Комбінаторні задачі та складність обчислень. Масові задачі, алгоритми та складність. Задачі розпізнавання, мови та кодування. Детерміновані машини Тюрінга та клас P. Недерміновані машини Тьюрінга та клас NP. Поліноміальна звідність і NP-повні задачі. Теорема Кука.

Масові задачі, алгоритми та складність

Ми розглядатимемо функції вигляду $t: \bar{A} \to \bar{B}$, де A і B — деякі алфавіти. Саме цей клас функцій виникає, коли ми цікавимось функціями, які можна ефективно обчислювати. Наприклад, функцію піднесення до квадрату у множині невід'ємних цілих чисел з обчислювальної точки зору природньо трактувати як функцію із множини {0,1,..., 9}* в себе. Слід лише домовитись, як бути з десятковими словами, що починаються цифрою 0. їх можна утотожнити з невід'ємними цілими десятковими числами, що отримуються після відкидання перших нулів. А можна вважати, що функція відображає слова, які не є десятковим записом натурального числа, у 0. Якщо функція двомісна, на зразок додавання натуральних чисел, то для розділення аргументів зручно ввести в алфавіт новий спеціальний символ, як от #.

Задача обчислення функції $t: \bar{A} \to \bar{B}$ полягає у знаходженні для вказаного слова $w \epsilon \bar{A}$значення функції $f(w)$. Ми будемо описувати задачі обчислення наступним чином:

Задано:	$w \epsilon \bar{A}.$
Обчислити:	$f(w)$

Втім, коли нам не буде настільки важливо, в якому алфавіті і як саме кодуються аргументи та значення функції, ми допускатимемо і менш формальні описи задач на кшталт

Задано:	$x \epsilon N$
Обчислити:	x^2

Іноді задачу у вищеозначеному сенсі ми будемо називати масовою. Конкретне задане $w \epsilon \bar{A}$ називатимемо індивідуальною задачею. Значення $f(w)$ будемо називати розв'язком індивідуальної задачі w . Масову задачу можна вважати нескінченним набором індивідуальних задач.

Нехай $L \epsilon \bar{A}$ – множина слів у алфавіті A*. Часто L називають мовою. Задача розпізнавання мови L полягає у визначенні, належить задане слово $w \epsilon \bar{A}$ цій мові, чи ні:

Задано:	$w \epsilon \bar{A}.$
Розпізнати:	$w \epsilon L$

Наприклад:

Задано:	$x \epsilon N$
Розпізнати:	чи є xповним квадратом ?

Індивідуальною задачею є будь-яке задання слова w. Розв'язком індивідуальної задачі є відповідь "так" чи "ні", яку прийнято кодувати символами 1 та 0 відповідно.

Ще один різновид задачі, який нам буде траплятися, називається задачею пошуку елемента із заданою властивістю. Нехай алфавіт A не містить символу # і $P \subseteq (A)\{\#\})^*$. Для кожного заданого $w \in A^n$ задача полягає або у знаходженні хоча б одного $u \in A^n$ такого, що $w \ \# \ u \in p$, або у констатації, що елемента u з такою властивістю немає:

Задано:	$w \in A^n$
Знайти:	uтаке, що $w\#u \in P$

Індивідуальною задачею є довільно задане слово $w \in A^n$, а її розв'язком є або відповідне u, або відповідь "не існує", яку зручно кодувати символом #. Наприклад:

Задано:	$x \epsilon N$
Знайти:	$y \in Z$, таке, що $x = y^2$

Ми ввели три типи задач – обчислення, розпізнавання та пошуку. Насправді, в обчислювальному відношенні всі вони рівносильні між собою – кожну задачу одного типу можна переформулювати як задачу будь-якого з двох інших типів.

Так, задача розпізнавання мови $L \subseteq A^n$ є задачею обчислення її характеристичної функції $X_L: A^n \to (0,1)$, що набуває значення 1 на аргументах з L і лише на них. Задачу обчислення функції $t: A^n \to B^n$ легко подати як задачу пошуку, взявши $P = (w)\#f(w): w \epsilon A^n$, де P – множина слів у алфавіті $A0B0$ #. Нарешті, кожну задачу пошуку можна звести до деякої задачі розпізнавання .

Поняття алгоритму в математиці таке ж давнє і фундаментальне, як, скажімо, поняття числа. Наприклад, алгоритмові Евкліда не менш як 22 сотні років. Подібно до того, як розвивалось уявлення про дійсні числа, від відкриття давньогрецькими математиками не співмірності довжин сторони квадрата і його діагоналі і аж до чітких конструкцій та означень, розроблених у другій половині 19 століття Г. Кантором, Ш. Мере, Р. Дедекіндом та К. Ваєрштрасом, поняття алгоритму теж уточнювалось і було строго означене у першій половині нашого століття. Перші формалізації запропонували у 1936 році Е. Пост і А. Тюрінг, випередивши появу обчислювальної техніки, яка значною мірою ґрунтується на тій же ідеології. Згодом були запропоновані й інші означення, зокрема А. А. Марковим та А. М. Колмогоровим, і всі вони виявились еквівалентними між собою. Формалізація поняття алгоритму є видатним досягненням математичної логіки, оскільки лише з появою строгого означення алгоритму стало можливим доводити для певних задач, що жоден алгоритм не здатен їх розв'язати. Такі задачі називаються алгоритмічно нерозв'язними. Так, задача розпізнавання, чи заданий многочлен від кількох змінних з цілими коефіцієнтами має цілі нулі, є алгоритмічно нерозв'язною (десята проблема Гільберта). Зазначимо для порівняння, що у випадку многочленів від однієї змінної для подібної задачі алгоритм відомий.

Коли йтиметься про алгоритм, то вважатимуться зафіксованими два алфавіти A – вхідний та вихідний B. Робота алгоритму полягає в тому, що він отримує на вхід слово у вхідному алфавіті – *вхід*, і як результат виконання послідовності елементарних операцій, подає на вихід слово у вихідному алфавіті – *вихід*. Поняття елементарної операції або кроку роботи є складовою формального означення алгоритму.

Алгоритм розв'язує масову задачу, якщо, отримавши на вхід будь-яку індивідуальну задачу $w \in A^n$, він за скінченну кількість кроків подає на вихід її розв'язок. Залежно від типу задачі говоримо, що алгоритм обчислює функцію, розпізнає множину чи мову, знаходить елемент із певною властивістю. У випадку задачі розпізнавання, якщо алгоритм подає на вихід 1, то кажемо, що він приймає вхід, а якщо 0, то кажемо, що алгоритм відхиляє вхід.

Довжиною входу $w \in A^n$ називається кількість букв у слові w, яку ми домовились позначати як $|w|$.

Нехай $t: N \to \bar{N}$ – деяка функція. Алгоритм розв'язує задачу за час t, якщо на кожному

вході w він робить не більше, ніж $t(w)$кроків. Якщо $t(n) \leqslant cn^0$ для деякої константи c, то алгоритм розв'язує задачу за поліноміальний від довжини входу час. Такий алгоритм називають поліноміальним, а задачу поліноміально розв'язною.

Алгоритм, який на нескінченній послідовності входів робить більше як 2^{n^0}кроків, де n – довжина входу, а $c > 0$ – деяка константа, називається *експоненційним*. Про такий алгоритм кажуть, що він вимагає експоненційного часу. Експоненційні алгоритми відповідають загальним уявленням про повільні, неефективні на практиці алгоритми.

Для оцінки алгоритмів існує багато критеріїв. Найбільшу увагу приділяють порядку росту, необхідного для розв'язання задачі часу та розміру пам'яті при збільшені вхідних даних. З кожною конкретною задачею пов'яжемо число, яке назвемо *розміром* задачі і яке б виражало міру кількості вхідних даних. Наприклад, розміром задачі множення матриць може бути найбільший розмір матриць-співмножників. Розміром задачі про графи може бути число ребер даного графа.

Час, витрачений на обчислення при виконанні алгоритму, являє собою суму часів окремих виконаних операторів. Програму, написану на мові високого рівня, можна перетворити прямим (хоча і не простим) шляхом в програму на машинному коді заданої ЕОМ. Це в принципі дає метод оцінки часу виконання вказаної програми, але такий підхід орієнтований на конкретну ЕОМ і не дає загальної залежності часу обчислення від розмірів задачі. В області аналізу та побудови алгоритмів прийнято виражати час виконання, як і будь-яку іншу міру ефективності, з точністю до мультиплікативної константи. Це, зазвичай, робиться шляхом підрахунку лише певних ключових операцій, виконаних алгоритмом (що легко здійснити, аналізуючи версію цього алгоритму, записану на мові високого рівня). Такий підхід абсолютно правомірний при визначені нижніх оцінок часу виконання, оскільки невраховані операції можуть лише збільшити їх; однак при роботі з верхніми оцінками ми повинні впевнитись у тому, що вклад вибраних ключових операцій в сумі відрізняється не більше, ніж в константу раз від вкладу усіх операцій, виконаних алгоритмом.

Час, який витрачається алгоритмом, як функція розміру задачі, називається *часовою складністю* цього алгоритму. Граничну поведінку цієї складності при збільшені розміру задачі називають асимптотичною часовою складністю. Аналогічно, можна виділити об'ємну складність та асимптотичну об'ємну складність. Просторова та часова складності як функції від розмірів задачі є дві фундаментальні оцінки ефективності при аналізі алгоритмів.

Кнут популяризував апарат позначень , які відрізняють верхні та нижні оцінки.

Оцінки складності.

$O(f(N))$ - $\{ g(N) \mid \exists\, C>0 \text{ і } N_0>0: \mid g(N) \mid \leq C f(N), \forall N \geq N_0 \}$ (верхні оцінки)

$\Omega(f(N))$ - $\{ g(N) \mid \exists\, C>0 \text{ і } N_0>0: \mid g(N) \mid \geq C f(N), \forall N \geq N_0 \}$ (нижні оцінки)

$\theta(f(N))$- $\{ g(N) \mid \exists\, C_1, C_2, N_0 > 0: C_2 f(N) \leq g(N) \leq C_2 f(N), \forall N \geq N_0 \}$ (ефективні оцінки).

Таким чином, $O(f(N))$ використовується для вказання верхніх оцінок швидкості росту функцій або для вказання множини усіх функцій, які ростуть не швидше, ніж $f(N)$. Наприклад, коли говорять, що алгоритм виконується за час , $O(f(N))$, то це означає, що час його виконання обмежений зверху значенням , $O(f(N))$ для всіх входів розміру , n. Так як при цьому передбачається, що час виконання в найгіршому випадку також обмежений величиною $O(f(N))$, то це просто час виконання даного алгоритму. $W(f(N))$

використовується для вказання нижніх оцінок швидкості росту функцій або для вказання множини усіх функцій, які ростуть не повільніше, ніж $f(N)$. Аналогічний спосіб, але для нижніх оцінок. Нарешті, $q(f(N))$ використовується для вказання функцій того ж порядку, що і $f(N)$; цей спосіб потрібний для опису "оптимальних" алгоритмів. Якщо алгоритм обробляє входи розміру n за час cn^2, де с – деяка константа, то часова складність цього алгоритму є $O(n^2)$, для усіх n, крім скінченої (можливо порожньої) множини, невід'ємних значень.

Наприклад, запис $O(n^2) \cap \Omega(n)$ позначає клас функцій, які мають швидкість росту не меншу за n, але не більшу за n^2.

Задачі розпізнавання, мови та кодування.

Задля зручності теорію складності будують тільки для задач розпізнавання властивостей, які мають лише два розв'язки: “так” або “ні”. На абстрактному рівні це означає, що задача розпізнавання π складається просто з двох множин множини D_π всіх моєливих індивідуальних задач і множини Y_π ($Y_\pi \subset D_\pi$) індивідуальних задач із відповіддю “так”. Стандартна форма опису таких задач складається з двох частин. У перші частині дають опис *умови* задачі в термінах різних компонент – множин, графів, функцій тощо. У другій частині в термінах умови формулюють *запитання,* на яке може бути лише дві відповіді – “так” або “ні”. Цей опис очевидним способом задає множини Y_π й D_π. Індивідуальна задача належить множині D_π в тому й лише в тому разі, коли її можна отримати зі стандартної форми опису підставленням конкретних значень в умову. Індивідуальна задача належить множині Y_π тоді й лише тоді, коли відповідь на запитання задачі – “так”.

Розглянемо добре відому задачу розпізнавання з теорії графів.

Приклад 1. Ізоморфний підграф.

Умова. Дано два графи $G_1 = (V_1, E_1)$ і $G_2 = (V_2, E_2)$.

Запитання. Чи містить граф G_1 підграф, ізоморфний графу G_2?

Задачу розпізнавання, яка відповідає задачі комівояжера можна сформулювати так.

Приклад 2. Комівояжер.

Умова. Дано скінченну множину $C = \{c_1, c_2, \ldots, c_m\}$ міст, віддалі $d(c_i, c_j) \in N$ для кожної пари міст $c_i, c_j \in C$ та межу $B \in N$ (тут N – множина натуральних чисел).

Запитання. Чи існує шлях довжиною не більше B, який починається в місті $c_{s(1)}$, проходить через кожне місто точно один раз і повертається в місто $c_{s(1)}$? Інакше кажучи, чи існує така перестановка $(c_{(s1)}, c_{(s2)}, \ldots, c_{s(m)})$елементів множини C, що

$$\sum_{i=1}^{m-1} d(c_{s(i)}, c_{s(i+1)}) + d(c_{s(m)}, c_{s(1)}) \leq B?$$

Останній приклад також ілюструє важливий метод – побудову задачі розпізнавання з відповідної оптимізаційної задачі.

Задачі розпізнавання мають дуже природний формальний еквівалент, зручний для вивчення методами теорії обчислень. Цей еквівалент – мова. Нагадаємо, що мова над алфавітом V — це будь-яка підмножина $L \subset V^*$, де V^*– множина всіх скінченних ланцюжків, утворених із символів алфавіту V; ці ланцюжки далі називатимемо *словами.*

Відповідність між задачами розпізнавання та мовами задають за допомогою схем кодування. Схема кодування e задачі π описує кожну індивідуальну задачу з π певним словом у якомусь фіксованому алфавіті V. Отже, задача π та її схема кодування e розбивають слова з

множини V^*на три класи: слова, що не є кодами індивідуальних задач із π; слова, що являють собою коди індивідуальних задач із π з негативною відповіддю на запитання задачі; слова – коди індивідуальних задач із π з позитивною відповіддю на запитання. Третій клас слів – це, власне, та мова, яку ставлять у відповідність задачі π в разі схеми кодування e її позначають як $L[\pi, e]$:

$$L[\pi, e] = \{\xi \in V^* | \ V - \text{алфавіт схеми кодування } e,$$
$$\xi - \text{код індивідуальної задачі } I \in Y_\pi \text{ в разу схеми кодування } e\}$$

Формально говорять, що якийсь результат справджується для задачі π в разі схеми кодування e, якщо він правдивий для мови $L[\pi, e]$. Якщо обмежитися лише "розумними схемами кодування", то будь-яке нове поняття чи властивість, сформульовані в термінах мови, фактично не залежать від схеми кодування. Це дає змогу неформально говорити про те, що певна властивість виконується чи не виконується для задачі π.

Якщо враховувати кодування, то втрачається зв'язок із формальним поняттям "довжина входу". Тому потрібен параметр, за допомогою якого можна виразити часову складність. Зручно вважати, що з кожною задачею розпізнавання пов'язана незалежна від схеми кодування функція Length: $D_\pi \to N$, яка "поліноміально еквівалентна" довжині коду індивідуальної задачі, одержаної за будь-якої "розумної" схеми кодування.

Поліноміальну еквівалентність розуміють так: для будь-якої "розумної" схеми кодування задачі π існують два поліноми p_1 і p_2 такі, що коли $I \in D_\pi$ та слово ξ — код індивідуальної задачі I в разі кодування e, то $Length(I) \le p_1(|\xi|)$і $|\xi| \le p_2$, де $|\xi|$- довжина слова ξ. Наприклад, у задачі про ізоморфний підграф (див. приклад 1) можна записати: Length(I) $=|V_1| + |V_2|$, де $G_1 = (V_1, E_1)$і $G_2 = (V_2, E_2)$— графи, які утворюють розглядувану індивідуальну задачу. У задачі про комівояжера (див. приклад 2) можна записати: Length(I) $= m + [\log_2 B] + max\{[\log_2 d(c_i, c_j)]; c_i, c_j \in C\}$

Оскільки будь-які дві "розумні" схеми кодування задачі дають поліноміально еквівалентні довжини кодів, то діапазон для вибору функції Length дуже великий, а отримувані результати справджуються для будь-якої функції Length, яка задовольняє сформульовані вище умови. Плідність такого неформального, незалежного від кодування підходу пов'язана з тим, що саме розуміють під „розумною" схемою кодування. Схему кодування вважають розумною, якщо вона достатньо стиснена та її можна декодувати. Ця властивість потрібна для того, щоб у разі кодування індивідуальної задачі штучно не збільшувалося вхідне слово. Таке збільшення може призвести, наприклад, до того, що експоненціальний алгоритм неприродним способом перетвориться на поліноміальний. Можливість декодування полягає в тому, що за даною компонентою умови задачі можна навести алгоритм із поліноміальним часом роботи, який би із заданого коду індивідуальної задачі міг відновити опис цієї компоненти.

Детерміновані машини Тюрінга та клас P.

Машина Тюрінга, що була описана А.Тюрінгом у 1936 р., є теоретичною моделлю обчислювальної машини Тюрінга. Її робота може бути описана таким чином. Розглянемо стрічку, розділену на окремі комірки; ця стрічка є потенційно нескінченною в обидва боки. В кожній комірці може бути записаний певний символ з деякого заданого алфавіту А. Машина

Тюрінга в будь-який момент часу може перебувати в певному стані (множина станів S є скінченною) і вказувати на певну комірку.

Машина Тюрінга в залежності від поточного символа, на який вона вказує, може записати на його місце будь-який інший символ (він може співпадати зі старим), зсунутися на один символ вліво або вправо, змінити свій вміст чи зупинитися (часто вважається, що машина зупиняється автоматично, якщо немає жодної інструкції, яку вона могла б виконати). Робота машини Тьюринга визначається її програмою.

Машина Тьюринга є дуже незручною для програмування. Ці незручності пов'язані з тим, що:

- немає довільного доступу до пам'яті; якщо, наприклад, машина вказує на першу комірку, а потрібно перейти до десятої, машина повинна послідовно переглянути другу, третю і т.д. комірку;
- неструктурованість записів на стрічці; заздалегідь невідомо, де закінчується одне число і починається інше;
- дуже обмежений набір команд; відсутні, наприклад, основні арифметичні операції.

Для формалізації поняття алгоритму потрібно зафіксувати певну модель процесу обчислень. Із цією метою використаємо машину Тюрінга (точніше, детерміновану однострічкову машину Тюрінга — скорочено ДМТ). Для цього пункту дещо модифікуємо розглянуте раніше означення машини Тюрінга.

1. В алфавіті зовнішніх символів A явно виділимо підмножину *вхідних символів* V, за допомогою яких записують слово в початковій конфігурації. Отже, $V \subset A$, причому порожній символ $\Lambda \in A \setminus V$

2. В алфавіті внутрішніх станів Q замість одного заключного стану q_0 виділимо два — q_Y і q_N (від слів Yes — так, No — ні), тобто $Q=\{q_Y, q_N, q_1, q_2, \ldots, q_k\}$. Стан q_1 уважатимемо початковим. Заключні стани q_Y і q_N не можуть бути в лівій частині жодної команди.

3. Комірки нескінченної стрічки будемо нумерувати цілими числами .... -2, -1, 0, 1, 2 …

Вхід для детермінованої машини (програми) — слово $\xi \in V^x$, яке записують на стрічці в комірках із номерами 1,2,...,$[\xi]$ по одному вхідному символу в кожній комірці. Як V^x тут позначено множину всіх слів у алфавіті V. Програма розпочинає роботу, перебуваючи в стані q_1 : при цьому головка перебуває над коміркою з номером 1. Далі процес обчислень виконується послідовно, крок за кроком; якщо поточний стан q — q_y або q_N, то процес обчислень закінчується з результатом „так” у разі $q = q_Y$ і „ні” у разі $q = q_N$.

Приклад простої ДМТ-програми M наведено в табл. 1.

Тут A = {0, 1, Λ), V={0,1}, Q={$q_Y, q_N, q_1, q_2, \ldots, q_k$}

Табл. 1

Стан	**0**	**1**	**Λ**
q_1	$0Пq_1$	$1Пq_1$	$ЛЛq_2$
q_2	$ЛЛq_3$	$ЛЛq_4$	$ЛЛq_N$
q_3	$ЛЛq_Y$	$ЛЛq_N$	$ЛЛq_N$
q_4	$ЛЛq_N$	$ЛЛq_N$	$ЛЛq_N$

Легко переконатись, що обчислення за програмою М для входу ξ = 10100 після восьми кроків завершується в стані q_Y,тому відповідь - „так”. Відповідні конфігурації на рис. 1.

Говорять, що програма М із вхідним алфавітом *V приймає* (допускає) слово $\xi \in V^x$тоді й лише тоді, коли в разі її застосування до вхідного слова ξ вона зупиняється в стані q_yМову L_m, розпізнавану програмою М, задають так:

$$L_m = \{\xi \in V^x, Mприймає\xi\}$$

1	0	1	0	0

q_1

1	0	1	0	0

q_1

1	0	1	0	0

q_1

1	0	1	0	0

q_1

1	0	1	0	0

q_1

1	0	1	0	0	Λ

q_1

1	0	1	0	0	Λ

q_2

1	0	1	0	Λ	Λ

q_3

1	0	1	Λ	Λ	Λ

q_Y

Рис.1

Приклад . Програма попереднього прикладу розпізнає мову

$$\{\xi \in \{0,1\}^x, \textit{двоостаннісимволислова}\xi - \textit{нулі}\}$$

Згідно з означенням розпізнавання мови програма M не має зупинятися на всіх вхідних словах із множини V^*; вона має зупинятися тільки на входах із мови L_m. Якщо слово ξ, належить множині $V^* \setminus L_m$, то робота програми M на ньому може закінчитись у стані q_N,або тривати без зупинки. Проте ДМТ-програма, яка відповідає нашому розумінню алгоритму, має зупинитися на всіх словах вхідного алфавіту V. Тому можна вважати, що програма з прикладу 3 алгоритмічна, бо, почавши роботу на будь-якому слові із символів 0, 1, вона зупиниться.

Відповідність між розпізнаванням мов і розв'язуванням задач розпізнавання задають так. Говорять, що ДМТ-програма M *розв'язує* задачу розпізнавання π в разі кодування e, якщо вона зупиняється на всіх словах із символів вхідного алфавіту й $L_m = L[\pi, e]$. Програма з попереднього прикладу ілюструє цю відповідність. Розглянемо таку[7] задачу[7] розпізнавання з теорії чисел.

Приклад. Подільність на 4.

Умова. Дано натуральне число n.

Запитання. Чи існує таке натуральне число m, що n=4m?

У разі стандартного кодування число n можна подати словом із 0 і 1, тобто двійковим записом цього числа. Оскільки натуральне число ділиться на 4 тоді й лише тоді, коли останні дві цифри його двійкового запису — нулі, то програма з прикладу 3 розв'язує цю задачу за такого стандартного кодування.

Тепер можна означити поняття часової складності. Час, потрібний ДМТ-програмі M, щоб виконати обчислення для вхідного слова $\xi \in V^x$, — це кількість кроків до моменту зупинки. Якщо програма M зупиняється для всіх вхідних слів $\xi \in V^x$, то її ***часову складність*** T_m: N—> N означають так:

$$T_M(n) = maxlbracem, \textit{існуєтакеслово}\xi \in V^x, |\xi| = n,$$
$$\textit{що для обчислення за програмою M на вході }\xi\textit{потрібно } m \textit{ кроків}$$

ДМТ-програму називають *поліноміальною*, якщо існує такий поліном *p,* що для всіх $n \in N$виконується нерівність $T_M(n) \leq p(n)$

Нарешті, можна формально означити перший важливий клас мов — P:

$$P = \{L, \textit{існує поліноміальна ДМТ} - \textit{програма M для якої } L = L_m\}$$

Говорять, що задача розпізнавання π належить до класу P в разі схеми кодування e, якщо $L_m = L[\pi, e]$, тобто існує поліноміальна ДМТ-програма, котра розв'язує задачу π в разі кодування за схемою *e.*

Формальним визначення є визначення:

P-задачею називається задача, яка може бути розв'язана на детермінованій машині Тюрінга за поліноміальний час O (nm), де n – розмір задачі. Іншими словами, P -задача – це задача, для розв'язку якої існує поліноміальний алгоритм.

Під детермінованою машиною Тюрінга мається на увазі машина Тюрінга у вищенаведеному розумінні. "Детермінованість" означає, що в будь-який момент дія машини є чітко визначеною.

Недерміновані машини Тьюрінга та клас NP.

Перш ніж перейти до формального означення в термінах мов і машин Тюрінга. пояснимо зміст поняття, на якому ґрунтується означення класу NP. Розглянемо задачу комівояжера: за множиною міст, віддалей між ними та межею Ввиявити, чи існує гамільтонів цикл, довжина якого не перевищує B. Поліноміальний алгоритм для цієї задачі невідомий. Припустимо, що для якоїсь індивідуальної задачі I одержано відповідь „так". Це можна довести, навівши відповідний гамільтонів цикл, і тим самим перевірити відповідне твердження. Більше того, цю процедуру перевірки можна подати у вигляді алгоритму, часова складність якого обмежена поліномом від $Length(\mathrm{I})$. Подібні властивості має також задача про ізоморфний підграф. Нехай в індивідуальній задачі I є два графи $G_1 = (V_1, E_1)$і$G_2 = (V_2, E_2)$. Якщо відповідь на запитання задачі — „так", то де можна довести, указавши підмножину $V' \subset V_1$і$\mathrm{E}' \subset \mathrm{E}_1$, та бієкцію $\varphi: V_2 \to V'$із потрібними властивостями. У цьому разі правильність твердження знову можна легко засвідчити за поліноміальний від $Length(I)$час, просто перевіривши, що V', E'і φзадовольняють вимоги, сформульовані в задачі. Поняття поліноміальної перевірки дає змогу виділити задачі класу NP. Зауважимо, що з можливості перевірки за поліноміальний час не випливає розв'язність за поліноміальний час. Неформально клас NP можна означити за допомогою поняття, яке називають недетермінованим алгоритмом. Такий алгоритм складається з двох різних стадій — угадування та перевірки. Для заданої індивідуальної задачі I на першій стадії просто „вгадують" якусь структуру Σ. Після цього Ita Σразом подають як вхід на стадію перевірки, котра викопується детерміновано та закінчуються відповіддю „так" або „ні" чи продовжується нескінченно без зупинки (відповідь „ні" й останню можливість розрізняти необов'язково). Недетермінований алгоритм „розв'язує" задачу розпізнавання πякщо для довільної індивідуальної задачі $\mathrm{I} \in D_\pi$виконано такі умови.

- 1. Якщо $\mathrm{I} \in Y_\pi$, то існує така структура , Σ угадування якої для входу I зумовлює те, що стадія перевірки, розпочавши роботу з входом , (I, Σ) завершується відповідаю „так".
- 2. Якщо $I \notin Y_\pi$, то не існує такої структури Σ, угадування котрої для входу I забезпечує закінчення стадії перевірки з входом (I, Σ)відповіддю „так".

Наприклад, недетермінований алгоритм розв'язування задачі комівояжера можна побудувати так. На стадії вгадування потрібно просто вибрати довільну перестановку елементів множини міст C, а на стадії перевірки — використати поліноміальну процедуру „перевірка доведення" для задачі комівояжера. Говорять, що недетермінований алгоритм, який розв'язує задачу розпізнавання πпрацює впродовж поліноміального часу, якщо є такий поліном p, що для довільної індивідутільної задачі $\mathrm{I} \in D_\pi$знайдеться здогад , Σякий на стадії детермінованої перевірки на вході (I, Σ)зумовить відповідь „так" за час $\mathrm{p}(Length(\mathrm{I}))$.

Неформально NP — це клас усіх задач розпізнавання π котрі в разі „розумного кодування" можна розв'язати недетершиоваїшми (N — nondeterministic) алгоритмами за поліноміальний (P — polynomianl) час. Зокрема, задачі комівояжера та про ізоморфний пілграф належать до класу NP. У таких неформальних означеннях поняттям розв'язування

слід користуватись обережно. Головне призначення так званого поліноміального недетермінованого алгоритму полягає в поясненні поняття можливості перевірки за поліноміальний час, а не в тому, щоб бути реальним методом розв'язування задач розпізнавання властивостей. Для кожного входу такий алгоритм містить не одне, а декілька можливих обчислень — по одному для кожного можливого здогаду. Формалізуємо означення класу NP в термінах мов і машин Тьюрінга. Формальний еквівалент недетермінованого алгоритму — програма для недетерміноваиої однострічкової машині Тьюрінга (НДМТ). Вона має декілька варіантів поведінки після зчитування букви, а одне вхідне слово зумовлює декілька обчислень. Це можна уявити собі так, що машина робить здогади чи використовує довільну кількість паралельних процесорів. Є різні версії НДМТ. Та, якою ми будемо користуватись, має таку саму структуру, як ДМТ; відмінність полягає лише в тому, що НДМТ доповнено модулем угадування зі своєю головкою, котра може тільки записувати на стрічку.

Модуль угадування дає інформацію для записування здогаду, і його застосовують лише для цього. Програму для НДМТ, або НДМТ-програму, означають абсолютно аналогічно до ДМТ-програми. Обчислення НДМТ-програми для входу $\xi \in V'$, на відміну від обчислення ДМТ-програми, має дві різні стадії На першій стадії відбувається вгадування. У початковий момент вхідне слово ξ, записано в комірках із номерами 1, 2. ..., $|\xi|$; головку для читання-запису розміщено над коміркою з номером 1, головку модуля вгадування — над коміркою з номером -1, а пристрій керування пасивний. Модуль угадування починає керувати своєю головкою, котра в кожен момент робить один крок. Головка чи записує в комірку під нею одну з букв алфавіту Ата зсувається на одну комірку вліво, чи зупиняється (у цьому разі модуль угадування переходить у пасивний стан, а пристрій керування починає роботу в стані q_1). Модуль угадування вирішує, чи продовжити роботу (або перейти в пасивний стан), і яку букву з алфавіту А записати на стрічці, причому робить це ділком довільно. Отже, модуль угадування до моменту закінчення своєї роботи може записати будь-яке слово з множини A , а може ніколи не зупинитись.

Поліномінальна звідність і NP-повні задачі.

Очевидно, що $P \subset NP$: будь-яка задача розпізнавання, розв'язна за поліноміальний час детермінованим алгоритмом, розв'язна за поліноміальний час і недетермінованим алгоритмом (стадію вгадування в такому разі просто ігнорують). Головна проблема теорії складності дискретних алгоритмів полягає в доведенні чи спростуванні гіпотези про те, що $P \neq NP$ Якщо гіпотеза справджується, то для розв'язування найважчих задач класу NPполіноміальних алгоритмів не існує. Нині широко розповсюджена думка, що $P \neq NP$, хоча доведення цієї гіпотези немає. Однак на сучасному рівні знань, мабуть, розумніше працювати за домовленості, що $P \neq NP$. На основі нагромадженого досвіду ми будемо уявляти клас NP так, як це зображено на рис. 2, очікуючи (хоча й не з повною впевненістю), що заштрихована область, яка позначає $NP \setminus P$,не порожня. Якщо $P \neq NP$, то відмінність між класами P та $NP \setminus P$ дуже істотна. Усі задачі з P можна розв'язати поліноміальними алгоритмами, а всі задачі з $NP \setminus P$ важкорозв'язні.

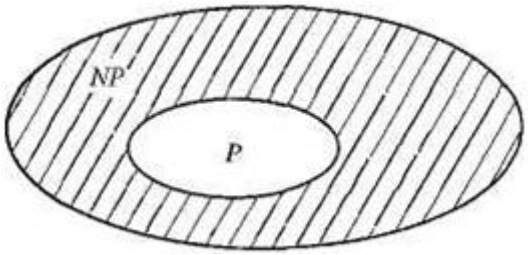

Рис.2

Зрозуміло, що доки не доведено, що $P \neq NP$, неможливо довести, що якась конкретна задача належить класу $NP \setminus P$. Тому мета теорії NP- повних задач полягає в доведенні слабших результатів типу "якщо $P \neq NP$, то $\pi \in NP \setminus P$ ". Такі умовні результати можна вважати підтвердженням важкорозв'язності з тим самим рівнем упевненості, з яким ми вважаємо, що $P \neq NP$. Головна ідея цього умовного підходу ґрунтується на понятті поліноміальної звідності.

Мова $L_1 \subset V_1^*$ поліноміально зводиться до мови $L_2 \subset V_2^*$, якщо існує функція $f: V_1^* \rightarrow V_2^*$, яка задовольняє такі умови.

1. Існує ДМТ-програма, яка обчислює функцію f із часовою складністю, обмеженою поліномом.
2. Для будь-якого слова $\xi \in V'_1$умова $\xi \in L_1$виконується тоді й тільки тоді, коли $f(\xi) \in L_2$.

Завдяки поняттю звідності можна падати точний зміст твердженню про те, що одна мова не простіша, ніж інша. Найскладніші в цьому розумінні NP- повні задачі. Мову L називають NP- повною, якщо виконуються дві властивості.

- $L \in NP$.
- $L' \alpha L$для будь-якого $L' \in NP$.

Клас NP-повних мов позначають NPC(Complete - повний). Говорячи неформально, задачу розпізнавання π називають NP - повною, якщо $\pi \in NP$ та будь-яка інша задача розпізнавання $\pi' \in NP$зводиться до π.

Отже, гіпотеза $P \neq NP$ означає, що NP - повні задачі не можна розв'язати за поліноміальний час, тобто вони важкорозв'язні. Більшість експертів уважають, що це дійсно так; передбачуване співвідношення між класами P, NP та NPC наведено на рис. 3.

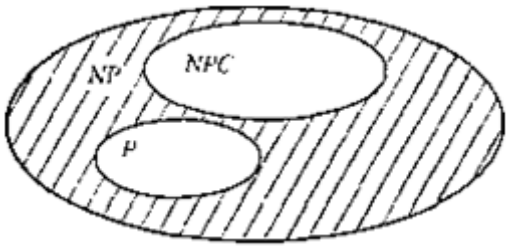

Рис.3

До задач класу складності NP належать:

- Розв'язність логічного виразу.
- Три-кольорове розфарбування графу.
- Побудова кліки з k вершин на неорієнтованому графі.
- Покриття множин: нехай задане сімейство F підмножин E_iдеякої множини E; необхідно знайти підсімейство G із F , так, що: $J_F E_i = U_G E_i$

- Розбиття множин: за попередніх умов, але, крім того, вимагається, щоб $E_i \cap E_j = \emptyset$ (для довільних E_i, E_j з G).
- Існування гамільтонового циклу на неорієнтованому графі.
- Задача паркування рюкзака: для змінних x_i, що приймають значення 0 та 1 розв'язати рівняння:
- $\sum a_j x_j = b$де a_j та b — наперед задані цілі числа.
- для загального випадку, ця задача є розв'язанням діофантового рівняння.
- Розбиття на дві частини: нехай дано n чисел з N, необхідно розбити на дві множини I_1та I_2що не перетинаються, так, щоб:
- $\sum_{i \in I_1} y_i = \sum_{i \in I_2} y_i$.

Теорема Кука.

Теорема Кука стверджує, що задача здійсненності булевих формул у КНФ (коротше,SAT) є NP - повною.

Доведення цієї теореми, отримане Стівеном Куком у його фундаментальній роботі в 1971 році, стало одним з перших найважливіших результатів теорії NP-повних задач. Незалежно в той же час ця теорема була доведена радянським математиком Леонідом Левіним.

У доведенні теореми Кука кожна задача з класу NP у явному вигляді зводиться до SAT. С. Кук визначив клас NP задач розпізнавання властивостей наступним чином. Задача належить класу NP, якщо :

- Розв'язком задачі є одна з двох відповідей: «так» чи «ні» (задача розпізнавання властивостей) ;
- Потрібна відповідь може бути отримана на недетермінірованому обчислювальному пристрої за поліноміальний час.

Цей клас, як пізніше показав Річард Карп, у свою чергу містить у собі інший широкий клас задач, названий Карпом NP-повні задачі, або NPC, — задачі, які зводяться в межах цього класу одна до одної.

Після появи результатів Кука була доведена NP-повнота для безлічі інших завдань. При цьому найчастіше для доказу NP-повноти деякої задачі наводиться поліноміальне зведення задачі SAT до даної задачі, можливо в кілька кроків, тобто з використанням декількох проміжних задач.