УДК 371.38:004.047 **DOI** https://doi.org/10.32782/3041-2021/2025-2-30

ПРО ДОЦІЛЬНІСТЬ ВИВЧЕННЯ ОСНОВ ДИНАМІЧНОГО ПРОГРАМУВАННЯ У СТАРШІЙ ШКОЛІ ТА ЗВО

Шпортько О. В.
кандидат технічних наук, доцент,
доцент кафедри інформаційних систем та обчислювальних методів
Приватного вищого навчального закладу «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука»
ORCID ID: 0000-0002-4013-3057

Янчук П. С.
кандидат фізико-математичних наук, доцент,
професор кафедри інформаційних систем та обчислювальних методів
Приватного вищого навчального закладу «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука»
ORCID ID: 0000-0002-1618-5228

Лотюк Ю. Г.
кандидат педагогічних наук, доцент,
завідувач кафедри інформаційних систем та обчислювальних методів
Приватного вищого навчального закладу «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука»
ORCID ID: 0000-0001-6696-5583

Соловей Л. Я.
старший викладач кафедри інформаційних систем та обчислювальних методів
Приватного вищого навчального закладу «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука»
ORCID ID: 0009-0001-2832-1741

У статті обґрунтована доцільність вивчення основ динамічного програмування на уроках інформатики у середній школі та на заняттях з програмування молодших курсів ЗВО в темах «Вкладені цикли», «Масиви» та «Рекурсії». Наведена модельна задача та описана послідовність її розв'язання для демонстрації основ методу динамічного програмування: виділення станів системи, де останній стан відповідає розв'язку всієї задачі; встановлення критерію оптимальності станів; послідовного визначення оптимального результату для кожного стану зі збереженням цього результату для подальшого розрахунку оптимальних результатів наступних станів; формування шляху досягнення оптимального результату для всієї задачі через зворотне визначення відповідних проміжних станів. Показано, що зв'язок між станами системи необов'язково має бути лінійним, головне, щоб оптимальний результат чергового стану визначався через збережені оптимальні результати попередніх станів. Визначені рекурентні формули для послідовного обчислення оптимальних результатів кожного стану поставленої задачі. Наведені фрагменти програми для реалізації розроблених алгоритмів основних етапів розв'язку поставленої задачі мовою програмування C#. Оцінку складності наведених алгоритмів для втілення окремих етапів розв'язку поставленої задачі проведено методами теорії алгоритмів. Встановлено, що асимптотична обчислювальна складність прямого та зворотного ходу методу динамічного програмування на основі рекурентних формул, як мінімум, на порядок нижча відповідних рекурсивних викликів підпрограм. Правильність теоретичних викладок та коректність наведених фрагментів програми підтверджена експериментально за допомогою віддаленого незалежного обчислювального середовища на сайті https://basecamp.eolymp.com. Наголошено, що вивчення основ динамічного програмування формуватиме у здобувачів освіти навички регресійного тестування та написання не лише правильних, а й ефективних програм.

Ключові слова: *доцільність вивчення динамічного програмування, методи оптимізації, методика навчання інформатики, алгоритмічне мислення, формування ІТ-компетентностей, асимптотична обчислювальна складність, оптимізація алгоритмів.*

Shportko A. V., Yanchuk P. S., Lotiuk Yu. G., Solovei L. Ya. On the Advantage of Studying the Basics of Dynamic Programming in High School and University

The article substantiates the feasibility of studying the basics of dynamic programming in computer science lessons in high school and in programming classes in junior years of universities in the topics "Nested Loops", "Arrays" and "Recursions". A model problem is presented and the sequence of its solution is described to demonstrate the basics of the dynamic programming method: allocation of system states, where the last state corresponds to the solution of the entire problem; establishment of a criterion for optimality of states; sequential determination of the optimal result for each state with saving this result for further calculation of optimal results of subsequent states; formation of a path to achieve an optimal result for the entire problem through the reverse determination of the corresponding intermediate states. It is shown that the relationship between the states of the system does not necessarily have to be linear, the main thing is that the optimal result of the next state is determined through the saved optimal results of the previous states. Recursive formulas are determined for sequential calculation of optimal results of each state of the problem. The program fragments for implementing the developed algorithms of the main stages of solving the problem in the C# programming language are presented. The complexity of the given algorithms for implementing individual stages of solving the problem was assessed using algorithm theory methods. It was established that the asymptotic computational complexity of the forward and backward steps of the dynamic programming method based on recurrent formulas is at least an order of magnitude lower than the corresponding recursive calls of subroutines. The correctness of the theoretical calculations and the correctness of the given program fragments were confirmed experimentally using a remote independent computing environment on the website https://basecamp.eolymp.com. It is emphasized that studying the basics of dynamic programming will form regression testing skills and writing not only correct but also effective programs in students.

Key words: *feasibility of studying dynamic programming, optimization methods, methods of teaching informatics, algorithmic thinking, IT competence development, efficient programming, asymptotic computational complexity, algorithm optimization.*

Вступ. На сьогодні основи динамічного програмування [10, розділ 15] вивчаються в курсі інформатики середньої школи [2; 3, с. 12; 6–7] лише в 11 класі на профільному рівні у темі «Алгоритми» або для підготовки до олімпіад з програмування [5]. У вищій школі цей метод, в основному, вивчається лише в дисципліні «Методи оптимізації та дослідження операцій». Хоча ідея динамічного програмування як підходу до розв'язання оптимізаційних задач, який полягає в розбитті задачі на підзадачі з послідовним розв'язанням цих підзадач і зберіганням їх розв'язків для вирішення наступних підзадач аж до розв'язування початкової задачі, є нескладною для розуміння і на цей час широко використовується в інформатиці, біології, економіці та інших галузях [4]. У вищій школі розбиття задач на типові підзадачі (кроки) і їх послідовне розв'язання – це вміння, потрібне для вивчення багатьох дисциплін.

Тому, на нашу думку, основи динамічного програмування на прикладах розв'язування найпростіших оптимізаційних задач доцільно додатково вводити під час вивчення:

– циклів у 7 класі загальноосвітніх шкіл [6, с. 34];

– вкладених циклів та рекурсій у 8 класі [7, с. 28–29];

– принципу рекурсії та побудови рекурсивної функції у темі «Функції» вибіркового модуля «Креативне програмування» в 10–11 класі для рівня стандарту [3, с. 60];

– масивів як структури даних, рекурсій для програмування обчислень за рекурентними формулами та обробки структур даних у 10 класі для профільного рівня [2, с. 6–7];

– вкладених циклів, масивів та рекурсій у дисциплінах, пов'язаних з вивченням мов програмування та в дискретній математиці у ЗВО.

Отже, **метою** цієї статті є пропонування методики вивчення основ динамічного програмування на уроках інформатики у середній школі та на заняттях з програмування молодших курсів ЗВО. Основні **завдання дослідження** випливають з поставленої мети:

1. Навести модельну задачу, придатну для розв'язання зазначеними здобувачами, та описати послідовність її розв'язання для демонстрації основ методу динамічного програмування.

2. Визначити рекурентні формули для послідовного обчислення оптимальних результатів кожного стану поставленої задачі.

3. Встановити асимптотичну обчислювальну складність прямого та зворотного ходу методу динамічного програмування на основі рекурентних формул для поставленої задачі.

4. Навести фрагменти програми для реалізації розроблених алгоритмів основних етапів розв'язку поставленої задачі мовою програмування C#.

Методи дослідження. В процесі опрацювання літературних джерел та проблематики статті використано такі загальні теоретичні педагогічні методи, як аналіз та синтез, узагальнення, порівняння, абстрагування, індукцію та дедукцію. Для дослідження сучасного стану вивчення основ динамічного програмування застосовувався емпіричний метод педагогічного спостереження. Ефективність наведеної методики запропоновано здійснити методом педагогічного експерименту.

Стосовно ж методу динамічного програмування, аспекти вивчення якого досліджуються у цій статті, то, як відомо, в задачах, які підпадають під дію цього методу, виділяють окремі кроки, після чого знаходять рішення для кожного кроку, щоб забезпечити оптимальний розв'язок для всього процесу загалом. Розв'язуються такі багатокрокові задачі на основі принципу оптимальності Беллмана [10], який ми процитуємо з [1]: «Який би не був стан системи перед черговим кроком, керування на цьому кроці необхідно обирати так, щоб оптимізувати результат на цьому кроці плюс результат на всіх наступних кроках». Тому задачі динамічного програмування найчастіше розв'язують «з кінця», визначаючи оптимальні результати від останнього кроку до першого для всіх допустимих станів.

Ми ж пропонуємо у старшій школі та на перших курсах ЗВО вивчати основи динамічного програмування, виділяючи лише стани системи та критерій оптимальності станів, після чого послідовно визначати оптимальні результати для кожного стану зі збереженням цих результатів для подальшого розрахунку оптимальних результатів наступних станів, і на завершення – формувати шлях досягнення оптимального результату для всієї задачі через зворотне визначення відповідних проміжних станів.

Результати дослідження. Покажемо, наприклад, як можна подавати основи динамічного програмування під час вивчення двовимірних масивів у 8 класі за допомогою однієї із задач на визначення траєкторії.

Задачу, яка розв'язується у цій статті, коротко сформулюємо так, як вона подана на сайті https://basecamp.eolymp.com під назвою «Мишка і зернинки» (№ 15): «В індійському храмі підлогу прямокутної форми вимощено однаковими квадратними плитками 1×1, на кожну з яких насипано від 0 до k (k≤30000) зернинок. Розміри підлоги m×n (1≤m, n≤100). Мишка вибігає з лівого нижнього кута підлоги храму і рухається до входу в іншу нірку, розміщену у протилежному кутку. Мишка може рухатись лише праворуч або вперед, збираючи всі зернинки з плитки, на якій вона знаходиться. Знайти маршрут, рухаючись по якому мишка збере найбільшу кількість зернин». На вхід програми після чисел m та n подається m рядків по n чисел. Програма має виводити маршрут руху мишки у форматі рядка без пробілів із символів R та F, де F позначає крок вперед, а R – крок праворуч. Ліміт часу для виконання програмою кожного тесту – 1 с, максимально можливий обсяг використання пам'яті – 128 Мб.

Зрозуміло, що для зберігання кількостей зернинок на кожній плитці і кількостей зернинок, зібраних мишкою від початку руху, потрібно використати двовимірні масиви. Нехай мишка починає рухатися з плитки з координатами (0, 0) до плитки з координатами (n-1, m-1). Тут і надалі будемо розглядати лише цілочисельні координати. Задання такої системи координат спонукатиме учнів замислитися над різницею між координатами точки на площині та індексом елемента в двовимірному масиві: на площині спочатку вказується координата x, а потім y, а в двовимірному масиві, як правило, спочатку задають індекс рядка, а потім – стовпця; на площині координати у збільшуються вгору, а в двовимірних масивах рядки нумеруються зверху вниз.

Опишемо процес формування розв'язку цієї задачі для тестового прикладу вхідних даних з web-сторінки https://basecamp.eolymp.com/uk/problems/15, наведених на рис. 1.

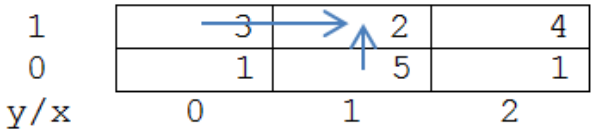

Рис. 1. Тестовий приклад кількостей зернин на підлозі розміром 3×2 плитки і напрямки переходу до плитки з координатами (1, 1) із сусідніх плиток

Навівши учням цей приклад кількостей зернин на плитках підлоги, варто у них поцікавитися, які шляхи переміщення мишки з плитки (0, 0) до плитки (2, 1) вони пропонують? Яку кількість зернин збере мишка, переміщуючись цими шляхами? Який шлях будемо вважати оптимальним? Чи завжди оптимальний шлях буде проходити через плитку з максимальною кількістю зернин? Від чого залежить максимальна сума зернин, зібраних від початку руху до кожної плитки?

Тільки після того, як учні зрозуміють, що максимальна сума зібраних зернин від плитки (0, 0) до кожної плитки залежить лише від таких самих сум для двох сусідніх плиток зліва та знизу і кількості зернинок на поточній плитці, а розв'язок всієї задачі – це ця сума для плитки (n-1, m-1), варто перейти до формування максимальних сум зібраних зернин від початку руху до кожної плитки.

Для якої плитки ми відразу можемо визначити максимальну суму зібраних зернин? Зрозуміло, що для стартової плитки з координатами (0, 0) така сума рівна кількості зернин на цій плитці, бо мишка з неї починає рух. На кожну плитку найнижчого рядка з координатами (j, 0), де j>0, мишка може потрапити лише з плитки (j-1, 0), збільшивши стосовно неї максимальну суму зібраних зернин на кількість зернин цієї плитки (j, 0). Аналогічно максимальна сума зібраних зернин для всіх плиток першого стовпця з координатами (0, i), де i>0, рівна сумі такого показника для сусідньої плитки знизу з координатами (0, i-1) та кількість зернин цієї плитки (0, i). Важливо, що максимальні суми для найнижчого рядка та першого стовпця ми можемо визначати відразу, рухаючись послідовно.

А ось у кожну плитку, розміщену у всіх рядках, крім останнього, і у всіх стовпцях, крім першого, мишка може потрапити або із сусідньої плитки зліва, або із сусідньої плитки знизу (див. рис. 1) і сума зібраних зернин тоді зросте на кількість зернин поточної плитки. Зрозуміло, що максимальна сума зібраних зернин для таких плиток – це максимум з цих двох варіантів. Виходячи з цих міркувань, запишемо послідовність рекурентних формул для визначення максимальної суми зібраних зернин від початку руху до кожної плитки (перший індекс тут вказує на координату по осі абсцис, а другий – по осі ординат, $corn_{i,j}$ – кількість зернинок на плитці з координатами (i, j)):

$$\begin{aligned} &suma_{0,0} = corn_{0,0};\\ &suma_{i,0} = suma_{i-1,0} + corn_{i,0}, \quad i = \overline{1, n-1};\\ &suma_{0,j} = suma_{0,j-1} + corn_{0,j}, \quad j = \overline{1, m-1};\\ &\left(suma_{i,j} = \max\left(suma_{i-1,j} + corn_{i,j}; suma_{i,j-1}\right.\right.\\ &\left.\left. + corn_{i,j}\right), i = \overline{1, n-1}\right), j = \overline{1, m-1}. \end{aligned} \quad (1)$$

Тут слід наголосити на важливості використання саме рекурентних формул (1), які дають змогу послідовно визначати $suma_{i,j}$ для всіх плиток аж до $suma_{n-1,\ m-1}$, замість аналогічної рекурсивної функції $suma(n–1,\ m–1)$, адже асимптотична обчислювальна складність рекурентних формул для всіх плиток становить $O(m{\times}n)$, а для рекурсивної функції – не менше $O(m{\times}n{\times}\max(m,\ n))$, враховуючи, що для всіх плиток, крім тих, що розміщені в останньому рядку і стовпці, ця рекурсивна функція викликається неодноразово. Саме рекурентні формули (1) відображають *прямий хід методу динамічного програмування* для цієї задачі. Для їх реалізації максимальні суми зернинок для попередніх плиток потрібно зберігати в масиві чи використовувати мемоізацію [8]. Максимальні суми зернинок, обчислені для тестового прикладу з рис. 1, наведені на рис. 2.

Наведемо фрагмент програми для реалізації прямого ходу методу динамічного програмування поставленої задачі мовою програмування C#, оскільки на сьогодні вона є однією з основних мов програмування прикладних додатків:

```
suma[0, 0] = corn[0, 0];
for (j = 1; j < n; j++)  suma[0, j] = suma[0, j – 1]
+ corn[0, j];
for (i = 1; i < m; i++)  suma[i, 0] = suma[i – 1, 0]
+ corn[i, 0];
```

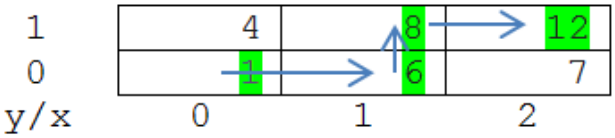

Рис. 2. Накопичені максимальні суми зернин від початку руху до кожної плитки та шлях накопичення такої суми для всієї підлоги

```
for (i = 1; i < m; i++)
for (j = 1; j < n; j++)
if (suma[i, j – 1] + corn[i, j] > suma[i – 1, j] + corn[i, j])
suma[i, j] = suma[i, j – 1] + corn[i, j];
else  suma[i, j] = suma[i – 1, j] + corn[i, j];
```

Але визначенням максимальної суми зібраних зернин від початкової до останньої плитки на шляху переміщення мишки розв'язання поставленої задачі не завершується, адже нам потрібно знайти не саму максимальну суму, а *визначити шлях переміщення*, який забезпечує цю максимальну суму. Формувати шлях, починаючи з початкової плитки недоцільно, оскільки однозначно визначити напрямок руху від неї неможливо, а рекурсивні виклики можуть у рази сповільнити виконання програми. Тому визначимо цей шлях від останньої плитки, задаючись щоразу питанням: «Звідки має переміститися мишка, щоб отримати максимальну суму зібраних зернин для поточної плитки?». Наприклад, для останньої плитки з координатами (2, 1) максимальну суму зібраних зернин 12 було отримано переміщенням з плитки (1, 1), для якої ця максимальна сума становить 8, шляхом додавання кількості зернин на останній плитці 4. Тому останнє переміщення мишки відбулося вправо (остання стрілка на рис. 2), отже, на початок рядка результату ми допишемо *R* і аналогічно продовжимо визначення маршруту з позиції (1, 1). Формування маршруту завершується після переходу до початкової позиції з координатами (0, 0). Це і є *зворотний хід методу динамічного програмування* для цієї задачі. Важливо пояснити учням, що, по-перше, за одну ітерацію зворотного ходу ми на один рядок чи стовпець наближаємося до початкової плитки, тому загальна кількість таких ітерацій становитиме лише *n+m-2* і, по-друге, реалізація зворотного ходу методу динамічного програмування можлива лише за умови зберігання максимальних сум зібраних зернин для кожної плитки. Шлях мишки, визначений за цим алгоритмом для тестового прикладу рис. 1, запишеться як *RFR* (див. стрілки на рис. 2). Фрагмент коду програми для реалізації зворотного ходу методу динамічного програмування для цієї задачі подамо так:

```
i = m - 1; j = n - 1; // починаємо визначення з останньої плитки
string res = "";
while (i>0 || j>0)
{if (i > 0 && suma[i, j] = suma[i-1, j] + corn[i, j]) // переміщення було знизу
{res= "F"+res; i--; }
else
{res= "R"+res; j--; }}
Console.WriteLine(res);
```

Подамо результати тестування програми з наведеними фрагментами коду з незалежного обчислювального середовища на сайті https://basecamp.eolymp.com (рис. 3).

На цей сервер до кожної задачі програміст може завантажити розв'язки у вигляді текстів різноманітних програм, написаних навіть на різних мовах програмування. Після цього сервер сайту компілює отримані програми і подає

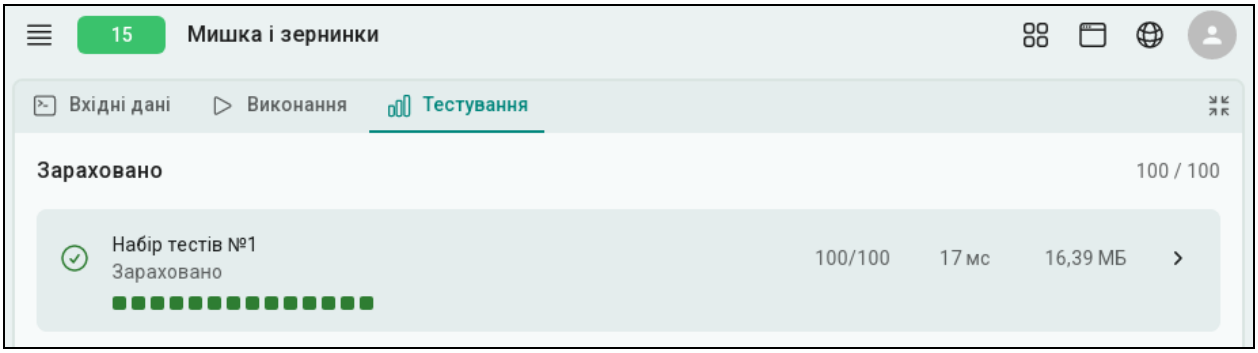

Рис. 3. Результати тестування програми для розв'язування поставленої задачі

їм на вхід виконання різні тести, невідомі програмістам та звіряє отримані результати на виході з очікуваними. Задача вважається розв'язаною лише тоді, коли відповідна програма пройшла кожен з тестів за відведений проміжок часу та не перевищила допустимий обсяг використання оперативної пам'яті (наприклад, описана тут програма витратила максимум 17 мс та 16.39 МБ оперативної пам'яті на кожен зі 100 пройдених тестів). Переваги від використання такого віддаленого незалежного обчислювального середовища очевидні, адже, використовуючи його, можна об'єктивно порівнювати різні розв'язки між собою [8]. Віддалене тестування розроблених програм по суті реалізує концепцію регресійного тестування, адже зменшення кількості пройдених тестів чи збільшення часу тестування наступним варіантом програми сигналізує про погіршення якості коду.

За результатами дослідження зробимо такі **висновки**:

1. Використання методу динамічного програмування кардинально зменшує обчислювальну складність алгоритму за рахунок використання напрямленого замість повного перебору варіантів, хоча й вимагає зберігання значень цільової функції для кожного з проміжних станів.

2. Задачі, що розв'язуються з використанням підходів динамічного програмування, варто вирішувати з учнями, починаючи з 7–8 класу під час вивчення вкладених циклів, а зі студентами – не тільки в дисциплінах, пов'язаних з програмуванням, а і з вивченням дискретної математики та тестування. Це формуватиме навички написання не лише правильних, а й ефективних (у контексті часу виконання та використання обчислювальних ресурсів) програм та дасть змогу здобувачам освіти поступово оволодівати цим потужним інструментом для розв'язування реальних задач з різних галузей економіки.

3. Застосування віддалених серверів для перевірки розроблених програм шляхом проходження наборів невідомих тестів дає змогу співставляти результати виконання кодів, написаних різними мовами програмування чи для різних методів, формує навички регресійного тестування та спонукає здобувачів освіти до усвідомлення значення ефективності програмного коду та критеріїв його оцінювання і тому рекомендується нами для практичного використання у середній і старшій школі та у ЗВО.

Список використаних джерел

1. Динамічне та нелінійне програмування. Методичні вказівки до проведення практичних та самостійних занять з курсу «Дослідження операцій» для студентів факультету кібернетики / упорядн. В. І. Тюптя, В. І. Шевченко, В. К. Стрюк. Київ : Електронна бібліотека факультету кібернетики Київського національного університету імені Тараса Шевченка, 2003, 30 с. URL: http://csc.univ.kiev.ua/uk/library/books/tiuptia-31.pdf (дата звернення: 27.05.2025).
2. Інформатика для 10–11 класів (профільне навчання). URL: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-10-11-klas/2018-2019/01/10-11-profilniy-riven.docx (дата звернення: 27.05.2025).
3. Інформатика. Навчальна програма вибірково-обов'язкового предмета для учнів 10–11 класів загальноосвітніх навчальних закладів (рівень стандарту). URL: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-10-11-klas/2018-2019/informatika-standart-10-11.docx (дата звернення: 27.05.2025).
4. Метод динамічного програмування та його особливості. URL: https://foxminded.ua/metod-dynamichnoho-prohramuvannia/ (дата звернення: 27.05.2025).
5. Олефіренко Н. В., Курганський А. Р. Розробка електронного посібника для навчання школярів основ динамічного програмування. *Наумовські читання* : збірник тез доповідей XIX науково-методичної конференції здобувачів вищої освіти та молодих учених (м. Харків, 2–24 листопада 2021 року). Харків, 2022, с. 158–160.
6. Програма курсу «Інформатика». 5–9 класи загальноосвітніх навчальних закладів. URL: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/onovlennya-12-2017/programa-informatika-5-9-traven-2015.pdf (дата звернення: 27.05.2025).
7. Програма курсу «Інформатика». 8–9 класи загальноосвітніх навчальних закладів з поглибленим вивченням інформатики. URL: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/informatika.pdf (дата звернення: 27.05.2025).

8. Шпортько О. В., Малаш К. М. Застосування мемоізації в задачах мінімізації кількості знаків арифметичних операцій. *Вісник Національного університету водного господарства та природокористування. Серія : Технічні науки*. Рівне : НУВГП, 2020. № 2 (90). С. 127–143.
9. Bellman R. On the Theory of Dynamic Programming. *Proc N. A S. USA*. Vol. 38 (8). 1952. Pp. 716–719. URL: https://www.pnas.org/doi/abs/10.1073/pnas.38.8.716.
10. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. Third Edition. Cambridge : MIT Press, 2009. 1320 p.

References

1. Dynamichne ta neliniine prohramuvannia. Metodychni vkazivky do provedennia praktychnykh ta samostiinykh zaniat z kursu "Doslidzhennia operatsii" dlia studentiv fakultetu kibernetyky [Dynamic and Nonlinear Programming. Methodological Guidelines for Conducting Practical and Independent Classes in the Course "Operations Research" for Students of the Faculty of Cybernetics] (2003) / uporiadn. V. I. Tiuptia, V. I. Shevchenko, V. K. Striuk. Kyiv: Elektronna biblioteka fakultetu kibernetyky Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka (30 p.). Retrieved from: http://csc.univ.kiev.ua/uk/library/books/tiuptia-31.pdf (Last accessed: 27.05.2025) [in Ukrainian].
2. Informatyka dlia 10–11 klasiv (profilne navchannia) [Informatics for grades 10–11 (profile lessons)]. Retrieved from: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-10-11-klas/2018-2019/01/10-11-profilniy-riven.docx (Last accessed: 27.05.2025) [in Ukrainian].
3. Informatyka. Navchalna prohrama vybirkovo-oboviazkovoho predmeta dlia uchniv 10–11 klasiv zahalnoosvitnikh navchalnykh zakladiv (riven standartu) [Informatics. Initial program of an elective-compulsory subject for students of grades 10–11 of secondary educational institutions (standard level)]. Retrieved from: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-10-11-klas/2018-2019/informatika-standart-10-11.docx (Last accessed: 27.05.2025) [in Ukrainian].
4. Metod dynamichnoho prohramuvannia ta yoho osoblyvosti [Dynamic programming method and its features]. Retrieved from: https://foxminded.ua/metod-dynamichnoho-prohramuvannia/ (Last accessed: 27.05.2025) [in Ukrainian].
5. Olefirenko N. V. and Kurhanskyi A. R. (2022). Rozrobka elektronnoho posibnyka dlia navchannia shkoliariv osnov dynamichnoho prohramuvannia [Development of an electronic manual for teaching students the basics of dynamic programming]. *Naumovski chytannia:* zbirnyk tez dopovidei XIX naukovo-metodychnoi konferentsii zdobuvachiv vyshchoi osvity ta molodykh uchenykh (m. Kharkiv, 23–24 lystopada 2021 roku). Kharkiv, 158–160 [in Ukrainian].
6. Prohrama kursu «Informatyka». 5–9 klasy zahalnoosvitnikh navchalnykh zakladiv. Retrieved from: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/onovlennya-12-2017/programa-informatika-5-9-traven-2015.pdf (Last accessed: 27.05.2025) [in Ukrainian].
7. Prohrama kursu «Informatyka». 8–9 klasy zahalnoosvitnikh navchalnykh zakladiv z pohlyblenym vyvchenniam informatyky. Retrieved from: https://mon.gov.ua/storage/app/media/zagalna%20serednya/programy-5-9-klas/informatika.pdf (Last accessed: 27.05.2025) [in Ukrainian].
8. Shportko A. V. and Malash K. M. (2020). Zastosuvannia memoizatsii v zadachakh minimizatsii kilkosti znakiv aryfmetychnykh operatsii [Use of memoization in problems of minimizing the number of signs of arithmetic operations]. *Visnyk Natsionalnoho universytetu vodnoho hospodarstva ta pryrodokorystuvannia (Seriia: Tekhnichni nauky) – Bulletin of the National University of Water Management and Environmental Management (Series: Technical Sciences)*. Rivne: NUWM, 2 (90), 127–143 [in Ukrainian].
9. Bellman R. (1952). On the Theory of Dynamic Programming. *Proc N. A S. USA*. 38 (8), pp. 716–719. Retrieved from: https://www.pnas.org/doi/abs/10.1073/pnas.38.8.716.
10. Cormen T. H., Leiserson C. E., Rivest R. L. and Stein C. (2009). Introduction to Algorithms. Third Edition. Cambridge: MIT Press. 1320 p.