1. Вступ у теорію алгоритмів. Базові поняття алгоритмів та їхні складності. Оцінювання алгоритмів. Необхідність формалізації поняття алгоритму.

Алгоритм - це формально-описана обчислювальна процедура, яка перетворює вхідні дані(аргумент) і видає результат обчислень на виході. Оцінюють алгоритми за різними критеріями. Розмір задачі - це обсяг вхідних даних, які потрібні для опису задачі. Наприклад, розміром задачі про сортування може бути кількість елементів масиву, який треба посортувати. Часова складність - це час витрачений алгоритмом. Асимптотична часова складність - це є поведінка часової складності зі збільшенням розміру задачі. Ємнісна складність - це обсяг пам'яті, який потрібен алгоритму для роботи, а також, аналогічно визначається асимптотична ємнісна складність. Асимптотична складність алгоритму показує розмір задач, які цей алгоритм може розв'язувати.

Задачу, для якої знайдено алгоритм розв'язування, називають ***розв'язною*** (такою, яку можна розв'язати). Щоб довести розв'язність задачі, достатньо побудувати відповідний алгоритм, а тому достатньо інтуїтивного поняття алгоритму. Для того ж, щоб довести, що задача є ***нерозв'язною*** (не існує алгоритму для її розв'язування), такого формулювання не достатньо. Для цього необхідно точно знати, що таке алгоритм, визначити це поняття чітко математично.

2. Абстрактні алфавіти й алфавітні оператори. Кодувальні алфавітні оператори. Способи задання алфавітних операторів. Поняття алгоритму. Основні властивості алгоритмів. Способи запису алгоритмів. Різновиди алгоритмів. Композиція алгоритмів.

Абстрактний алфавіт - це довільна скінченна сукупність елементів. **Букви алфавіту** - можуть бути довільні, але повинні бути різні і кількість скінченною. **Слово** в певному алфавіті - це довільна скінченна впорядкована послідовність із букв даного алфавіту. **Довжина слова** - це кількість букв у слові.

Алфавітний оператор φ - це відповідність між словами в одному або різних алфавітах. **Приклад**: нехай $\{P\}_X$ - це деяка множина вхідних слів в алфавіті X, $\{Q\}_Y$ - множина вихідних слів в алфавіті Y. Алфавітним оператором φ буде функція такого вигляду $\varphi: \{P\}_X \to \{Q\}_Y$. **Однозначний оператор** - це оператор, який кожному вхідному слову у відповідність ставить не більше одного вихідного слова. **Багатозначний оператор** - ставить у відповідність декілька різних вихідних слів:

$$P \to \begin{cases} Q_1 \\ \vdots \\ Q_n \end{cases}$$

Нехай A - деякий алфавіт, який називають стандартним, B - довільний алфавіт. Нехай $\{L\}_A, \{M\}_B$ - множини слів у відповідних алфавітах. Тоді, множина $\{M\}_B$ закодована в алфавіті A, якщо задано такий однозначний оператор: $\varphi: \{M\}_B \to \{L\}_A$. Оператор φ називають **кодувальним**, а слова з $\{L\}_A$ - кодами об'єктів з $\{M\}_B$.

Кодування об'єктів алфавіту B словами однакової довжини називають **нормальним кодуванням**.

Отже, ***алгоритм*** – це алфавітний оператор разом із системою правил, яка визначає його дію:

$$A = < \varphi, \Pi \mid \Pi - \text{система правил} >$$

*Два алфавітні оператори називають **рівними**, якщо вони мають одну і ту ж область визначення й однаковим вхідним словам з цієї області ставлять у відповідність однакові вихідні слова.*

Два алгоритми $A_1 = <\varphi_1, \Pi_1>$, $A_2 = <\varphi_2, \Pi_2>$ *називаються **рівними**, якщо відповідні їм алфавітні оператори рівні та системи правил збігаються:*

$$A_1 = A_2 \Leftrightarrow \varphi_1 = \varphi_2,\ \Pi_1 = \Pi_2$$

Два алгоритми A_1, A_2 *називають **еквівалентними**, якщо відповідні їм оператори рівні, а системи правил різні:*

$$A_1 \cong A_2 \Leftrightarrow \varphi_1 = \varphi_2,\ \Pi_1 \neq \Pi_2$$

Еквівалентні алгоритми задають розв'язок однієї й тієї ж задачі, проте різними способами.

Розрізняють два способи задання операторів.

1. Табличний спосіб задання операторів. Такий спосіб застосовують тоді, коли область визначення оператора скінченна. Оператор φ задають таблицею відповідності, у якій для кожного вхідного слова P з області визначення φ виписано відповідне вихідне слово Q:

$$P_1 \to Q_1,$$
$$\dots$$
$$P_n \to Q_n.$$

2. Задання оператора скінченною системою правил, яка дає змогу за скінченну кількість кроків знайти значення φ на довільному вхідному слові, наприклад, система правил для додавання натуральних чисел у системі числення з основою p або система правил знаходження найбільшого спільного дільника двох додатних чисел та ін.

Властивості:

1) Дискретність алгоритму. Алгоритм описує послідовний процес, який відбувається в дискретному часі. В кожен інтервал часу у системі величин за певними правилами виконують елементарні кроки алгоритму.
2) Ефективність алгоритму. Елементарні кроки алгоритму повинні бути ефективними, тобто виконуватися точно і за короткий відрізок часу.
3) Скінченність. Алгоритм завжди повинен закінчуватися після скінченної кількості кроків.
4) Результативність. Має забезпечувати отримання результату розв'язування задачі, що є наслідком скінченної кількості елементарних кроків та їхньої ефективності.
5) Масовість. Алгоритм можна застосувати до цілого класу задач, а не тільки до однієї.

Способи запису:

·**словесний** (запис на природній мові);

·**графічний** (зображення з графічних символів);

·**псевдокоди** (напівформалізовані описи алгоритмів на умовній алгоритмічній мові, що включають як елементи мови програмування, так і фрази природної мови, загальноприйняті математичні позначення та ін.);

·**програмний** (тексти на мовах програмування).

Ознаки поділу алгоритмів за властивостями Детермінованість (в визначені алгоритму алфавітний оператор однозначний, тоді алгоритм детермінований,

інакше недетермінований) Самозмінність (самозмінний, якщо в процесі переробки алгоритмом вхідних слів система P змінюється, в іншому випадку він не самозмінний) Самозастосовність (самозастосовний, якщо слово входить в область визначення алгоритму, інакше не самозастосовний) Універсальність(універсальний, якщо він еквівалентний довільному наперед заданому алгоритму)

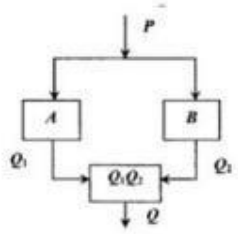
Композиція алгоритмів

1. Суперпозиція алгоритмів A і B При суперпозиції двох алгоритмів A і B вихідне слово одного з них розглядають як вхідне слово іншого C(P) = B(A(P)), P є D(A), A(P) с D(B) Суперпозиція може виконуватись для довільної скінченної к-сті алгоритмів

2.

Об'єднання алгоритмів

$$P \in D(A) \cap D(B),\ C(P) = A(P)B(P)$$



На всіх інших словах алгоритм С вважається невизначеним

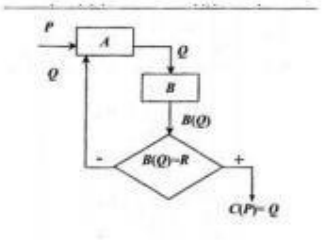
3. Розгалуження алгоритмів

Розгалуженням називають композицію трьох алгоритмів A,B,C, задану співвідношенням

$$F(P)\begin{cases} A(P), \text{якщо } C(P) = R \\ B(P), \text{якщо } C(P) \neq R \end{cases}$$

4. Ітерація алгоритмів

Ітерацією двох алгоритмів A та B наз. Алгоритм C, який визначають так: для довільного вхідного слова P вихідне слово C(P) отримують як результат послідовного багаторазового застосування алгоритму A, доки не буде отримано слово, перетворене алгоритмом B в деяке фіксоване слово R



3. Поняття про алгоритмічні системи. Система нормальних алгоритмів Маркова. Принцип нормалізації. Універсальний нормальний алгоритм.

Алгоритмічною системою називають спосіб задання алгоритмів у деякому фіксованому формалізмі F, який дозволяє для довільного алгоритму A задати формальний еквівалент Af . Алгоритмічні системи є формалізацією поняття алгоритму. Алгоритмічні системи діляться на типи відповідно до визначення алгоритму: Приклади

Система нормальних алгоритмів Маркова

Алгоритми Маркова — це формальна математична система. В алгоритмічній системі Маркова існує лише оператор підстановки та розпізнавач входження.

Загальна схема: застосувавши декілька разів оператор підстановки до вхідного рядка R, перетворити його у вихідний рядок Q.

Простою продукцією називають запис вигляду $u \to w$. u - ангицедент, w - консеквент.

Уважають, що формула $u \to w$ може бути застосована до рядка $Z \in V$, якщо є хоча б одне входження и в Z.

Заключною продукцією називають запис вигляду $u \to w$, де u,w - рядки в V.

Нормальним алгоритмом, чи алгоритмом Маркова, називають упорядковану множину продукцій $P_1, P_2, \ldots, P_n$.

Кожна з продукцій містить розпізнавання входження підрядка u в рядок Z та підстановку w замість u у разі успішного розпізнавання. Послідовність виконання продукцій залежить від того, чи може бути застосована до рядка чергова формула підстановки.

Алгоритм Маркова завершується в одному з двох випадків:

- до рядка не може бути застосована жодна з наявних формул підстановок;
- до рядка застосована заключна підстановка.

Тоді алгоритм вважають **застосовним** до заданого вхідного слова.

Нормальні алгоритми іноді зображають за допомогою граф-схеми. Алгоритми, які задають граф-схемами, складеними винятково з розпізнавачів входження і операторів підстановки, називають **нормальними**, якщо їхні граф-схеми задовольняють такі умови:

1. Кожний розпізнавальний вузол Q і відповідний йому операторний вузол Q→ R об 'єднані в один узагальнений вузол, який називають **операторно-розпізнавальним**; усі узагальнені вузли схеми впорядковані за допомогою нумерації від 1 до n;
2. негативний вихід i-го вузла приєднаний до (i +1)-го вузла (i =$\overline{1, n-1}$), а негативний вихід n -го вузла — до вихідного вузла граф-схеми;
3. позитивні виходи всіх узагальнених вузлів приєднані або до першого, або до вихідного вузла граф-схеми;
4. вхідний вузол приєднаний до першого узагальненого вузла.

Нормальні алгоритми прийнято задавати не граф-схемами, а впорядкованими наборами підстановок, у яких кожна підстановка відповідає операторно-розпізнавальному вузлу. Q →R - звичайна підстановка, а заключна - Q→ .R

Упорядкований набір підстановок визначеного типу називають **схемою** заданого алгоритму.

Виконання алгоритму починається з першої формули. Послідовність виконання алгоритму залежить від того, чи може бути застосована до рядка чергова формула підстановки. Процес виконання підстановок закінчується лише тоді, коли жодна з підстановок схеми не може бути застосована до отриманого слова, або коли виконана деяка заключна підстановка.

си стем: машина Тюрінга, нормальні алгоритми Маркова, рекурсивні функці.

Принцип нормалізації.

Будь-яка алгоритмічна система має задовольняти 2 вимоги: бути математично строгою та універсальною. За допомогою певного математичного апарату досягають математичної строгості. Універсальність теорії нормальних алгоритмів формулюють у вигляді принципу нормалізації.

Теорема. Для того, щоб реалізувати в схемах нормальних алгоритмів довільний алгоритм $A = \{\varphi, P\}$, необхідно, щоб у системі нормальних алгоритмів були як звичайні, так і заключні підстановки.

Принцип нормалізації. Для будь-якого алгоритму $A = \{\varphi, P\}$ в довільному алфавіті X можна побудувати еквівалентний йому нормальний алгоритм над алфавітом X.

Перехід від інших способів опису алгоритмів до еквівалентних нормальних алгоритмів називають **зображенням у нормальній формі, або нормалізацією**.

Алгоритм A = (φ,P) в алфавіті X називають **нормалізованим**, якщо можна побудувати еквівалентний йому нормальний алгоритм над алфавітом X. В іншому випадку алгоритм називають **ненормалізованим**.

Усі алгоритми є нормалізовані. Принцип нормалізації не може бути математично доведений або заперечений.

Головні факти, що підтверджують принцип нормалізації:

1. Правильність алгоритму ґрунтується на тому, що всі відомі алгоритми - нормалізовані.
2. Якщо вихідні алгоритми вже нормалізовані, то нормалізованою буде і композиція цих алгоритмів.
3. Еквівалентність системи нормальних алгоритмів усім іншим алгоритмічним системам.
4. Усі спеціальні спроби побудови алгоритмів найбільш загального вигляду не вивели за межі класу нормалізованих алгоритмів.

Систему нормальних алгоритмів прийнято вважати практично універсальною алгоритмічною системою.

В іншому випадку алгоритм не нормалізований.
Універсальним нормальним алгоритмом (УНА) називають алгоритм,здатний виконувати роботу нормального алгоритму *A*.
УНА виконує над вхідним словом *P* підстановки згідно з схемою інформації конкретного алгоритму *A та цього слова P*.
УНА, будучи нормальним алгоритмом, теж має один стандартний фіксований алфавіт.Щоб він міг сприймати схему, вона повинна бути закодована в стандартному алфавіті УНА.
А.Марков довів наступну теорему про універсальний нормальний алгоритм:
Теорема. Існує такий нормальний алгоритм U, який нази-вають універсальним, що для будь-якого нормального алгоритму A і будь-якого вхідного слова P з області визначення *A* перетворює наступне зха допомогою конкатенації зображень *A* і *P*

$$A^{cod} P^{cod} \xrightarrow{U} A(P)^{cod}$$

де Q^{cod}, *де* $Q = A(P)$

4. Рекурсивні функції. Обчислювані функції. Найпростіші функції. Основні оператори. Оператор суперпозиції. Функції елементарні відносно множини функцій. Оператор примітивної рекурсії. Оператор мінімізації. Примітивно-рекурсивні функції. Частковорекурсивні функції. Теза Черча.

Рекурсивні функції - це алгоритмічні системи, що ґрунтуються на використанні конструктивно визначених арифметичних (цілочислових) функцій.

Числові функції, значення яких можна обчислити за допомогою деякого (єдиного для заданої функції) алгоритму, називають **обчислюваними функціями**.

Найпростіші функції - наступності, тотожно рівно 0, тотожню(вибору аргументу).

Числову функцію φ : N -> N називають **функцією наступності**, якщо φ(x) = x + 1.

Операції над функціями називаються операторами. Під час побудови частково-рекурсивних функцій використовують три оператори:

- Суперпозиції. Нехай:

  $$g^m(x_1,\dots,x_m) = f^n(f_1^m(x_1,\dots,x_m),\dots,f_n^m(x_1,\dots,x_m))$$

  f^n, f_n^m, g^m – це деякі функції. Тоді оператор, який дасть нам g^m називають операторм суперпозиції і позначають $\mathbb{S}^{n+1}(f^n, f_1^m, \dots, f_n^m)$, де n+1 – кількість функцій.

- Примітивної рекурсії. Розглянемо функцію $f^{n+1}: N^{(n+1)} \to N$, яка визначена наступним чином:

  $$\begin{cases} f(x_1,\dots,x_n,0) = g(x_1,\dots,x_n), \\ f(x_1,\dots,x_n,y+1) = h(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y)) \end{cases}$$

  Отже, оператор примітивної рекурсії($\mathbb{R}$) дасть можливість з g, h побудувати функцію f.

- Якщо з функції $f(x_1,\dots,x_n)$ можна утворити функцію $\mu_y(f(x_1,\dots,x_{n-1},y) = x_n)$, то оператор, який дає таку можливість називають оператором мінімізації і позначають $\mathbb{M}$:

  $$\mu_y(f(x_1,\dots,x_{n-1},y) = x_n) = \mathbb{M}f$$

Числову функцію $\varphi : N^{(n)} \longrightarrow N$ називають **нуль-функцією**, якщо $\varphi(x_1,...,x_n) = 0$.

Числову функцію $\varphi_i : N^{(n)} \longrightarrow N$ називають **функцією вибору аргументів**, якщо вона повторює значення свого і-го аргумента:

$\varphi(x_1,...,x_n) = x_i, (1 < i < n)$.

Існують також примітивно- рекурсивні, загально рекурсивні та частково-рекуурсивні ф-ї.

*Функцію f називають **примітивно-рекурсивною**, якщо її можна отримати із застосуванням скінченної кількості операторів суперпозиції і примітивної рекурсії на підставі лише найпростіших функцій S^1, O^1, I_m^n:*

$$\{S^1, O^1, I_m^n\} \xrightarrow{S^{n+1},\ R} f .$$

---

*Часткову функцію f називають **частково-рекурсивною відносно** σ, якщо її можна отримати з функцій системи σ і найпростіших функцій із застосуванням скінченної кількості операторів суперпозиції, примітивної рекурсії та мінімізації:*

$$\{S^1, O^1, I_m^n, \sigma\} \xrightarrow{S^{n+1},\ R,\ M} f .$$

*Часткову функцію f називають **частково-рекурсивною**, якщо її можна отримати з найпростіших функцій із застосуванням скінченної кількості операторів суперпозиції, примітивної рекурсії та мінімізації:*

$$\{S^1, O^1, I_m^n\} \xrightarrow{S^{n+1},\ R,\ M} f .$$

---

*Функції, які можна отримати з найпростіших функцій S^1, O^1, I_m^n за допомогою оператора примітивної рекурсії, суперпозиції та слабкої мінімізації, називають **загальнорекурсивними**.*

Теза Черча. *Клас алгоритмічно (або машинно) обчислюваних часткових числових функцій співпадає з класом всіх частково-рекурсивних функцій: $K_A \cong K_{ч.р.}$*

Як вже згадувалось, ця теза, як і теза Маркова, в принципі не може бути доведена, оскільки в її формулюванні використовується інтуїтивне поняття алгоритму.

Разом з тим в теорії алгоритмів строго математично доведено таку **теорему**:

Алгоритм тоді і тільки тоді може бути нормалізований, коли він може бути реалізований за допомогою частково-рекурсивних функцій:

$$K_A \cong K_{ч.р.}, \qquad K_H = K_{ч.р.} .$$

Тут K_H - клас нормальних алгоритмів.

5. Алгоритмічна система Тьюрінга. Машина Тьюрінга. Формальне визначення МТ. Теза Тьюрінга. Універсальна машина Тьюрінга. Зв”язок частково-рекурсивних функцій та машини Тьюрінга. Різновиди машин Тьюрінга. Проблема розпізнавання самозастосованості алгоритмів.

В працях Е. Поста і А. Тьюрінга сказано, що **алгоритмічні процеси** — це процеси, які може виконувати побудована для цього “машина”. **Машина Тьюрінга** - це математична модель пристрою, який породжує обчислювальні процеси. Цю машину використовують для теоретичного уточнення поняття алгоритму та його дослідження.

Машина Тьюрінга (МТ) - це умовна машина, яка складається з трьох головних компонент: інформаційної стрічки, головки для зчитування і запису та пристрою керування. **Інформаційна стрічка** призначена для записування інформації (вхідна/вихідна/проміжна), яка виникає внаслідок обчислень. **Зовнішній алфавіт машини** - це алфавіт $S = \{s_1, s_2, \dots, s_k\}$, він є фіксований для кожної машини. **Головка зчитування і запису** - це елемент, який читає і змінює комірки стрічки, ходячи по ній вліво і вправо. **Пристрій керування** керує всією роботою машини відповідно до програми обчислень, яка є задана. **Внутрішній алфавіт машини** - це алфавіт $Q = \{q_1, q_2, \dots, q_n\}$ станів, він є фіксований для кожної машини. Пристрій керування перебуває в одному із цих станів в кожен момент часу.

Функціонує МТ дискретними кроками, кожен з яких має три операції:

1) символ s_i, на який вказує головка, замінюється іншим символом з алфавіту S;
2) головка або зсувається вправо/вліво на одну позицію, або залишається на місці
3) пристрій керування змінює свій стан q_i на інший стан з алфавіту Q.

МТ – певний алгоритм для переробки вхідних слів. Кожна МТ реалізує ф-ю

$\sigma: S \times \{Q \backslash q_F\} \to S \times Q \times Z$

МТ обчислює деяку словникову ф-ію $\varphi(\mathrm{P})$, яка відображає слово з деякої множини слів у не порожні слова, якщо для довільного слова P процес МТ(P) досягає заключної конфігурації асоційованої зі словом $\varphi(\mathrm{P})$ – результатом ф-ії

Якщо для часткової словникової ф-ії існує МТ, яка її обчислює, то ф-ію називають обчислювальною за Тюрінгом.

Теза Тюрінга

Для довільного алгоритму A = <φ, P> у довільному скінченному алфавіті X існує ф-ія $\varphi(\mathrm{P}): \{\mathrm{P}\}_{\mathrm{x}} \to \{Q\}_{\mathrm{x}}$ обчислювана за Тюрінгом.

Будь-який алгоритм заданий у довільній формі можна замінити еквівалентною йому МТ

Теорема 1

Всі часткові словникові ф-ії обчислювані за Тюрінгом є частково-рекурсивні

Теорема 2

Для кожної частково-рекурсивної словникової ф-ії визначеної на алфавіті $\mathrm{A} = \{\mathrm{a}_1, \dots \mathrm{a}_n\}$ існує МТ з відповідним внутрішнім алфавітом, яка обчислює задану ф-ію.

Універсальна машина Тьюрінга: Універсальна машина Тьюрінга - це спеціальна Машина Тьюрінга, яка може симулювати будь-яку іншу Машину Тьюрінга з допомогою вхідних даних, що представляють коди програм. Вона може виконувати ті ж самі обчислення, що і будь-яка інша Машина Тьюрінга, і вважається універсальною, оскільки може моделювати будь-який обчислювальний процес.

Зв'язок частково-рекурсивних функцій та машини Тьюрінга: Теорія частково-рекурсивних функцій та Машина Тьюрінга мають тісний зв'язок. Частково-рекурсивні функції визначаються за допомогою простих базових функцій та операторів рекурсії та примітивної рекурсії. Машина Тьюрінга, з іншого боку, може ефективно обчислювати ці частково-рекурсивні функції, що робить їх обчислювально доступними.

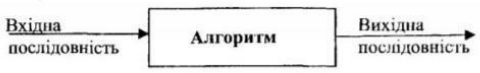
Різновиди машин Тьюрінга: Існує кілька різновидів Машин Тьюрінга, які розширюють базову модель, додавши до неї певні функціональність або обмеження. Деякі з різновидів включають:

- Недетермінована Машина Тьюрінга, де перехід з одного стану в інший може бути неоднозначним.
- Багентна Машина Тьюрінга, в якій кілька Машин Тьюрінга працюють паралельно та взаємодіють одна з одною.
- Квантова Машина Тьюрінга, яка використовує принципи квантової механіки для здійснення обчислень.

Проблема розпізнавання самозастосованості алгоритмів: Проблема розпізнавання самозастосованості алгоритмів відноситься до області теорії обчислювальної складності. Ця проблема полягає у визначенні, чи є певний алгоритм самозастосовним, тобто, чи може він виявити, чи є його власний вихідний код допустимим розв'язком.

6. Важкорозв’язні задачі. Поліноміальні алгоритми та важкорозв’язні задачі. Недетерміновані машини Тьюрінга. Класи P та NP. Мови і задачі. Приклади NP-повних задач. Застосування теорії NP-повноти для аналізу задач.

Алгоритм можна розглядати, як “чорний ящик”, який за вхідною послідовністю будує вихідну послідовність:



Зокрема, можна вважати, що вхідна і вихідна послідовності складаються з 0 і 1, які кодують вхід і вихід алгоритму. Тоді алгоритм розглядають як послідовність елементарних двійкових операцій, таких як *або, і, не, друк* і тощо, що працюють з пам’яттю двійкових символів, яка може бути досить великою.

Алгоритм з поліноміальним часом - це алгоритм, час роботи якого обмежений зверху деяким поліномом Рк (n) (к - степінь полінома). Точні оцінки залежать від реалізації алгоритму.

Якщо оцінка алгоритму зростає швидше, ніж поліном, то такі алгоритми називають **експоненціальними**.

Задачі вважають **легкорозв’язними,** якщо вони мають поліноміальні алгоритми розв’язування. Задачу називають **важкорозв’язною**, якщо не існує поліноміальних алгоритмів для її розв’язування.

1) *Недетерміновані машини Тьюрінга*

Недетермінованість можна пояснити як алгоритм, який виконує обчислення до певного місця, у якому повинен бути зроблений вибір з кількох альтернатив.

Наприклад, *X* на стрічці і стан 3 допускає як стан 4 із записом на стрічку символу *Y* та зміщенням головки вправо, так і стан 5 з записом на стрічку символу *Z* і зміщенням головки вліво.

НМТ має скінченну кількість можливих кроків, з яких у черговий момент вибирається якийсь один.

Можна уявити, що в разі неоднозначності переходу (поточна комбінація стану і символу на стрічці допускає кілька переходів) НМТ ділиться на кілька НМТ, кожна з яких діє за одним з можливих переходів.

Тобто на відміну від ДМТ, яка має єдиний «шлях обчислень», НМТ має «дерево обчислень».

Визначимо клас ***P*** як множину всіх мов, які допускають ДМТ(детерміновану машину тюрінга) з поліноміальною часовою складністю, тобто

P={L| існують такі ДМТ *M* і поліном P(n), що часова складність машини *M* дорівнює P(n) і L(*M*)=*L*}.

Клас **NP** - це множина всіх мов, які допускають НМТ(недетерміновану машину тюрінга) з поліноміальною часовою складністю.

Клас *P* можна уявити як клас задач, які можна швидко розв'язати, а *NP* - як клас задач, розв'язок яких можна швидко перевірити.

Мову L_0 з *NP* називають *NP*-повною, якщо за заданим детермінованим алгоритмом розпізнавання L_0 з часовою складністю T(n) >= n і довільною мовою *L* з *NP*, можна ефективно знайти детермінований алгоритм, який розпізнає *L* за час $T(P_L(n))$, де P_L — деякий поліном, що залежить від *L*. Говорять, що *L* поліноміально зводиться до L_0.

Мову *L* називають **поліноміально трансформовною в L_0**, якщо деяка ДМТ *M* з поліноміально обмеженим часом роботи перетворює кожен ланцюжок w в алфавіті мови L в такий ланцюжок w_0 в алфавіті мови L_0, що $w \epsilon$ *L* тоді і лише тоді, коли w_0<=L_0.

1. Застосування теорії *NP*-повноти для аналізу задач.

Однією з найскладніших проблем теорії обчислень є так звана проблема “P = NP”, тобто чи тотожні класи P та NP

Найсерйознішою причиною вважати, що ці класи не тотожні це існування NP - повних задач. У класі NP є NP-повні (універсальні) задачі, тобто такі, що до них поліноміально зводиться довільна задача з класу NP. їх використовують як еталони складності.

Задача є NP-трудною, якщо будь-яка задача з класу до неї зводиться, а NP-повною є задача, яка є NP-трудною, але належить до класу NP. Важливою властивістю NP-повних задач є те, що всі вони “еквівалентні” в такому сенсі: якщо хоча б для однієї з них буде доведено, що вона є легкорозв’язною, то такими будуть і решта задач з цього класу. Якщо в класі NP існує задача, нерозв’язна за поліноміальний час, то всі NP-повні задачі такі ж - то N = NP.

Отже, гіпотеза N = NP означає, що NP -повні задачі не можуть бути розв’язані за поліноміальний час. Доведення NP -повноти деякої задачі є суттєвим аргументом на користь її практичної нерозв’язності. Для такої задачі доцільніше будувати достатньо точні наближені алгоритми, ніж затрачати час на пошук швидких алгоритмів, що розв’язують її точно.

“Трудність” задач можна порівнювати, зводячи одну задачу до іншої. Метод зведення є головним у доведенні NP-повноти багатьох задач.

Сьогодні визначено NP -повноту багатьох задач, еквівалентних між собою щодо поліноміальної звідності. NP-повні задачі й задачі P сильно відрізняються за трудомісткістю розв’язування, проте в строгому сенсі ця різниця і, отже, різниця між класами P та NP не доведена.

Знайти точний розв’язок задачі з класу NP важко, оскільки кількість можливих комбінацій вхідних значень, що потребують перевірки, надзвичайно велика. Для кожного набору вхідних значень I можна створити множину можливих розв’язків PSi. Тоді оптимальним уважають такий розв’язок $S_{optimal}$ є PSi, що:

$$\text{Value}(S_{optimal}) < \text{Value}(S') \text{ для всіх } S' \in PS_I ,$$

якщо ми маємо справу з задачою мінімізації, або

$$\text{Value}(S_{optimal}) > \text{Value}(S') \text{ для всіх } S' \in PS_I ,$$

якщо розв’язуємо задачу максимізації.

Приклади NP-повних задач

1.
Розкладання на множники (Integer Factorization Problem): Полягає у знаходженні простих множників цілого числа. Наприклад, для числа 15 розкладання на множники буде 3 * 5. Ця задача має велике значення у криптографії, особливо в системах шифрування з використанням RSA. Задача орієнтованого покриття вершин (Vertex Cover Problem): Полягає у знаходженні найменшого можливого підмножини вершин в орієнтованому графі, таких що кожне ребро має хоча б один кінець у цьому підмножині. Ця задача має застосування в оптимізації мереж, плануванні розкладу ресурсів.
2.

7. Методи розробки ефективних алгоритмів. Метод “Поділяй і володарюй”. Евристичні алгоритми. Метод гілок і меж. Динамічне програмування. Жадібні алгоритми.

2. Методи розробки ефективних алгоритмів.

Я наведу приклади декількох основних методів:

- ***Метод часткових цілей**. Зведення важкої або великої задачі до декількох простіших задач. У цьому разі розв'язок початкової задачі отримують з розв'язків легших задач(не існує загального набору правил для визначення класу задач, які можна розв'язати за допомогою такого підходу).*
- ***Метод підйому**. Починається з прийняття початкового припущення або обчислення початкового розв'язку задачі. Потім відбувається якнайшвидший рух "вверх" від початкового розв'язку в напрямі до ліпших розв'язків. Коли алгоритм досягає такої точки, з якої більше неможливо рухатися вверх, алгоритм зупиняється. У цьому разі, немає жодних гарантій, що кінцевий розв'язок буде оптимальним - ми отримаємо лише наближений розв'язок.*
- ***Метод відпрацювання назад**. Його починають з цілі чи розв'язку і рухаються назад за напрямом до початкового формулювання задачі. Далі, якщо ці дії оборотні, рухаються знову від формулювання задачі до розв'язку. Цей метод широко застосовують під час розв'язування різноманітних головоломок.*
- ***Рекурсія**. Процедура, яка прямо чи опосередковано звертається до себе. Застосування рскурсії часто дає змогу побудувати зрозуміліші та стислі алгоритми, ніж це було б зроблено без неї. Рекурсія сама по собі не приводить до ефективнішого алгоритму. Однак у поєднанні з іншими методами, наприклад "поділяй і володарюй", дає алгоритми, одночасно ефективні й елегантні.*

Це фундаментальні методи, з використанням яких побудовані класичні алгоритми.

Поділяй і володарюй - важлива парадигма розробки алгоритмів, що полягає в рекурсивному розбитті розв'язуваної задачі на дві або більше підзадач того ж типу, але меншого розміру, і комбінуванні їх рішень для отримання відповіді до вихідного завдання; розбиття виконуються до тих пір, поки всі підзадачі не опиняться елементарними.

Прикладом алгоритму такого виду є сортування злиттям. Цей алгоритм ділить масив навпіл і викликає себе для кожної з половин і так доки розмір половин не стане 1, опісля посортовані половини зливаються і отримуємо результат.

Інші приклади важливих алгоритмів, в яких застосовується парадигма «розділяй і володарюй»: двійковий пошук, метод бісекції, швидке сортування та інше.

Евристичний алгоритм - це алгоритм, спроможний видати прийнятне рішення проблеми серед багатьох рішень, але неспроможний гарантувати, що це рішення буде найкращим.

Наприклад, якщо для розв'язку задачі всі відомі точні алгоритми потребують декількох років машинного часу, то можна прийняти довільний наближений розв'язок, який може бути отримано за розумний час.

Приклад (задача про комівояжера). Нехай задано N міст та матрицю вартостей M, елементи якої m_{ij} дорівнюють вартості подорожі з міста i в місто j. Починаючи з деякого міста *r,* необхідно відвідати всі міста (лише один раз) і повернутися в початкове місто. Отриманий таким способом шлях називають туром. Задача полягає в знаходженні туру мінімальної вартості.

Якщо ми захочемо розв'язати цю задачу для повного графа з 25 вершинами, то потрібно розглянути 24!/2 різних гамільтонових циклів. Якщо припустити, що для розгляду одного такого циклу потрібно одну наносекунду, то загалом буде потрібно приблизно десять мільйонів років для знаходження гамільтонового циклу найменшої довжини. Однак для цієї задачі існує наближений алгоритм (наприклад GTS), який знаходить відповідь з невеликою похибкою від точного результату.

Метод гілок і меж — один з комбінаторних методів. Застосовується як до повністю, так і частково цілочисельних задач. Його суть полягає в упорядкованому переборі варіантів і розгляді лише тих з них, які виявляються за певними ознаками корисними для знаходження оптимального рішення.

Результатом роботи алгоритму є знаходження максимуму функції на допустимій множині. При чому множина може бути як дискретною, так і раціональною. В ході роботи алгоритму

Динамічне програмування - спосіб вирішення складних завдань шляхом розбиття їх на більш прості підзадачі.

Динамічне програмування корисно, якщо на різних шляхах багаторазово зустрічаються одні й ті ж підзадачі; основний технічний прийом - запам'ятовувати рішення підзадач на випадок, якщо та ж підзадача зустрінеться знову.

У типовому випадку динамічне програмування застосовується до завдань оптимізації. У такої задачі може бути багато можливих рішень, але потрібно вибрати оптимальне рішення, при якому значення деякого параметра буде мінімальним або максимальним.

Найпростішим прикладом будуть числа Фібоначчі – щоб обчислити деяке число в цій послідовності, нам треба спершу обчислити третє число, склавши перші два, потім четверте *так само* на основі другого і третього, і так далі