Лекція 3. ПОШУК РІШЕНЬ ІНТЕЛЕКТУАЛЬНОЇ ЗАДАЧІ У ПРОСТОРІ СТАНІВ. «СЛІПІ МЕТОДИ»

- Класифікація методів пошуку ІЗ у просторі станів
- Пошук в глибину та ширину. Переваги і недоліки методів «сліпого» пошуку
- Методи евристичного пошуку

Людина переважно не розв'язує задачу в тій формі, в якій задача формулюється на початку. Вона прагне представити задачу таким чином, щоб їй було зручно її розв'язувати. Для цього виконуються перетворення початкового представлення задачі з метою скорочення простору, в якому необхідно виконувати пошук розв'язування задачі. При виборі ***способу представлення задачі*** зазвичай враховують дві обставини:

• представлення задачі повинно достатньо точно моделювати реальність;

• способи представлення повинні бути такими, щоб розв'язувачу задач було зручно з ними працювати.

Так як під розв'язувачем задач у штучному інтелекті є комп'ютер, тому розглянемо способи представлення задач, зручні для їх розв'язування на ЕОМ. До них відносяться наступні ***способи*** (форми), що найбільш часто використовуються :

• представлення задач у просторі станів;

• представлення, що зводить задачу до підзадач (методи пошуку рішень ІЗ у разі зведення задач до сукупності підзадач).

1. Класифікація методів пошуку ІЗ у просторі станів

Функціонування багатьох ІС носить цілеспрямований характер (прикладом можуть служити автономні інтелектуальні роботи). Типовим актом такого функціонування є рішення задачі планування шляху для досягнення потрібної мети з деякої фіксованої початкової ситуації. Результатом вирішення задачі повинні бути *план дій*.

Пошук плану дій виникає в ІС лише тоді, коли вона стикається з нестандартною ситуацією, ***для якої немає заздалегідь відомого набору дій***, що приводять до потрібної мети.

Розрізняються ***сліпі*** і ***направлені*** методи пошуку шляху (рис. 1).

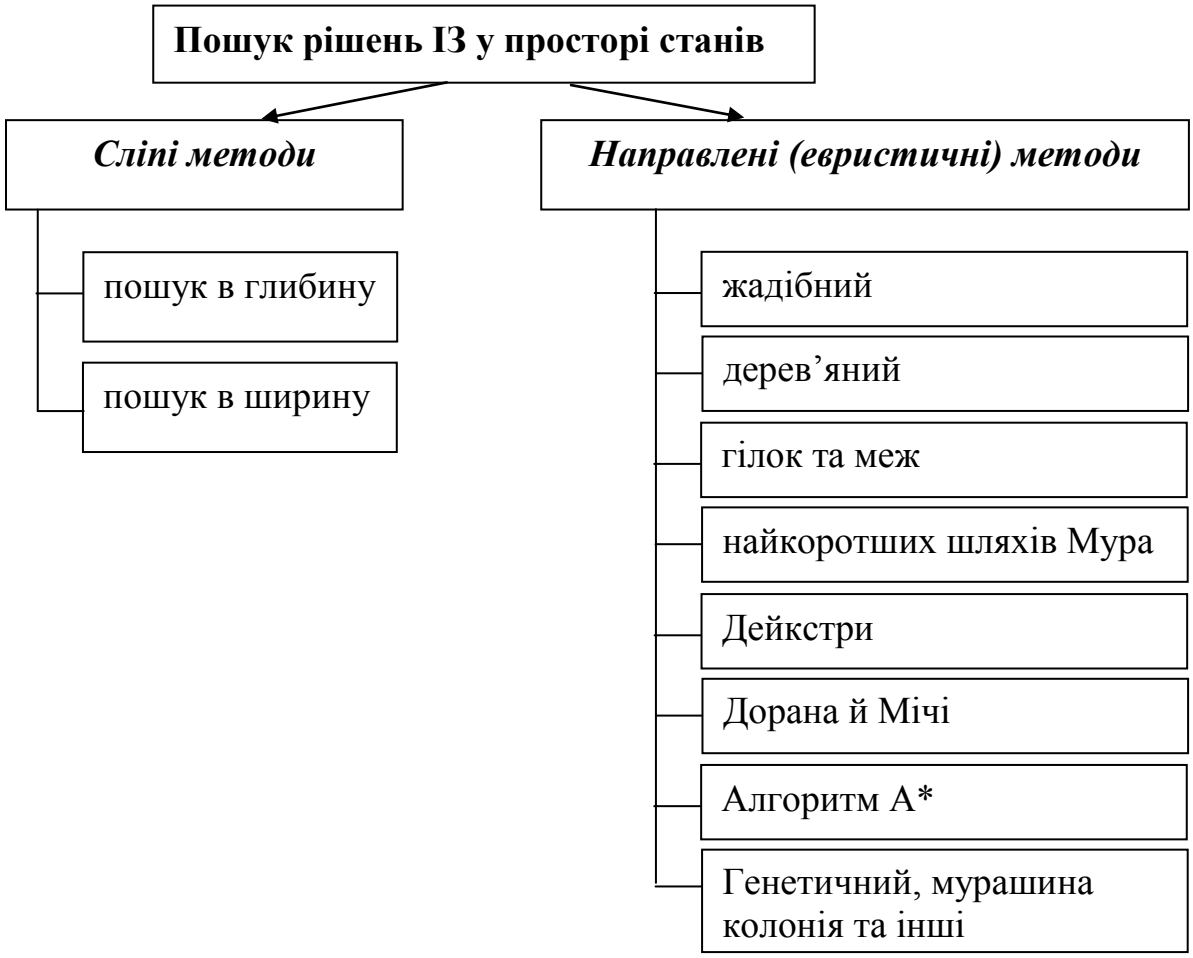


Рис 1. Класифікація методів пошуку ІЗ у просторі станів

2. Сліпі методи пошуку рішень просторі станів

Застосовуються для обходу дерев, структури подібної до дерев, або графів.

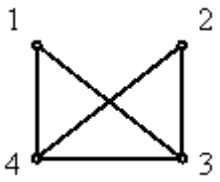
Графом G називається пара множин (V,E), де E – довільна підмножина множини V (2) (E $\subseteq$V (2)) та позначається G =(V,E).

Існує декілька способів завдання графів.

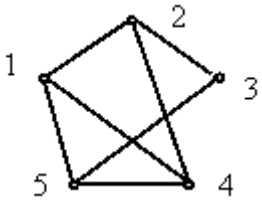
Перший спосіб задавання графа G =(V,E) за допомогою переліку їх елементів.

Приклад: граф G1=(V1,E1), V1={v1,v2,v3,v4} і E1={(v1,v3), (v1,v4), (v2,v3), (v2,v4), (v3,v4)} – це граф із чотирма вершинами і п'ятьма ребрами, граф G2=(V2,E2), V2={v1,v2,v3,v4,v5} і E2={(v1,v2),(v2,v4),(v1,v5), (v3,v2),(v3,v5),(v4,v1),(v5,v4)} – граф із п'ятьма вершинами і сімома ребрами.

Другий спосіб. Граф G =(V, E) зручно зображати за допомогою рисунка на площині, який називають діаграмою графа G. На рисунку 2 зображені діаграми графів G1 і G2 з попереднього прикладу.



G1



G2

Рис 2. Графічний спосіб задавання графів

Третій спосіб. Графи можна задавати також за допомогою матриць. Занумеруємо всі вершини графа G натуральними числами від 1 до *n*. Матрицею суміжності A графа G називається квадратна $n \times n$ матриця, в якій елемент a_{ij} *i*-го рядка і *j*-го стовпчика дорівнює 1, якщо вершини v_i та v_j з номерами *i* та *j* суміжні, і дорівнює 0 у противному разі.

Для графів G1 і G2 маємо відповідно:

$$A1 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \text{ та } A2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Очевидно, що матриці суміжності графів – симетричні.

Занумеруємо всі вершини графа G числами від 1 до *n* і всі його ребра – числами від 1 до *m*. Матрицею інцидентності B графа G називається $n \times m$-матриця, в якій елемент b_{ij} *i*-го рядка і *j*-го стовпчика дорівнює 1, якщо вершина v_i з номером *i* інцидентна ребру e_j з номером *j*, і дорівнює 0 у противному разі.

Для графів G1 і G2 маємо

$$B1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}, \qquad B2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Четвертий спосіб. Ще одним способом завдання графів є списки суміжності. Кожній вершині графа відповідає свій список. У список, що відповідає вершині *v*, послідовно записуються всі суміжні їй вершини.

Для графів G1 і G2 маємо списки

G1:	G2:
v1: v3,v4	v1: v2,v4,v5
v2: v3,v4	v2: v1,v3,v4
v3: v1,v2,v4	v3: v2,v5
v4: v1,v2,v3	v4: v1,v2,v5
	v5: v1,v3,v4

Вибір та зручність того чи іншого зі способів завдання графів залежать від особливостей задачі, яка розв'язується.

Для різних практичних застосувань теорії графів важливою є проблема систематичного обходу (перебору) всіх вершин і/або ребер графа.

Двома класичними методами розв'язання цих проблем є: *метод пошуку в ширину* та *метод пошуку в глибину*.

2.1. Пошук в глибину (англ. Depth-first search, DFS). Робота алгоритму починається з кореня дерева (або іншої обраної вершини в графі) і здійснюється обхід в максимально можливу глибину до переходу на наступну вершину (рис. 3)

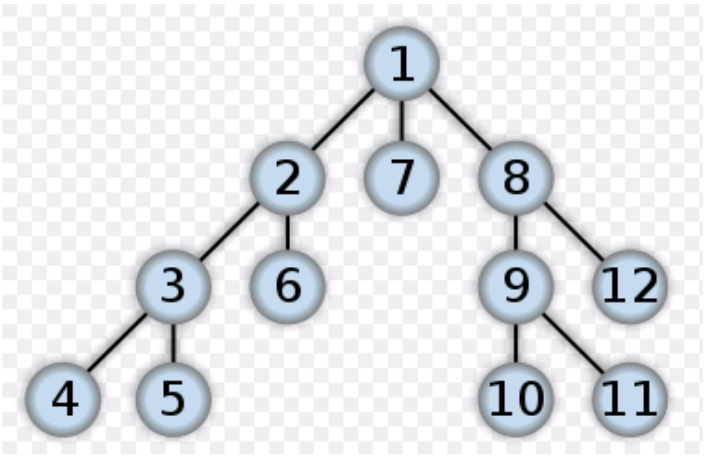


Рис. 3. Схема пошуку у глибину

2.2. Пошук у ширину. Якщо задано граф $G = (V, E)$ та початкову вершину s, алгоритм пошуку в ширину систематично обходить всі досяжні із s вершини. На першому кроці вершина s позначається, як пройдена, а в список додаються всі вершини, досяжні з s без відвідування проміжних вершин. На кожному наступному кроці всі поточні вершини списку відмічаються, як пройдені, а новий список формується із вершин, котрі є ще не пройденими сусідами поточних вершин списку (рис. 4). Для реалізації списку вершин найчастіше використовується черга. Виконання алгоритму продовжується до досягнення шуканої вершини або до того часу, коли на певному кроці у список не включається жодна вершина. Другий випадок означає, що всі вершини, доступні з початкової, уже відмічені, як пройдені, а шлях до цільової вершини не знайдений.

Алгоритм має назву пошуку в ширину, оскільки «фронт» пошуку (між пройденими та непройденими вершинами) одноманітно розширюється вздовж всієї своєї ширини. Тобто, алгоритм проходить всі вершини на відстані k перед тим, як пройти вершини на відстані $k+1$.

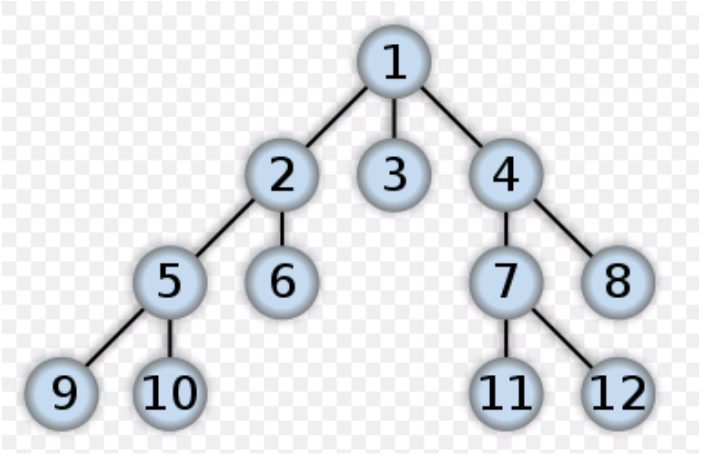


Рис. 4. Схема пошуку в ширину

2.3. Алгоритм пошуку вшир / вглиб

1. Всі "порожні" ребра розмістити у списку ВІДКР (у довільному порядку).

2. Якщо ВІДКР = ∅, то РОЗВ = ∅ (тобто сформульована задача не має розв'язку) і алгоритм завершує свою роботу.

3. Закрити перше ребро (v,w) з ВІДКР, тобто перенести ребро (v,w) зі списку ВІДКР у список ЗАКР.

4. Якщо вершина w закритого ребра є кінцевою вершиною ($w \in V_K$), то шуканий список РОЗВ (тобто шуканий шлях з V_O у V_K) міститься серед ребер списку ЗАКР і може бути виділений з нього послідовно, починаючи з останнього закритого ребра шляху. Зауважимо, що при побудові результуючого шляху для кожного з ребер (v,w) списку ЗАКР необхідно вибирати ребро-попередника (u,v) так, що воно є першим ребром з кінцем v у списку ЗАКР. Алгоритм завершує свою роботу.

У противному разі ($w \notin V_K$) перейти до пункту 5.

5. Визначити $S(w)$ – множину синів вершини w останнього закритого ребра, а також множину ребер $R(w)=\{(w,z) \mid z \in S(w)\}$.

Розмістити у списку ВІДКР усі ребра з множини $R(w) \setminus (\text{ВІДКР} \cup \text{ЗАКР})$ **після / перед** усіх ребер, що вже містяться в цьому списку.

6. Перейти до пункту 2.

Відмінність між обома алгоритмами пошуку знаходиться в позиції 5 і полягає в тому, що в алгоритмі пошуку вшир необхідно розміщувати відповідні ребра **після**, а в алгоритмі пошуку вглиб – **перед** усіма ребрами, що знаходяться в списку ВІДКР.

Для наведених алгоритмів вживають скорочені назви **АПШ** і **АПГ** (відповідні англійські назви – BFS (breadth first search) і DFS (depth first search)).

Таким чином, для АПШ список ВІДКР є *чергою*, тобто такою сукупністю елементів, в якій нові елементи розміщуються в кінці сукупності, а елемент, що "обслуговується" (закривається), вибирається з голови (початку) цієї сукупності. У той час для АПГ список ВІДКР є так званим *стеком*, тобто сукупністю, в якій елементи, що додаються до сукупності, і елементи, що відбираються для "обслуговування", розміщуються тільки на початку сукупності - у верхівці стеку (за принципом: “останній прийшов - перший обслуговується”).

Приклад. Розглянемо дію алгоритму пошуку вшир для графа, зображеного на рис.5 (див. табл. 1). Задано, що $V_0 = \{3\}$ і $V_K = \{11\}$.

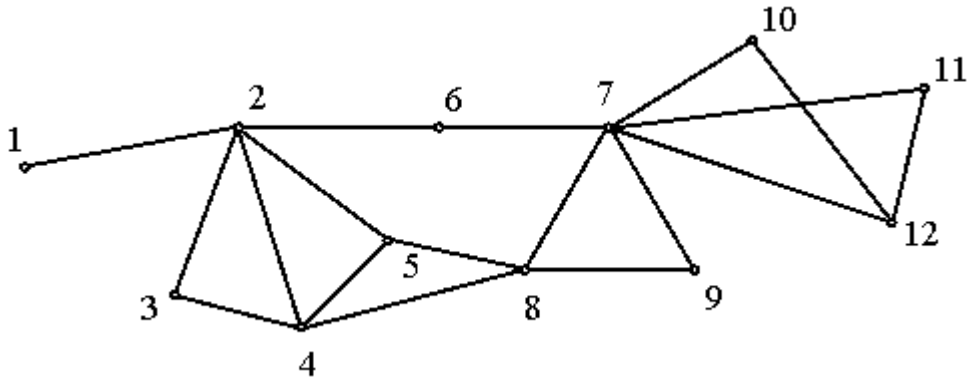


Рис. 5. Вхідний граф пошуку шляху

Кожний рядок таблиці описує результат виконання одного циклічного кроку (позиції 2–6) алгоритму. Оскільки список ЗАКР на кожному кроці поповнюється тільки одним ребром, то в таблиці записуємо лише це ребро (не повторюючи всі елементи, які ввійшли до складу ЗАКР на попередніх кроках). Нагадаємо також, що ребра (v,w) і (w,v) збігаються.

Таблиця 1

Алгоритм пошуку вшир (АПШ)

Крок	ВІДКР	ЗАКР	w	$R(w)$
0	(p,3)			
1	(3,2),(3,4)	(p,3)	3	(3,2),(3,4)
2	(3,4),(2,1),(2,4),(2,5),(2,6)	(3,2)	2	(2,1),(2,4),(2,5),(2,6)
3	(2,1),(2,4),(2,5),(2,6),(4,5),(4,8)	(3,4)	4	(4,5),(4,8)
4	(2,4),(2,5),(2,6),(4,5),(4,8)	(2,1)	1	
5	(2,5),(2,6),(4,5),(4,8)	(2,4)	4	

6	(2,6),(4,5),(4,8),(5,8)	(2,5)	5	(5,8)
7	(4,5),(4,8),(5,8),(6,7)	(2,6)	6	(6,7)
8	(4,8),(5,8),(6,7)	(4,5)	5	
9	(5,8),(6,7),(8,7),(8,9)	(4,8)	8	(8,7),(8,9)
10	(6,7),(8,7),(8,9)	(5,8)	8	
11	(8,7),(8,9)	(6,7)	7	(7,12),(7,11),(7,9),(7,10)
12	(8,9),(7,12),(7,11),(7,9),(7,10)	(8,7)	7	
13	(7,12),(7,11),(7,9),(7,10)	(8,9)	9	
14	(7,11),(7,9),(7,10),(12,10),(12,11)	(7,12)	12	(12,10),(12,11)
15	(7,9),(7,10),(12,10),(12,11)	(7,11)	11	

Алгоритм завершує свою роботу на п’ятнадцятому кроці. Аналізуючи список ребер ЗАКР від його кінця до початку, будуємо список РОЗВ, тобто знаходимо шуканий шлях від вершини 3 у вершину 11: 3,(3,2),2,(2,6),6,(6,7),7,(7,11),11. Довжина цього шляху – 4.

2.4. Переваги і недоліки методів «сліпого» пошуку

Сліпий метод має два види: пошук в глибину і пошук в ширину.

При пошуку в глибину кожна альтернатива досліджується до кінця, без врахування решти альтернатив.

Метод поганий для "високих" дерев, оскільки легко можна пройти повз потрібну гілку і витратити багато зусиль на дослідження "порожніх" альтернатив.

При пошуку в ширину на фіксованому рівні досліджуються всі альтернативи і лише після цього здійснюється перехід на наступний рівень.

Метод може виявитися гіршим за метод пошуку в глибину, якщо в графі всі шляхи, що ведуть до цільової вершини, розташовані приблизно на одній і тій же глибині.

Обидва сліпі методи вимагають великої витрати часу і тому необхідні направлені методи пошуку.

3. Методи евристичного пошуку

Евристичні алгоритми, або просто евристика, це алгоритми, які спроможні знайти прийнятне рішення проблеми серед багатьох рішень, але неспроможні гарантувати, що це рішення буде найкращим. ***Такі алгоритми є наближеними.***

Зазвичай такі алгоритми знаходять рішення близьке до найкращого і роблять це швидко.

Серед властивості евристичних алгоритмів варто відзначити:

- вони знаходять добрі, хоча і не завжди найкращі розв'язки з усіх, що існують.
- метод пошуку або побудови розв'язку звичайно значно простіший, ніж той що гарантує оптимальність розв'язку.

Для багатьох практичних проблем евристичні алгоритми, можливо, єдиний шлях для отримання прийнятого рішення в розумний проміжок часу.

Деякі з евристичних методів використовуються антивірусним ПЗ для виявлення вірусів та іншого шкідливого ПЗ.

Зазвичай ***евристичні алгоритми використовують допоміжну функцію*** (евристику), аби скеровувати напрям пошуку та скоротити його тривалість.

3.1. Жадібний алгоритм – алгоритм знаходження найкоротшої відстані шляхом вибору найкоротшого, ще не обраного ребра, за умови, що воно не утворює циклу з вже обраними ребрами. "Жадібний" цей алгоритм названий тому, що на останніх кроках доводиться жорстоко розплачуватися за жадібність.

Розглянемо роботу жадібного алгоритму при вирішенні задачі комівояжера (ЗК).

Комівояжер (бродячий торговець, турист тощо) повинен вийти з першого міста А, відвідати тільки по одному разу в невідомому порядку решту міст і повернутися в перше місто. Відстані між містами відомі. У якому порядку слід обходити міста, щоб замкнутий шлях (тур) комівояжера був найкоротшим?

Проблему комівояжера можна представити у вигляді моделі на графі, тобто, використовуючи вершини та ребра між ними. Таким чином, вершини графу відповідають містам, а ребра між вершинами – відстані (вартості, тривалості подорожі) між цими містами. Маршрутом (також гамільтоновим маршрутом) називається маршрут на цьому графі, до якого входить по одному разу кожна вершина графа. Задача полягає у відшуканні найкоротшого маршруту.

З метою спрощення задачі та гарантії існування маршруту, зазвичай вважається, що модельний граф задачі є повністю зв'язним, тобто, що між довільною парою вершин існує ребро. Це можна досягти тим, що в тих випадках, коли між окремими містами не існує сполучення, вводити ребра з максимальною вагою (довжиною, вартістю тощо). Через велику довжину таке ребро ніколи не потрапить до оптимального маршруту, якщо він існує.

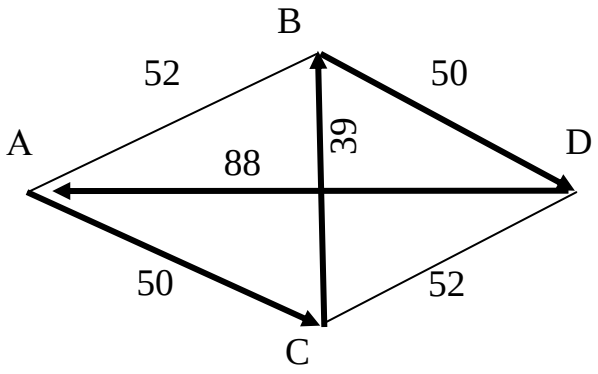


Рис. 6. Графічне представлення ЗК

Нехай комівояжер стартує з міста A. Алгоритм "Іти в найближче місто" виведе його в місто C, потім B, потім D; на останньому кроці доведеться платити за жадібність, повертаючись по довгій діагоналі ромба (рис. 6). У результаті вийде не найкоротший, а довжелезний тур: 50+39+50+88=227!

Наприклад, маршрут ребрами A-B-D-C-A матиме довжину лише 52+50+62+50=204.

3.2. Дерев'яний алгоритм. Розглянемо роботу дерев'яного алгоритму (побудову найкоротшого кістяка) на прикладі ЗК (рис. 7).

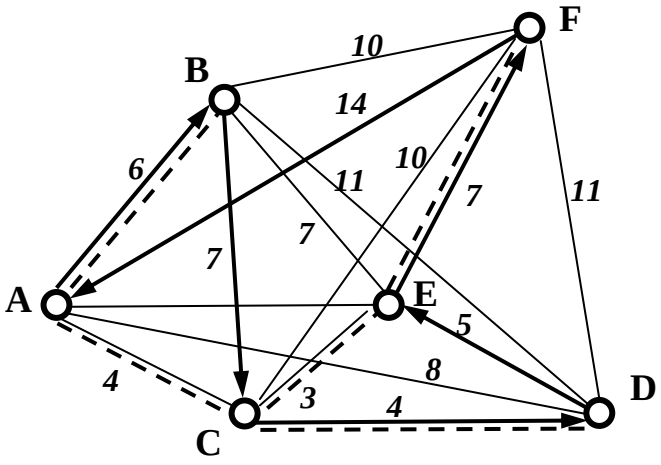


Рис. 7. Приклад роботи дерев'яного алгоритму

Дерев'яний алгоритм базується на Ейлеревому циклі. Замкнуту лінію, яка покриває всі ребра графа, тепер називають ***Ейлеровим циклом***. По-суті, Ейлером була доведена наступна теорема: Ейлерів цикл в графі існує тоді і тільки тоді, коли:

1) граф зв'язний;

2) всі його вершини мають парні ступеня.

Ця теорема показує, що якщо потрібно прокреслити фігуру однією замкненою лінією, то всі її вершини повинні мати парну ступінь.

Вірне і зворотне твердження: якщо всі вершини мають парну ступінь, то фігуру можна намалювати однієї незамкненою лінією. Дійсно, процес проведення лінії може скінчитися, тільки якщо лінія прийде в вершину, звідки вже виходу немає: всі ребра, приєднані до цієї вершини (зазвичай кажуть: інцидентних цій вершині), вже прокреслені.

Отже, дерев'яний алгоритм спочатку будує кістяк, показаний на рис. 2б штриховою лінією, потім Ейлеровий цикл A-B-A-C-D-C-E-F-E-C-A.

Далі переглянемо перелік вершин, починаючи з A, і будемо закреслювати кожну вершину, яка повторює вже зустрінуту у послідовності. Залишиться тур A-B-C-D-E-F-A завдовжки 6+7+4+5+7+14=43, який і є результатом алгоритму і показаний суцільний лінією.

Приклад: згадаємо старовинну головоломку – чи можна накреслити однією лінією відкритий конверт (рис. 8)? А закритий? Поясніть, чому?

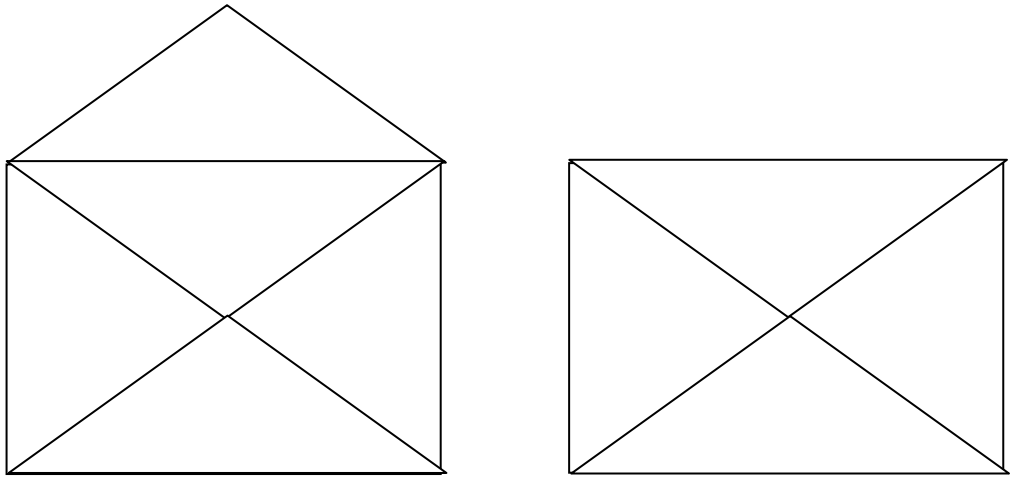


Рис. 8. Самостійне завдання роботи дерев'яного алгоритму

3.3. Метод гілок та меж (розгалужень і границь, розгалужень та меж). Розглянемо задачу про три процесори, які здатні виконувати завдання з однаковою швидкістю. Є набір 6 завдань, час виконання яких відповідно 7, 8, 9, 10, 11, 12. Порядок виконання завдань неважливий. Якщо процесор почав виконувати завдання, то виконує його до кінця протягом зазначеного часу. Переключення процесора на виконання нового завдання відбувається миттєво. Треба так розподілити завдання між процесорами, щоб момент закінчення останнього завдання був мінімальним. Назвемо цю величину ***вартістю розподілу***.

Якщо в зазначеному порядку розподілити перші три завдання між процесорами, а потім давати їх у тому ж порядку процесорам, що звільняються, то перший процесор закінчить роботу в момент 7+10=17, другий – у момент 8+11=19, а третій – 9+12=21. Маємо вартість 21. Проте їх можна розподілити інакше – 7+12, 8+11, 9+10, одержавши вартість 19.

Організуємо обхід дерева розподілів таким чином, що:

1) для кожного з вузлів обчислюється зазначена оцінка вартості,

2) вузли розглядаються у порядку зростання їх оцінок,

3) вузли з оцінкою, більшою від вартості вже одержаного повного розподілу, взагалі не розглядаються.

Ці міркування ***складають суть методу гілок і меж***. Впорядкування вузлів робить обхід цілеспрямованим, а відкидання явно неперспективних піддерев скорочує його.

Метод гілок і меж є варіацією методу повного перебору з тією різницею, що ми відразу виключаємо завідомо неоптимальні рішення.

Приклад. Нехай для чотирирівневого дерева є обмеження (рис. 9). Тоді, застосовуючи метод гілок і меж, можна скоротити кількість варіантів для перебору з 24-х до 8-ми. Однак метод гілок і меж працює не для всіх наборів даних. В деяких задачах час виконання буде таким же, як і для повного перебору.

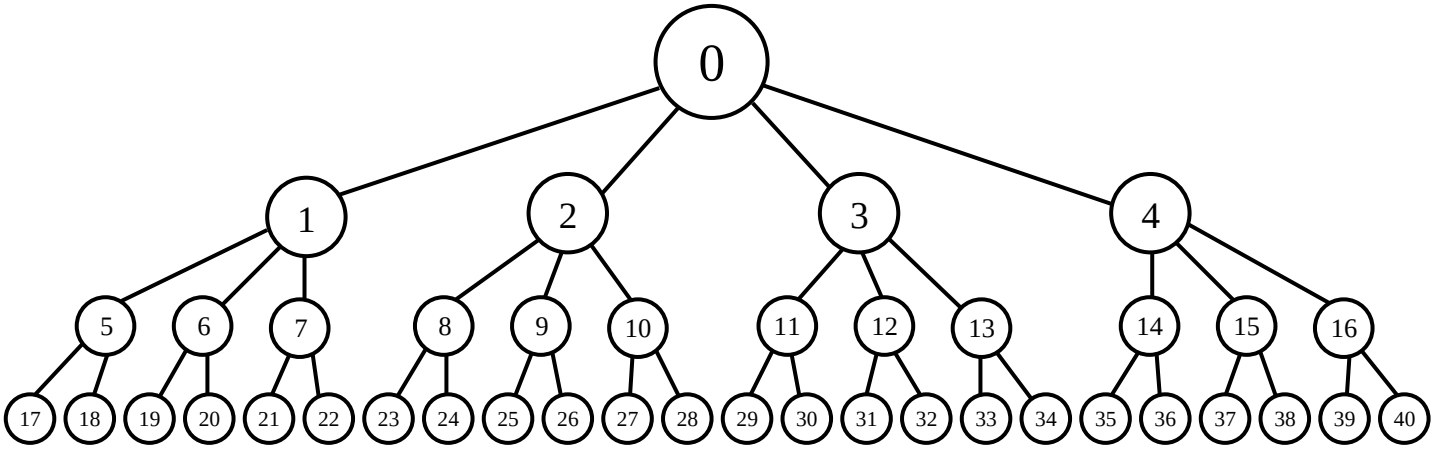


Рис. 9а. Дерево повного перебору

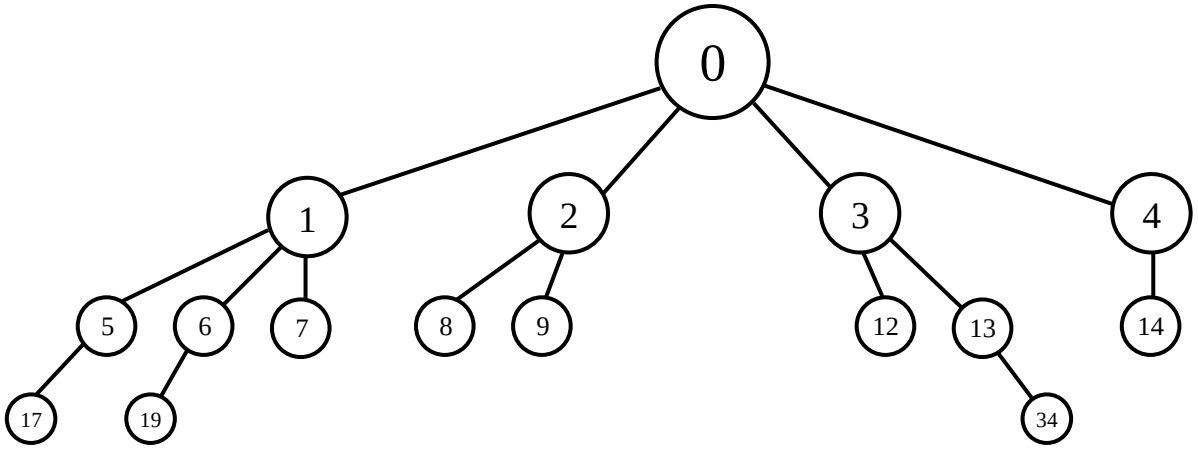


Рис. 9б. Дерево спрощене методом гілок і меж

3.4. Алгоритм найкоротших шляхів Мура. Вихідна вершина X_0 позначається числом 0. Нехай у ході роботи алгоритму на поточному кроці отримана множина дочірніх вершин $X(x_i)$ вершини x_i. Тоді з неї викреслюються всі раніше отримані вершини, ті, що залишилися, позначаються міткою, збільшеною на одиницю в порівнянні із міткою вершини x_i, і від них проводяться покажчики до X_i. Далі на множині позначених вершин, що ще не фігурують як адреси покажчиків, *вибирається вершина з найменшою міткою* й для неї будуються дочірні вершини. Розмітка вершин повторюється доти, поки не буде отримана цільова вершина.

3.5. Алгоритм Дейкстри визначення шляхів з мінімальною вартістю є узагальненням алгоритму Мура за рахунок введення дуг змінної довжини.

3.6. Алгоритм Дорана й Мічі пошуку з низькою вартістю. Використається, коли вартість пошуку велика в порівнянні з вартістю оптимального рішення. У цьому випадку замість вибору вершин, найменш вилучених від початку, як в алгоритмах Мура й Дейкстри, вибирається вершина, для якої евристична оцінка відстані до мети найменша. При гарній оцінці можна швидко одержати рішення, але немає гарантії, що шлях буде мінімальним.