ЛАБОРАТОРНА РОБОТА 1

Методи пошуку у просторі станів.

1 Ціль.

Дослідження методів пошуку рішень у просторі станів. Програмування базових алгоритмів пошуку рішень, що застосовуються у системах штучного інтелекту.

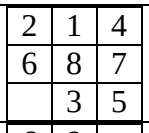
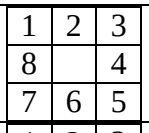
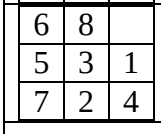
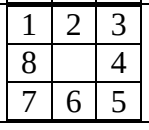
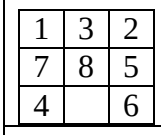
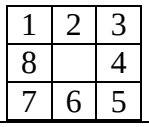
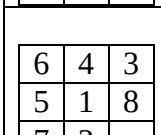
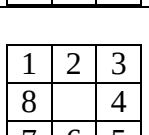
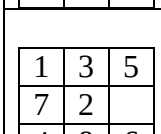
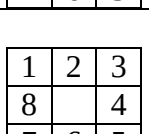
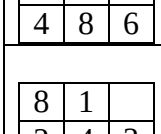
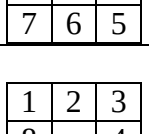
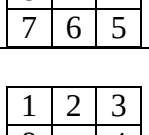
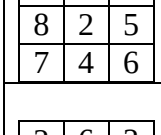
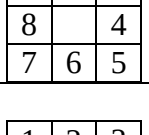
2 Завдання

Для рішення задач застосувати наступний алгоритм:

1) пошук вшир;
2) пошук вглиб;
3) жадібний алгоритм

1) Подати задачу, згідно з варіантом завдання, у просторі станів.
2) Знайти рішення інтелектуальної задачі за допомогою двох методів (алгоритмів): для парних варіантів - метод пошуку вшир, для непарних варіантів - метод пошуку вглиб; та для всіх варіантів застосувати метод з використанням евристик.
3) Порівняти час знаходження розв'язку.

Таблиця 1.

№	Завдання		
1	Гра в вісім. В грі використовується вісім фішок, пронумерованих з 1 до 8. фішки розташовані у дев'яти комірках, що утворюють матрицю 3×3. Одна з комірок завжди порожня. Будь-яку фішку, сусідню з порожньою коміркою, можна пересунути на її місце, неприпустимо пересуватися через фішку та по діагоналі. Задача с полягає в тому, щоб перетворити конфігурацію, надану ліворуч, в кінцеву конфігурацію (праворуч):		
2			
3			
4			
5			
6			
7			
8			

<table>
<tr><td>9</td><td rowspan="3"></td><td><table><tr><td></td><td>4</td><td>3</td></tr><tr><td>5</td><td>2</td><td>8</td></tr><tr><td>7</td><td>1</td><td>6</td></tr></table></td><td><table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td></tr>
<tr><td>10</td><td><table><tr><td>1</td><td>3</td><td>4</td></tr><tr><td>7</td><td>8</td><td>2</td></tr><tr><td></td><td>6</td><td>5</td></tr></table></td><td><table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td></tr>
<tr><td>11</td><td><table><tr><td>2</td><td>1</td><td>4</td></tr><tr><td></td><td>7</td><td>3</td></tr><tr><td>6</td><td>5</td><td>8</td></tr></table></td><td><table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td></tr>
<tr><td>12</td><td colspan="3">Задача про комівояжера.
Комівояжер повинен побувати у кожному місті, яке зображене на мапі. Між судніми городами є шлях, довжина вказана на мапі. Потрібно , вирушити зі стартового міста, знайти найкоротший шлях, по якому комівояжер проходить по одному разу через кожне місто, щоб опинитися в стартовому місті.</td></tr>
<tr><td>13</td><td colspan="3">Задача про рюкзак.
Дана впорядкована «за незменшенням» послідовність цілих додатних чисел. Кожне число може бути асоційовано з об'ємом деякого предмету, який турист може взяти з собою у похід. Дано також ціле додатне число, яке може бути асоційовано з об’ємом рюкзака. Потрібно : знайти усі підпослідовності початкової послідовності, сума елементів яких дорівнює цьому числу (знайти сукупності тих предметів, які увійдуть до рюкзака при повному його заповненні).</td></tr>
<tr><td>14</td><td colspan="3">Завдання про місіонерів і канібалів.
Троє місіонерів і троє канібалів знаходяться на лівому березі річки. Всі хочуть опинитися на іншому березі. Є невеликий човен, що вміщує не більше двох людей. Якщо на якомусь березі канібалів виявиться більше, ніж місіонерів, то вони з'їдять місіонерів. Якщо виявиться більше місіонерів, то вони навернуть канібалів до своєї віри. Потрібно знайти послідовність переміщень човна з одного берега на інший, що гарантує безпеку місіонерам і свободу віросповідання канібалам</td></tr>
<tr><td>15</td><td colspan="3">Завдання про шахового коня (завдання Ейлера).
Потрібно обійти всі клітини шахівниці ходом коня.</td></tr>
<tr><td>16</td><td colspan="3">Завдання “Ханойська башта”.
Є три стрижня, на першому з яких поміщені N дисків різного діаметру, причому менший диск обов'язково лежить на більшому. Потрібно перемістити всі диски на третій стрижень, рухаючи їх по черзі. Хід в цій головоломці полягає в знятті верхнього диска з одного із стрижнів і переміщенні його поверх дисків (якщо вони є) іншого стрижня. Обмеження полягає в тому, що більший диск не можна класти на менший.</td></tr>
<tr><td>17</td><td colspan="3">Магараджі і пері.
Сталося так, що до берега великого Гангу під’їхали відразу троє магараджів разом зі своїми пері. Всі вони хотіли переправитися на інший берег, але звичаї не дозволяли жодній пері залишатися в човні або на березі одній з чужим чоловіком, якщо поруч не буде свого. Біля берега стояв невеликий човен, який може витримати не більше двох людей. Звичаї не забороняють пері самій керувати човном. Потрібно знайти послідовність переміщень човна з одного берега на інший, що гарантує переміщення всіх магараджі і пері на інший берег.</td></tr>
</table>

Теоретичні відомості

Метод пошуку вшир

Розглянемо граф на рис. 1.4. Стани в ньому позначено буквами, A, B, C,..., щоб на них можна було посилатись у подальших міркуваннях. Пошук вшир досліджує простір станів за рівнями, один за одним. І лише коли станів на даному рівні більше немає, алгоритм переходить до наступного рівня. Пошук вшир досліджує вершини графа на рис. 1.5 у такому порядку: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

Пошук вшир виконується з використанням списків OPEN і CLOSED, які дають змогу відслідковувати просування в просторі станів. Список OPEN містить вершини, які підлягають розгортанню, а список CLOSED - вершини, які вже були розгорнуті.

Алгоритм пошуку вшир у кореневому графі

Ініціалізація

Крок l.OPEN:=$S_{поч}$

Крок 2. CLOSED:=∅

Ітерація

Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершина X

Крок 4. Якщо X=$S_{ціль}$, то Вихід(УСПІХ); інакше перейти до кроку 5

Крок 5. Розгорнути вершину X (тобто згенерувати всіх її синів)

Крок 6. Помістити вершину X у список CLOSED

Крок 7. Виключити синів вершини X, які є в списку OPEN або CLOSED (це перевірка на цикл)

Крок 8. Решту синів помістити в правий кінець списку OPEN (отже, список OPEN - це черга)

Перевірка закінчення

Крок 9. Якщо OPEN≠∅, то перейти до кроку 3, інакше Вихід (НЕВДАЧА)

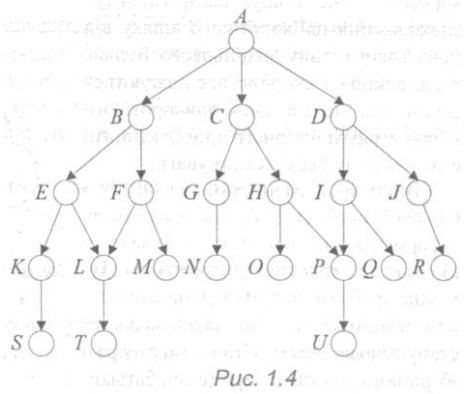


Рис. 1.4

Синівські вершини генерують за операторами з множини F, допустимими ходами гри, правилами виведення або іншими операціями переходу станів. На кожній ітерації генерують усі вершини, які є синами вершини X (розгортають X). Саму вершину X записують у список CLOSED, а всіх її синів - у список OPEN, причому синів, які вже є в списках OPEN або CLOSED, повторно не записують. Вказівники дають змогу в разі успіху легко відновити шлях від початкової вершини до цільової. Зазначимо, що список OPEN працює як черга й опрацьовує дані в порядку їхнього надходження (або „першим надійшов - першим обслужений"). Це структура даних FIFO (first in - first out). Стани добавляють у список OPEN справа, а вилучають зліва. Отже, у пошуку беруть участь стани, які перебувають у списку OPEN найдовше - це й забезпечує пошук вшир. Якщо алгоритм завершається через невиконання умови OPEN≠∅, то можна зробити висновок, що весь граф досліджено, а бажану мету досягти не вдалося. Отже, пошук зазнав невдачі.

Приклад 1.6. Прослідкуємо роботу алгоритму пошуку вшир на графі, зображеному на рис. 1.4. Числа 2, 3, 4,... означають номер ітерації, і нехай U-цільовий стан.

1. OPEN = [A]; CLOSED = [ ].
2. OPEN = [B, C, D]: CLOSED = [A].
3. OPEN = [C, D, E, F]; CLOSED = [B, A].
4. OPEN = [D, E, F, G, H]; CLOSED = [C, B, A].
5. OPEN = [E, F, G, H, I, J]; CLOSED = [D, C, B, A].
6. OPEN = [F, G, H, I, J, K, L]; CLOSED = [E, D, C, B, A].
7. OPEN = [G, H, I, J, K, L, M] (оскільки L уже в OPEN); CLOSED = [F, E, D, C, B, A].
8. OPEN = [H, I, J, K, L, M, N]; CLOSED = [G, F, E, D, C, B, A].

І так далі, доки не буде знайдено U, або OPEN=0.

На рис. 1.5 показано граф, зображений на рис. 1.4, після шістьох ітерацій пошуку вшир. Стани зі списків OPEN і CLOSED на рис. 1.5 виділено точковою й пунктирною лініями, відповідно. Зазначимо, що „прикордонні" стани пошуку на будь-якій стадії записують в OPEN, а вже розглянуті - у CLOSED.

Оскільки під час пошуку вшир вершини графа розглядають за рівнями, то спочатку досліджуються ті стани, шляхи до яких коротші. Отже, пошук вшир гарантує знаходження найкоротшого шляху від початкового стану до цільового. Більше того, оскільки скоріше досліджуються стани, знайдені вздовж найкоротшого шляху, у разі повторного проходження ці стани вже не беруться до уваги.

Іноді крім імен станів в OPEN і CLOSED необхідно зберігати додаткову інформацію. Якщо шлях є розв'язком, то він має повертатись алгоритмом. Це можна зробити, наприклад, накопиченням інформації про батька для кожного стану вздовж шляху. Стан у такому разі зберігають у вигляді пари (стан, батько).

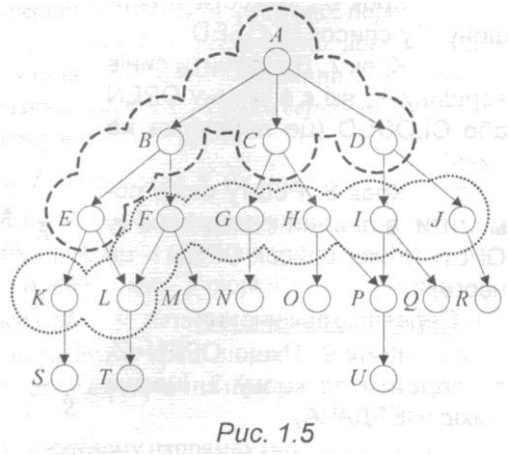


Рис. 1.5

Приклад 1.7. Для графа з прикладу 1.6 вміст списків OPEN і CLOSED на четвертій ітерації був би таким:

OPEN = [(D, A) (E, B), (F, B), (G, C), (H, C)]; CLOSED = [(C, A), (B, A), (A, nil)].

Використовуючи цю інформацію, можна легко побудувати шлях A, B, F, який веде від A до F. Коли ціль знайдено, алгоритм може відновити розв'язок, відслідковуючи його у зворотному напрямку від цілі до початкового стану за батьківськими вершинами. Зазначимо, що стан A має батька nil (нуль), тобто є початковим станом. Це слугує сигналом припинення відновлення шляху. Оскільки пошук ушир знаходить кожний стан уздовж найкоротшого шляху та зберігає першу версію кожного стану, то цей шлях від початку до цілі є найкоротшим.

МЕТОД ПОШУКУ ВГЛИБ

На відміну від методу перебору вшир, цей метод пропонує розгортати, передусім, ту вершину, яку було побудовано останньою. Першою вершиною, що розгортають, очевидно, є коренева вершина. Але процес спочатку завжди буде йти по самій лівій гілці вершин. Якщо утворений на поточний момент шлях виявився марним, тобто за заданої межі глибини цільову вершину не досягнуто, необхідно повернутись у вершину, яка передувала розгорнутій вершині, і спробувати ще раз застосувати до неї оператор розгортання. І так діяти доти, доки не буде одержано цільову вершину

У цьому алгоритмі стани-сини додаються й вилучаються з лівого кінця списку OPEN, тобто список OPEN організовано як стек, або структура LIFO (last in - first out, що означає „останнім надійшов - першим обслужений"). У разі організації списку OPEN у вигляді стека перевага надається „наймолодшим" (нещодавно генерованим) станам, тобто здійснюється принцип пошуку вглиб.

Алгоритм пошуку вглиб у кореневому графі

Ініціалізація

Крок l.OPEN:=$S_{поч}$.

Крок 2. CLOSED:=∅.

Ітерація

Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершина X.

Крок 4. Якщо X=$S_{ціль}$, то Вихід(УСПІХ); інакше перейти до кроку 5.

Крок 5. Розгорнути вершину X(тобто згенерувати всіх її синів)

Крок 6. Помістити вершину X у список CLOSED.

Крок 7. Виключити синів вершини X, які є в списку OPEN або CLOSED (це перевірка на цикл)

Крок 8. Решту синів помістити в лівий кінець списку OPEN (отже, список OPEN - це стек).

Перевірка закінчення

Крок 9. Якщо OPEN≠∅, то перейти до кроку 3, інакше Вихід(НЕВДАЧА).

Приклад 1.8. Прослідкуємо роботу алгоритму пошуку вглиб на графі, зображеному на рис. 1.4. Числа 2, 3, 4, ... означають номер ітерації, і нехай U - цільовий стан. Початковий вміст списків OPEN і CLOSED показано в рядку 1.

1. OPEN = [A]; CLOSED = [ ].
2. OPEN = [B, C, D]; CLOSED = [A].
3. OPEN = [E, F, C, D]; CLOSED = [B, A].
4. OPEN = [K, L, F, C, D]; CLOSED = [E, B, A].
5. OPEN = [S, L, F, C, D]; CLOSED = [K, E, B, A].
6. OPEN = [L, F, C, D]; CLOSED = [S, K, E, B, A].
7. OPEN = [T, F, C, D]; CLOSED = [L, S, K, E, B, A].
8. OPEN = [F, C, D]; CLOSED = [T, L, S, K, E, B, A].
9. OPEN = [M, C, D] (оскільки L уже в CLOSED); CLOSED = [F, T, L, S, K, E, B, A]. 10 OPEN = [C, D]; CLOSED = [M, F, T, L, S, K, E, B, A].
11. OPEN = [G, H, D]; CLOSED = [C, M, F, T, L, S, K, E, B, A]. І так далі, доки не буде знайдено U, або OPEN=0.

Як і в разі пошуку вшир, у списку OPEN перелічені всі виявлені, але ще не оцінені стани (поточна „межа пошуку"), а в CLOSED записано вже розглянуті стани. На рис. 1.6 показано граф, зображений на рис. 1.4, після шістьох ітерацій пошуку вшир. Стани зі списків OPEN і CLOSED на рис. 1.6 виділено, як і раніше, точковою та штриховою лініями, відповідно. Як і в алгоритмі пошуку вшир,уданому алгоритмі можна зберігати для кожного стану запис про батька. Це дасть змогу алгоритму відновити шлях, який веде від початкового стану до цільового. Якщо під час роботи алгоритму позначати дуги, то повернення можна також здійснити за допомогою міток на дугах.

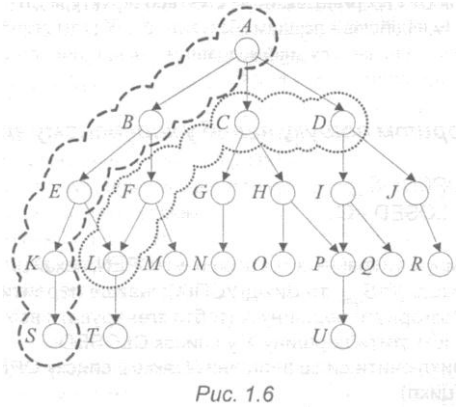


Рис. 1.6

На відміну від пошуку вшир, пошук углиб не гарантує знаходження оптимального шляху до стану, якщо він зустрівся вперше. Пізніше в процесі пошуку можуть бути знайдені різні шляхи до будь-якого стану. Якщо довжина шляху є важливою для розв'язку задачі, то в разі знаходження алгоритмом деякого стану повторно, необхідно зберегти коротший шлях. Це можна зробити, якщо зберігати для кожного стану трійку значень (стан, батько, довжина_шляху). Під час генерування синівських вершин довжину шляху просто збільшують на одиницю й зберігають разом із синами. Якщо синівський стан досягнуто уздовж декількох шляхів, цю інформацію можна використати для збереження кращого варіанту. Зазначимо, що збереження кращого варіанту стану під час звичайного пошуку вглиб зовсім не гарантує, що мету буде досягнуто саме уздовж найкоротшого шляху.

МЕТОД ПОШУКУ ПО ПЕРШОМУ НАЙКРАЩОМУ ЗБІГУ

Найпростіший спосіб евристичного пошуку - це застосування процедури пошуку екстремуму. Стратегії, які ґрунтуються на пошуку екстремуму, оцінюють не тільки поточний стан пошуку, але і його синів. Для продовження пошуку вибирають найкращого сина; при цьому про його братах і батька просто забувають. Пошук припиняють, коли досягнуто стан, котрий ліпший ніж будь-який із його синів. Головна проблема стратегій пошуку екстремуму - це їх тенденція зупинятись у локальному максимумі. Інакше кажучи, як тільки вони досягають стану, який має кращу оцінку, ніж його сини, алгоритм завершується. Якщо цей стан не є розв'язком задачі, а лише локальним максимумом, то такий алгоритм не підходить для даної задачі. Це означає, що розв'язок може бути оптимальним на обмеженій множині, але через форму всього простору, можливо, ніколи не буде вибрано найліпший розв'язок.

Незважаючи на ці обмеження, алгоритм пошуку екстремуму може бути достатньо ефективним, якщо оцінна функція дає змогу уникнути локального максимуму й зациклювання алгоритму. Загалом евристичний пошук потребує гнучкого методу, передбаченого в алгоритмі пошуку по першому найкращому збігу (best first search, його ще називають „жадібним" алгоритмом), коли накопичення відповідної інформації дає змогу відновити алгоритм із точки локального максимуму.

Подібно до алгоритмів пошуку вглиб і пошуку вшир, „жадібний" алгоритм пошуку використовує списки збережених станів: список OPEN відслідковує поточний стан пошуку, а в CLOSED записують уже перевірені стани. На кожному кроці алгоритм записує в список OPEN

стан із урахуванням евристичної оцінки його „близькості до цілі". Отже, на кожній ітерації розглядають найбільш перспективні стан із списку OPEN

Алгоритм пошуку по першому найкращому збігу у кореневому графі

Ініціалізація

Крокl.OPEN:= $S_{поч}$

Крок2. CLOSED:= ∅

Ітерація

Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершинаX

Крок 4. Якщо X= $S_{ціль}$, то Вихід(шлях від $S_{поч}$ до X); інакше перейти до кроку 5

Крок 5. Розгорнути вершину X(тобто згенерувати всіх її синів)

Крок 6. Для кожного сина вершини X виконати залежно від випадку:

1) син не міститься в OPEN або CLOSED: присвоїти сину евристичне значення та записати його в OPEN;

2) син уже міститься в OPEN: якщо його було досягнуто коротшим шляхом, то присвоїти цьому стану в списку OPEN цей коротший шлях;

3) син уже міститься в CLOSED: якщо його було досягнуто коротшим шляхом, то виключити цей стан зі списку CLOSED і записати його в OPEN

Крок 7. Помістити вершину X у список CLOSED

Крок 8. Переупорядкувати стани в списку OPEN відповідно до евристики (кращі зліва)

Перевірка закінчення

Крок 9. Якщо OPEN≠∅, то перейти до кроку 3, інакше Вихід(НЕВДАЧА).

На кожній ітерації алгоритм вилучає перший елемент зі списку OPEN. Досягнувши мети, алгоритм повертає (тобто видає) шлях, котрий веде до цілі. Зазначимо, що кожний стан зберігає попередню інформацію для визначення, чи було його раніше досягнуто коротшим шляхом. Це і дає змогу алгоритму повернути завершальний шлях до цілі. Збереження попередньої інформації здійснюють так, як це описано в підрозділі 1.7, тобто накопиченням інформації про батька для кожного стану вздовж шляху.

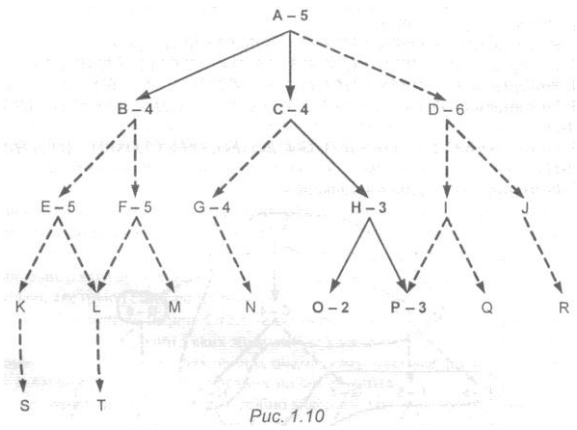


Рис. 1.10

Якщо перший елемент в OPEN - не $S_{ціль}$, то алгоритм використовує оператори з множини F для отримання всіх синів даного елемента. Якщо син уже наявний у списку OPEN або CLOSED, то алгоритм вибирає коротший з двох можливих шляхів досягнення цього стану. Оновлюючи історію предків вершин у списках OPEN і CLOSED, коли вони досягаються повторно, алгоритм з більшою ймовірністю знайде коротший шлях до мети $S_{ціль}$

Алгоритм обчислює евристичну оцінку станів в OPEN і сортує список за цими евристичними значеннями. При цьому „кращі" стани поміщаються в початок списку.

Зазначимо, що через евристичну природу оцінювання наступний стан має перевірятись на кожному рівні простору станів. Посортований список OPEN часто називають пріоритетною чергою.

На рис. 1.10 зображено гіпотетичний простір із евристичними оцінками деяких станів. Стани, поруч із якими є евристична оцінка, були генеровані алгоритмом пошуку за першим найліпшим станом. Стани, за якими здійснювався евристичний пошук, позначено напівжирним шрифтом; зазначимо, що пошук не відбувається по всьому простору. Суть „жадібного" алгоритму пошуку полягає в тому, щоб підійти до цільового стану аналізом можливо меншої кількості станів; що більш обгрунтована евристика, то менше станів потрібно перевірити при пошуку цілі.

Шлях, за яким алгоритм пошуку по першому найкращому збігу знаходить цільовий стан, показано на рисунку напівжирними стрілками. Нехай P - цільовий стан (див. рис. 1.10). Оскільки P - ціль, то стани на шляху до P мають низькі евристичні значення (передбачається що оцінна функція h(N) оцінює довжину найкоротшого шляху від стану N до цільового стану). Евристика може помилятись: стан O має нижче евристичне значення ніж цільовий, і тому його досліджено раніше. На відміну від пошуку екстремуму, який не зберігає пріоритетну чергу для відбору „наступних" станів, даний алгоритм відновлюється після помилки і знаходить цільовий стан.

1. OPEN = [A-5]; CLOSED = 0.
2. Розгортаємо A-5: OPEN = [B-4, C-4, D-6]; CLOSED = [A-5].
3. Розгортаємо B-4: OPEN = [C-4, E-5, F-5, D-6]; CLOSED = [B-4, A-5].
4. Розгортаємо C-4: OPEN = [H-3, G-4, E-5, F-5, D-6]; CLOSED = [C-4, B-4, A-5].
5. Розгортаємо H-3: OPEN = [O-2, P-3, G-4, E-5, F-5, D-6]; CLOSED = [H-3, C-4, B-4,A-5].
6. Розгортаємо O-2: OPEN = [P-3, G-4, E-5, F-5, D-6]; CLOSED = [O-2, H-3, C-4, B-4,A-5].
7. Розгортаємо P-3: розв'язок знайдено!

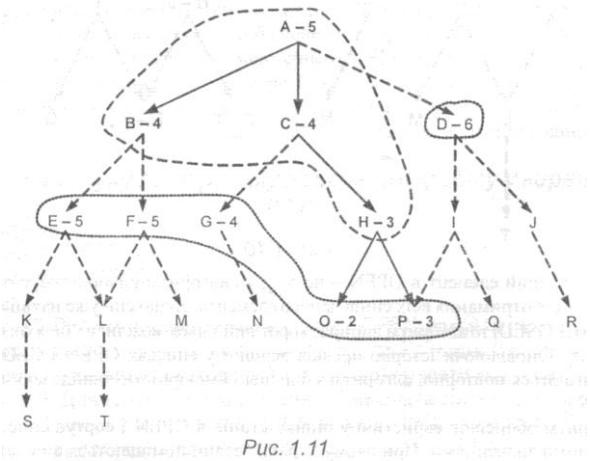


Рис. 1.11

На рис. 1.11 зображено простір після п'ятої ітерації. Стани зі списків OPEN і CLOSED виділено, відповідно, точковою та штриховою лініями. В OPEN зберігається поточна інформація, а в CLOSED - уже розглянуті стани. Зазначимо, що межа пошуку дуже викривлена, що відображає здатність наведеного алгоритму пристосовуватись.

„Жадібний"алгоритм пошуку завжди вибирає найбільш перспективний стан в OPEN для продовження. Проте, оскільки під час вибору стану використовується евристика, яка може виявитись помилковою, алгоритм не відмовляється від інших станів, і зберігає їх у списку OPEN. У даному випадку евристика спрямувала пошук шляхом, який виявився хибним, але алгоритм у результаті повернувся до деякого раніше генерованого „кращого" стану в OPEN і після цього продовжив пошук в іншій частині простору. У прикладі на рис. 1.10 після знаходження синів стану B здалося, що вони мають погані евристичні оцінки, і тому пошук змістився до стану C. Сини B збережені в OPEN на випадок, якщо алгоритму необхідно буде повернутись до них пізніше

Список літератури.

1. Нильсон Н. Искусственный интеллект : Пер. с англ. – М.: Мир, 1973
2. Слэйгл Дж. Искусственный интеллект : Пер. с англ. – М.: Мир, 1973
3. Гарднер М. Математические головоломки и развлечения : Пер. с англ. – М.: Мир, 1971
4. Берж К. Теория графов и ее применения : Пер. с фр. - М.: Издательство иностранной литературы, 1962
5. Харари Ф. Теория графов : Пер. с англ. - М.: Мир, 1973.