Hryhoryeva V., Batareiev V., Kalchuk S.
State University of Economics and Technology: Kryvyi Rih, UA

COMBINATORIAL PROGRAMMING METHODS, AS A SET OF METHODS, TOOLS AND TECHNOLOGIES FOR CREATING PROGRAMS.

Григор´єва В.Г., Батареєв В.В., Кальчук С.О.
Державний університет економіки і технологій

КОМБІНАТОРНІ МЕТОДИ ПРОГРАМУВАННЯ, ЯК СУКУПНІСТЬ СПОСОБІВ, ЗАСОБІВ ТА ТЕХНОЛОГІЙ СТВОРЕННЯ ПРОГРАМ.

Abstract. In connection with the actualization and activation of the Olympiad movement in programming throughout the world in general and in Ukraine in particular, the problem of preparing students for participation in the Olympiads is becoming increasingly acute. Such activity is the main basis for the training of a certain specialist. Knowledge of Olympiad programming methods and methods provides a solid foundation for software creation.

Анотація. У зв'язку з актуалізацією та активізацією олімпіадного руху в програмуванні в усьому світі загалом та в Україні зокрема, все гостріше постає проблема підготовки студентів до участі в олімпіадах. Така діяльність є основною базою для підготовки певного фахівця. Знання методів та способів олімпіадного програмування дає тверду основу створення програмного забезпечення.

Математика є невід’ємною частиною програмування, і її застосування можна розглядати в багатьох різних аспектах. В Україні щороку проводяться олімпіади з програмування, де учні змагаються та демонструють свої вміння. Ці змагання включають різні математичні поняття, такі як алгоритми, структури даних, логічний аналіз і розв’язання задач. Використання математики має важливе значення для успішного виконання цих завдань, оскільки це допомагає учасникам знаходити найефективніші рішення, які призведуть до кращих результатів, ніж результати їхніх опонентів.

Для того, щоб ефективно підготуватися до цих змагань, українські учасники повинні добре розуміти математику, яка включає такі теми, як обчислення, лінійна алгебра та теорія ймовірностей серед інших. Опанувавши основи, вони можуть застосовувати їх під час пошуку рішень під час конкурсів або навіть під час щоденного написання коду на роботі чи в шкільних проектах, де їм потрібно придумати творчі ідеї, використовуючи математичні підходи, такі як методи оптимізації або методи динамічного програмування.

В роботах представлених для розглядання зачасту використовують задачі, пов’язані з циклами, ітераціями для обчислення.

Комбінаторна методика підходу до задачі зазвичай дає реальний результат в найкоротший час.

Комбінаторні методи багато в чому мають евристичний характер, індивідуальність як різних класів, так окремих завдань. Причому нерідко, що оригінальнішим є такий метод, то ефективніше завдання вирішується на ЕОМ.

Комбінаторне програмування — це тип підходу до розв’язування задач, який поєднує елементи з різних областей математики та інформатики для оптимізації розв’язків складних задач. Ця техніка використовується в різних галузях промисловості, таких як машинобудування, фінанси, логістика та дослідження операцій. Комбінаторне програмування фокусується на пошуку найкращого рішення шляхом розгляду всіх можливих комбінацій у межах заданих обмежень або параметрів.

Першим кроком у комбінаторному програмуванні є ідентифікація проблеми та чітке визначення її цілей. Потім важливо встановити можливі обмеження, щоб розглядалися лише дійсні рішення; інакше буде згенеровано занадто багато можливостей, що може призвести до неоптимальних результатів або навіть до неправильних. Після завершення цих двох кроків можна використовувати алгоритми, які швидко шукають усі потенційні комбінації, враховуючи також будь-які переваги, указані користувачем щодо певних критеріїв, таких як вартість або рівень ефективності, бажані результати тощо. Нарешті має з’явитися оптимальне рішення з від початку до кінця потрібно мінімум зусиль!

Загальне комбінаторне програмування пропонує потужний спосіб ефективного розв’язання складних проблем без необхідності покладатися виключно на евристики чи припущення – те, з чим традиційні методи часто стикаються через їх обмежений обсяг під час роботи з великими наборами даних і кількома змінними, задіяними одночасно. Крім того, цей підхід забезпечує гнучкість, оскільки користувачі можуть налаштовувати власні цільові функції залежно від конкретних потреб і вимог, які вони можуть мати; що робить його ідеальним для вирішення унікальних сценаріїв, де інші методи можуть з тріском провалитися!

Під комбінаторними методами програмування розуміємо сукупність способів, засобів, технологій створення програм. До них відносяться: перебір, рекурсія, динамічне програмування та перебір з відходом назад.

Хочеться докладно розглянути саме методи динамічного програмування та рекурентних формул, оскільки саме із застосуванням цих методів і пов'язано найбільшу кількість помилок та розбіжностей.

Рекурентний підхід та динамічне програмування є повністю взаємозамінними методами рішення. Будь-яке завдання, задане рекурентною формулою, можна вирішити будь-яким із представлених методів. Тому в ході вирішення кожної рекурентної задачі виникає питання доцільності

використання будь-якого методу. Адже від успішного вибору залежить швидкість виконання програми, легкість її читання та ефективність використання ресурсів комп'ютера.

Рекурсивними називаються функції, що викликають самі себе. Рекурсія дозволяє просто (без використання циклів) програмувати обчислення функцій, заданих рекурентно. Наприклад, загальновідома функція факторіалу f(n) = n! f (0) = 1; f(n) = n*f(n – 1).

Ця операція наочно розбиває завдання кілька менших під-задач, рішення кожної у тому числі призводить до вирішення вихідної завдання. Отже, запущена функція хіба що «розгортається» до вирішення тривіальної завдання (для факторіалу – випадок, коли n=1 чи n=0), та був «згортається» у вихідної. Але бачимо, такі дії цілком збігаються з визначенням методу динамічного програмування. Різниця лише у "розгортанні".

Першим кроком у створенні рекурсивної функції є визначення базового сценарію, який, по суті, є місцем, де ми встановлюємо точку зупинки перед тим, як продовжити подальші обчислення. Після того, як це буде встановлено, нам потрібно визначити, як кожна послідовна ітерація буде просуватися до досягнення нашого базового сценарію - зазвичай через певний тип структури циклу або за допомогою умовних операторів, таких як оператори if/else і перемикання випадків залежно від того, які дані потребують обробки. Крім того, будь-які додаткові змінні, необхідні під час рекурсії, також мають бути визначені заздалегідь, щоб вони були доступні, коли це необхідно протягом усього часу виконання (тобто параметри).

Після того, як усі ці компоненти будуть належним чином налаштовані та протестовані окремо в їх власних межах (якщо необхідно), ви будете готові почати писати свій фактичний код рекурсивної функції! Тут вам слід подумати про те, які операції потрібно виконувати на кожному рівні, наприклад, додавання/віднімання тощо), а також переконатися, що ваш результат правильно збігається між рівнями - таким чином ви не зіткнетеся з такими проблемами, як переповнення стеку через неправильне використання повертати значення з попередніх викликів тощо... Але якщо реалізація виконана належним чином, рекурсія може значно заощадити час розробки порівняно з іншими рішеннями, такими як цикли та ітерації!

У програмному забезпеченні переповнення стека (англ. stack overflow) виникає, коли в стеку викликів зберігається більше інформації, ніж може вмістити. Зазвичай ємність стека задається під час старту програми/потоку. Коли покажчик стека виходить за межі, програма аварійно завершує роботу.

Рекурсивні функції є важливою частиною математики, оскільки вони надають спосіб розв’язувати складні проблеми в стислій формі. Рекурсивна функція — це така, яка викликає сама себе у своєму власному визначенні; це дозволяє повторювати ту саму операцію кілька разів з різними вхідними даними. Це робить його ідеальним для розв’язування математичних формул, які включають повторювані операції, наприклад знаходження суми всіх чисел від 1 до n або обчислення чисел Фібоначчі. Рекурсивні функції також

можна використовувати для більш складних завдань, таких як алгоритми сортування та алгоритми обходу дерева.

Синтаксис рекурсивних функцій може змінюватися залежно від мови, яка використовується, але загалом слідує деяким основним принципам: по-перше, має бути початковий стан, де рекурсія не відбувається; по-друге, кожен наступний виклик повинен зменшувати проблему, розбиваючи її на більш дрібні частини до досягнення базового випадку, який не потребує подальшої рекурсії; нарешті, кожен виклик повинен повертати певне значення на основі того, що було обчислено на кожному кроці. Дотримуючись цих кроків, ми можемо створити потужні рішення, які в іншому випадку зайняли б набагато більше часу за допомогою традиційних методів, таких як цикли та оператори if-else.

Підсумовуючи, рекурсивні функції є безцінними інструментами під час роботи зі складними математичними формулами завдяки їхній здатності розбивати великі проблеми на керовані блоки, водночас швидко й ефективно повертаючи значущі результати, не вимагаючи надто багато коду чи ресурсів для ефективного виконання цього завдання. Таким чином, вони залишаються популярними серед математиків, яким потрібні швидкі надійні рішення без шкоди для точності чи продуктивності, що робить їх основними компонентами будь-якої сучасної системи програмного забезпечення, що має справу з математичними обчисленнями!

Таким чином, рекурсивний метод проходить лише необхідні вузли завдання, стільки разів, скільки вони трапляються. Метод динамічного програмування навпаки, вирішує всі під-завдання, навіть ті, які не потрібні для вирішення вихідного завдання, але кожен вузол гарантовано буде пройдено лише один раз.

Динамічне програмування в свою чергу, потужна математична техніка для вирішення складних задач. Він передбачає розбиття проблеми на менші підпроблеми, пошук рішень для кожної з підпроблем, а потім об’єднання цих рішень для отримання оптимального рішення вихідної проблеми. Цей підхід застосовувався в багатьох різних галузях, таких як економіка, інженерія, інформатика та дослідження операцій.

Сучасні методи зробили динамічне програмування більш ефективним завдяки використанню складних математичних методів, таких як лінійна алгебра та алгоритми оптимізації на основі обчислень, які дозволяють нам вирішувати великі проблеми з більшою точністю, ніж будь-коли раніше. Наприклад, сучасне динамічне програмування можна використовувати в програмах машинного навчання, де воно допомагає оптимізувати параметри нейронних мереж або підтримувати векторні машини на основі навчальних наборів даних. Крім того, його також можна використовувати у фінансах під час впровадження моделей оптимізації портфеля або точного визначення ціни похідних інструментів за невизначених ринкових умов.

Загальне динамічне програмування є безцінним інструментом для вирішення складних проблем реального світу, які вимагають складних процесів прийняття рішень завдяки його здатності швидко визначати оптимальні рішення шляхом систематичного дослідження всіх можливих результатів з урахуванням певних обмежень і цілей, визначених користувачами. Сучасні методи дозволяють цій потужній техніці стати ще кориснішою завдяки підвищеній обчислювальній ефективності, що робить її застосовною в різних дисциплінах, включаючи економіку, інженерію, інформатику, математику тощо.

Аби в повній мірі перевірити здатність учасника олімпіади до програмування та застосувань динамічного програмування та рекурсивних функцій є набори стартових задача.

В області невід'ємних чисел простішими, як правило, вважаються меншими за величиною числа. Наприклад, для N більше простими є числа N-1, N-2 тощо. Очевидно, таке зменшення числа можна зробити лише кінцеву кількість разів – до 0.

Приклад 1. Потрібно описати рекурсивно функцію f(x,n), що обчислює величину $x^n/n!$ при будь-якому речовому x і будь-якому невід'ємному цілому n.

Рішення. Насамперед, згідно з рекомендаціями, обчислення f(x,n) треба звести до підзадачі - до обчислення $x^k/k!$ при деякому k<n. Яке саме k взяти? Можна, наприклад, вибрати k=n-1, т.к. за $x^{n-1}/(n-1)!$ легко отримати $x^n/n!$ Оскільки ми описуємо f(x,n) у

У положенні, що функція правильно обчислює $x^k/k!$ при будь-якому k<n, то в описі функції ми повинні обчислювати x n-1 /(n-1)! рекурсивно звернутися до f(x,n-1), а потім отриману величину помножити на x/n, щоб отримати значення f(x, n). Нерекурсивний випадок – обчислення f(x,0),т.к. обчислення $x^0/0!$ не можна звести до обчислення $x^k/k!$ при k<0.

Отже, отримуємо опис нашої функції мовою Паскаль:

```
function f(x:real; n:integer): real; {n ≥ 0}
begin
if n=0 then f:=1 else f:=x/n*f(x,n-1)
end;
```

Може виникнути питання: якщо вихідне завдання можна звести до різним підзавдань, за відповіддю кожної з яких можна побудувати відповідь вихідного завдання, то яке з цих підзавдань вибрати? В данному прикладі звели обчислення f(x,n) до обчислення f(x,n-1), тобто. зменшили другий параметр на 1. А чи можна було вибрати іншу підзадачу, наприклад, обчислення f(x,n-2)? Так, можна, оскільки за відповіддю цієї підзавдання легко виходить відповідь вихідного завдання:

$$f(x,n)=\frac{x^2}{n\cdot(n-1)}\cdot f(x,n-2)$$

Але за такого вибору нам доведеться вказувати дві нерекурсивні гілки – за n=0 і n=1, т.к. при цих значеннях n вираз x^{n-2} /(n-2)! безглуздо. У результаті отримуємо:

```
function f(x:real; n:integer): real; {n ≥ 0}
begin if n=0 then f:=1 else
 if n=1 then f:=x else f:=x*x/n/(n-1)*f(x,n-2)
end;
```

Цей приклад показує, що при виборі підзавдання, до якого ми зводимо вихідне завдання, небажано занадто «далеко» йти від вихідного завдання, краще брати підзавдання, найбільш «близьке» до нього.

Однак буває і так, що за відповідями «близьких» завдань не вдається побудувати відповідь вихідного завдання, і тому доводиться використовувати досить «далеку» підзавдання. Розглянемо пару відповідних прикладів

Приклад 2. Не використовуючи операції множення та поділу, рекурсивно описати функцію M(a,b) від цілих чисел a і b (a≥0, b>0), яка обчислює залишок від поділу a на b, тобто. M(a,b)=a mod b.

Рішення. Тут насамперед треба визначитися з тим, яким параметру функції M будемо вести рекурсію, тобто. який параметр спрощуватимемо. Вести рекурсію за другим параметром b не можна, оскільки ніякої хорошої залежності між a mod b і a mod c, де c<b, ні. Тому рекурсію будемо вести за першим параметром a.

При a≥b правильна рівність a mod b = (a-b) mod b. Тому якщо ми зможемо обчислити M(a-b,b), то зможемо обчислити і M(a,b) – ці величини збігаються; отже, в даному випадку вихідне завдання треба звести до підзадачі, де величина a зменшена не на 1, а на b.

Що стосується нерекурсивного випадку, то він виникає при a<b, оскільки різниця a-b виходить з області допустимих значень першого параметр функції. І тут відповіддю є саме число a mod b = a. Отже, отримуємо такий опис нашої функції:

```
function M(a, b:integer): integer; {a≥0, b>0}
begin if a<b the M:=a else M:=M(a-b,b) end;
```

Приклад 3. Описати рекурсивну функцію degree5(N), яка обчислює, яким ступенем числа 5 є натуральне число N. Якщо N не ступінь п'яти, функція має повернути число -1. Наприклад, degree5(50) = -1, degree5(125) =3, degree5(5) =1, degree5(1) =0.

Рішення. Ідея циклічного розв'язання задачі зрозуміла: отже ділити наше число на 5, поки це можливо. В результаті серії таких поділів ми або

дійдемо до одиниці (це позитивний результат, при якому шуканий показник відповідає числу виконаних поділів), або в якийсь момент з'ясується, що ділити націло на 5 далі неможливо (негативний результат із відповіддю -1).

Подібні міркування покладемо і в основу рекурсивного рішення завдання. При цьому зауважимо, що якщо величина N div 5 – ступінь п'ятірки з показником k, очевидно, як і число N – теж ступінь п'ятірки, але із показником k+1. Також зауважимо, що одиниця – це найменша ступінь п'ятірки з показником 0.

Звернемо увагу, що вихідне завдання для числа N слід звести до підзадачі для числа N div 5 лише тоді, коли наше число N ділиться націло на 5 (бо в цьому випадку залишається надія на позитивний результат розв'язання задачі), інакше відповідь вже зрозуміла (-1). Спрощення завдання тут йде у напрямку переходу до ступеня з меншим показником та спроби дійти до ступеня з мінімальним показником.

Нерекурсивних випадків тут два: перший виникає за N=1 (позитивний результат із відповіддю 0), другий – якщо N вдалося поділити націло на 5 (негативний результат із відповіддю -1).

```
function degree5(N: integer): integer; {N>=1}
var k: integer;
begin if N=1 then degree5:=0 else
if N mod 5 <> 0 then degree5:= -1 else
begin k:=degree5(N div 5);
if k=-1 then degree5:=-1
else degree5:=k + 1
end
end;
```

Ми розглянули випадки, де значенням параметрів були негативні цілі числа, а тепер розглянемо випадок будь-яких цілих чисел.

Приклад 4. Рекурсивно описати функцію pow(x,n), яка обчислює x^n для будь-якого речовинного x (≠0) та будь-якого цілого n.

Рішення. Оскільки $x^n=x\cdot x^{n-1}$, то, здавалося б, обчислення x^n потрібно зводити до обчислення x^{n-1}. Однак, це не так.

Особливість цього завдання полягає в тому, що другий параметр (n) функції pow, яким будемо вести рекурсію, може бути будь-яким цілим числом, а в області цілих чисел процес спрощення числа n шляхом вичитування з нього 1 нескінченний: n, n-1, …, 1, 0, -1, -2, … Тому ми ніколи не дійдемо до такого значення n , при якому рекурсія зупиниться. З урахуванням цього, в області цілих чисел треба якось інакше визначити поняття простішого числа. Зробити це можна по-різному.

Можливий варіант: розглянути в області цілих чисел дві за добластю – позитивні та негативні числа, і в кожній з них використовувати своє

розуміння більш простого числа, вважаючи простішим числом те, що ближче до 0:

n>0: n, n–1, n–2, …, 2, 1 (тут n–1 простіше n)
n<0: n, n+1, n+2, …, -2, -1 (тут n+1 простіше n)

А для n=0 можна дати відповідь: pow(x,0) = 1. Якщо так і зробити, то отримаємо наступну рекурсивну формулу:

$$x^n = \begin{cases} 1, & n=0 \\ x \cdot x^{n-1}, & n>0 \\ x^{n+1}/x, & n<0 \end{cases}$$

У цьому випадку опис функції pow(x,n) виглядає так:

```
function pow(x:real; n:integer): real; {x<>0}
begin if n=0 then pow:=1 else
 if n>0 then pow:=x*pow(x,n-1)
 else pow:=pow(x,n+1)/x
```

end;Інший варіант: оскільки вірна формула $x^{-n}=1/x^n (n>0)$, то можна при негативному показнику перейти до позитивного згідно цій формулі, а потім традиційно спрощувати позитивне число, віднімаючи з нього 1. Тут для негативного числа простішим вважається його модуль, а для позитивного – число, на 1 менше його.

Це дає таку рекурсивну формулу:

$$x^n = \begin{cases} 1, & n=0 \\ 1/x^{|n|}, & n<0 \\ x \cdot x^{n-1}, & n>0 \end{cases}$$

І тоді опис функції pow виглядає так:

```
function pow(x:real; n:integer): real; {x≠0}
begin if n=0 then pow:=1 else
 if n<0 then pow:=1/pow(x,abs(n))
 else pow:=x*pow(x,n-1)
end;
```

Можна придумати і якісь інші визначення більш простого

числа в ділянці цілих чисел, але в будь-якому випадку треба уважно стежити за тим, щоб процес спрощення був кінцевим.

Використання математичних методів стає дедалі важливішим у зв’язку зі зростаючою складністю, пов’язаною з сучасними процесами

розробки програмного забезпечення, тому знання цього предмета дає перевагу перед тими, хто не має подібного рівня знань, конкуруючи з іншими програмістами з усього світу. такі престижні події, як Українські олімпіади з програмування, де перемагають лише найкращі!

Література

1.Елементи математичної логіки та теорії рекурсії : навч. посіб. / М. Комарницький, В.Андрійчук , І.Мельник. – Львів : ЛНУ імені Івана Франка, 2013.

2.Алгоритмічна мова Паскаль: Навчальний посібник для студентів бакалаврату напрямку електроніка/ Уклад. Д.Д. Татарчук. – ІВЦ “Політехніка”,. 2006

3.C. A. R. Hoare: "Notes on data structuring". In O.-J. Dahl, E. W. Dijkstra and C. A. R. Hoare, editors, Structured Programming, pages 83–174. Academic Press, 1972.

4.Niklaus Wirth: Algorithms + Data Structures = Programs. Prentice-Hall, 1975, ISBN 0-13-022418-9.

5.Rubio-Sanchez, Manuel (2017). Introduction to Recursive Programming. CRC Press. ISBN 978-1-351-64717-5.

6..Dijkstra, Edsger W. (1960). "Recursive Programming". Numerische Mathematik. 2 (1): 312–318. doi:10.1007/BF01386232. S2CID 127891023.

7. Алгоритми і складні структури мов програмування з використанням математичного аналізу: https://www.mathsisfun.com/sets/function.html

8. Сайт Всеукраїнських олімпіад з інформатики:https://oi.in.ua/