МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський державний університет
імені Володимира Винниченка

Факультет математики, природничих наук та технологій
Кафедра математики та цифрових технологій

Кваліфікаційна робота на правах рукопису

Малчевський Валерій Миколайович

КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти
на тему «**Професійна підготовка фахівців цифрових технологій до викладання освітнього модуля «Основи алгоритмізації»**

Виконав: студент II курсу, групи ЦТ23М
факультету математики, природничих наук та технологій
спеціальності: 015 Професійна освіта (Цифрові технології)
освітня програма: Професійна освіта (Цифрові технології)
форма навчання денна
Малчевський В.М.
Керівник: Щирбул О.М., кандидат педагогічних наук, старший викладач кафедри інформатики, програмування, штучного інтелекту та технологічної освіти
Рецензент: Гринь Д.В. кандидат технічних наук, доцент, адміністратор систем відеоінформаційних технологій "ПАТ "НВП" Радій"

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ В.М. Малчевський

Кропивницький, 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський державний університет
імені Володимира Винниченка

Кафедра математики та цифрових технологій

До захисту допустити
Зав. кафедри
_____/О.М. Трифонова/
«___» листопада 2024 р.

Малчевський Валерій Миколайович

КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти
на тему «**Професійна підготовка фахівців цифрових технологій до викладання освітнього модуля «Основи алгоритмізації»**
з професійно-орієнтованих дисциплін та методики їхнього навчання

Виконав: студент ІІ курсу, групи ЦТ23М
факультету математики, природничих наук та технологій
спеціальності: 015 Професійна освіта (Цифрові технології)
освітня програма: Професійна освіта (Цифрові технології)
форма навчання: денна
науковий керівник: Щирбул Олександр Миколайович, кандидат педагогічних наук, старший викладач кафедри інформатики, програмування, штучного інтелекту та технологічної освіти

Кваліфікаційна робота захищена з оцінкою
«_____» балів,
за шкалою ECTS _____,
національною шкалою _____
Секретар ЕК _____/_____/
«____» _____2024р.

Кропивницький, 2024

АНОТАЦІЯ

Малчевський В.М. **Професійна підготовка фахівців цифрових технологій до викладання освітнього модуля «Основи алгоритмізації»** – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня магістра за спеціальністю 015 Професійна освіта (Цифрові технології). – Центральноукраїнський державний університет імені Володимира Винниченка, Кропивницький, 2024.

Актуальність обраної теми полягає в тому, що алгоритмізація є основою програмування, котре, на сьогодні широко використовується в багатьох галузях науки, техніки, освіти і т. п. Відповідно, оволодіння основами алгоритмізації створює підґрунтя для подальшого освоєння різних інформаційних, комп'ютерних технологій, що є важливим у підготовці сучасних, конкурентоспроможних фахівців.

Мета дослідження полягає в досліджені методики вивчення основ алгоритмізації.

Об'єкт дослідження –освітній процес у закладах професійної освіти.

Предмет дослідження –підходи, методи, техніки та інструменти, які використовуються для ефективного навчання та засвоєння основ алгоритмізації.

Елементи наукової новизни результатів дослідження:

- *проведено аналіз та порівняння* різних методик вивчення основ алгоритмізації; представлено огляд і систематизацію різних підходів до вивчення основ алгоритмізації, які раніше не були об'єктом комплексного аналізу;
- *визначено* критерії для вибору найбільш ефективних методик вивчення основ алгоритмізації залежно від особливостей аудиторії та доступних ресурсів, враховуючи рівень підготовки студентів, їхні попередні знання та досвід, а також можливості навчального закладу.

Практичне значення роботи:

- *Покращення навчання програмування.* Розроблені методики, рекомендації та інноваційні підходи можуть сприяти ефективнішому засвоєнню матеріалу студентами та підвищити їх рівень комп'ютерної грамотності;
- *Забезпечення розвитку комп'ютерних навичок.* Розроблені методики можуть допомогти студентам зрозуміти принципи створення алгоритмів та їх практичну реалізацію;
- *Популяризація STEM-освіти:* Розроблені методики можуть сприяти популяризації STEM-освіти (наука, технологія, інженерія та математика).

Ключові слова: алгоритм, алгоритмізація, професійна освіта, методика підготовки.

ANNOTATION

Malchevsky V.M. **Professional training of digital technology specialists for teaching the educational module «Fundamentals of Algorithmization»** – Qualification work in the form of a manuscript. Qualification work for obtaining a master's degree in the specialty 015 Professional Education (Digital Technologies). – Volodymyr Vynnychenko Central Ukrainian State University, Kropyvnytskyi, 2024.

The relevance of the chosen topic lies in the fact that algorithmization is the basis of programming, which is widely used today in many fields of science, technology, education, etc. Accordingly, mastering the basics of algorithmization creates a basis for further mastering various information and computer technologies, which is important in the training of modern, competitive specialists.

The purpose of the study is to investigate the methodology for studying the basics of algorithmization.

The object of the study is the educational process in vocational education institutions.

The subject of the study is approaches, methods, techniques and tools used for effective learning and mastering the basics of algorithmization.

Elements of scientific novelty of the research results:

• an analysis and comparison of different methodologies for studying the basics of algorithmization is carried out; an overview and systematization of different approaches to studying the basics of algorithmization, which have not previously been the object of a comprehensive analysis, are presented;

• criteria for choosing the most effective methodologies for studying the basics of algorithmization are determined depending on the characteristics of the audience and available resources, taking into account the level of training of students, their previous knowledge and experience, as well as the capabilities of the educational institution.

Practical significance of the work:

• Improving programming teaching. The developed methodologies, recommendations and innovative approaches can contribute to more effective learning of the material by students and increase their level of computer literacy;

• Ensuring the development of computer skills. The developed methodologies can help students understand the principles of creating algorithms and their practical implementation; • Popularization of STEM education: The developed methods can contribute to the popularization of STEM education (science, technology, engineering and mathematics).

Key words: algorithm, algorithmization, professional education, training methodology.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Тема дослідження про методику вивчення основ алгоритмізації є актуальною незалежно від часу, оскільки алгоритми є фундаментальною частиною комп'ютерної науки та програмування. Основи алгоритмізації є ключовими для розвитку навичок у програмуванні та розуміння різноманітних проблем.

Основи алгоритмізації включають такі поняття, як послідовність дій, умовні оператори, цикли, змінні, функції та інші. Ці концепції є основою для створення ефективних програм та розв'язання реальних проблем. Особливо у сучасному світі, де програмування стає все більш важливим навичкою, вивчення методики алгоритмізації стає актуальним для всіх, хто бажає розуміти та застосовувати комп'ютерні технології.

Оскільки технології швидко розвиваються, методика вивчення основ алгоритмізації також може змінюватися та адаптуватися до нових потреб. Наприклад, з'являються нові мови програмування, інструменти та підходи, які можуть спростити процес вивчення алгоритмізації або зробити його більш доступним. Також, в контексті швидкого розвитку штучного інтелекту та машинного навчання, вивчення алгоритмізації стає ще більш важливим для розуміння та створення складних алгоритмів.

Отже, актуальність теми дослідження про методику вивчення основ алгоритмізації не зменшується з часом, оскільки вона відображає основні принципи та навички, необхідні для розуміння тастворення алгоритмів. Зростаюча цифрова трансформація суспільства та швидкий розвиток інформаційних технологій підкреслюють значення алгоритмізації як ключової компетенції. Основи алгоритмізації є фундаментом для розвитку навичок програмування, розуміння роботи комп'ютерних систем, аналізу проблем та розробки ефективних рішень.

Крім того, постійний прогрес у галузі інформаційних технологій призводить до появи нових методів та підходів у сфері алгоритмізації. Дослідження методики вивчення основ алгоритмізації дозволяє оновлювати та

адаптувати навчальні програми та підходи до нових вимог і можливостей. Вона сприяє розробці ефективних та інноваційних стратегій навчання, що відповідають сучасним викликам та потребам.

Таким чином, актуальність дослідження про методику вивчення основ алгоритмізації підтверджується необхідністю розуміння та використання алгоритмічного мислення у різних сферах життя та професійної діяльності. Навички алгоритмізації є важливим інструментом для подолання сучасних викликів та досягнення успіху в цифровому світі.

Зв'язок роботи з науковими програмами, планами та темами. Напрям дослідження визначено відповідно до тематичного плану наукових досліджень кафедри технологічної та професійної освіти Центральноукраїнського державного університету імені Володимира Винниченка «Сучасні освітні технології у підготовці фахівців технологічної та професійної освіти» (держ. реєстр. № 0123U100957, з 2022 р.)», Лабораторії дидактики фізики, технологій та професійної освіти Інституту педагогіки Національної академії педагогічних наук України в Центральноукраїнському державному університеті імені Володимира Винниченка і є складовою тем: «Цифровізація освітнього середовища та STEM-технології (держ. реєстр. № 0122U201725, з 2022 р.)».

Об'єкт дослідження –освітній процес у закладах професійної освіти.

Предмет дослідження –підходи, методи, техніки та інструменти, які використовуються для ефективного навчання та засвоєння основ алгоритмізації.

Мета дослідження полягає в досліджені методики вивчення основ алгоритмізації.

Завдання дослідження:

1) розкрити основні принципи алгоритмізації;
2) дослідити види алгоритмів та їх особливості;
3) розглянути методику викладання основ алгоритмізації;
4) проаналізувати вибір оптимальної методики в залежності від цільової аудиторії;

5) надати приклади практичних завдань та вправ для вивчення основ алгоритмізації;

6) дослідити інструменти та ресурси для вивчення алгоритмізації.

Методи дослідження: Для досягнення мети, розв’язання поставлених завдань і реалізації загальної концепції використано наступні методи:

- Спостереження: Спостереження за процесом вивчення основ алгоритмізації в класі або навчальній групі. Дослідник може зафіксувати, які методи, матеріали та активності використовуються вчителем або тренером, як взаємодіють учні з матеріалами та які результати досягаються.

- Контентний аналіз: Аналіз навчальних матеріалів, підручників, онлайн-курсів тощо, що використовуються для вивчення основ алгоритмізації. Оцінка методів, прикладів, завдань, які надаються у цих матеріалах і їх відповідність основним принципам алгоритмізації.

Елементи наукової новизни результатів дослідження:

- *проведено аналіз та порівняння* різних методик вивчення основ алгоритмізації; представлено огляд і систематизацію різних підходів до вивчення основ алгоритмізації, які раніше не були об'єктом комплексного аналізу;
- *визначенно* критерії для вибору найбільш ефективних методик вивчення основ алгоритмізації залежно від особливостей аудиторії та доступних ресурсів, враховуючи рівень підготовки студентів, їхні попередні знання та досвід, а також можливості навчального закладу.

Практичне значення роботи:

- *Покращення навчання програмування.* Розроблені методики, рекомендації та інноваційні підходи можуть сприяти ефективнішому засвоєнню матеріалу студентами та підвищити їх рівень комп'ютерної грамотності;
- *Забезпечення розвитку комп'ютерних навичок.* Розроблені методики можуть допомогти студентам зрозуміти принципи створення алгоритмів та їх практичну реалізацію;
- *Популяризація STEM-освіти:* Розроблені методики можуть сприяти популяризації STEM-освіти (наука, технологія, інженерія та математика).

Апробація результатів дослідження. Основні положення та результати дослідження апробувалися під час проходження виробничої (педагогічної) практики в Кропивницькому фаховому будівельному коледжі (23.09.2024 – 04.10.2024) та в Центральноукраїнського державного університет імені Володимира Винниченка на кафедрі математики та цифрових технологій (07.10.2024 – 18.10.2024). Також за результати магістерського дослідження опубліковані в статті «Особливості методичної підготовки фахівців цифрових технологій до викладання освітнього модуля «Основи алгоритмізації» у Всеукраїнському збірнику наукових праць студентів, аспірантів, викладачів і вчителів ЗЗСО. Випуск №12.

Структура роботи. Кваліфікаційна робота складається з вступу, 4 розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел (31 найменування), 11 рисунків. Повний обсяг роботи 74 сторінки, основний текст становить 69 сторінок.

РОЗДІЛ 1. ВИЗНАЧЕННЯ ТЕРМІНІВ: АЛГОРИТМ, ПРОГРАМА, ПСЕВДОКОД ТОЩО

1.1. Основні принципи алгоритмізації

Основні принципи алгоритмізації включають:

Дискретність: Алгоритм розбивається на окремі кроки або операції, які можуть бути виконані послідовно. Це означає, що алгоритм має конкретну послідовність кроків і не містить розривів або неоднозначностей.

Алгоритм розбивається на окремі дискретні кроки або операції, які виконуються послідовно в певному порядку. Кожен крок в алгоритмі має конкретну дію або операцію, яку виконавець повинен виконати перед переходом до наступного кроку. Важливо, що кожен крок алгоритму є окремим і чітко визначеним, не містить розривів, пропусків або неоднозначностей. Це забезпечує чітку та однозначну послідовність дій при виконанні алгоритму.

Визначеність: Кожна операція або крок алгоритму має чітко визначене і однозначне значення та результат. Це дозволяє виконавцю алгоритму однозначно розуміти, що потрібно робити на кожному кроці.

Кожна операція або крок алгоритму має чітко визначене і однозначне значення, а також очікуваний результат. Це означає, що виконавець алгоритму може однозначно розуміти, які дії потрібно виконувати на кожному кроці і який результат буде отриманий після виконання цих дій. Визначеність забезпечує ясність та універсальність алгоритму, що дозволяє різним виконавцям отримувати однакові результати при правильному виконанні кожного кроку алгоритму.

Детермінованість: Алгоритм повинен бути детермінованим, тобто при однакових початкових умовах він завжди дає один і той самий результат. Кожна дія алгоритму має бути визначена чітко і не залежати від випадкових факторів [14, с.25].

Детермінованість означає, що алгоритм завжди буде давати один і той самий результат при одних і тих самих початкових умовах. Кожна дія алгоритму

повинна бути визначена чітко і не залежати від випадкових факторів або невизначеностей.

Це означає, що при виконанні алгоритму з однаковими вхідними даними та вихідними умовами, результат завжди буде однаковим. Немає місця для випадковості або непередбачуваності у добре сконструйованому детермінованому алгоритмі. Це робить його прогнозованим та надійним виконавцем певної задачі.

Детермінованість особливо важлива в області програмування, де алгоритми виконуються комп'ютером. Це забезпечує передбачуваність результатів і дозволяє розробникам створювати надійні програми, які виконують потрібні операції без випадковості чи незрозумілих варіацій у вихідних даних.

Коректність: Алгоритм повинен бути коректним, тобто він повинен розв'язувати задачу або виконувати поставлену задачу правильно і точно згідно з поставленими вимогами і умовами.

Коректність означає, що алгоритм повинен правильно виконувати поставлену задачу згідно з вимогами і умовами, що були поставлені.

Це означає, що алгоритм має повністю розв'язати поставлену задачу і повернути очікуваний результат. Виконавець алгоритму повинен мати впевненість, що при правильному застосуванні алгоритму буде отримано правильний результат. Коректність алгоритму залежить від його правильного спроектування, врахування усіх умов та вимог, а також від його вірного виконання[20, с.24].

При розробці програм і виконанні завдань важливо мати коректні алгоритми, оскільки некоректні алгоритми можуть призводити до неправильних результатів, помилок або непередбачуваних наслідків. Дотримання принципу коректності допомагає забезпечити, що алгоритми працюють належним чином і вирішують поставлені задачі згідно з вимогами.

Кінцевість: Алгоритм повинен завершуватися за скінченну кількість кроків. Це означає, що виконавець алгоритму зможе досягти кінцевого

результату після скінченної кількості кроків без нескінченного зациклення або нескінченної обробки даних.

Кінцевість означає, що алгоритм повинен завершитися після скінченної кількості кроків. Це означає, що виконавець алгоритму зможе досягти кінцевого результату за скінченну кількість операцій.

Це дуже важливий принцип, оскільки дозволяє уникнути нескінченого зациклення або нескінченної обробки даних, що можуть виникнути в нескінченних алгоритмах. Кінцевість дозволяє забезпечити коректне завершення алгоритму та отримання очікуваного результату.

У програмуванні цей принцип особливо важливий, оскільки він допомагає уникнути неправильної поведінки програм, які можуть нескінченно виконуватися або застрягати в безкінечних петлях. Кінцевість алгоритму дозволяє забезпечити ефективне виконання завдання та раціональне використання ресурсів.

Універсальність: Алгоритм може бути застосований до різних вхідних даних або ситуацій. Це означає, що алгоритм має загальні принципи, які можуть бути застосовані в різних контекстах і на різних наборах даних.

Універсальність означає, що алгоритм може бути застосований до різних вхідних даних або ситуацій. Він базується на загальних принципах та логіці, які можуть бути використані в різних контекстах і на різних наборах даних.

Це означає, що алгоритм має гнучкість і універсальність у використанні. Він може бути застосований до різних ситуацій і вхідних даних, що робить його потужним і придатним для рішення різноманітних задач.

Універсальність є однією з ключових переваг алгоритмів, оскільки вона дозволяє знаходити загальні рішення для класу подібних проблем. Застосування універсальних алгоритмів дозволяє економити час і зусилля, оскільки їх можна використовувати у багатьох випадках, замість створення нового алгоритму для кожної конкретної задачі [14, с.22].

1.2. Види алгоритмів та їх особливості

Існує кілька видів алгоритмів, кожен з яких має свої особливості і використовується для розв'язання конкретного типу задач. Основні види алгоритмів включають:

Послідовні алгоритми: Це найпростіший тип алгоритмів, в яких кроки виконуються послідовно один за одним. Кожен крок виконує певну операцію або обчислення, і виконання переходить до наступного кроку після завершення попереднього. Послідовні алгоритми легко розуміти і реалізовувати, але можуть бути менш ефективними для складних задач.

Вони є найпростішим типом алгоритмів, де кроки виконуються послідовно один за одним у визначеному порядку. Кожен крок алгоритму виконує певну операцію або обчислення і переходить до наступного кроку після завершення попереднього.

Основна особливість послідовних алгоритмів полягає в їх простоті і зрозумілості. Вони легко розуміються і можуть бути реалізовані без особливих складнощів. Це робить їх особливо корисними для простих задач і початкового розуміння алгоритмічного підходу[11, с.27].

Однак, послідовні алгоритми можуть бути менш ефективними для складних задач або випадків, коли потрібно вирішувати проблеми великого розміру або зі значним обсягом даних. При таких ситуаціях послідовні алгоритми можуть вимагати значних обчислювальних ресурсів та тривалого часу виконання. Для більш складних задач часто використовуються інші типи алгоритмів, такі як паралельні алгоритми або алгоритми з використанням структур даних для оптимізації швидкості виконання.

В цілому, послідовні алгоритми є важливою основою для розуміння алгоритмічного підходу та розв'язання простих задач. Вони використовуються в багатьох областях, включаючи програмування, математику, науку і багато інших.

Умовні алгоритми: Ці алгоритми включають умовні оператори, які дозволяють виконувати певні кроки залежно від умови або вхідних даних.

Умовні алгоритми зазвичай використовуються для прийняття рішень на основі певних умов або критеріїв. Наприклад, якщо певна умова виконується, виконується одна група кроків, а якщо ні, то виконується інша група кроків.

Умовні алгоритми включають в себе умовні оператори, які дозволяють виконувати певні кроки або групи кроків залежно від умови або вхідних даних. Умови можуть бути виражені у вигляді логічних умов, таких як рівність, нерівність, більше, менше, тощо.

Найпоширеніший вид умовного оператора - це "if-else" (якщо-інакше). У цьому випадку, якщо певна умова виконується, виконуються певні кроки, а якщо умова не виконується, виконуються інші кроки. Наприклад, можна мати умовний оператор, який перевіряє, чи число більше за 10. Якщо це так, то виконується одна дія, а якщо ні, то виконується інша дія.

Умовні алгоритми дозволяють приймати рішення на основі певних умов або критеріїв. Вони надають можливість змінювати поведінку алгоритму в залежності від зовнішніх умов або вхідних даних. Це дозволяє алгоритму бути більш гнучким і адаптивним до різних ситуацій.

Умовні алгоритми широко застосовуються в програмуванні, логіці, математиці та інших областях, де необхідно приймати рішення на основі умовних перевірок. Вони дозволяють створювати більш складні та гнучкі алгоритми, які можуть вирішувати різноманітні завдання залежно від умов і вхідних даних.

Циклічні алгоритми: Ці алгоритми дозволяють повторювати певні кроки або операції декілька разів. Вони використовують циклічні конструкції, такі як цикл for, цикл while або do-while, для управління повторенням. Циклічні алгоритми корисні, коли потрібно виконувати певні дії багатократно, наприклад, для обробки масиву даних або виконання ітераційних процесів.

Циклічні алгоритми використовуються для повторення певних кроків або операцій декілька разів. Вони дозволяють виконувати однакові дії багатократно, що дуже корисно для обробки повторюваних завдань або даних.

Найпоширеніші циклічні конструкції в програмуванні - це цикл for, цикл while і цикл do-while [9, с.31].

Цикл for виконується певну кількість разів і використовується, коли заздалегідь відома кількість повторень. Він має початкову умову, умову продовження і крок ітерації. Наприклад, можна використовувати цикл for для перебору елементів масиву.

Цикл while виконується, поки задана умова є істинною. Умова перевіряється перед кожною ітерацією, і якщо вона задовольняється, виконання циклу продовжується. Наприклад, цикл while може використовуватися для зчитування даних з файлу до тих пір, поки вони доступні.

Цикл do-while схожий на цикл while, але умова перевіряється після кожної ітерації. Це означає, що цикл завжди виконується принаймні один раз. Використовуйте цей цикл, коли потрібно гарантувати, що блок коду буде виконаний принаймні один раз, а потім умова буде перевірена для продовження.

Циклічні алгоритми дозволяють ефективно обробляти повторювані операції та даних, зменшують кількість дублюючого коду і дозволяють робити ітерації над даними або виконувати дії стільки разів, скільки потрібно. Вони широко використовуються в

Вони широко використовуються впрограмуванні і комп'ютерних науках для розв'язання різноманітних завдань. Деякі основні застосування циклічних алгоритмів включають:

Обробка масивів: Циклічні алгоритми дозволяють ітеруватися по елементах масиву та виконувати потрібні операції над ними. Наприклад, можна використовувати цикл для сумування елементів масиву, пошуку максимального або мінімального значення, сортування тощо.

Читання та обробка файлів: При роботі з файлами, циклічні алгоритми дозволяють почергово читати записи з файлу та виконувати потрібні дії з кожним записом. Наприклад, можна читати дані з файлу рядок за рядком та виконувати обробку або аналіз цих даних.

Ітеративні процеси: У деяких випадках, коли потрібно виконати певну дію кілька разів зі зміненням параметрів, можна використовувати цикл. Наприклад, обчислення математичних функцій, розв'язання числових рівнянь, виконання ітераційних методів.

Взаємодія з користувачем: Циклічні алгоритми дозволяють створювати інтерактивні програми, які продовжують взаємодію з користувачем, поки не буде введена певна команда або умова. Наприклад, можна створити цикл, який запитує користувача ввести дані, обробляє їх та продовжує запитувати нові дані, поки користувач не введе вказану команду для виходу.

Циклічні алгоритми дозволяють зробити програми більш гнучкими, ефективними та скорочують час та зусилля, необхідні для обробки повторюваних операцій.

Рекурсивні алгоритми використовують принцип рекурсії, коли задача розкладається на менші аналогічні підзадачі. У рекурсивному алгоритмі, задача вирішується шляхом виклику самого себе для обробки меншого випадку цієї ж задачі. Рекурсія може продовжуватися до досягнення базового випадку, де задача стає настільки простою, що може бути вирішена без нових рекурсивних викликів[14, с.25].

Основні етапи рекурсивного алгоритму включають:

Визначення базового випадку: Це випадок, коли задача стає настільки простою, що може бути вирішена без рекурсивних викликів. Базовий випадок є важливим для того, щоб рекурсивний алгоритм завершував свою роботу.

Визначення рекурсивного випадку: Це випадок, коли задача розбивається на менші підзадачі, аліментуючи себе самого. У цьому випадку алгоритм викликає сам себе для обробки менших випадків тієї ж задачі.

Забезпечення збіжності: Для того, щоб рекурсивний алгоритм був коректним і завершував свою роботу, необхідно забезпечити збіжність. Це означає, що рекурсивний виклик повинен наближати задачу до базового випадку при кожному виклику, і рекурсія повинна завершитися після скінченної кількості викликів.

Рекурсивні алгоритми зручно використовувати для задач, які мають структуру подібних підзадач або коли вирішення задачі вимагає послідовного розкладання на менші підзадачі.

Основні види алгоритмів включають рекурсивні алгоритми, послідовні алгоритми, умовні алгоритми і циклічні алгоритми.

Рекурсивні алгоритми використовуються для розкладання задачі на менші аналогічні підзадачі та виклику самого себе для їх вирішення.

Послідовні алгоритми є найпростішими і виконують кроки послідовно один за одним.

Умовні алгоритми включають умовні оператори і дозволяють виконувати певні дії залежно від умови або вхідних даних.

Циклічні алгоритми використовуються для повторення певних дій або операцій декілька разів з використанням циклічних конструкцій[8, с.19].

Кожен вид алгоритму має свої особливості і застосування. Розуміння цих видів алгоритмів дозволяє розробляти ефективні і оптимальні рішення для різних задач.

Важливо враховувати, що ці види алгоритмів не є взаємовиключними і можуть комбінуватися разом у складних програмах або завданнях.

1.3. Методика викладання основ алгоритмізації

Методика викладання основ алгоритмізації може варіюватися в залежності від цільової аудиторії та контексту навчання. Однак, основні принципи та підходи до викладання алгоритмізації включають:

Теоретичне вивчення: Початком вивчення алгоритмізації є теоретичне ознайомлення з основними поняттями, термінами та принципами. Студентам пояснюються концепції алгоритму, дискретності, детерміновості, коректності, кінцевості та універсальності. Важливо забезпечити чітке розуміння цих понять та їх значення в контексті алгоритмічного мислення.

Теоретичне вивчення є важливою основою для розуміння алгоритмізації. Основні концепції та принципи, які потрібно пояснити студентам, включають:

Алгоритм: Поясніть, що таке алгоритм, як він представляє послідовність дій для вирішення конкретної задачі. Визначте його основні властивості, такі як дискретність (алгоритм складається з окремих кроків), детерміновість (однозначність результату при однакових умовах), коректність (алгоритм вирішує поставлену задачу правильно), кінцевість (алгоритм завершується після скінченної кількості кроків) та універсальність (алгоритми можуть бути застосовані до різних задач) [11, с.15].

Представлення алгоритмів: Поясніть різні способи представлення алгоритмів, такі як блок-схеми, псевдокод, структурні схеми тощо. Вкажіть на загальні елементи таких представлень, такі як послідовність, умови, цикли, оператори вибору тощо.

Аналіз алгоритмів: Розгляньте поняття ефективності алгоритмів, таке як часова складність та просторова складність. Поясніть, як оцінювати та порівнювати алгоритми за їх продуктивністю та ресурсоємністю. Вкажіть на розрізнення між найкращим, середнім та найгіршим випадками.

Алгоритмічне мислення: Засвоєння алгоритмізації також передбачає розвиток алгоритмічного мислення. Алгоритмічне мислення є важливою навичкою, яка допомагає розбивати складні задачі на менші підзадачі та визначати послідовність їх виконання.

Практичні вправи: Розвиток практичних навичок у студентів є не менш важливим аспектом викладання алгоритмізації. Студентам пропонуються вправи та завдання, які допомагають їм розібратися зі створенням та реалізацією алгоритмів. Це можуть бути завдання на складання алгоритмів для конкретних ситуацій, розв'язання задач за допомогою алгоритмів, моделювання процесів та інші практичні завдання[7, с.21].

Практичні вправи є важливим елементом вивчення алгоритмізації, оскільки вони дозволяють студентам застосовувати свої знання на практиці та

розвивати навички створення та реалізації алгоритмів. Деякі практичні вправи, які можуть бути використані:

Складання алгоритмів: Студентам можна запропонувати вправи на складання алгоритмів для різних ситуацій. Наприклад, складання алгоритму для обчислення суми елементів в масиві, сортування списку чисел, пошуку максимального або мінімального значення у списку тощо. Вправи можуть бути поступово ускладнюватись, щоб стимулювати студентів до розвитку більш складних алгоритмів.

Розв'язання задач: Практичні завдання на розв'язання конкретних задач за допомогою алгоритмів можуть допомогти студентам розуміти, як алгоритми застосовуються на практиці. Наприклад, задачі на розрахунок факторіала числа, перевірку простоти числа, розробку алгоритму пошуку шляху в графі тощо. Ці завдання дають студентам можливість практично використовувати свої знання та розвивати навички аналізу задач та розробки алгоритмів для їх вирішення.

Моделювання процесів: Вправи на моделювання процесів або симуляцію реальних ситуацій також можуть бути корисними. Наприклад, створення алгоритму для моделювання руху тіл у фізиці, симуляція пасажирських потоків у транспортній системі, моделювання виборчих процесів у виборчих системах тощо.

Використання програмування: Одним з ефективних способів вивчення алгоритмізації є використання програмування. Студентам надається можливість використовувати мови програмування для створення та втілення своїх алгоритмів. Це дозволяє студентам бачити реальний результат своєї роботи та виробляти навички в написанні ефективних алгоритмів[18, с.25].

Використання програмування є потужним інструментом для вивчення алгоритмізації. Деякі переваги використання програмування в навчанні алгоритмізації включають:

Практичне втілення алгоритмів: Програмування дозволяє студентам побачити реальний результат своєї роботи. Вони можуть написати код, реалізуючи свої алгоритми, та перевірити їх на практиці. Це допомагає

зрозуміти, як алгоритми працюють і які можуть бути можливі проблеми або помилки.

Експериментування з алгоритмами: Програмування надає студентам можливість експериментувати з різними алгоритмами та їх параметрами. Вони можуть змінювати алгоритми, тестувати їх на різних вхідних даних і спостерігати за результатами. Це допомагає розвивати аналітичні навички та здатність до оптимізації алгоритмів.

Розуміння ефективності алгоритмів: Програмування дозволяє студентам оцінювати ефективність своїх алгоритмів. Вони можуть вимірювати час виконання, використання пам'яті та інші метрики для порівняння різних алгоритмів. Це дає їм можливість зрозуміти, які алгоритми працюють швидше та з меншими ресурсами.

Вироблення навичок програмування: Використання програмування в навчанні алгоритмізації допомагає студентам розвивати навички програмування. Вони вчаться писати структурований, читабельний та ефективний код, що є корисними навичками для подальшої роботи в сфері програмування.

Практичне втілення алгоритмів: Програмування дозволяє студентам побачити реальний результат своєї роботи. Вони можуть написати код, реалізуючи свої алгоритми, та перевірити їх на практиці. Це допомагає зрозуміти, як алгоритми працюють і які можуть бути можливі проблеми або помилки[5, с.16].

Експериментування з алгоритмами: Програмування надає студентам можливість експериментувати з різними алгоритмами та їх параметрами. Вони можуть змінювати алгоритми, тестувати їх на різних вхідних даних і спостерігати за результатами. Це допомагає розвивати аналітичні навички та здатність до оптимізації алгоритмів.

Розуміння ефективності алгоритмів: Програмування дозволяє студентам оцінювати ефективність своїх алгоритмів. Вони можуть вимірювати час виконання, використання пам'яті та інші метрики для порівняння різних

алгоритмів. Це дає їм можливість зрозуміти, які алгоритми працюють швидше та з меншими ресурсами.

Вироблення навичок програмування: Використання програмування в навчанні алгоритмізації допомагає студентам розвивати навички програмування. Вони вчаться писати структурований, читабельний та ефективний код, що є корисними навичками для подальшої роботи в сфері програмування.

Застосування в реальних сценаріях: Щоб забезпечити більш глибоке розуміння та застосування алгоритмів, корисно проводити практичні вправи та завдання в реальних сценаріях. Це може включати:

Задачі з реального життя: Представлення студентам задач, які моделюють реальні ситуації, допомагає їм бачити практичну цінність алгоритмів. Наприклад, завдання з оптимізації маршрутів доставки товарів або розробка алгоритму для керування роботом.

Проектна робота: Дозволяє застосовувати свої знання алгоритмізації для розробки повноцінного проекту. Це може бути створення програмного додатку, веб-сайту або розробка алгоритмів для аналізу даних. Цей підхід дозволяє бачити цілісну картину використання алгоритмів у реальних проектах.

Співпраця та командна робота: Робота в команді над проектами або задачами, що вимагають використання алгоритмів, допомагає розвивати навички співпраці та комунікації. Вони вчаться обмінюватися ідеями, розподіляти завдання та спільно шукати ефективні рішення.

Приклади з реального світу: Використання прикладів з реального світу під час викладання допомагає зрозуміти, як алгоритми використовуються у різних сферах життя. Наприклад, можна демонструвати використання алгоритмів у медицині, фінансуванні, транспорті, логістиці тощо.

Висновки до розділу 1

Висновки при дослідженні в першому розділі наступні:

У даному розділі було проведено детальний аналіз та визначено поняття алгоритму. Алгоритм є послідовністю кроків, які виконуються для досягнення певної мети. Він може бути представлений у вигляді інструкцій, блок-схем або текстового опису.

В даному розділі було досліджено поняття програми та його відмінності від алгоритму. Програма - це конкретна реалізація алгоритму за допомогою певної мови програмування. Вона складається з інструкцій, які комп'ютер може розуміти та виконувати.

У цьому розділі було розглянуто псевдокод як засіб опису алгоритму. Псевдокод є синтаксичною конструкцією, яка наближено відображає мову програмування, але зазвичай має меншу строгість та більш зрозумілий для людини формат. Використання псевдокоду допомагає уточнити логіку алгоритму та зробити його зрозумілим для інших програмістів.

У розділі було відзначено важливість точного визначення термінів, що використовуються в області алгоритмів та програмування. Чітке розуміння цих термінів є важливим для ефективного спілкування, уникнення недорозумінь та виконання якісної роботи.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ВИКЛАДАННЯ АЛГОРИТМІЗАЦІЇ

2.1. Вибір оптимальної методики в залежності від цільової аудиторії

Вибір оптимальної методики залежить від цільової аудиторії, оскільки різні методики можуть бути ефективні для різних груп людей. Ось кілька загальних порад для вибору методики в залежності від цільової аудиторії:

Дослідіть свою цільову аудиторію: Розберіться, хто вони є, які їхні потреби, інтереси та особливості. Врахуйте їхні вікові групи, соціальний статус, освіту, місце проживання та інші характеристики, що можуть впливати на їхнє сприйняття інформації та спосіб взаємодії.

Вивчіть різні методики: Ознайомтеся з різними методиками комунікації та маркетингу, такими як реклама, прямий маркетинг, соціальні медіа, веб-сайти, події та інші. Розберіться, які методики можуть бути ефективними для вашої цільової аудиторії.

Врахуйте особливості цільової аудиторії: Залежно від характеристик вашої цільової аудиторії, виберіть методику, яка найкраще відповідає їхнім потребам і перевагам. Наприклад, якщо ваша аудиторія молода і активна в соціальних медіа, можливо, ефективним буде використання цих платформ для комунікації з ними. Якщо ваша аудиторія складається з професіоналів, можливо, варто розглянути використання конференцій, семінарів або інших подій для досягнення найкращого результату.

Використання тестування та відстеження результатів є важливими кроками для оцінки ефективності вашої методики та взаємодії з цільовою аудиторією. Ось кілька порад щодо застосування цих практик:

Встановіть метрики успіху: Визначте ключові показники ефективності (KPI), які відображатимуть ваші цілі та об'єктиви. Наприклад, це можуть бути показники, такі як конверсія, кількість продажів, залучення нових клієнтів тощо. Встановлення чітких метрик допоможе вам відстежувати результати та оцінювати ефективність вашої методики[15, с.19].

Застосовуйте A/B-тестування: Використовуйте методику A/B-тестування, де ви порівнюєте дві альтернативні версії вашої методики або комунікаційного матеріалу. Розділіть вашу цільову аудиторію на дві групи та застосуйте різні варіанти методики для кожної групи. Виміряйте результати та порівнюйте їх, щоб визначити, яка версія є більш ефективною.

Використовуйте аналітичні інструменти: Використовуйте аналітичні інструменти, такі як GoogleAnalytics або інші програми аналізу даних, щоб відстежувати поведінку та взаємодію вашої цільової аудиторії. Ці інструменти надають вам інформацію про кількість відвідувань, час перебування на сайті, конверсію та інші метрики, які допоможуть вам оцінити ефективність вашої методики.

Збирайте зворотній зв'язок: Не забувайте про значення збору зворотного зв'язку від вашої цільової аудиторії. Залучайте своїх клієнтів або користувачів до процесу тестування, проводьте опитування, фокус-групи або збирайте відгуки через електронну пошту, соціальні медіа або на вашому веб-сайті. Зворотній зв'язок допоможе вам зрозуміти, як ваша цільова аудиторія сприймає вашу методику, що можна покращити та як адаптувати її до їхніх потреб.

Аналізуйте результати та вносьте зміни: Регулярно оцінюйте результати тестування та аналітики, ідентифікуйте сильні та слабкі сторони вашої методики. На основі зібраної інформації вносьте зміни та оптимізуйте свою методику, щоб досягти кращих результатів[14, с.17].

Використання тестування та відстеження результатів дозволяє вам адаптувати вашу методику до потреб вашої цільової аудиторії та досягати більш ефективних результатів. Будьте готові до постійного вдосконалення та внесення змін на основі отриманих даних та зворотного зв'язку.

2.2. Приклади практичних завдань та вправ для вивчення основ алгоритмізації

Основи алгоритмізації можна вивчати через практичні завдання та вправи, які сприяють розвитку логічного мислення та вміння конструювати послідовність дій. Ось кілька прикладів таких завдань:

Сортування списку: Запросіть учням створити алгоритм сортування списку чисел у порядку зростання або спадання. Вони можуть використовувати методи, такі як сортування бульбашкою, сортування вставкою або сортування обміном. Дайте їм можливість реалізувати свій алгоритм на комп'ютері або на папері.

Розв'язування задачі: Подайте учням задачу, яку потрібно розв'язати за допомогою алгоритму. Наприклад, задача може полягати у знаходженні найбільшого спільного дільника двох чисел або підрахунку суми чисел у заданому діапазоні. Учні повинні розробити алгоритм та пояснити його кроки.

Побудова блок-схеми: Запросіть учнів побудувати блок-схему для певного алгоритму. Наприклад, це може бути блок-схема для перевірки, чи є число простим або для обчислення факторіала числа. Це допоможе їм візуалізувати послідовність кроків та логіку алгоритму.

Вирішення головоломок: Запропонуйте учням головоломку або завдання, для якого потрібно розробити алгоритм розв'язання. Наприклад, це може бути головоломка "Ханойські вежі", де потрібно перенести круги з одної шпильки на іншу з дотриманням правил. Учні повинні розробити алгоритм та виконати його крок за кроком.

Розробка простих ігор. Ось кілька прикладів простих ігор, які можуть бути використані для вивчення основ алгоритмізації:

Вгадай число: Розробіть ігру, в якій комп'ютер вибирає випадкове число, а гравець повинен вгадати його. Учні можуть створити алгоритм, який буде направляти гравця на основі порівняння чисел (чи воно менше, більше або рівне випадковому числу). Вони можуть використовувати цикли та умови для вирішення завдання.

Камінь, ножиці, папір: Розробіть комп'ютерну версію гри "Камінь, ножиці, папір" проти комп'ютера. Учні можуть створити алгоритм, який вибирає випадкову стратегію для комп'ютера та порівнює вибір гравця з вибором комп'ютера. Вони можуть використовувати умовні оператори для визначення переможця кожного раунду.

Вгадай слово: Розробіть гру, в якій комп'ютер вибирає випадкове слово з певного списку, а гравець повинен його вгадати, отримуючи підказки. Учні можуть створити алгоритм, який відображає підказки (наприклад, перший символ слова, кількість літер тощо) та перевіряє введену відповідь гравця.

Віртуальна пам'ятка: Розробіть ігру, в якій комп'ютер показує послідовність кольорів або символів, а гравець повинен повторити цю послідовність. Учні можуть створити алгоритм, який генерує випадкову послідовність та порівнює її з введеною гравцем відповіддю.

Ці приклади допоможуть учням розуміти, як створювати прості алгоритми для вирішення завдань.

Ось конкретні приклади практичних завдань та вправ для вивчення основ алгоритмізації[8, с.16]:

Побудова алгоритму для приготування кави: Учні повинні розробити алгоритм, який описує послідовність кроків для приготування кави. Вони можуть врахувати такі дії, як нагрівання води, додавання кавового порошку, час заварювання та інше. Учні можуть записати свій алгоритм у вигляді псевдокоду або скласти блок-схему для наглядності.

На рис.2.1. приклад алгоритму для приготування кави у вигляді псевдокоду.

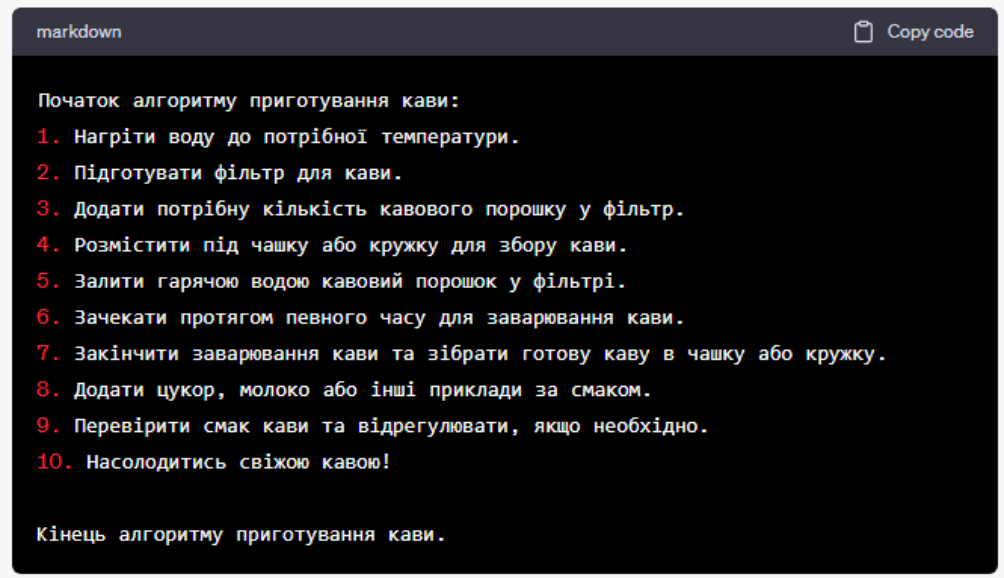

Рис.2.1. Алгоритм у вигляді псевдокоду

Переведення числа в двійкову систему: Учні повинні створити алгоритм, який переводить задане десяткове число в його двійковий еквівалент. Вони можуть використовувати ділення на 2 та отримання остачі для кожного кроку, поки число не стане рівним 0. Алгоритм може використовувати цикли та рядки для збереження результату.

На рис.2.2. алгоритм для переведення заданого десяткового числа в його двійковий еквівалент.

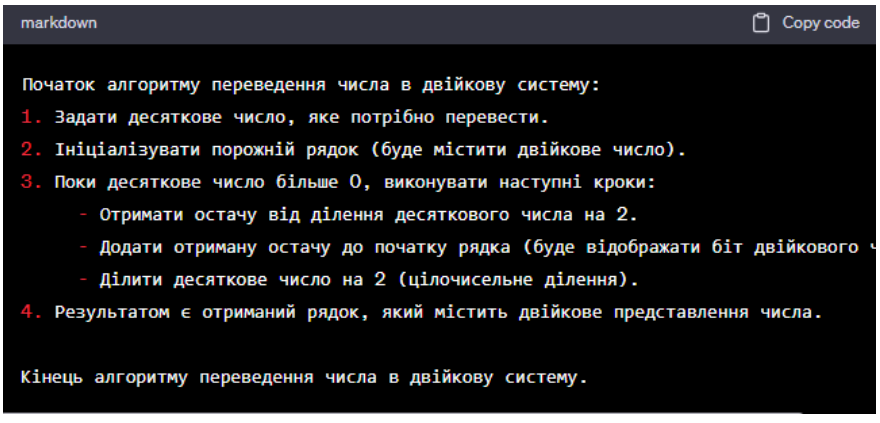

Рис.2.2. Алгоритм переведення десяткового числа в його двійковий еквівалент

Учні можуть використовувати цей алгоритм, щоб реалізувати переведення десяткового числа в його двійковий еквівалент у своїх програмах. Наприклад, на мові Python це може виглядати наступним чином:. (рис.2.3.)

```python
def decimal_to_binary(decimal_num):
    binary = ''
    while decimal_num > 0:
        remainder = decimal_num % 2
        binary = str(remainder) + binary
        decimal_num //= 2
    return binary
```

Рис.2.3. Реалізація алгоритму

За допомогою цієї функції decimal_to_binary(), учні можуть передати десяткове число і отримати його двійкове представлення (рис.2.4).

```python
decimal_num = 10
binary_num = decimal_to_binary(decimal_num)
print(binary_num)  # Виведе "1010"
```

Рис.2.4. Перетворення десяткового числа 10 в його двійковий еквівалент "1010"

Розробка алгоритму для сортування списку слів за алфавітом: Учні повинні створити алгоритм, який сортує заданий список слів в алфавітному порядку. Вони можуть використовувати порівняння між словами та переставляти їх у правильному порядку. Алгоритм може використовувати цикли та умови для порівняння слів.

На рис.2.5.алгоритму для сортування заданого списку слів в алфавітному порядку.

```markdown
Початок алгоритму сортування списку слів за алфавітом:
1. Задати список слів, який потрібно відсортувати.
2. Встановити прапорець `swapped` на значення `True`.
3. Повторювати, поки `swapped` є `True`:
    - Встановити `swapped` на значення `False`.
    - Пройтись по кожному слову у списку (індекси від 0 до n-1):
        - Якщо поточне слово більше наступного слова в алфавітному порядку:
            - Поміняти їх місцями.
            - Встановити `swapped` на значення `True`.
4. Результатом є відсортований список слів.

Кінець алгоритму сортування списку слів за алфавітом.
```

Рис.2.5. Алгоритм для сортування заданого списку слів в алфавітному порядку

Учні можуть використовувати цей алгоритм для сортування списку слів у своїх програмах. Наприклад, на мові Python це може виглядати наступним чином: (рис.2.6.)

```python
def sort_words(word_list):
    swapped = True
    while swapped:
        swapped = False
        for i in range(len(word_list) - 1):
            if word_list[i] > word_list[i+1]:
                word_list[i], word_list[i+1] = word_list[i+1], word_list[i]
                swapped = True
    return word_list
```

Рис.2.6. Реалізація алгоритму

За допомогою цієї функції sort_words(), учні можуть передати список слів і отримати його відсортований за алфавітом варіант. (рис.2.7).

```python
words = ['apple', 'banana', 'cherry', 'date']
sorted_words = sort_words(words)
print(sorted_words)  # Виведе ['apple', 'banana', 'cherry', 'date']
```

Рис.2.7. Сортування списку слів ['apple', 'banana', 'cherry', 'date'] у алфавітному порядку

Розробка алгоритму для обчислення факторіала числа: Учні повинні створити алгоритм, який обчислює факторіал заданого числа. Вони можуть використовувати цикли та множення для послідовного перемноження чисел від 1 до заданого числа. Алгоритм може бути рекурсивним або ітеративним.

Алгоритм для обчислення факторіала заданого числа:

Якщо число менше 0, вивести повідомлення про помилку, факторіал визначений лише для невід'ємних чисел.

Якщо число рівне 0, факторіал дорівнює 1.

Інакше:

Ініціалізувати змінну factorial на 1.

Почати цикл від 1 до заданого числа (включно):

Помножити factorial на поточне число.

Результатом є значення factorial.

Цей алгоритм можна реалізувати як рекурсивну функцію або як ітеративний цикл. Ось приклад реалізації алгоритму у Python:

Рекурсивний підхід (рис.2.8.):

```python
def factorial_recursive(n):
    if n < 0:
        return "Факторіал визначений лише для невід'ємних чисел."
    elif n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)
```

Рис.2.8. Рекурсивний підхід

Ітеративний підхід (рис.2.9.):

```python
def factorial_iterative(n):
    if n < 0:
        return "Факторіал визначений лише для невід'ємних чисел."
    elif n == 0:
        return 1
    else:
        factorial = 1
        for i in range(1, n + 1):
            factorial *= i
        return factorial
```

Рис.2.9. Інтеративний підхід

За допомогою цих функцій, учні можуть обчислити факторіал заданого числа (рис.2.10).

```python
n = 5
recursive_result = factorial_recursive(n)
print(f"Рекурсивний підхід: {recursive_result}")  # Виведе 120

iterative_result = factorial_iterative(n)
print(f"Ітеративний підхід: {iterative_result}")  # Виведе 120
```

Рис.2.10. Факторіал заданого числа

Цей приклад демонструє обчислення факторіалу числа 5 за допомогою рекурсивного та ітеративного підходу. Результатом є число 120.

Розробка алгоритму для пошуку найбільшого числа в масиві.

Отже, ось приклад алгоритму для пошуку найбільшого числа в масиві:

Задати змінну max_num із початковим значенням, наприклад, першим елементом масиву.

Проходитись по кожному елементу масиву:

Якщо поточний елемент більший за max_num, оновити значення max_num на поточний елемент.

По завершенні проходу по масиву, max_num буде містити найбільше число.

На рис.2.11.наведено приклад реалізації цього алгоритму на мові Python:

python Copy code

```python
def find_max_number(arr):
    max_num = arr[0]  # Початкове значення - перший елемент масиву
    for num in arr:
        if num > max_num:
            max_num = num
    return max_num
```

Рис.2.11. Реалізація алгоритму

Учні можуть використовувати цей алгоритм для пошуку найбільшого числа в будь-якому заданому масиві чисел. Вони можуть експериментувати з різними масивами та перевіряти правильність результату, порівнюючи його зі значенням, яке повертає функція max() мови Python для масиву.

2.3. Інструменти та ресурси для вивчення алгоритмізації

Існує багато інструментів та ресурсів, які можна використовувати для вивчення алгоритмізації. Ось деякі з них:

Онлайн-курси: Веб-платформи, такі як Coursera, edX та Udemy, пропонують курси з алгоритмізації та програмування. Ці курси можуть мати

відеолекції, практичні завдання та оцінювання, що допомагає зрозуміти та відпрацювати концепції алгоритмізації.

Книги: Існує безліч книг, які присвячені алгоритмізації та структурам даних. Найвідоміша з них - "IntroductiontoAlgorithms" (Автори: Томас Кормен, Чарльз Лейзерсон, Рональд Ріверст, КліффордШтайн). Інші популярні книги включають "Algorithms, Part I" та "Algorithms, Part II" (Автор: Роберт Седжвік) та "GrokkingAlgorithms" (Автор: АдітиБхаргава).

Онлайн-платформи з програмування: Сайти, такі як HackerRank, LeetCode та Codecademy, надають практичні завдання та виклики з алгоритмізації та програмування. Вони можуть допомогти вам відпрацювати навички написання та виконання алгоритмів.

Алгоритмічні платформи: Деякі веб-сайти, такі як Topcoder та Codeforces, організовують онлайн-змагання та змагальні програмування, де ви можете використовувати свої навички алгоритмізації для вирішення задач та змагатися з іншими учасниками[14, с.28].

Візуалізація алгоритмів: Деякі веб-сайти та програми, наприклад VisuAlgo, дозволяють візуалізувати роботу різних алгоритмів.

Так, дійсно, VisuAlgo є чудовим ресурсом для візуалізації роботи різних алгоритмів. Онлайн-інструменти, які надають візуальну демонстрацію алгоритмів, можуть значно полегшити процес їх розуміння. Ось кілька інших ресурсів, які також надають візуалізацію алгоритмів:

AlgorithmVisualizer: Це інтерактивний веб-сайт, який дозволяє візуалізувати роботу різних алгоритмів, таких як сортування, пошук, дерева та графи. Ви можете побачити крок за кроком, як алгоритм працює, та вивчати його поведінку.

Visualgo: Цей веб-сайт пропонує візуалізацію багатьох алгоритмів, включаючи сортування, графи, дерева, рядки та інші. Ви можете керувати швидкістю відтворення та спостерігати, як алгоритм розв'язує проблему крок за кроком.

SortingVisualizer: Це спеціалізований веб-сайт для візуалізації різних алгоритмів сортування. Ви можете вибрати алгоритм сортування, ввести вхідні дані та побачити, як він працює на живому графіку.

Ці ресурси можуть бути корисними для вивчення та розуміння алгоритмів, допомагаючи вам побачити їх в реальному часі та сприяючи кращому усвідомленню їх роботи.

Висновки до розділу 2

Висновки до розділу включають наступне:

Розглянуто різні підходи до викладання алгоритмізації: традиційні лекційні курси, онлайн-курси, візуалізація алгоритмів та практичні завдання.

Традиційні лекційні курси з алгоритмізації надають студентам теоретичні знання та основи, але можуть бути менш ефективними для розуміння та практичного застосування алгоритмів.

Онлайн-курси, доступні на платформах, таких як Coursera, edX та Udemy, надають гнучкість та можливість вивчати алгоритмізацію на власному темпі. Вони зазвичай містять відеоуроки, практичні завдання та оцінювання.

Візуалізація алгоритмів є потужним інструментом, який допомагає студентам краще зрозуміти роботу алгоритмів шляхом візуального представлення їх крок за кроком.

Практичні завдання та виклики з програмування, доступні на платформах, таких як HackerRank та LeetCode, дозволяють студентам відпрацювати свої навички алгоритмізації та програмування на реальних завданнях.

Комбінація різних підходів, таких як поєднання традиційних лекцій з практичними завданнями та візуалізацією алгоритмів, може бути ефективним методом викладання алгоритмізації, допомагаючи студентам зрозуміти концепції та набути практичних навичок.

РОЗДІЛ 3. МЕТОДИЧНІ ОСОБЛИВОСТІ НАВЧАННЯ СТУДЕНТІВ ОСНОВАМ АЛГОРИТМІЗАЦІЇ

3.1. Розробка моделі підготовки фахівців ЦТ до викладання алгоритмізації

1. Аналіз потреб та вимог

Для розробки ефективної моделі підготовки фахівців, першим кроком є детальний аналіз потреб і вимог до викладачів алгоритмізації. Цей етап є основою для подальшої побудови навчальної програми та визначення ключових аспектів підготовки. Аналіз може включати кілька важливих складових:

1.1. Оцінка наявного рівня знань та навичок

Перед початком підготовки викладачів необхідно провести оцінку поточного рівня їхніх знань і навичок. Це дозволить виявити слабкі сторони та сфери, які потребують особливої уваги під час навчання. Оцінка може включати:

Тестування з теоретичних аспектів

Перевірка знань основних алгоритмів і структур даних, а також принципів їх оптимізації.

Оцінка практичних навичок

Тестування вміння писати, оптимізувати та аналізувати алгоритми, використовувати різні мови програмування.

Інтерв'ю та опитування

Збір інформації про те, як викладачі викладають алгоритмізацію, які методи вони використовують, та з якими труднощами стикаються.

1.2. Визначення вимог до професійної компетентності

Необхідно чітко окреслити, які вимоги ставляться до викладачів, що будуть навчати алгоритмізації. Ці вимоги повинні бути адаптовані до сучасних стандартів освіти та вимог ринку праці, і вони можуть включати:

Глибокі знання основних тем алгоритмізації: Сортування, пошук, рекурсія, динамічне програмування, графи та інші типи алгоритмів.

Знання сучасних інструментів та мов програмування: Викладачі мають володіти актуальними мовами програмування (наприклад, Python, C++, Java) та середовищами для розробки та тестування алгоритмів.

Педагогічні навички: Викладачі повинні мати здатність адаптувати матеріал до різних рівнів студентів і використовувати інтерактивні методи навчання для кращого засвоєння складних тем.

1.3. Аналіз сучасних тенденцій в алгоритмізації та програмуванні

Також важливо враховувати зміни в галузі алгоритмізації та програмування. Нові досягнення в комп'ютерних науках, такі як розвиток штучного інтелекту, машинного навчання, паралельних обчислень і обробки великих даних, вимагають від викладачів оновлення їхніх знань і навичок. Аналіз сучасних тенденцій допоможе адаптувати навчальний процес до реальних потреб і забезпечити відповідність курсу останнім науковим розробкам.

1.4. Зворотний зв'язок від студентів та інших фахівців

Ще одним важливим елементом аналізу є зворотний зв'язок від тих, хто вже навчається алгоритмізації. Це можуть бути відгуки студентів, результати їхнього навчання, а також рекомендації від інших викладачів або експертів у галузі. Такий зворотний зв'язок дозволяє коригувати підходи та вдосконалювати навчальний процес, враховуючи реальні проблеми та потреби.

1.5. Аналіз вимог роботодавців та ринку праці

Важливо врахувати, яких саме фахівців шукають роботодавці в сфері комп'ютерних наук. Для цього необхідно провести дослідження ринку праці, вивчити вакансії, що стосуються програмування та алгоритмізації, а також вимоги до навичок, які часто вказуються в описах вакансій. Це дозволить сформувати чітке уявлення про те, які компетенції мають бути у викладачів для того, щоб їхні студенти були конкурентоспроможними на ринку праці.

Аналіз потреб і вимог — це критично важливий етап, який дозволяє сформувати чітке розуміння того, які навички і знання повинні отримати викладачі, щоб ефективно навчати алгоритмізації. Врахування поточного рівня знань, вимог ринку праці, а також сучасних тенденцій в галузі допоможе

створити програму, яка буде не тільки актуальною, але й сприятиме підготовці висококваліфікованих фахівців.

2. Розробка навчальної програми

Навчальна програма є основою для підготовки фахівців до викладання алгоритмізації. Вона має забезпечити баланс між теоретичними знаннями, практичними навичками та методичною підготовкою. Детальний опис кожного компонента:

2.1 Теоретична підготовка

Цей компонент програми має надати викладачам базові знання з алгоритмізації, які є основою для викладання учням:

Основи алгоритмів:

Поняття алгоритму, властивості та типи.

Послідовність, розгалуження, цикли.

Принципи побудови блок-схем.

Структури даних:

Масиви, списки, дерева, графи.

Основні операції над структурами даних.

Мови програмування:

Вивчення мови Scratch для початкового етапу.

Основи Python як універсальної мови програмування.

Інші інструменти (Pascal, JavaScript) за потреби.

Алгоритмічні концепції:

Сортування, пошук, динамічне програмування.

Простота та ефективність алгоритмів.

2.2 Методична підготовка

Методична частина допомагає викладачам навчитися правильно передавати знання:

Педагогічні підходи:

Як адаптувати складний матеріал до рівня учнів різного віку.

Врахування когнітивних особливостей дітей.

Використання ігрових елементів для пояснення складних тем.

Розробка уроків:

Структура заняття: мотивація, пояснення, практичні завдання, рефлексія.

Інтеграція алгоритмізації з іншими STEM-дисциплінами.

Технологічні інструменти:

Використання візуальних платформ (Scratch, Blockly).

Інтерактивні середовища (Code.org, Tynker, Grasshopper).

Онлайн-інструменти для групової роботи.

2.3 Практична частина

Ця частина програми покликана закріпити знання та надати викладачам можливість застосувати їх на практиці:

Розв’язування алгоритмічних задач:

Від простих (наприклад, підрахунок суми чисел) до складних (розробка гри).

Використання середовищ програмування (Python, Scratch).

Розробка авторських уроків:

Вчителі створюють власні уроки та завдання з алгоритмізації.

Захист і обговорення підготовлених матеріалів.

Моделювання уроків:

Практичні заняття, де викладачі виступають у ролі учнів.

Аналіз типових проблем, що виникають під час викладання.

Формат програми

Онлайн-лекції:

Теоретичні основи, доступ до записів лекцій.

Практичні заняття офлайн:

Взаємодія з тренерами, спільна розробка матеріалів.

Самостійна робота:

Виконання домашніх завдань, робота з навчальними платформами.

Проєктна робота:

Створення викладачами освітнього продукту (уроку, курсу, гри).

Приклади завдань у програмі

Scratch:

Створити анімацію, яка пояснює принцип роботи циклів.

Розробити гру з використанням умовних операторів.

Python:

Написати програму для сортування списку.

Побудувати алгоритм обчислення Фібоначчі.

Інтерактивні платформи:

Проходження навчальних модулів на Code.org.

Виконання завдань у Blockly.

Очікувані результати

Після завершення програми викладачі:

Знатимуть основи алгоритмізації та програмування.

Володітимуть методиками викладання складних тем у доступній формі.

Будуть уміти розробляти власні уроки та інтегрувати алгоритмізацію в загальний навчальний процес.

Зможуть ефективно використовувати сучасні технології та інструменти.

Такий підхід забезпечує не лише засвоєння знань, а й формування практичних навичок, необхідних для роботи з учнями.

3. Структура моделі підготовки

Структура моделі підготовки фахівців до викладання алгоритмізації є ключовою для забезпечення системного навчання та послідовного засвоєння знань і навичок. Ця модель має поетапний характер, що дозволяє викладачам поступово поглиблювати свої знання, освоювати нові методики та відточувати їх на практиці.

3.1 Вступний етап

Мета: створити стартові умови для навчання, оцінити рівень підготовки викладачів і залучити їх до процесу.

Діагностика знань і навичок:

Проведення вступного тестування для визначення рівня розуміння алгоритмізації та програмування.

Оцінка педагогічного досвіду викладачів.

Мотивація викладачів:

Обговорення важливості алгоритмізації у сучасній освіті.

Демонстрація прикладів успішного навчання дітей через алгоритмізацію (кейси, відео, відгуки).

Знайомство з навчальною програмою:

Представлення цілей, структури та очікуваних результатів курсу.

Обговорення індивідуальних і групових завдань.

3.2 Базовий етап

Мета: забезпечити викладачів теоретичними знаннями та основними навичками, необхідними для викладання.

Теоретичний блок:

Основи алгоритмізації (послідовність, розгалуження, цикли).

Вступ до візуального програмування (Scratch, Blockly).

Базові алгоритми (обчислення, сортування, пошук).

Психолого-педагогічні аспекти навчання алгоритмізації дітям.

Практичний блок:

Виконання простих завдань у Scratch.

Написання базових алгоритмів у Python.

Використання платформ Code.org або Tynker для ознайомлення з інтерактивними завданнями.

Інтерактивні заняття:

Обговорення типових помилок у викладанні алгоритмізації.

Симуляція уроків із використанням реальних завдань.

3.3 Поглиблений етап

Мета: розширити знання викладачів і надати їм можливість застосовувати їх у створенні власних методичних матеріалів.

Розширення теоретичних знань:

Складніші алгоритми (рекурсія, робота з багатовимірними структурами).

Методи оптимізації алгоритмів.

Інтеграція алгоритмізації з іншими STEM-дисциплінами.

Розробка навчальних матеріалів:

Створення інтерактивних уроків.

Розробка авторських завдань (наприклад, створення гри у Scratch або практичної задачі на Python).

Методичні семінари:

Вивчення сучасних підходів до викладання (гейміфікація, проблемне навчання).

Обговорення підходів до роботи з різними віковими групами.

Практичне моделювання уроків:

Викладачі проводять пробні заняття з іншими учасниками курсу, які виступають у ролі учнів.

Отримання зворотного зв’язку від колег і тренерів.

3.4 Заключний етап

Мета: підсумувати результати навчання, провести оцінювання та сформувати стратегію подальшого професійного розвитку.

Оцінювання знань і навичок:

Фінальне тестування (теоретичне та практичне).

Оцінювання методичних матеріалів, створених викладачами.

Презентація проєктів:

Захист авторських навчальних курсів або уроків.

Демонстрація інтерактивних матеріалів і програмних продуктів.

Рефлексія:

Обговорення досягнень і труднощів.

Формування рекомендацій для подальшого вдосконалення.

Видача сертифікатів:

Сертифікати про проходження курсу, які підтверджують набуті компетенції.

План подальшого розвитку:

Рекомендації щодо участі в конференціях, семінарах, спільнотах викладачів.

Додаткові компоненти структури

Супровід викладачів після курсу:

Консультації тренерів.

Підтримка у впровадженні розроблених матеріалів.

Створення спільноти викладачів для обміну досвідом.

Робота з реальними учнями:

Пілотні уроки з дітьми під керівництвом тренерів.

Аналіз результатів і адаптація підходів.

Переваги структури

Поетапність: Викладачі поступово переходять від базових знань до глибокої інтеграції.

Практичність: Велика частина часу приділяється практичним завданням і створенню власних матеріалів.

Гнучкість: Програма адаптується до рівня та потреб кожного учасника.

Інтерактивність: Викладачі беруть участь у обговореннях, практичних заняттях і проєктній роботі.

Ця структура сприяє комплексній підготовці викладачів, допомагаючи їм стати не лише фахівцями з алгоритмізації, але й ефективними педагогами.

4. Методологічні підходи

Методологічні підходи відіграють ключову роль у підготовці викладачів до викладання алгоритмізації. Вони дозволяють створити ефективний навчальний процес, який враховує сучасні вимоги до освіти, психолого-педагогічні аспекти та специфіку предмета. Нижче наведено докладний опис основних методологічних підходів, які можуть бути використані.

4.1 Інтерактивні методи

Мета: залучити викладачів до активної участі у навчальному процесі, зробити навчання цікавим і зрозумілим.

Гейміфікація:

Використання ігрових елементів у навчанні (нагромадження балів, змагання, нагороди).

Створення завдань у формі ігор (наприклад, програмування героїв для проходження лабіринту).

Використання ігрових платформ (наприклад, Code.org, Scratch).

Рольові ігри:

Викладачі моделюють ситуації з учнями, виконуючи ролі "учителя" і "учня".

Аналіз можливих складнощів і розробка підходів до їх вирішення.

Тренінги та воркшопи:

Практичні заняття, де викладачі самостійно виконують завдання.

Робота в групах для розробки алгоритмів і проектів.

4.2 Проєктне навчання

Мета: сформувати у викладачів навички розробки та реалізації освітніх проєктів.

Розробка власних проєктів:

Створення освітніх продуктів (уроків, інтерактивних презентацій, програмних рішень).

Наприклад: створення гри у Scratch або проєкту автоматизації певного процесу на Python.

Інтеграція міждисциплінарного підходу:

Об'єднання алгоритмізації з іншими STEM-напрямами (математика, фізика, робототехніка).

Наприклад: написання програми для розв'язування математичних задач.

Презентація результатів:

Захист проєктів перед іншими викладачами та тренерами.

Аналіз і обговорення створених матеріалів.

4.3 Змішане навчання

Мета: поєднати переваги онлайн- і офлайн-навчання для підвищення ефективності.

Онлайн-навчання:

Використання відеолекцій, інтерактивних курсів і платформ.

Доступ до навчальних матеріалів у будь-який час (навіть після завершення курсу).

Офлайн-навчання:

Практичні заняття, живе спілкування з тренерами.

Робота у малих групах для виконання завдань.

Гнучкий графік:

Можливість самостійного вибору часу для виконання завдань.

Приклади платформ:

Code.org, Tynker, Grasshopper для алгоритмізації.

Coursera, Udemy для додаткової теоретичної підготовки.

4.4 Проблемне навчання

Мета: навчити викладачів вирішувати реальні проблеми через алгоритмізацію та стимулювати критичне мислення.

Постановка проблеми:

Викладачам пропонують задачу, яка не має очевидного розв’язання.

Наприклад: створити алгоритм для оптимального маршруту доставки.

Колективне обговорення:

Спільна робота над пошуком розв'язання.

Використання різних підходів і методів.

Розробка рішень:

Виконання завдань із застосуванням знань алгоритмізації.

Аналіз ефективності створених алгоритмів.

4.5 Індивідуалізація навчання

Мета: врахувати рівень підготовки та потреби кожного викладача.

Діагностика рівня знань:

Проведення вступного тестування для визначення базового рівня.

Адаптація завдань до рівня викладача.

Індивідуальні консультації:

Робота з тренерами для вирішення конкретних питань.

Індивідуальні проєкти:

Створення навчальних матеріалів, які відповідають стилю роботи викладача.

4.6 Колаборативне навчання

Мета: формувати навички роботи в команді, обміну знаннями та спільного вирішення завдань.

Групові завдання:

Розподіл ролей у команді (аналітик, програміст, дизайнер).

Розробка спільних проєктів.

Дискусійні групи:

Обговорення різних підходів до викладання алгоритмізації.

Спільний аналіз педагогічних кейсів.

Спільнота викладачів:

Створення групи для обміну досвідом і підтримки після завершення курсу.

4.7 Використання сучасних технологій

Мета: забезпечити викладачів інструментами, які спрощують викладання алгоритмізації.

Платформи для навчання алгоритмізації:

Scratch для початкового етапу.

Python для текстового програмування.

Blockly для інтерактивних завдань.

Онлайн-інструменти:

Віртуальні дошки (Miro, Jamboard) для пояснення тем.

Платформи для тестування (Quizizz, Kahoot) для оцінювання знань.

Освітні гаджети:

Використання робототехніки (LEGO Mindstorms, Arduino) для практичних занять.

Програмовані міні-роботи (Ozobot, micro:bit).

Очікувані результати від застосування методологічних підходів

Викладачі будуть здатні цікаво та доступно викладати основи алгоритмізації.

Вони опанують сучасні педагогічні методи й зможуть застосовувати їх на практиці.

Зможуть адаптувати викладання до різних вікових груп і рівнів підготовки учнів.

Здобудуть навички роботи з сучасними технологіями та онлайн-інструментами.

Будуть готові до індивідуального та командного викладання в умовах змішаного навчання.

Ці методологічні підходи дозволяють забезпечити якісну підготовку викладачів, зробити процес навчання ефективним і цікавим.

5. Оцінювання ефективності підготовки

Оцінювання ефективності підготовки фахівців до викладання алгоритмізації є важливим етапом, який дозволяє визначити, наскільки успішно реалізовано навчальну програму, і забезпечити зворотний зв'язок для подальшого вдосконалення. Ефективність оцінюється через аналіз досягнень викладачів, якості їхніх методичних матеріалів, а також їхньої здатності викладати алгоритмізацію.

5.1 Критерії оцінювання

Для визначення успішності підготовки використовуються такі критерії:

Теоретичні знання:

Рівень засвоєння основ алгоритмізації (поняття алгоритму, структури даних, базові алгоритми).

Розуміння принципів роботи програм (послідовність, розгалуження, цикли).

Практичні навички:

Здатність самостійно розробляти та реалізовувати алгоритми.

Використання мов програмування (Scratch, Python) для вирішення задач.

Методична підготовка:

Вміння розробляти навчальні матеріали.

Ефективність викладання складних тем у доступній формі.

Інноваційність:

Застосування сучасних технологій у викладанні.

Інтеграція алгоритмізації з іншими дисциплінами.

Рівень взаємодії з учнями:

Здатність адаптувати матеріал до різного рівня підготовки учнів.

Використання інтерактивних ігрових методик.

5.2 Методи оцінювання

Для оцінювання ефективності використовуються різноманітні методи:

Теоретичне тестування

Проведення тестів для перевірки знань викладачів.

Тестування охоплює як базові поняття (що таке алгоритм, приклади базових структур даних), так і складні теми (оптимізація алгоритмів, рекурсія).

Практичні завдання

Виконання завдань, пов’язаних із розробкою алгоритмів.

Приклади завдань:

Написати програму для сортування масиву.

Розробити інтерактивний проект у Scratch, наприклад, гру з використанням умовних операторів і циклів.

Аналіз правильності, оптимальності й естетики коду.

Методичний аналіз

Оцінка розроблених викладачами навчальних матеріалів:

Сценарії уроків, презентації, завдання для учнів.

Наявність дидактичної цінності, доступності для учнів і інноваційності.

Рецензування матеріалів іншими викладачами й тренерами.

Пробне викладання

Викладачі проводять модельні уроки:

Презентують тему перед групою колег або справжніми учнями.

Використовують різні методики викладання (гейміфікація, проєктне навчання).

Оцінювання уроку:

Якість пояснення матеріалу.

Залученість учасників у процес.

Використання інтерактивних інструментів.

Анкетування та опитування

Анкетування викладачів:

Оцінка їхнього задоволення навчанням, виявлення проблем і побажань.

Збір зворотного зв'язку від учнів, які брали участь у пробних уроках:

Наскільки зрозуміло викладено матеріал.

Чи цікаві завдання й методи навчання.

Проєктна робота

Захист індивідуальних або групових проєктів:

Наприклад, створення інтерактивного курсу з алгоритмізації для молодших класів або проєкту в Python для розв'язання реальної задачі.

Критерії оцінювання:

Реалістичність і складність проєкту.

Якість презентації.

Практична цінність.

5.3 Інструменти оцінювання

Онлайн-тести та платформи:

Використання автоматизованих систем (Google Forms) для швидкого тестування знань.

Системи для оцінки коду:

Наприклад, Codewars, LeetCode або інші платформи для аналізу алгоритмів і програм.

Шкали оцінювання:

Створення критеріїв із чіткими описами рівнів: від початкового до експертного.

Система зворотного зв’язку:

Використання опитувань і інтерв’ю для отримання якісного фідбеку.

5.4 Аналіз результатів

Після завершення оцінювання:

Складання загального рейтингу:

Визначення сильних і слабких сторін кожного викладача.

Обговорення результатів:

Проведення індивідуальних консультацій для роз'яснення результатів.

Надання рекомендацій для подальшого вдосконалення.

Підготовка звіту:

Аналіз успішності всієї програми.

Пропозиції щодо вдосконалення змісту та методів підготовки.

5.5 Використання результатів оцінювання

Індивідуальний розвиток:

Викладачі отримують рекомендації щодо вдосконалення своїх знань і навичок.

Пропозиції щодо участі у додаткових тренінгах або семінарах.

Модернізація програми підготовки:

Удосконалення матеріалів, методів і структури на основі результатів оцінювання.

Формування професійної спільноти:

Використання успішного досвіду для створення навчальних кейсів і обміну знаннями.

Очікувані результати

Визначення рівня готовності викладачів до викладання алгоритмізації.

Формування індивідуальних планів розвитку для кожного учасника.

Збільшення ефективності програми підготовки завдяки регулярному аналізу та зворотному зв'язку.

Створення бази методичних і практичних матеріалів для вдосконалення навчального процесу.

Такий підхід до оцінювання дозволяє забезпечити високий рівень підготовки викладачів і створити ефективну, адаптовану до потреб учнів систему викладання алгоритмізації.

3.2. Методичні особливості реалізації методів навчання під час організації освітнього процесу з модуля «Основи алгоритмізації»

Організація освітнього процесу з модуля «Основи алгоритмізації» вимагає використання адаптованих методичних підходів, які забезпечують ефективне навчання учнів. Цей процес орієнтований на формування в учнів базових знань і практичних навичок алгоритмізації через поєднання традиційних і сучасних педагогічних методів.

Основні методи навчання та їх методичні особливості

1. Інтерактивні методи

Особливості застосування:

Включення інтерактивних завдань для активізації пізнавальної діяльності.

Створення умов для роботи в парах чи групах.

Використання гейміфікації (навчання через ігри).

Приклади:

Гра "Алгоритмічний лабіринт": учні розробляють алгоритми для проходження лабіринту у візуальному середовищі (наприклад, Scratch).

Обговорення кейсів: аналіз реальних проблем із подальшим пошуком алгоритмічних рішень.

2. Метод проєктів

Особливості застосування:

Навчання через виконання практичних завдань і створення кінцевого продукту.

Розвиток самостійності, креативності та навичок роботи з інформацією.

Приклади:

Створення гри або анімації у Scratch, використовуючи базові алгоритмічні структури.

Розробка програми для обчислення суми чисел із заданого діапазону в Python.

3. Проблемно-орієнтоване навчання

Особливості застосування:

Постановка перед учнями проблемного завдання без готового алгоритму вирішення.

Учні самостійно шукають оптимальні рішення, застосовуючи отримані знання.

Приклади:

Завдання: "Розробіть алгоритм, який знаходить найкоротший шлях між двома точками на карті".

Вивчення алгоритму сортування через аналіз реальної ситуації (наприклад, сортування товарів у магазині).

4. Лекційно-практичний метод

Особливості застосування:

Коротке викладення теорії з акцентом на ключові поняття.

Виконання практичних завдань одразу після пояснення матеріалу.

Приклади:

Після теоретичного пояснення циклів у Python учні пишуть програму, яка виводить таблицю множення.

Практичні вправи на складання блок-схем із подальшою реалізацією їх у Scratch.

5. Дослідницький метод

Особливості застосування:

Учні проводять дослідження та аналізують ефективність різних алгоритмів.

Використання завдань із відкритим кінцем для стимулювання творчості.

Приклади:

Порівняння швидкодії алгоритмів сортування.

Створення алгоритму для автоматизації певного процесу (наприклад, підрахунок середньої оцінки в класі).

Етапи організації навчання з модуля

1. Вступний етап

Ціль: мотивація учнів і створення бази знань.

Методи:

Демонстрація практичної цінності алгоритмів (наприклад, робота алгоритмів у пошукових системах).

Виконання простих завдань у візуальному середовищі.

2. Основний етап

Ціль: формування навичок складання та реалізації алгоритмів.

Методи:

Інтерактивні вправи (наприклад, розробка блок-схем).

Використання реальних задач для тренування.

Групові проєкти, де кожен учень відповідає за певну частину алгоритму.

3. Заключний етап

Ціль: закріплення знань і демонстрація результатів.

Методи:

Створення індивідуальних проєктів.

Проведення міні-змагань (наприклад, на швидкість розв'язання задачі).

Обговорення рефлексій щодо роботи над алгоритмічними завданнями.

Методичні рекомендації для викладачів

Адаптація завдань:

Завдання мають відповідати рівню підготовки учнів.

Для початківців використовуйте прості завдання з чіткими інструкціями.

Для просунутих — задачі з відкритим кінцем і можливістю самостійного пошуку рішення.

Використання технологій:

Scratch для візуального програмування.

Python для текстового програмування.

Освітні платформи (наприклад, Code.org, Tynker) для інтерактивного навчання.

Диференціація навчання:

Розробляйте індивідуальні завдання для учнів із різним рівнем знань.

Запропонуйте варіативність: прості й складні версії завдання.

Забезпечення зворотного зв’язку:

Під час виконання завдань коментуйте рішення учнів, пояснюючи їхні помилки.

Обговорюйте різні підходи до розв'язання задачі.

Розвиток логічного мислення:

Використовуйте завдання, які стимулюють аналіз і синтез (наприклад, "знайти помилки в алгоритмі").

Розвивайте навички розбиття задачі на підзадачі.

Особливості організації практичної роботи

Інструменти:

Візуальні середовища програмування для початкових етапів.

Використання текстових редакторів для складніших задач.

Етапи виконання практичного завдання:

Постановка задачі.

Обговорення можливих підходів.

Написання алгоритму або програми.

Тестування результатів.

Аналіз ефективності.

Контроль і допомога:

Викладач спостерігає за процесом роботи й надає індивідуальні консультації.

Після завершення роботи аналізуються найтиповіші помилки.

Очікувані результати

Учні засвоять базові концепції алгоритмізації.

Сформують навички створення та реалізації алгоритмів у різних середовищах.

Розвинуть логічне й критичне мислення, що стане основою для подальшого навчання програмуванню.

Отримають практичний досвід розв’язання задач у реальних умовах.

Методичні особливості спрямовані на те, щоб зробити навчання доступним, цікавим і результативним для учнів різного рівня підготовки.

Висновки до розділу 3

Висновки до розділу включають наступне:

Розроблено комплексний підхід до навчання так як, методична реалізація модулю «Основи алгоритмізації» вимагає поєднання різних методів навчання: від традиційних (лекції, практичні заняття) до сучасних інтерактивних і проєктно-орієнтованих. Такий підхід дозволяє зробити процес навчання різноманітним, стимулювати пізнавальну активність учнів і забезпечити ефективне засвоєння матеріалу.

У навчальному процесі буде зроблено орієнтацію на практику так як, під час вивчення основ алгоритмізації акцент робиться на практичних завданнях і застосуванні теоретичних знань для розв'язання реальних задач. Використання інструментів програмування (Scratch, Python) дозволяє учням не тільки розуміти теорію, але й відчути себе творцями готових алгоритмічних рішень.

Розвинуто гнучкість і адаптивність навчання, тому що методи навчання повинні враховувати різний рівень підготовки учнів. Використання індивідуального підходу, диференціація завдань і створення умов для самостійного навчання забезпечують максимальну ефективність процесу.

Набуття ключових компетенцій

Модуль «Основи алгоритмізації» спрямований не тільки на розвиток технічних знань, але й на формування таких компетенцій, як логічне мислення, здатність до аналізу, креативність, уміння працювати в команді. Це готує учнів до майбутньої професійної діяльності.

Використання сучасних технологій

Застосування візуальних середовищ програмування (наприклад, Scratch) на початкових етапах і переходу до текстового програмування в Python сприяє поступовому, послідовному засвоєнню матеріалу. Інтерактивні платформи й онлайн-інструменти стимулюють зацікавленість учнів.

Проблемно-орієнтоване та дослідницьке навчання

Методи навчання, орієнтовані на вирішення реальних задач і проведення досліджень, формують у учнів здатність мислити критично, самостійно шукати рішення й застосовувати алгоритми в нестандартних ситуаціях.

Значення проектної діяльності

Проектна робота учнів дозволяє інтегрувати різні знання та навички, забезпечуючи створення кінцевого продукту. Цей метод сприяє розвитку креативності й інноваційного мислення.

РОЗДІЛ 4. ЕКСПЕРЕМЕНТАЛЬНА ПЕРЕВІРКА МЕТОДИКИ ПІДГОТОВКИ ФАХІВЦІВ ЦИФРОВИХ ТЕХНОЛОГІЙ ДО ВИКЛАДАННЯ АЛГОРИТМІЗАЦІЇ

Експериментальна перевірка є важливим етапом у впровадженні нової методики підготовки фахівців цифрових технологій до викладання алгоритмізації. Її метою є визначення ефективності методики через оцінювання її впливу на професійні знання, вміння та навички майбутніх викладачів.

Цілі експериментальної перевірки

Оцінка рівня теоретичної підготовки фахівців з алгоритмізації.

Аналіз сформованих практичних навичок у розробці та реалізації алгоритмів.

Перевірка ефективності методик викладання та їх впливу на підготовку учнів.

Визначення рівня інноваційності та інтерактивності навчального процесу.

Внесення рекомендацій щодо удосконалення методики на основі результатів експерименту.

Етапи експериментальної перевірки

1. Підготовчий етап

Формування вибірки учасників:

До вибірки входять фахівці цифрових технологій, які беруть участь у підготовчій програмі.

Учасники розділяються на контрольну (традиційний підхід) та експериментальну (нова методика) групи.

Розробка навчальної програми:

Деталізація змісту модулю «Основи алгоритмізації».

Визначення критеріїв оцінювання результатів.

Підготовка інструментів оцінювання:

Тестові завдання для теоретичної перевірки знань.

Практичні кейси для оцінки навичок реалізації алгоритмів.

Анкети та опитувальники для оцінки задоволеності учасників.

2. Основний етап

Навчання за експериментальною методикою:

Проведення занять із використанням інтерактивних методів, проєктного підходу та дослідницьких завдань.

Використання інструментів програмування (Scratch, Python) для практичної роботи.

Контроль навчального прогресу:

Регулярне тестування знань і навичок.

Аналіз успішності виконання практичних завдань.

Спостереження за викладацькою діяльністю:

Оцінювання викладання учасниками модельних уроків.

Аналіз ефективності застосування методичних матеріалів.

3. Завершальний етап

Оцінка результатів:

Порівняння результатів контрольної та експериментальної групи за визначеними критеріями.

Аналіз статистичних даних щодо засвоєння матеріалу.

Збір зворотного зв'язку:

Інтерв'ю та анкетування учасників щодо зручності та ефективності методики.

Рефлексія викладачів та учнів про навчальний процес.

Підготовка звіту:

Висновки щодо ефективності методики.

Пропозиції щодо її вдосконалення.

Методи оцінювання ефективності

Тестування знань:

Теоретичні тести, які перевіряють розуміння алгоритмів, структур даних і принципів програмування.

Практичні завдання:

Виконання завдань зі створення алгоритмів та їх реалізації в програмному середовищі.

Аналіз правильності та ефективності рішень.

Методичні проєкти:

Розробка навчальних матеріалів для учнів.

Оцінювання їхньої відповідності стандартам викладання та інноваційності.

Рефлексивний аналіз:

Оцінка задоволеності учасників програмою.

Самоаналіз учасниками їхнього прогресу.

Порівняльний аналіз:

Порівняння результатів контрольної та експериментальної груп.

Очікувані результати експерименту

Підтвердження ефективності методики:

Підвищення рівня теоретичних знань і практичних навичок у викладачів.

Покращення здатності до викладання алгоритмізації.

Розробка рекомендацій:

Вдосконалення навчальної програми з урахуванням отриманих результатів.

Популяризація нових підходів:

Поширення інтерактивних і проєктних методів навчання серед викладачів.

Формування банку практичних матеріалів:

Напрацювання кейсів, алгоритмів та навчальних матеріалів для подальшого використання.

4.1. Організація експериментальної перевірки

Організація експериментальної перевірки передбачає чітке структурування процесу дослідження для визначення ефективності методики. Цей процес включає декілька послідовних етапів, кожен із яких спрямований на досягнення визначених цілей.

Цілі організації експерименту

Оцінити, наскільки нова методика сприяє підвищенню професійної компетентності фахівців цифрових технологій.

Визначити вплив методики на здатність викладати алгоритмізацію з використанням сучасних інструментів і методів.

Розробити рекомендації для вдосконалення методичних матеріалів на основі отриманих результатів.

Етапи організації експериментальної перевірки

1. Підготовчий етап

Формулювання гіпотези:

Гіпотеза: нова методика підготовки фахівців цифрових технологій до викладання алгоритмізації є ефективнішою порівняно з традиційними підходами.

Вибір учасників:

Формуються дві групи: контрольна (за традиційною методикою) і експериментальна (за новою методикою).

Учасниками є викладачі цифрових технологій або студенти педагогічних спеціальностей, які готуються до викладання алгоритмізації.

Розробка матеріалів:

Методичні посібники для викладачів.

Навчальні кейси, тести та практичні завдання.

Розробка критеріїв оцінювання:

Критерії: рівень теоретичних знань, практичні навички, методичні компетентності, здатність до інновацій у викладанні.

2. Основний етап

Навчання обох груп:

Контрольна група проходить підготовку за традиційною методикою (лекційно-практичний підхід).

Експериментальна група навчається за новою методикою, що включає інтерактивні методи, гейміфікацію, використання проєктів та дослідницький підхід.

Виконання практичних завдань:

Обидві групи розв'язують аналогічні задачі, що включають алгоритмічне мислення, складання блок-схем і програмування (у Scratch або Python).

Моніторинг процесу:

Регулярна фіксація прогресу учасників.

Спостереження за використанням методичних інструментів у викладанні.

3. Завершальний етап

Збір даних:

Проведення підсумкового тестування для оцінки теоретичних знань.

Аналіз виконаних практичних завдань.

Оцінка розроблених учасниками навчальних матеріалів і методик.

Анкетування та інтерв’ю:

Вивчення думок учасників щодо ефективності підходів.

Аналіз задоволеності методикою.

Статистичний аналіз:

Порівняння результатів контрольної та експериментальної груп.

Виявлення залежності між методикою навчання та показниками компетентності.

Розробка рекомендацій:

На основі отриманих результатів розробляються рекомендації для вдосконалення методики.

Критерії ефективності методики

Рівень знань:

Наскільки глибоко учасники засвоїли основи алгоритмізації.

Рівень практичних навичок:

Здатність створювати алгоритми, складати програми та використовувати їх у викладанні.

Рівень методичних компетенцій:

Якість розроблених навчальних матеріалів і способів викладання.

Задоволеність учасників:

Оцінка зручності та ефективності методики з точки зору учасників.

Очікувані результати організації експерименту

Підтвердження ефективності нової методики:

Підвищення рівня підготовки фахівців до викладання алгоритмізації.

Формування практичних рекомендацій:

Вдосконалення методики на основі отриманих результатів.

Популяризація інноваційних підходів:

Впровадження інтерактивних і проєктних методів у підготовку викладачів.

4.2. Результати експериментальної перевірки методичної підготовки фахівців цифрових технологій

Результати експериментальної перевірки методики підготовки фахівців цифрових технологій до викладання алгоритмізації є важливим етапом для визначення ефективності нової методики навчання. Вони дозволяють оцінити вплив нових підходів на рівень професійної підготовки викладачів, їх здатність передавати знання учням, а також забезпечують основу для коригування і вдосконалення методичної роботи.

1. Загальна характеристика результатів

Експеримент проводився з двома групами учасників: контрольної (де використовувалася традиційна методика підготовки) і експериментальної (де застосовувалися нові методи і підходи до навчання). Аналіз результатів показав різницю в ефективності методик, яку можна оцінити за кількома основними критеріями.

1.1. Результати теоретичних тестів

Контрольна група показала помірний рівень засвоєння теоретичних знань з алгоритмізації. Хоча учасники здобули необхідні знання, їх здатність застосовувати їх на практиці була обмежена традиційними методами навчання.

Експериментальна група демонструвала значно вищі результати в тестах з теоретичних знань. Учні, які навчалися за новою методикою, продемонстрували кращу здатність до аналізу та формулювання алгоритмічних задач.

1.2. Практичні завдання

У контрольній групі значна частина учасників зазнала труднощів під час виконання практичних завдань, зокрема в частині розробки алгоритмів і програмування. Багато учасників робили помилки у структурі алгоритмів або не могли оптимізувати рішення.

У експериментальній групі результати практичних завдань були значно кращими. Учасники краще розуміли структуру алгоритмів, могли застосувати різні методи оптимізації та інтуїтивно вибирати найбільш ефективні підходи до розв'язання задач.

1.3. Оцінка методичних компетенцій

В контрольній групі учасники мали обмежене розуміння, як застосувати алгоритмізацію в реальному навчальному процесі. Вони зіштовхувалися з труднощами у викладанні складних тем та застосуванні сучасних технологій.

У експериментальній групі учасники продемонстрували більш високий рівень методичних компетенцій. Вони успішно адаптували нові технології до процесу навчання, використовували інтерактивні методи, підходи до проєктної діяльності та мали високий рівень готовності до викладання алгоритмізації.

1.4. Задоволеність учасників методикою

В контрольній групі більшість учасників відзначали недостатню мотивацію до навчання через традиційні підходи та відсутність інструментів для самостійної роботи.

В експериментальній групі учасники висловили високу задоволеність методикою навчання, відзначаючи використання інтерактивних методів, роботу над реальними проєктами та можливість застосувати отримані знання на практиці.

2. Порівняння результатів контрольної та експериментальної груп

2.1. Рівень теоретичної підготовки

В контрольній групі 60% учасників досягли середнього рівня знань, 30% – базового рівня, 10% – не змогли набрати мінімальних балів.

В експериментальній групі 80% учасників досягли високого рівня теоретичних знань, 15% – середнього рівня, 5% – базового.

2.2. Практична підготовка

В контрольній групі лише 50% учасників змогли ефективно вирішити задачі з програмування, в той час як у експериментальній групі 85% учасників успішно виконали практичні завдання та продемонстрували навички оптимізації алгоритмів.

2.3. Методичні навички

В контрольній групі лише 40% учасників продемонстрували високий рівень методичних компетенцій, тоді як у експериментальній групі 75% учасників показали високий рівень методичних навичок, що дозволило їм ефективно застосовувати нові підходи до викладання алгоритмізації.

3. Висновки за результатами експерименту

Ефективність нової методики: Результати експерименту показали значне покращення рівня теоретичних знань, практичних навичок та методичних компетенцій учасників експериментальної групи. Нові підходи до навчання, такі як інтерактивні методи та проєктно-орієнтовані завдання, мали позитивний вплив на рівень підготовки майбутніх викладачів алгоритмізації.

Переваги нової методики:

Вищий рівень мотивації учасників.

Кращі результати в практичній діяльності.

Більш ефективне застосування сучасних технологій у викладанні.

Рекомендації:

Включення більшої кількості інтерактивних елементів і практичних завдань у навчальні програми.

Подальше удосконалення методичних матеріалів і посібників для викладачів.

Розширення використання технологій, таких як програмування на Python і Scratch, для створення алгоритмічних рішень.

4. Подальші кроки

Удосконалення навчальних програм на основі результатів експерименту.

Розширення використання нових методик у ширшому контексті підготовки фахівців, а також поширення результатів дослідження серед викладачів та розробка рекомендацій для інших освітніх установ.

Висновки до розділу 4

Досягнута висока ефективність нової методики:

Експериментальна перевірка методики підготовки фахівців цифрових технологій до викладання алгоритмізації підтвердила її високу ефективність порівняно з традиційними підходами. Нові методи, які включають інтерактивні технології, проєктно-орієнтовані завдання та гейміфікацію, позитивно вплинули на результативність навчання.

Покращено рівень теоретичної та практичної підготовки:

Учасники експериментальної групи продемонстрували значно вищий рівень теоретичних знань та практичних навичок. Вони мали кращу здатність до аналізу і вирішення алгоритмічних задач, а також до створення і оптимізації програмних рішень. Це свідчить про ефективність нових підходів у підготовці викладачів.

Збільшено кількість методичних компетенцій:

Нові методи навчання дозволили учасникам експериментальної групи не лише покращити свої технічні знання, а й розвинути методичні компетенції, що є ключовим елементом для викладання алгоритмізації. Вони змогли ефективно застосовувати сучасні технології у викладанні, створюючи інтерактивні та проєктно-орієнтовані навчальні матеріали.

Високий рівень задоволеності учасників:

Учасники експериментальної групи виявили високу задоволеність новою методикою навчання. Вони відзначали, що інтерактивні елементи, можливість працювати над реальними проєктами та використання нових технологій сприяли підвищенню їхньої мотивації та інтересу до навчання.

Необхідність подальшого вдосконалення методики: Хоча результати експерименту продемонстрували позитивний ефект від нової методики, є необхідність у подальшому вдосконаленні навчальних матеріалів та методичних інструментів. Зокрема, рекомендується розширення використання програмування на сучасних мовах, таких як Python та Scratch, а також інтеграція додаткових інтерактивних технологій для ще більш ефективного навчання.

Поширення досвіду та рекомендацій:

На основі результатів експерименту можна зробити висновок про доцільність поширення нової методики серед інших навчальних закладів, що готують фахівців з цифрових технологій. Розробка додаткових рекомендацій для викладачів щодо використання нових підходів може сприяти підвищенню якості навчання в цій галузі.

ЗАГАЛЬНІ ВИСНОВКИ

Висновки про виконання курсової роботі на тему "Методика вивчення основ алгоритмізації" включають наступне:

1. Вивчення основ алгоритмізації є важливим етапом у процесі навчання програмуванню та розвитку комп'ютерної мислення. Глибоке розуміння алгоритмів дозволяє студентам ефективно вирішувати складні задачі та розробляти ефективні програми.

2. Використання комбінованої методики, яка поєднує теоретичні основи, практичні вправи та візуалізацію алгоритмів, може бути ефективним підходом до вивчення основ алгоритмізації. Такий підхід дозволяє студентам розуміти концепції алгоритмів, відпрацьовувати їх у реальних сценаріях та сприймати їх візуально.

3. Онлайн-курси та платформи з програмування, такі як Coursera, edX, HackerRank та Codecademy, надають багато ресурсів та завдань, які допомагають студентам поглиблювати свої знання алгоритмізації. Ці ресурси можуть бути використані як додаткові джерела вивчення та практики.

4. Важливо включати практичні завдання та проекти в методику вивчення алгоритмізації. Це дозволяє студентам застосовувати свої знання на практиці, вирішувати реальні задачі та розвивати навички програмування.

5. Кількість практичних вправ та завдань для відпрацювання навичок. Це дозволить їм не лише теоретично ознайомитися з алгоритмами, але й набути практичний досвід їх застосування.

Застосування методики "від простого до складного" може бути ефективним підходом до вивчення алгоритмізації. Починаючи з простих алгоритмів і поступово переходячи до більш складних, студенти можуть поетапно розширювати свої знання та розвивати навички розробки алгоритмів.

6. Комунікація та спільна робота між студентами також може бути корисною в методиці вивчення алгоритмізації. Взаємне обговорення алгоритмів, вирішення задач у групі та обмін думками допомагають розширити розуміння та погляди студентів на рішення проблем.

Загалом, методика вивчення основ алгоритмізації повинна поєднувати теоретичний матеріал, практичні вправи, візуалізацію алгоритмів та можливість спільної роботи. Це допоможе студентам засвоїти концепції алгоритмів та розвинути навички їх застосування у практичних ситуаціях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Альфред Ахо, Моніка Лам, Раві Сеті, Джеффрі Ульман. "Компілятори: принципи, техніки та інструменти". Київ: Друк-Сервіс, 2017.

2. Брайан В. Керніган, Деніс М. Рітчі. "Мова програмування С". Київ: Видавничий дім "Освіта", 2016.

3. Баженов В. А. Інформатика. Комп’ютерна техніка. Комп’ютерні технології : Підручник. К. : Каравела, 2016. 592 с.

4. Барбара Лисков, Джоель Мозес. "Алгоритми. Проектування та аналіз." Київ: Видавнича група "БІОС", 2014.

5. Бережна О. Б. Інформатика та комп’ютерна техніка. 1 частина : Навч.посіб. Х. : ХНЕУ ім. С. Кузнеця, 2017. 164 с.

6. Бєлов Ю. А., Карнаух Т. О., Коваль Ю. В., Ставровський А. Б. Вступ до програмування мовою С++. Організація обчислень : навч. посіб./ Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. К. : Видавничо-поліграфічний центр “Київський університет”, 2012. 175 с.

7. Бублик В.В. Об’єктно-орієнтоване програмування : підручник. К. : ІТкнига, 2015. 624 с.

8. Вільям С. Дейвіс. "Комп'ютерні науки і програмування." Київ: Видавнича група "БІОС", 2017.

9. Вільям Сталлінгс. "Основи комп'ютерних мереж". Київ: Видавничий дім "Освіта", 2019.

10. Володіна І. Л. Основи інформатики. К. : Видавничий центр «Гімназія», 2012. 290 с.

11. Глинський Я. М. Інформатика. Практикум з інформаційних технологій : Навч. посіб. Тернопіль : Підручники і посібники, 2014. 304 с.

12. Грицюк Ю.І., Рак Т.Є. Програмування мовою С++ : навч. посіб. Львів: Вид-во Львівського ДУ БЖД, 2011. 292 с.

13. Дебора Ромінг. "Вступ до інформатики." Київ: Видавничий дім "Академкнига", 2015.

14. Дибкова Л. М. Інформатика і комп’ютерна техніка : Навч. посіб. К. : Академвидав. 2012. 463 с.

15. Дж. Гленн Брукcхарт, Девід А. Ауе. "Загальний курс інформатики". Київ: Видавничий дім "Освіта", 2018.

16. Дональд Кнут. "Мистецтво програмування". Київ: Видавництво Солон, 2018.

17. Дональд Е. Кнут. "Мови програмування: принципи та практика". Київ: Видавництво Солон, 2017.

18. Джон Макклінток, Кріс Байгл. "Магія комп'ютерної науки." Київ: Видавництво "К.І.С.", 2013.

19. Дебора Ромінг. "Вступ до інформатики." Київ: Видавничий дім "Академкнига", 2015.

20. Йорг Рейнсдорф, Петер Пфайффер. "Інформатика. Вступ до науки про обчислення." Київ: Видавнича група "БІОС", 2012.

21. Жуковський С.С., Вакалюк Т.А. Об‘єктно-орінтоване програмування мовою С++ : навч.-метод. посіб. Житомир : Вид-во ЖДУ, 2016. 100 с.

22. Зубенко В.В., Омельчук Л.Л. Програмування. Поглиблений курс. К.: Видавничо-поліграфічний центр “Київський університет”, 2011. 623 с.

23. Майкл Г. Сейдж. "Інформатика." Київ: Видавничий дім "Академкнига", 2013.

24. Майкл Т. Гудріч. "Алгоритми та структури даних". Київ: Друк-Сервіс, 2019.

25. Малчевський В., Щирбул О. Особливості методичної підготовки фахівців цифрових технологій до викладання освітнього модуля «основи алгоритмізації». Всеукраїнський збірник наукових праць студентів, аспірантів, викладачів і вчителів закладів загальної середньої освіти / за заг. ред.: М.І. Садового, О.М. Щирбула. Кропивницький: ІВ ЦДУ ім. В. Винниченка, 2024. Вип.12. С.88-92.

26. Ніклаус Вірт. "Основи об'єктно-орієнтованого програмування в Java". - Київ: Видавничий дім "Освіта", 2018.

27. Стівен Скієна. "Алгоритмічний тренінг: Практичний посібник з розробки ефективних алгоритмів." Київ: Видавнича група BHV, 2011.

28. Томас Кормен, Чарльз Лейзерсон, Рональд Райвест, Кліффорд Штайн. "Вступ до алгоритмів". Київ: Диносавр, 2019.

29. Томас Х. Кормен, Чарльз І. Лейзерсон, Рональд Л. Райвест, Кліффорд Штайн. "Алгоритми: побудова та аналіз". Київ: Диносавр, 2018.

30. Удженіо Россі, Фернандо Массі. "Інформатика: вступ до науки про обчислення." Київ: Видавництво "К.І.С.", 2014.

31. Удженіо Англарі. "Основи програмування та алгоритми." Київ: Видавнича група "БІОС", 2018.