FRI-1.414-MIP-06

RECURSION AND ITERATION: BRIDGING MATHEMATICAL PUZZLES WITH COMPUTATIONAL SOLUTIONS [6]

Stanaila Neykova-Karagaeva, PhD Student
Department of Informatics and Information Technology,
University of Ruse "Angel Kanchev"
Tel.: +359 877544182
E-mail: stanaila.neykova@gmail.com

Assoc. Prof. Svetlozar Tsankov, PhD - Supervisor
Department of Informatics and Information Technology,
University of Ruse "Angel Kanchev"
E-mail: stsankov@uni-ruse.bg

Abstract: *Recursion and iteration are not merely programming constructs; they serve as pivotal tools linking computational practices to mathematical reasoning. This article delves into mathematical puzzles and challenges that can be addressed or illustrated through recursive or iterative approaches. It elucidates how computer science education can enrich and amplify critical thinking by employing mathematics as both a foundation and inspiration.*

Keywords: *Recursion, Iteration, Mathematical puzzles, Education, Critical thinking.*

ВЪВЕДЕНИЕ

В необятния свят на информатиката, рекурсията и итерацията се явяват като ключови концепции, които съчетават елегантност и сложност. Рекурсията и итерацията, въпреки различния си характер, се допълват една друга в контекста на алгоритмичното програмиране. Докато рекурсията предлага елегантен начин за разбиване на проблемите на по-малки части, итерацията предоставя пряк и интуитивен подход за тяхното последователно решаване. Тези методи не само улесняват разработчиците на програмни решения, но също така служат като важни инструменти за развитие и усъвършенстване на абстрактно мислене и алгоритмично разбиране. Изучаването и усвояването на рекурсията и итерацията е ключово за всеки, стремящ се да постигне твърда основа в дисциплините на компютърните науки и програмирането.

Интегрирането на математически пъзели в учебния процес по информатика обогатява обучението, като го прави по-динамично и взаимодействащо. Учениците не само научават как да решават задачи, но и разбират как компютърните програми могат да бъдат използвани за моделиране и решаване на специфични математически проблеми. Този подход спомага за развитието на ценни умения в областта на компютърните науки у учениците, които са приложими в разнообразни учебни дисциплини и професионални пътища.

ИЗЛОЖЕНИЕ

Рекурсия и итерация

Рекурсията е фундаменталена концепция в компютърните науки, играеща централна роля в методологията на програмирането. Определя се като техника, при която функция (или метод) се самоизвиква в рамките на своето изпълнение. Този процес се характеризира с два основни компонента: базов случай и рекурсивен стъпков механизъм. Базовият случай, известен още като 'дъно на рекурсията', е критичен за прекратяването на рекурсивния процес, докато стъпковият механизъм управлява логиката на последователните самоизвиквания на функцията.

[6] Докладът е представен на конференция на Русенския университет на 27 октомври 2023 г. в секция „Математика, информатика и физика“ с оригинално заглавие на български език: РЕКУРСИЯ И ИТЕРАЦИЯ: ВРЪЗКА НА МАТЕМАТИЧЕСКИ ПЪЗЕЛИ С ИЗЧИСЛИТЕЛНИ РЕШЕНИЯ

Рекурсията като техника е особено ефективна при разработването на алгоритми за решаването на сложни проблеми, като често води до по-елегантни и лесно четими решения. Въпреки тези предимства, рекурсията се свързва с някои ограничения, като увеличено използване на системни ресурси, включително памет, и потенциално по-бавно изпълнение при определени условия [6]. Специфично, дълбоките рекурсивни функции могат да доведат до препълване на стека, което от своя страна може да причини логически грешки и забавяне на програмата.

Итерацията от друга страна е основен програмен процес, който включва повтаряне на набор от инструкции определен брой пъти или до настъпването на специфично условие [3]. Тази техника се реализира чрез използването на оператори, известни като цикли или повторения, които представляват основни инструменти за управление на повторяемите команди и тяхното интегриране в алгоритмите в областта на програмирането.

При достигане до програмно решение, което е по-просто, по-кратко и по-лесно за разбиране чрез използването на рекурсия, без това да води до неефективност и странични ефекти, предпочитанието към рекурсивния подход е оправдано. От друга страна, ако рекурсията се оказва неподходяща поради потенциални недостатъци, препоръчително е разглеждането на итеративни алтернативи за достигане на желаното решение [2].

Математически пъзели и програмни решения

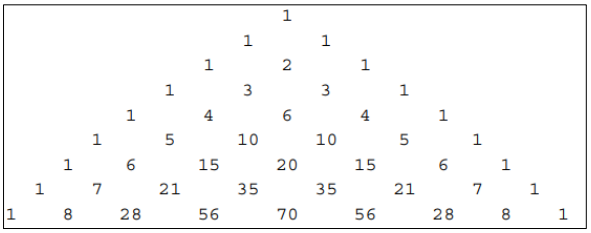
Математически пъзел е вид интелектуална игра или проблем, който изисква използването на математически познания и логическо мислене при решаването му [1]. Такива пъзели могат да включват различни математически концепции и методи – от основни аритметични операции до по-сложни области като алгебра, геометрия, теория на числата, комбинаторика и други.

Математическите пъзели предлагат уникална възможност за развиване на алгоритмичното и логическото мислене на учениците. Те насърчават разбирането на сложни математически структури и предизвикват използването или придобиването на умения за логическо анализиране и проблемно решаване. Програмните решения на такива пъзели изискват приложение на рекурсивни и итеративни подходи, като всяка техника предлага различни стратегии за достигане до решение. При рекурсивния подход, учениците учат как един проблем може да бъде разгледан като поредица от по-малки, подобни проблеми, докато при итеративния подход учащите разбират значението на последователните повторения и прогресивното приближаване към решение. Тези методи не само подобряват разбирането на учениците за математически концепции, но също така им помагат да развият умения за структуриране и кодиране на алгоритми.

Програмни решения на математически пъзели в C#

Пъзел 1: Триъгълник на Паскал

Триъгълникът на Паскал представлява интересен математически пъзел, във вид на триъгълна аранжировка от числа. Тази структура започва с единица на върха и продължава с редове, в които всяко число е равно на сумата на двете числа непосредствено над него в предходния ред [5]. Например, първият ред съдържа една единица, вторият – две единици, а третият ред е формиран от числата 1, 2, 1, като числото 2 произлиза от сумата на двете единици от втория ред. Следващата фигура (фиг. 1), представя първите девет реда от триъгълника на Паскал.



Фиг. 1. Триъгълник на Паскал

Задача 1: Съставяне на процедура, която при подаден единичен входен аргумент – цяло число N, генерира визуализация на Триъгълника на Паскал до неговия N-ти ред. Изходът е серия от редове, изобразяваща структурата на триъгълника, като се започва от върха и се продължава до зададения ред.

```
static void PrintTriangle(int numRows)
{
    for (int row = 0; row < numRows; row++)
    {
        int value = 1;
        for (int col = 0; col <= row; col++)
        {
            Console.Write(value + " ");
            value = value * (row - col) / (col + 1);
        }
        Console.WriteLine();
    }
}
```

Фиг. 2

```
static void PrintRow(int row, int col = 0, int value = 1)
{
    if (col > row)
        return;
    if (col == 0 || row == 0)
        value = 1;
    else
        value = value * (row - col + 1) / col;
    Console.Write(value + " ");
    PrintRow(row, col + 1, value);
}

static void PrintTriangle(int numRows)
{
    for (int row = 0; row < numRows; row++)
    {
        PrintRow(row);
        Console.WriteLine();
    }
}
```

Фиг. 3

Итеративният метод (фиг. 2) се опира на вложени цикли, за да изчисли стойностите на всяко число в реда, използвайки пряк подход. Този подход е интуитивен и следва линейна логика, улеснява разбирането на алгоритмични структури и е подходящ за начинаещи в програмирането, които развиват умения за управление на цикли и последователно изчисление. Рекурсивният метод (фиг. 3), от друга страна, използва самоизвикване на функция, като всеки елемент се изчислява въз основа на стойностите от предходните редове, вече определени от предходни извиквания на функцията. Този подход е по-компактен и изисква дълбоко разбиране на рекурсивните процеси, като по този начин развива абстрактно мислене и стимулира разбирането на сложните алгоритмични взаимозависимости.

Тези два подхода илюстрират различни стратегии на алгоритмично мислене и програмно решаване на първа задача. Итеративният метод предоставя основа за разработване на алгоритми с ясно дефинирани стъпки, докато рекурсията разкрива мощта на функционалното самоизвикване за разбиване на проблеми на по-малки подзадачи. Важно е да се отбележи, че въпреки различните си подходи, и двата метода достигат до едно и също решение, демонстрирайки уникалността и взаимозаменяемостта на алгоритмични подходи в контекста на компютърните науки.

Задача 2: Разработване на алгоритъм, който по даден ред и колона изчислява стойността на елемента на съответните позиции в Триъгълника на Паскал. Методът трябва да приема две цели числа като аргументи и да върне числото, разположено на тяхната пресечна точка в триъгълника.

```
static int GetPascalValue(int row, int col)
{
    if (col < 0 || col > row)
        return 0;
    int value = 1;
    for (int i = 1; i <= col; i++)
    {
        value = value * (row - i + 1) / i;
    }
    return value;
}
```

Фиг. 4

```
static int GetPascalValue(int row, int col)
{
    if (col < 0 || col > row)
        return 0;
    if (col == 0 || col == row)
        return 1;
    int valueAbove = GetPascalValue(row - 1, col - 1);
    int valueLeft = GetPascalValue(row - 1, col);
    return valueAbove + valueLeft;
}
```

Фиг. 5

Итеративният метод (фиг. 4) се основава на използването на биномните коефициенти, като последователно изчислява стойността на всеки елемент в Триъгълника на Паскал. Този процес включва последователно умножение и деление, което позволява пряко определяне на крайната стойност без изчисляването на целия предишен ред. Така итеративният подход е ефективен за големи стойности на входните данни и избягва натоварването, което рекурсията ще предизвика при такива мащаби. Рекурсивният метод (фиг. 5) се основава на характерната

рекурсивна природа на Триъгълника на Паскал, където всеки елемент е резултат от сумата на двете числа разположени непосредствено над него. Този подход визуализира структурата на триъгълника по интуитивен начин и може да бъде по-лесен за разбиране от учащите, но е по-малко ефективен в контекста на производителността при големи изчислителни задачи, поради повтарящи се извиквания на функции и висок риск от препълване на стека.

Обобщавайки, итеративният подход е предпочитан в ситуации, които изискват оптимална производителност и ефективност, особено когато се работи с голяма стойност на входните данни. Въпреки това, за образователни цели, рекурсивният подход може да бъде по-подходящ, тъй като улеснява разбирането на основните принципи на рекурсията и развива алгоритмичното мислене.

Решаването на задачите за Триъгълника на Паскал с итеративни и рекурсивни подходи допринася за обогатяването на образователния процес. Итеративният метод развива у учениците умения за последователно решаване на проблеми, ефективност в кодирането и практическо прилагане на математически понятия като биномните коефициенти. От друга страна, рекурсивният метод насърчава дълбокото разбиране на абстрактни концепции и усъвършенстване на логическо мислене чрез разбиване на задачите на по-малки подзадачи. Тези различни подходи не само стимулират алгоритмичното мислене, но също така предоставят на учениците възможност да опитат различни стратегии за решаване на проблеми, подобрявайки тяхната способност да избират подходящи методи в зависимост от ситуацията. Това от своя страна допринася за развитие на адаптивно мислене и гъвкавост в подхода към решаваните проблеми, които са съществени в съвременната динамична образователна и професионална среда.

Пъзел 2: Фрактал „триъгълник на Серпински“

Фракталът 'триъгълник на Серпински' е класически пример за самоподобни структури, които са основен компонент в изучаването на фракталната геометрия. Той се състои от един равностранен триъгълник, който е разделен на по-малки триъгълници, като се премахва централният инвертиран триъгълник от всяко повторение [4]. Този процес на рекурсивно намаляване и премахване се повтаря неограничен брой пъти, като всеки следващ етап увеличава детайлността и сложността на фрактала.

Задача 3: Разработване на алгоритъм, който визуализира триъгълник на Серпински. Задачата включва създаването на процедура, която при подаден единичен входен аргумент – цяло число N, генерира визуализация на фрактала до N-тото ниво на итерация. Фигурата по-долу (фиг. 6), визуализира до четвърто ниво на повторение.

Фиг. 6

Рекурсивният подход (фиг. 7), използва функция, която се самоизвиква, за да генерира всяко следващо ниво на фрактала, като ефективно прилага принципа на самоподобност, характерен за фракталите. Итеративният метод ще използва вложени цикли, за да създаде всяко ниво на фрактала, като постепенно добавя нови слоеве на детайлност.

Изучаването и програмното създаване на този фрактал предоставя на учениците уникална възможност да изследват концепциите на фракталната геометрия и самоподобността. Тази задача помага на учащите да развият умения за алгоритмично мислене и да прилагат както итеративни, така и рекурсивни подходи в програмирането. Работата с фрактал 'Триъгълник на Серпински' стимулира творческото мислене и предоставя практически пример за приложение на математически концепции в компютърните науки. Също така той, демонстрира как проста рекурсивна структура може да доведе до сложни и визуално впечатляващи модели, което е важно за разбирането на сложността и красотата на математическите и програмните конструкции.

```
private static void PrintSierpinski(int depth)
{
    int rows = 1 << depth;
    char[][] triangle = new char[rows][];
    for (int i = 0; i < rows; i++)
    {
        triangle[i] = new char[2 * rows - 1];
        for (int j = 0; j < 2 * rows - 1; j++)
            triangle[i][j] = ' ';
    }
    FillTriangle(triangle, depth, 0, rows - 1);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < 2 * rows - 1; j++)
        {
            Console.Write(triangle[i][j]);
        }
        Console.WriteLine();
    }
}

private static void FillTriangle(char[][] triangle, int depth, int top, int left)
{
    if (depth == 0)
    {
        triangle[top][left] = '#';
    }
    else
    {
        int offset = 1 << (depth - 1);
        FillTriangle(triangle, depth - 1, top, left);
        FillTriangle(triangle, depth - 1, top + offset, left - offset);
        FillTriangle(triangle, depth - 1, top + offset, left + offset);
    }
}
```

Фиг. 7

ЗАКЛЮЧЕНИЕ

Програмните решения на математически пъзели демонстрират значителни предимства в образователната сфера, предоставяйки средства за автоматизирано решаване на сложни задачи. Те са не само са полезни инструменти за учене, но и стимулират изучаването на разнообразни математически концепции. Итеративният метод развива линейно мислене и дава основа на алгоритмичното мислене, докато рекурсивният метод насърчава абстрактното мислене, като показва как повтарящите се процеси могат да решават комплексни проблеми.

Подходящият избор между итеративния и рекурсивния подход зависи от учебните цели и уменията на учениците, като комбинирането на двата подхода предлага балансирано образование. Овладяването на тези методи е ключово за развиване на критично мислене и умения за решаване на проблеми, които са важни във всички аспекти на ученето и професионалния живот.

БЛАГОДАРНОСТИ

Тази публикация отразява изследване от научен проект 23-ФПНО-02 „Изследване на ефективни механизми за управление на знанието, прилагани в софтуерното инженерство при създаване на проекти с Agile методологии“ – на фонд „Научни изследвания“ на Русенски университет „Ангел Кънчев“, 2023 г.

ЛИТЕРАТУРА

[1] Kalai G., “Three Puzzles on Mathematics, Computation, and Games”. ICM 2018 Rio de Janeiro, vol. 1 pp. 551-606, 2018.

[2] Nakov S., et al, “Fundamentals of Computer Programming With C#”. Sofia, 2013..

[3] Sulov V. (2016) Iteration vs Recursion in Introduction to Programming Classes: An Empirical Study. Cybernetics and Information Technologies, Vol.16 (Issue 4), pp. 63-72. https://doi.org/10.1515/cait-2016-0068.

[4] Rubio-Sánchez M., “Introduction to Recursive Programming”. CRC Press, 2018.

[5] Wellin P., Gaylord R., Kamin S., “An Introduction to Programming with Mathematica”. New York: Cambridge University Press, 2013.

[6] Наков П., Добриков П., 2015. Програмиране = ++ Алгоритми. (Nakov Preslav, Dobrikov Panayot, Programming = ++Algorithms. 2015).