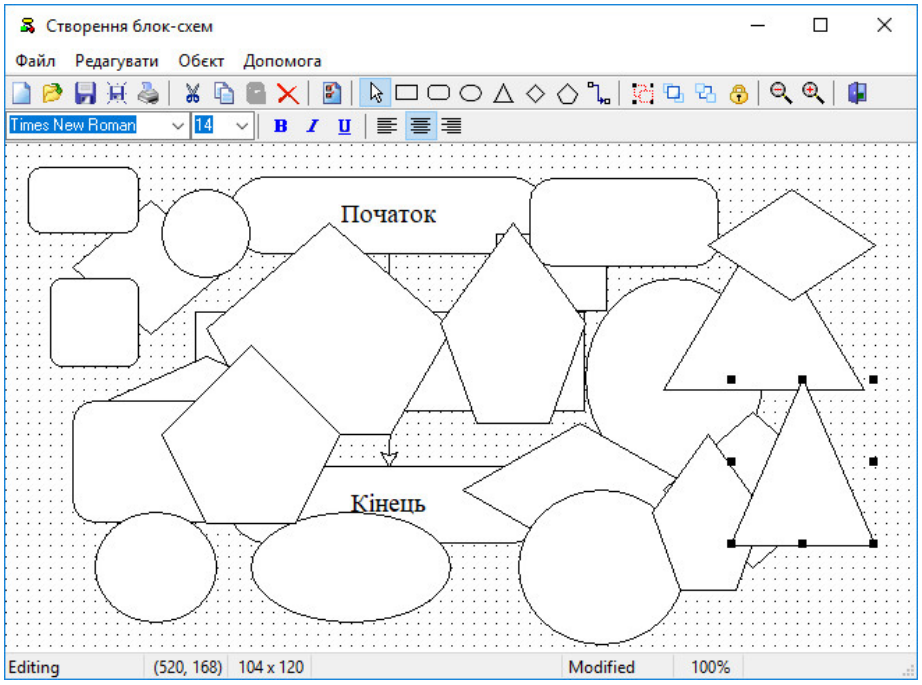


Рисунок 3.6 – Результати другого етапу тестування програмного додатку створення та редагування блок-схем алгоритмів

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 кваліфікаційної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності програмного додатку створення та візуалізації алгоритмів у вигляді блок-схем на основі використання механік теорії графів 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 додатку (K) на основі запропонованих алгоритмів та розробленій структурі, що враховують результати аналізу програм-аналогів 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, ст. викладач	7449
2	Консультант техніко-економічного розділу, ст. викладач	7449
3	Студент	1400

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;

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.

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	64,3	1028,8
2	Консультант техніко-економічного розділу, доцент	2	59,9	119,8
3	Студент	144	8,33	1199,52
Разом				2348,12

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:

$$B_Ф = \frac{20,5}{100} \cdot 2348,12 = 481,36 \text{ грн.}$$

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	1000	1000	100	1100
	Папір (формат А4)	уп	2	100	200	20	220
	Ручка кулькова	шт	2	10	20	2	22
	Олівець простий	шт	2	10	20	2	22
	Диски CD-R	шт	2	20	40	4	44
	Зошит, 96 арк	шт	1	50	50	5	55
	Тонер для принтера	уп	1	90	90	9	99
	Канцелярські маркери (синій, зелений)	шт	2	20	40	4	44
	Р а з о м						1606,00

Витрати на використання комп'ютерної техніки.

Витрати на використання комп'ютерної техніки($B_{КТ}$) включають витрати на амортизацію комп'ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп'ютером. За даними обчислювального центру ЗУНУ для комп'ютера типу IBM PC/ATX вартість години роботи становить 12 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 у вигяді звіту	60	7200
2	Оформлення техніко-економічного розділу	8	96
3	Оформлення ДП	12	144
Разом		80	960

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:

$$H = 1{,}5 \cdot 2348{,}12 = 3522{,}18 \text{ (грн)}.$$

Інші витрати.

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

$$I_B = 2348{,}12 \cdot 0{,}1 = 234{,}81 \text{ (грн).}$$

Витрати на розробку програмного забезпечення складають:

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

$$K_1 = 2348{,}12+481{,}36+1606{,}00+3522{,}18+234{,}81+960{,}00=8849{,}47 \text{ (грн)}.$$

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 12 грн., тому $K_2 = 12 \cdot 60 = 720$ грн.

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблювальної програмної системи 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 відповідних 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* = 6061 грн. Тоді:

$$c_1 = \frac{6061(1+0{,}57)}{22*8} = 56{,}64 \text{ грн/год}$$

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	56,64	56,64
	Аналог		
1	1,5	56,64	84,95

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2П} = t * S_{МГ}$$

Де *t* – витрати машинного часу для реалізації рішення (аналогу), год.;

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

$E_{2П}$ = 1 * 12 = 12 грн.; $E_{2А}$=1,5 * 12 = 18 грн.

$E_{П}$=56,64+12=68,64 грн. ; $E_{А}$=84,95+12 = 96,95 грн.

$B_{ЕП}$= 68,64 * 252=17297,28 грн. ; $B_{ЕА}$=96,95 * 252=24431,4 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.

$$H_B = 0{,}7 * B_{ОП} = 0{,}7 * (c_1 * 168), \quad (4.7)$$

де H_B – накладні витрати.

$$H_B = 0{,}7 * 9515{,}52 = 6660{,}86 \text{ грн.}$$

Складання кошторису витрат та визначення собівартості. Результати проведених розрахунків зведемо у таблицю 4.6.

Таблиця 4.6 – Кошторис витрат ($B_{КС}$)

№ п/п	Найменування витрат	Сума витрат, грн.
1	Витрати на оплату праці ($B_{ОП}$)	2348,12
2	Відрахування у спеціальні державні фонди ($B_Ф$)	481,36
3	Витрати на матеріали та комплектуючі (B_M)	1606,00
4	Накладні витрати на розробку (H)	3522,18
5	Інші витрати (I_B)	234,81
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту (K_2)	720
7	Накладні витрати експлуатацію (H_B)	6660,86
8	Річні експлуатаційні витрати (B_{EA})	24431,4
Разом		40004,73

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 20% за погодженням з керівником).

$$Ц_Д = 40004,73 \cdot (1+0,2) = 48005,68 \text{ грн.}$$

4.3 Визначення економічної ефективності та терміну окупності

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

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

де $П = Ц_Д - B_{КС}$ – прибуток, грн.;

$B_{КС}$ – кошторисна вартість, грн..

$$E_ф = 8000,95 \text{ грн.} / 40004,73 \text{ грн.} = 0,2.$$

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

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

Тобто: $T_P = 1/0{,}2 = 5$р.

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.	40004,73
2.	Плановий прибуток, грн.	8000,95
3.	Ціна, грн.	48005,68
4.	Економічна ефективність	0,2
5.	Термін окупності, рік	5

Враховуючи основі економічні показники з таблиці 4.7, можна зробити висновок, що при економічній ефективності 0,2 та терміні окупності – 5 роки проводити роботи по впровадженню даного програмного додатку є доцільним та економічно вигідним.

ВИСНОВКИ

На основі аналізу сучасних алгоритмів та технологій візуалізації даних з використаннят елементів теорії графів та проведених експериментів, можна зробити наступні висновки:

1) Досліджено та проведено класифікацію груп алгоритмів, на основі існуючих стандартів, що надало можливість виділити основні принципи при візуалізації алгоритмів за допомогою блок-схем.

2) Проаналізовано можливості теорії графів для відображення блок-схем алгоритмів у вигляді дерев, що дозволило при пороектування програмної системи закласти принципи взаємодії для окремих елементів блок-схем.

3) Проведено аналіз сучасних програмних додатків побудови та відображення блок-схем, що дозволило виділити їх основні функції та структурні одиниці.

4) Досліджено функції та структури цифрової бібліотеки tsimplegrap, що дозволило використати її при реалізації системи візуалізації алгоритмів за допомогою блок-схем на основі графів;

5) Спроектовано та здійснено моделювання структури додатку створення та відображення блок-схем алгоритмів, що дозволило програмно його реалізувати;

6) Програмно реалізовано систему створення та графічного відображення алгоритмів у вигляді-блок, проведено її тестування та порівняльний аналіз з програмами-аналогами, що дозволило проілюструвати високий потенціал розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тимчук О.Ю., Рудницький В.І. Оцінка якості результатів обробки цифрових зображень на основі об’єктивних критеріїв. Збірник тез IV Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі»., Тернопіль, 02 червня 2021 р. с. 40.

2. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня “Бакалавр” спеціальності 123 «Комп’ютерна інженерія» галузі знань 12 Інформаційні технології / О.М. Березький, Л.О.Дубчак, Г.М. Мельник, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2020. 60с.

3. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп’ютерних систем”/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

4. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

5. Бартлетт Н. Программирование на C++ Путеводитель. The Coriolis Group,Inc.,1996, Издательство НИПФ "ДиаСофт Лтд.",2016. 116с.

6. Вебер Дж. Технология C++в подлиннике. QUE Corporation, 2016, "BHV-Санкт-Петербург",2017. 256с.

7. Волш А. И. Основы программирования на C++для World Wide Web. IDG Books Worldwide,Inc.,1996, Издательство "Диалектика",2016. 458с.

8. Марков А. С. «Базы данных. Введение в теорию и методологию. Финансы и статистика». 2016. С.24-35.

9. Абрамов С. А. Задачи по программированию. М.: Наука, 2018. 256с.

10. Березин Б.И., Начальный курс Delphi. М.: ДИАЛОГ-МИФИ, 2016. 331с.

11. Бондарев В.М. Основы программирования. Харьков: Фолио, Ростов н/Д: Феникс, 2017. 446с.

12. Вирт Н. Алгоритмы и структуры данных. М.: Мир, 2019. 345с.

13. Гладков В. П. Задачи по информатике на вступительном экзамене в вуз и их решения: Учебное пособие. Пермь: Перм. техн. ун-т, 2014. 516с.

14. Грогоно П. Программирование на языке Delphi. М.: Мир, 2012. 216с.

15. Дагене В.А. 100 задач по программированию. М.: Просвещение, 2013. 106с.

16. Джамса К. Библиотека программиста Java. Jamsa Press, 2016, ООО "Попурри", 2016. 656с.

17. Марков А. С. «Базы данных. Введение в теорию и методологію. Финансы и статистика». 2016. Р. 24-35.

18. Заварыкин В.М. Основы информатики и вычислительной техники. М.: Просвещение, 2019. 556с.

19. Касаткин В. Н. Информация. Алгоритмы и примеры. ЭВМ. М.: Просвещение, 2011. 219с.

20. Кен А. Язык программирования Delphi. Addison-Wesley Longman,U.S.A.,1996, Издательство "Питер-Пресс", 2017. 378с.

21. Керниган Б. Язык программирования Delphi. Пер. с англ. М.: Финансы и статистика, 2012. 391с.

22. Ляхович В.Ф. Руководство к решению задач по основам информатики и вычислительной техники. М.: Высшая школа, 2014. 127с.

23. Мейнджер Дж. Delphi Основы программирования. McGraw-Hill,Inc.,1996, Издательская группа BHV, Киев, 2017. 346с.

24. Миков А. И. Информатика. Введение в компьютерные науки. Пермь: Изд-во ПГУ, 2018. 442с.

25. Могилев А. В. Информатика: Учеб. пособие для студ. пед. Вузов. М.: Изд. центр «Академия», 2019. 629с.

26. Нотон П. JAVA:Справ.руководство. М.:БИНОМ:Восточ.Кн.Компания, 2016: Восточ.Кн.Компания. 447с.

27. Нотон П. Полный справочник по Java. McGraw-Hill,1997, Издательство "Диалектика",2017. 556с.

28. Ренеган Э.Дж. 1001 адрес WEB для программистов :Новейший путеводитель программиста по ресурсам World Wide Web:Пер.. Минск:Попурри, 2017. 512с.ил.

29. Родли Дж. Создание Java-апплетов. The Coriolis Group,Inc.,1996, Издательство НИПФ "ДиаСофт Лтд.", 2016. 466с.

30. Секреты программирования для Internet на Java. Ventana Press, Ventana Communications Group, U.S.A., 2016, Издательство "Питер Пресс", 2017. 396с.

31. Семакина И. Г. Информатика. Задачник-практикум: В 2 т.. М.: Лаборатория Базовых Знаний, 2019. 476с.

32. Сокольский М.В. Все об Intranet и Internet. М.:Элиот, 2018. 254с.

33. Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. М.: Мир, 2011. 56с.

34. Тюрин Ю.Н. Анализ данных на компьютере. М.: ИНФРА-М, Финансы и статистика, 2015. 384с.

35. Флэнэген Д. Java in a Nutshell. O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 2018. 473с.

36. Чен М.С. Программирование на С++:1001 совет:Наиболее полное руководство по Java и Visual J++ :Пер.с англ. Минск:Попурри, 2017. 640с.ил.

37. Эферган М. С++: справочник. QUE Corporation, 2017, Издательство "Питер Ком", 1998. 256с.

38. G. Yang. «Human face detection in a complex background. Pattern Recognition », 27 (1): 2014. P.53-63.

39. Kotropoulos C. «Acoustics, Speech, and Signal Processing», 2017. ICASSP-97, 2017. IEEE International Conference on pp.2537-2540 v. 4

40. Leung TK. «Finding Faces in Cluttered Scenes Using Random Labeled Graph Matching» 2015.p P.83-95.

41. Yow KC. Feature-based human face detection. Image and vision computing 15 (9), 2017. P.713-735.

42. Sinha, P. Perceiving and Recognizing threedimensional forms. PhD thesis, Massachusetts Institue of Technology, 2016. 278p.

43. Lanitis, A «Image Anal. Classifying variable objects using a flexible shape model »Image Processing and its Applications, 2015., P.70-74.

44. Viola P. «Rapid Object Detection using a Boosted Cascade of Simple Features», proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2011., vol. 1, 518p.

45. Jones MJ. «Robust real-time face detection», International Journal of Computer Vision, vol. 57, no. 2, 2014., P.137-154.

46. Buchatskiy AN. «Selection of the Optimal Color Space for Reducing False Positives Rate in the Viola-Jones Method», Актуальні проблеми інфотелекомунікацій в науці та освіті, II Міжнародна науково-технічна та науково-методична конференція. Санкт-Петербург, 2013.

47. Ethan R. ORB: an efficient alternative to SIFT or SURF. Computer Vision (ICCV), IEEE International Conference on. IEEE, 2011. P. 2564–2571.

48. Stefan L. BRISK: Binary Robust Invariant Scalable Keypoints. Computer Vision (ICCV), 2011. P. 2548–2555.

49. Pablo F. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In British Machine Vision Conference (BMVC), 2013.

50. Martin A. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. Of the ACM24: 2001. P.381–395,.

51. Патин М.В. Сравнительный анализ методов поиска особых точек и дескрипторов при группировке изображений по схожему содержанию. Молодой ученый. 2016. №11. С. 214-221.

52. Гонсалес Р. Цифровая обработка изображений в среде MATLAB //М.: Техносфера. 2016. 616с.