Lezione 4

RICERCA

Stati e ricerca

- Un agente si puo' trovare in uno stato
- Eseguendo determinate azioni, cambia il proprio stato
- Lo scopo e' raggiungere un obbiettivo (goal), caratterizzabile come insieme di stati che verificano determinate condizioni
- Problema
  - cerca una successione di mosse che porti ad uno stato-obbiettivo

Problemi di Ricerca

- Problemi di ricerca = problemi caratterizzabili mediante:
  - Uno ***spazio di ricerca*** (insieme di configurazioni) contenete un sottoinsieme di ***soluzioni*** potenziali
  - Un ***metodo di ricerca***
    - generare o trovare le soluzioni, in modo possibilmente affidabile
    - decidere se una configurazione è una soluzione
- Esempi
  - Trovare la mossa migliore in una partita a scacchi
  - Trovare il percorso minimo in un grafo
  - Trovare una dimostrazione di un teorema

Algoritmi di ricerca

- Ricercano una soluzione (o le soluzioni, o una soluzione ottimale) di un problema di ricerca
  - un agente "computazionale" spesso risolve i problemi in modo diverso da un umano, passando in rassegna tutti i casi ("forza bruta")
- Sono ***algoritmi di base*** per IA
  - Algoritmi di ricerca non informati (o "ciechi") : forza bruta
  - Vi è modo di aggiungere euristiche, algoritmi "informati"

Ricerca e non determinismo

- Spesso un problema di ricerca è formulabile come un problema di implementazione di un algoritmo non deterministico
- Un algoritmo non deterministico ***ND*** è un algoritmo che
  - in ogni stato può scegliere fra più passi elementari;
  - una computazione è una successione di passi elementari; può:
  - portare ad una soluzione, ad un fallimento o essere infinita
  - è il non determinismo "don't know" già incontrato
- ***ND risolve un problema*** se esiste una computazione che porta ad una soluzione (un oracolo può dire all'algoritmo le scelte giuste)
- Non disponendo di un oracolo, si procede con una ricerca esaustiva nell'albero delle computazioni possibili
- Si ricordi: **P** classe dei problemi solubili in tempo polinomiale deterministicamente, **NP** classe dei problemi solubili in tempo polinomiale non deterministicamente, problema aperto **P=NP**?

Esempio

- Cercare il cammino più breve in un grafo
  - Gli stati sono i nodi del grafo;
  - I passi dell'algoritmo sono il passaggio dal nodo corrente ad un nodo collegato da un arco
  - Un oracolo può dire quale sia la scelta giusta
  - L' algoritmo non deterministico è

```
Nodo := nodo di partenza
while not NodoDiArrivo(Nodo)
    {trova arco(Nodo, Nodo1);
    Nodo := Nodo1;}
```

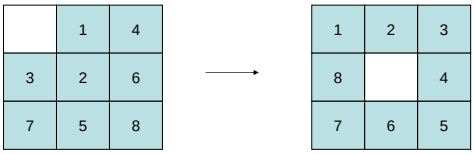
Spazi di ricerca come grafi

- Lo spazio di ricerca è l'insieme delle configurazioni all'interno delle quali avviene la ricerca (i nodi nel precedente esempio), assieme alle relazioni esistenti fra le configurazioni
- Molti problemi di ricerca sono formalizzabili (mediante le opportune astrazioni) come problemi di ricerca in un grafo; le configurazioni sono i nodi del grafo e gli archi rappresentano i possibili passi verso la soluzione. In questo caso:
- Uno spazio di ricerca è un grafo in cui si distinguono un
  - insieme S di ***nodi iniziali*** e un
  - predicato ***goal :*** Nodi → Boolean
- il problema è di trovare i cammini (un cammino, i cammini ottimali, …) dai nodi di S a nodi g in cui valga goal(g)

Si hanno essenzialmente due tipi di spazi:

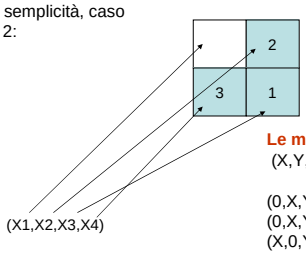
- ***Spazio degli stati:*** i nodi rappresentano stati del "mondo":
  - ad esempio, trovarsi in un nodo di un grafo che rappresenta i collegamenti stradali in un insieme di città, nella ricerca del cammino più breve
- ***Spazio dei problemi:*** i nodi rappresentano ipotesi di soluzione intermedie nella ricerca di una soluzione di un problema
  - ad esempio, gli alberi di prova, eventualmente con delle foglie ancora da risolvere, nella ricerca della dimostrazione di un goal

Esempio di spazio degli stati



Gli stati sono le configurazioni delle tessere; gli archi sono le coppie (config1, config2) tali che si passa da config1 a config2 con una mossa (spostare una tessere nella posizione libera)

CODIFICA DEGLI STATI.
Per semplicità, caso 2 X 2:

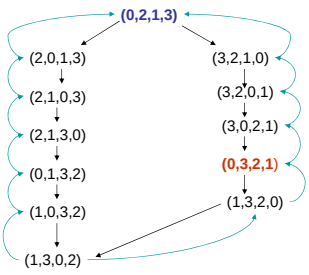


Le mosse possibili, con (X,Y,Z) permutazione di (1,2,3):

(0,X,Y,Z) → (X,0,Y,Z)
(0,X,Y,Z) → (Z,X,Y,0)
(X,0,Y,Z) → (0,X,Y,Z)
(X,0,Y,Z) → (X,Y,0,Z)
(X,Y,0,Z) → (X,0,Y,Z)
(X,Y,0,Z) → (X,Y,Z,0)
(X,Y,Z,0) → (X,Y,0,Z)
(X,Y,Z,0) → (0,X,Y,Z)

STATI INIZIALI: una configurazione
GOAL **(0,1,2,3)** oppure **(0,3,2,1)**

Vediamo uno spazio di ricerca con stato iniziale
(0,2,1,3);
gli archi di ritorno alla mossa precedente, **in verde,** possono essere tagliati

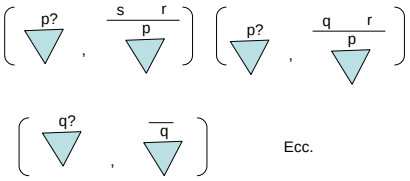


Esempio di spazio dei problemi

p:- s,r.
p :- q,r.
q.
r :- s.
r :- q.

Stati: alberi di prova

Archi:

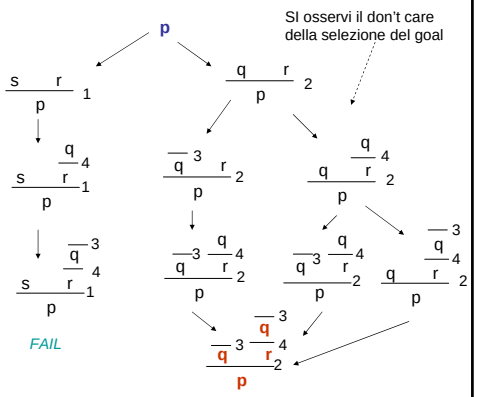


Ecc.

- Stati iniziali: una tesi scelto fra p,q,r,s
- goal(X): vero per X = albero con radice = alla tesi e senza foglie da risolvere

- Lo spazio di ricerca con nodo iniziale p è:

1. p:- s,r.
2. p :- q,r.
3. q.
4. r :- q.



Ricerca in Grafi – Concetti eTerminologia

- ***Grafo Diretto***
  G = (Nodi, Archi) con Archi $\subseteq$ Nodi $\times$ Nodi
- ***Archi labellati:*** λ : Archi $\rightarrow$ Etichette
  - ***Archi pesati:*** le etichette sono numeri, i pesi
- ***Cammini:*** $(n_0,n_1,\ldots n_k)$ tale $(n_i, n_{i+1}) \in$ Archi
  - possono essere infiniti
- ***Grafi Diretti Aciclici (DAG):*** nessun cammino contiene cicli
  - cicli = cammini (n ….. n)

- ***Problema di Ricerca***: R=(G,S,goal), con
  - G = (N, A) grafo
  - S $\subseteq$ N nodi iniziali
  - goal : N → boolean
- ***Soluzione*** = cammino (i,…….,g) con i $\in$ S, goal(g) = true
- ***Costi.***
  - se vi sono dei costi (pesi degli archi), si associa ad ogni nodo n raggiunto con un cammino (i,n1,……nk,n) un costo:
    g(n) = costo(i,n_1) + … + costo(n_k,n)
- ***Soluzioni ottimali*** (se vi è costo):
  - soluzioni il cui costo è <= del costo di ogni altra soluzione

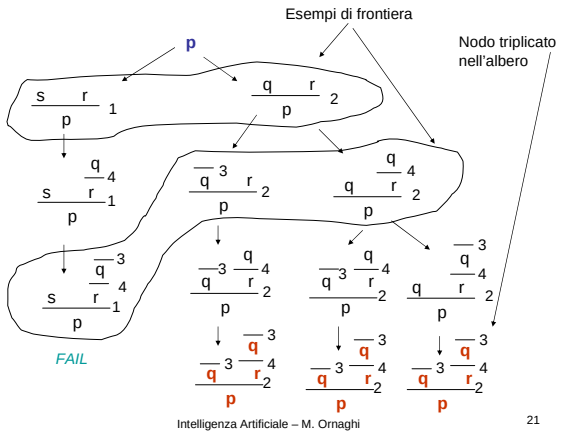
- Considereremo degli spazi in cui il grafo è un DAG.
- Fattori che influenzano la complessità in spazio e tempo:
  - Fattore di ramificazione in avanti (numero archi uscenti)
  - Fattore di ramificazione all'indietro (numero archi entranti)
- ***Caratteristiche di un algoritmo*** (supposto corretto):
  - ***Completezza:*** se vi è una soluzione, viene trovata
  - ***Ottimalità*** (nel caso di costi): le soluzioni trovate sono ottimali
  - ***Complessità*** in tempo e spazio: misurata in funzione del fattore di ramificazione e della profondità massima di soluzione

L'algoritmo generico di ricerca

- Vediamo un programma generico
  - con tipi dati e operazioni aperti, cioé non completamente definiti
- le cui istanze
  - che definiscono le parti aperte
- rappresentano le varie strategie di ricerca che considereremo
- Chiamiamo tale programma l'algoritmo generico di ricerca.
- La nozione centrale è quella di FRONTIERA

Le frontiere dell'albero di ricerca

- Possiamo rappresentare la ricerca come ***albero delle scelte*** di un algoritmo non deterministico, che da ogni nodo sceglie un arco.
- Utilizzando idealmente un albero (ma l'algoritmo lavora sul grafo) consideriamo ***distinti*** nodi che coincidono nel grafo ma che occupano posizioni diverse nell'albero.
- Una ***frontiera*** di tale albero è un insieme di cammini tale che ogni cammino verso un nodo non ancora esplorato estenda uno ed un solo cammino della frontiera
- ESEMPIO:




```
procedure Search(G = (Nodi, Archi), S, goal):  Cammini;
local   frontiera: setOf(Cammini);
begin
  frontiera := S;
  while (frontiera <> {}) do
     togli (s0 ... sk) da frontiera;
     if goal(sk) then
          return (s0 ... sk)
    else
        aggiungi { (s0 ... sk, s) | (sk,s) in Archi }  a  frontiera
  end while
end
```

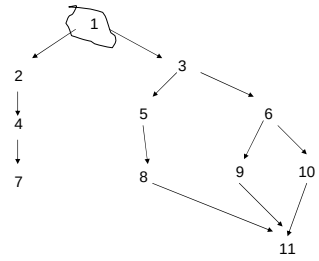

Le procedure aperte

- Dipendono dal problema specifico:
  - *S*, *goal*(n:*Nodi*), *Archi*
- Dipendono dalla strategia di ricerca
  - **togli** cammino **da** frontiera
    - vi sono diversi modi di selezionare il cammino da espandere
  - {(...s_k,s) | (s_k,s) **in** Archi}
    - ordine con cui prendere i diversi archi?
  - **aggiungi** cammino **a** frontiera
    - in che posizione aggiungere?

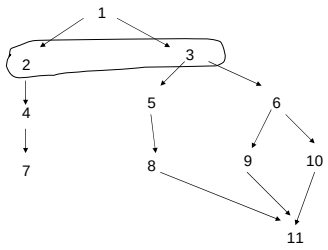
Depth-first e breadth-first con l'algoritmo semplificato

- La ricerca depth-first (in profondità) implementa la frontiera come
  - uno stack;
- mentre breadth-first (in larghezza) la implemeta come
  - una coda
- Vediamo un esempio

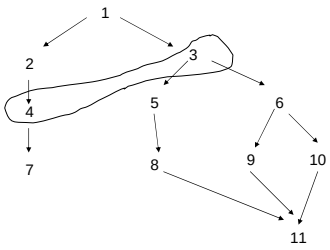
DEPTH FIRST



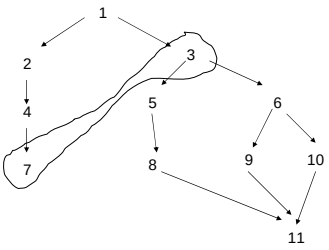
frontiera = {(1)}
tolgo (1);
espansioni di (1):
{(1,2), (1,3)}



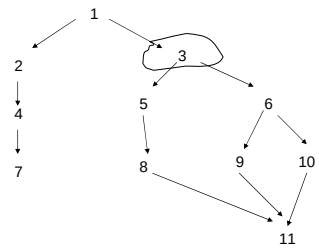
frontiera = {(1,2),(1,3)}
tolgo (1,2)
espansioni:
{(1,2,4)}



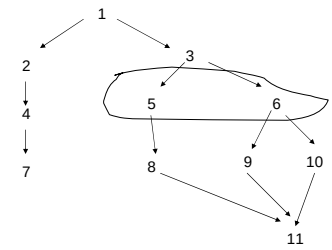
frontiera = {(1,2,4),(1,3)}
tolgo (1,2,4)
espansioni:
{(1,2,4,7)}



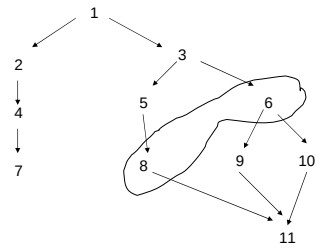
frontiera: {(1,..,7),(1,3)}
tolgo (1,...,7)
espansioni:
{ }



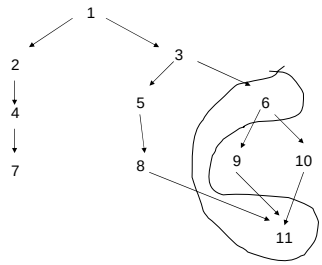
frontiera = {(1,3)}
tolgo (1,3)
espansioni
{(1,3,5),(1,3,6)}



frontiera ...
tolgo ...
espansioni ...

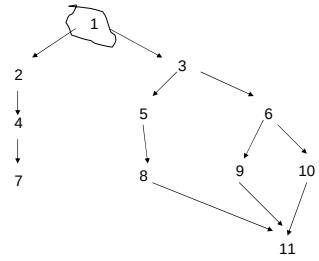


frontiera ...
tolgo ...
espansioni ...

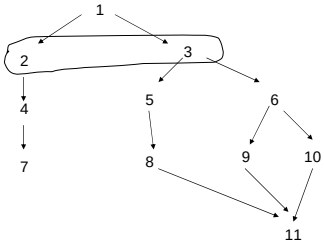


frontiera {(1,..,11),(1,..6)}
tolgo (1,..,11)
goal(11)

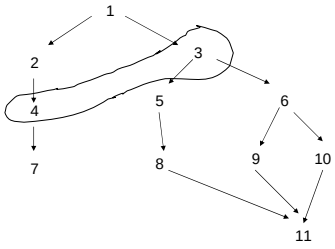
BREADTH FIRST



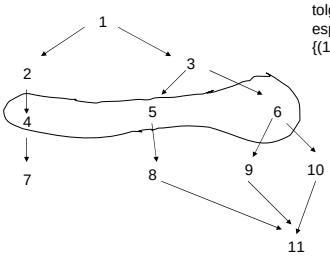
frontiera {(1)}
tolgo (1)
espansioni {(1,2),(1,3)}



frontiera {(1,2),(1,3)}
tolgo (1,2)
espansioni (1,2,4)

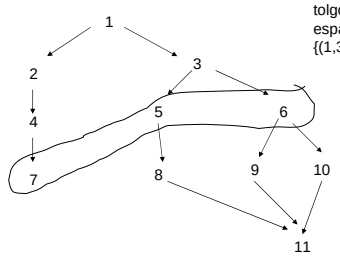


frontiera {(1,3),(1,2,4)}
tolgo (1,3)
espansioni
{(1,3,5),(1,3,6)}

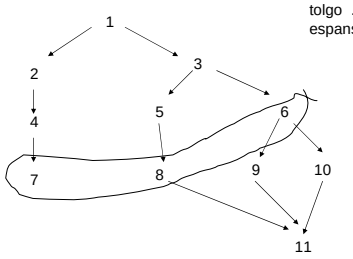


frontiera {(1,2,4), (1,3,5),(1,3,6)}
tolgo (1,2,4)
espansioni
{(1,2,4,7)}

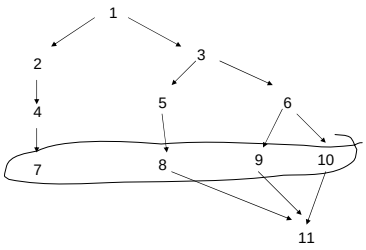
frontiera {(1,3,5),(1,3,6),(1,..,7)}
tolgo (1,3,5)
espansioni
{(1,3,5,8)}



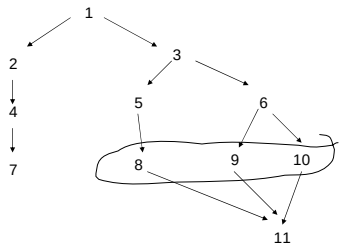
frontiera {(1,3,6),(1,..,7),(1,..,8)}
tolgo ....
espansioni ....



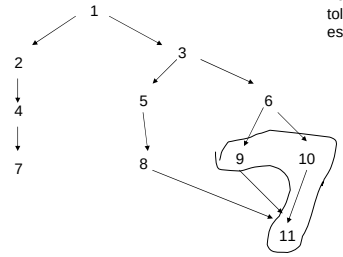
frontiera .....
tolgo ....
espansioni ....



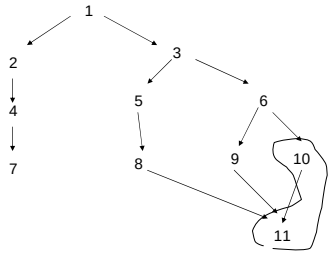
frontiera .....
tolgo ....
espansioni ....

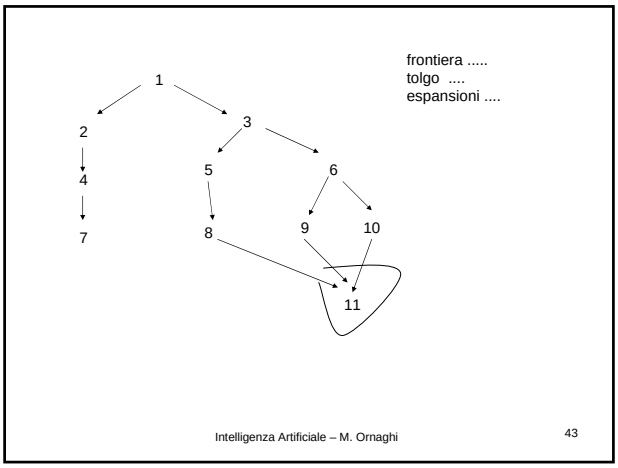


frontiera .....
tolgo ....
espansioni ....



frontiera .....
tolgo ....
espansioni ....




frontiera .....
tolgo ....
espansioni ....
1
2
3
4
5
6
7
8
9
10
11