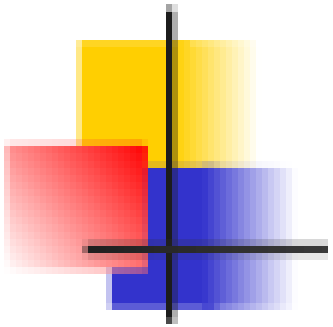


Методи та системи штучного інтелекту

Лекція 3. Розв’язання проблем за допомогою пошуку

Зміст

- Агенти, що розв’язують задачі
- Формулювання проблем
- Приклади проблем
- Базові алгоритми пошуку
- Пошук з частковою інформацією

Агенти, що розв’язують задачі

- Агенти визначають, що робити, знаходячи послідовності дій, які приводять до бажаних станів
- Цілі дозволяють організувати поведінку, обмежуючи вибір проміжних етапів, які намагається виконати агент
- Першим кроком є формулювання цілі за умов поточної ситуації та показників продуктивності агента
- Ціль – це множина станів світу, в яких досягається ця ціль
- Формулювання задачі представляє процес визначення того, які дії та стани слід розглядати з урахуванням певної цілі

Агенти, що розв’язують задачі

- Процес пошуку – це визначення найкращої послідовності дій, серед таких послідовностей, що ведуть від початкового до цільового стану
  - Будь-який алгоритм пошуку на вхід приймає деяку задачу та повертає розв’язок у формі послідовності дій
- Після пошуку йде стадія виконання дій, що були рекомендовані алгоритмом
- Загальна схема:
  - Сформулювати проблему
  - Знайти розв’язок
  - Виконати дії

Агенти, що розв’язують задачі

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Приклад: Україна

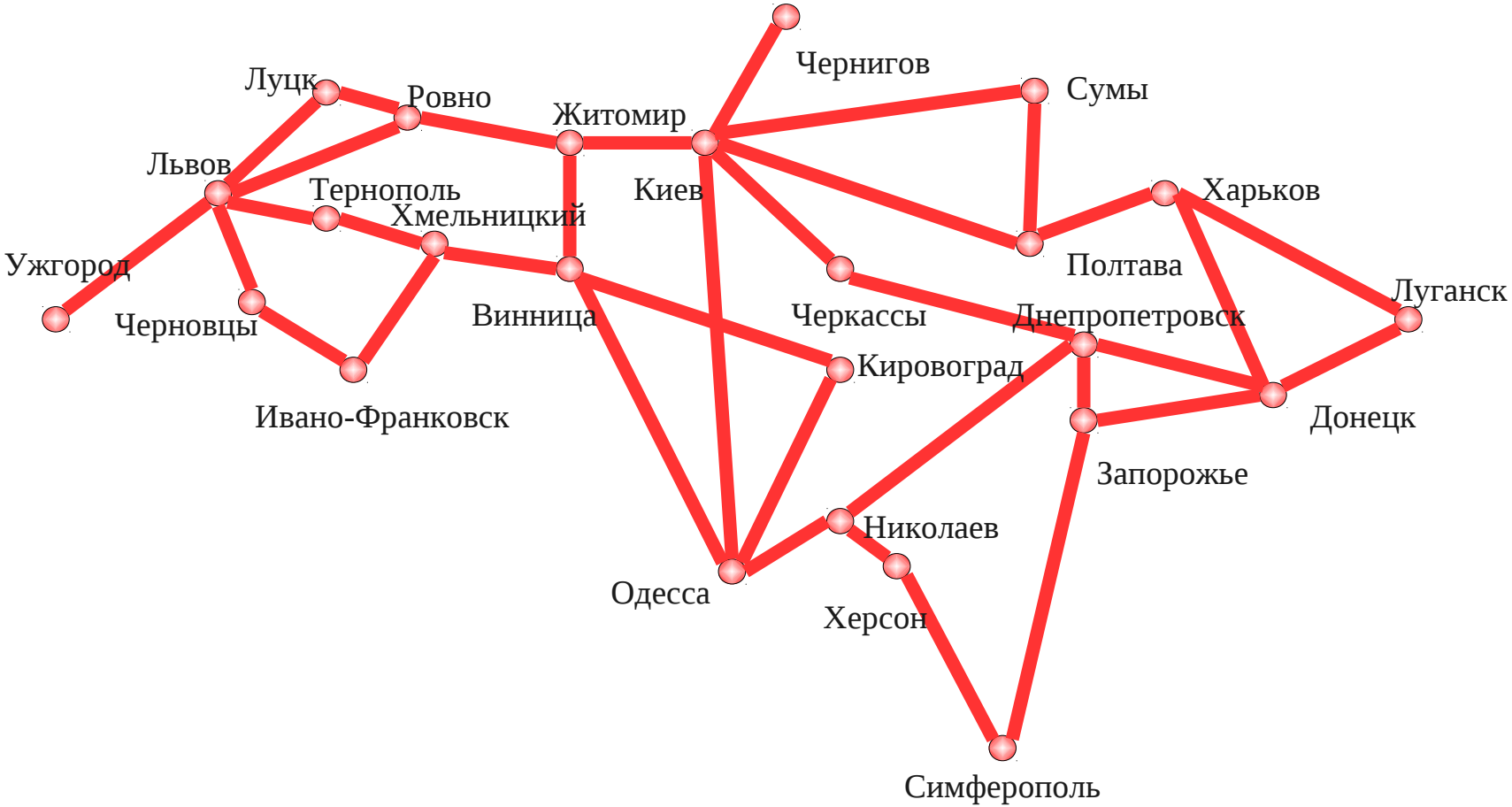
- На вихідних у Карпатах
- Потрібно завтра бути в Києві
- Ціль:
  - Знаходитись в Києві
- Проблема:
  - Стани: різні міста
  - Дії: подорож між містами
- Пошук рішення:
  - Послідовність міст, наприклад: Івано-Франківськ, Тернопіль, Дубно, Рівне, Житомир, Київ

Приклад: Україна

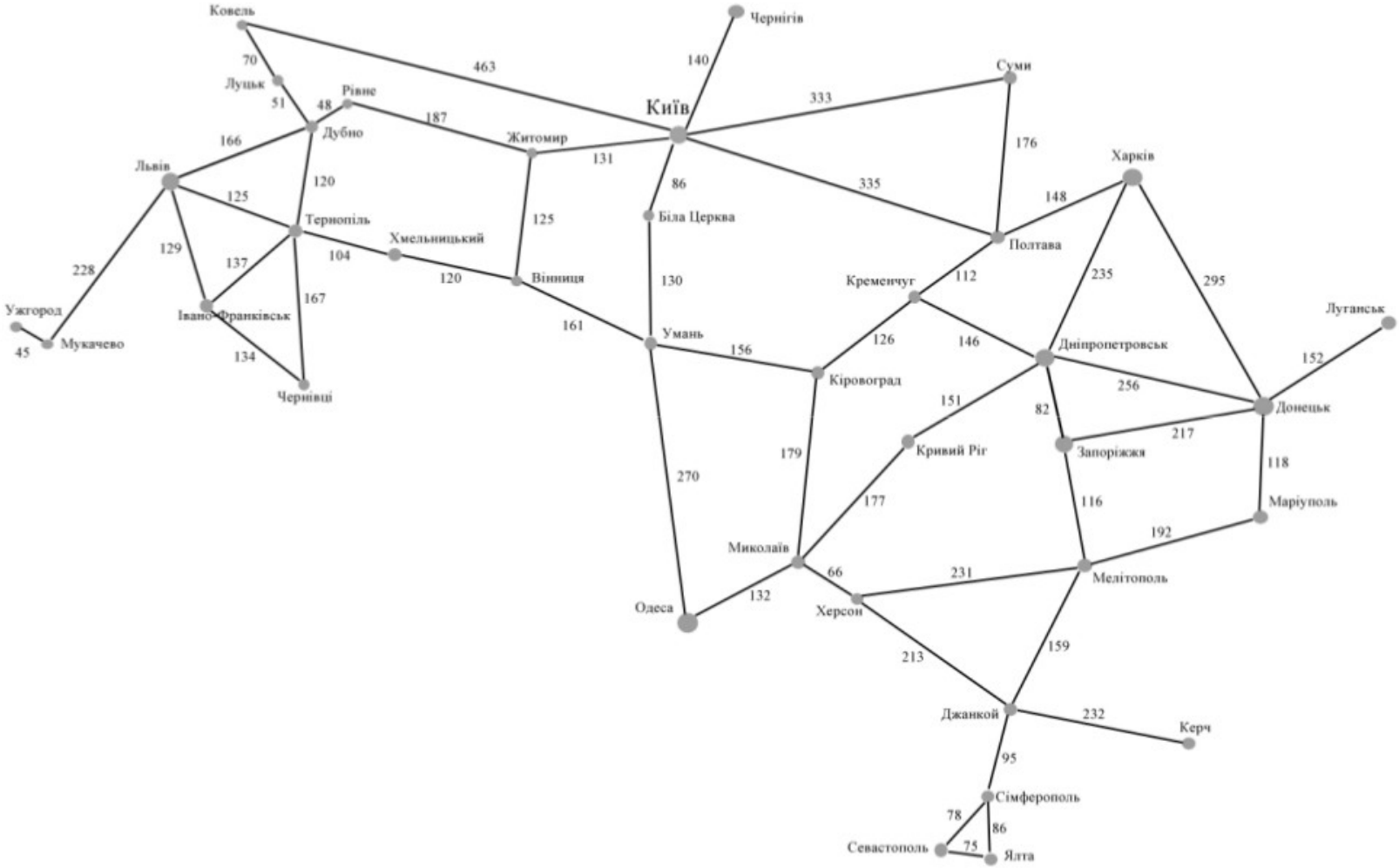
Приклад: Україна

	Города	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1.	Винница	0	571	812	126	637	373	266	317	972	387	369	466	429	576	313	801	611	239	593	720	533	122	340	312	423
2.	Днепропетровск	571	0	250	630	89	952	479	246	401	888	948	329	463	183	814	458	366	818	1172	222	316	701	326	891	585
3.	Донецк	812	250	0	880	243	1202	729	496	151	1138	1198	579	713	391	1064	571	488	1068	1422	283	535	951	576	1141	826
4.	Житомир	126	630	880	0	719	459	140	434	987	261	407	592	555	494	187	927	485	325	679	638	659	208	352	398	297
5.	Запорожье	637	89	243	719	0	1018	568	314	394	977	1014	352	486	277	903	371	460	884	1238	303	292	767	415	957	674
6.	Ивано-Франковск	373	952	1202	459	1018	0	599	698	1353	273	135	785	658	953	292	1124	944	137	301	1097	856	251	721	143	756
7.	Киев	266	479	729	140	568	599	0	299	836	398	544	517	480	343	324	852	339	465	819	487	584	348	201	538	151
8.	Кировоград	317	246	496	434	314	698	299	0	647	692	694	180	337	251	618	524	434	564	918	395	251	447	126	637	436
9.	Луганск	972	401	151	987	394	1353	836	647	0	1245	1349	730	864	493	1171	722	535	1219	1573	303	686	1102	727	1292	873
10.	Луцк	387	888	1138	261	977	273	398	692	1245	0	152	853	816	752	70	1188	743	163	432	896	920	268	610	336	555
11.	Львов	369	948	1198	407	1014	135	544	694	1349	152	0	843	743	898	215	1178	889	127	278	1042	910	247	717	278	701
12.	Николаев	466	329	579	592	352	785	517	180	730	853	843	0	134	488	779	339	671	713	1067	551	71	596	368	642	671
13.	Одесса	429	463	713	555	486	658	480	337	864	816	743	134	0	596	742	473	779	676	959	685	205	559	453	515	634
14.	Полтава	576	183	391	494	277	953	343	251	493	752	898	488	596	0	678	631	183	819	1173	144	499	702	271	892	412
15.	Ровно	313	814	1064	187	903	292	324	618	1171	70	215	779	742	678	0	1114	669	158	495	823	846	192	536	331	481
16.	Симферополь	801	458	571	927	371	1124	852	524	722	1188	1178	339	473	631	1114	0	814	1048	1402	657	279	931	649	981	959
17.	Сумы	611	366	488	485	460	944	339	434	535	743	889	671	779	183	669	814	0	810	1164	185	682	693	343	883	338
18.	Тернополь	239	818	1068	325	884	137	465	564	1219	163	127	713	676	819	158	1048	810	0	353	963	780	117	587	176	622
19.	Ужгород	593	1172	1422	679	1238	301	819	918	1573	432	278	1067	959	1173	495	1402	1164	353	0	1317	1134	471	941	444	976
20.	Харьков	720	222	283	638	303	1097	487	395	303	896	1042	551	685	144	823	657	185	963	1317	0	538	846	415	1036	523
21.	Херсон	533	316	535	659	292	856	584	251	686	920	910	71	205	499	846	279	682	780	1134	538	0	663	411	713	738
22.	Хмельницкий	122	701	951	208	767	251	348	447	1102	268	247	596	559	702	192	931	693	117	471	846	663	0	470	190	505
23.	Черкассы	340	326	576	352	415	721	201	126	727	610	717	368	453	271	536	649	343	587	941	415	411	470	0	660	311
24.	Черновцы	312	891	1141	398	957	143	538	637	1292	336	278	642	515	892	331	981	883	176	444	1036	713	190	660	0	695
25.	Чернигов	423	585	826	297	674	756	151	436	873	555	701	671	634	412	481	959	338	622	976	523	738	505	311	695	0

Приклад: Україна



Приклад: Україна



Формулювання проблем

- Початковий стан
  - В Івано-Франківську, In(Ів.-Фр.)
- Функція визначення наступника
  - S(x) = множина пар “дія-стан”
  - S(Ів.-Фр.) = {[Go(Львів),In(Львів)],
    [Go(Тернопіль),In(Тернопіль)],
    [Go(Чернівці),In(Чернівці)]}
- Початковий стан разом із функцією визначення наступника неявно задають простір станів задачі – множину всіх станів, які можна досягнути з початкового стану
  - Простір станів утворює граф, в якому вузли – окремі стани задачі, дуги – дії, що призводять до переводу задачі з одного стану в інший
  - Шляхом в просторі станів є послідовність станів, які з’єднані послідовністю дій

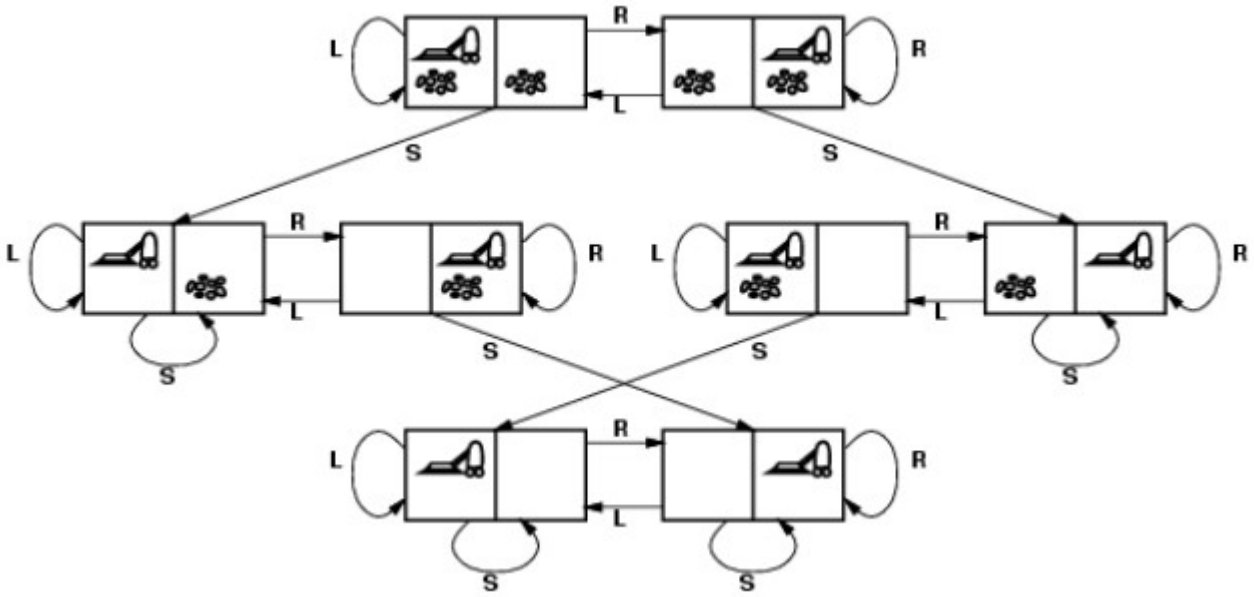
Формулювання проблем

- Перевірка цілі
  - Явна перевірка, x=“In(Київ)”
  - Неявна перевірка, NoDirt(x)
- Вартість шляху
  - Сума відстаней, кількість виконаних дій, тощо
  - c(x,a,y) – вартість етапу для переходу з x в y за допомогою дії a, c(x,a,y)≥0
- Рішення – це шлях від початкового стану до цільового стану. Якість рішення вимірюється функцією вартості шляху.
- Оптимальне рішення має найменшу вартість

Вибір простору станів

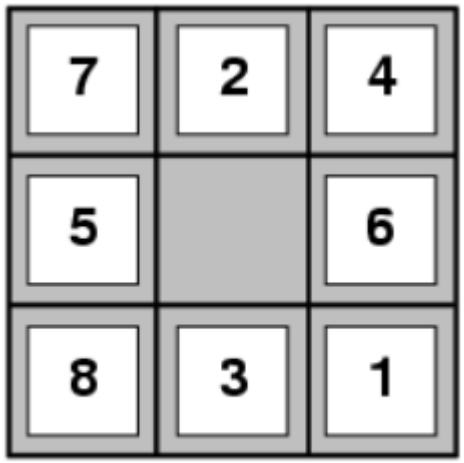
- Реальний світ є надскладним → повинно використовуватись абстрагування для розв’язання проблеми
- (Абстрактний) Стан = множина реальних станів
- (Абстрактна) Дія = комплексна комбінація реальних дій
- (Абстрактне) Рішення = множина реальних шляхів, які є розв’язками у реальному світі
- Будь-яка абстрактна дія повинна бути “легшою” ніж реальна

Приклад: граф простору станів для світу пилососу

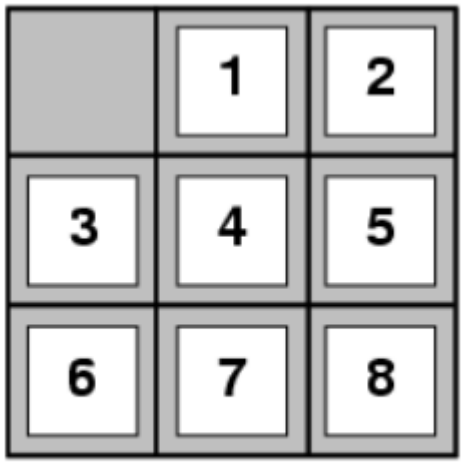


- Стани: положення та наявність сміття
- Початковий стан: будь-який стан
- Дії: Left, Right, Suck, NoOp
- Перевірка цілі: чи є чистими всі квадрати
- Вартість шляху: 1 за кожну дію (0 за NoOp)

Приклад: задача гри в 8 (8-puzzle)



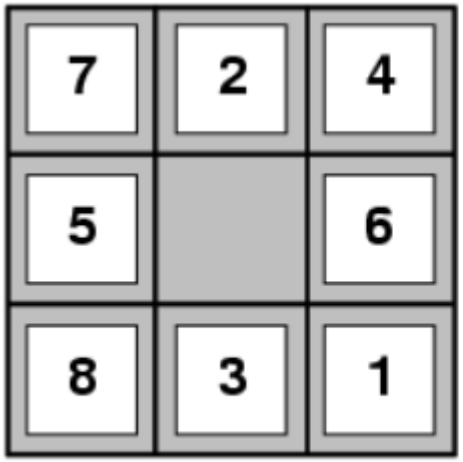
Start State



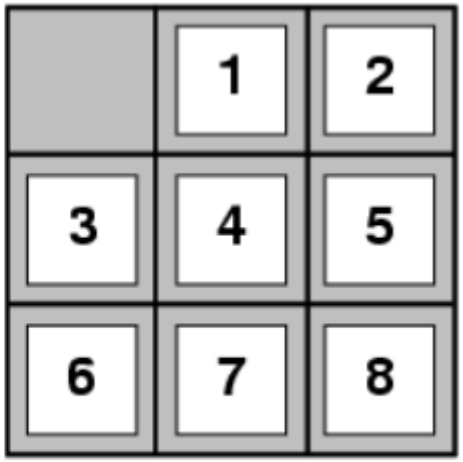
Goal State

- Стани: комбінація положень фішок
- Початковий стан: будь-який стан (для якого існує розв’язок)
- Дії: Left, Right, Up, Down
- Перевірка цілі: співпадає поточний стан з цільовою конфігурацією?
- Вартість шляху: 1 за кожний крок

Приклад: задача гри в 8 (8-puzzle)



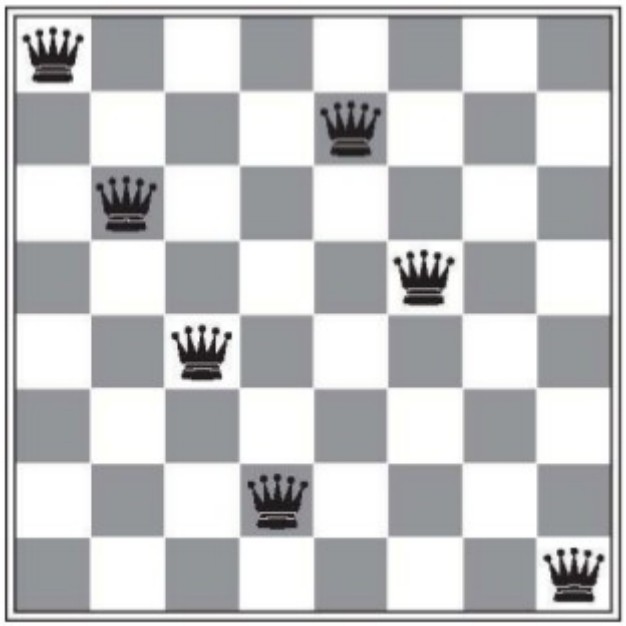
Start State



Goal State

- Пошук оптимального розв’язку - NP-повна задача
- 8-puzzle: 9!/2 = 181 440 станів
- 15-puzzle: ~1,3x10^{12} станів
- 24-puzzle: ~10^{25} станів

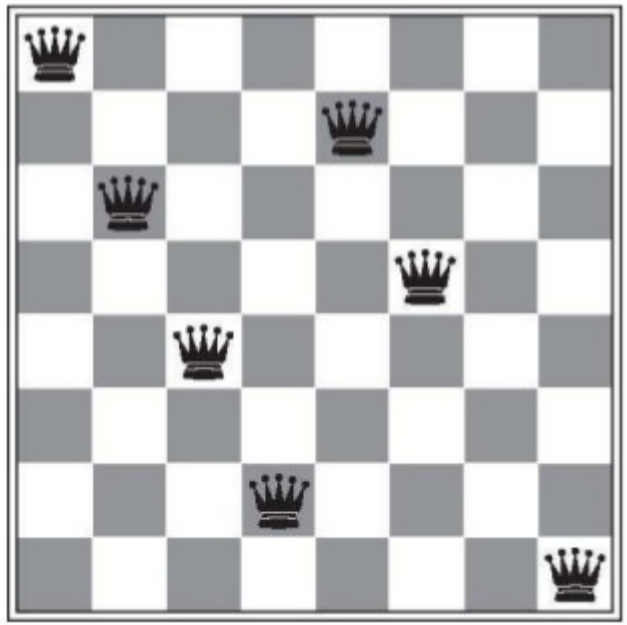
Приклад: 8 ферзів



Формулювання повних станів

- Стани: На дошці присутні всі вісім ферзів
- Початковий стан: Будь-який стан
- Дії: Переставити одного ферзя на вільну клітину
- Перевірка цілі: Жоден ферзь не є атакованим

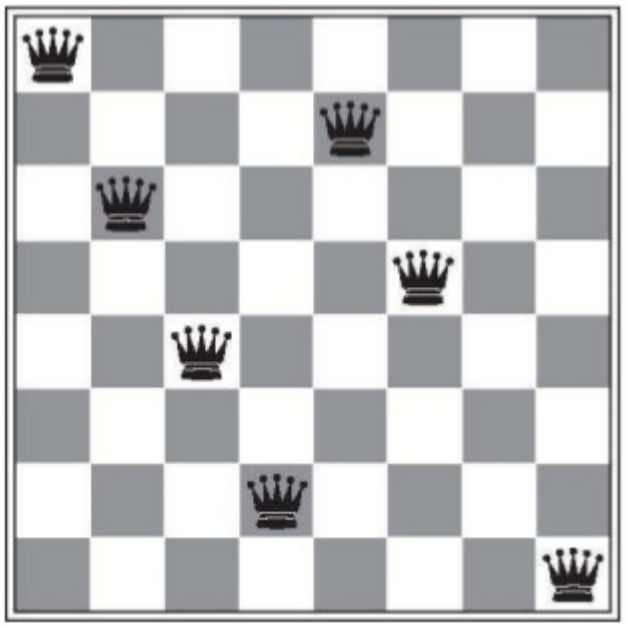
Приклад: 8 ферзів



Інкрементне формулювання

- Стани: Будь-яке розташування ферзів на дошці в кількості від 0 до 8
- Початковий стан: Відсутність ферзів на дошці
- Дії: Встановити ферзя на будь-яку порожню клітину
- Перевірка цілі: На дошці 8 ферзів і жоден не є атакованим
- Кількість послідовностей: $64 \times 63 \times \ldots \times 57 \sim 3 \times 10^{14}$

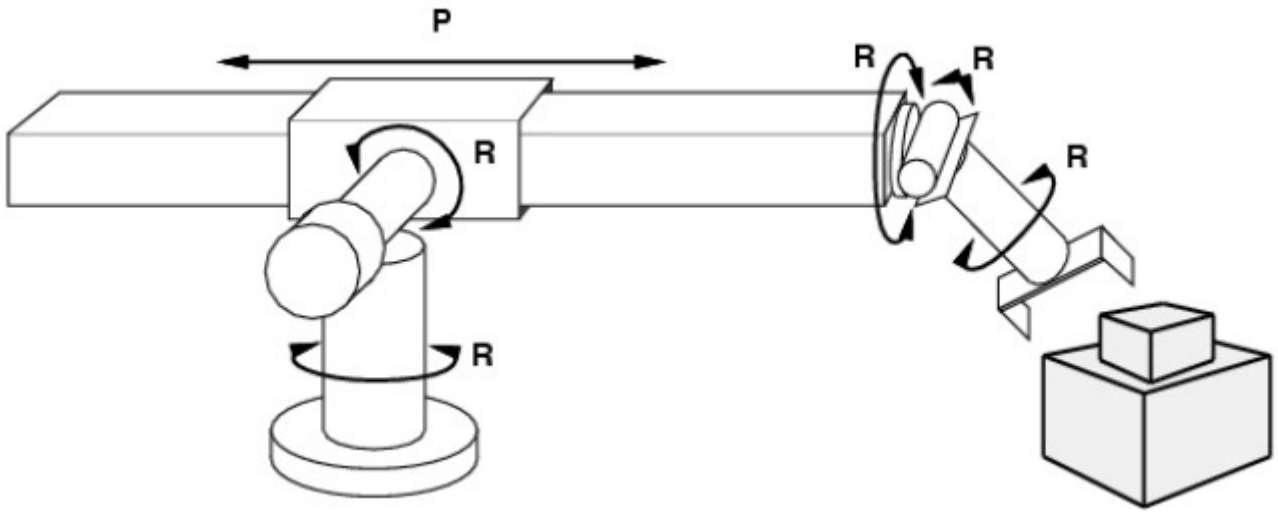
Приклад: 8 ферзів



Покращення інкрементного формулювання

- Стани: Будь-яке розташування ферзів на дошці в кількості від 0 до 8
  Кожний ферзь на окремій вертикалі і не атакує попереднього
- Дії: Встановити ферзя на будь-яку порожню клітину у першій вільній справа вертикалі так, щоб він не атакувався іншими ферзями
- Кількість станів: 2057

Приклад: роботизоване збирання деталей



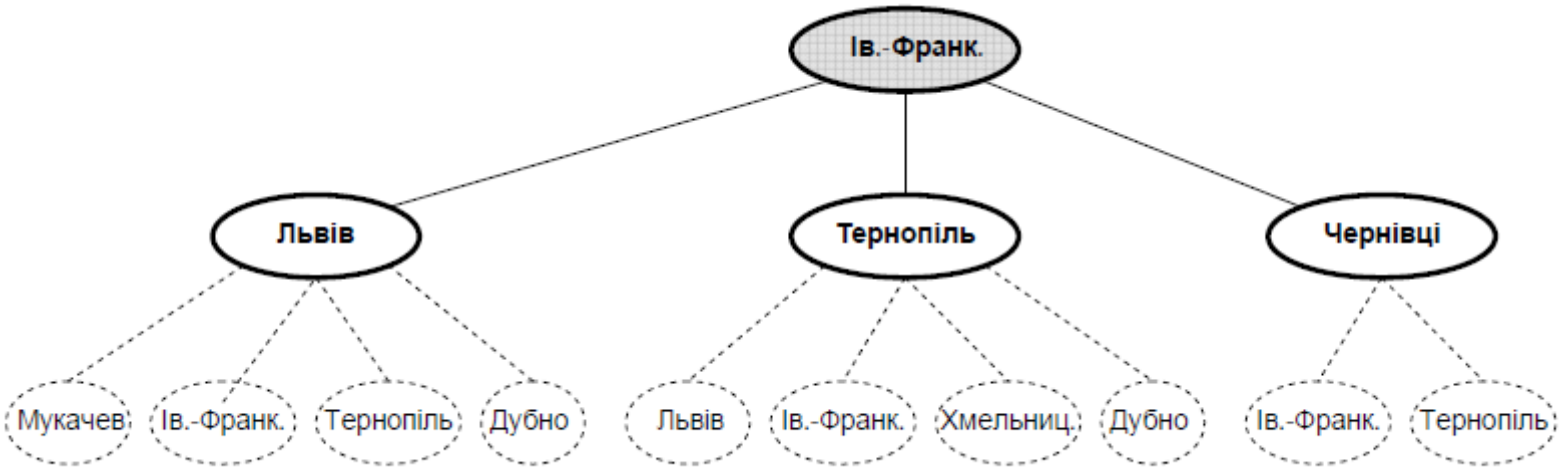
- Стани: дійсні значення координат кутів маніпулятору деталі об’єкту, який необхідно зібрати
- Дії: безперервні рухи частин роботу
- Перевірка цілі: зібраний об’єкт без наявності роботу
- Вартість шляху: використаний час

Алгоритми пошуку в дереві

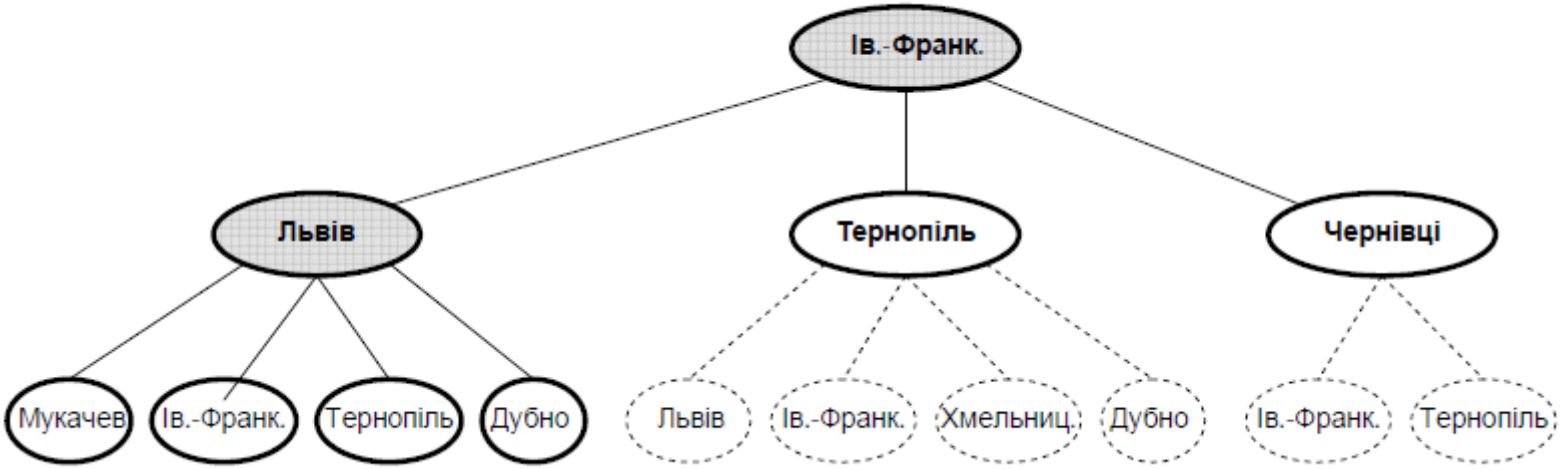
- Ідея:
  - симулювати дослідження простору станів “із закритими очима”
  - шляхом генерування спадкових (наступних) станів для вже досліджених станів

Приклад пошуку в дереві

Приклад пошуку в дереві

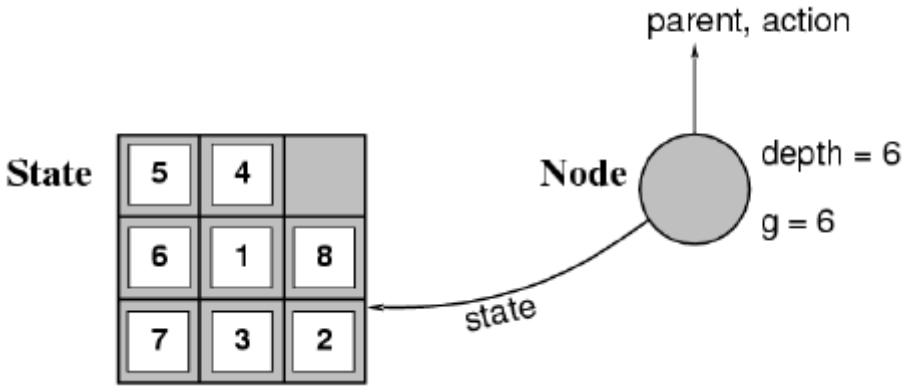


Приклад пошуку в дереві



Реалізація: стани та вузли

- Стан – це представлення фізичної конфігурації світу
- Вузол – структура даних, яка відображає частину дерева пошуку
- Вузол включає:
  - State – стан у просторі станів
  - Parent-node – батьківський вузол, який використовувався для створення поточного
  - Action – дія, яка була застосована до батьківського вузла
  - Path-Cost – вартість шляху (від початкового стану до поточного вузла) g(x)
  - Depth – кількість етапів шляху від початкового стану



Реалізація: стани та вузли

- Під час пошуку будується дерево пошуку. Його вузлами – є стани задачі, корінь – початковий стан.
- В кожний момент часу розглядається деякий вузол дерева, що відповідає поточному стану. Під час розгортання поточного стану-вузла до дерева додаються вузли, які відповідають станам, що згенеровані функцією визначення наступника
- Порядок, в якому розгортаються стани, визначається стратегією пошуку
  - Колекція вузлів, які були сформовані, але ще не розгорнуті, називається периферією – множина листових вузлів

Реалізація: загальний пошук в дереві

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Пошукові стратегії

- Стратегії характеризуються наступними параметрами:
  - Повнота – чи завжди буде знайдено рішення, якщо воно існує?
  - Оптимальність – чи завжди знаходиться оптимальне рішення?
  - Часова складність – за який час алгоритм знаходить розв’язок?
  - Просторова складність - який об’єм пам’яті потрібен для виконання пошуку?

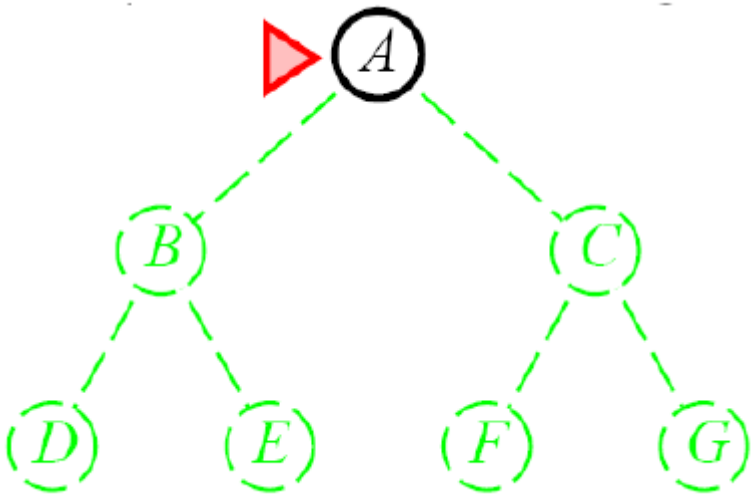
- Часова та просторова складність вимірюються в термінах:
  - b – максимальний фактор розгалуження у дереві
  - d – глибина найбільш поверхневого цільового вузла
  - m – максимальна довжина будь-якого шляху у просторі станів

Неінформативні стратегії пошуку

- Неінформативні стратегії використовують тільки інформацію з визначання проблеми (тобто не роблять жодних додаткових припущень)
  - Пошук вшир
  - Пошук за критерієм вартості
  - Пошук вглиб
  - Пошук з обмеженням глибини
  - Пошук вглиб з ітеративним заглибленням

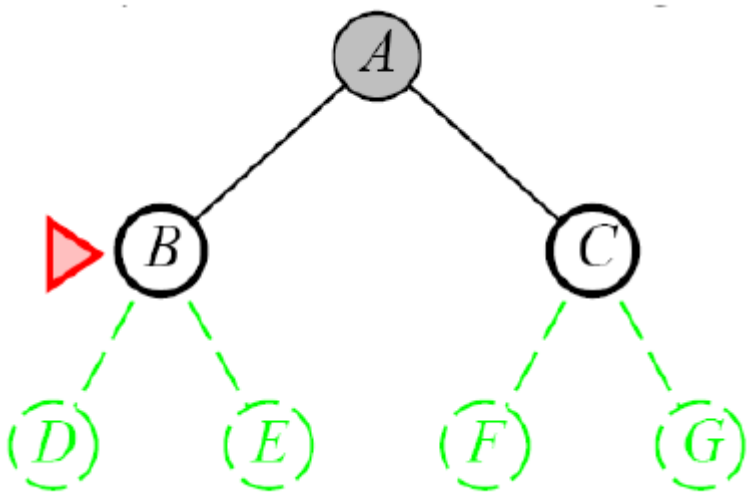
Пошук вшир

- Периферія – FIFO черга



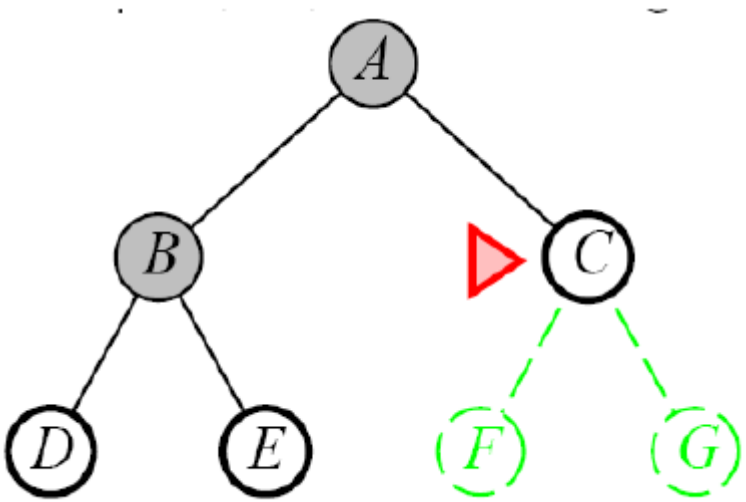
Пошук вшир

- Периферія – FIFO черга



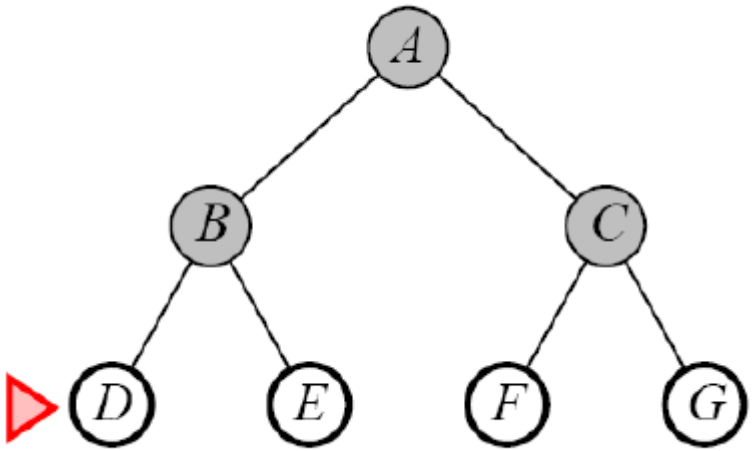
Пошук вшир

- Периферія – FIFO черга



Пошук вшир

- Периферія – FIFO черга



Пошук вшир

- Властивості
  - Повний? Так, якщо b скінченне
  - Час? $1 + b + b^2 + \ldots + b^d + b(b^d-1) = O(b^{d+1})$
  - Простір? $O(b^{d+1})$ (зберігає всі вузли у пам’яті)
  - Оптимальний? Так, якщо вартість етапу додатна

Пошук вшир

- b = 10
- швидкість формування вузлів – 10 000 вузлів/секунда
- об’єм пам’яті – 1Кб/вузол

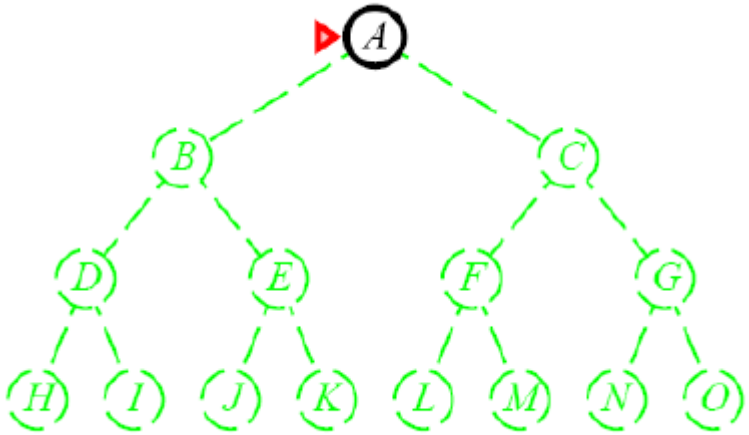
Глибина	Кількість вузлів	Час	Пам’ять
2	1100	0,11 секунди	1 Мб
4	111 100	11 секунд	106 Мб
6	10^7	19 мінут	10 Гб
8	10^9	31 година	1 Тб
10	10^{11}	129 діб	101 Тб
12	10^{13}	35 років	10 Пб
14	10^{15}	3523 роки	1 Еб

Пошук за критерієм вартості

- Периферія – черга, впорядкована за вартістю шляхів
  - Еквівалентний пошуку вшир, якщо вартості етапів рівні
- Властивості
  - Повний? Так, якщо вартість $\geq \varepsilon$
  - Час? $O(b^{\lceil C/\varepsilon \rceil})$, де C – вартість оптимального рішення
  - Простір? $O(b^{\lceil C/\varepsilon \rceil})$
  - Оптимальний? Так – вузли розкриваються із збільшенням $g(n)$

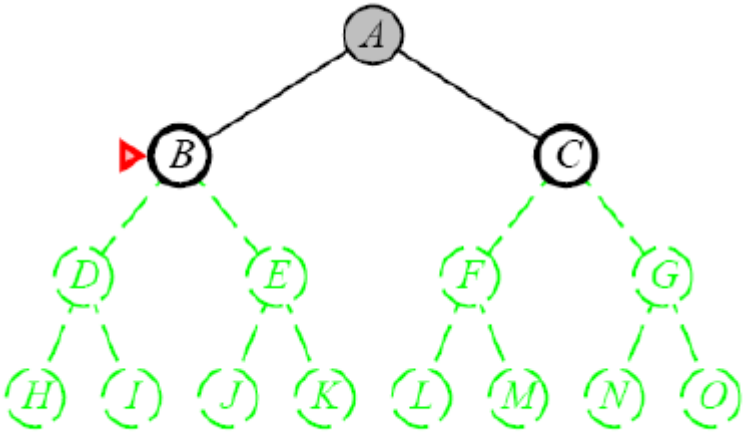
Пошук вглиб

- Периферія – LIFO черга



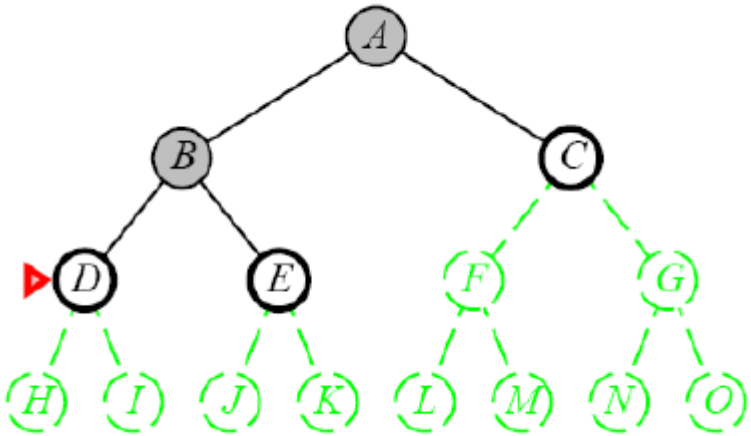
Пошук вглиб

- Периферія – LIFO черга



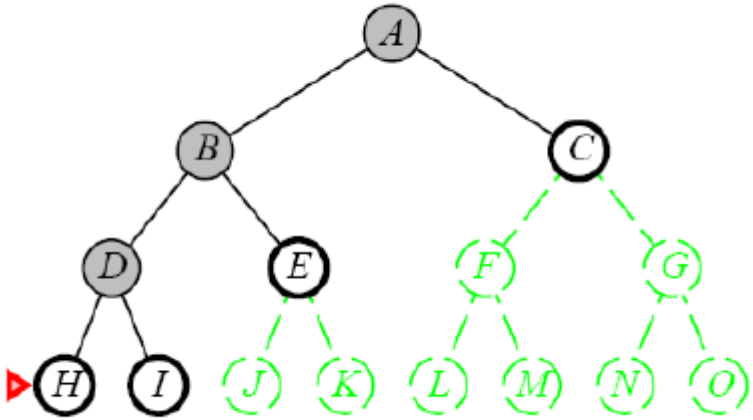
Пошук вглиб

- Периферія – LIFO черга



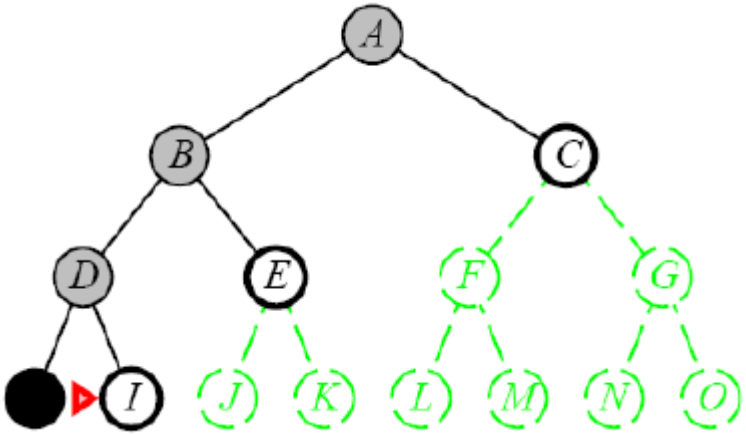
Пошук вглиб

- Периферія – LIFO черга



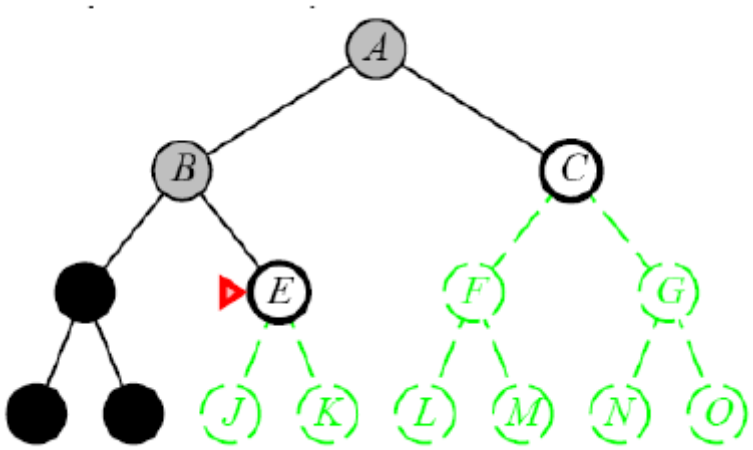
Пошук вглиб

- Периферія – LIFO черга



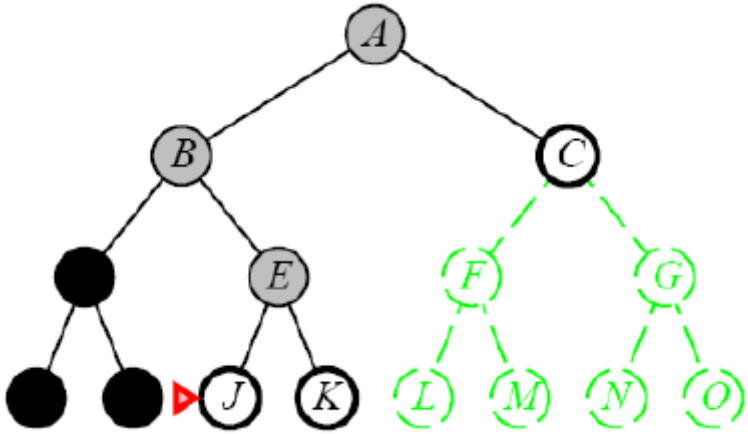
Пошук вглиб

- Периферія – LIFO черга



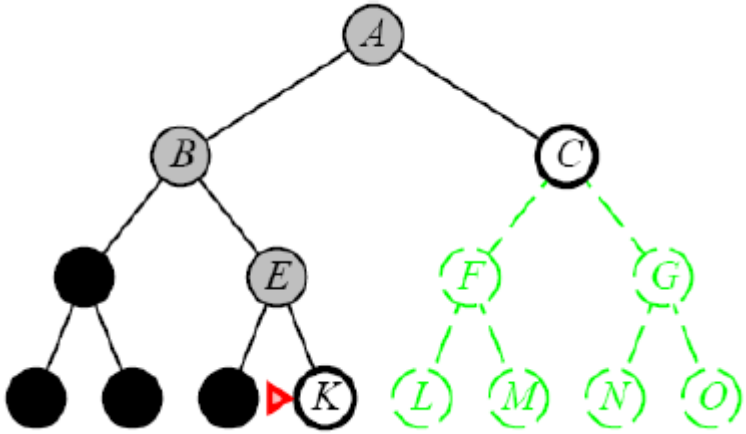
Пошук вглиб

- Периферія – LIFO черга



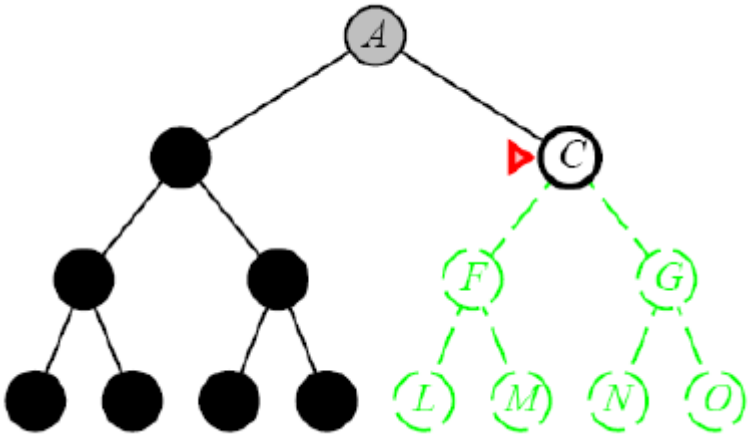
Пошук вглиб

- Периферія – LIFO черга



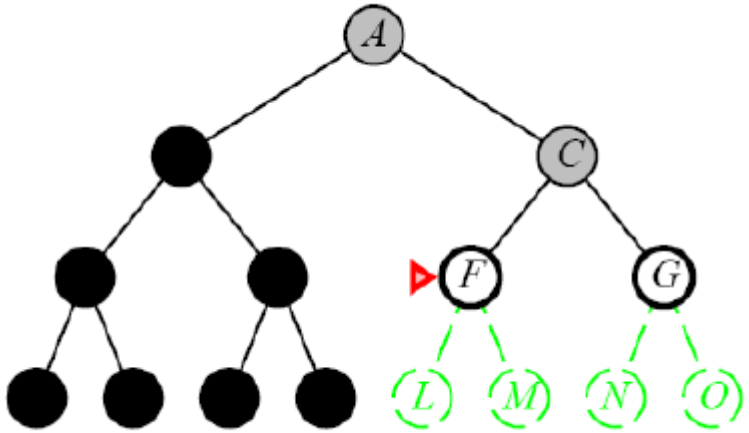
Пошук вглиб

- Периферія – LIFO черга



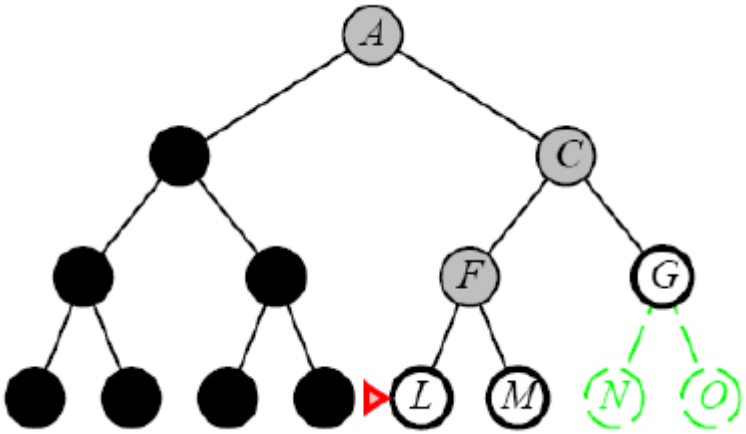
Пошук вглиб

- Периферія – LIFO черга



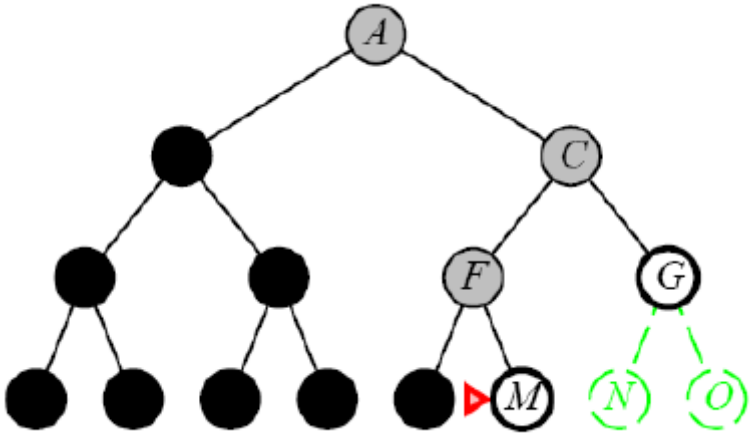
Пошук вглиб

- Периферія – LIFO черга



Пошук вглиб

- Периферія – LIFO черга



Пошук вглиб

- Властивості
  - Повний? Ні: потрапляє до нескінченних просторів, просторів із циклами
  - Час? $O(b^m)$, дуже погано при $m>>d$
    Якщо рішення розташовані щільно, то працює набагато швидше пошуку вшир
  - Простір? $O(bm)$ – лінійний простір!
  - Оптимальний? Ні

Пошук з обмеженням глибини

- = пошук вглиб з обмеженням глибини l

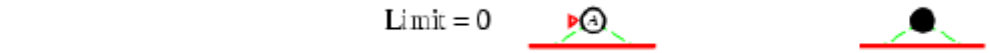
  тобто вузли на глибині l не мають нащадків

- Властивості
  - Повний? Ні, якщо $l < d$
  - Час? $O(b^l)$
  - Простір? $O(bl)$
  - Оптимальний? Ні, якщо $l > d$

Пошук вглиб з ітеративним заглибленням

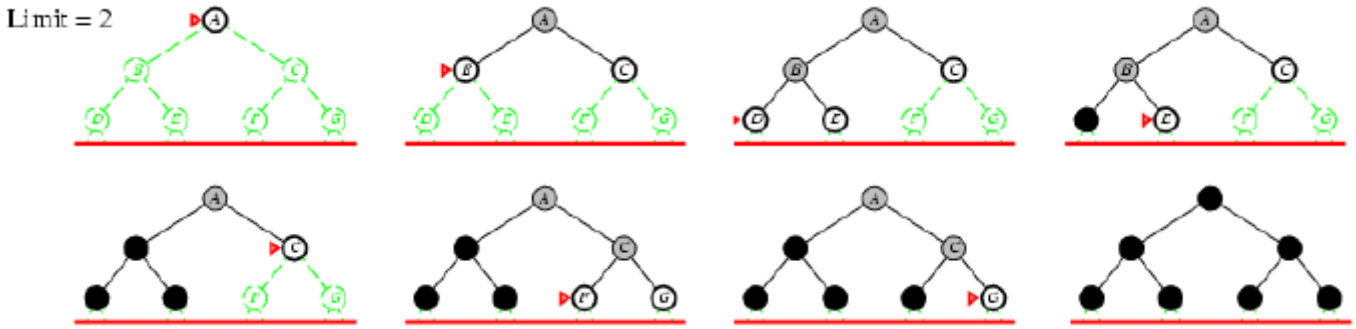
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

Пошук вглиб з ітеративним заглибленням, l =0

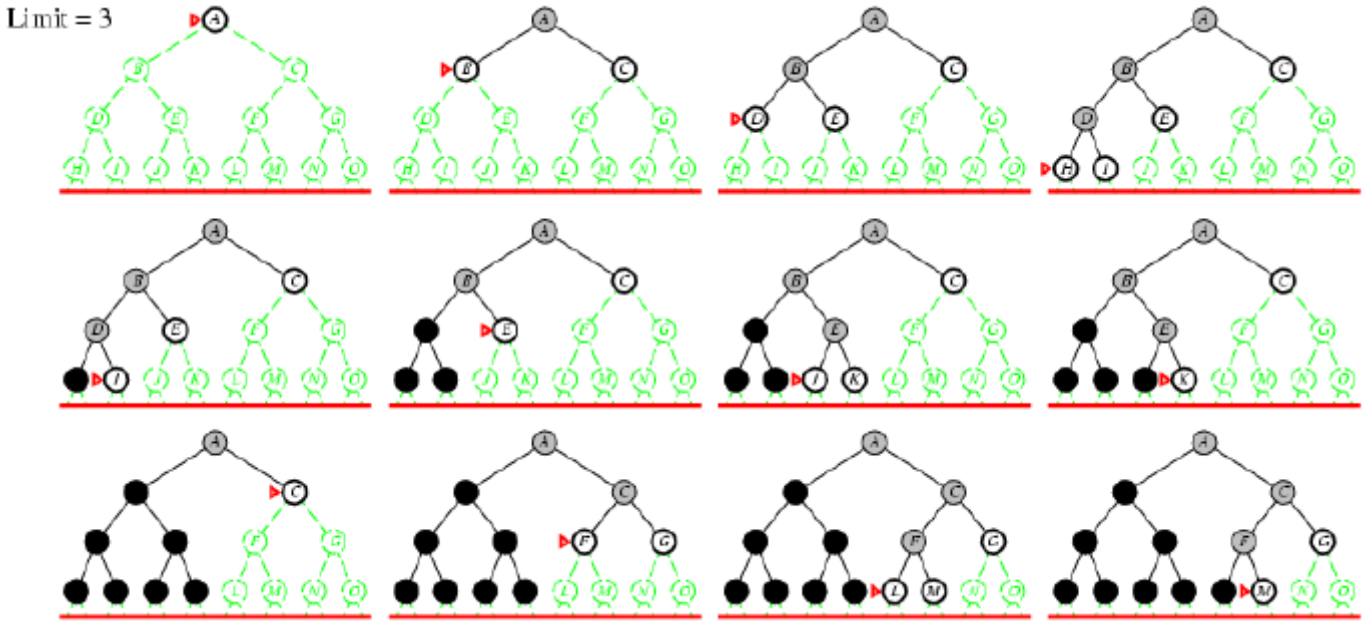


Пошук вглиб з ітеративним заглибленням, l =1

Пошук вглиб з ітеративним заглибленням, l =2



Пошук вглиб з ітеративним заглибленням, l =3



Пошук вглиб з ітеративним заглибленням

- Властивості
  - Повний? Так
  - Час? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
  - Простір? $O(bd)$
  - Оптимальний? Так, якщо вартість етапу = 1

- Порівняння для $b=10$ та $d=5$, рішення знаходиться у дальньому правому листку

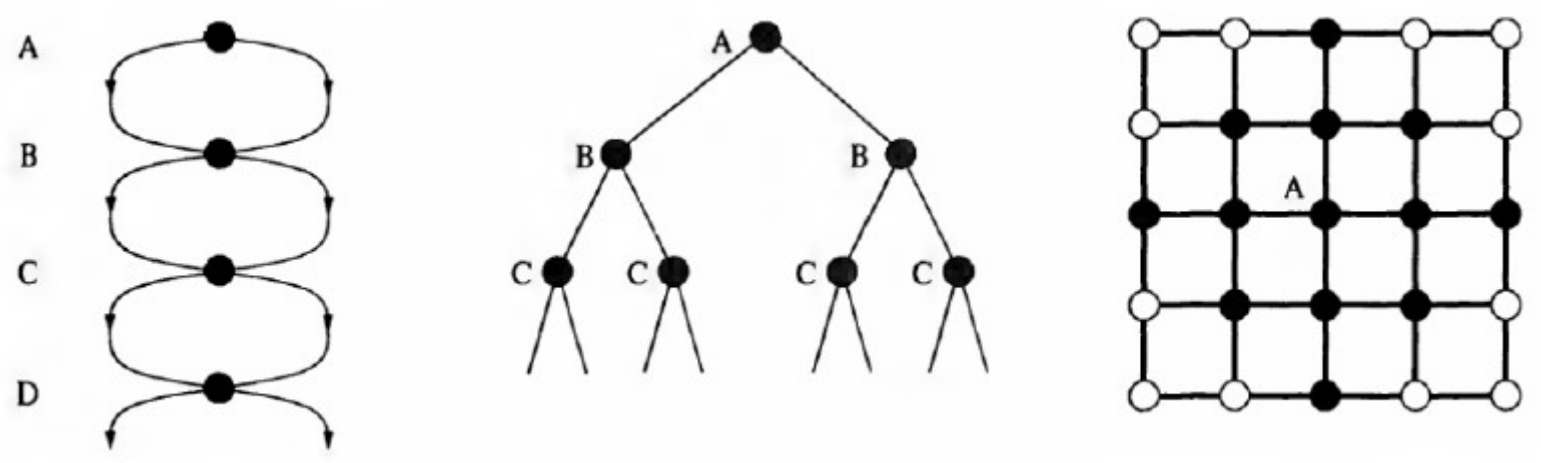
  $N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123\ 450$

  $N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1\ 111\ 100$

Порівняння алгоритмів

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Повторювальні стани



- Неможливість алгоритму визначати повторювальні стани може перетворити лінійну задачу у експоненціальну
- Алгоритми, які забувають свою історію, приречені на те, щоб її повторювати!!

Пошук в графі

- Якщо алгоритм запам’ятовує всі стани, які він відвідав, то він може розглядатись як такий, що безпосередньо досліджує граф простору станів

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

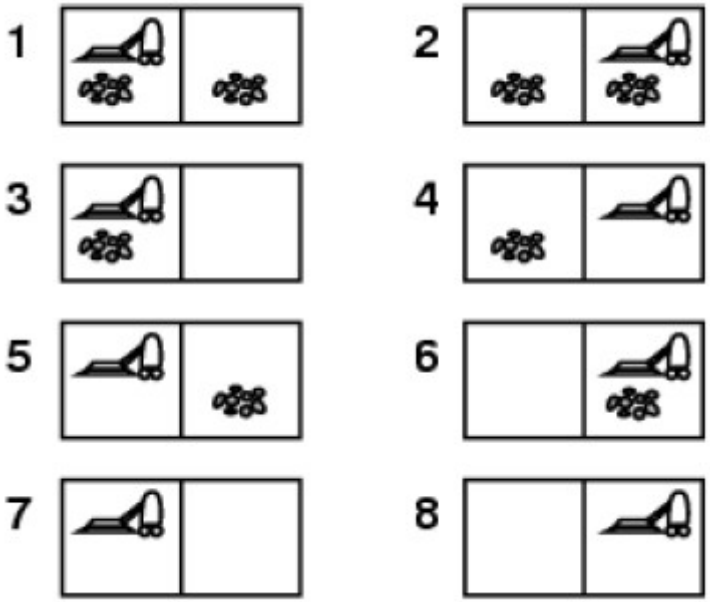
Пошук з частковою інформацією

- Простір детермінований, спостерігається повністю
  - Агент повністю знає, в який стан він перейде
- Проблема відсутності сенсорів → проблема сумісності
  - Агент може не знати де він
- Недетерміновані та/або спостерігаються частково → проблема непередбачуваних ситуацій
  - Сприйняття кожного разу надає нову інформацію про поточний стан
- Стан та дії невідомі → проблема дослідження

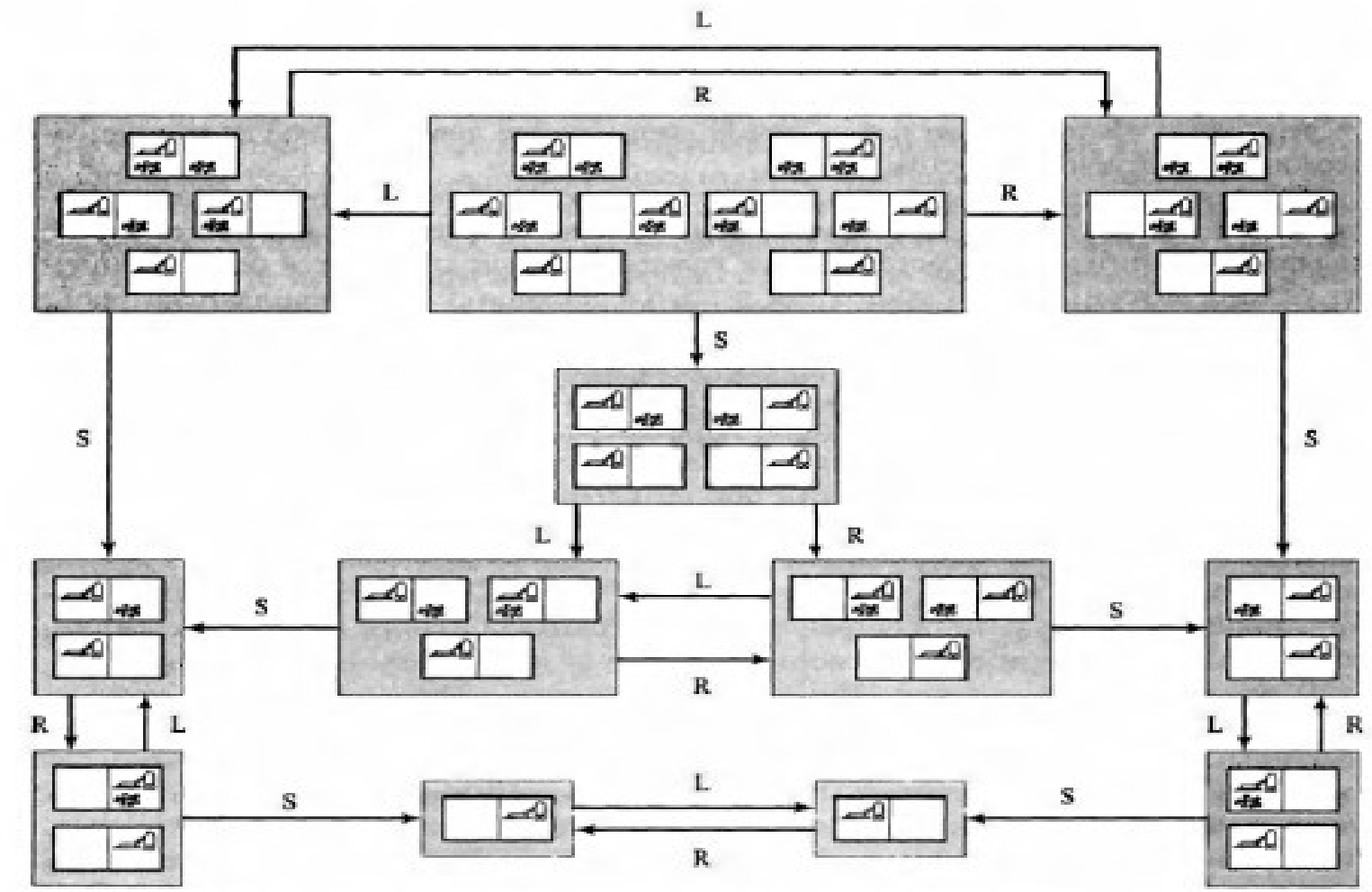
Приклад: світ пилососу

- Детермінована проблема
  початок у №5
  Рішення: [Right, Suck]

- Проблема сумісності
  початок у {1,2,3,4,5,6,7,8};
  Right переводить у {2,4,6,8}



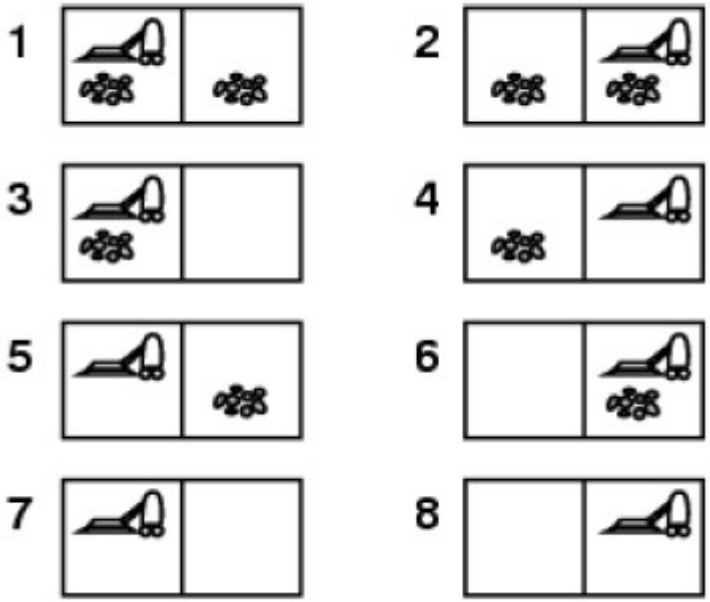
Приклад: світ пилососу



Приклад: світ пилососу

- Детермінована проблема
  початок у №5
  Рішення: [Right, Suck]

- Проблема сумісності
  початок у {1,2,3,4,5,6,7,8};
  Right переводить у {2,4,6,8}
  Рішення: [Right,Suck,Left,Suck]



- Непередбачуваність
  Закон Мерфі: дія Suck може забруднити чистий килим
  початок у №5
  Рішення: [Right, if dirt then Suck]

Література

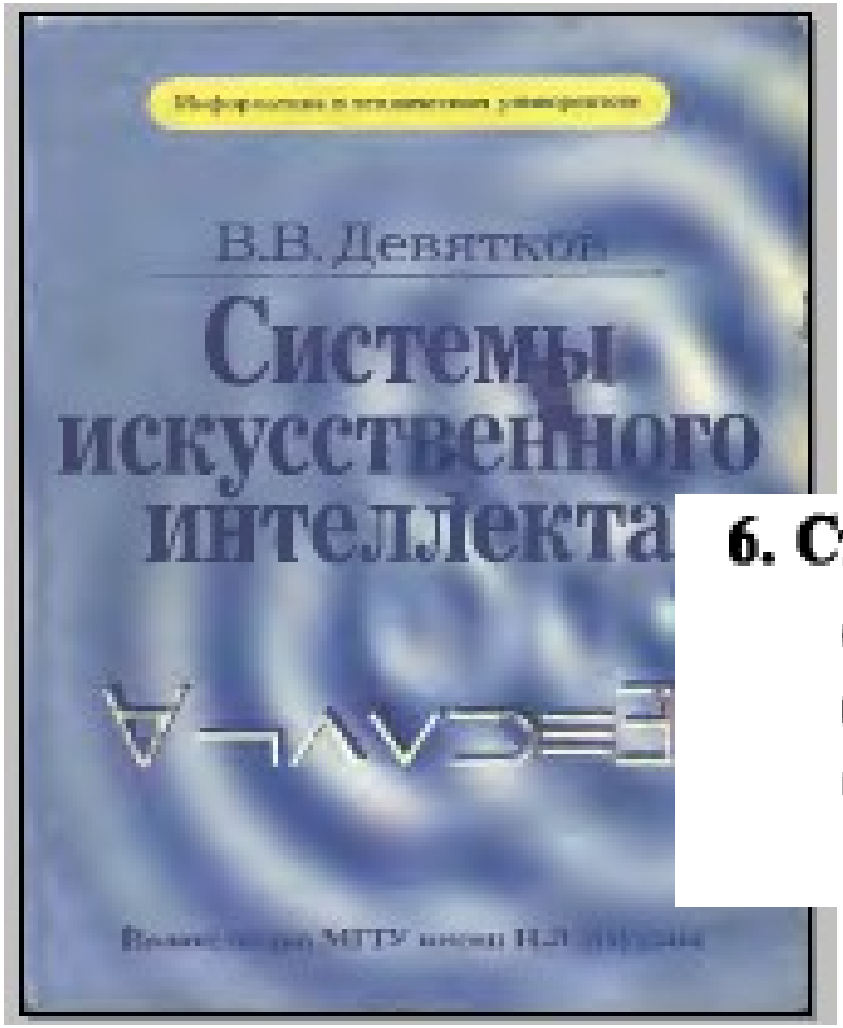
ЧАСТЬ II. РЕШЕНИЕ ПРОБЛЕМ

ГЛАВА 3. РЕШЕНИЕ ПРОБЛЕМ ПОСРЕДСТВОМ ПОИСКА

3.1. Агенты, решающие задачи
Хорошо структурированные задачи и решения
Формулировка задачи
3.2. Примеры задач
Упрощенные задачи
Реальные задачи
3.3. Поиск решений
Измерение производительности решения задачи
3.4. Стратегии неинформированного поиска
Поиск в ширину

Поиск в глубину
Поиск с ограничением глубины
Поиск в глубину с итеративным углублением
Двунаправленный поиск
Сравнение стратегий неинформированного поиска

Література



Література

Смолин Д. В. **Введение в искусственный интеллект: конспект лекций.** — М.: ФИЗМАТЛИТ, 2004. — 208 с. — ISBN 5-9221-0513-2.

В работе представлены базовые модели современного искусственного интеллекта, теоретические обоснования и практически полезные примеры построения разумных систем. Изложен авторский взгляд на основные достижения и пути дальнейшего развития программ с искусственным интеллектом. Рассмотрены практические аспекты применения интеллектуальных систем в предметных областях. Работа отличается простотой изложения — многие формулы дополнены или заменены словесным описанием, что, по мысли автора, должно послужить глубокому пониманию материала.

Для студентов информационных специальностей, аспирантов и специалистов в области применения современных информационных технологий.