

Lezione 15

RICERCA I

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 della soluzione mediante **backtracking deterministico**
- 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;}
```

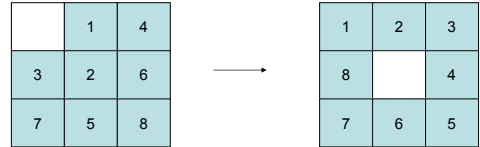
Spazi di ricerca come grafi

- 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 $R(S,G)$ in cui si distinguono un insieme S di **nodi iniziali** e G di **nodi obiettivo** (goals);
 - il problema è di trovare i cammini (un cammino, i cammini ottimali, ...) dai nodi di S a quelli di 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

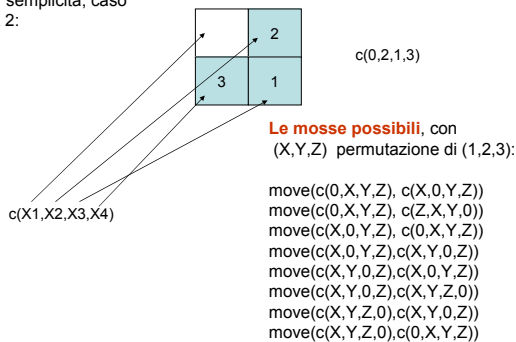
Esempio di spazio degli stati



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 tessera nella posizione libera)

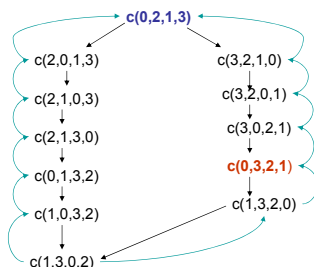
CODIFICA DEGLI STATI.

Per semplicità, caso 2 X 2:



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

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

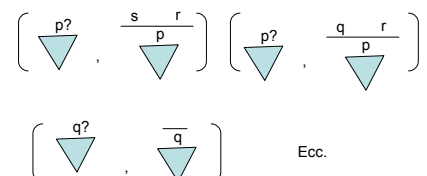


Esempio di spazio dei problemi

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

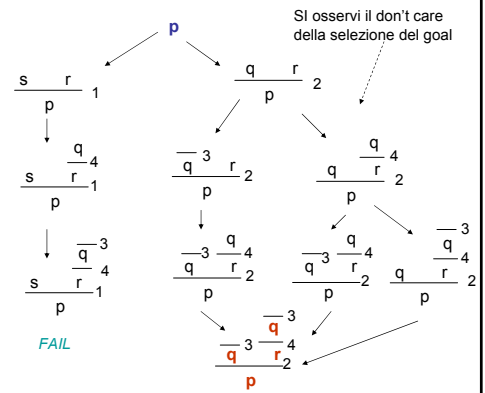
Stati: alberi di prova

Archivi:



- Stati iniziali: un goal scelto fra p,q,r,s
- Stati finali: alberi con radice = al goal 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.



Una rappresentazione degli alberi di prova (le configurazioni)

a(p,?) albero costituito da una foglia non risolta a
a(p,k) albero costituito da una foglia a risolta per il fatto k. a.
a(p,Alb1,...,Albh,k) albero con radice p e sottoalberi Alb1,...,Albh e regola k

Esempio:

$\frac{\overline{q} \ 3 \ r}{p} \ 2$ a(p,a(q,3),a(r,?))

Ricerca in Grafi – Concetti e Terminologia

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

- **Problema di Ricerca**: $R=(D,S,G)$, con
 - $D = (N, A)$ grafo
 - $S \subseteq N$ nodi iniziali
 - $G \subseteq N$ goals (o soluzioni)
- **Soluzione** = cammino $(i, \dots, g)$ con $i \in S, g \in G$
- **Costi**.
 - se vi sono dei costi (pesi degli archi), si associa ad ogni nodo n raggiunto con un cammino $(i, n_1, \dots, n_k, n)$ un costo:
 $g(n) = \text{costo}(i, n_1) + \dots + \text{costo}(n_k, n)$
- **Soluzioni ottimali** (se vi è costo):
 - soluzioni il cui costo è $\leq$ 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

- Abbiamo già parlato di programmi aperti.
- Vediamo un programma logico aperto le cui istanze rappresentano le varie strategie di ricerca che considereremo
- Chiamiamo tale programma l'algoritmo generico di ricerca.
- Partiamo da un algoritmo generico semplificato
- Vediamo poi l'algoritmo generico completo.

La rappresentazione dello spazio di ricerca

- Si fissa l'insieme dei nodi e il modo di rappresentarlo; ad esempio:
 - $p(0,1,3,2)$ ecc. nel problema delle tessere
 - $a(p,a(q,3),a(r,?))$ nel problema degli alberi di prova
- Si rappresenta poi lo spazio di ricerca (il grafo) definendo gli archi, mediante il predicato:

$neighbors(Nodo, Lista)$

dove Lista è la lista dei nodi raggiungibili da Nodo con un arco. Ad esempio:

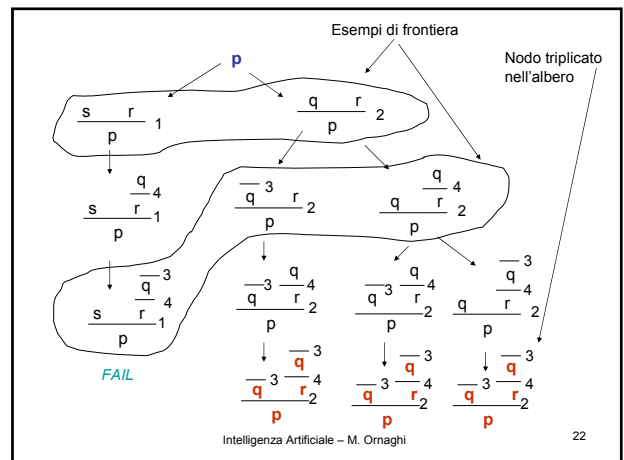
$neighbors(p(0,X,Y,Z), [p(X,0,Y,Z),p(Z,X,Y,0)])$.

$neighbors(p(X,0,Y,Z), [p(0,X,Y,Z), p(X,Y,0,Z)])$.

.....

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 una successione di nodi non imparentati tale che ogni cammino completo contenga uno ed un solo nodo della frontiera.
 - Nodi imparentati: connessi da un cammino nell'albero
 - Cammino completo: dalla radice ad una foglia o infinito
- ESEMPIO:



I predicati dell'algoritmo generico semplificato

- $search(in:F)$
 - Vero se vi è un cammino sino ad un goal da un nodo della frontiera F:ingresso
 - inizialmente F=nodo iniziale, poi ricorsivamente altre frontiere
- $select(in:N,out:F0,out:F1)$
 - seleziona un nodo di F0:ingresso; N,F1:uscite
 - N è un nodo di F0 (il nodo selezionato) e F1 è F0 senza N;
- $is_goal(in:N)$
 - Vero se N è un goal
- $add_to_frontier(in:NL, in:F1, out:F2)$
 - F2:uscita si ottiene aggiungendo i nodi della lista NL:ingresso ad F1:ingresso
- $neighbors(in:N, out:NL)$ già spiegato, rappresenta lo spazio di ricerca

L'algoritmo generico

```
search(F0) :- select(N,F0,F1),
               is_goal(N).
search(F0) :- select(N,F0,F1),
               neighbors(N, NL),
               add_to_frontier(NL,F1,F2),
               search(F2).
```

NB. Ricorsione di coda.

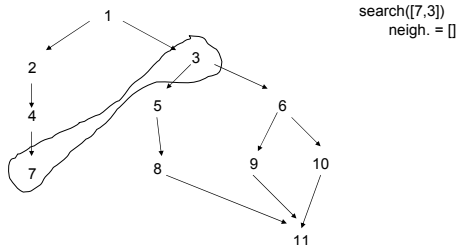
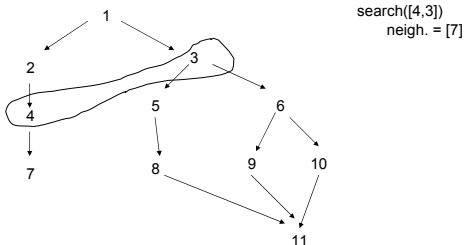
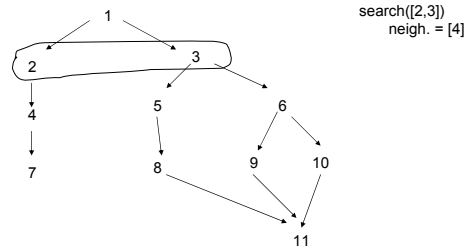
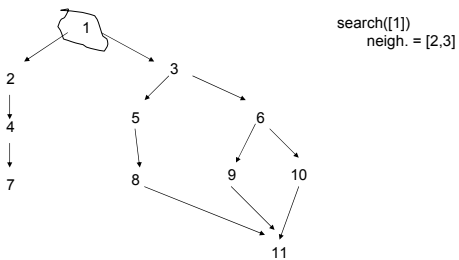
I predicati aperti

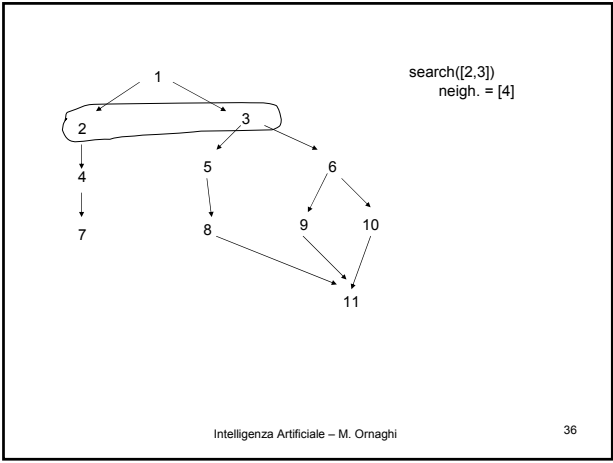
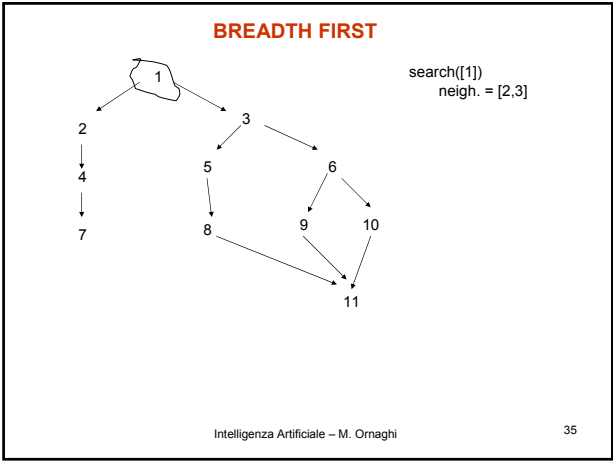
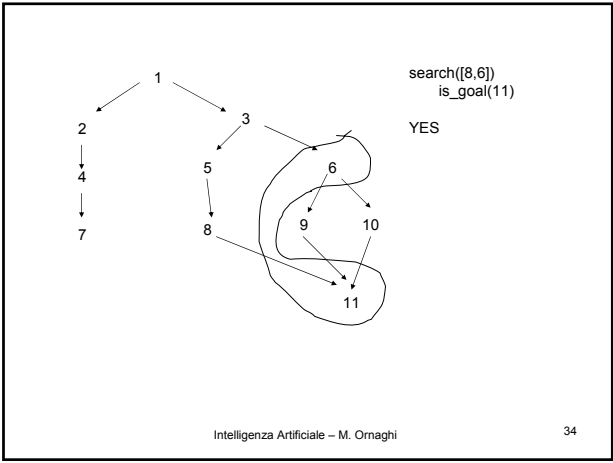
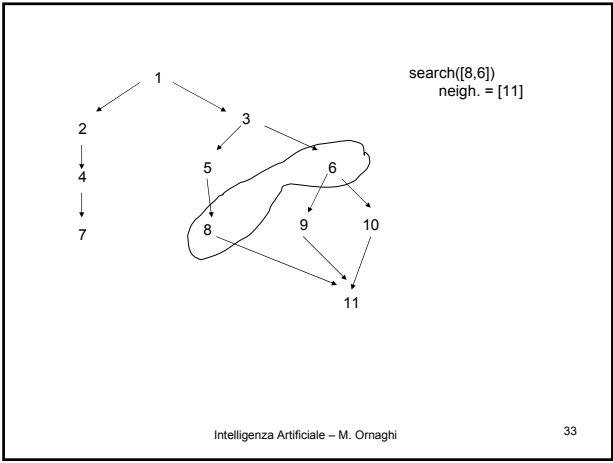
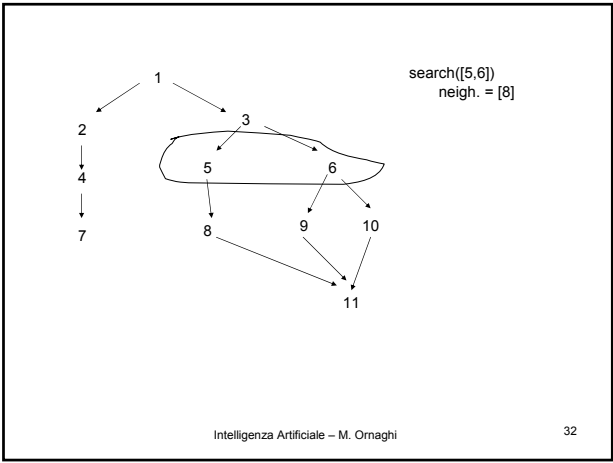
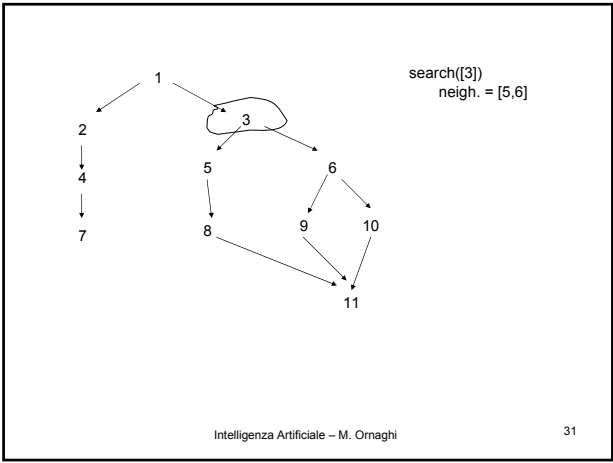
- Problema specifico:
 - `is_goal(N)`, `neighbors(N, NL)` dipendono dallo spazio di ricerca specifico
- Strategia di ricerca
 - `select` dipende dalla strategia di ricerca
 - `add_to_frontier` dipende dalla strategia di ricerca

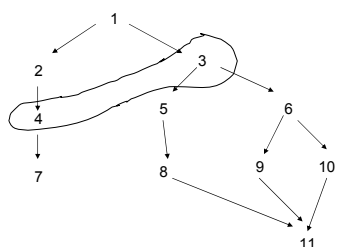
Depth-first e breadth-first con l'algoritmo semplificato

- La ricerca depth-first (in profondità) implementa `add_to_frontier` come segue
`select(N,[N|F],F).`
`add_to_frontier(NN,F1,F2) :- append(NN,F1,F2).`
- mentre breadth-first (in larghezza) lo implementa come:
`select(N,[N|F],F).`
`add_to_frontier(NN,F1,F2) :- append(F1,NN,F2)`
- Vediamo un esempio

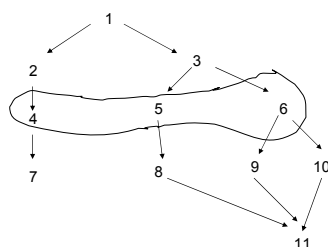
DEPTH FIRST



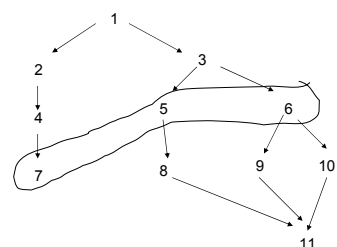




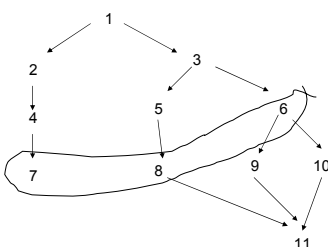
search([3,4])
neigh. = [5,6]



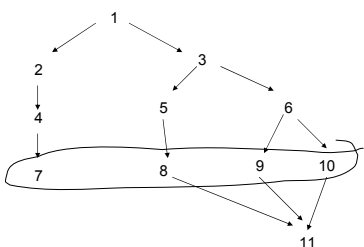
search([4,5,6])
neigh. = [7]



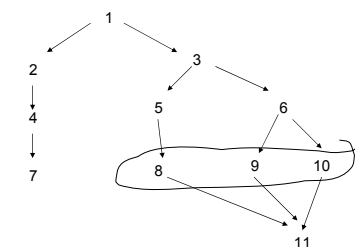
search([5,6,7])
neigh. = [8]



search([6,7,8])
neigh. = [9,10]



search([7,8,9,10])
neigh. = []



search([8,9,10])
neigh. = [11]

