Intelligenza Artificiale

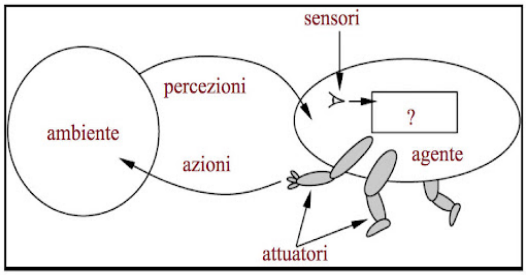
Un approccio moderno

Capitolo 1: Introduzione sulla storia dell'IA

0

Capitolo 2: Gli Agenti

- **Agente**: sistema che percepisce il suo **ambiente** attraverso dei **sensori** e agisce su esso mediante **attuatori**
- Es. Agente Umano: usa i 5 sensi per percepire → mani/gambe/voce per agire
- Es. Agente Robotico: telecamere e sensori → motori per agire
- Es. Agente Software: Input (dati, file…) → Output (scrive file, manda pacchetti in rete...)



Percezione: dati che i sensori percepiscono

Sequenza Percettiva: storico della percezione in tutto il tempo.

Azione→dipende da percezione istantanea o tutta sequenza percettiva, ma mai da ambienti mai percepiti

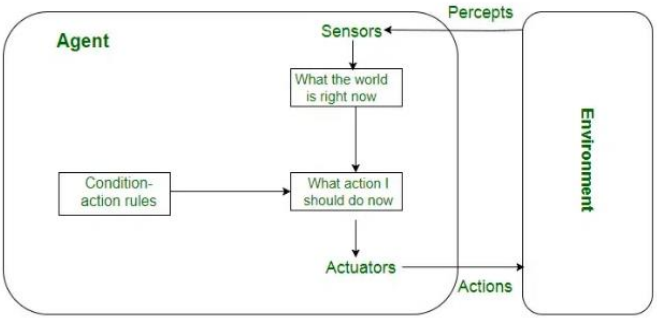
- Comportamento dell'agente è descritto, matematicamente, dalla **funzione agente** (associa tutte le percezioni ad una specifica azione) (f: percezioni → azioni)
- In genere cerco l'agente con le **performance migliori** per la combinazione ambiente – obiettivo.
- Internamente, la funzione agente è implementata concretamente da un **programma agente.**
- La caratterizzazione in agente/non-agente è solo uno strumento usato per descrivere un sistema, a volte ha senso altre no (la calcolatrice visualizza 4 se l'input è 2+2, ma non è una descrizione significativa al fine di capirne il funzionamento)
- **Agente Razionale:** Agente che fa la cosa giusta, ottenendo il miglior risultato o, in condizioni di incertezza, il miglior risultato atteso (togliere la mano dal fuoco generalmente è la l'azione più vantaggiosa, quindi lo faccio.)
- AI → Consequenzialismo → valuto il mio comportamento in base alle conseguenze (percezioni dello stato dell'ambiente successivo alla mia azione)
- Necessaria una **misura di prestazione**, per capire se la conseguenza della mia azione è desiderabile oppure no.
- Umani→ desideri personali, non razionali → non sono misura affidabile
- Regola generale: progettare le misure di prestazione in base all'effetto desiderato sull'ambiente, non su come si vuole che agisca l'agente (aspirapolvere con misura: "pulisci più sporco che puoi" potrebbe rovesciarlo e ri-aspirarlo all'infinito, per soddisfare il vincolo)
- **Razionalità:** dipende da 4 fattori, e ci porta alla def. di agente razionale
    - misura di prestazione che definisce il criterio del successo.
    - la conoscenza pregressa dell'ambiente da parte dell'agente.
    - le azioni che l'agente può effettuare.
    - la sequenza percettiva dell'agente fino all'istante corrente.
- **Definizione** di **Agente Razionale:** per ogni possibile sequenza di percezioni, un agente razionale dovrebbe scegliere un'azione che massimizza il valore atteso della misura di prestazione, date le informazioni fornite dalla sequenza percettiva e da ogni ulteriore conoscenza dell'agente.
- **Agente Omnisciente**: agente che conosce il risultato effettivo delle sue azioni (nella realtà improbabile)

- Es. Se vedo un amico dall'altra parte della strada, e non ho altri impegni, attraverso la strada per andare a salutarlo. Se nel mentre vengo colpito da un portellone di un aeroplano[1] che si è staccato per sbaglio, ho sbagliato ad attraversare e sono quindi irrazionale? (Razionalità vs Perfezione)
- Razionalità → massimizzare il risultato **atteso**
- **Information gathering:** raccogliere informazioni → migliori azioni (guardare prima di attraversare la strada)
- Information gathering presuppone che l'agente possa **apprendere** dalle proprie azioni, auto-modificarsi col tempo, ovvero possiede un grado di **autonomia.**
- In genere, nessun agente è progettato per essere autonomo fin dall'inizio, perché vorrebbe dire che inizierebbe la sua "esistenza" con un comportamento casuale.
- Ambiente Operativo **(Task Environment) → P.E.A.S.** (Performance, Environmen, Actuators, Sensors)
  → necessaria una descrizione dettagliata dell'ambiente e di ciò che ci serve per agire.
- Ci sono vari tipi di Task Environment
    - **Fully** or **Partially** observable: I sensori danno acceso allo stato completo/parziale dell'ambiente in ogni momento, oppure i sensori misurano tutti/alcuni degli aspetti rilevanti per la scelta dell'azione.
      Se sono completamente osservabili l'agente non deve tenere traccia del mondo esterno, vede ed immediatamente agisce (comodo).
      Se l'agente ha i sensori danneggiati o non ottimizzati, potrebbe avere una visione parziale dell'ambiente.
    - **Unobservable** (inosservabile): non ci sono sensori (vedi cap. 4)
    - **Single** or **Multi agent:** se ci sono due entità che puntano a massimizzare una misura di prestazione (es due giocatori di scacchi in partita) allora siamo in un ambiente Multi agente, altrimenti è agente singolo (es macchina autonoma in una strada con ostacoli casuali)
    - **Deterministic** or **Stochastic:** Un ambiente è deterministico se lo stato successivo può essere previsto con certezza dopo un'azione dell'agente. Nel caso contrario è Non-Deterministico.
      Un ambiente è Stocastico se è possibile associare una probabilità agli eventi (es domani piove al 25%, ma non so con certezza cosa accadrà, fa parte dell'ambiente non-deterministico)
    - **Episodic** or **Sequential**: Episodico se l'agente ha una serie di episodi, e decide l'azione in base a quale episodio si presenta. Si usa per eventi sporadici ed indipendenti dallo storico delle azioni.
      Non influenza il futuro (classificatore di email spam).
      Sequenziale se ogni azione può influenzare le successive (giocatore di scacchi, o taxi autonomo)
    - **Static** or **Dynamic**: un ambiente è dinamico se potrebbe cambiare mentre l'agente sta svolgendo l'azione, in caso contrario è statico. È più facile affrontare un ambiente statico di uno dinamico.
      Nel caso di uno dinamico, è necessario porre un limite di tempo al "pensiero" dell'agente, altrimenti l'ambiente cambia e quell'azione potrebbe essere svantaggiosa (Es. taxi autonomo in autostrada)
      Semi-dynamic: se l'ambiente stesso non cambia nel tempo, ma la valutazione della prestazione dell'agente sì (Es. scacchi con orologio).
    - **Discrete** or **Continuous**: distinzione che si applica al modo in cui gestisco lo stato dell'ambiente, il tempo, le percezioni e le azioni. In generale, Discreto = intervalli di tempo / scelte definite e contabili.
      Continuo: Intervalli di tempo/ variabili continue.
    - **Known** or **Unknown** (noto/ignoto): se le "leggi fisiche" dell'ambiente stesso sono conosciute (stato di conoscenza dell'agente, non ambiente)
      In un ambiente ignoto, l'agente dovrà prima apprendere le leggi che lo regolano, per prendere azioni.
      Un ambiente può essere noto, ma parzialmente osservabile (Es. gioco a carte, so che carte ho in mano ma non so le carte degli altri) perciò Noto =! Osservabile
- **Agente** = **Architettura** (dispositivo computazionale con sensori/attuatori) + **Programma** (codice)
- Programma Agente: posso generalizzare il programma come una funzione che associa ad ogni percezione un'azione tramite una *tabella* percezione i → azione j, ma realisticamente è infattibile perché non posso memorizzare ogni singola percezione (es. fotocamera, 2 Mb di spazio al secondo) per poi decidere l'azione a priori nel caso si ripresenti quella specifica percezione.

[1] Nell Henderson, "**NEW DOOR LATCHES URGED FOR BOEING 747 JUMBO JETS**", Washington Post 24 agosto 1989

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

- Ci possono essere 5 tipi di Agenti:
  - **Simple Reflex**



L'agente più semplice: sceglie l'azione in base alla percezione attuale, ignorando lo storico precedente.

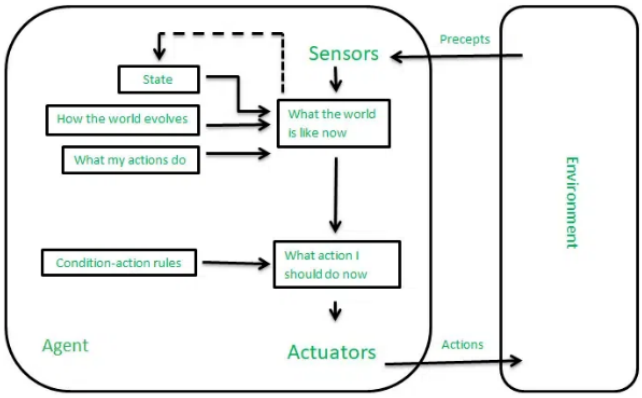
Può essere implementato con le "regole if-then-else".

È necessaria la fully observability → molti sensori

Semplicità → Limitati nel reagire, solo se nella casistica prestabilita

(Es. aspirapolvere semplice, se è sporco dove sono pulisco)

  - **Model-based Reflex Semplici**



Osservabile parzialmente → tengo traccia di ciò che ho visto ma che non vedo ora (**stato interno**)

Sfrutto conoscenza pregressa immagazzinata nel **Modello di Transizione** (conseguenze delle mie azioni) e **Modello Sensoriale** (regole che si celano nell'ambiente) per decidere che azione intraprendere.

Ad ogni azione aggiorno lo stato interno, perciò immagazzino nuove info.

Le **predizioni** possono **non** essere del **tutto corrette** e l'osservazione dell'ambiente potrebbe non essere completa e corretta.

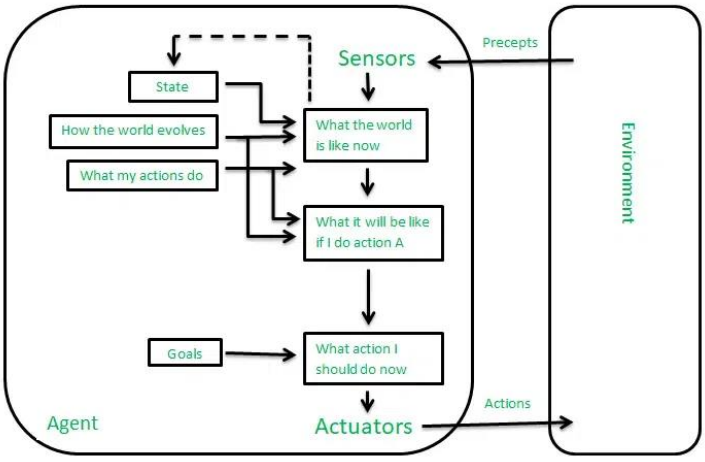
Es. Kalman filter-based localisation (robot con navigazione autonoma):

→Utilizzano un **modello di movimento** (modello di transizione) riesce a predire la propria posizione.

→ Utilizzano un **modello sensoriale** il robot predice e compara lo stato attuale con lo stato corretto ("se sono in questo stato dovrei avere questo tipo di sensazioni").

→ Il robot conosce la propria posizione e riesce a muoversi in una posizione differente.

  - **Goal-based**:



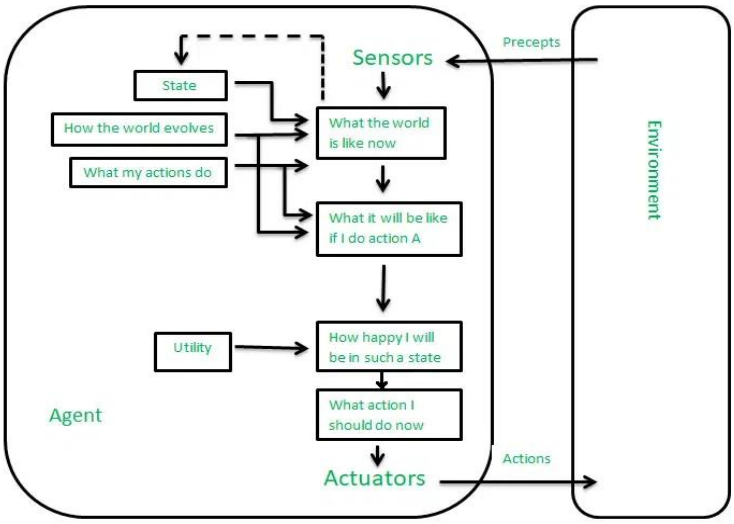
Robot che si basa su rilevazione dell'ambiente, conoscenza delle conseguenze e **persecuzione di obiettivi futuri** (situazioni desiderabili)**.**

Es. Algoritmi di ricerca e pianificazione

Pro: flessibilità nel cambiare obiettivo

Una macchina può frenare perché gli è stato insegnato come riflesso alle luci di frenata di quello davanti, o perché questa è l'unica azione per evitare il tamponamento.

- **Utility-based**:



Evoluzione del goal-based (aggiunge utilità).

Utilità: valore che misura la funzione di utilità, ovvero l'efficienza nel fare l'azione, la prestazione.

Robot che **massimizza l'utilità** nel perseguire il suo obbiettivo ("azione che **in media** darà il risultato più utile").

PRO: con la funzione di utilità può **bilanciare più obbiettivi** incompatibili assieme (es sicurezza e velocità) oppure bilanciare una situazione in cui più obbiettivi sono compatibili ma con **incertezza e/o importanza diversa** (probabilità di riuscita e importanza dell'obbiettivo, es guida autonoma che tiene conto del consumo del carburante)

- **Learning Agent**: agenti capaci di apprendere. Come?
- Idea di Turing: invece di codificare caso per caso, costruisco una macchina in grado di imparare, e insegno.
- Agenti in grado di migliorare costantemente, all'aumentare del tempo di addestramento, capaci di gestire **ambienti inizialmente sconosciuti e/o cambiamenti.**
- Divisibili in 4 componenti astratte:

  →Elemento di **Apprendimento** (learning element): responsabile del miglioramento interno

  →Elemento **Esecutivo** (performance element): responsabile della selezione delle azioni esterne.

  →Elemento **Critico**: valutare i risultati (sensori + standard di prestazione)

  →**Generatore di Problemi**: suggerire delle azioni nuove ed "esplorative". Senza di esso, il robot si concentrerebbe esclusivamente sugli elementi critici, eseguendo sempre azioni che portano a conseguenze già note, perdendo così la possibilità di scoprire **potenziali soluzioni** più efficaci adatte all'ambiente che si trova ad affrontare.

- In pratica: L'elemento di APPRENDIMENTO utilizza le informazioni dell'elemento CRITICO riguardo le prestazioni correnti dell'agente e determina se e come modificare l'elemento ESECUTIVO affinché in futuro si comporti meglio.
- Senza l'elemento Esecutivo il sensore potrebbe dirmi "Una mossa allo scacco matto" ma io non so se è una cosa buona o no senza lo "standard di prestazione" (che ovviamente deve essere fisso, esterno, non modificabile).
- Lo Standard di Prestazione potrebbe essere una **ricompensa** (o una penalità) in base all'esperienza effettuata.
- **Generatore di Problemi:** se continuo a fare gli stessi esperimenti, non arriverò mai a teorie diverse (esempio Galileo: non faceva cadere sassi dalla torre perché fosse utile in quel momento, ma per analizzare situazioni diverse). Es. se freno male quando piove e ogni volta sbatto, il generatore di problemi potrebbe individuare in questo una problematica nel modello e quindi suggerire strade diverse per provare diverse frenate in condizioni differenti.
- **Apprendimento**: processo che modifica ogni suo componente affinché si accordi meglio con l'informazione di feedback disponibile, migliorando cosi le prestazioni globali dell'agente.

- Rappresentazione delle informazioni:

  →Rappresentazione **Atomica**: stato singolo (es nome città), due stati sono confrontabili solo per questo attributo.

  →Rappresentazione **Fattorizzata:** stato descritto da più proprietà (es macchina: posizione GPS, carburante rimanente, velocità…) slegate tra loro.

→Rappresentazione **Strutturata:** stato descritto da proprietà correlate (es camion fermo per mucca sulla strada)

- Tipi di Problema: ci sono 4 insiemi di problemi in base al tipo di task di ambiente

  → Problemi a **Stati Singoli**: tasks ambiente deterministici e completamente osservabili. Noi ci concentriamo qui.

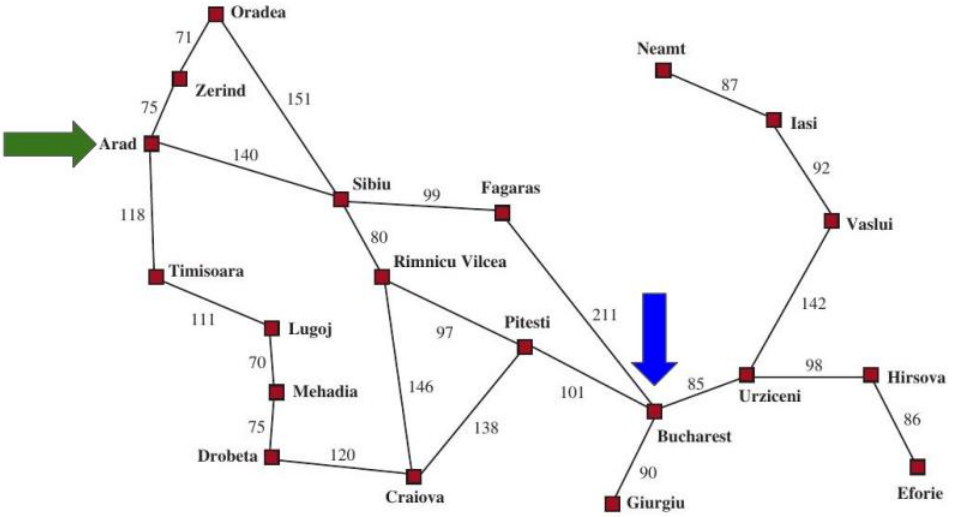
  →Problemi **Conformi (compliant)**: tasks ambiente non deterministici e non osservabili.

  →Problemi di **Contingenza**: tasks ambiente non deterministici e/o parzialmente osservabili.

  →Problemi di **Esplorazione**: tasks ambiente dei quali non si conosce lo spazio degli stati.

Capitolo 3: La ricerca

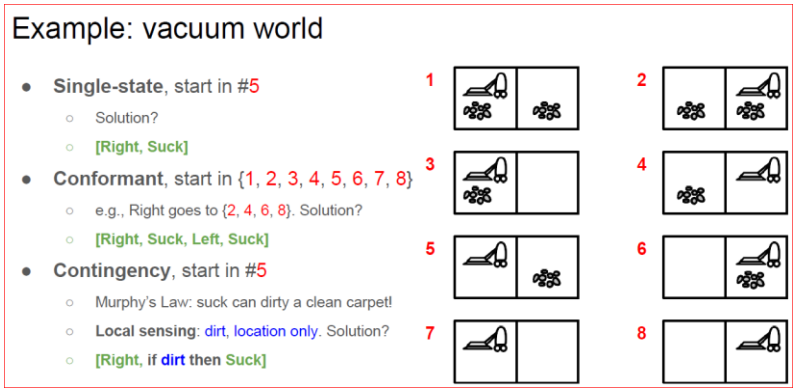
- **Problema RUMENI:** voglio andare da Arad a Bucarest con la strada minore possibile.



- **Problem solving agents:** agenti che risolvono problemi
- Sono un caso speciale degli agenti goal-based (il loro obbiettivo è risolvere un dato problema) con la necessità implicita di un piano ben preciso per operare.
- In genere utilizzano rappresentazioni **atomiche** degli stati (semplicità, sono a Arad, ora sono a Bucarest…) attuando un processo di **rimozione dei dettagli** (es. traffico, incidenti, polizia...) chiamato **Astrazione**.
- La definizione di uno spazio degli stati implica l'astrazione di stati e azioni reali per semplificare la risoluzione di problemi complessi, dove uno stato astratto rappresenta un insieme di stati reali, un'azione astratta corrisponde a un insieme di azioni concrete, e una soluzione astratta consiste in una sequenza di soluzioni reali.
- Come arrivare alla soluzione?
- Se l'agente non ha informazioni (ambiente **ignoto**) dovrà eseguire un'azione a caso
- Se l'agente dispone della mappa (come in questo caso) può risolvere il problema in 4 passi: Formulare l'Obbiettivo (Bucarest), Formulare il Problema (descrizione degli stati e delle azioni utili alla soluzione), Ricerca (simulare le proprie azioni future per trovare la sequenza di azioni che porta ad una soluzione ottimale) ed esecuzione.
- Esecuzione: ad **anello aperto** (senza percezioni) o ad **anello chiuso** (monitorando le percezioni)
- In un ambiente completamente osservabile, deterministico, esiste sempre una **sequenza finita** di passi.
- Definire il Problema:
  →**Spazio degli Stati**: insieme dei possibili stati in cui si può trovare l'ambiente.
  →**Stato Iniziale**: lo stato nel momento di partenza (Arad).
  →Insieme degli **Stati Obbiettivo**: stato nel momento del raggiungimento dell'obbiettivo (Bucarest), può essere uno o possono essere infiniti (tutti gli stati in cui non c'è sporco da nessuna parte, dall'esempio di aspirapolvere)
  →**Azioni** applicabili dall'agente: cambia in base allo stato, es. AZIONI (Arad) = {VersoSibiu, VersoZerind, VersoTim.}
  →**Modello di Transizione**: descrizione delle conseguenze rispetto alle azioni, es RISULTATO(Arad,VersoSibiu)=Sibiu
  →**Funzione di Costo dell'Azione**: costo dell'azione per passare dallo stato s allo stato s' (numero)
- La **soluzione** è una sequenza di azioni (detta **cammino**) che porta dallo stato iniziale ad uno stato obbiettivo.

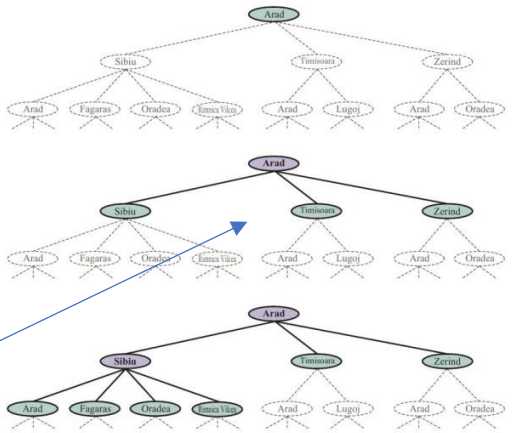
- Assumendo che il costo sia additivo, posso immaginare la soluzione come il cammino con il costo minore.
- Il problema è illustrabile come un grafo, con nodi (le città) collegati da archi pesati (p positivi) non-orientati.

Es. Problema dell'Aspirapolvere
→Quanti stati ci sono? Posso avere Robot in A o B (n caselle = 2), ognuna delle quali può contenere sporco o no, quindi 2 * 2^2 = 8 stati (in generale ci sono n * 2^n stati).
→Qual è lo stato iniziale? Uno qualsiasi degli stati può essere quello iniziale.
→Quali azioni l'agente ha a disposizione? Avevamo definito Sinistra, Destra, Aspira per un mondo a 2 caselle, ma se avessimo una scacchiera dovremmo definire anche Su, Giù (azioni di movimento **assolute**) oppure Avanti,Indietro,Giradestra,GiraSinistra (azioni di movimento egocentriche)
→Modello di Transizione: l'azione Aspira rimuove lo sporco nella casella sottostante all'agente, Avanti muove l'agente avanti nella direzione in cui è orientato a meno che non ci sia un muro, Indietro muove l'agente indietro… GiraDestra e GiraSinistra muovono l'agente di 90°.
→Stati Obbiettivo: tutti gli stati in cui ogni cella è pulita
→Costo di Azione: ogni azione costa uguale, 1.



- **Algoritmi di Ricerca:** input (problema di ricerca) → output (una soluzione o errore)
- Ora ci concentriamo sugli Algoritmi di Ricerca su **Alberi:** nodi (spazio degli Stati) e rami (Azioni). La radice corrisponde invece allo Stato Iniziale.
- L'idea di base degli algoritmi di ricerca sugli alberi è quella di **esplorare tutto lo spazio degli stati** generando man mano nodi associati ciascuno ad uno stato partendo dai nodi già generati.

function Tree-Search(problem, strategy) ritorna una soluzione oppure un errore
inizializzare l'albero di ricerca utilizzando lo stato iniziale del problema
loop do
if non ci sono candidate per l'espansione **then**
return failure
choose a leaf node for expansion according to strategy
if se il nodo contiene lo stato obbiettivo **then**
return la soluzione trovata
else
espandi il nodo e aggiungi il nodo risultato all'albero di ricerca
end



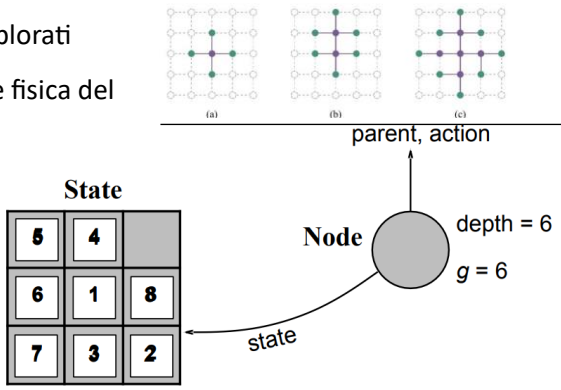
- Concetti chiave degli Algoritmi di Ricerca:

→**Frontiera**: insieme dei nodi non ancora espansi. La frontiera separa i nodi esplorati dai nodi ancora da esplorare.

→(alternativamente): Lista Aperta: frontiera, Lista Chiusa: i nodi già esplorati

→ Uno **Stato** è una rappresentazione di una particolare configurazione fisica del problema, mentre un **Nodo è una struttura dati** che compone un albero di ricerca. Un **Nodo** può includere:

→stato
→riferimento al nodo genitore
→riferimenti (o una lista iterabile con i riferimenti) dei figli
→una azione



→una depth: cioè il numero azioni (rami) dal nodo iniziale al nodo attuale

→un costo: la somma dei costi lungo il percorso per raggiungere tale nodo.

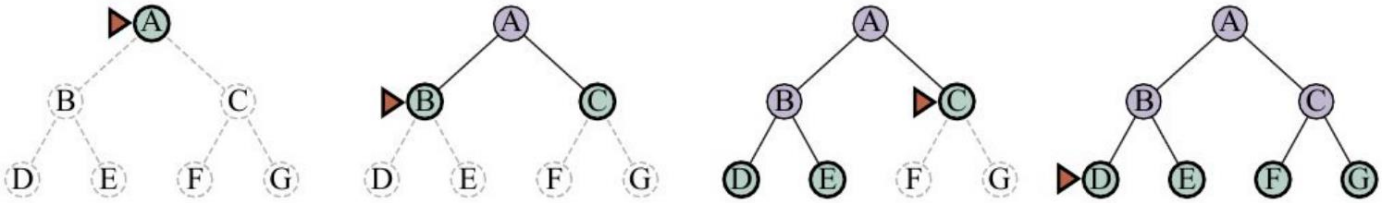
Due nodi diversi possono contenere lo stesso stato.

- **Strategie di Ricerca:** definisce in che direzione e ordine deve avvenire l'espansione dei nodi.
- Sono definite/valutate in base a dei parametri:
  →**Completezza**: l'algoritmo garantisce di trovare una soluzione, se questa esiste, o di riportare correttamente il fallimento, se questa non esiste?
  →**Ottimalità rispetto al costo**: trova sempre la soluzione con il costo di cammino minimo fra tutte?
  →**Complessità temporale:** quanto tempo impiega a trovare (in media) la soluzione? Tempo misurato in secondi o in stati esplorati o in azioni effettuate.
  →**Complessità spaziale:** di quanta memoria necessita per la ricerca?
- Le complessità sono a loro volta definite da:
  →**b =** massimo **fattore di ramificazione (branching factor)** per un albero di ricerca (numero di figli massimo, o numero di azioni massime, per un nodo)
  →**d = profondità della** soluzione con **costo minore**
  →**m = profondità massima** dello **spazio degli stati** (può essere anche infinita)

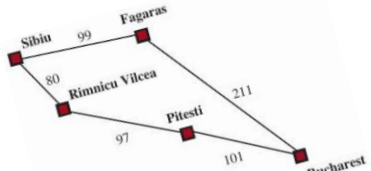
- Algoritmo **Best-First** (MACROCATEGORIA DI ALGORITMI DI RICERCA): scelgo una **funzione di valutazione** f(n) per gli stati, e poi espando il nodo la cui valutazione è migliore (**minimo f(n) tra tutti quelli nella frontiera**). Poi da quel nodo, se corrisponde all'obbiettivo lo restituisco, altrimenti itero ancora. Ad ogni iterazione aggiungo un nodo alla frontiera se non è stato raggiunto in precedenza, o viene aggiunto di nuovo se è stato raggiunto ma con un costo minore.

- Strategie di ricerca non informata: **Breadth-first, Uniform-cost, Depth-first, Depth-limited (**che è la **depth-first search** con una limitazione di altezza massima, non iterativa**), Iterative deepening, Bidirectional**

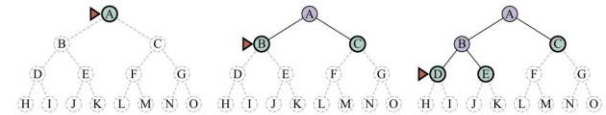
- **Breadth-first search** (ricerca in ampiezza): espando prima tutti i **nodi più in superfice** e poi scendo in profondità. Utile quando tutte le azioni hanno lo stesso costo.
  →Maggiore efficienza se, per la frontiera, viene utilizzata una coda **FIFO** (first-in-first-out): più veloce di una coda con priorità e ci fornisce l'ordine corretto dei nodi: i nodi nuovi (profondità maggiore) vanno in fondo, i nodi vecchi (profondità minore) vengono espansi per primi.
  →La BFS garantisce che, se trova la soluzione, questa è DI SICURO quella più breve (al contrario di DFS che potrebbe trovare una soluzione ma non è detto che sia quella ottimale)
  →Appena trovo il nodo posso fare subito un **test obiettivo anticipato** (ovvero subito), invece che **a posteriori** come con altri algoritmi come best-first che non trovano sempre e subito il percorso più breve.
  →BFS trova il percorso con il minor numero di azioni, perché quando è arrivato a una profondità d, ha già generato tutti i nodi a profondità d-1, quindi se fossero stati soluzione sarebbero stati già trovati e non saremmo arrivati a d.
  Caratteristiche:
  →è **Completa**: se b è finito, esplora sempre tutti i nodi e quindi trova sempre la soluzione
  →**Complessità Temporale O(b^d)**: se ogni stato ha b successori, la radice genera b stati, ognuno dei quali genera b stati, quindi all'aumentare della profondità avremo 1 + b + b*b + b*b*b + … + b^d nodi, ovvero O(b^d).
  →**Complessità Spaziale O(b^d)**: memorizza tutti i nodi, è il problema principale di questo algoritmo (sia temp. Che spazio sono esponenziali, ma in genere 3 ore di tempo non è un problema maggiore di 10 Terabyte di spazio).
  →**Ottimale:** solo se **ogni step** ha **costo 1**, anche se di solito non è così.



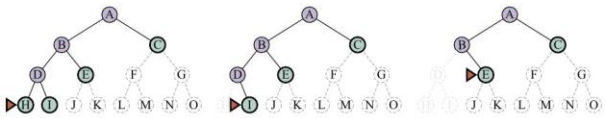
- **Uniform-cost (Algoritmo di Dijkstra)**: a differenza della ricerca per ampiezza, che si diffonde per a ondate di profondità uniforme, l'algoritmo di **ricerca a costo uniforme** si diffonde a ondate di **costo** uniforme.
- In pratica è una ricerca best-first con COSTO-CAMMINO come funzione di valutazione.
- Questo algoritmo si impone di **espandere prima i nodi con costo minore**.
- L'azione di estensione ha lo stesso costo su tutti i nodi, e la frontiera potrebbe essere modellizzata come una **coda ordinata** in base all'**ordine di costo di percorso.**
- Se il costo è unitario, questo algoritmo è equivalente al BFS.
  Caratteristiche:
  →**Completa:** se il costo è ≥ ε (con ε>0), cioè se il percorso diventa sempre più lungo.
  →**Complessità Temporale e Spaziale molto peggiori di O(b^d):** specialmente per step di costo molto ravvicinati (esplora alberi molto lunghi ma con costi bassi, prima di esplorare direttamente un nodo con costo alto, ma che però potrebbe essere l'obbiettivo)
  →**Ottimo**: trova sempre la soluzione con il costo di cammino minimo fra tutte le soluzioni possibili.



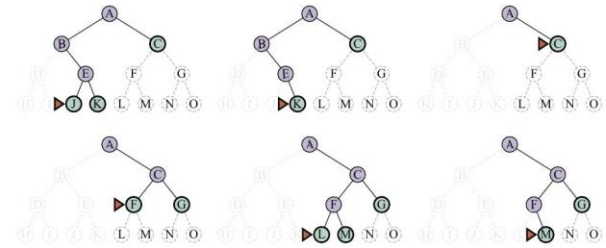
- **Depth-first search** (ricerca in profondità): espande sempre il primo nodo a profondità maggiore (non ancora esplorato) nella frontiera.
- Di solito gestiamo la **Frontiera** utilizzando una **coda LIFO** (stack), che inserisce prima i nodi più profondi e poi quelli meno profondi. Successivamente, estrae i nodi in ordine inverso rispetto a come sono stati inseriti, **espandendo prima i nodi più profondi**."



Frontiera = {**A**} Frontiera = {**B**, C} Frontiera = {**D**, E, C}



Frontiera = {**H**, I, E, C}



Caratteristiche:
→**Non Completa**: Non è detto che esplori tutti i nodi e quindi trovi al 100% la soluzione, per esempio se ho un albero di cui la profondità cresce all'infinito, anche la soluzione è in un nodo di un ramo finito la DFS non la raggiunge mai, oppure potrebbe rimanere bloccata in un loop.
→Strategie per risolvere: evitare gli stati già visitati (no loop) e introdurre una profondità massima (**depth-limited search**)
→**Complessità Temporale: O(b^m)**, terribile se la profondità massima m è molto più grande di d.
→**Complessità Spaziale: O(bm)**, ovvero una ricerca lineare.
→**Non Ottima**: se trova una soluzione, non è detto che sia quella di costo minore (se due nodi portano alla soluzione, la dfs esplora la strada più lunga tra i due)

- Strategie di ricerca informata: