МЕТОДИ І СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

3 курс, весна 2021

- Доц. Баклан І.В.
- Email: iaa@ukr.net
- Web: baklaniv.at.ua

Лекція 5

Інтелектуальні агенти для розв’язання проблем

Рефлекторні агенти

Простий рефлекторний засіб - це той, який вибирає дію лише на основі поточного сприйняття.

Він ігнорує решту історії сприйняття.

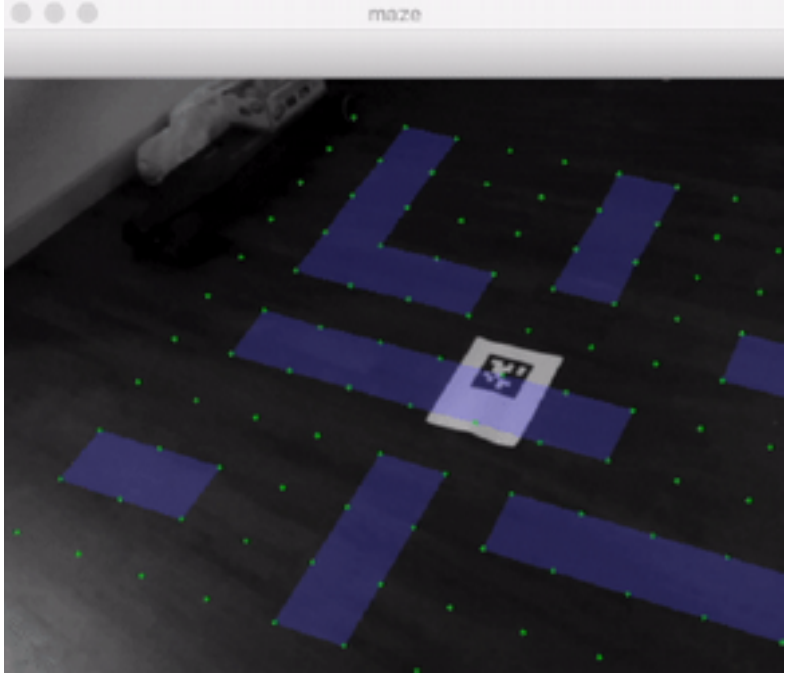
Агент для вирішення проблем

Агент з вирішення проблем повинен **планувати заздалегідь**.

Процес обчислення, який він здійснює, називається **пошуком**.

Він розгляне **послідовність дій**, що формують шлях до **стану цілі**.

Така послідовність називається **рішенням**.



Вплив середовищ завдань

Властивості середовищ завдань змінюють потрібні нам типи рішень.

Якщо середовище:

- **Повністю спостерігається**
- **Детермінований**
- **Відоме середовище**

Рішенням будь-якої проблеми в такому середовищі є фіксована послідовність дій.

У середовищах, які

- **Частково спостерігаються** або

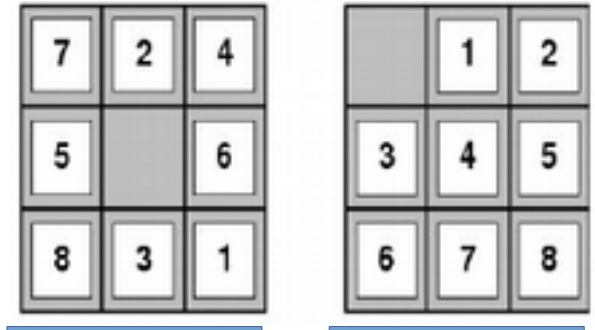
- **Недетерміновані**

Рішення повинно рекомендувати різні майбутні дії залежно від того, яке сприйняття воно отримує. Це може бути у формі *стратегії розгалуження*.

Приклад завдання пошуку: 8-мяшка

Сформулюємо **мету**

• Шматочки розмістити в порядку, як показано ...



Сформулюємо **пошукову задачу**

• **Стани**: конфігурації головоломки (9! Конфігурацій)

• **Дії**: перемістіть одну з рухомих частин (≤4 можливо)

• **Показник ефективності**: мінімізувати загальну кількість ходів

Знайдіть **рішення**

• Послідовність переміщень: 3,1,6,3,1, ...

Приклад проблеми пошуку: Відпустка в Румунії

Відпочинок в Румунії

На відпочинку в Румунії; в даний час в Араді
• Рейс відправляється завтра з Бухареста

Сформулюємо **мету**
• Бути в Бухаресті

Сформулюємо **пошукову задачу**
• Стани: різні міста
• Дії: їздити між містами
• Показник ефективності: мінімізуйте час / відстань у дорозі

Знайдемо **рішення**
• Послідовність міст; напр. Арад, Сібіу, Фагарас, Бухарест, ...

Більш формально проблема визначається:

1. *States*: a set S
2. An *initial state* $s_i \in S$
3. *Actions*: *a set* A

   $\forall s$ $Actions(s)$ = *the set of actions that can be executed in* s, *that are applicable* in s.
4. *Transition Model*: $\forall s \forall a \in Actions(s)$ $Result(s, a) \rightarrow s_r$

   s_r *is called a successor of* s

   $\{s_i\} \cup Successors(s_i)^*$ = *state space*
5. *Path cost (Performance Measure)*: Must be additive

   e.g. sum of distances, number of actions executed, ...

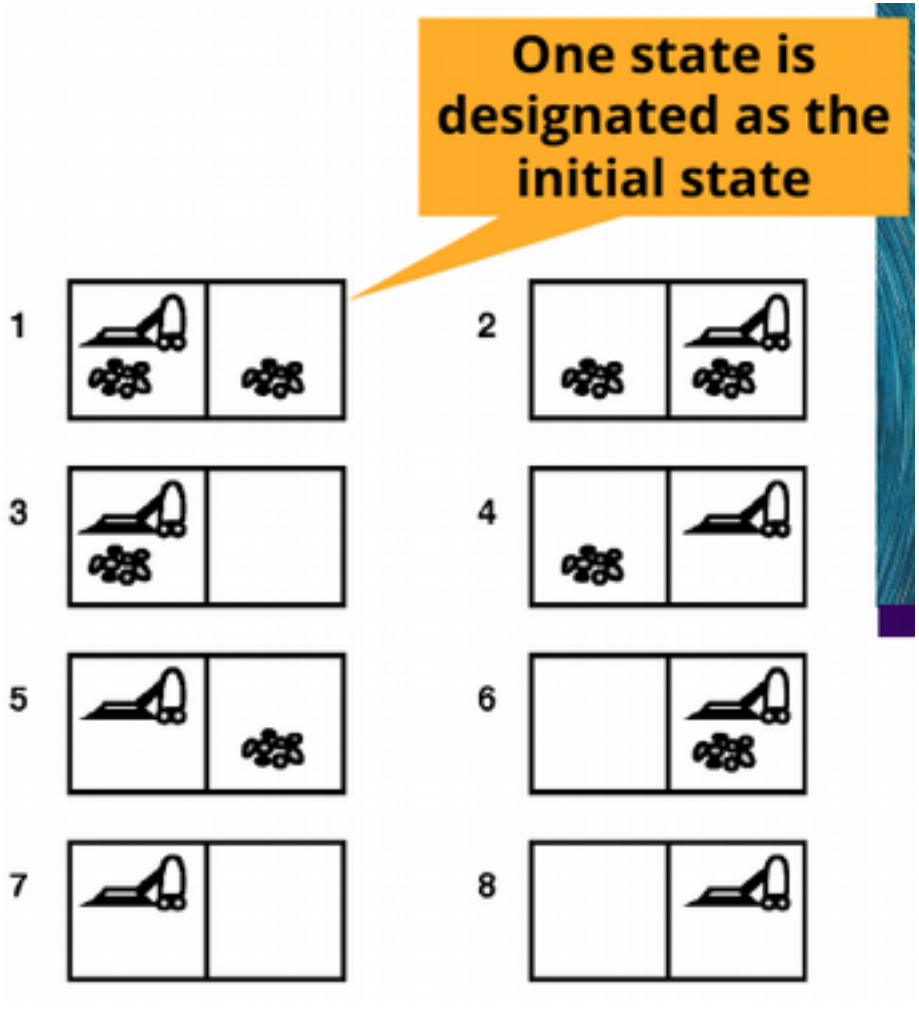
   $c(x,a,y)$ is the step cost, assumed ≥ 0
   - (where action a goes from state x to state y)
6. *Goal test*: $Goal(s)$

   Can be implicit, e.g. $checkmate(s)$

   s is a *goal state* if $Goal(s)$ is *true*

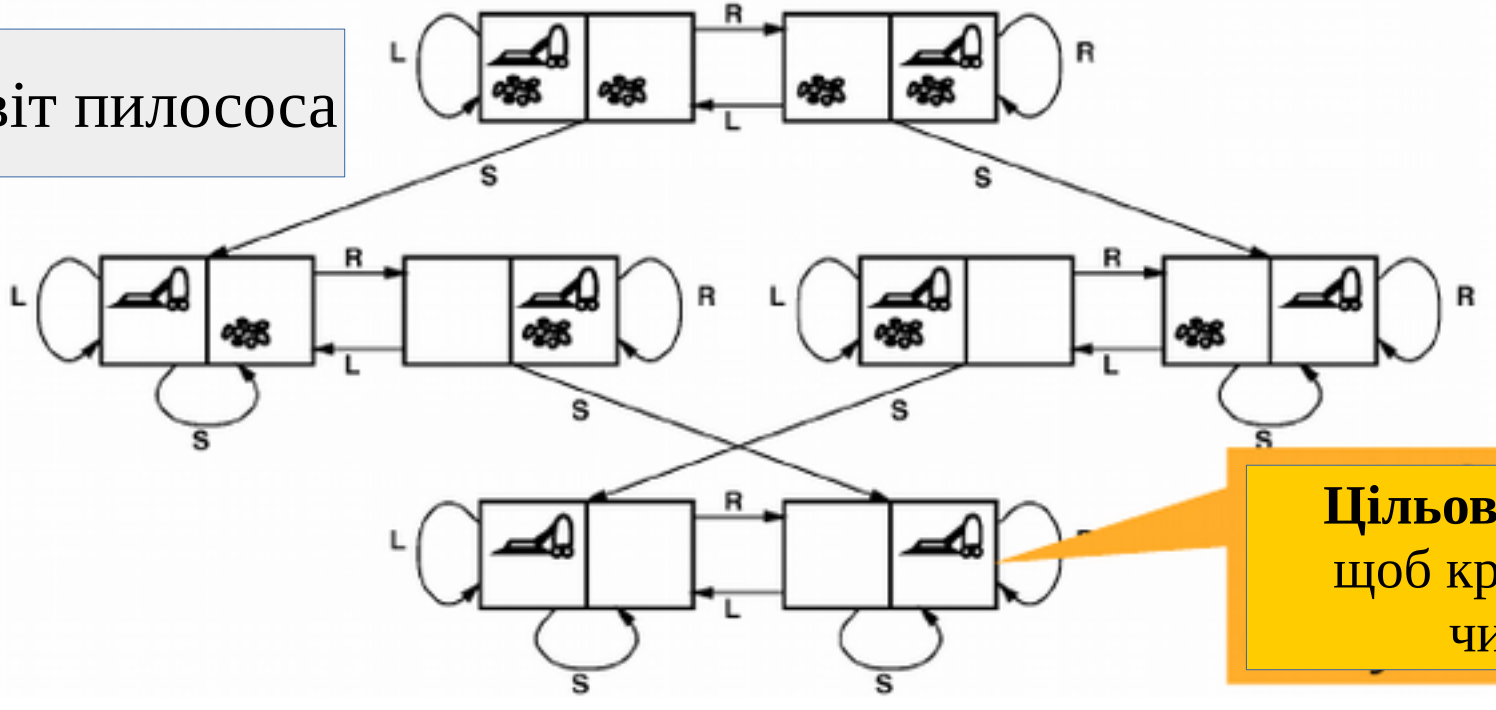
Світ пилососа

Стани: Стани світу пилососа говорять, які об’єкти знаходяться в яких клітинах.
У простій двоячейковій версії,
• агент може знаходитися в будь-якій клітині
• кожна клітина може мати бруд чи ні
2 клітини * 2 позиції для агента * 2 можливості для бруду = 8 станів.
З `n` клітинами існує `n*2n` станів.



Світ пилососа

Світ пилососа



Цільовий стан:
щоб крізь було чисто

Дії (кожна коштує 1):

- Смоктати
- Переміщення вліво
- Рухатися вправо
- Рухатися вгору
- Рухатися вниз

Перехід:

Смоктати - видаляє бруд

Переміщення - рухається в цьому напрямку, якщо тільки агент не вдаряється об стіну, і в цьому випадку він залишається на місці.

Рішення та оптимальні рішення

• **Рішення** - це послідовність **дій** від **початкового стану** до **стану цілі**.

• **Оптимальне рішення**: Рішення є **оптимальним**, якщо жодне рішення не має нижчої **вартості шляху**.

Мистецтво: Формулювання пошукової задачі

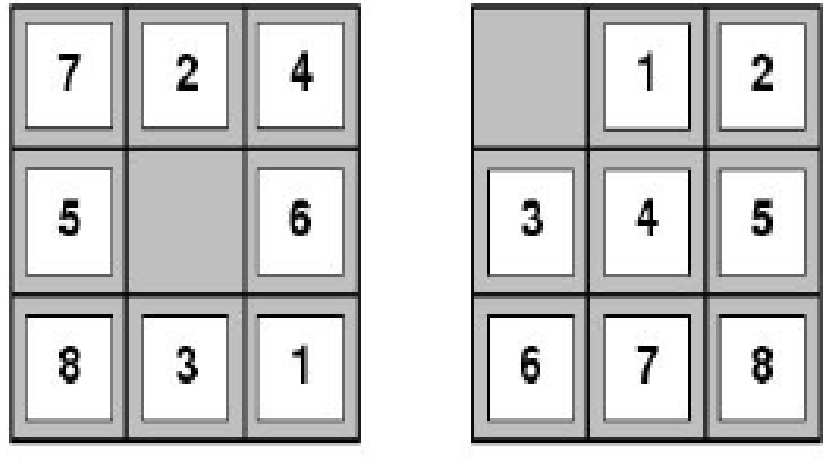
Вирішити:
Які властивості мають значення та як їх представити
- **Початковий стан, стан цілі, можливі проміжні стани**

Які дії можливі та як їх представити
- **Набір операторів: дії та модель переходу**

Яка дія наступна
- **Функція вартості шляху**

Формулювання сильно впливає на комбінаторику простору пошуку, а отже і на швидкість пошуку

Example: 8-puzzle



Start State

Goal State

States?

Initial state?

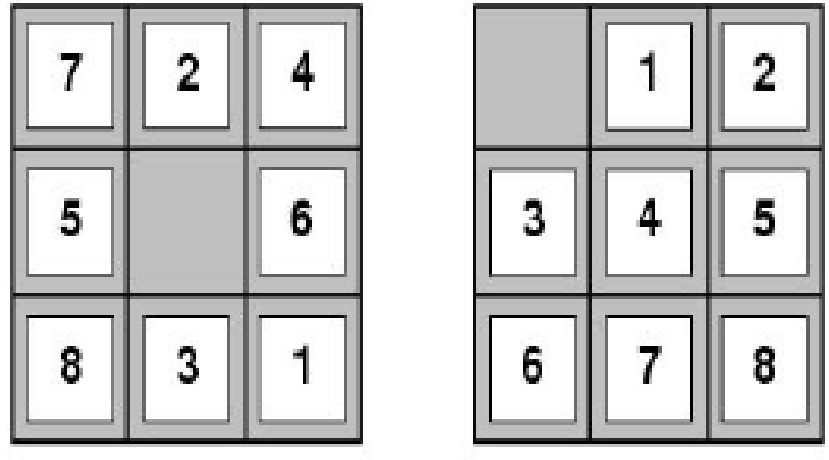
Actions?

Transition Model?

Goal test?

Path cost?

Example: 8-puzzle



States? List of 9 locations- e.g., [7,2,4,5,-,6,8,3,1]

Initial state? [7,2,4,5,-,6,8,3,1]

Actions? *{Left, Right, Up, Down}*

Transition Model? ...

Goal test? Check if goal configuration is reached

Path cost? Number of actions to reach goal

Важка підзадача: Вибір простору станів

Реальний світ абсурдно складний.

Для розв’язання проблем має бути **абстраговано** простір станів

(абстракція) **Стан** = сукупність (клас еквівалентності) реальних станів

(абстракція) Дія = клас еквівалентності поєднань дій у реальному світі:

• напр. *Arad до Zerind* представляє складний набір можливих маршрутів, об'їздів, зупинок відпочинку, тощо

• абстракція дійсна, якщо шлях між двома станами відображується у реальному світі

Кожна абстрактна дія повинна бути "легшою", ніж реальна проблема.

Корисні поняття

Простір станів: сукупність усіх станів, досяжних з початкового стану будь-якою послідовністю дій

- *Коли до кожного стану можуть подати заявки кілька операторів, це дуже швидко стає великим*
- *Може бути належним підмножиною набору конфігурацій*

Шлях: послідовність дій, що ведуть з одного стану s_j в інший стан s_k

Обмеження: ті стани, які доступні для *розширення* (для застосування юридичних дій до)

Рішення: шлях від початкового стану s_i до стану s_f, який задовольняє цільовому тесту

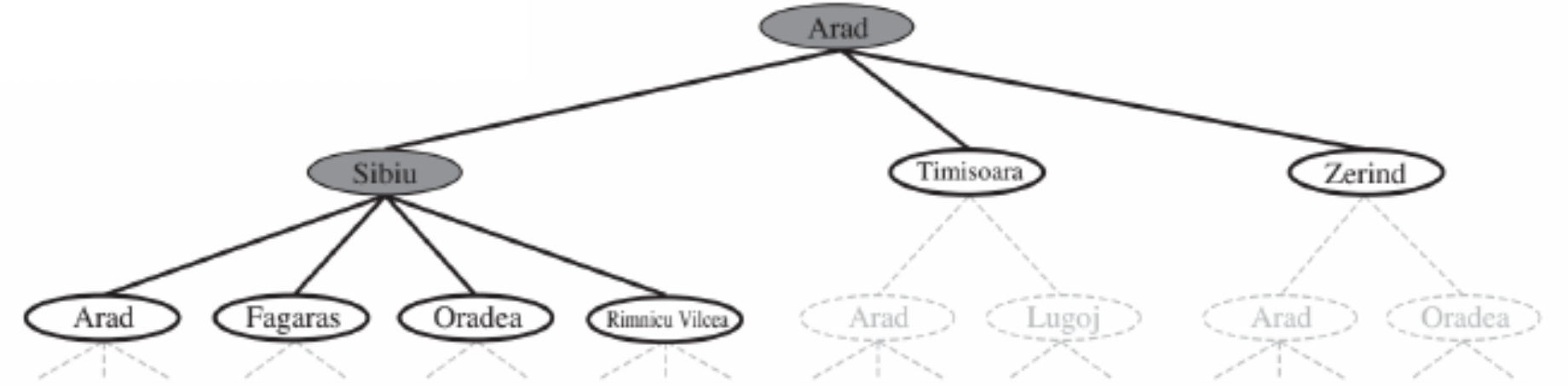
Основні алгоритми пошуку: пошук по дереву

Узагальнений алгоритм вирішення пошукових задач

Перелічити у певному порядку всі можливі шляхи від початкового стану

- Тут: пошук за допомогою *явного генерування дерева*
- ROOT = початковий стан.
- Вузли в дереві пошуку, створені за допомогою *моделі переходу*
- Пошук по дереву трактує різні шляхи до одного і того ж вузла як різні

Узагальнений пошук по дереву



функція ПОШУК ДЕРЕВА (*проблема, стратегія*) повернення рішення або помилки

Ініціалізуйте кордон до *початкового стану проблеми*

робити

якщо кордон порожній, поверніть *помилку*

вибрати листовий вузол для розширення відповідно до стратегії та зняти з кордону

якщо вузол містить стан цілі, тоді повертається *рішення*

в іншому випадку розширте вузол і додайте отримані вузли до кордону

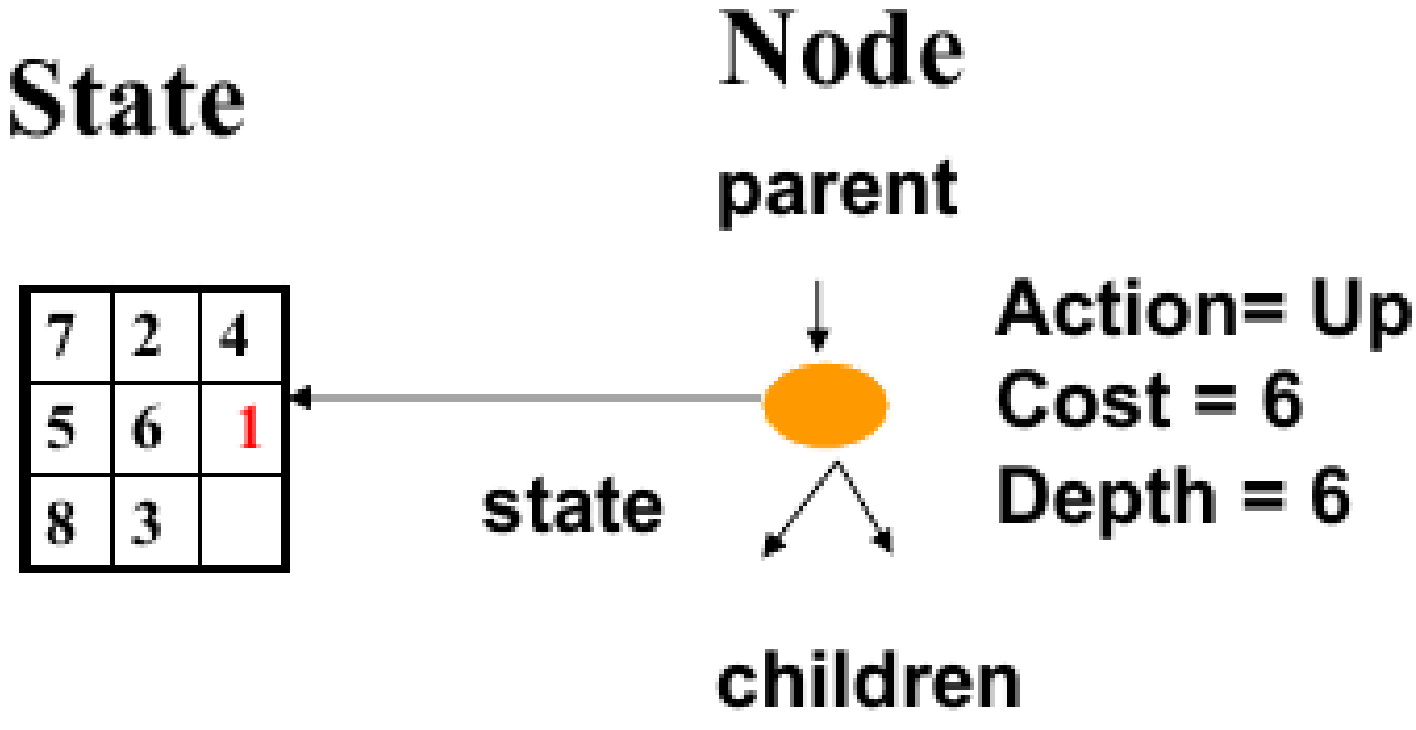
8-мяшка: СТАНИ і ВУЗЛИ

Стан - це (подання) ***фізична конфігурація***

Вузол - це структура даних, що становить ***частину дерева пошуку***

- Також включає *батьків, дітей, глибину, вартість шляху g (x)*
- Тут вузол = <*стан, батьківський вузол, діти, дія, вартість шляху, глибина*>

Стани не мають батьків, дітей, глибини або вартості шляху!

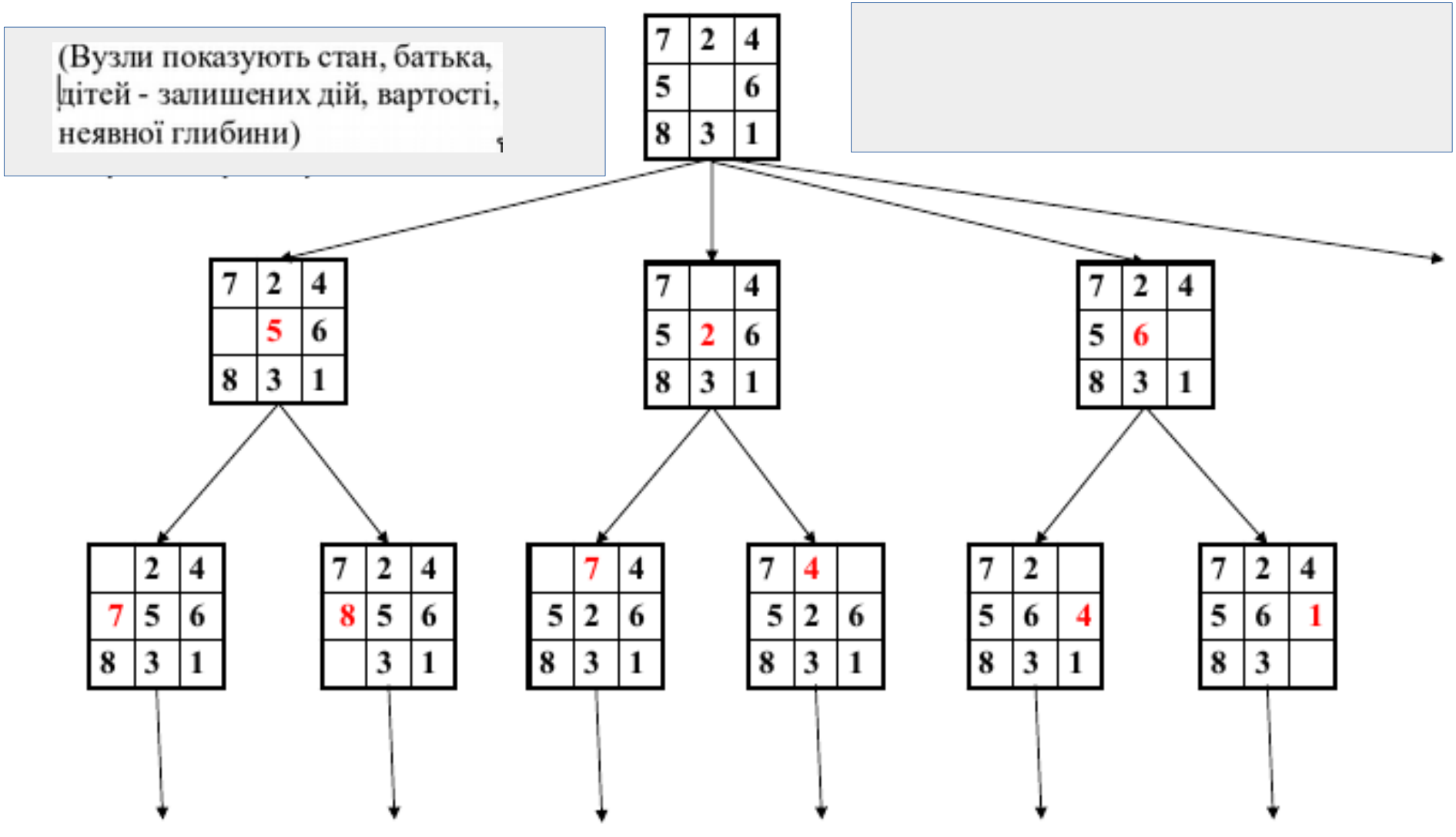


Функція **EXPAND**

- використовує модель дій та переходу для створення відповідних станів
- створює нові вузли,
- заповнює різні поля

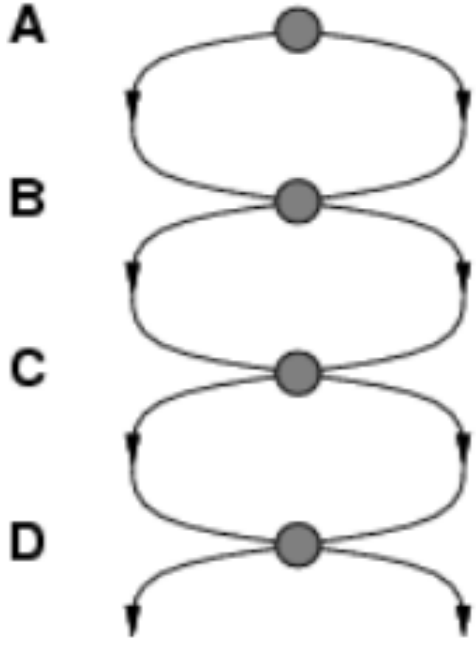
Дерево пошуку 8-мяшки

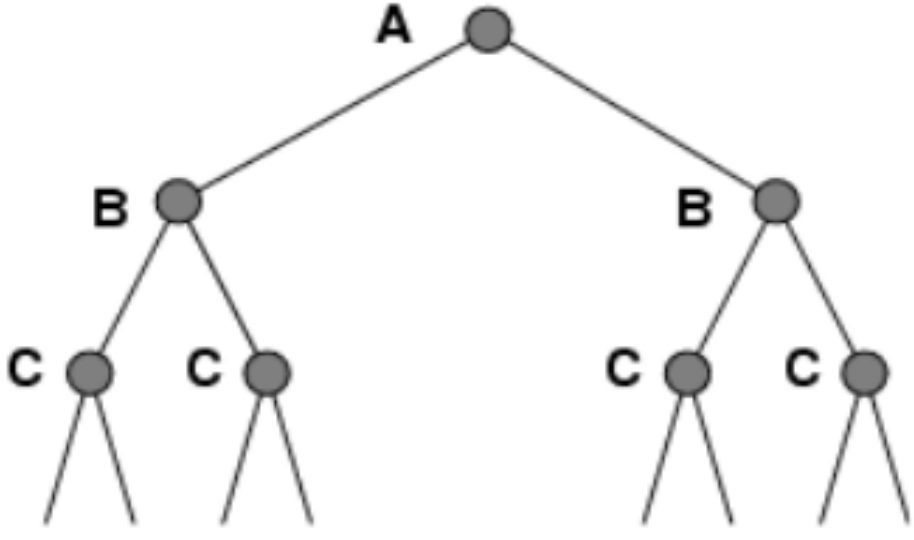
(Вузли показують стан, батька, дітей - залишених дій, вартості, неявної глибини)

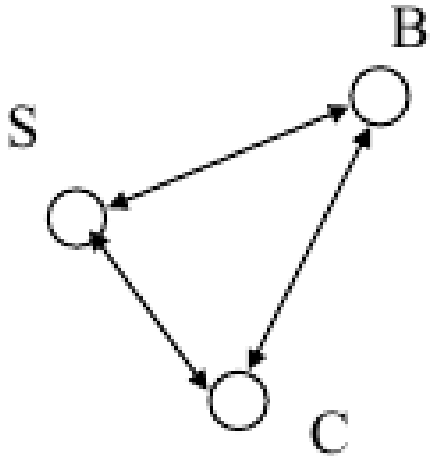


Проблема: повторні стани

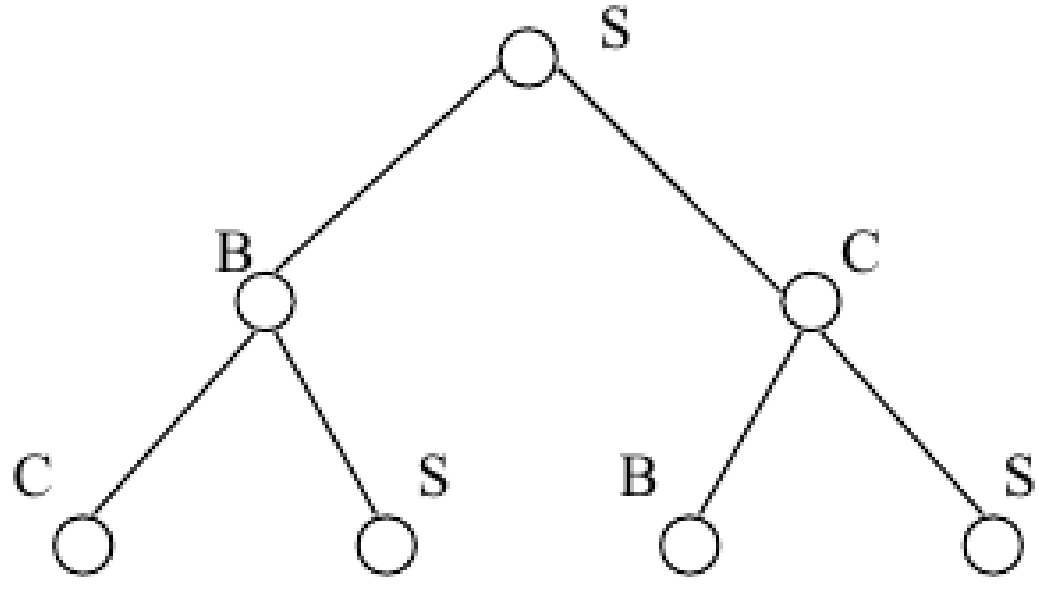
Неможливість виявити **повторювані стани** може перетворити лінійну задачу в **експоненційну**!







State Space



Search Tree

Рішення: граф пошуку

• простий хід із пошуку по дереву: перевірте, чи не було відвідано вузол перед додаванням до черги пошуку

• треба відстежувати всі можливі стани (це може використовувати багато пам'яті)

• наприклад, проблема з 8 головоломками, ми маємо 9! / 2 »182 тис. станів

Граф пошуку проти дерева пошуку

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Стратегії неінформованого пошуку

"Сліпий пошук"

Використовує лише інформацію, доступну у визначенні проблеми.
Неформально:

Неінформований пошук: усі нецільові вузли на кордоні виглядають однаково добре

Інформований пошук: Деякі нецільові вузли можна класифікувати вище інших.

Стратегії пошуку

Огляд: Стратегія = порядок розширення дерева
• Реалізовано різними структурами черг (LIFO, FIFO, пріоритет)
Розміри для оцінки
• **Повнота** - завжди знаходити рішення?
• **Оптимальність** - спочатку знаходить рішення із найменшими витратами (найнижча вартість шляху)?
• **Часова складність** - кількість генерованих вузлів (*найгірший випадок*)
• **Складність простору** - кількість вузлів одночасно в пам'яті (*найгірший випадок*)

Змінні складності часу / простору

- **`b`**, **максимальний коефіцієнт розгалуження** дерева пошуку
- **`d`**, **глибина** найдрібнішого вузла воріт
- **`m`**, максимальна довжина будь-якого шляху в просторі станів (потенційно — нескінченність)

Вступ до *космічної* складності

Ви знаєте про:

• Позначення “Велике О” - складність алгоритмів

• *Складність часу*

Складність простору аналогічна складності часу

Одиниці простору довільні

- Не має значення, оскільки позначення Big O ігнорує постійні мультиплікативні фактори
- Вірогідні космічні одиниці:
  - Одне слово пам'яті
  - Розмір будь-якої структури даних фіксованого розміру
    - Наприклад, розмір вузла фіксованого розміру в дереві пошуку

Огляд: пошук по ширині

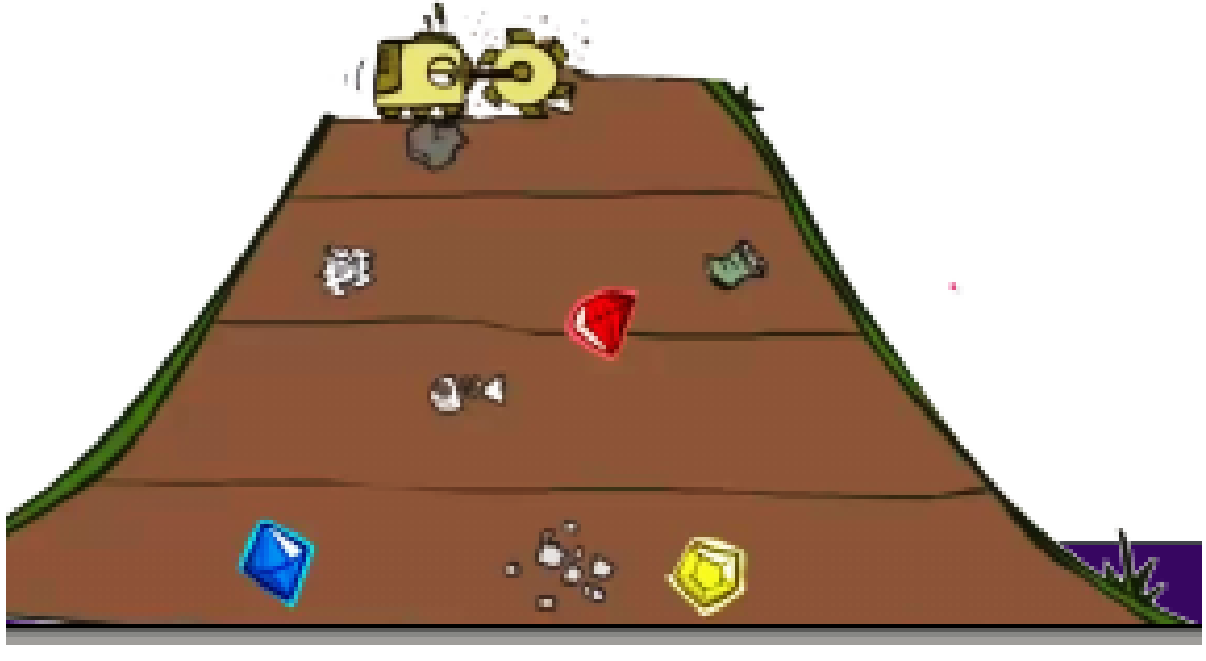
Ідея:

• Розширити ***найдрібніший*** нерозгорнутий вузол

Реалізація:

• *кордон* - це черга FIFO (First-In-First-Out):

• помістіть наступників в кінці списку наступників кордону.



Пошук по ширині (спрощений)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTIO
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then retu  failure
        node ← POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier th
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Position within queue of new items determines search strategy

Subtle: *Node inserted into queue only after testing to see if it is a goal state*

Властивості пошуку по ширині

Повне? Так (якщо b кінцевий)

Складність часу? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$

Складність простору? $O(b^d)$ (зберігає кожен вузол у пам'яті)

Оптимальна?

Так, якщо вартість = 1 за крок
(загалом не оптимально)

b: максимальний коефіцієнт розгалуження дерева пошуку

d: глибина рішення з найменшими витратами

m: максимальна глибина простору станів (нескінченність)

Експоненціальний простір (і час) не добрі ...

• Неінформовані пошукові завдання експоненціальної складності не можуть бути вирішені для будь-якого, крім найменших випадків.

• (Вимоги до пам'яті є більшою проблемою, ніж час виконання.)

DEPTH	NODES	TIME	MEMORY
2	110	0.11 milliseconds	107 kilobytes
4	11110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabytes
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabytes
14	10^{14}	3.5 years	99 petabytles

Assumes b=10, 1M nodes/sec, 1000 bytes/node

Огляд: пошук по глибині

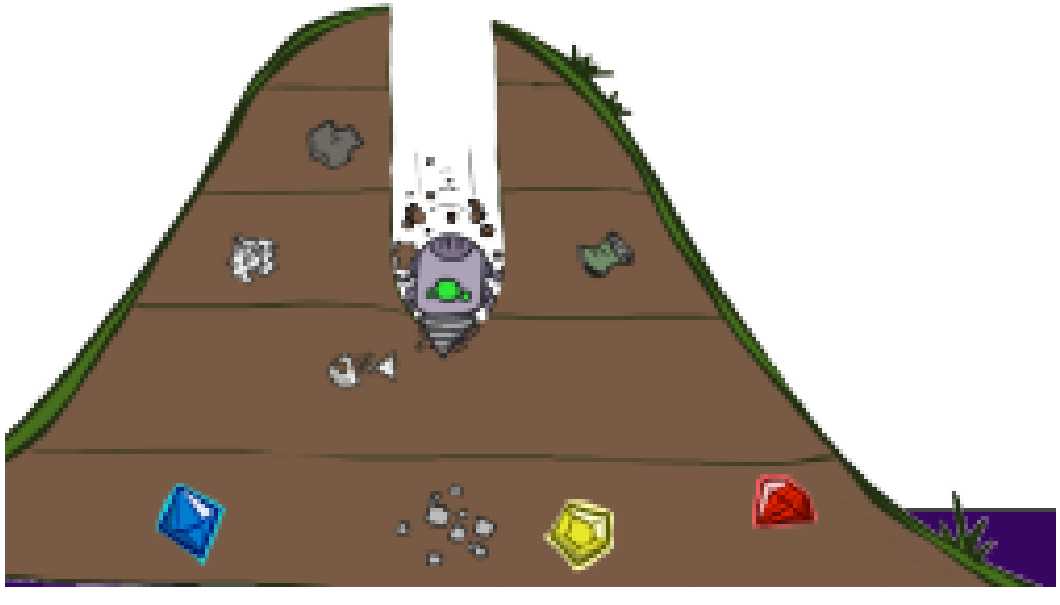
Ідея:

• Розширити *найглибший* нерозгорнутий вузол

Реалізація:

• *кордон* - це черга LIFO (Last-In-First-Out):

• Розмістіть наступників на початку списку наступників *кордону*.



Властивості глибинного пошуку

Повне?

Ні: не вдається в просторах з нескінченною глибиною, просторах із циклами

• Змінюйте, щоб уникнути повторення станів по шляху **à** завершені в скінченних просторах

Час? O ($\mathbf{b}^{\mathbf{m}}$): страшний, якщо **m** набагато більше **d**

• але якщо розчини щільні, це може бути набагато швидше, ніж ширина

Простір? O (**b * m**), тобто лінійний простір!

Оптимальна?

Ні

b: *максимальний коефіцієнт розгалуження дерева пошуку*

d: *глибина рішення з найменшими витратами*

m: *максимальна глибина простору станів (нескінченність)*

Спочатку в глибину проти спочатку в ширину

Використовуйте спочатку пошук у глибину, якщо
- *Простір обмежений*
- *Є багато можливих рішень із довгими та неправильними шляхами зазвичай швидко припиняються*
- *Пошук можна швидко налаштувати*

Використовуйте спочатку пошук у ширину, якщо
- *Можливі нескінченні шляхи*
- *Деякі рішення мають короткий шлях*
- *Можна швидко відкидати малоймовірні шляхи*

Наступна лекція буде присвячена інтелектуальним агентам для інформованого пошуку