УДК 004.421

І. Вітковська, Ю. Крамар

ФОРМУВАННЯ АЛГОРИТМІЧНОГО МИСЛЕННЯ СТУДЕНТІВ ІТ-СПЕЦІАЛЬНОСТЕЙ

Анотація: У статті розкрито роль навчальної дисципліни «Алгоритми та структури даних» (АСД) у формуванні алгоритмічного мислення студентів ІТ-спеціальностей. Проаналізовано підходи до організації освітнього процесу, що сприяють адаптації до неоднорідного рівня підготовки студентів та формуванню алгоритмічного стилю мислення.

Мета статті – описати особливості формування алгоритмічного мислення, інтеграцію мультимодальних і візуальних інструментів, а також адаптивних технологій навчання.

Ключові слова: алгоритмічне мислення, візуалізація, адаптивне навчання, програмування, структура даних.

Вступ

Дисципліна «Алгоритми та структури даних» є фундаментальною для підготовки спеціалістів в галузі комп’ютерних наук і програмної інженерії. Вона формує основи алгоритмічного мислення, навички оптимізації рішень та аналітичного підходу до побудови програмного забезпечення. В умовах цифрової трансформації освіти зростає потреба в адаптації змісту, методів і засобів викладання цієї дисципліни до потреб нових поколінь студентів і вимог ІТ-індустрії.

Роль дисципліни в системі підготовки фахівців

«Алгоритми та структури даних» є теоретичною та практичною основою сукупності знань та вмінь, що формують профіль фахівця в галузі інженерії програмного забезпечення, та знайомить їх з базовими концепціями побудови структур даних та пов’язаних з ними алгоритмів. Вивчення дисципліни надає фундаментальні знання, необхідні для розуміння принципів побудови, ефективності та масштабованості програмного забезпечення, дозволяє:

– аналізувати часову та просторову складність рішень;

– будувати ефективні моделі зберігання та обробки даних;

– розуміти, чому певні підходи є швидшими або стабільнішими;

– закладати фундамент для подальших курсів (ООП, бази даних, системне програмування, ШІ тощо).

Методологічні особливості викладання

Неоднорідність рівня знань в студентських групах є важливим фактором, який потрібно враховувати при викладанні дисципліни «Алгоритми та структури даних». Студенти в групах мають різний рівень загальної підготовки, досвіду в програмуванні, знання математичних основ та здатності до абстрактного мислення.

За статистикою, у більшості університетів лише 20-40% студентів мають попередній досвід програмування[9], а решта вивчає ці навички лише в рамках поточного курсу. Це означає, що важливо використовувати різноманітні методи для задоволення потреб як новачків, так і досвідчених програмістів.

Програми багатьох університетів включають математичні дисципліни, але не всі студенти мають однаковий рівень знань у таких областях, як дискретна математика, теорія графів та комбінаторика, що є важливими для розуміння структури даних та алгоритмів [9].

Студенти можуть мати різні рівні мотивації до вивчення складних тем, таких як алгоритми. Часто це пов'язано з їхніми майбутніми кар'єрними планами (деякі студенти обирають програмування як основну кар'єру, інші - як додаткову навичку) [17].

Неоднорідність — системне явище, яке потребує адаптивних підходів у навчанні: гібридного навчання, змішаних методів, адаптивних платформ та індивідуальних траєкторій навчання.

Це може бути викликом для викладача, оскільки необхідно адаптувати методи навчання до різних типів студентів для забезпечення ефективного засвоєння матеріалу.

Компетентнісний підхід до навчання

Основною метою підходу є не лише передача знань, а формування здатності аналізувати, моделювати та реалізовувати алгоритми. Орієнтація студентів на розвиток алгоритмічного мислення, логіки та вміння самостійно розв’язувати задачі.

Когнітивна теорія подвійного кодування (Dual Coding Theory), запропонована Алланом Пайвіо, стверджує, що інформація запам’ятовується і обробляється ефективніше, якщо [14] подається одночасно через два канали:

– вербальний (текст, мова);

– невербальний/візуальний (зображення, діаграми, схеми).

Два незалежні канали обробляють інформацію паралельно. Якщо матеріал закодований обома способами – ймовірність його запам’ятовування зростає. Візуальні образи викликають асоціації, а текст – структурне мислення. Разом вони підсилюють когнітивну інтеграцію.

Для ефективного засвоєння студентами матеріалу з дисципліни АСД необхідно вирішити наступне.

 ISSN 1560-8956

1. Полегшити перехід від абстракції до візуального мислення. Алгоритми часто є абстрактними. Подання через блок-схеми, анімації або графічні моделі активує візуальні канали — що покращує розуміння, особливо для новачків.

2. Сприяти формуванню подвійних когнітивних слідів. Паралельне подання текстової інформації (опис умови, постановка задачі) і візуальної (блок схема) забезпечує більшу стійкість до забування.

3. Зменшити когнітивне навантаження. Візуалізація знижує кількість абстрактних операцій у робочій пам’яті. Студенти краще фокусуються на суті алгоритму, а не на синтаксичній формі.

4. Підтримати мультісенсорність навчання для студентів з різними стилями навчання (візуальний, аудіальний, кінестетичний).

Для ефективного засвоєння студентами матеріалу з дисципліни необхідно розвивати алгоритмічне мислення[1] , це здатність:

- аналізувати проблему;
- виокремлювати ключові дії;
- будувати послідовність кроків для досягнення цілі;
- перевіряти логіку дій;
- шукати універсальні рішення, що можна застосувати до подібних задач.

Розвиток алгоритмічного мислення — це поступове формування вміння мислити послідовно, логічно та структуровано, щоб ставити задачі, розбивати їх на підзадачі й знаходити точні способи розв’язання (алгоритми). Процес розвитку можна поділити на декілька етапів (табл.1)

Таблиця 1.

Процес розвитку алгоритмічного мислення

Етап	Характеристика
Початковий	Просте слідування інструкціям
Розвивальний	Усвідомлення логіки, спроби аналізу
Алгоритмічний	Самостійна побудова алгоритмів
Оптимізаційний	Пошук кращих/універсальних рішень

Конструктивістський підхід до навчання.

Підхід передбачає, що студенти самостійно добувають знання через діяльність, дослідження та візуалізацію. Замість пасивного засвоєння готових алгоритмів — активне конструювання рішень за допомогою візуальних інструментів (блок-схеми, візуалізатори).

Системна візуалізація алгоритмічного мислення допомагає студентам краще розуміти та опрацьовувати алгоритми й логіку програмування[10] за допомогою візуальних інструментів (табл.2). У навчанні цей підхід поєднує схематичне подання, моделювання та покрокову демонстрацію виконання алгоритмів, що сприяє розвитку алгоритмічного мислення.

1. Представлення алгоритму як системи: описуємо не просто окремі кроки, а взаємозв’язані елементи, які працюють разом (вхід → обробка → вихід). Важливо продемонструвати логіку взаємодії, а не лише окремі інструкції.

2. Інтеграція з текстовим кодом (поєднання візуального і абстрактного рівнів): спочатку – постановка задачі, псевдокод, блок-схема, код на мові програмування. Важливо продемонструвати, як алгоритм виглядає в різних формах.

3. Подальший розвиток алгоритмічного мислення: студенти навчаються:
   - декомпозувати (розкладати) задачу на послідовні дії;
   - бачити зв’язки між діями;
   - передбачати результат алгоритму.

Застосування блок-схем, дерев, графів у вигляді інтерактивних візуальних схем допомагає краще засвоїти структури (heap, tree, stack), підвищити коефіцієнт розуміння[12]. Це відповідає когнітивній теорії подвійного кодування сприяє глибшому запам’ятовуванню та розумінню, забезпечує сильнішу когнітивну інтеграцію.

За результатами досліджень[9] , група студентів, що використовувала блок-схеми показала на 25% кращі результати при аналізі логіки алгоритмів у порівнянні з тією, що працювала лише з кодом та зниження кількості логічних помилок на 40%.

Таблиця 2.

Ефективність блок-схем у вивченні алгоритмів

Спосіб навчання	**Середній бал**	**Середня кількість логічних помилок**	**Рівень розуміння (самооцінка, /5)**
Лише текст (код)	65	3,8	2,9
Текст + блок-схема	82	1,9	4,1

Студенти, які використовували текст та блок-схеми, в середньому мали на 17 балів вищі результати (82 проти 65). Кількість помилок вдвічі зменшилася (1,9 проти 3,8), що свідчить про краще розуміння структури алгоритмів. Самооцінка зросла з 2,9 до 4,1 із 5 — студенти краще усвідомлюють власне розуміння завдяки візуальній підтримці.

 ISSN 1560-8956

Системна візуалізація алгоритмічного мислення

Гарний результат з ефективного засвоєння навчального матеріалу дає поєднання блокових, структурних та інтерактивних візуалізацій, з поступовим переходом до текстового коду. Поєднання різних типів візуалізації з поступовим переходом до текстового коду створює оптимальні умови для засвоєння матеріалу, незалежно від стартового рівня студента. Такий підхід відповідає як психологічним закономірностям навчання, так і методичним вимогам сучасної ІТ-освіти.

Приклади візуальних інструментів наведено в табл.3.

Таблиця 3.

Візуальні інструменти

Назва	**Призначення**	**Мови програмування**	**Рівень складності**
Flowgorithm	Створення та виконання блок-схем	C++, Java, Python, C#, псевдокод	Початковий
Visualgo	Візуалізація алгоритмів та структур даних	Не програмує напряму (анімація)	Середній
Blockly	Блочне програмування з текстовим кодом	JavaScript, Python, Lua тощо	Початковий –середній
App Inventor	Створення мобільних застосунків	Блоки + Java-подібна логіка	Середній
Python Tutor	Візуалізація коду з поясненням змінних	Python, Java, JavaScript, C/C++	Середній
AlgoExpert	Підготовка до інтерв’ю, глибоке розуміння	Python, JavaScript, C++, Java тощо	Середній–просунутий
Replit + Mermaid	Візуалізація коду/логіки через діаграми	Будь-яка (основне — текстовий код)	Середній–просунутий

Для курсу АСД можна рекомендувати використовувати комбінацію візуальних інструментів, яка забезпечує когнітивну послідовність і багаторівневе сприйняття матеріалу, що особливо важливо при вивченні алгоритмів і структур даних (табл.4).

Таблиця 4.

Комбінація візуальних інструментів

Розділ	**Інструмент**	**Для чого використовувати**
Вступ до алгоритмів	Flowgorithm	Побудова блок-схем, покрокове виконання алгоритмів
	Python Tutor	Візуалізація виконання простих програм
	Blockly (для вправ)	Інтерактивне створення простих алгоритмів
Основи структур даних	Visualgo	Анімація роботи з масивами, стеками, чергами
	Python Tutor	Пояснення змінних, списків, стекових викликів
	Replit + Mermaid	Побудова логічних діаграм структур вручну

Закінчення табл. 4

Розділ	Інструмент	Для чого використовувати
Сортування і пошук	Visualgo	Покрокова візуалізація сортувань і пошуку
	Python Tutor	Перевірка кроків сортування на коді
	Replit	Реалізація сортувань вручну, тестування
Дерева і графи	Visualgo	Анімація обходів дерев, графів, алгоритмів Dijkstra/Kruskal
	Replit + Mermaid	Створення схем графів, вручну моделювання структур
	Python Tutor	Трасування рекурсивних обходів дерев
Перевірка знань	AlgoExpert (або LeetCode/GeeksforGeeks)	Завдання + візуальні пояснення
	Replit	Самостійна реалізація, тестування алгоритмів

Такій підхід дозволить здійснити покроковий супровід — від новачка до впевненого студента, спростити опанування складних алгоритмічних конструкцій.

Дослідницько-проєктна інтеграція

Базується на потребі перевести абстрактні знання в реальні практичні навички та розвинути в студентів аналітичне, системне і творче мислення. Проєктні задачі змушують студентів думати глибше, ніж на стандартних прикладах. Виникає потреба обирати оптимальні структури даних, оцінювати складність, комбінувати алгоритми. Знання перетворюються в реальні програмні рішення (бот, калькулятор графів, емулятор черги тощо). Студенти виконують повний життєвий цикл розробки програмного забезпечення — від постановки задачі до реалізації та тестування. Під час колективних проєктів студенти розвивають навички з комунікації, управління часом, планування. Презентація рішень тренує комунікативну функція мовлення і критичне мислення. Можна поєднувати АСД з математикою, фізикою, біологією (моделювання процесів), економікою (алгоритми оптимізації), інженерією (керування чергами, графами), зміцнюючи междісциплінарную інтеграцію. Інтеграція проєктної та дослідницької діяльності у курсі АСД дасть можливість активізувати пізнавальну діяльність студентів, зробити абстрактні теми практично значущими, підготувати до професійної діяльності у сфері розробки та аналізу алгоритмів.

Впровадження адаптивного алгоритмічного навчання

Адаптивне навчання — це освітній підхід, який динамічно підлаштовує навчальний контент, темп та методи до індивідуальних потреб кожного студента на основі аналізу його знань, помилок та стилю навчання. Завдяки правильному підходу до різнорівневої підготовки студентів викладання алгоритмів може стати більш

ефективним. Важливо враховувати різний рівень знань і вмінь у групах, застосовувати адаптивні методи навчання, створити можливості для колективної роботи та використовувати різні інструменти для закріплення матеріалу. Ці заходи дозволяють кожному студенту знайти свою траєкторію навчання та досягти успіху у вивченні складних тем.

Основними принципами адаптивного навчання[2] є наступні.

1. Індивідуалізація. Кожен студент отримує навчальний маршрут, що враховує його стартові знання, прогрес і помилки. Наприклад, студент, який не розуміє рекурсію, отримає більше базових прикладів перед переходом до складних задач.

2. Безперервна діагностика. Система або викладач постійно збирає дані: правильність відповідей, час виконання, типи помилок – і коригує зміст у реальному часі.

3. Реагування на зони найближчого розвитку. Контент подається так, щоб завдання були різної ступені складності, коригувалися під рівні знань студентів.

4. Гнучкість у форматах подання. Одна тема може подаватися через відео, інтерактивні візуалізації або тести – залежно від того, який формат краще засвоюється студентом.

5. Мотиваційна підтримка. Адаптивні системи використовують ігрові елементи, нагороди, сповіщення про досягнення, щоб зберігати зацікавленість студента.

Технології для впровадження адаптивного навчання

Існуючі технології для адаптивного навчання умовно можна класифікувати наступним чином.

1. Адаптивні платформи навчання. Використання спеціалізованих систем і платформ, таких як Knewton, Smart Sparrow або ALEKS, дозволяє автоматизувати адаптивний процес і підлаштовувати навчальні матеріали під індивідуальні потреби студентів.

2. Системи на основі штучного інтелекту (ШІ). Платформи, які використовують алгоритми ШІ для оцінки прогресу студентів і надають рекомендації щодо подальших кроків навчання, допомагають створити адаптивне навчальне середовище.

Приклади адаптивних систем наведено в таблиці 5.

Адаптивне алгоритмічне навчання є важливим інструментом для підвищення ефективності навчання, особливо у таких складних предметах, як «Алгоритми та структури даних». Воно дозволяє зменшити розрив між рівнями підготовки студентів, персоналізувати навчання та покращити загальну мотивацію студентів. Дає можливість побудувати динамічні освітні траєкторії, де матеріал подається поступово – від простого до складного, а повторення та підкріплення знань відбувається вчасно і в тій формі, яка найбільш ефективна для конкретного студента. Особливо важливо це у темах з високою

когнітивною складністю – таких як сортування, рекурсія, робота з деревами та графами. В цих розділах адаптивне навчання дозволяє значно зменшити когнітивне навантаження, підвищити розуміння алгоритмів і зменшити кількість помилок у реалізації.

Таблиця 5.

Приклади адаптивних систем

№	Технологія / Платформа	Призначення (ціль)	Рівень адаптивності	Приклад використання в АСД
1	Moodle + плагіни адаптації[13]	Персоналізація змісту, адаптивні тести	Високий	Генерація індивідуальних задач за темою "Черги"
2	Khan Academy[11]	Автоматичне підлаштування темпів навчання	Середній	Теоретичні відео з перевіркою розуміння структур
3	Smart Sparrow[16]	Побудова адаптивних уроків із гілкуванням логіки	Високий	Створення модуля "Сортування: вибір алгоритму"
4	EdApp[7]	Мікронавчання з адаптацією під мобільні пристрої	Середній	Короткі вправи на розпізнавання алгоритмів
5	Python Tutor[15]	Візуалізація кроків коду зі зворотним зв’язком	Низький	Аналіз рекурсивного алгоритму (наприклад, обходу дерева)
6	CodeCombat / CodinGame[5]	Ігрове адаптивне середовище для програмування	Середній	Закріплення основних принципів через ігрові задачі
7	H5P (інтерактивний контент)[8]	Інтерактивні вправи, що реагують на відповіді	Середній	Побудова гілки рішень для пошуку помилок в алгоритмах

В таблиці 6 наведено рекомендації щодо використання адаптивних систем при викладанні курсу АСД.

Таблиця 6.

Використання адаптивних систем в АСД.

Тема курсу АСД	Рекомендована технологія	Пояснення
Вступ, логічне мислення, умовні оператори	CodeCombat / Khan Academy	Для мотивації та поступового входу в складність
Сортування, вибір алгоритмів	Smart Sparrow / Moodle + H5P	Дає змогу адаптувати траєкторію залежно від типових помилок
Стек, черга, дерева	Python Tutor / Flowgorithm / Visualgo	Візуалізація роботи структур, рекурсії, обходів
Підсумкове оцінювання	Moodle з адаптивними тестами	Контроль знань з адаптацією до рівня складності студента

 ISSN 1560-8956

Висновки

Формування алгоритмічного мислення є ключовим завданням у підготовці майбутніх ІТ-фахівців, оскільки саме воно лежить в основі здатності до аналітичного мислення, вирішення складних задач і створення ефективних програмних рішень. Дисципліна «Алгоритми та структури даних» виконує провідну роль у цьому процесі, забезпечуючи фундаментальні знання, а також навички логічного моделювання, абстрагування та оптимізації.

У сучасних умовах значної неоднорідності рівня підготовки студентів та динамічного розвитку ІТ-індустрії ефективним є впровадження гібридного підходу до викладання, що поєднує класичні методи з інноваційними освітніми практиками – візуалізацією, адаптивними технологіями, інтерактивними інструментами та проєктною діяльністю.

Такий підхід дозволяє не лише підвищити якість засвоєння матеріалу, а й формувати гнучкі, стійкі до змін професійні компетентності. Важливими чинниками успішного розвитку алгоритмічного мислення є поетапність, індивідуалізація навчання, мультимодальність подання матеріалу та стимулювання дослідницької активності студентів.

Отже, ефективне формування алгоритмічного мислення потребує цілісної освітньої стратегії, яка враховує когнітивні особливості здобувачів, технологічні можливості та сучасні вимоги до ІТ-спеціалістів.

Список використаних джерел

1. Adorni G., Dodero G., et al. Development of Algorithmic Thinking Skills in K-12 Education [Електронний ресурс] // Procedia Computer Science. – 2024. – Vol. 231. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S245195882400099X. – Дата звернення: 08.05.2025.

2. Brusilovsky P. Adaptive hypermedia // User Modeling and User-Adapted Interaction. – 2001. – Vol. 11, № 1–2. – С. 87–110.

3. Brusilovsky P., Millán E. User Models for Adaptive Hypermedia and Adaptive Educational Systems // The Adaptive Web. – Springer, 2007. – С. 3–53.

4. Caspersen M. E., Bennedsen J. Instructional design of a programming course: a learning theoretic approach // Proceedings of the Third International Workshop on Computing Education Research. – 2007. – С. 111–122.

5. CodeCombat – Навчання програмування через гру [Електронний ресурс]. – Режим доступу: https://codecombat.com. – Дата звернення: 08.05.2025.

6. Dale N., Lewis J. Computer Science Illuminated. – 4th ed. – Boston : Jones & Bartlett Learning, 2010. – 656 с.

7. EdApp. Microlearning платформа [Електронний ресурс]. – Режим доступу: https://www.edapp.com. – Дата звернення: 08.05.2025.

8. H5P: Інтерактивний навчальний контент [Електронний ресурс]. – Режим доступу: https://h5p.org. – Дата звернення: 08.05.2025.

9. IEEE Education Society Reports [Електронний ресурс]. – IEEE, 2021. – Режим доступу: https://ieee-edusociety.org. – Дата звернення: 08.05.2025.

10. Kay R. H., Leung S. Exploring factors that influence technology-based assessments in higher education // Journal of Educational Computing Research. – 2017. – Vol. 55(3). – P. 353–380.

11. Khan Academy. Комп’ютерні науки [Електронний ресурс]. – Режим доступу: https://www.khanacademy.org/computing/computer-science. – Дата звернення: 08.05.2025.

12. Kintsch W. Comprehension: A Paradigm for Cognition. – Cambridge : Cambridge University Press, 1998. – 278 с.

13. Moodle. Офіційний сайт [Електронний ресурс]. – Режим доступу: https://moodle.org. – Дата звернення: 08.05.2025.

14. Paivio A. Mental Representations: A Dual Coding Approach. – Oxford : Oxford University Press, 1986. – 322 с.

15. Python Tutor: Візуалізація кроків виконання програм [Електронний ресурс]. – Режим доступу: http://pythontutor.com. – Дата звернення: 08.05.2025.

16. Visualgo.net [Електронний ресурс]. – Режим доступу: https://visualgo.net/en. – Дата звернення: 08.05.2025.

17. Whalley J. L., Lister R., Thompson E., Clear T., Robbins P., Ajith Kumar P. T., Prasad C. An evaluation of a diagramming tool for teaching programming // Proceedings of the 8th Australasian Conference on Computing Education (ACE 2006). – 2006. – Vol. 52. – С. 155–163.

 ISSN 1560-8956