Intelligenza Artificiale

Anno Accademico 2022 - 2023

Problem Solving e Search

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Agenti Risolutori di Problemi
- Tipi di problemi
- Formulazione di problemi
- Esempi di problemi

Agenti risolutori di problemi

Agente che ha un obiettivo da raggiungere
e che deve determinare una sequenza di azioni
che permetta di raggiungerlo

Primi passi da effettuare:

- Formulazione dell'obiettivo: insieme di stati del mondo (soltanto quegli stati in cui è soddisfatto l'obiettivo)
- Formulazione del problema: processo di decidere quali azioni e stati considerare

Esempio: raggiungere Bucarest

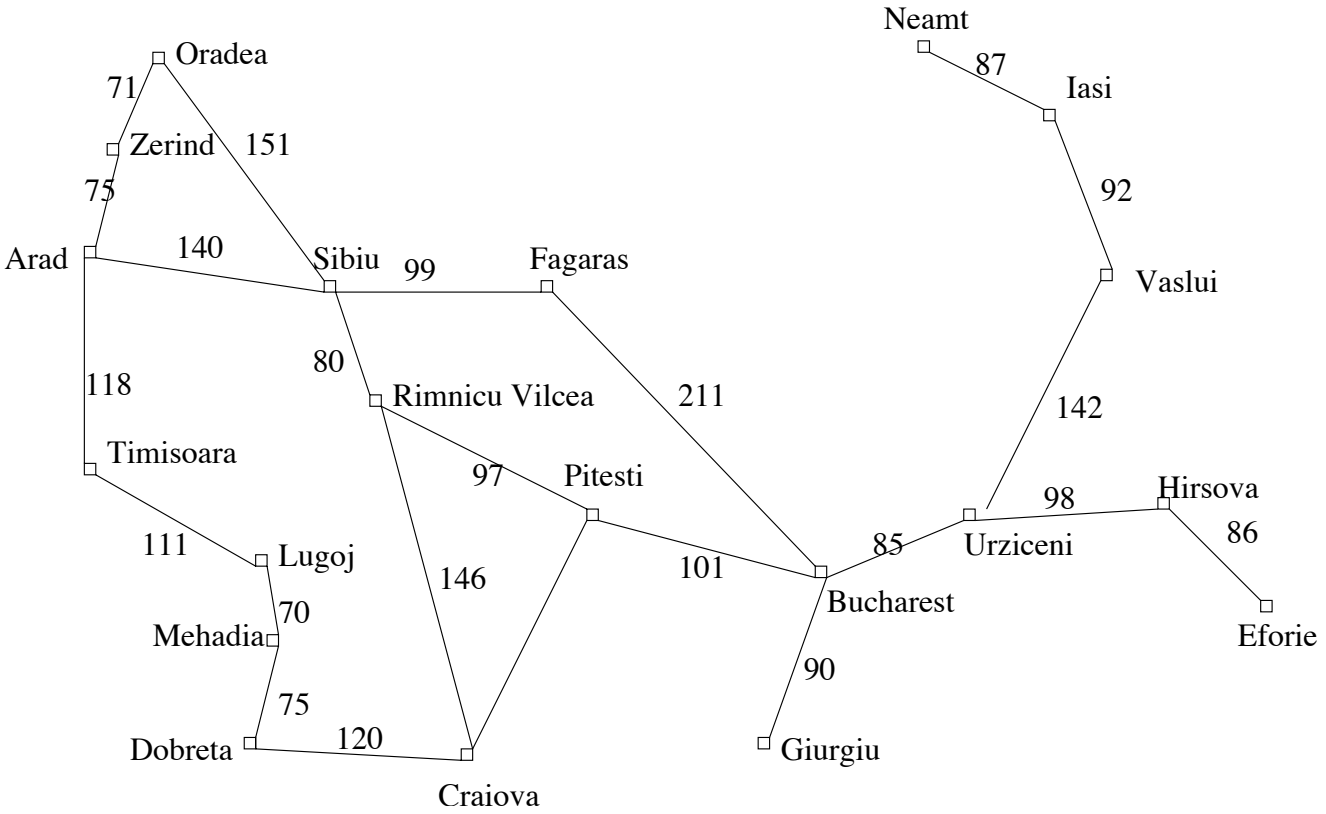
- In vacanza in Romania, attualmente in Arad
- Biglietto aereo per partire da Bucarest il giorno seguente
- Formulazione dell'obiettivo: guidare fino a Bucarest
- Formulazione del problema:
  - stati: le diverse città
  - azioni: guidare da una città all'altra
- Cercare la soluzione: sequenza di città, ad es.: Arad, Sibiu, Fagaras, Bucarest.

Esempio: raggiungere Bucarest

- L'agente ha adottato l'obiettivo di guidare fino a Bucarest, e sta valutando dove andare partendo da Arad.
- Ci sono tre strade possibili, che conducono a:
  - Sibiu
  - Timisoara
  - Zerind
- Nessuna di esse soddisfa l'obiettivo.
- L'agente non sa quale sia l'azione preferibile.

Esempio: raggiungere Bucarest

- Supponiamo che l'agente possieda una mappa della Romania:



Esempio: raggiungere Bucarest

- L'agente può usare la mappa per considerare i passi successivi di un viaggio ipotetico attraverso ciascuna delle tre città, cercando un cammino che lo conduca a Bucarest:

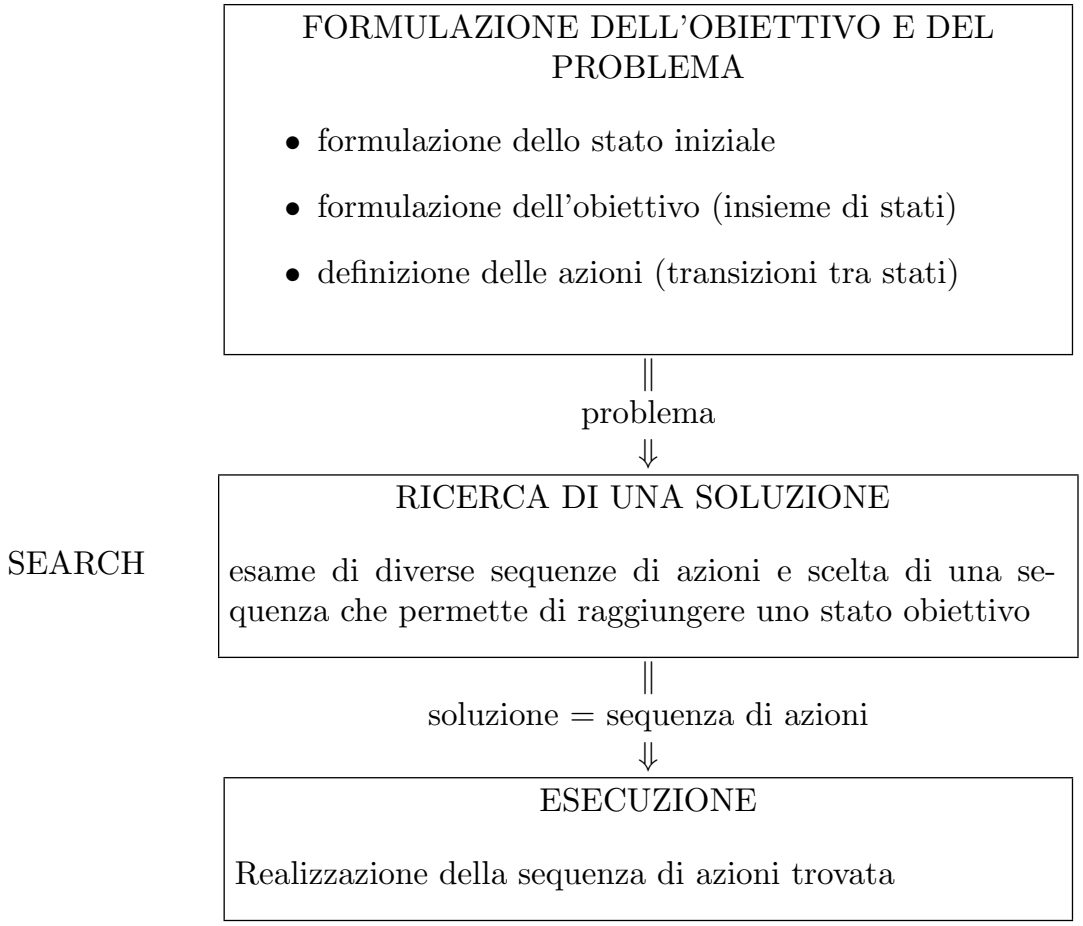
  Un agente che ha a disposizione diverse opzioni immediate di valore sconosciuto può decidere cosa fare esaminando diverse possibili sequenze di azioni che portano a stati di valore conosciuto, scegliendo quindi la sequenza migliore.

- Questo processo di selezione è detto **ricerca**.

- Un algoritmo di ricerca prende un problema come input e restituisce una **soluzione** costituita da una sequenza di azioni.

Agenti risolutori di problemi

Le tre fasi principali di un problem solving agent:



Esempi di problemi

C'è un'ampia gamma di ambienti relativi a compiti che possono essere caratterizzati da problemi ben definiti.

Possiamo distinguere tra:

- Problemi giocattolo (toy problems), che sono progettati come illustrazione o esercitazione di vari metodi di risoluzione di problemi.
- Problemi del mondo reale, che tendono ad essere più difficili e alle cui soluzioni le persone sono veramente interessate.

Esempi di problemi

Esempi di Problemi giocattolo:

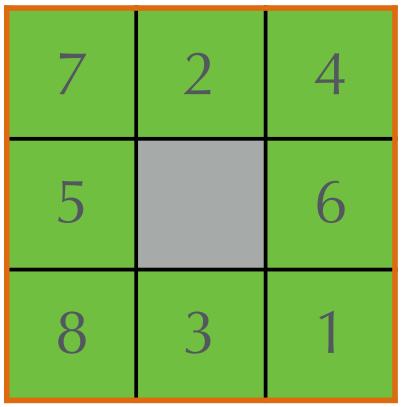
- Il rompicapo a 8 tasselli.
- Il problema delle 8 regine.
- Il mondo dell'aspirapolvere.

Esempi di problemi

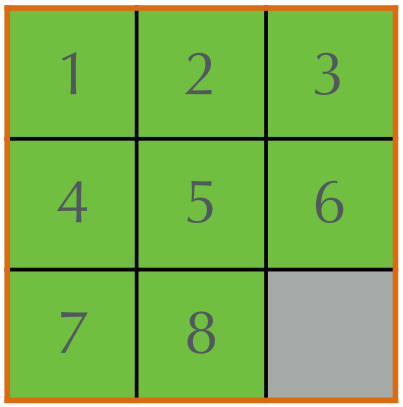
Esempi di Problemi del mondo reale:

- Configurazione VLSI.
- Navigazione di robot.
- Sequenza di montaggio.
- Ricerca di itinerario.
- Problemi di viaggio.
- Il problema del commesso viaggiatore.

Esempio di toy problem:
Il Rompicapo a 8 Tasselli

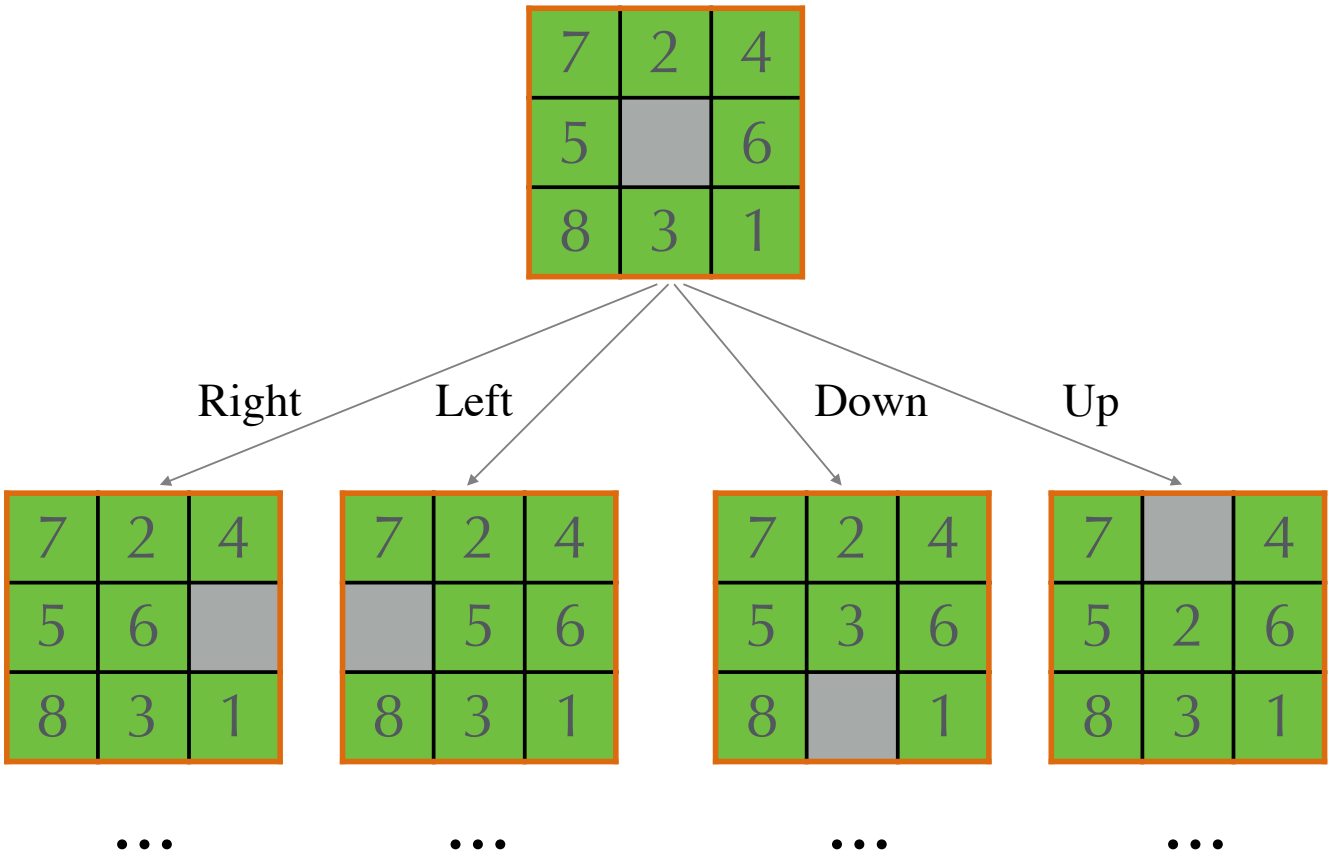


Stato Iniziale



Stato Obiettivo

Esempio di toy problem:
Il Rompicapo a 8 Tasselli



Esempio di toy problem:
Il Rompicapo a 8 Tasselli

- Stati: Posizione dei tasselli nella matrice.
- Stato Iniziale: Assegnato.
- Azioni: Left, Right, Up, Down.
- Goal Test: Assegnato.
- Costo del Cammino: Ogni mossa costa 1.

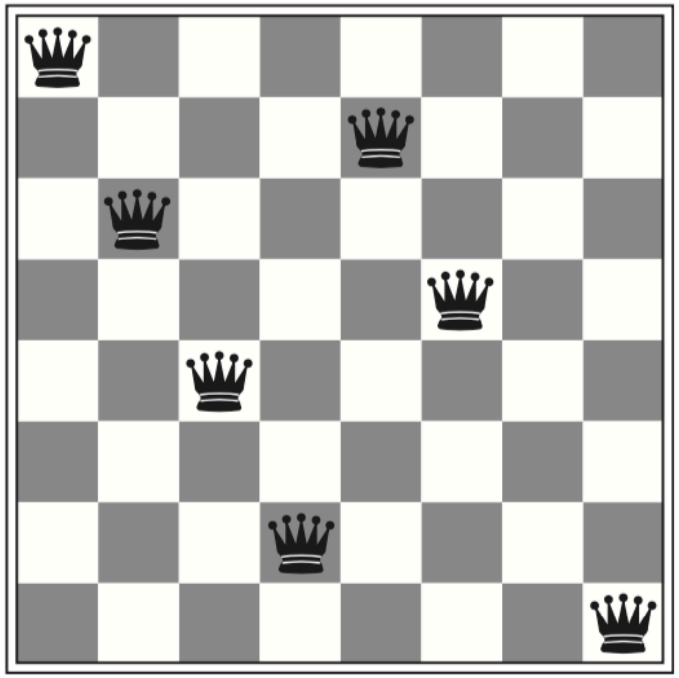
Esempio di toy problem:
Il Rompicapo a 8 Tasselli

- Il rompicapo a 8 tasselli è il fratello minore di quello a 15, inventato da Noyes Chapman, un postino di Canastota, New York, a metà degli anni '70 del 1800.
- Negli Stati Uniti il rompicapo a 15 tasselli ottenne subito un'immensa popolarità, paragonabile al successo ottenuto nei giorni nostri dal Cubo di Rubik.
- I curatori del *Journal of Mathematics* scrissero: "Il rompicapo a 15 tasselli, nelle ultime settimane, ha goduto di una posizione preminente nell'attenzione del pubblico americano, e si può affermare con certezza che ha interessato nove persone su dieci della comunità, di entrambi i sessi e di tutte le età."

Esempio di toy problem:
Problema delle 8 Regine

- Il problema consiste nel trovare il modo di disporre otto regine su una scacchiera in modo che nessuna di esse sia minacciata dalle altre.
- In altre parole (dato che la regina può spostarsi in orizzontale, in verticale e in diagonale di un qualsiasi numero di caselle) ogni regina deve avere la sua riga, la sua colonna e le sue due diagonali libere.
- Le soluzioni possibili sono 92.

Esempio di toy problem:
Problema delle 8 Regine



Esempio di toy problem:
Problema delle 8 Regine

Ci sono due principali categorie di formulazione:

- Formulazione incrementale
  - Utilizza operatori che estendono progressivamente la descrizione di stato, cominciando dallo stato vuoto.
  - Si comincia con la scacchiera senza regine e ogni azione ne aggiunge una sulla scacchiera.
- Formulazione a stato completo
  - Comincia con le otto regine già sulla scacchiera e le sposta.

Esempio di toy problem:
Problema delle 8 Regine

Una prima formulazione incrementale:

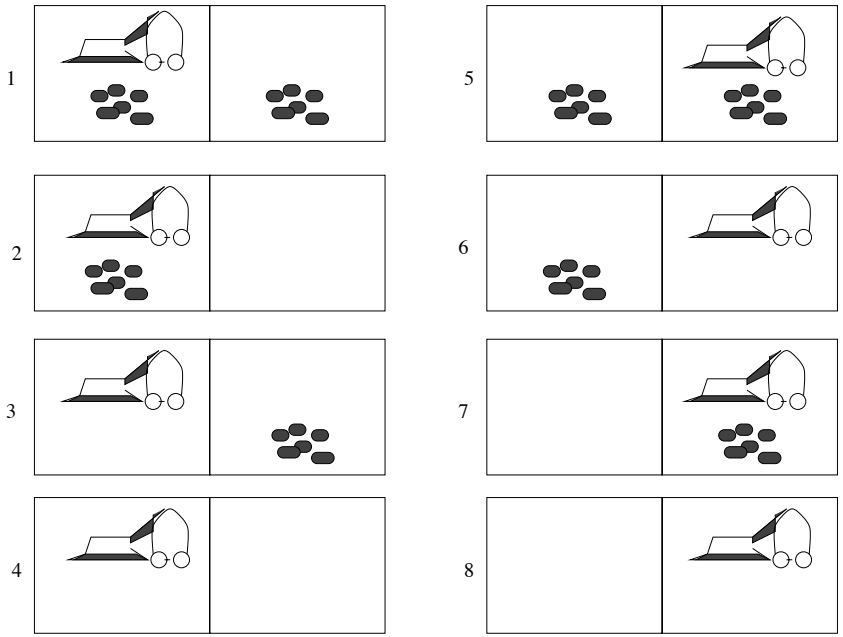
- Stati: Ogni piazzamento sulla scacchiera di un numero da 0 a 8 regine è uno stato.
- Stato Iniziale: La scacchiera è vuota.
- Azioni: Aggiunge una regina in una casella vuota.
- Goal Test: Sulla scacchiera ci sono 8 regine e nessuna è attaccata.
- Costo del Cammino: Ogni passo costa 1.

Esempio di toy problem:
Problema delle 8 Regine

- Il problema delle 8 regine fu pubblicato per la prima volta nel 1848 in forma anonima, nella rivista tedesca di scacchi *Schach*; più tardi fu attribuito a un certo Max Bezzel.
- Fu ripubblicato nel 1850 e attirò l'attenzione del famoso matematico Carl Friederich Gauss, che cercò di enumerare tutte le possibili soluzioni, ma ne trovò solo 72.
- Nauck pubblicò tutte le 92 soluzioni più tardi quello stesso anno.

Esempio di toy problem:
Il Mondo dell'Aspirapolvere

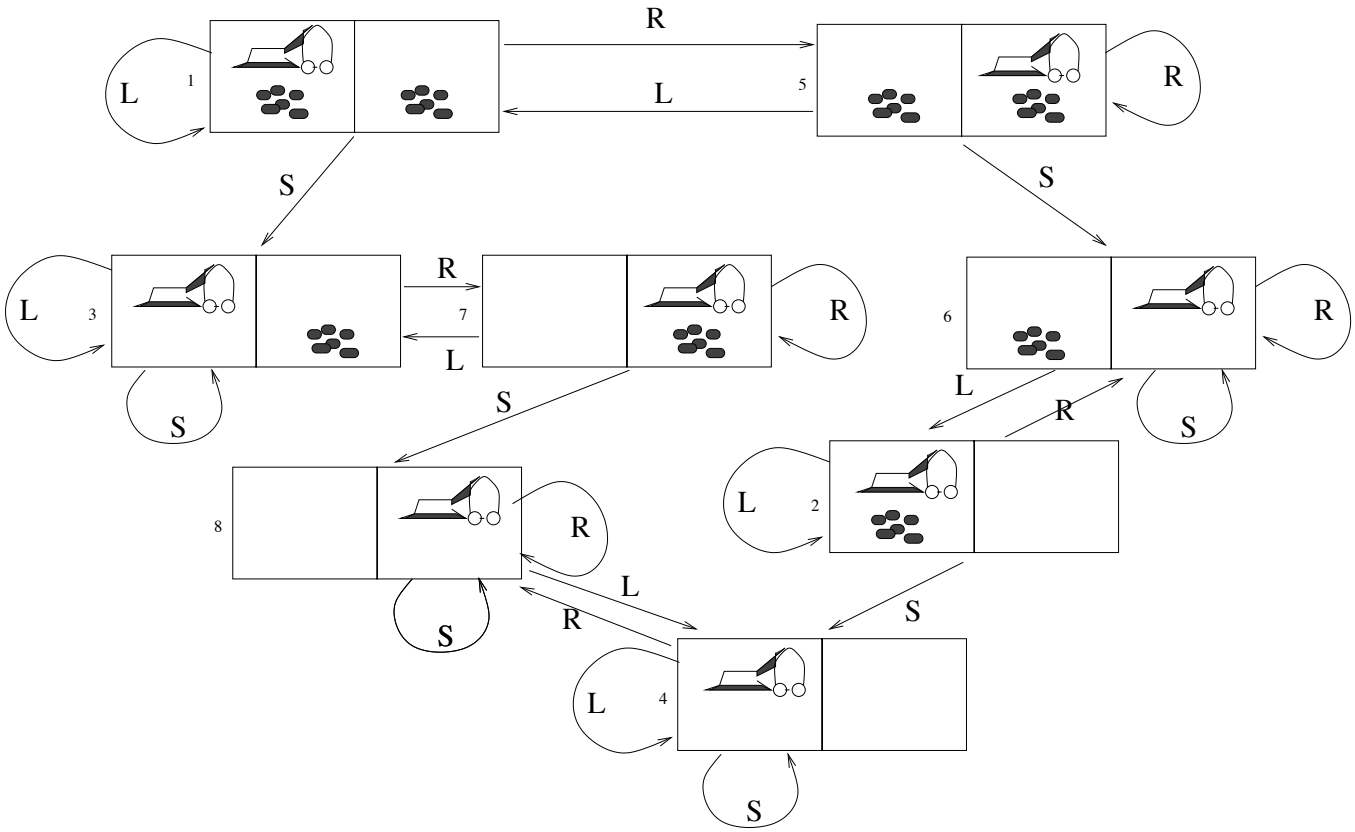
Gli otto stati possibili del mondo del robot aspirapolvere:



Azioni: **Left**, **Right**, **Suck**

Esempio di toy problem:
Il Mondo dell'Aspirapolvere

Spazio degli stati:



Esempio di toy problem:
Il Mondo dell'Aspirapolvere

- Stati: L'agente si trova in uno di due possibili caselle, ognuna delle quali può contenere sporco oppure no.
- Stato Iniziale: Ogni stato (1, …, 8) può essere designato come stato iniziale.
- Azioni: Tre azioni possibili: Left, Right, Suck.
- Goal Test: Controlla se entrambi i riquadri sono puliti (stati {4, 8}).
- Costo del Cammino: Ogni passo costa 1.

PROBLEMI BEN DEFINITI E SOLUZIONI

Un problema può essere definito formalmente da quattro componenti:

- Lo **stato iniziale** in cui si trova l'agente.
- Una descrizione delle **azioni** possibili dell'agente. Una formulazione possibile è quella che usa una **funzione successore** (lo stato iniziale e la funzione successore definiscono lo **spazio degli stati** del problema). Una formulazione alternativa usa un insieme di operatori che possono essere applicati a uno stato per generare i successori.
- Il **test obiettivo**, che determina se un particolare stato è uno stato obiettivo.
- La funzione **costo di cammino**, che assegna un costo numerico ad ogni cammino.

PROBLEMI BEN DEFINITI E SOLUZIONI

- Possiamo definire un tipo di dato con cui rappresentare i problemi:

  datatype Problema
  components: Stato-Iniziale, Operatori, Test-Obiettivo, Funzione-Costo-Cammino

- Istanze di questo tipo saranno l'input dei nostri algoritmi di ricerca.

Scegliere lo spazio degli stati

Processo di astrazione:

Il mondo reale è enormemente complesso
⇒ lo spazio degli stati deve essere individuato mediante un processo di astrazione

Stato (astratto) = insieme di stati reali

Azione (astratta) = combinazione complessa di azioni reali
ad es: "Arad → Zerind" rappresenta un complesso insieme di possibili percorsi, deviazioni, soste, ecc.

Ogni stato reale "in Arad" deve condurre in uno stato reale "in Zerind"

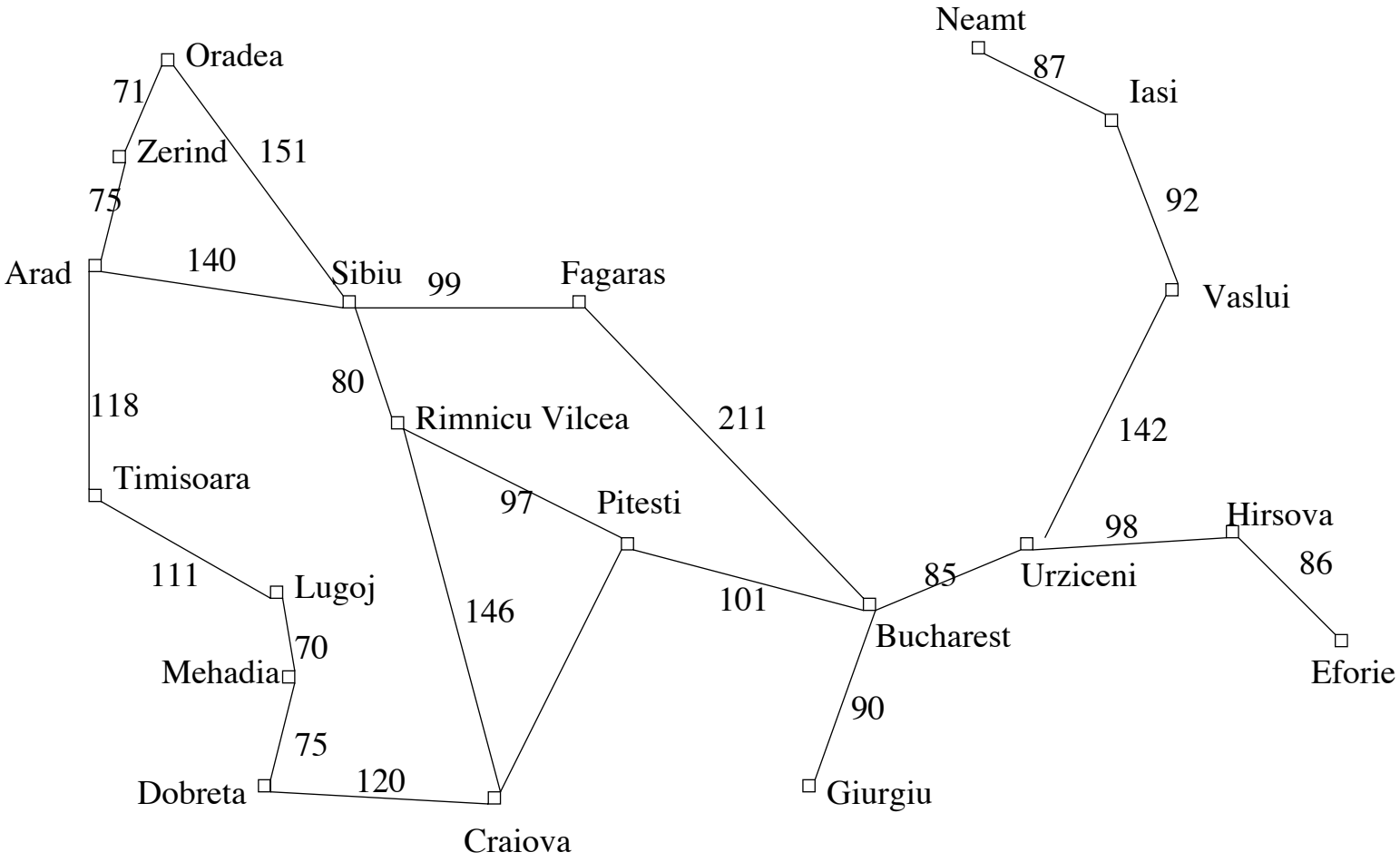
Soluzione (astratta) = insieme di percorsi reali che sono soluzioni nel mondo reale

Ogni soluzione astratta dovrebbe essere più "facile" del problema originale

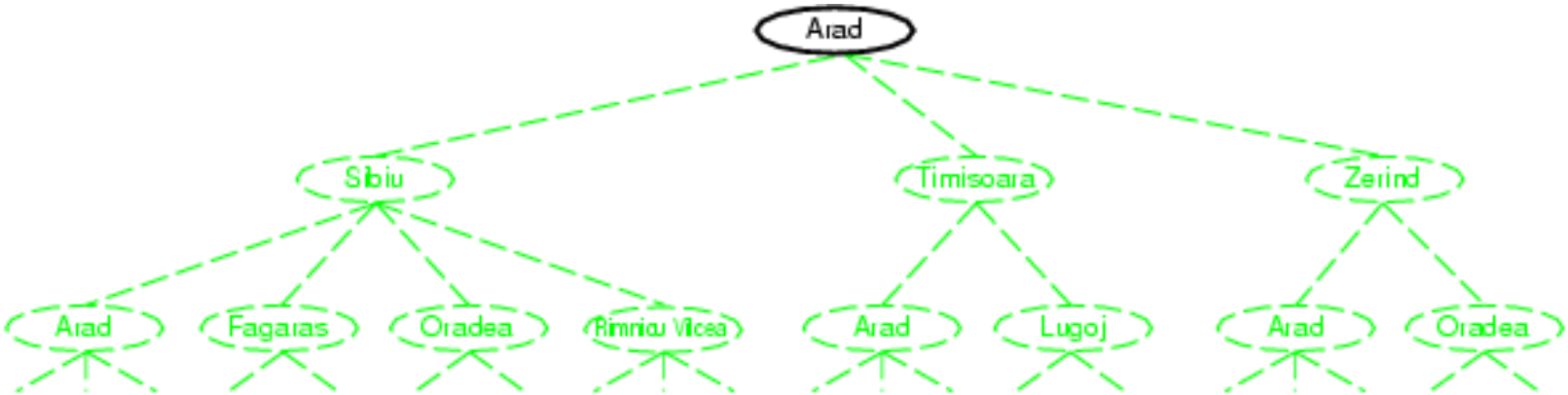
Ricerca di una Soluzione

- Dopo aver formulato un problema, occorre risolverlo.
- Una soluzione al problema è una sequenza di azioni.
- Gli algoritmi di ricerca operano considerando varie possibili sequenze di azioni.
- Le possibili sequenze di azioni a partire dallo stato iniziale formano un **albero di ricerca**.
- Tale albero ha come **radice** lo stato iniziale, i **rami** sono azioni e i **nodi** corrispondono a stati nello spazio degli stati.

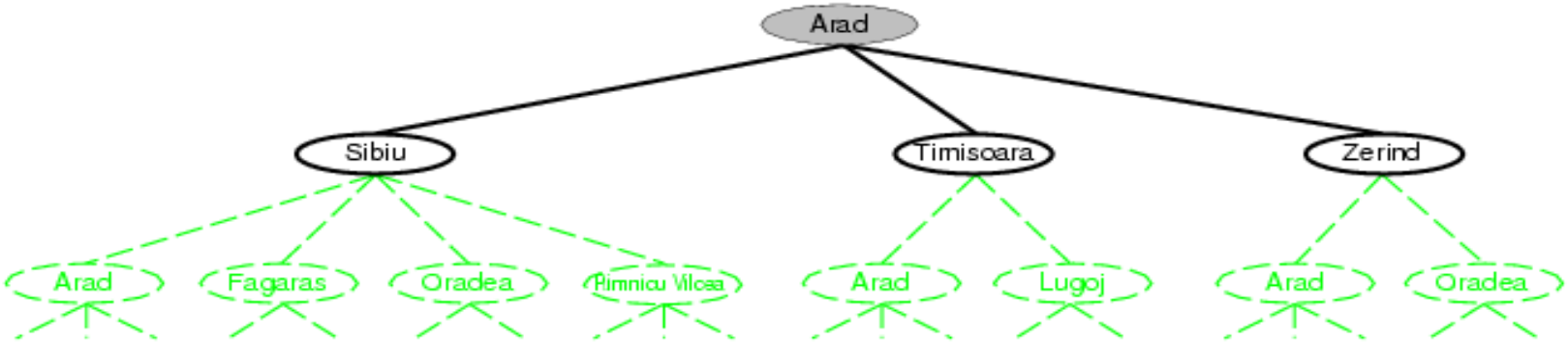
Esempio: raggiungere Bucarest



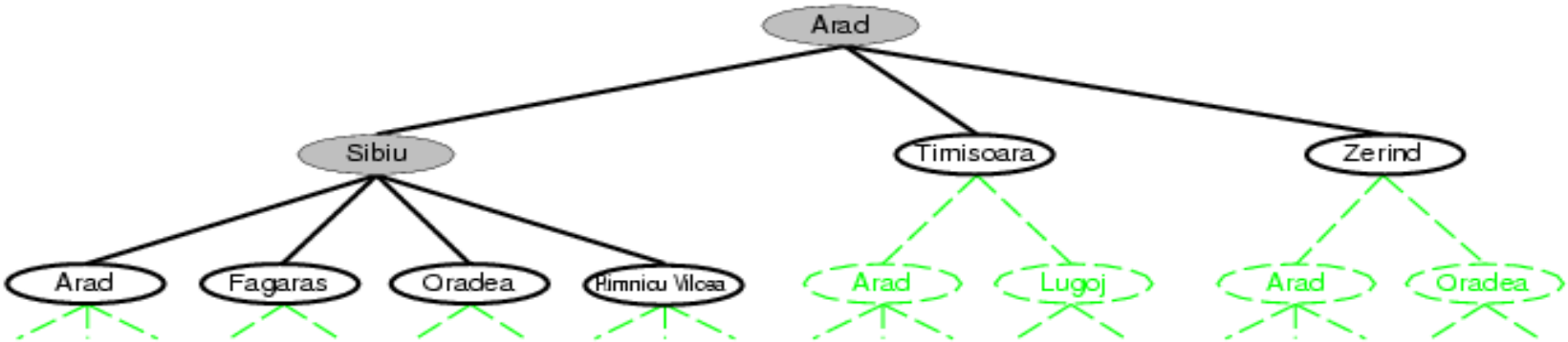
Esempio di albero di ricerca (1)



Esempio di albero di ricerca (2)



Esempio di albero di ricerca (3)



Albero Di Ricerca

Ogni nodo dell'albero contiene uno stato, insieme ad altre informazioni:

```
NODO = <stato, genitore, operatore, profondità,...>
```

Nodo:

- stato
- nodo genitore
- operatore che, applicato allo stato del nodo genitore, genera lo stato del nodo
- profondità del nodo
- costo del cammino dallo stato iniziale al nodo
- ...

La radice dell'albero contiene lo stato iniziale.

N.B.: nodi diversi possono contenere lo stesso stato

Spazio degli stati $\neq$ Albero di ricerca

Processo di ricerca

Il processo di ricerca comporta i seguenti passi:

- scegliere (tra le foglie dell'albero) un nodo da "espandere", secondo una data *strategia*
- controllare se il nodo scelto è un obiettivo
- se non lo è, "espandere" il nodo: generare i suoi figli, ciascuno dei quali contiene uno stato risultante dall'applicazione di un operatore allo stato del nodo espanso.

  Quando si espande un nodo si calcolano tutte le componenti dei nodi generati.

La collezione di nodi in attesa di essere espansi (le foglie dell'albero) è chiamata **confine**, **frontiera**, **frangia**, **lista aperta**.

Algoritmo Generale di Ricerca (1)
Tree-Search

```
function TREE-SEARCH(problema, strategia) returns una soluzione o
                                                un fallimento

Inizializzare l'albero di ricerca usando lo stato iniziale del
problema.

loop do
      • Se non ci sono candidati per l'espansione (frontiera vuota)
        riportare fallimento.
      • Scegliere una foglia per l'espansione (in base alla
        strategia).
      • Se il nodo scelto contiene uno stato obiettivo, riportare la
        soluzione corrispondente (costruita seguendo i "puntatori al
        padre").
      • Altrimenti espandere il nodo e aggiungere i nodi risultanti
        all'albero di ricerca.
end
```

Attenzione: la frontiera può contenere un nodo relativo a uno stato obiettivo, ma la ricerca non termina finché tale nodo non viene scelto per l'espansione: soltanto allora infatti si riconosce che contiene uno stato obiettivo.

La strategia di ricerca (1)

Come rappresentare la strategia di ricerca, ossia il criterio per la scelta del prossimo nodo da espandere?

Strategia di ricerca ≡ funzione per la scelta di un elemento da un insieme di nodi (frontiera)

≡ funzione di inserimento di un elemento (o un insieme di elementi) in una sequenza

Scelta di un elemento ≡ scelta del primo elemento della sequenza

La strategia di ricerca (2)

Quindi: la frontiera è implementata da una struttura che chiamiamo "coda" (queue) (anche se non è necessariamente una struttura FIFO), sulla quale sono definite le seguenti operazioni:

`MAKE-QUEUE : node -> queue` — `MAKE-QUEUE(n)`: coda contenente solo il nodo `n`

`EMPTY? : queue -> bool` — test coda vuota

`REMOVE-FRONT : queue -> node`
`REMOVE-FRONT(q)`: elimina il primo elemento da `q` e lo riporta come valore

`QUEUING-FN : queue * node list -> queue`
`QUEUING-FN(q,lista)`: aggiunge a `q` tutti i nodi in `lista`

La QUEUING-FN non inserisce necessariamente in coda: diverse QUEUING-FN producono diverse strategie di ricerca e, corrispondentemente, diverse versioni dell'algoritmo di ricerca.

Algoritmo generale di ricerca (2)

Assumiamo che siano definite:

- Operazioni sul tipo di dato PROBLEMA

  datatype Problema
  components: Stato-Iniziale, Operatori, Test-Obiettivo, Funzione-Costo-Cammino

- Operazioni sul tipo di dato NODO

  datatype Nodo
  components: Stato, Genitore, Operatore, Profondità, ...

Algoritmo generale di ricerca (3)

Operazioni sul tipo di dato PROBLEMA:

`INITIAL-STATE : problem -> state`. Riporta lo stato iniziale del problema

`GOAL-TEST : problem * state -> bool`.
`GOAL-TEST(p,s)`: vero se lo stato `s` soddisfa il test obiettivo del problema `p`.

`OPERATORS : problem -> operators`. Riporta una lista con tutti gli operatori del problema.
Ciascun operatore si applica a uno stato e riporta una lista di stati

```
op:  state -> state list
```

Algoritmo generale di ricerca (4)

Operazioni sul tipo di dato NODO:

`MAKE-NODE : state -> node`. Costruisce il nodo radice contenente lo stato dato

`STATE : node -> state`. Riporta lo stato contenuto nel nodo

`EXPAND : node * operators -> node list`.

`EXPAND(n,ops)`: lista di tutti i nodi che si ottengono applicando un operatore in `ops` al nodo `n`

Algoritmo Generale di Ricerca (5)
TREE-Search

```
function TREE-SEARCH(problem) returns a solution or failure

fringe ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem])
loop do
     If EMPTY?(fringe) then return failure
     node ← REMOVE-FRONT(fringe)
     if GOAL-TEST(problem, STATE[node]) then return SOLUTION(node)
     fringe ← QUEUING-FN(fringe, EXPAND(node, OPERATORS(problem)))
end
```

Osservazione: In questa versione dell'algoritmo, in memoria non viene conservato l'intero albero di ricerca, ma soltanto la "coda" (fringe) con i nodi della frontiera.

Criteri per valutare una strategia di ricerca

Una strategia di ricerca può essere valutata secondo i seguenti criteri:

Completezza : la strategia garantisce di trovare una soluzione quando ne esiste una

Complessità in tempo : quanto tempo ci vuole per trovare una soluzione (nel caso peggiore)

Complessità in spazio : quanta memoria occorre per effettuare la ricerca

Ottimalità : quando il problema ha diverse soluzioni, la strategia trova una delle migliori (a costo minimo)

Sintesi degli Argomenti Trattati nella Lezione

- Un Agente, prima che possa cominciare a cercare soluzioni, deve formulare un obiettivo e poi usare l'obiettivo per formulare un problema.
- Un **problema** consiste in quattro parti: lo **stato iniziale**, un insieme di **operatori** o **azioni** (o una **funzione successore**), una funzione **test obiettivo** (**GOAL TEST**) e una funzione **costo di cammino**.
- L'ambiente del problema è rappresentato da uno **spazio degli stati**. Un cammino attraverso lo spazio degli stati dallo stato iniziale a uno stato obiettivo è una **soluzione**.
- Nella vita reale la maggior parte dei problemi sono mal definiti; però, con alcune analisi, molti problemi possono rientrare nel modello dello spazio degli stati.
- Un unico algoritmo di **ricerca generale** (**TREE-SEARCH**) può essere utilizzato per risolvere qualsiasi problema; specifiche varianti dell'algoritmo realizzano differenti strategie.
- Gli algoritmi di ricerca sono giudicati sulla base di **completezza**, **ottimalità**, **complessità temporale** e **complessità spaziale**. La complessità dipende da **b**, il fattore di ramificazione (**branching factor**) dello spazio degli stati e da **d**, la profondità della soluzione più superficiale.

Riferimenti

Russell, S., Norvig, P. *Artificial Intelligence - a Modern Approach*, fourth Edition, Pearson Education, 2021.

Nilsson, N.J. *Artificial Intelligence - a New Synthesis*, Morgan Kaufman, 1998.