тощо. Найвідомішими серед них є алгоритми Форда-Фалкерсона, Едмондса-Карпа та Дініца. Вони широко застосовуються для поліпшення людського побуту і значно полегшують наше життя.

Список літератури

1. *Ключка Т.А Комп'ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона 2020, с. 5, с. 25*
2. *Harris, T. E.; Ross, F. S. (1955). "Fundamentals of a Method for Evaluating Rail Net Capacities" (PDF). Research Memorandum. Archived from the original (PDF) on 8 January 2014.*
3. *Ford, L. R.; Fulkerson, D. R. (1956). "Maximal flow through a network". Canadian Journal of Mathematics. 8: 399–404*
4. *Yefim Dinitz (2006). "Dinitz' Algorithm: The Original Version and Even's Version". In Oded Goldreich; Arnold L. Rosenberg; Alan L. Selman (eds.). Theoretical Computer Science: Essays in Memory of Shimon Even. Springer. pp. 218–240*
5. *Ключка Т.А Комп'ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона 2020, с. 8, с. 5*

УДК 004.42

Ватаманеску С. В., студент 1 курсу спеціальності 122 «Комп'ютерні науки»
Ніколюк П. К., професор, професор кафедри інформаційних технологій

РЕКУРСИВНІ АЛГОРИТМИ

Донецький національний університет імені Василя Стуса, м. Вінниця

Вступ

Найважливішим навиком, яким повинен володіти фахівець з комп'ютерних наук, є вміння логічно мислити, яке, з одного боку, дано від природи, з іншого боку потребує старанної праці та відшліфовування цієї навички. Успішний майстер – це той, хто уміє мислити. Тому важливою складовою процесу підготовки фахівців ІТ-сфери є вироблення знань та навичок, які стосуються розуміння та використання сучасних моделей та методів обробки, аналізу та перетворення дискретної інформації.

Наразі, ми розберемо тему «Рекурсивних алгоритмів», де розглянемо його сенс, принцип роботи, використання та розуміння цього розділу.

Рекурсія, рекурсивні алгоритми: сенс і синтаксис

Рекурсія – це є фундаментальним поняттям у математиці та інформатиці. У мовах програмування *рекурсивна програма* — це програма, яка посилається на

саму себе (як і в математиці, рекурсивна функція визначається поняттям самої функції).

Основні відмінності алгоритму, що припускає рекурсивне рішення:

1) Є алгоритм, який потрібно виконати кілька разів;
2) Алгоритм вимагає змінних даних щоразу;
3) Алгоритм не обов'язково повинен змінюватись щоразу;
4) Є фінальна умова: рекурсивний алгоритм – не нескінченний.

У випадку, не можна стверджувати, що одноразове виконання – обов'язкова умова відсутності приводу для рекурсії. Не можна також вимагати наявності обов'язкової фінальної умови: нескінченні рекурсії мають свою область застосування.

Алгоритм рекурсивний: коли послідовність операцій виконується багаторазово, на даних, що змінюються щоразу і дає новий результат.

Категорії завдань, що допускають рекурсивні визначення:

1) Завдання, математичні моделі яких записуються у вигляді рекурсивно певних функцій.
2) Структура даних завдання визначається рекурсивним чином. Наприклад: графи, дерева, списки.
3) Методи вирішення задачі допускають рекурсивне визначення (ігри, головоломки та ін.).

Явна рекурсія характеризується наявністю оператора з посиланням на самого себе у визначенні підпрограми.

Неявна (непряма) рекурсія характеризується тим, що одна підпрограма посилається на іншу підпрограму, яка рекурсивно посилається на першу підпрограму через ланцюжок викликів другої підпрограми.

Що є рекурсією та її приклади

Рекурсивним називається будь-який об'єкт, який частково визначається через себе.

Наприклад, наведене нижче визначення двійкового коду є рекурсивним:

<двійковий код> ::= <двійкова цифра> | <двійковий код><двійкова цифра> <двійкова цифра> ::= 0 | 1

(*Для опису поняття були використані так звані металінгвістичні формули Бекуса-Наура (мова БНФорма); знак "::=" означає "за визначенням є", знак "|" - "або".)*

Приклад рекурсивних програм. У ряді випадків рекурсивну підпрограму можна побудувати безпосередньо з формального математичного опису задачі.

Факторіал	
$\begin{cases} n! = n * (n-1)!, & \text{при } n > 0 \\ 1, & n = 0 \end{cases}$	*Function Fact(n:byte) :longint;* *begin* *if n = 0* *then Fact :=1* *else Fact :=n*Fact (n-1)* *end;*
Числа Фібоначчі	

$\begin{cases} F_n = F_{n-1} + F_{n-2}, \text{ при } n > 1 \\ F_0 = 1, \ F_1 = 1 \end{cases}$	*Function F(n:byte) :longint;* *begin* *if n <= 1* *then F:=1* *else F:= F(n-1) + F(n-2)* *end;*

Складність рекурсивних обчислень

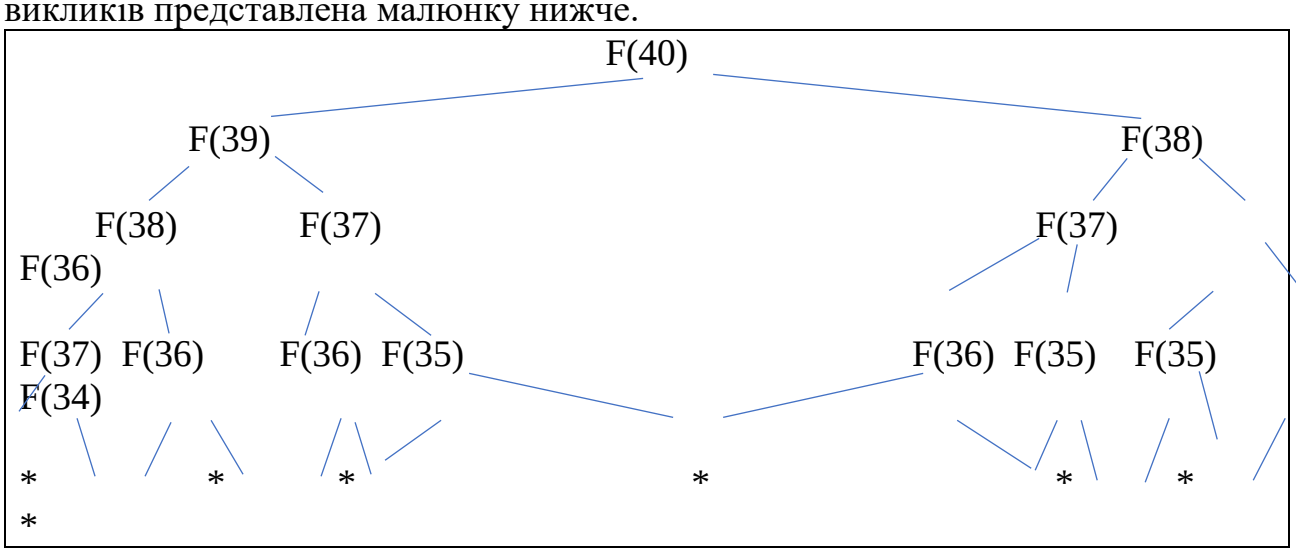
Отже, ***рекурсією*** називається такий спосіб організації обробки даних, при якому програма (або функція) викликає сама себе або безпосередньо, або з інших програм (функцій).

Ітерація – такий спосіб організації обробки даних, при якому деякі дії багаторазово повторюються, не призводячи до рекурсивних викликів програм (функцій).

Теорема. Довільний алгоритм, реалізований у рекурсивній формі, може бути переписаний в ітераційній формі і навпаки.

При відносній простоті написання у рекурсивних підпрограм часто зустрічається істотний недолік - неефективність. Так, порівнюючи швидкість обчислення чисел Фібоначчі за допомогою ітеративної та рекурсивної функції можна помітити, що ітеративна функція виконується майже миттєво, незалежно від значення n. При використанні рекурсивної функції вже при n = 40 помітна затримка при обчисленні, а при великих n результат з'являється дуже не скоро.

Неефективність рекурсії в тому, що одні й ті самі обчислення виробляються багато разів. Так, для обчислення 40-го числа Фібоначчі схема рекурсивних викликів представлена малюнку нижче.



Ви можете використовувати рекурентне відношення для оцінки складності рекурсивного обчислення (кількості рекурсивних викликів).

Рекурентне відношення — це рекурсивна функція з цілими значеннями. Значення будь-якої такої функції можна визначити шляхом обчислення всіх її

значень, починаючи з найменшого значення, використовуючи обчислене значення на кожному кроці для обчислення поточного значення.

Як приклад, можемо розглянути різницю між рекурсивною та ітеративною реалізацією. *(Ітеративний варіант функцій реалізовуватимемо за допомогою оператора циклу* ***for****.)*

Факторіал числа. Факторіалом цілого невід'ємного числа n називається добуток усіх натуральних чисел від 1 до n і позначається n! Якщо *f(n) = n!*, то має місце рекурентне співвідношення: $n! = \begin{cases} f(n) = n * f(n-1) \\ f(0) = 1 \end{cases}$

Перша рівність визначає крок рекурсії – спосіб обчислення *f(n)* через *f(n – 1)*. Друга рівність вказує, коли під час обчислення функції слід зупинитися. Якщо його не поставити, то функція працюватиме нескінченно довго.

Наприклад, значення *f(3)* можна обчислити так:

$$f(3) = 3 * f(2) = 3 * 2 * f(1) = 3 * 2 * 1 * f(0) = 3 * 2 * 1 * 1 = 6$$

Вочевидь, що з обчислення *f(n)* слід здійснити n рекурсивних викликів.

Рекурсивна реалізація	Циклічна реалізація
int f(int n) *{* *if (!n) return 1;* *return n*f(n-1);* *}*	*int f(int n)* *{* *int i, res = 1;* *for (i = 1, i <= n, i++) res = res * 1;* *return res;* *}*

А сама ідея циклічної реалізації полягає у безпосередньому обчисленні факторіалу числа за допомогою оператора циклу:$f(n) = 1 * 2 * 3 * \ldots * n$

Висновок

Рекурсивний алгоритм є універсальним засобом вирішення різноманітних алгоритмічних задач. Результати показують, що будь-яке подібне завдання має рекурсивне рішення, яке водночас відрізняється складністю та простотою, яку сприймає людина.

Рекурсивні алгоритми часто мають нижчу асимптотичну складність, ніж еквівалентні їм ітераційні. Тобто теоретично вони швидші.

Розвиток сучасних програмних засобів зробив практичне використання рекурсії досить простим, а нові концепції та методи програмування подолали неефективність рекурсивних програм через необхідність викликати велику кількість квантових процедур.

Звичайно, рекурсивні алгоритми не слід вважати панацеєю від усіх професійних хвороб програмістів. Але в той же час не варто применшувати їх значення. Головне швидко та якісно знайти розв’язок задачі, тут слід враховувати можливість використання рекурсивних алгоритмів.

Список літератури:

1. *Що є дискретна математика? URL: https://lobachevsky.su*

2. *Коцовський В. М. конспект лекцій «Дискретна математика та теорія алгоритмів. Частина 1» - Ужгород , 2016.*
3. *Давидов, В.Г. Програмування та основи алгоритмізації [Текст]/В.Г. Давидов - М.: Вища школа, 2003*
4. *Бондарчук Ю. В., Олійник Б. В. посібник «Основи дискретної математики» (для студентів-інформатиків) – Київ, 2007*
5. *Алгоритм рекурсивний: опис, аналіз, особливості та приклади URL: https://presa.com.ua/navchannia/algoritm-rekursivnij-opis-analiz-osoblivosti-ta-prikladi*

УДК 519.1

Зимич А. П., студент 1 курсу
спеціальності 122 «Комп'ютерні науки»
Ніколюк П. К., д-р фіз.-мат. наук,
Професор кафедри комп'ютерних наук

АЛГОРИТМ ФОРДА-ФАЛКЕРСОНА. ЗНАХОДЖЕННЯ МАКСИМАЛЬНИХ ПОТОКІВ В ГРАФАХ

Донецький національний університет імені Василя Стуса, м. Вінниця

Поняття потоку та розрізу

Коли ми говоримо про потоки та розрізи у графах, ми зазвичай маємо на увазі орієнтовані графи, де кожне ребро має напрямок.

Потік в орієнтованому графі – це функція, яка надає кожному ребру графа невід'ємне число, яке ми називатимемо "потоком". При цьому необхідно задовольняти наступні дві умови:

- Пропускна здатність: потік через кожне ребро не може перевищувати його пропускну здатність.
- Закон збереження потоку: кількість потоку, що втікає у вершину, має дорівнювати кількості потоку, що випливає з цієї ж вершини, за винятком джерела та стоку.

Джерелом ми називаємо вершину графа, з якої починається потік. Стоком ми називаємо вершину, в яку потік повинен потрапляти.

Максимальний потік у графі – це максимальна кількість одиниць потоку, яка може пройти від витоку до стоку. Іншими словами, це максимальна кількість товарів, інформації чи будь-яких інших одиниць, які можуть бути перевезені за графом від витоку до стоку.

Мінімальний розріз у графі – це мінімальна кількість ребер, які необхідно видалити з графа, щоб розділити його на дві частини, що не перетинаються, що містять виток і стік відповідно. Іншими словами, розріз - це безліч ребер, які, якщо їх видалити з графа, призведуть до того, що джерело і стік будуть у різних компонентах зв'язності.

Мінімальний розріз завжди дорівнює максимальному потоку у графі (малюнок 1), це випливає з теореми Форда-Фалкерсона. Отже, знаходження