28/03/2024 | LEZ 5

AI: PROBLEM SOLVING E ALGO. RISOLUZIONE

AGENTI COME RISOLUTORI DI PROBLEMI ⇒

AGENTI U OBBIETTIVO
U RAPP. ATOMICA OF STATI

FASI RISOLUZ

1. FORMULAZIONE OBBIETT. (STATO OBBIETTIVO).
2. FORMULAZIONE PROB. → RAPP. STATI, RAPP. AZIONI → ARCHI. → U GRAFI (o TREE)
3. ? SOL. TRAMITE RICERCA. (PATH OF GRAFO)
   - SIMULA NEL SUO MODELLO.
4. EXE SOL.

AMBIENTE DI LAVORO ⇒ AGENTI IN CAP 3

- STATICO
- OSSERVABILE
- DISCRETO → NUM. FINITO DI STATI
- DETERMINISTICO → ESITO AZIONE NOTO E NON INCERTO.
  - CON WORK AT "OCCHI" CHIUSI (PERCEPTION) (QUELLO APERTO)

FORMULAZIONE PROBLEMA

DEF. BY 5 COMPONENTI:

1. STATO INIZIALE s_0 → $s_0 \in S = \{\text{STATI}\}$
2. AZIONI POSSIBILI IN STATO s: AZIONI(s)

③ F. TRANSIZIONE

TR: STATO × AZIONE → STATO

TR(s, a) → s'

N.B. 1, 2, 3 = SPAZIO DEGLI STATI (FINITO)

④ STATI - GOAL

FUNZIONE goal: STATO → {T, F}

· STATO GOAL = s t.c. goal(s) = T

⑤ COSTO:

UALORE

· COSTO CAMMINO OF ACTION/TRANSIZIONE → c(s, a, s')

· ~~A~~ COSTO NEGATIVO

· COSTO TOT = COSTO P SOL + COST ESE SOL.

ALGORITMI DI RICERCA

ALGORITMO DOVE, DA:

PROBLEMA (INPUT) ⟶ PATH GOAL (OUTPUT) $(s_i \to s_{goal})$

EFFICIENZA ALGO

PARAMETRI:

- P SOLUZIONE? → AMMISSIBILITÀ
- COSTO P SOLUZIONE → CREAZIONE ISTANZA SI
- EFFICIENZA SOLUZIONE → VERIFICA ISTANZA.

DA QUI VEDIAMO UN PAIO DI ESEMPI DI

PROBLEMI STANDARDIZZATI ⟹ PROBLEMI "ASTRATTI" USATI COME BASE "TEORICA" PER ALGO.

1° PROBLEMA

ITINERARIO IN CITTÀ

FORMULAZIONE PROBL: SPAZIO AS GRAFO FINITO

CITTÀ PROBLEMA

- STATI = CITTÀ ⇝ ESE. = IN (PITESI)
- S0 = CITTÀ PARTENZA
- AZIONI = MOVE TO CITTÀ COLLEGATE
- TRANSI. = TT (IN (PITESI), GO (BUCAR..)) = IN (BUCAREST).
- COSTO = Σ LENGHT OF STRADE.

2° ESEMPIO

ASPIRAPOLVERE

FORM. PROBLEMA:

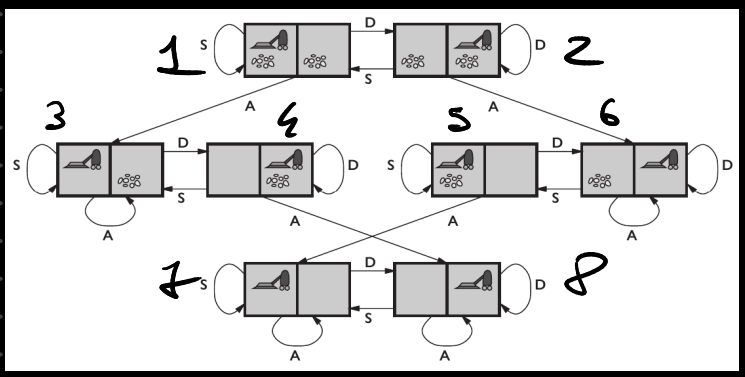
PREMESSA: AMB: 2 STANZE, CHE CAN BE CLEAN VDRT. VACUUM CAN GO IN ANY ROOM

- STATI → ∀ POSSIBILE SCENARIO

$$\underset{\text{N STANZE}}{2} \times \underset{\substack{\text{POSS.}\\ \text{POSIZ.}\\ \text{DI VACCUM}}}{2} \times \underset{\substack{\text{POSS.}\\ \text{DI}\\ \text{DIRT}}}{2} = 8 \text{ STATI}$$

- STATO INIZIALE → CAN BE QUALSIASI STATO.

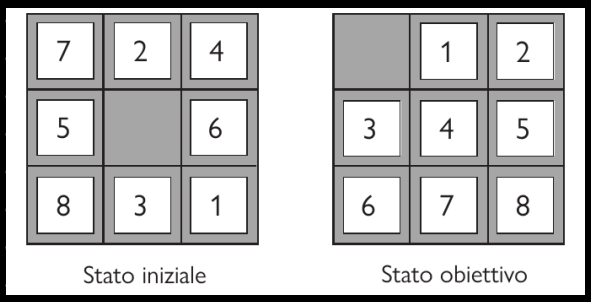
- GOAL → STATO $1 \in \{7, 8\}$ (FIGURA SPAZIO STATI)
- AZIONI → $3 =$ {SINISTRA, DESTRA, ASPIRA}
- TRANSI: TR
- COSTO = OGNI AZIONE COSTA 1



3° PROBLEMA

PUZZLE DELL'OTTO

ESEMPIO



- STATI → ∀ POSSIBILE CONFIGURAZIONE → DISPOS. TASSELLI
- STATO GOAL → 1 SOLO
- AZIONI → MOVE SINGOLA CASELLA BIANCA. $(\uparrow, \rightarrow, \leftarrow, \downarrow)$
- COSTO AZIONE = 1
- SPAZIO STATI = GRAFO CON CICLI ∞
  - ↓
  - ≠ RISPETTO A CAR / VACUUM

(COME IMPOSTARE STATI E AZIONI, CAMBIA ANCHE SPAZIO E PRESTAZIONI)

NB VEDIAMO 1 MODO IMPOSTARE LA POSSE DETERMINA LE PRESTAZIONI.

4° PROBLEM

8 REGINE

- STATI = DISTR. POSS. 8 → POSS. CONFIG. (POS. QUEEN) ↑ NUM. QUEEN.
- STATO INIZ = SCACCHIERA VUOTA
- GOAL = SCHACC. CON 8 REGINE, TUTTE NESSUNA È SOTTO SCACCO.

SEE ≠ FORMUL. OF ACTION ⇒ IMPO SPECIFICARE ACTION GIUSTO (WHY?)

FORMULATION 1

COST ACTION = 0

ACTION = ADD REGINA

INEFFICIENT ... (WHY?) → HAI $64 \times 63 \times \cdots \times 57 \approx 1.8 \cdot 10^{14}$ STATI POSSIBILI DA CONSIDERARE! (TROPPI "NODI")

↳ COSTO P SOL. ALTISSIMO

FORMULAZIONE 2 + BUONA

C(ACTION) = 0

ACTION = ADD QUEEN IN COLONNA + DESTRA, TUTTE SOTTO SCACCO.

WHY BETTER?

HAI 2047 MOSSE DA CONSIDERARE

QUESTE SOL. SONO DETTE SOLUZIONI INCREMENTALI

FORMULAZIONE 3 (STATO COMPLETO)

STATO (NUOVO) = SCACCHI. COMPLETA CON 8 QUEEN. (1 x COLONNA).

ACTION = MUOVE QUEEN IN SUA COLONNA, AVOID SCACCO.

QUINDI DA 1 PROBLEMA POSSIAMO FORMULARE ≠ SOLUZIONI (DI CUI 1 È LA + "EFFICIENTE").

QUI SOTTO DEGLI ESEMPI, INVECE, DI PROBLEMI REALI (O APPLICATI)

- Pianificazione di viaggi aerei
- Problema del commesso viaggiatore (o TSP)
- Configurazione VLSI
- Navigazione di robot
- Progettazione di proteine
- Ranking delle risposte di un search engine
- Il riconoscimento delle categorie grammaticali delle parole in un testo
- La classificazione tematica di tweet o documenti

PROBLEMI DOVE NA SOL. E FORM. PROBLEMA È "SPECIFICO" DEL PROBLEMA STESSO (E + COMPLESSO).

ALGO. RICERCA: TIPOLOGIE

COME DETTO PRIMA:

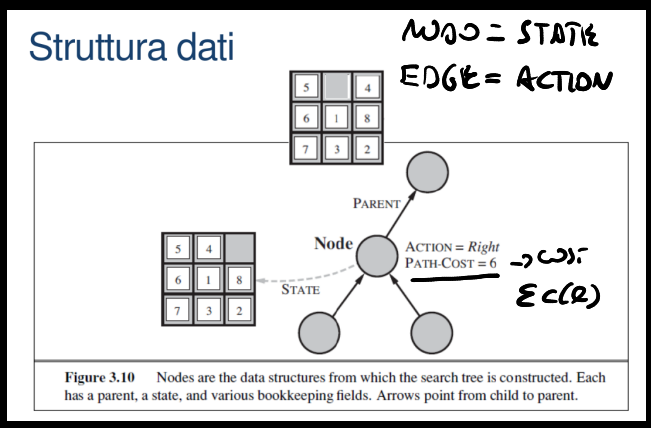
ALGO ρ:

INPUT (GRAFO SPAZIO OF STATI) ⟹ OUTPUT (PATH TO GOAL "MIGLIORE")

NELLA PRATICA:

OUTPUT (ρ DURANTE): CHE ESPANDO DURANTE ρ

CREATO UN ALBERO DI ρ



quindi la ricerca di una soluzione è una ricerca su un grafo che genera un albero di ricerca contenente il path 'corretto'

COME ALGO, CI SERVE CONC. IMPORT. COME:

- NODO → 4 COMP. → M.STATO, M.PADRE, M.AZIONE → X RAGGIUNGERLO, M.COST → COST PER $n_0 \rightsquigarrow n$
  - STRUTTURE DATI
- FRONTIERA → NODI DA ESPLORARE (COME ALGO COME USATO DIFFERENZIA TIPI DI ALGO)

↳ IMPLEMENTABILE COME CODA, / IN:
- → CODA SEMPLICE (FIFO)
- → CODA U PRIORITÀ
- → PILA (LIFO).

ANCHE QUI, ABBIAMO LA COMPL. TIME E SPACE, MA X VALUTARE ALGO, CI VIENE UTILI ALTRI 2 FATTORI:

- COMPLETEZZA → ALGO TROVA O NO LA SOLUZIONE.
- OTTIMALITÀ (AMMISSIBILITÀ): ALGO TROVA SOL U MIN. COST.

N.B. * STRUTTURA DATI.

IN ALGO, I COSTI DI ALGO SUI GRAFI, I PARAMETRI ERANO |V| e |E|, AVENDO GIÀ ESPLICITO L'INTERO GRAFO, MA IN MOLTI PROBL. IL GRAFO DEGLI STATI È IMPLICITO (DEF. BY FORM. PROBLEMA, ϕ S.D.) USIAMO ≠ PARAMETRI DI MISURA:

- b = FATTORE DI RAMIFICAZIONE (NUM. MAX DI SUCCESSORI DI UN NODO)
- d = PROFONDITÀ (LUNGH. CAMMINO $n_0 \rightsquigarrow n_y$ MINIMO)
- m = MAX LENGHT OF PATH GENERICO

VEDREMO UNA SERIE DI ALGO., DOVE ANALIZZIAMO CARATTERISTICHE, CODICE E PRESTAZIONI

ALGO ℘ NON INFORMATA

↝ NON INFORMATO = = ∅ INFO ABOUT DISTANZA GOAL.

1] RICERCA IN AMPIEZZA (SU TREE)

↳ + NULLA COMPLESSO PERCHÉ SPECIFICARLO

O BFS

CARATTERISTICHE

· ESPANDIAMO NODO RADICE

⇓

EXPAND ∧ EXPLORE SUCCESSORI OF ROOT

⇓

" " " SUCCESSORI OF SUCC.

E COSÌ VIA

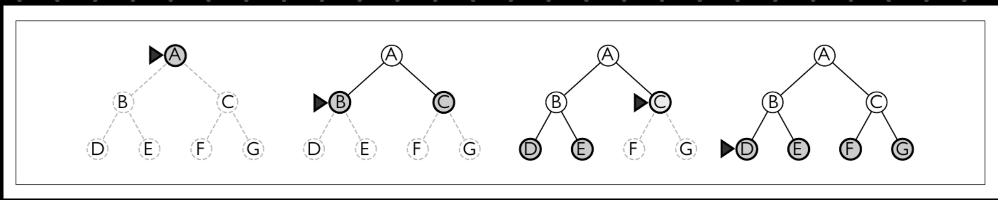


FIGURA ESEMPIO

CODICE

```
function RICERCA-IN-AMPIEZZA(problema) returns un nodo soluzione o fallimento
    nodo ← NODO(problema.STATOINIZIALE)
    if problema.È-OBIETTIVO(nodo.STATO) then return nodo      → GOAL FOUND
    frontiera ← una coda FIFO, con nodo come elemento iniziale   → CODA FE
    raggiunti ← {problema.STATOINIZIALE}
    while not VUOTA?(frontiera) do
        nodo ← POP(frontiera)
        for each figlio in ESPANDI(problema, nodo) do
            s ← figlio.STATO
            if problema.È-OBIETTIVO(s) then return figlio
            if s non è in raggiunti then      → "STATO" RAGGIUNTO, NON NODO!
                aggiungi s a raggiunti
                aggiungi figlio a frontiera
    return fallimento
```

EFFICIENZA

- COMPLETEZZA: SI, SOL ∃ SEMPRE (℘ TUTTI I NODI)
- AMMISSIBILITÀ: SE SUPPONIAMO CHE ∀ NODO HA COSTO OP. $= k \cdot d(n)$ ↳ $d(n)$ = PROFONDITÀ NODO, THEN YES.

CIF ρ NODI IN $d < m$, TUTTI NODI IN $d = m-1$ GIÀ CONTROLLATI.

- COMPLESSITÀ: CON IMPOSTO b = n. succ., d = PROFONDITÀ (LIVELLI)

(QUASI SO CAZZI)

I NODI GENERATI, (QUINDI ANCHE IN MEM)

$$= b^1 + b^2 + b^3 .... + b^d = O(b^d)$$

↳ TIME
↳ SPACE

↓

PIÙ NO 1 = b NODI GENERATI

Profondità	Nodi	Tempo	Memoria
2	110	0,11 ms	107 kilobyte
4	11.100	11 ms	10,6 megabyte
6	10^6	1.1 sec	1 gigabyte
8	10^8	2 min	103 gigabyte
10	10^{10}	3 ore	10 terabyte
12	10^{12}	13 giorni	1 petabyte
14	10^{14}	3,5 anni	10 esabyte

PER CAPIRE QUANTO FA SCHIFO.

2] ALGO A COSTO UNIFORME

CARATTERISTICHE

- VIENE VISITATO (SCELTO) NODO CON COSTO (CAMMINO $N_0 \rightsquigarrow n$) + PICCOLO
- USATO S.D. CODA U PRIORITÀ (PRIORITÀ = COSTO CAMMINO).
- NEL PRATICO È IL BEST-FIRST CON F = COSTO-CAMMINO

CODE

NEXT PAGE

```
function RICERCA-BEST-FIRST(problema, f) returns un nodo soluzione o fallimento
    nodo ← NODO(STATO=problema.STATOINIZIALE)
    frontiera ← una coda con priorità ordinata in base a f, con nodo come elemento iniziale
    raggiunti ← una tabella di lookup, con un elemento con chiave problema.STATOINIZIALE e valore nodo
    while not VUOTA?(frontiera) do
        nodo ← POP(frontiera)
        if problema.È-OBIETTIVO(nodo.STATO) then return nodo
        for each figlio in ESPANDI(problema, nodo) do
            s ← figlio.STATO
            if s non è in raggiunti or figlio.COSTO-CAMMINO < raggiunti[s].COSTO-CAMMINO then
                raggiunti[s] ← figlio
                aggiungi figlio a frontiera
    return fallimento

function ESPANDI(problema, nodo) yields nodi
    s ← nodo.STATO
    for each azione in problema.AZIONI(s) do
        s' ← problema.RISULTATO(s, azione)
        costo ← nodo.COSTO-CAMMINO + problema.COSTO-AZIONE(s, azione, s')
        yield NODO(STATO=s', PADRE=nodo, AZIONE=azione, COSTO-CAMMINO=costo)
```

→ CREO NUOVI NODI DA VISITARE

BEST-FIRST:

VISITO NODO n IN FRONTIERA CON f(n) MIGLIORE

f = COST-PATH(n): COSTO CAMMINO N₀ → n

OTTIMO? SI, A PATTO CHE A NODO ARCHI COSTO $> \varepsilon$, c.c. $\varepsilon > 0$ → ε LOWER BOUND.

(POSSIAMO SCEGLIERE edge CON COSTO + BASSO.

AMMISSIBILE? YES, EXACTLY A PATH.

COSTO? SIA C^* = COSTO PATH OTTIMO.

NEL WORST C., ESPLORA FINO A PROFONDITÀ → $\lfloor \frac{C^*}{\varepsilon} \rfloor$ ~ DISTRIBUISCO IL + POSS. C^* CON ε.

COME IN BFS:

TIME = SPACE = $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$ ~ IF A NODO HA = COSTO, = b^{1+d}

3] ALGO ∂ IN PROFONDITÀ & DFS

CARATTERISTICHE

- USA PILA = FRONTIERA

- EXPLORE NODI U DEPTH MAX, POI "TORNA INDIETRO" AL 1° DISPONIBILE.

CODICE

CODICE BFS MA CON PILA COME FRONTIERA, WITHOUT TAB. NODI RAGGIUNTI.

PRESTAZIONI

- COMPLETO? NO, IN ALCUNI TIPI DI SPAZI (SPAZIO INFINITO DI STATI), PUÒ NON ↗ SOLUZIONE E ↗ ALL'∞.

- CORRETTO? NOP, TAKE 1° SOLUZIONE GIUSTA (∅ ALWAYS OPT).

- COMPLEXITY? MAX PROFONDITÀ (MAX AZIONI)

  - TIME = $O(b^{m+1})$

  - SPACE = $O(bm)$ → O GNI PASSO, ESPANDO b NODI, POI PRENDO IL PRIMO, RICOMINCIO.

UTILE IN DET. CASI (SPAZIO FINITO E ACICLICO), E RISPARMIA TOLTA MEMORIA.

3.5 DFS U BACKTRACKING

CARATTERISTICHE

- DFS BUT U RICORSIONE

- (NUMERO DI ESPANSIONI TUTTI I b NODI, RESPANDO SOLO 1, E CA LA CHIAMATA RICORS.
- STATO NODO SALVATO IN STACK, COSÌ DA "TORNARE" BACK.

CODE

```
function Ricerca-DF-A (problema)
    returns soluzione oppure fallimento
    return Ricerca-DF-ricorsiva( CREANODO(problema.Stato-iniziale), problema)

function Ricerca-DF-ricorsiva(nodo, problema)
    returns soluzione oppure fallimento
    if problema.TESTOBIETTIVO(nodo.Stato) then return Soluzione(nodo)
    else
    for each azione in problema.Azioni(nodo.Stato) do
        figlio = NODO-FIGLIO(problema, nodo, azione)
        risultato = Ricerca-DF-ricorsiva(figlio, problema)
        if risultato ≠ fallimento then return risultato
    return fallimento
```

RIPRENDE SOLO LUI, QUANDO FINISCE RETURN

COMP., CORR: = A DFS

COMPLRX.: TIME = $O(b^{m+1})$

SPACE = $O(m)$.

4] DFS O PROFONDITÀ LIMITATA

CARATTERISTICHE

- DFS MA VIENE SETTATO UN LIMITE DI PROFONDITÀ ℓ.

CODE

DFS MA SI TIENE CONTO DI d, RAGG. ℓ NON SI VA OLTRE

PROPRIETÀ

- COMPLETEZZA: YES IF d OF SPACE IS $< \ell$

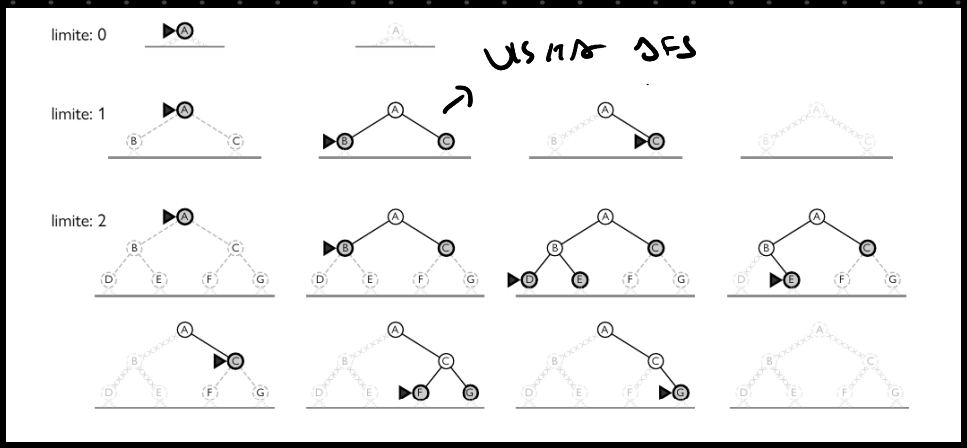
- OTTIMALE: NO, SE SI SCEGLIE MALE ℓ, DFS $\notin$ PSOL.
- COMPLEXITY: TIME $= O(b^e)$

SPACE $= O(b \cdot \ell)$

5] DI APPROFONDIMENTO ITERATIVO

CARATTERISTICHE

- UN MIX TRA BFS E DFS-LIMITATO.
- DFS-ℓ, MA LIVELLO ℓ ++ GRADUALMENTE.
- INIZIO $\ell=0$, POI $\ell=1$ E COSÌ VIA
- ESEMPIO



CODE

```
function RICERCA-APPROFONDIMENTO-ITERATIVO(problema) returns un nodo soluzione o fallimento
    for profondità = 0 to ∞ do
        risultato ← RICERCA-PROFONDITÀ-LIMITATA(problema, profondità)
        if risultato ≠ soglia then return risultato

function RICERCA-PROFONDITÀ-LIMITATA(problema, ℓ) returns un nodo soluzione o fallimento o soglia
    frontiera ← una coda LIFO (stack) con NODO(problema.STATOINIZIALE) come elemento iniziale
    risultato ← fallimento
    while not VUOTA?(frontiera) do
    nodo ← POP(frontiera)
    if problema.È-OBIETTIVO(nodo.STATO) then return nodo
    if PROFONDITÀ(nodo) > ℓ then
        risultato ← soglia
    else if not È-CICLO(nodo) do
        for each figlio in ESPANDI(problema, nodo) do
            aggiungi figlio a frontiera
    return risultato
```

PRESTAZIONI:

- COMPLETE: YES → IN A SP. STATI INFINITI CON CICLO CHECK.
- CORRETT: YES IF A AZIONE HA COST UGUALE.
- COMPLEXITY: SPACE = $O(b \cdot d)$ IF ∃ SOL.
  $O(b \cdot m)$ ELSE
  TIME = $O(b^d)$ ⇝ ANCHE SE "RIVISITO" NODI AI LIVELLI SUP.
  ⇓
  $N(RAI) = (d)b^1 + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d$ ⇝ CON ∃ SOL.
  - $N(RAI)$: NODI GENERATI
  - $(d)b^1$: QUANTE VOLTE HO RIVISITO.

6 ⚲ BIDIREZIONALE

CARATTERISTICHE

- RICERCA IN 2 DIREZIONI: $n_0 \rightsquigarrow n_g$ e $n_g \rightsquigarrow n_0$
- TROVANO SOL. QUANDO FRONTIERE DI COLLIDONO.

TAB. ALGO ⚲ (SU TREE).

Criterio	BF Breadth-first	UC Uniform Cost	DF Depth-first	DL Limited Depth	ID Iterative Depth	Bidir Bidirectional
Completa?	si	si(^)	*no*	si (+)	*si*	si
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Ottimale?	si(*)	si(^)	no	no	si(*)	si

(*) se gli operatori hanno tutti lo stesso costo
(^) per costi degli archi $\geq \varepsilon > 0$
(+) per problemi per cui si conosce un limite alla profondità della soluzione (se $l > d$)

OSS IF NON LISTA "NODI VISITATI" AVOIDING LOOP, DIVERGONO RICERCHE SUI GRAFI.