

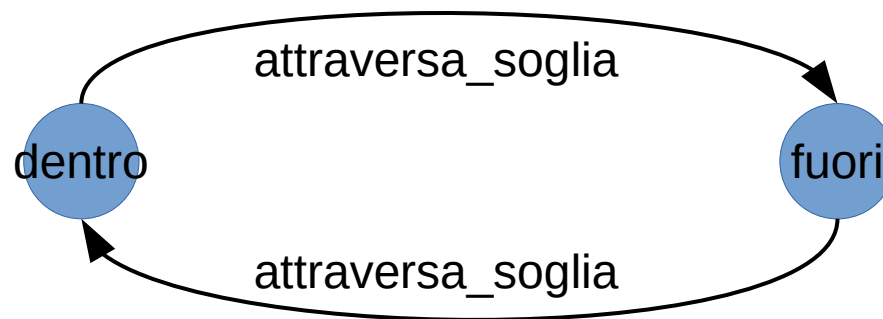
# Risoluzione automatica di problemi

*In questa parte si affronta la problematica di come definire il concetto di problema e di soluzione, di distinguere tra soluzione e soluzione ottima. Sono studiati tre approcci alla risoluzione di problemi: ricerca nello spazio degli stati, ricerca in spazi con avversario (giochi ad informazione completa) , risoluzione di problemi mediante soddisfacimento di vincoli*

# Quali problemi?

# Stati e azioni

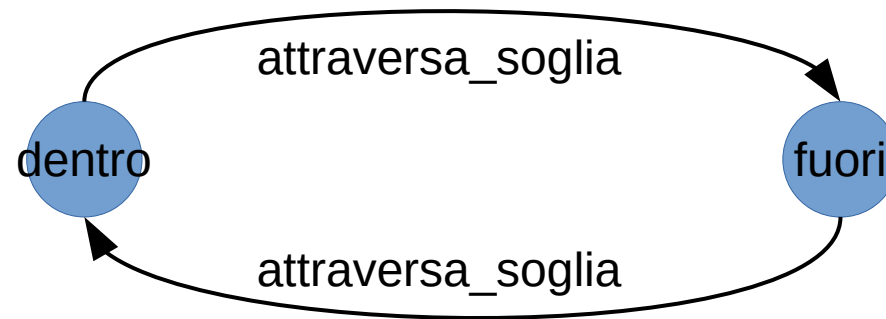
- La realtà che definisce un problema può essere astratta in un insieme di **stati**
- La realtà transisce da uno stato ad un altro tramite l'esecuzione di **azioni** (o operazioni)
- **Esempio:**  
 $\text{stato} \in \{\text{dentro}, \text{fuori}\}, \text{azione} \in \{\text{attraversa\_soglia}\}$



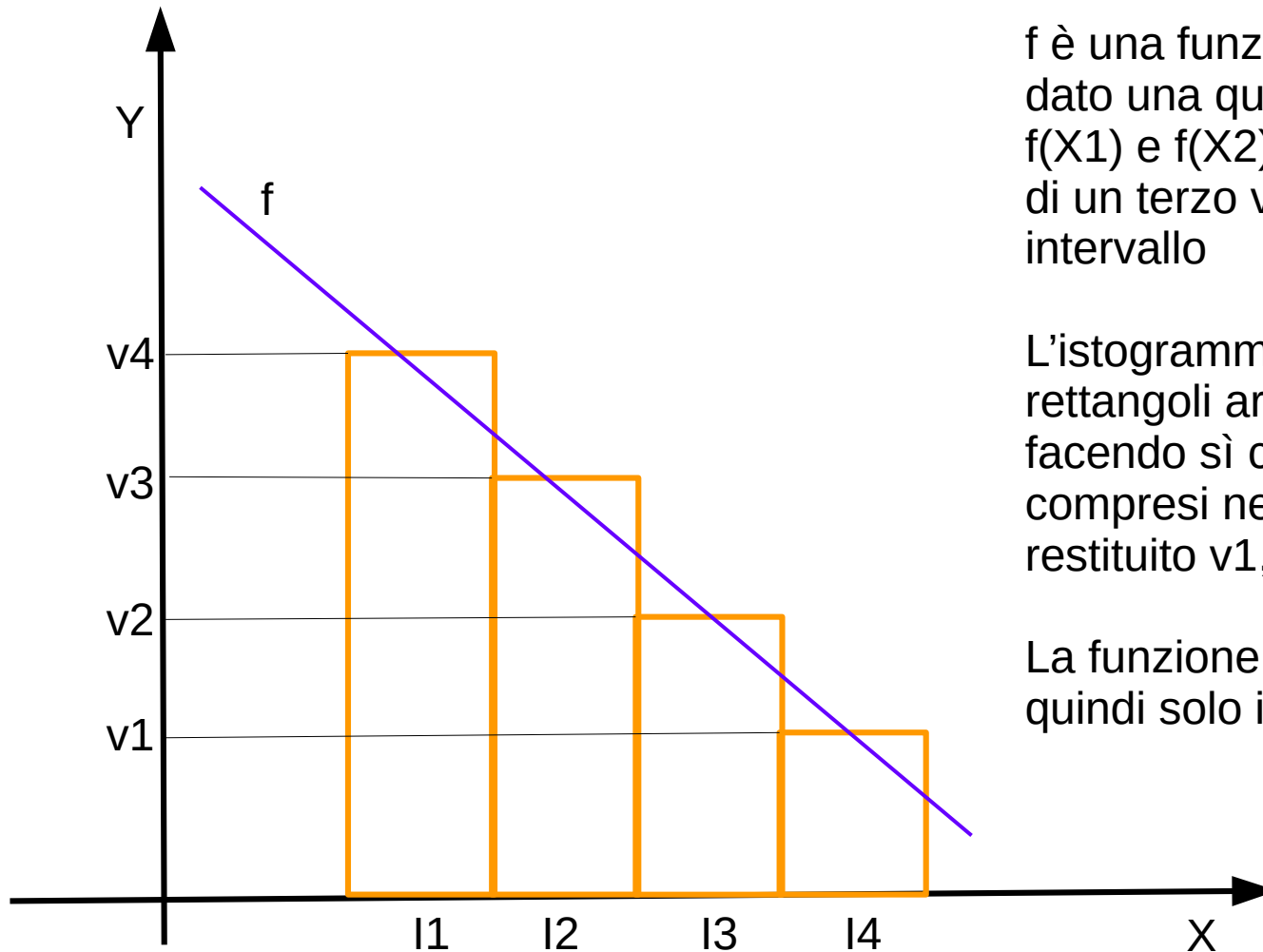
Grafo di transizione  
di stato

# Stati e azioni

- $stato \in \{\text{dentro}, \text{fuori}\}$ ,  $azione \in \{\text{attraversa\_soglia}\}$
- Caratteristiche:
  - Stati discreti (o dentro o fuori anche se nel mondo fisico esiste una transizione graduale attraverso la soglia)
  - Effetto deterministico delle azioni
  - Dominio statico (non cambia durante l'esecuzione delle azioni)



# Valori discreti e valori continui



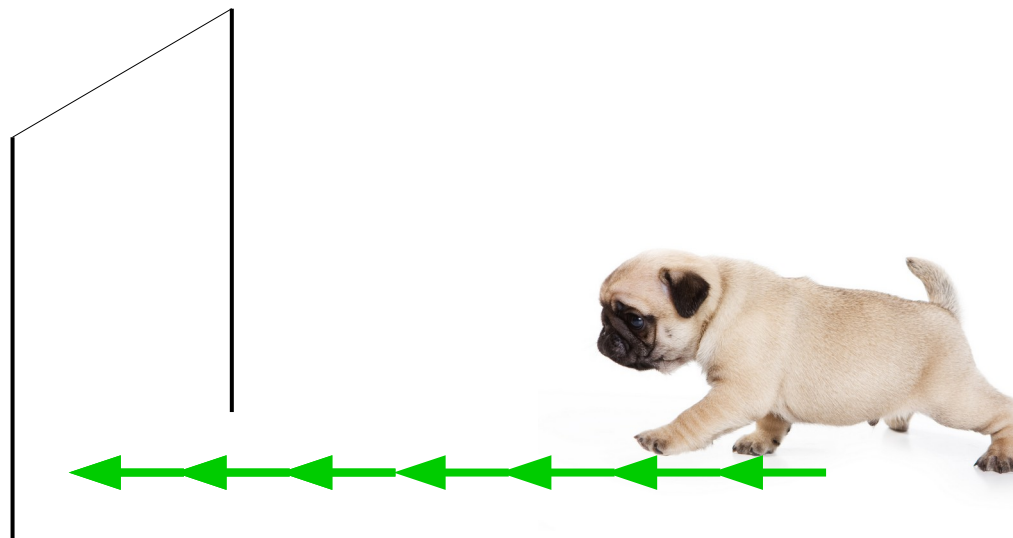
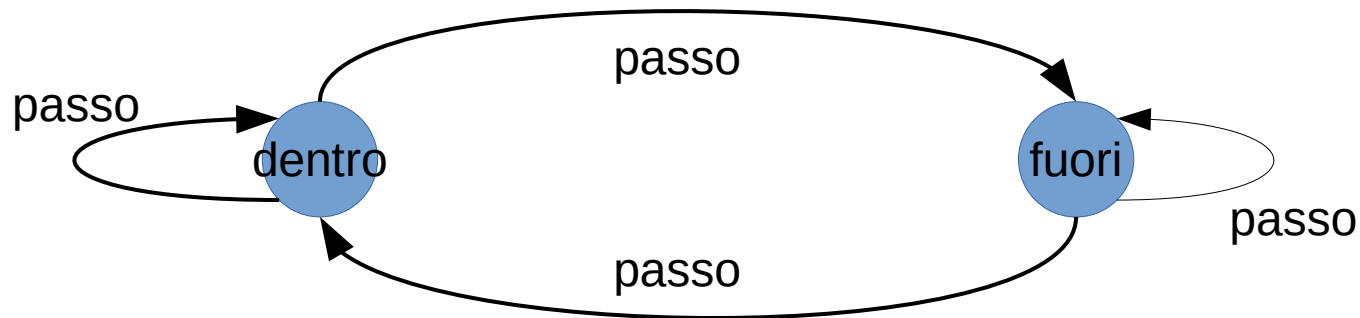
$f$  è una funzione a valori continui:  
dato una qualsiasi coppia di valori di  $f(X1)$  e  $f(X2)$  si può supporre l'esistenza di un terzo valore compreso in questo intervallo

L'istogramma rappresentato tramite i rettangoli arancioni discretizza  $f$  facendo sì che per tutti i valori di  $X$  compresi nell'intervallo  $I1$  venga restituito  $v1$ , per quelli di  $I2$ ,  $v2$ , ecc.

La funzione  $f$  discretizzata assume quindi solo i valori  $v1$ ,  $v2$ ,  $v3$  e  $v4$

# Esempio non deterministico

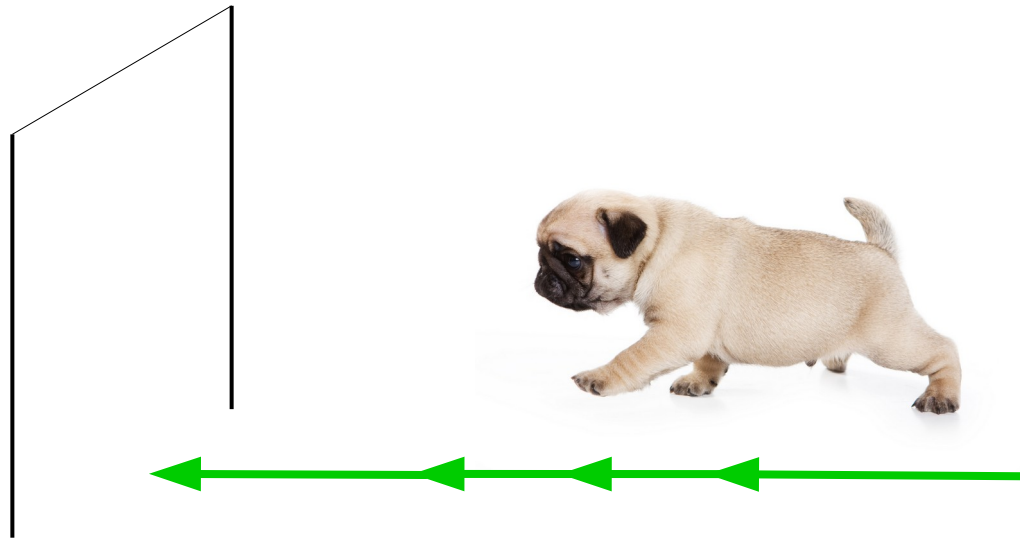
- $\text{stato} \in \{\text{dentro}, \text{fuori}\}$ ,  $\text{azione} \in \{\text{passo}\}$



Passo dopo passo uscirà dalla stanza, quindi nello stato “dentro” l’azione “passo” può avere due effetti diversi, a seconda della posizione (non rappresentata) nella stanza

# Esempio azione a valori continui

- azione  $\in \{\text{passo}(\text{spinta})\}$



A seconda della spinta impressa  
i passi avranno una lunghezza minore  
o maggiore in un intervallo continuo

# Obiettivi e ricerca

- **Obiettivo**: risultato verso il quale gli sforzi sono diretti

La risoluzione automatica di problemi è centrata su questa nozione

- Un obiettivo è una condizione data in termini di:
  - 1) situazione
  - 2) prestazione (es. raggiungere un luogo – situazione – in un certo tempo – prestazione -)
- Insieme degli stati obiettivo: tutti gli stati in cui vale la condizione che lo definisce  
Spesso sono detti stati goal o stati target

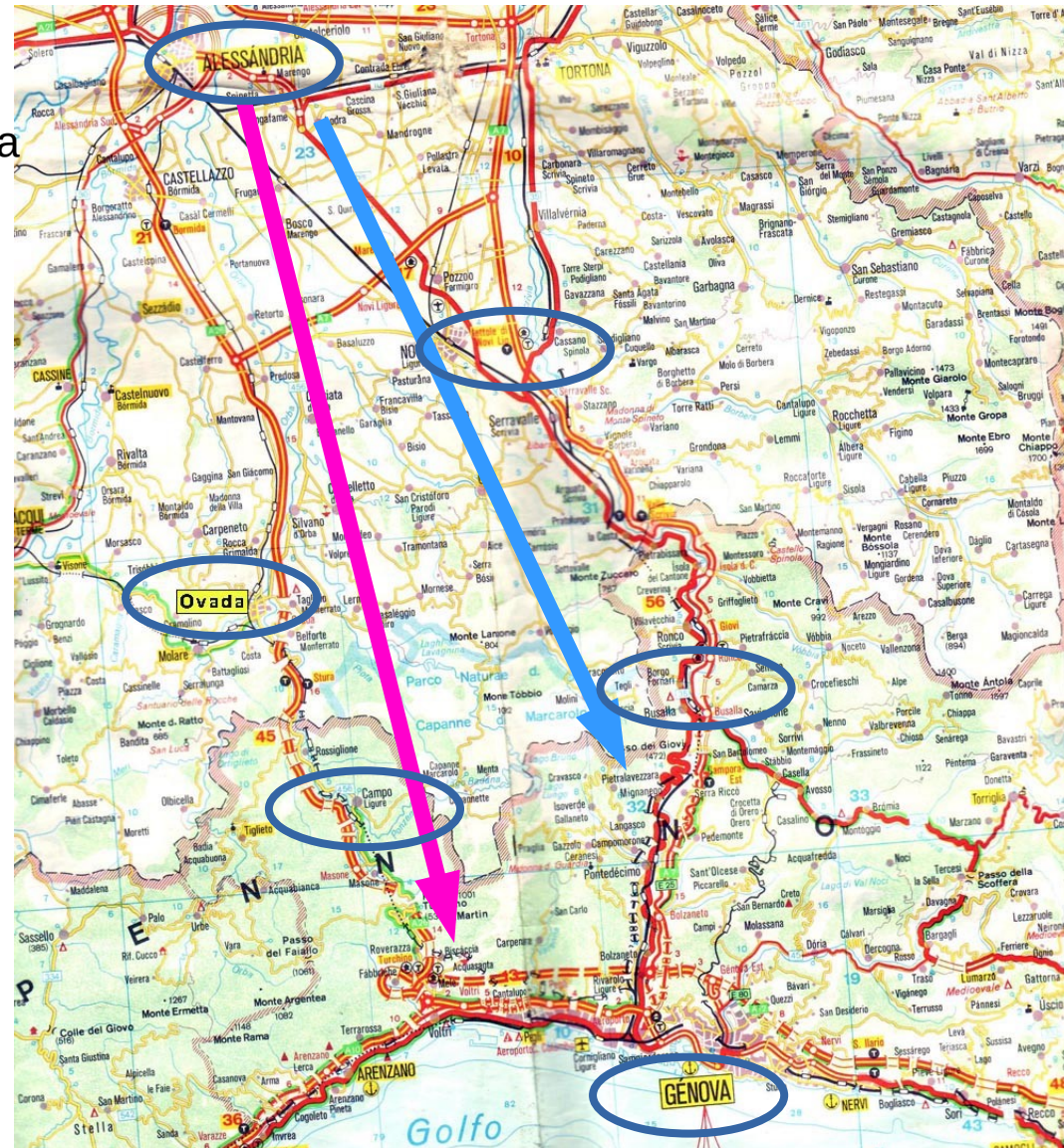


# Obiettivi e ricerca

- Obiettivo: risultato verso il quale gli sforzi sono diretti
- L'**algoritmo di ricerca** determina una **soluzione** che, a partire da uno **stato iniziale**, permette di raggiungere un dato stato **obiettivo**
- Usa:
  - 1)una **descrizione del problema**
  - 2)un **metodo di ricerca** attraverso lo spazio degli stati
- Una soluzione è un percorso nello spazio degli stati

# Esempio: percorso

Stato = luogo  
Stato iniziale: Alessandria  
Obiettivo: Genova  
Transizione: passaggio  
da una città ad un'altra  
direttamente collegata  
Effetti deterministici  
Stati discreti





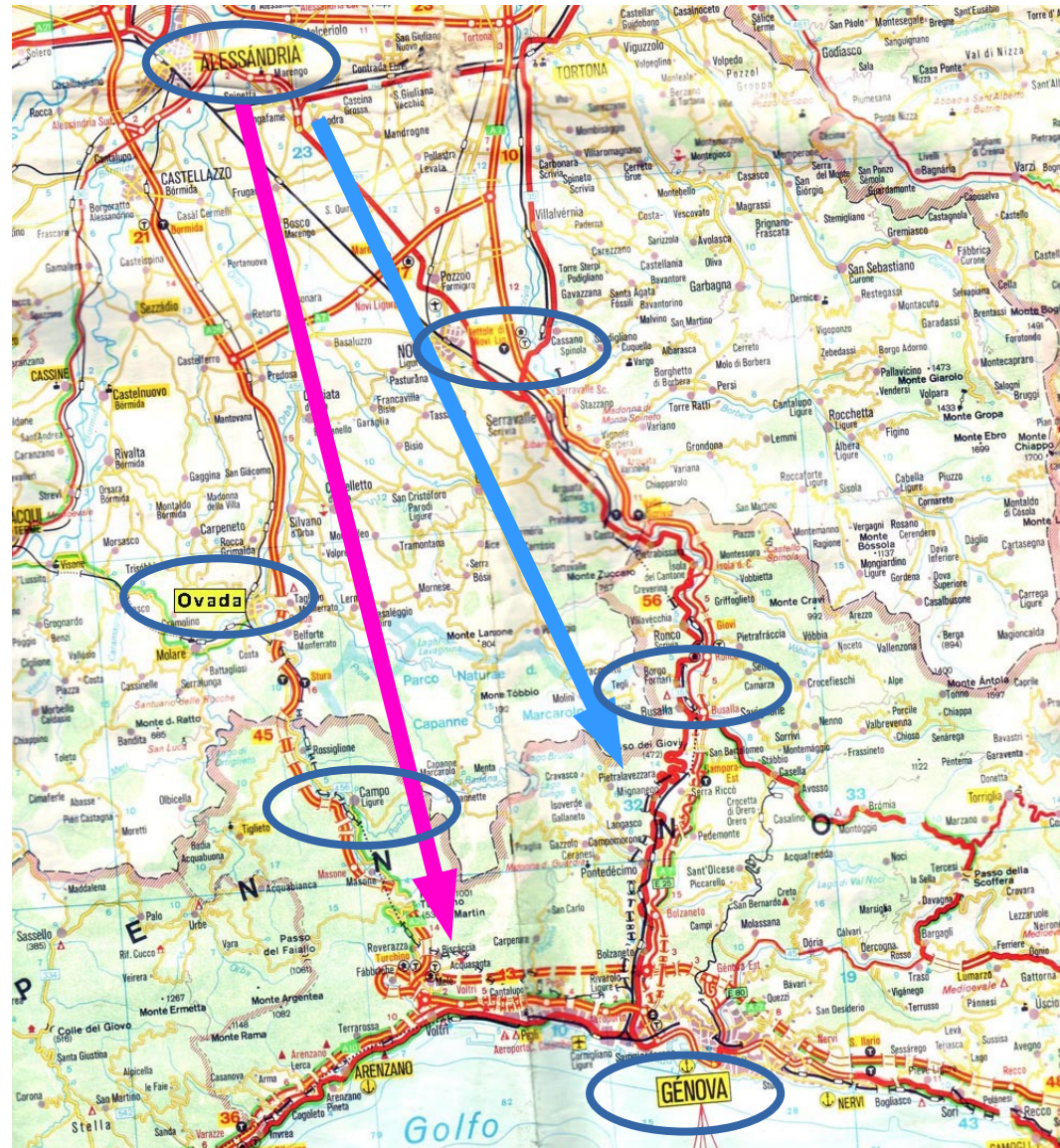
# Esempio: percorso

## Avendo:

Una descrizione degli stati e una descrizione delle azioni e dei loro effetti

**Diventa possibile:**  
effettuare la ricerca di una soluzione intesa come sequenza di passi per raggiungere l'obiettivo

**Occorre un criterio di scelta**



# Definizione formale del problema

- Un problema di ricerca può essere definito come una tupla di 4 elementi:
  1. Stato iniziale
  2. Funzione successore
  3. Test obiettivo
  4. Funzione di costo del cammino
- L'insieme degli stati possibili è spesso implicito (generativo) ma tali stati possono essere enumerati

# Definizione formale del problema

- Un problema può essere definito formalmente come una tupla di 4 elementi:
  1. *Stato iniziale*  
cattura la situazione a partire dalla quale viene computata la soluzione
  2. *Funzione successore*  
dato uno stato e un'azione legale in esso, calcola lo stato a cui si transisce eseguendo quell'azione in quello stato
  3. *Test obiettivo*  
determina se lo stato a cui è applicato è lo stato goal: può verificare una proprietà o verificare l'appartenenza dello stato all'insieme degli stati target
  4. *Funzione di costo del cammino*  
dato un percorso possibile, gli assegna un costo numerico

# Definizione formale del problema: esempio

- Problema di navigazione:

1. *Stato iniziale*

stato che cattura la situazione iniziale, esempio: `in(Alessandria)`

2. *Funzione successore*

esempio: lo stato successore dello stato `in(Alessandria)`, eseguendo l'azione `go(Ovada)`, è `in(Ovada)`. La transizione è possibile perché questa città è raggiungibile dalla prima

3. *Test obiettivo*

verifica se uno stato è quello obiettivo, esempio se la destinazione è Genova, controllerà se lo stato corrisponde a `in(Genova)`

4. *Funzione di costo del cammino*

potrebbe essere semplicemente il numero di chilometri percorsi, o una misura più complessa, che per esempio combina i Km percorsi con i pedaggi pagati

# Astrazione di stati e azioni

- **Stati**: occorre rappresentare solo l'informazione rilevante alla soluzione del problema
- **Azioni**: occorre rappresentare solo gli aspetti (es. gli effetti) relativi alla soluzione del problema

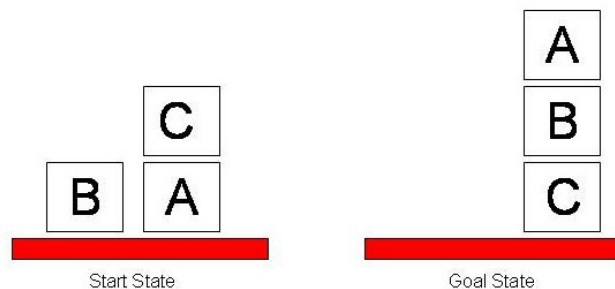
# Astrazione di stati e azioni

- **Stati**: occorre rappresentare solo l'informazione rilevante alla soluzione del problema
  - Esempio:
    - posizione sì,
    - panorama no
- **Azioni**: occorre rappresentare solo gli aspetti (es. gli effetti) funzionali alla soluzione del problema
  - Esempio:
    - posizione raggiunta sì,
    - inquinamento prodotto no



# Toy problem – Real-world problem

- **Toy problem**: è un problema artificiale avente lo scopo di illustrare o mettere alla prova dei metodi di risoluzione. Ha una formulazione precisa e univoca. Utile per confrontare metodi diversi

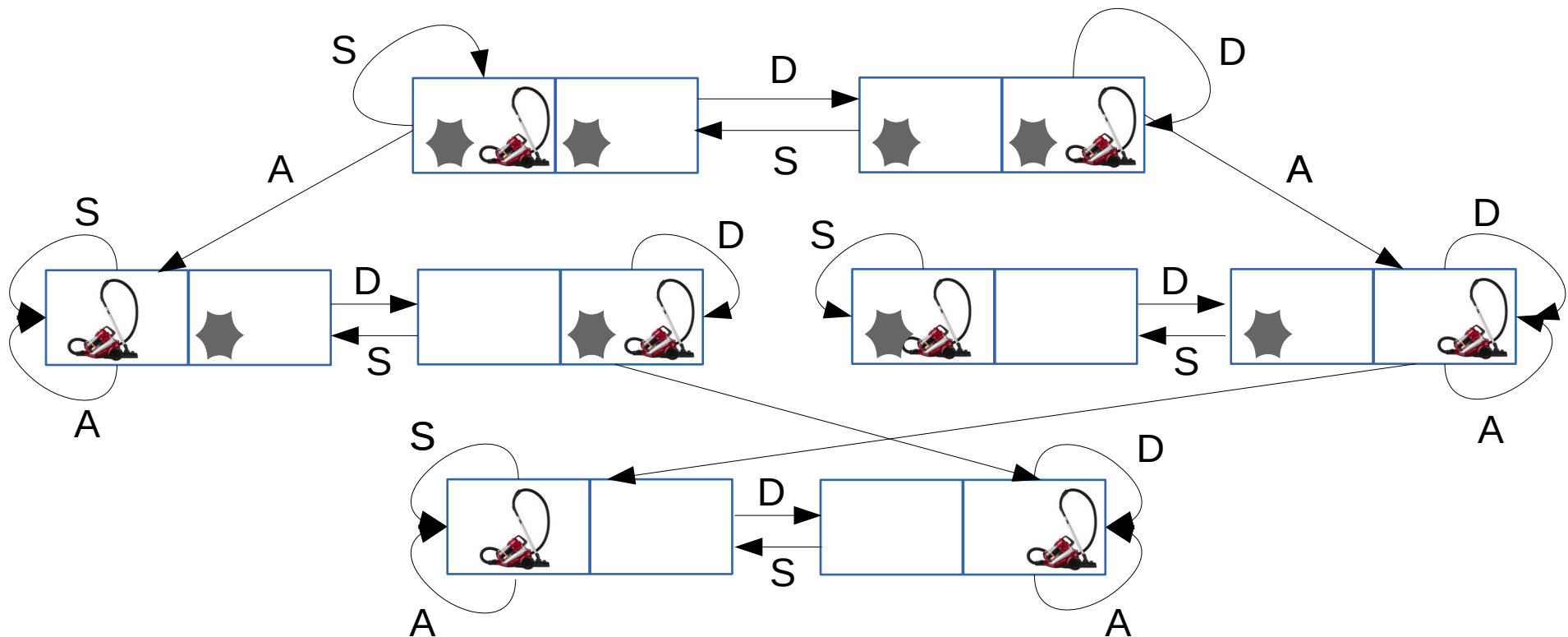


- **Real-world problem**: problemi concreti, effettivi. Spesso non hanno una formulazione unica. Es. configurazioni VLSI, itinerario, navigazione robotica

# Toy problem: mondo dell'aspirapolvere



Due stanze, ciascuna delle quali può essere sporca o pulita  
Un aspirapolvere che si trova in una delle due stanze  
S=sinistra, D=destra, A=aspira



Stati possibili e transizioni

# Problema dell'aspirapolvere

- **Caratteristiche di ogni stanza:**

sporca o pulita, con o senza aspirapolvere, quindi una stanza può essere in 4 possibili configurazioni <pulita, vuota>, <pulita, aspirapolvere>, <sporca, vuota>, <sporca, aspirapolvere>. Si dice che la stanza può avere 4 possibili valori

- Tutte le azioni sono applicabili a tutti gli stati

- **Obiettivo: pulire entrambe le stanze**

- *Stato iniziale*

è quello in cui si trova l'aspirapolvere. Qualsiasi stato può essere lo stato iniziale

- *Funzione successore*

restituisce lo stato prodotto dall'applicazione dell'azione scelta allo stato corrente

- *Test obiettivo*

verifica se entrambe le stanze sono pulite

- *Funzione di costo del cammino*

ogni azione costa 1, il costo del cammino è dato dal numero di azioni che lo costituiscono

# Gioco dell'8

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Start State

Goal State

Figure 3.4 FILES: figures/8puzzle.eps (Tue Nov 3 16:22:11 2009). A typical instance of the 8-puzzle.

**Stato:** posizione di ciascun numero e dello spazio vuoto. Vi sono 181.440 stati possibili, cioè  $9! / 2$

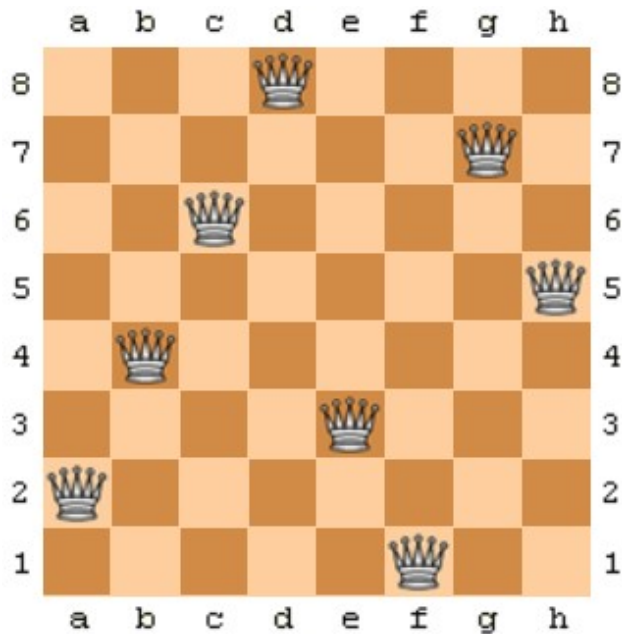
**Stato iniziale:** qualsiasi stato può esser lo stato iniziale

**Funzione successore:** quella che si ottiene spostando un numero adiacente allo spazio vuoto nello spazio vuoto

**Test obiettivo:** verifica che la posizione dei numeri corrisponda a quella a destra

**Costo del cammino:** ogni passo costa 1, il costo di un cammino è il numero di passi che lo costituiscono

# Problema delle 8 regine



**Posizionare 8 regine su una scacchiera 8x8 in modo tale che nessuna possa attaccarne nessu'altra**

**Nessuna regina deve occupare una riga, colonna o diagonale occupata da un'altra regina**

Stati ?  
Stato iniziale?  
Funzione successore?  
Test obiettivo?  
Costo?

Cercate una soluzione su internet e trascrivetene la formulazione in modo da rispondere alle precedenti domande

# Metodi di ricerca non informati (blind search)

*Questi metodi si appoggiano per lo più a una struttura ad albero detta ALBERO DI RICERCA (in alcuni casi si utilizzano GRAFI di ricerca). I nodi dell'albero corrispondono a stati del problema. Gli archi a transizioni di stato causate dall'applicazione di azioni o operatori. La soluzione cercata viene ottenuta espandendo via via i nodi dell'albero secondo una politica che caratterizza il metodo di ricerca.*

# Albero/grafico di ricerca

- Un albero di ricerca è una struttura dati usata per trovare una soluzione a problema di ricerca
- Ogni nodo corrisponde a uno stato
- I nodi figli sono costruiti tramite la funzione successore
- Ogni nodo ha un riferimento al nodo padre (per ricostruire le soluzioni)
- L'albero è costruito a partire dal nodo corrispondente allo stato iniziale
- L'albero diventa un grafo quando lo stesso nodo (NB: non lo stesso stato) può essere raggiunto tramite più percorsi
- Un percorso che porta dal nodo iniziale a un nodo obiettivo è una soluzione

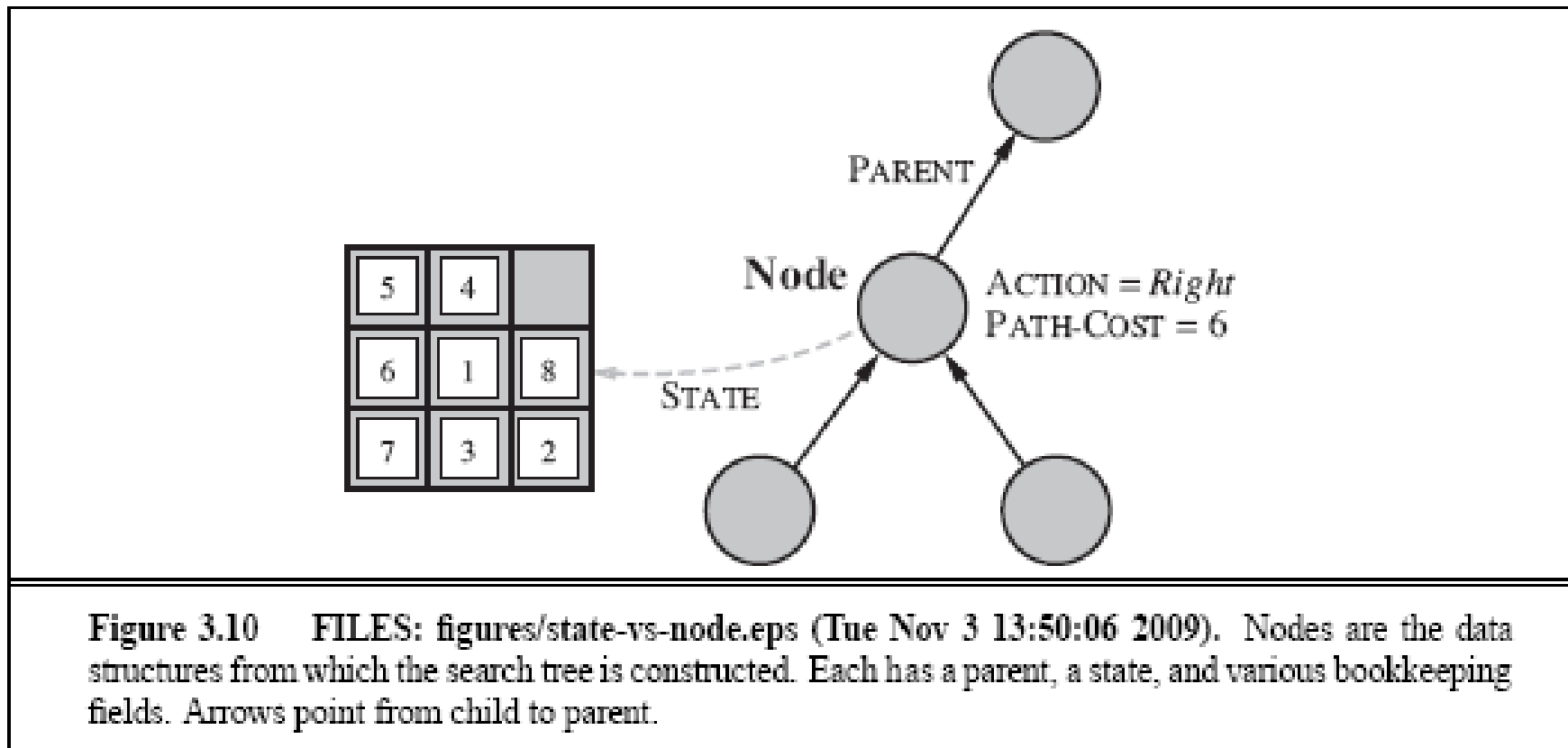
# In modo più formale: grafo di ricerca

- Un grafo di ricerca  $G = (\{n_i\}, \{e_{ij}\})$  è costituito da un insieme di nodi  $n_i$  e di archi  $e_{ij}$
- $e_{pq} \in \{e_{ij}\}$  rappresenta che esiste un arco dal nodo  $n_p$  al nodo  $n_q$ , quindi  $n_q$  è uno dei successori di  $n_p$
- Ciascun arco  $e_{ij}$  ha associato un costo  $c_{ij}$
- L'esistenza di  $e_{ij}$  non implica l'esistenza di  $e_{ji}$  ma se questo esiste in generale  $c_{ji} \neq c_{ij}$
- NB: un nodo contiene ma non si limita ad essere informazione su uno stato del problema

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. on Systems Science and Cybernetics, SSC-4(2), 1968

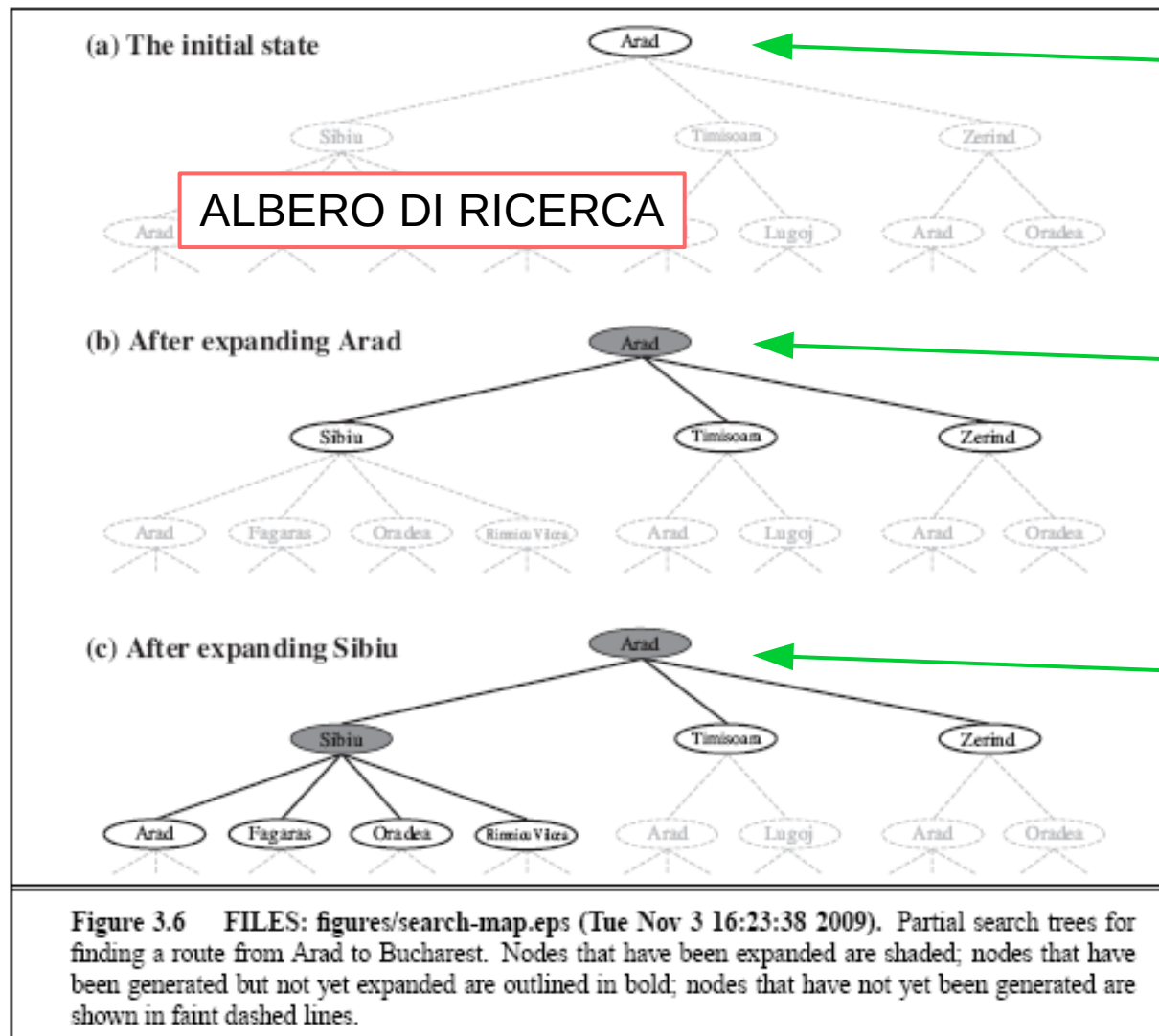


# Stati e nodi



Un **nodo** è una struttura dati che mantiene molte informazioni: (1) lo stato del problema a cui si riferisce; (2) informazioni di struttura (genitore, figli); (3) informazioni di contabilità. Si noti la **direzione degli archi**: da figlio a genitore.

# Metodi di ricerca non informati (blind search)



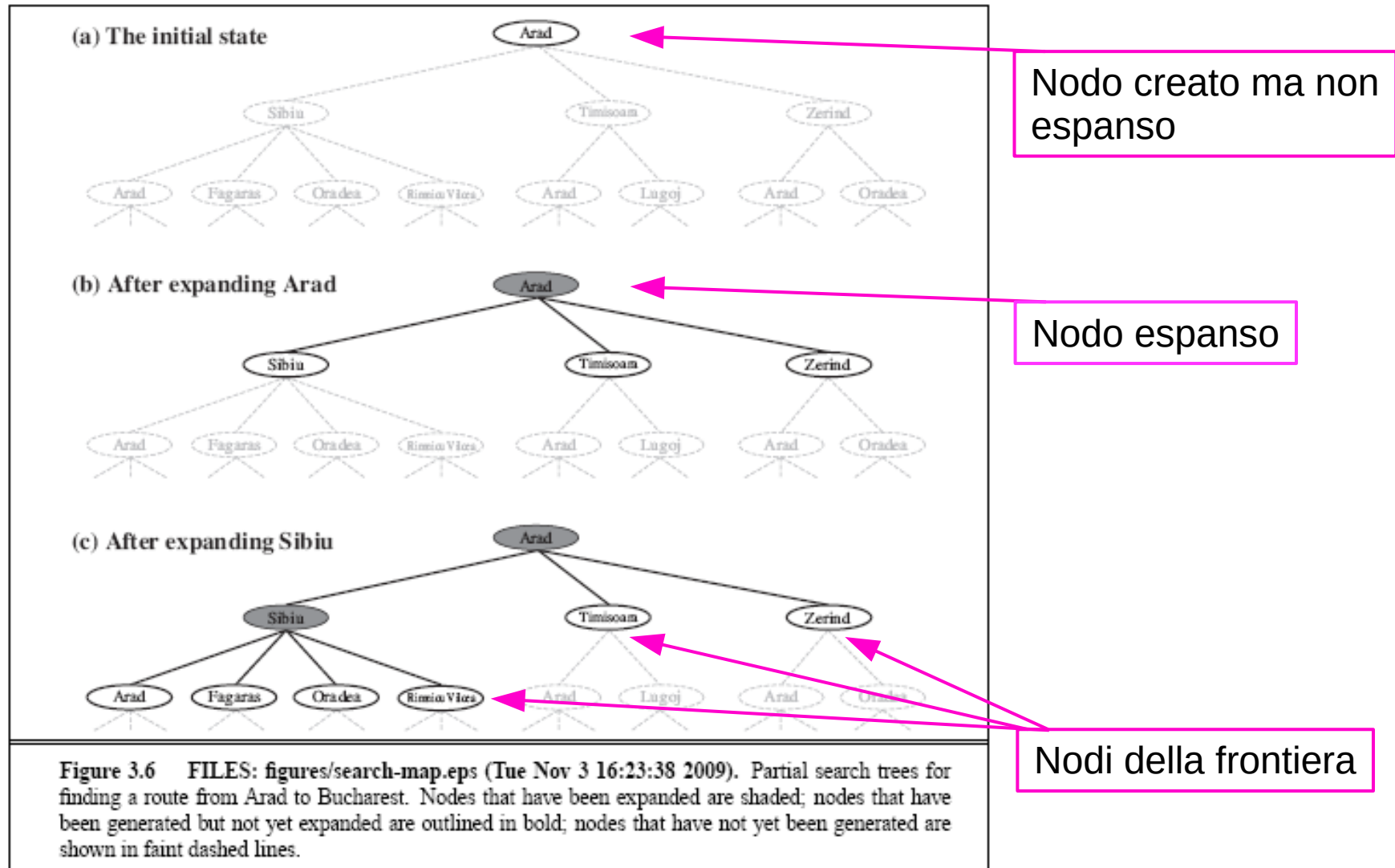
Dato uno stato è possibile identificare le operazioni applicabili

Di conseguenza è possibile identificare i possibili stati successori

E quindi scegliere quali alternative esplorare

NB: le possibili soluzioni sono più di quelle esplorate effettivamente

# Metodi di ricerca non informati (blind search)



# Schema di una strategia di ricerca

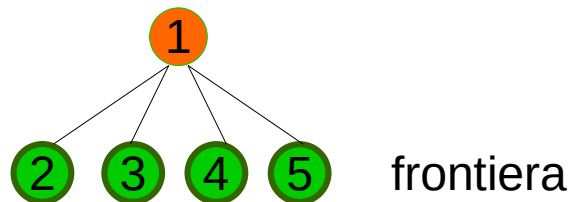
```
function RICERCA-ALBERO(problema) returns una soluzione, o il fallimento
  inizializza la frontiera usando lo stato iniziale di problema
  loop do
    if la frontiera è vuota then return fallimento
    scegli un nodo foglia e rimuovilo dalla frontiera
    if il nodo contiene uno stato obiettivo then return la soluzione corrispondente
    espandi il nodo scelto, aggiungendo i nodi risultanti alla frontiera
```

1

frontiera

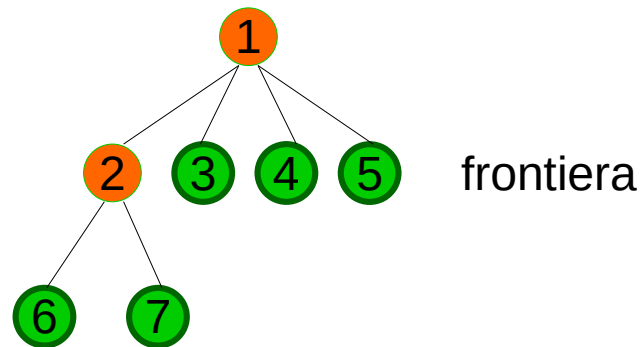
# Schema di una strategia di ricerca

```
function RICERCA-ALBERO(problema) returns una soluzione, o il fallimento  
  inizializza la frontiera usando lo stato iniziale di problema  
  loop do  
    if la frontiera è vuota then return fallimento  
    scegli un nodo foglia e rimuovilo dalla frontiera  
    if il nodo contiene uno stato obiettivo then return la soluzione corrispondente  
    espandi il nodo scelto, aggiungendo i nodi risultanti alla frontiera
```



# Schema di una strategia di ricerca

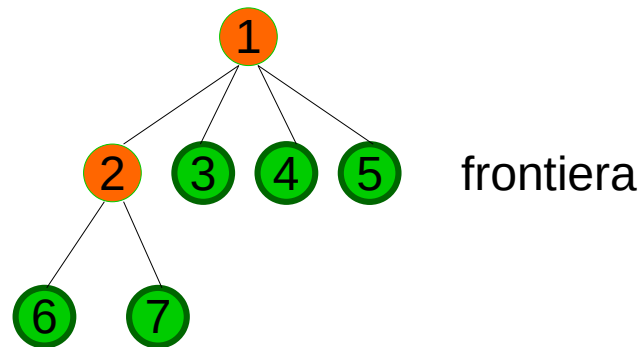
```
function RICERCA-ALBERO(problema) returns una soluzione, o il fallimento  
  inizializza la frontiera usando lo stato iniziale di problema  
  loop do  
    if la frontiera è vuota then return fallimento  
    scegli un nodo foglia e rimuovilo dalla frontiera  
    if il nodo contiene uno stato obiettivo then return la soluzione corrispondente  
    espandi il nodo scelto, aggiungendo i nodi risultanti alla frontiera
```





# Schema di una strategia di ricerca

```
function RICERCA-ALBERO(problema) returns una soluzione, o il fallimento
  inizializza la frontiera usando lo stato iniziale di problema
  loop do
    if la frontiera è vuota then return fallimento
    scegli un nodo foglia e rimuovilo dalla frontiera
    if il nodo contiene uno stato obiettivo then return la soluzione corrispondente
    espandi il nodo scelto, aggiungendo i nodi risultanti alla frontiera
```



Perché scegliere proprio il nodo 2?  
A seconda del modo in cui è gestita la frontiera è  
in realtà possibile realizzare **diverse strategie** di  
ricerca ...

# Confronto delle strategie

- Se le strategie sono tante ve ne è una migliore?  
Come le confronto?
- Criteri di valutazione:
  1. Completezza
  2. Ottimalità
  3. Complessità temporale
  4. Complessità spaziale



# Confronto delle strategie

- Criteri di valutazione:
  1. **Completezza:**  
garanzia di trovare una soluzione, se esiste
  2. **Ottimalità:**  
garanzia di trovare una soluzione ottima (a costo minimo)
  3. **Complessità temporale:**  
quanto tempo occorre per trovare una soluzione
  4. **Complessità spaziale:**  
quanta memoria occorre per effettuare la ricerca

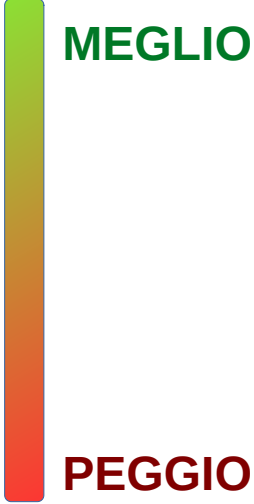
# Come si valuta la complessità?

- **Complessità computazionale:** dato un problema, esistono infiniti algoritmi che lo risolvono, si può dire che uno sia migliore di un altro?
- **Termine di paragone?**
  - **Tempo:** quanto tempo richiede l'esecuzione nel caso (migliore, medio) peggiore?
  - **Spazio:** quanta memoria richiede l'esecuzione nel caso (migliore, medio) peggiore?
- **Criterio di preferenza: economicità**

# Come si valuta la complessità?

- Gli algoritmi non sono programmi: **sono astratti**
- Se anche fossero programmi il tempo di esecuzione dipenderebbe dal calcolatore su cui sono eseguiti. Vogliamo una valutazione che prescindano da questi aspetti.
- **Tempo e spazio non metrici** ma parametrici:
  - e.g. calcolati in termini di **numero di nodi** (di un albero, di un grafo) **creati** o **visitati**
  - Tempo di esecuzione vero: costante ignota per il valore parametrico calcolato (es. numero passi, numero di nodi visitati)
- È interessante l'**andamento del costo** man mano che l'algoritmo viene applicato a strutture sempre più grandi o complesse

# Notazione O-grande

- La notazione O-grande specifica la complessità di un algoritmo, cioè quanto tempo/spazio l'esecuzione richiede in funzione di uno o più parametri, ad es:
    - **$O(1)$** : costante
    - **$O(n)$** : lineare
    - **$O(n^2)$** : quadratica
    - **$O(n^3)$** : cubica
    - **$O(2^n)$** : esponenziale
- 
- $O(f(n))$** :  $f$  è una funzione definita su numeri interi non negativi. Si legge “O-grande di  $f(n)$ ” o “dell’ordine di  $f(n)$ ”

# Notazione O-grande

- **$n$  = taglia dei dati**
- Chiamiamo  **$T(n)$**  il tempo di esecuzione di un programma, espresso in termini di  **$n$**
- **$T(n)$  è  $O(f(n))$**  se esiste un numero naturale  **$n_0$**  e una costante  **$c > 0$**  tale che  **$\forall n > n_0$**  si ha  **$T(n) \leq c \cdot f(n)$**

# Approcci: quale e quanta conoscenza?

## 1) **Approcci blind:**

usano esclusivamente la **struttura del problema** per cercare (e trovare) una soluzione

## 2) **Approcci informati:**

usano la struttura del problema + **ulteriore conoscenza** per guidare la ricerca

a) Monoagente

b) Multiagente (giochi)

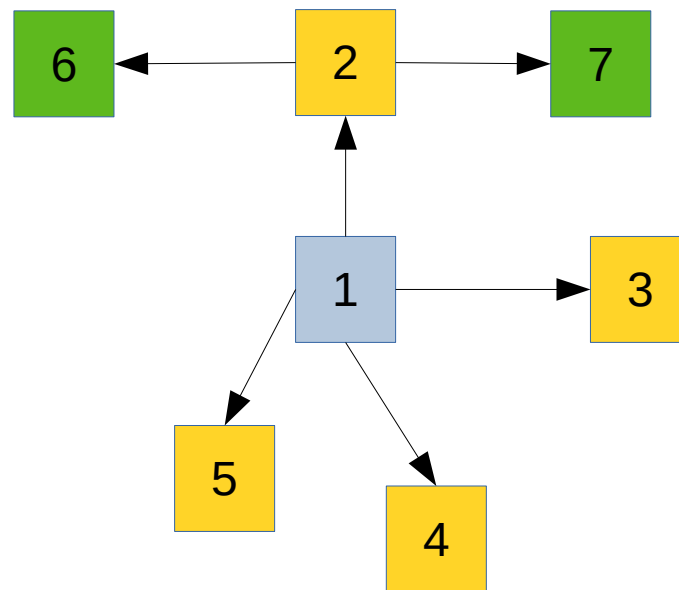
c) Problemi di assegnamento (CSP)

# Lista delle strategie

- Ricerca in ampiezza
- Ricerca a costo uniforme
- Ricerca in profondità (senza backtracking)
  - e variante a profondità limitata
- Iterative deepening
- Ricerca bidirezionale

# 1. Ricerca in ampiezza

- La ricerca espande il nodo radice, poi tutti i suoi successori, poi tutti i discendenti di secondo livello, ecc.





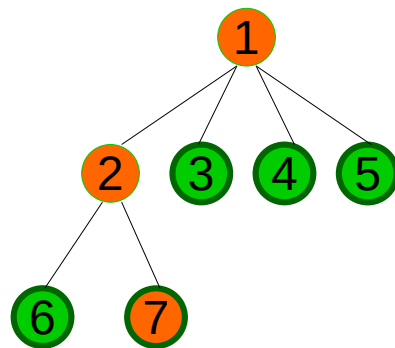
# 1. Ricerca in ampiezza

- La ricerca espande il nodo radice, poi tutti i suoi successori, poi tutti i discendenti di secondo livello, ecc.
- Si realizza gestendo la frontiera come una coda FIFO:



# 1. Ricerca in ampiezza

- Supponiamo che 7 sia il nodo obiettivo, allora la soluzione è catturata dal percorso  $1 \rightarrow 2 \rightarrow 7$
- Tutti i nodi sulla frontiera e tutti i loro antenati vanno tenuti in memoria per poter ricostruire la soluzione quando si trova il nodo obiettivo

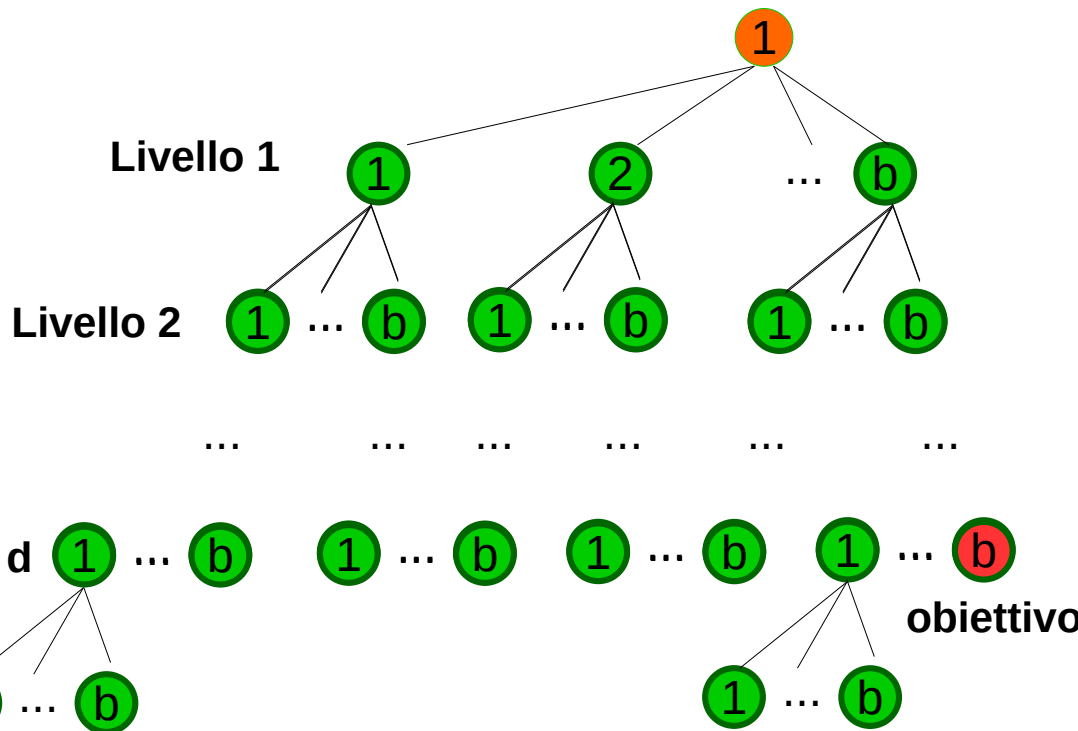


# 1. Ricerca in ampiezza: valutazione

- **Completezza**: se esiste un nodo obiettivo a una profondità finita  $d$ , la ricerca in ampiezza lo troverà a patto che il fattore di ramificazione  $b$  (cioè il numero di figli che un nodo può avere) sia finito
- **Ottimalità**: la soluzione trovata è ottima solo se il costo del cammino è una *funzione monotona crescente della profondità* (es. tutte le azioni hanno lo stesso costo)
- **Complessità temporale**: quanto tempo occorre per trovare una soluzione col crescere dello spazio di ricerca
- **Complessità spaziale**: quanta memoria occorre per effettuare la ricerca col crescere dello spazio di ricerca

# 1. Ricerca in ampiezza: valutazione

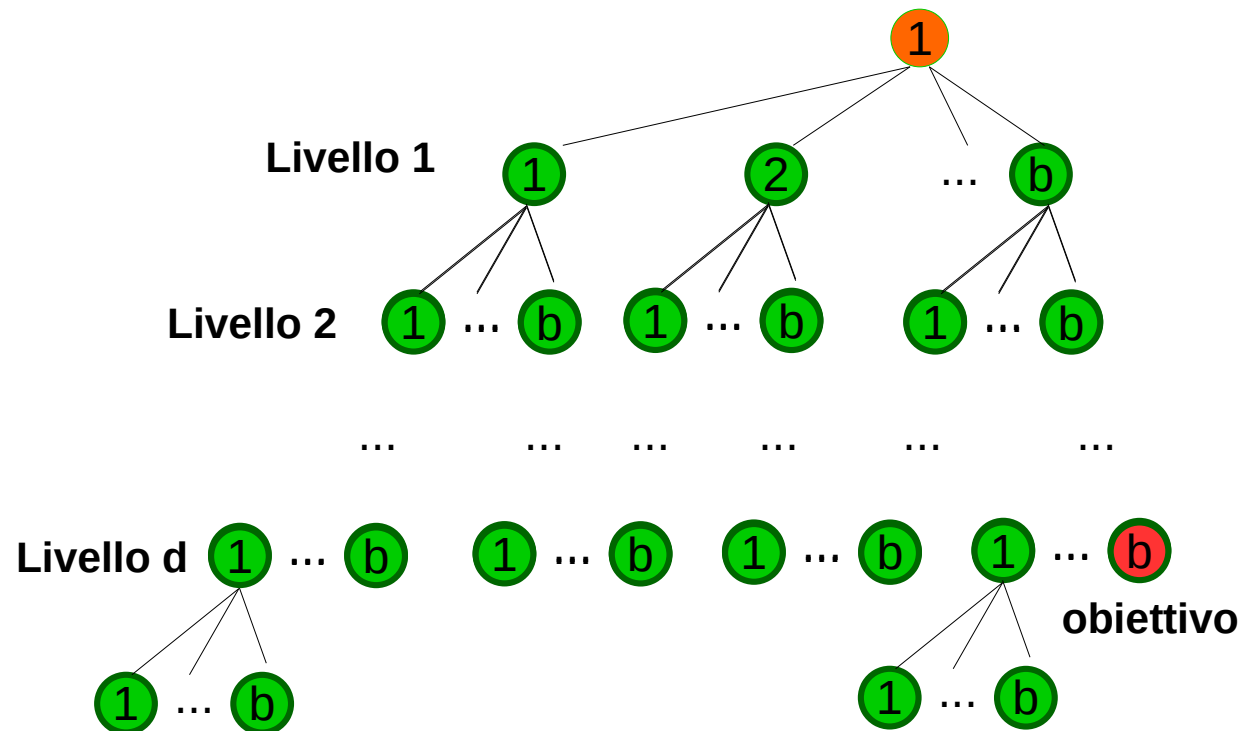
- **Complessità temporale:** di tipo esponenziale  $O(b^{d+1})$



- Il tempo è misurato in termini di **numero di nodi generati**
- Nel caso peggiore ogni nodo avrà **b figli**
- **Livello 1**: b nodi (figli della radice)
- **Livello 2**:  $b^2$  nodi (b nodi per ciascuno di quelli del Livello 1)
- **Livello 3**:  $b^3$  nodi
- ...
- **Livello d** (**livello del nodo obiettivo più vicino**): nel caso peggiore il nodo obiettivo sarà l'ultimo quindi si espandono tutti i nodi del livello d meno 1 producendo  $(b^{d+1} - b)$  nodi.

# 1. Ricerca in ampiezza: valutazione

- **Complessità spaziale:**  $O(b^{d+1})$  perché tutti i nodi della frontiera e tutti i loro antenati vanno mantenuti in memoria



# 1. Ricerca in ampiezza: valutazione

- $O(b^{d+1})$  è bene o male?
- Supponiamo che:
  - il branching factor  $b=10$ ,
  - che vengano generati 10.000 nodi/sec
  - e che 1 nodo occupi 1000 byte

# 1. Ricerca in ampiezza: valutazione

profondità	2	4	6	8	10	12	14
nodi	1100	111.100	$10^7$	$10^9$	$10^{11}$	$10^{13}$	$10^{15}$
tempo	0,11 sec	11 sec	19 min	31 h	129 gg	35 anni	3523 anni
memoria	1MB	106MB	10GB	1TB	101TB	10 peta B	1 exa B

- $O(b^{d+1})$  è bene o male?
- La precedente tabella riporta le misure

# 1. Ricerca in ampiezza: approccio dichiarativo

- **Strategia di ricerca:** indipendente dal problema considerato
- **Descrizione del problema:** conoscenza
- **COSA:** la conoscenza cattura com'è fatto e come funziona il mondo, qual è l'effetto delle azioni sul mondo
- **OBIETTIVO:** compreso nella descrizione del problema (test obiettivo), può essere cambiato senza cambiare il resto