Аналіз основних цілей SMM демонструє, що чітке розуміння цих аспектів є необхідною умовою для досягнення успіху в цифровому маркетингу. Кожна з цих цілей відіграє свою унікальну роль у розвитку бізнесу, тому важливо забезпечити їх збалансоване поєднання у стратегії. Систематичний підхід до аналізу результатів та адаптації стратегії у відповідь на зміни в конкурентному ринку забезпечить компаніям стійке зростання та успішну взаємодію з цільовою аудиторією.

Abstract. The article is devoted to an overview of the role of SMM (Social Media Marketing) in the formation of digital marketing strategies. It describes the features of key social platforms – Facebook, Instagram, Twitter, TikTok, and YouTube – and their impact on user interaction. The authors emphasize the importance of adapting content to the specifics of each network for maximum audience engagement and building long-term loyalty. Six key SMM goals are highlighted, serving as the foundation for developing effective strategies. The article also presents the «POST» model for managing interactions and the SMART approach for effective goal setting, which allows for increased competitiveness and achievement of strategic business objectives.

Keywords: SMM, social media, POST, SMART goals, digital marketing.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Яцківська А. Соціальні мережі як ефективний засіб просування бізнесу у світі. *Society and Security.* 2024. № 1(2). P. 34–39. URL: https://sas.ztu.edu.ua/article/view/301984

2. Люльчук А. В., Кияниця Є. О. Ефективні реляції використання Social Media Marketing. *Журналістика та реклама: вектори взаємодії:* тези доповідей ІІІ міжнародної науково-практичної конференції. Київ. 2021. С. 187–191. URL: https://knute.edu.ua/file/MjIxNw==/d5a4cef59a0a732694d08dd5d6454628.pdf#page=188

3. Карпенко Ю. М. Цілі SMM: особливості визначення та узгодження. *Актуальні питання теорії та практики в галузі права, освіти, соціально-гуманітарних та поведінкових наук в умовах воєнного стану:* матеріали міжнар. наук.-практ. конф. (м. Чернігів, 25–26 квітня 2023 р.): у двох томах. Т. 2 / гол. ред. В. Ф. Пузирний; Академія Державної пенітенціарної служби. Чернігів: Академія ДПтС, 2023. С. 253–256. URL: https://academysps. edu.ua/wp-content/uploads/2023/06/Konferenciya-Tom-2-_25-26-kvitnya_2023.pdf

4. Чуніхіна Т. С. Теоретичні засади формування SMM-стратегії підприємства. *International scientific journal «Grail of Science».* 2022. № 23. URL: https://archive.journal-grail.science/index.php/2710-3056/article/view/749

УДК 004:005:51

АЛГОРИТМИ І ПРОГРАМУВАННЯ: ЗАСТОСУВАННЯ ТА ВАЖЛИВІСТЬ

В. В. Гураш, Н. А. Потапова

Анотація. У статті розглядаються основи алгоритмів та їх важливість і застосування у сфері програмування. Описуються ключові властивості алгоритмів, які роблять програмний код надійним і ефективним. Також досліджено різні методи опису алгоритмів та їх широке застосування в сучасних технологіях. Наведені приклади реалізації алгоритмів на мові програмування C#, які демонструють практичні аспекти ефективності різних алгоритмічних підходів шляхом емпіричного тестування. Стаття підкреслює важливість правильного вибору алгоритмічних структур для оптимізації продуктивності та надійності програмного забезпечення.

Ключові слова: алгоритми, програмування, структури даних, задачі, ресурси.

Вступ. У сучасному програмуванні структуроване пояснення програмного коду відіграє важливу роль у забезпеченні оптимізації та ефективності використання структур даних. Розуміння логіки алгоритму дає змогу не лише зменшити час виконання операцій, але й зробити код більш зрозумілим та доступним для інших розробників. Впровадження чітких і продуманих алгоритмів для вирішення задач дає можливість ефективніше використовувати ресурси користувача, зменшувати ризик помилок та спрощувати подальшу підтримку й оновлення програмного забезпечення [1].

Метою цієї статті є обґрунтування визнання причинно-наслідкового зв’язку між категоріями алгоритмізації та програмування, з її подальшим використанням у процесі оцінки ефективності програмних ресурсів.

Основна частина. Люди щоденно користуються різноманітними правилами, інструкціями, рецептами тощо, що складаються з певної послідовності команд. Деякі з них настільки увійшли до нашого життя, що ми виконуємо їх, майже не замислюючись. Ці послідовності можна на-

звати алгоритмом – це скінченний набір команд, який потрібно виконати, щоб досягти певної мети, або отримати певний результат.

Алгоритми – це основа будь-якого програмного забезпечення. У світі програмування, розуміння та ефективне використання алгоритмів відіграє важливу роль у розробці програмного забезпечення. Кожна команда алгоритму вказує на дію, яку має виконати його виконавець. Виконавцем може бути як жива, так і нежива істота, наприклад: людина, тварина, електронна обчислювальна машина тощо [2].

Алгоритми мають свої властивості, дотримання яких гарантує успішність виконання завдання. Такими властивостями є:

– елементарність – кожна команда містить вказівку виконати деяку елементарну дію, яка є однозначною для сприйняття і точною для виконання;

– визначеність – на кожному кроці не тільки точно виконується команда, але й визначається наступна;

– масовість – алгоритми мають бути призначені для розв'язання набору однотипних задач, а не для однієї унікальної;

– результативність – виконання будь-якого алгоритму повинно бути закінчене через скінченну кількість кроків з деяким результатом;

– скінченність – алгоритм завжди завершує роботу;

– дискретність – алгоритм будується з окремих блоків команд, які потрібні для виконання завдання;

– правильність – алгоритм працює правильно, тобто повертає правильну відповідь.

Дотримання цих властивостей забезпечує надійність, передбачуваність і продуктивність програмного коду. Це також полегшує тестування та підтримку програми в майбутньому, оскільки код, побудований на правильних алгоритмах, легше аналізувати та оптимізувати. Алгоритм можна описати одним із таких способів: мовний, графічний, табличний, програмний [3].

Застосування алгоритмів дуже широке, їх існування та використання є важливою частиною багатьох прикладних сфер і окремих їх частин та спеціалізацій. Як приклад можна навести застосування алгоритмів малоресурсної криптографії, структурної оптимізації нейронної мережі, нечіткої логіки в системах розумного будинку тощо.

Основні критерії оцінки алгоритмічних структур базуються на продуктивності, ефективності та здатності вирішувати поставлені задачі. Отже, щоб логічна реалізація алгоритму була правильна, потрібно просто дотримуватись базових властивостей алгоритму. Реалізація алгоритму в програмному коді вимагає оцінки таких характеристик: швидкість виконання, важкість написання, важкість читання та розуміння програмного коду. Це означає, що один алгоритм може мати декілька реалізацій, для яких будуть використовуватись різні команди і способи написання програми [5].

Для оцінки ефективності реалізації алгоритму на мові програмування потрібно: провести практичне емпіричне тестування, вимірявши реальний час виконання; провести тестування алгоритму на різних вибірках даних; порівняти ефективність з іншими реалізаціями. Оцінка ефективності на реальних та граничних наборах даних дає змогу визначити поведінку алгоритму в стресових умовах і дає змогу уникнути помилок під час виконання коду. Необхідно оцінювати, як алгоритм поводиться у випадках помилок або непередбачуваних ситуацій [6].

Покажемо, багатоваріантність алгоритму на умовах практичного прикладу використання двох рішень для алгоритмічних рішень задач: з розгалуженням (рис. 1) і з повтореннями:

$$y = \begin{cases} 2x(4-x)(3+x)\ x > 3{,}5; \\ 3x\cos(4{,}5-x)\ 0 \le x \le 3{,}5; \\ x^4\ x < 0. \end{cases}$$

Рис. 1. Умова алгоритмічної задачі розгалуження

Реалізуємо алгоритмічну задачу розгалуження за допомогою мовних конструкцій C#: switch та if. Проведемо порівняння цих реалізацій емпіричним методом тестування. Реалізація за допомогою функції Switch:

```
public static double Switch(double x)
{
switch (x > 3.5)
{
case true:
return 2*x*(4-x)*(3+x);
case false:
switch (x >= 0 && x <= 3.5)
{
case true:
return 3*x*Math.Cos(4.5 – x);
case false:
return Math.Pow(x, 4);
}
}
}
```

Два switch у цьому випадку обирають значення true та false, які залежать від введеного користувачем значення змінної x, як і наші випадки case. Розглянемо тепер реалізацію через функцію Condition(If):

```
public static double Condition(double x)
{
if (x > 3.5)
return 2*x*(4-x)*(3+x);
else if (x >= 0 && x <= 3.5)
return 3*x*Math.Cos(4.5 – x);
else
return Math.Pow(x, 4);
}
```

У цьому випадку ми просто перевіряємо умову на правильність. Залежно від введеного значення змінної x, ми повертаємо відповідний результат виразу, як і у випадку зі switch, але можна побачити, що структуру if у цій ситуації легше та швидше відтворити в програмному коді, ніж switch. Ці два програмні коди реалізують один алгоритм під час використання різних команд. Проведемо емпіричне тестування за допомогою бібліотеки System.Diagnostics і перевіримо, яка з двох функцій працює швидше (рис. 2).

```
            > dotnet run
Введіть x: -2
Y(If): 16
00:00:00.0053203
Y(Switch): 16
00:00:00.0004487
PS C:\Users\Володимир Гураш\OneDrive
Введіть x: -2
Y(If): 16
00:00:00.0024598
Y(Switch): 16
00:00:00.0006124
PS C:\Users\Володимир Гураш\OneDrive
Введіть x: -2
Y(If): 16
00:00:00.0030043
Y(Switch): 16
00:00:00.0007981
```

Рис. 2. Результати емпіричного тестування реалізацій розгалуження

Незважаючи на те, що структуру if легше і швидше записувати, switch виявилася швидшою в середньому в 3 рази.

Проаналізуємо реалізацію алгоритму повторень (цикл), постановка якої така: обчислити добуток розрахованих значень функції $f(x)$, дробова частина яких < 0,5. Функція $f(x)$задана виразом:

$$f(x) = \frac{\cos(4\pi x)}{(1+x)}. \quad (1)$$

Програмний код реалізації має вигляд:

```
public static double For(double a, double b, double n, double h)
{
double d = 1;
double function;
for (double i = a; i <= b; i += h)
{
function = Math.Cos(4*Math.PI*i)/(1+i);
if (Math.Round(i,1) == -1)
{
Console.WriteLine("Dividing by zero error.");
return 0;
}
if (Math.Abs((function – Math.Truncate(function))) < 0.5)
{
d *= function;
}
}
return Math.Round(d,6);
}
public static double While(double a, double b, double n, double h)
{
double d = 1;
double function;
while (a <= b)
{
function = Math.Cos(4*Math.PI*a)/(1+a);
if (Math.Round(a,1) == -1)
{
Console.WriteLine("Dividing by zero error.");
return 0;
}
if (Math.Abs((function – Math.Truncate(function))) < 0.5)
{
d *= function;
}
a += h;
}
return Math.Round(d,6);
}
```

Проведемо емпіричне тестування нашого програмного коду (рис. 3).

```
Введіть a: -2
Введіть b: 2
Введіть n: 10
For: 0,004108
00:00:00.0030355
While: 0,004108
00:00:00.0007392
PS C:\Users\Володимир Гураш\OneDrive\
Введіть a: -3
Введіть b: 2
Введіть n: 10
For: 0,005391
00:00:00.0025923
While: 0,005391
00:00:00.0004490
PS C:\Users\Володимир Гураш\OneDrive\
```

Рис. 3. Результати емпіричного тестування реалізацій циклічного розгалуження

Оцінка реалізації показала, що структура for у цьому випадку працює повільніше за while, хоча вони реалізують один і той самий алгоритм, просто використовують різні команди.

Висновки. Алгоритми є невід'ємною частиною програмування і багатьох прикладних сфер діяльності, їх ефективна реалізація відіграє ключову роль у створенні надійного продукту. Практичні приклади та результати тестування показують, що один і той самий алгоритм може бути реалізований різними способами з різною ефективністю, що обумовлює необхідність вибору оптимального рішення для конкретних умов. Правильне використання алгоритмізації є основним шляхом підвищення ефективності програмного коду та його оптимізації. Часову ефективність реалізації алгоритму на програмному коді можна визначити за допомогою емпіричного тестування та вбудованих бібліотек.

Abstract. The article discusses the basics of algorithms and their importance and application in the field of programming. The article describes the key properties of algorithms that make a program code reliable and efficient. Different methods of describing algorithms and their widespread use in modern technologies are also investigated. Examples of algorithm implementation in the C# programming language are provided, which demonstrate the practical aspects of the effectiveness of various algorithmic approaches through empirical testing. The article emphasises the importance of choosing the right algorithmic structures to optimise software performance and reliability.

Keywords: algorithms, programming, data structures, tasks, resources.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Алгоритми та структури даних – від «десь чув» до «ефективно застосовую». *DOU.* 02.11.2022. URL: https://dou.ua/forums/topic/40645/

2. Що таке алгоритми: кроки, приклади, конструкції, мислення. *Dan-it education.* 21.10.2024. URL: https://dan-it.com.ua/uk/blog/shho-take-algorytmy-kroky-pryklady-konstrukcziyi-myslennya/

3. Романов В. Поняття алгоритму. Властивості алгоритмів. Форми подання алгоритму. Виконавець алгоритму. *Інформатика – шкільний курс.* URL: https://romanov.in.ua/11-3/

4. Алгоритм та його властивості. *КЗ «Піщанський ліцей № 1». Step by step.* 2021–2022. URL: https://step.org.ua/konspekt/algoritm/tema1

5. Чому важливо оцінювати складність алгоритму. *foxminded.ua. Курси програмування.* 23.02.2024. URL: https://foxminded.ua/skladnist-alhorytmu/

6. Емпіричні методи програмної інженерії: електронний конспект. Київ: НТУ «Київський Політехнічний Інститут», 2015. 119 с. URL: http://tc.kpi.ua/content/kurs/EMPI/EMPI_konspect.pdf

7. Sedgewick R., Wayne K. Algorithms. Addison-Wesley Professional, 2020. 956 p.

8. Korman T. H. Algorithms. Unlocked. The MIT Press, Massachusetts Institute of Technology, 2013. 207 p.

9. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ «Київський Університет», 2021. 200 с.

УДК 004.42: 006.73: 793.4

СТВОРЕННЯ 2D-ГРИ У СТИЛІ ПІКСЕЛЬ-АРТ

І. С. Діброва, О. В. Зелінська

Анотація. Дослідження фокусується на процесі створення піксельної арт-гри з використанням ігрового рушія Unity. У статті проаналізовано основні етапи розробки, включго з підготовкою графічних ресурсів, інтеграцією анімації, налаштуванням ігрової механіки та створенням користувацького інтерфейсу. Підкреслюється важливість детального вивчення кожного з цих етапів через їх вплив на загальний ігровий досвід. Також приділяється увага різним інструментам, доступним в Unity, та їх ролі у спрощенні процесу розробки, що дає змогу зосередитися на творчих аспектах.

Ключові слова: Aseprite, Unity, анімація, інтеграція, піксель-арт, скрипт, розробка ігор.

Комп'ютерна гра – це програмне забезпечення, призначене для забезпечення ігрового процесу. Ігри можуть спілкуватися з гравцями та співпрацювати з ними. В ігровому процесі використовуються різноманітні пристрої введення, зокрема комп'ютерні миші, клавіатури, камери та джойстики. Створення відеогри нічим не відрізняється від створення будь-якого іншого програмного продукту. До команди входять програмісти, менеджери, дизайнери, звукорежисери, дизайнери інтерфейсів.

Із розвитком ігрової індустрії з'являлись і нові жанри: шутери, пригоди, головоломки, навчальні, симулятори, платформери, RPG, стелс та багато інших.

Піксельне мистецтво – це форма цифрового мистецтва, створена на рівні пікселів. Здебільшого асоціюється з графікою відеоігор 1980-х та 1990-х років. У той час художникам доводи-