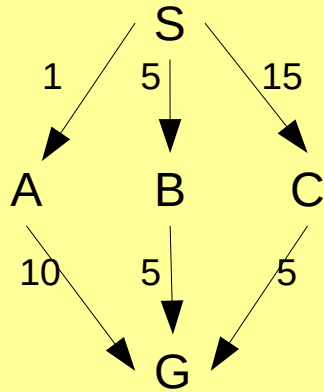


2. Ricerca a costo uniforme

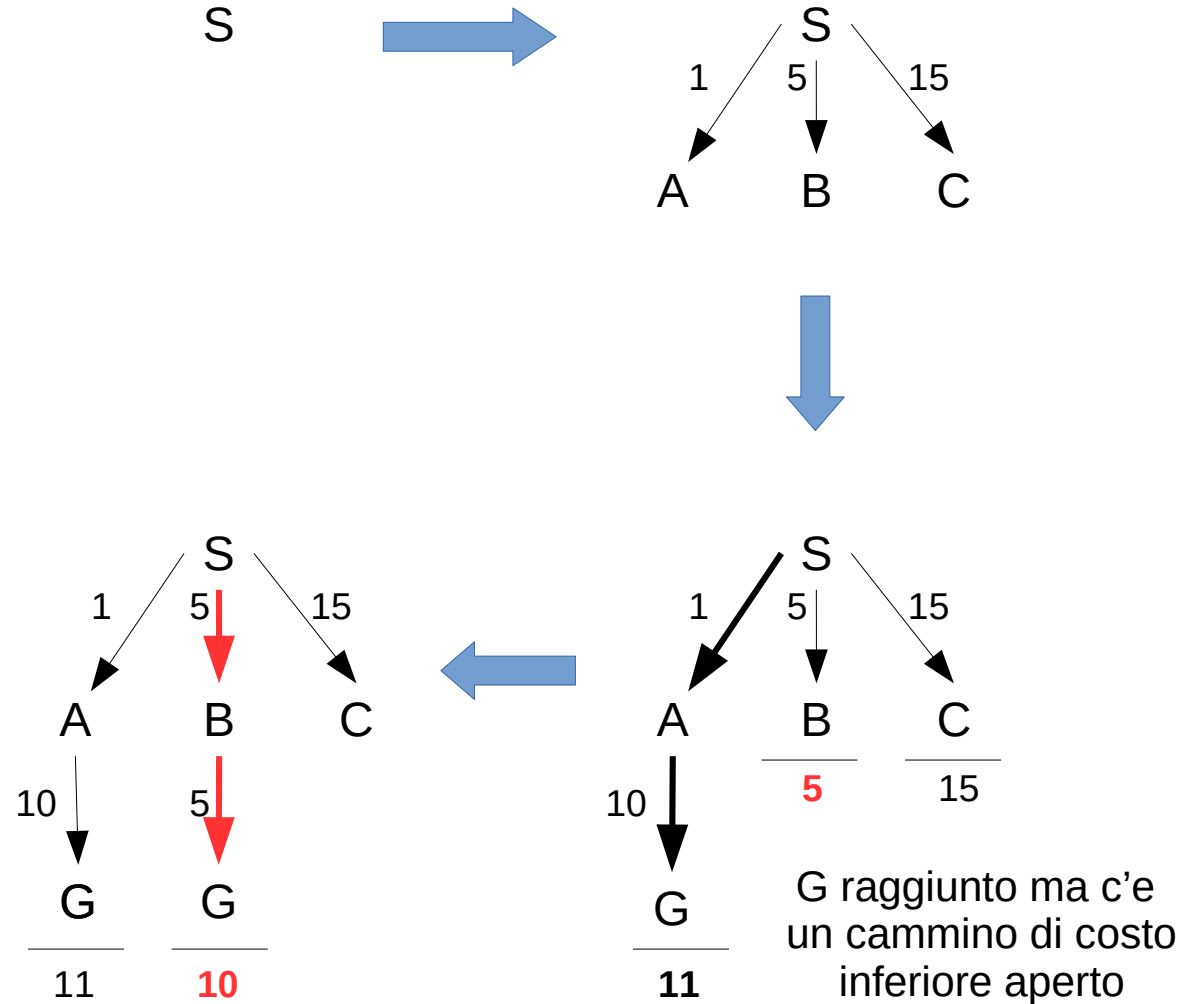
2. Ricerca a costo uniforme

- Quando i costi dei passi **non sono** tutti identici, può essere utile utilizzare la strategie di ricerca a costo uniforme:
 - ogni nodo ha associato il costo del cammino con cui è stato raggiunto
 - la frontiera è mantenuta ordinata
 - a ogni iterazione espande il nodo appartenente a un cammino di costo minimo
- Quando i costi sono tutti uguali diventa la ricerca in ampiezza
- **Attenzione:**
 - **COSTO \neq NUMERO DEI PASSI**
il numero dei passi effettuati non conta, conta solo il costo dei cammini
 - quando trova il nodo obiettivo non si ferma subito, prima controlla se vi sono cammini aperti di costo inferiore e nel caso prova a espanderli

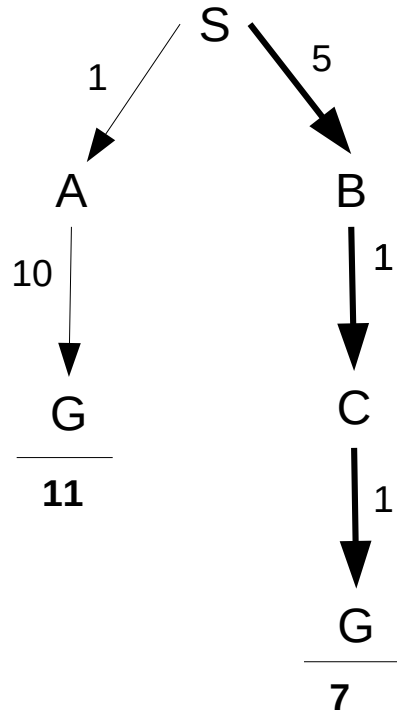
2. Ricerca a costo uniforme, esempio



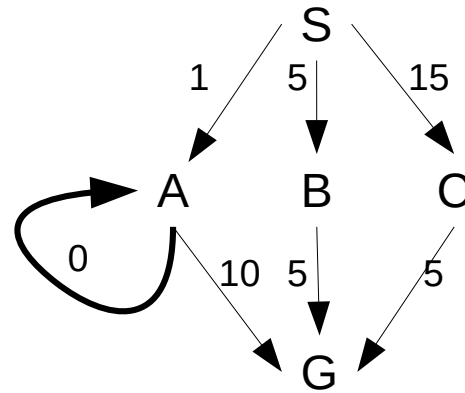
Grafo di transizione completo
(ignoto al risolutore di problemi)



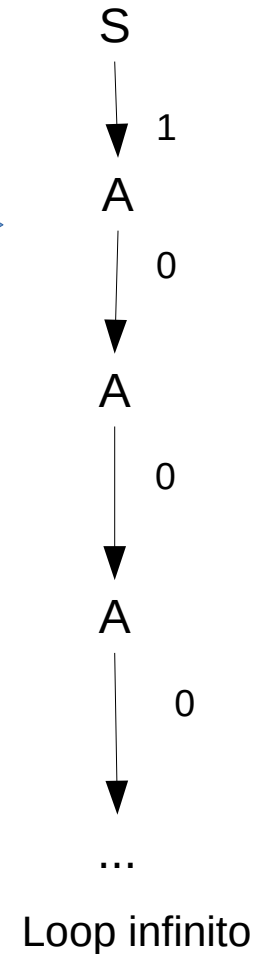
2. Ricerca a costo uniforme, altri esempi



Il percorso a costo minimo ha più passi



Grafo di transizione completo
(ignoto al risolutore di problemi)
**Supponiamo contenga un passo
a costo 0** (esempio noOp)



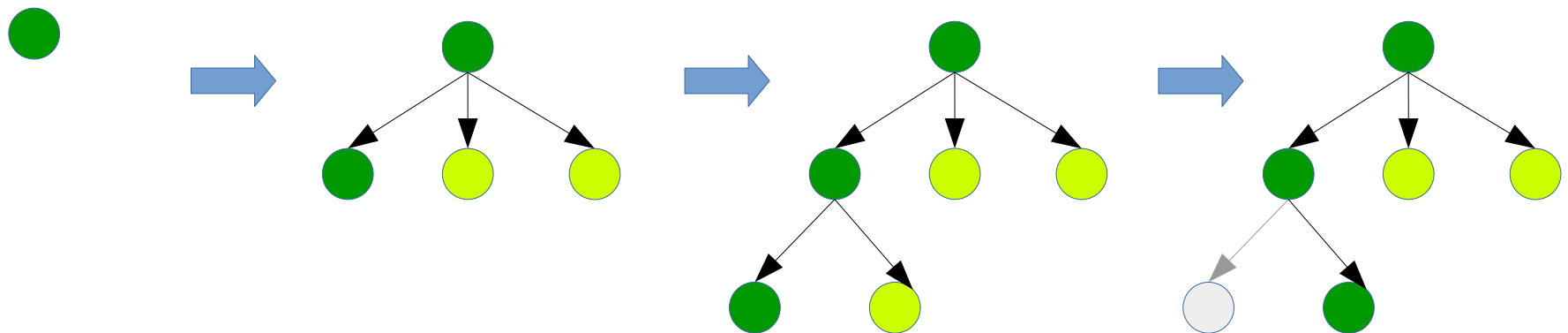
2. Ricerca a costo uniforme: valutazione

- **Completezza:**
garantibile solo se tutti i passi hanno **costo $\geq \epsilon > 0$** .
È il costo minimo delle operazioni
- **Ottimalità:**
 - garantibile solo se tutti i passi hanno **costo $\geq \epsilon > 0$**
MOTIVO: i costi dei cammini aumentano sempre con l'aumentare dei passi e l'algoritmo espande sempre quello attualmente di costo minimo
 - In altri termini: non sa gestire la nozione di guadagno unita a quella di costo
- **Complessità temporale e spaziale:**
 - **indipendenti da branching factor e profondità**; nel caso peggiore, detto C^* il costo della soluzione ottima ed ϵ costo minimo delle azioni avremo $O(b^{1 + \lfloor C^*/\epsilon \rfloor})$
 - In parole è dell'ordine del branching factor elevato al numero di passi del percorso ottimale se i costi dei passi fossero uniformi
 - Quando i costi sono tutti uguali diventa come per la *visita in ampiezza*

3. Ricerca in profondità senza backtracking

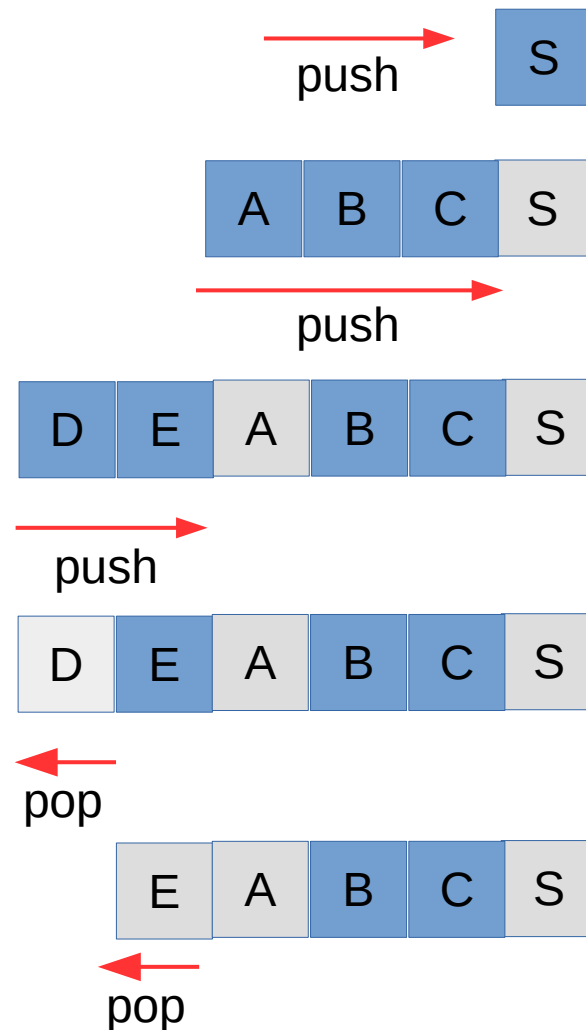
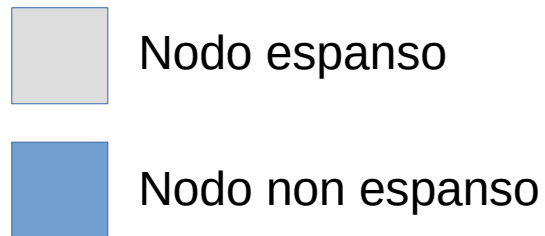
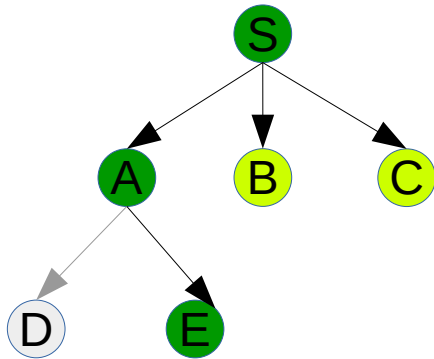
3. Ricerca in profondità senza backtracking

- La ricerca espande sempre uno dei **nodi più profondi** della frontiera, cioè uno dei più lontani dalla radice
- L'espansione produce tutti i successori di un nodo
- Quando la ricerca tenta di espandere un nodo che non ha successori, l'effetto è che il nodo viene rimosso e si “torna indietro” nell'albero per esplorare eventuali alternative



3. Ricerca in profondità

La frontiera è gestita come un coda LIFO, cioè come uno stack:



eccetera ...

3. Ricerca in profondità, valutazione

- **Complessità spaziale: $O(b*m)$** , dove:
 - b = branching factor
 - m = profondità massima
 - Infatti occorre mantenere tutti i nodi del cammino esplorato più tutti i loro fratelli
- Nel caso peggiore occorrerà mantenere l'informazione relativa a un numero di nodi pari a: $b * m + 1$
- **Confronto con ricerca in ampiezza, esempio di slide 47:**
di gran lunga migliore, se $d = 12$, basterebbero 118 KB invece di 10 peta B

3. Ricerca in profondità, valutazione

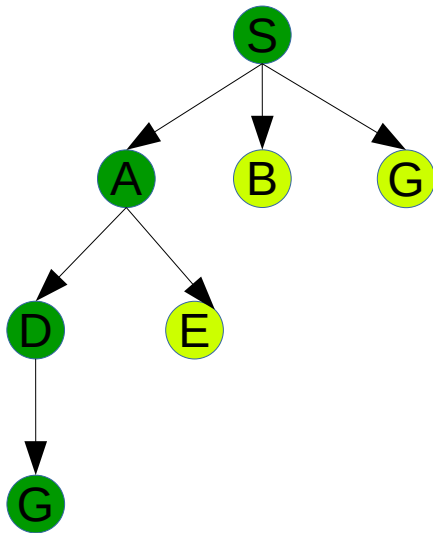
- **Complessità spaziale:**
- nella variante **con backtracking**, un nodo non viene espanso totalmente ma viene generato solo uno dei suoi possibili successori
- Se il cammino generato non porta alla soluzione si tornerà indietro e arrivati al nodo si genererà un altro dei suoi possibili successori
- Quindi la complessità spaziale diventa **$O(m)$**

3. Ricerca in profondità, valutazione

- **Complessità temporale:**
- Nel caso peggiore vengono percorsi tutti i nodi dell'albero
- Supponendo che b sia il branching factor medio e m la profondità massima la complessità temporale nel caso peggiore è $O(b^m)$

3. Ricerca in profondità, valutazione

- **Completezza**: garantibile solo se tutti i cammini sono finiti
- **Ottimalità**: in generale non è garantita, esempio:



Supponiamo che i passi abbiano lo stesso costo

La ricerca restituisce la soluzione:

$S \rightarrow A \rightarrow D \rightarrow G$

Invece della soluzione ottimale

$S \rightarrow G$

3. Variante: Ricerca a profondità limitata

- **Per evitare che la ricerca entri in percorsi infiniti** che non conducono alla soluzione, viene introdotto un limite artificiale, l :
 - Un nodo viene espanso solo se la sua **profondità $p \leq l$**
 - Altrimenti viene trattato come un nodo privo di successori
- Tutti i cammini saranno lunghi al più l
- **Problemi:**
 - Si *riduce la completezza* perché la profondità dell'obiettivo è ignota a priori, quindi potrebbe essere superiore a l
 - Se invece $l \gg d$, profondità minima dell'obiettivo, si *perde efficienza*

4. Iterative deepening depth first search

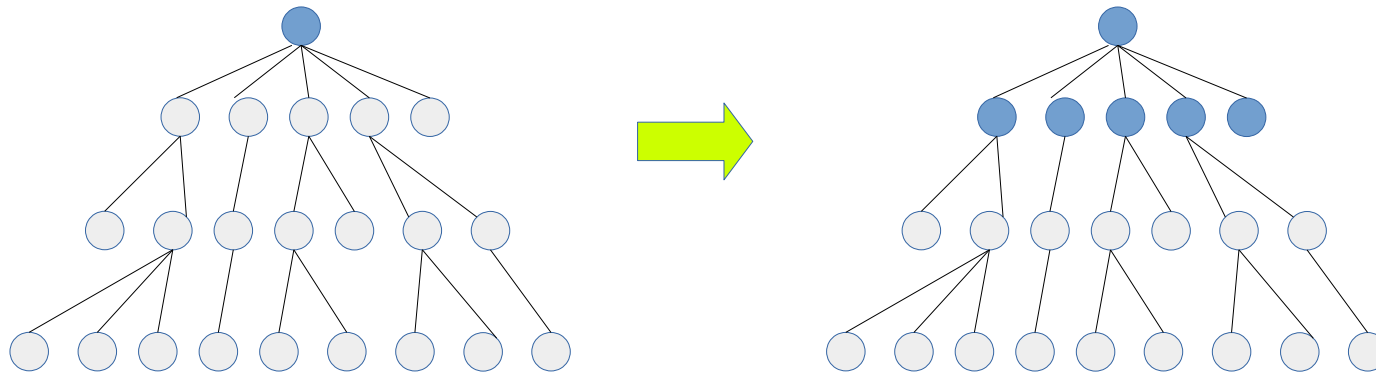
4. Iterative deepening depth first search

- **Esegue una ricerca in profondità limitata, con iterazioni successive in cui la profondità massima viene aumentata via via**
- Si noti che a ogni iterazione l'albero di ricerca viene ricostruito da zero
- L'indicazione "taglio" è la convenzione per dire che la ricerca è fallita

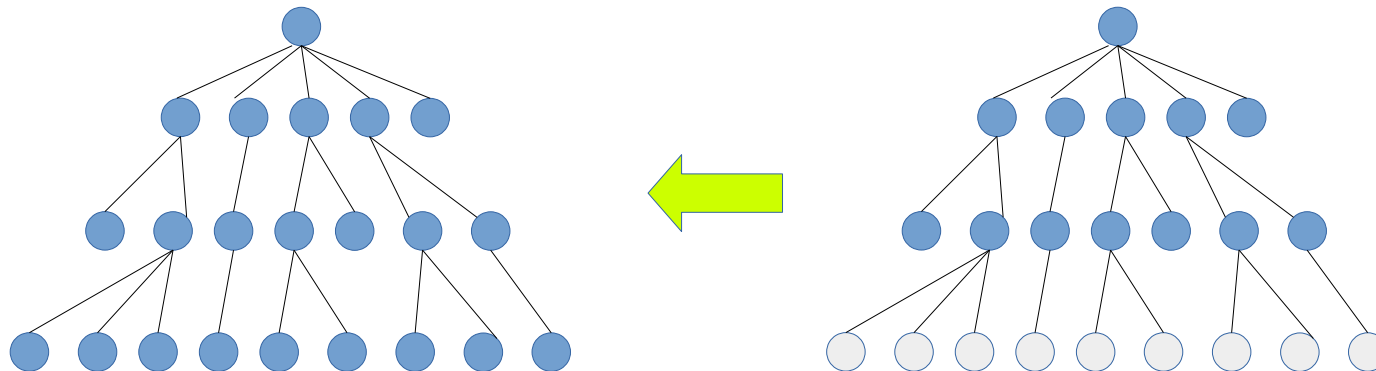
```
function RICERCA-APPROFONDIMENTO-ITERATIVO(problema) returns una soluzione, o il fallimento
  inputs: problema, un problema

  for profondità  $\leftarrow$  0 to  $\infty$  do
    risultato  $\leftarrow$  RICERCA-PROFONDITÀ-LIMITATA(problema, profondità)
    if risultato  $\neq$  taglio then return risultato
```

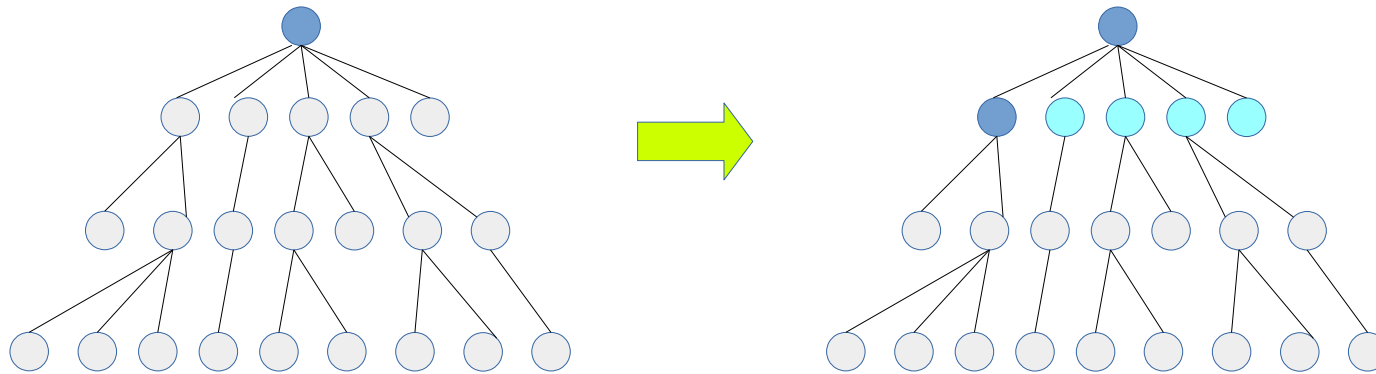
4. Ricerca per iterative deepening



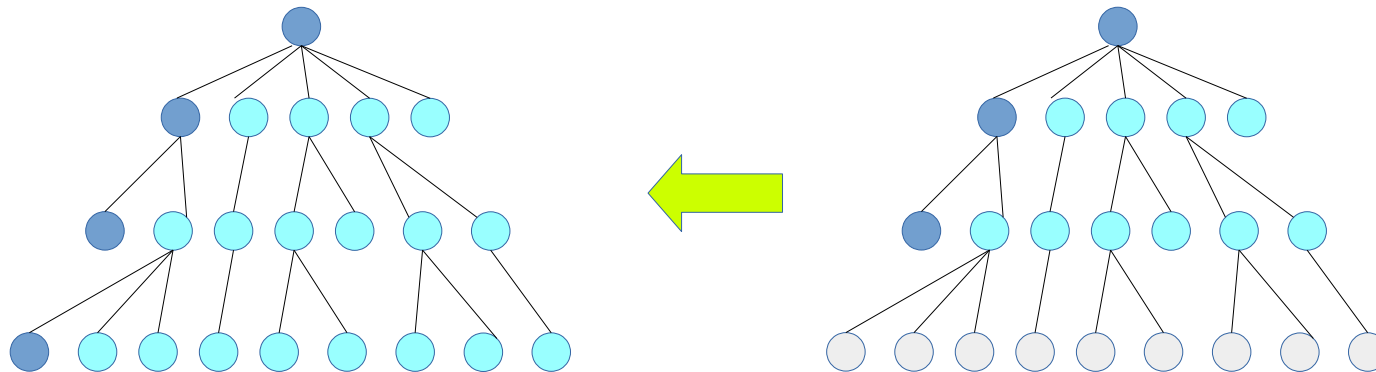
In questo esempio è come se venissero esplorati 4 alberi diversi, in successione



4. Ricerca per iterative deepening



I 4 alberi sono esplorati applicando la ricerca in profondità



4. Ricerca per iterative deepening

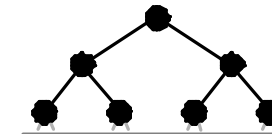
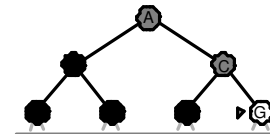
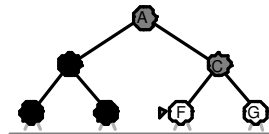
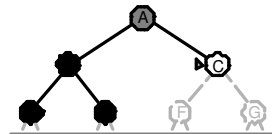
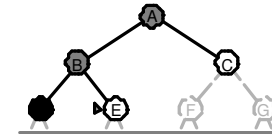
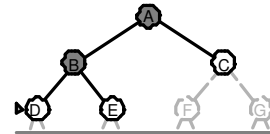
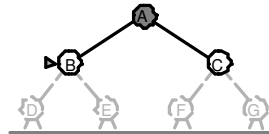
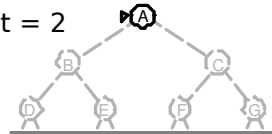
Limit = 0



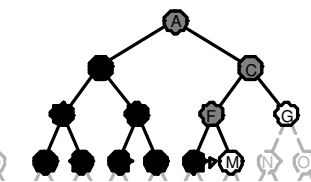
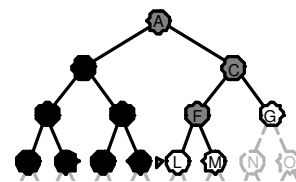
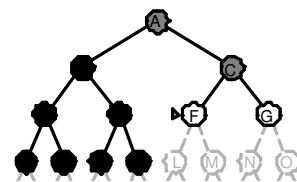
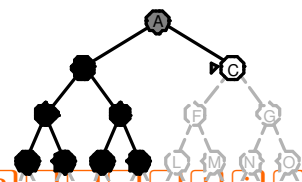
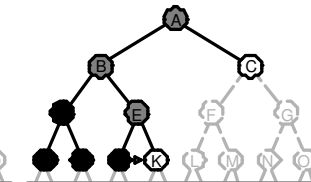
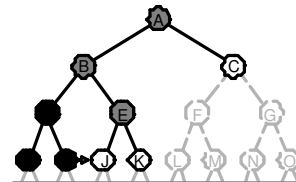
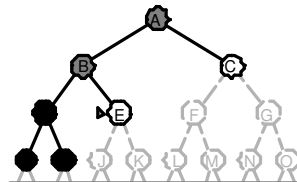
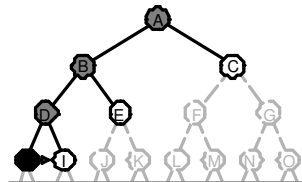
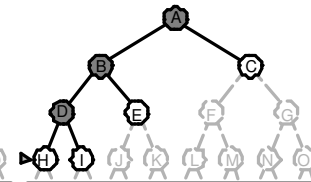
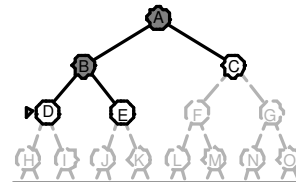
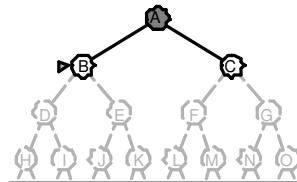
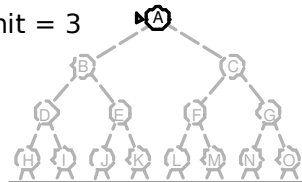
Limit = 1



Limit = 2

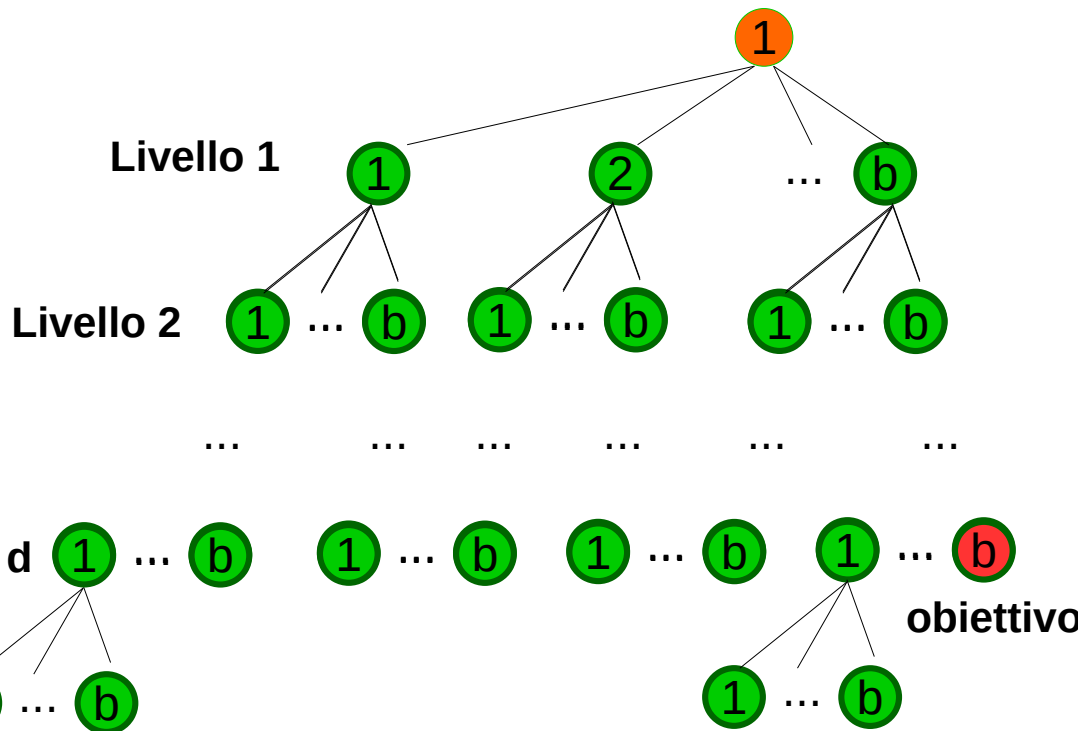


Limit = 3



Ampiezza: nodi espansi

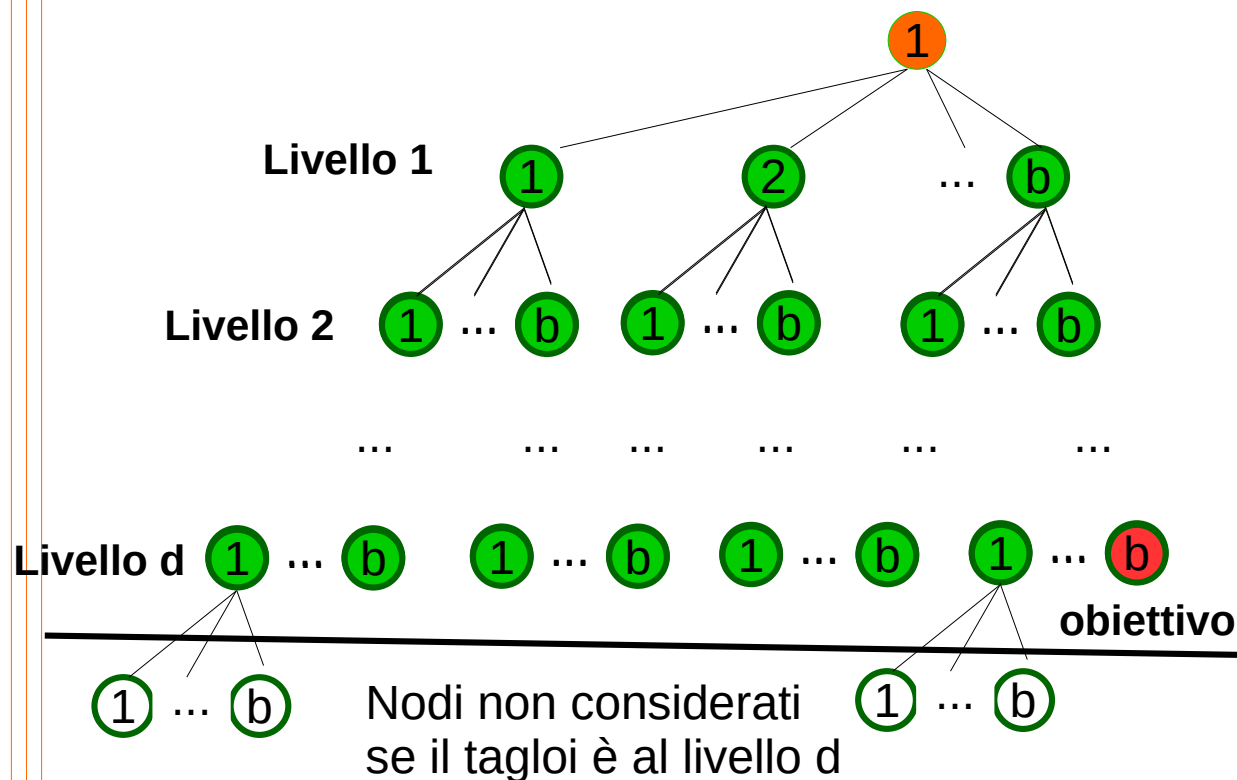
- $O(b^{d+1})$: ricordiamo che vengono generati anche i nodi di livello $d+1$



- 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.

Iterative deepening: nodi espansi

- Ricordiamo che vengono generati i nodi solo fino al livello d , dove viene posto il taglio

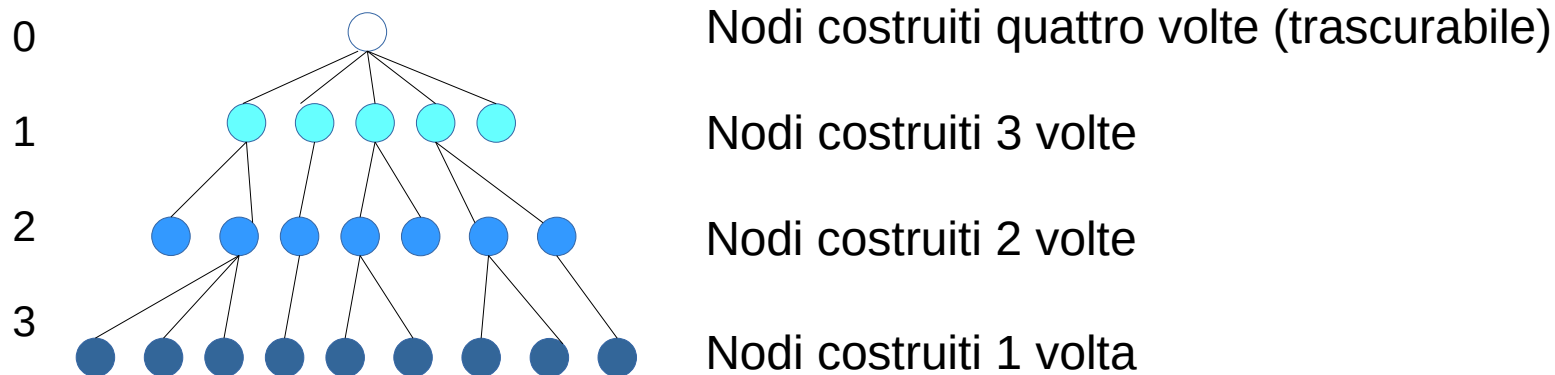


- Il tempo è misurato in termini di **numero di nodi espansi**
- Nel caso peggiore ogni nodo avrà b figli
- Livello 1**: b nodi (figli della radice)
- Livello 2**: b^2 nodi (b nodi per ciascuno di quelli del Livello 1)
- Livello 3**: b^3 nodi
- ...
- Livello d** (**livello del nodo obiettivo più vicino**):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

Attenzione alla versione del libro: questa spiegazione non è corretta in tutte

4. Ripartire da zero costa molto?

Esempio, sia $d = 3$



La maggior parte dei nodi si trova vicina alla frontiera, per cui ricostruire l'albero non risulta significativamente più costoso di espandere la sola frontiera. Il numero di nodi N_{id} complessivamente generato dall'iterative deepening è:

$$N_{id} = d*b + (d-1)b^2 + \dots + (d-i)b^{i+1} + \dots + 1*b^d = \sum_{i \in [1,d]} (d-i)b^i$$

dove d è la profondità minima dell'obiettivo e b il branching factor.

Complessità temporale: $O(b^d)$. N_{id} risulta MINORE del numero di nodi generati dalla ricerca in ampiezza N_{amp} nel caso peggiore:

$$N_{amp} = b + b^2 + \dots + b^i + \dots + b^d + (b^{d+1} - b)$$

Il motivo è che in questo caso N_{amp} produce anche nodi di livello $d+1$

4. Perché ripartire da zero?

Poiché è controintuitivo che iterative deepening possa essere una strategia più efficiente della ricerca in ampiezza facciamo un esempio:

Siano $b=10$ e $d=5$

$$\begin{aligned} N_{id} &= (d+1)*b^0 + d*b + (d-1)*b^2 + \dots + (d-i)*b^{i+1} + \dots + 1*b^d = \\ &= 6 + 50 + 400 + 3000 + 20.000 + 100.000 = 123.456 \end{aligned}$$

$$\begin{aligned} N_{amp} &= b + b^2 + \dots + b^i + \dots + b^d + (b^{d+1} - b) = \\ &= 10 + 100 + 1000 + 10.000 + 100.000 + 999.990 = 1.111.100 \end{aligned}$$

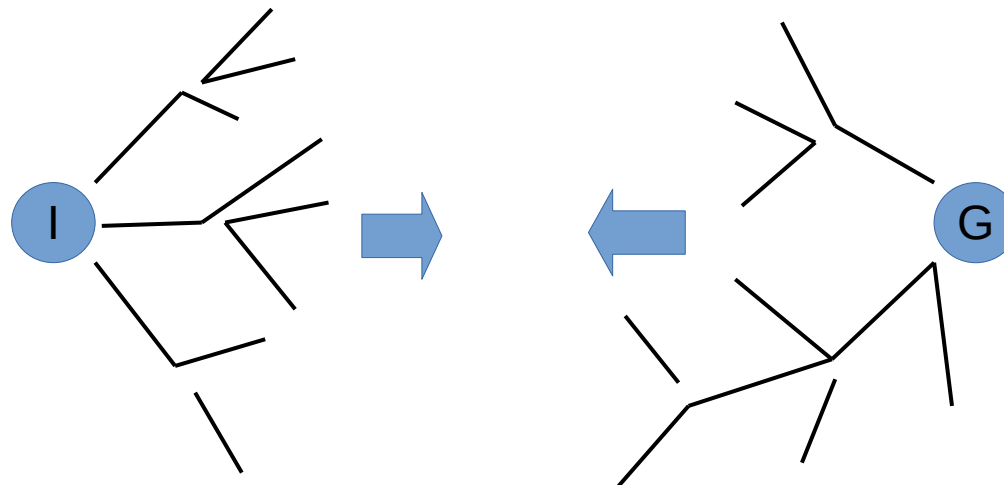
La ricerca in ampiezza crea meno nodi quando **esplora in minima parte il $(d+1)$ -mo livello**, Con $b=10$ produce circa l'11% dei nodi prodotti dalla ricerca in ampiezza

4. Iterative deepening, valutazione

- Combina gli aspetti migliori delle ricerche in ampiezza e in profondità
- È preferito quando lo spazio di ricerca è ampio e la profondità della soluzione non prevedibile
- Come la ricerca in profondità ha una **complessità spaziale** modesta: $O(b*d)$, b = branching factor, d = profondità minima del nodo obiettivo
- La complessità temporale è alta, simile a ricerca in ampiezza: $O(b^d)$
- Come quella in ampiezza:
 - è **completa** quando b è finito
 - è **ottima** quando il costo è funzione non decrescente della profondità

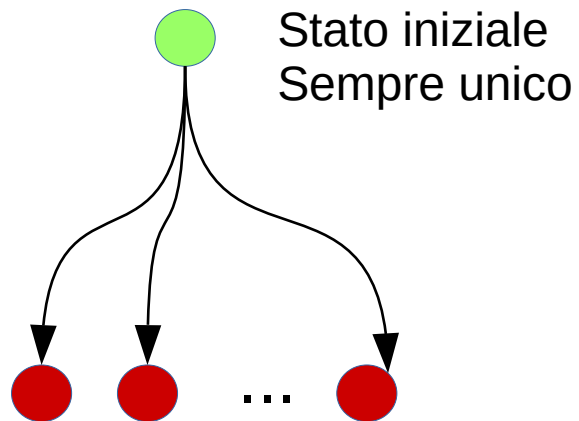
4. Ricerca bidirezionale

- È composta da due ricerche:
 - Una forward dallo stato iniziale
 - Una backward dallo stato obiettivo
- Termina quando le due ricerche si incontrano, cioè quando le frontiere hanno intersezione non vuota
- Il test si fa alla selezione (o espansione) di un nodo; se si usano hash table il tempo sarà costante

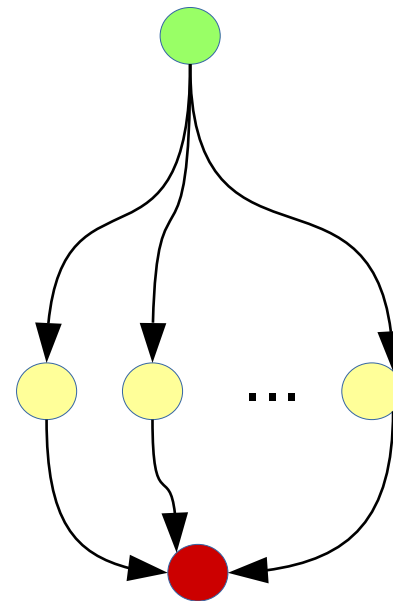


4. Ricerca bidirezionale

- E se lo stato obiettivo non è unico? Da dove iniziare la ricerca backward?
- Un modo è introdurre un nuovo stato di comodo, e renderlo raggiungibile in un passo da tutti gli stati obiettivo reali



Molti possibili stati obiettivo



Nuovo stato obiettivo "di comodo"

4. ricerca bidirezionale

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

PROBLEMA: come determinare i possibili stati predecessori di un nodo?

Nel gioco dell'8 ricerca in avanti e all'indietro sono molto simili ma non è così per qualsiasi problema. Quando lo spazio degli stati è molto grande può essere difficile generare predecessori in modo efficiente.

4. ricerca bidirezionale

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Immaginiamo gli operatori come aventi la forma

IF (antecedente) THEN (conseguente)

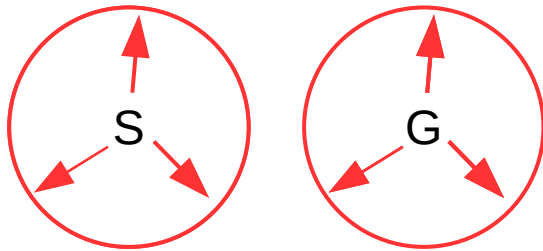
Ricerca forward: controllo quali operatori hanno l'antecedente verificato dallo stato corrente e produco i possibili stati successivi applicando il conseguente

Ricerca backward: controllo quali operatori hanno il conseguente verificato dallo stato corrente e produco i possibili stati successivi utilizzando la conoscenza degli antecedenti

Terminazione: quando gli stati ottenuti ricercando backward hanno intersezione non vuota con quelli ottenuti ricercando forward

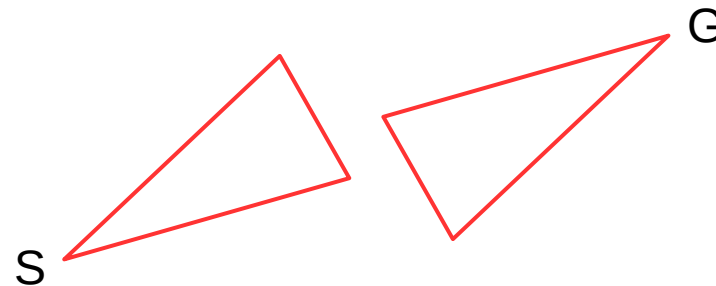
4. Ricerca bidirezionale

- Quando le due ricerche procedono si possono avere due andamenti



A fronte d'onda

Quando non si ha informazione aggiuntiva si esplorano tutte le possibili operazioni

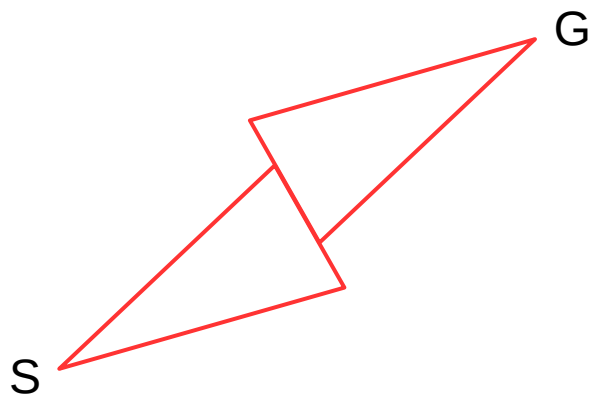


A cono

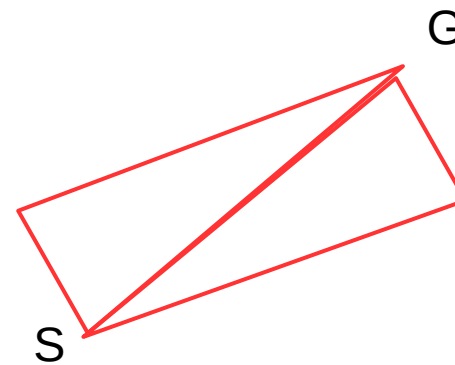
Quando si ha informazione aggiuntiva si esplora solouna parte delle operazioni

4. Ricerca bidirezionale: coni

- Quando le due ricerche esplorano una parte delle possibili operazioni l'ideale è che si incontrino a metà strada o si rischia di fare il doppio del lavoro



ideale



Da evitare

4. Ricerca bidirezionale: motivazione

- Interessante perché dai **costi molto contenuti**
- **Complessità temporale e spaziale:**
 $O(b^{d/2})$ più precisamente $2 * O(b^{d/2})$.
È vantaggioso rispetto a $O(b^d)$?
Molto! Esempio: se $b=2$ e $d=6$, $b^d = 64$, $b^{d/2} = 8$
- **Completezza:**
se il branching factor è finito e se si usa una ricerca in ampiezza in entrambe le direzioni
- **Ottimalità:**
se i costi dei passi sono identici e se si usa una ricerca in ampiezza in entrambe le direzioni

Grafi di ricerca e nodi già esplorati

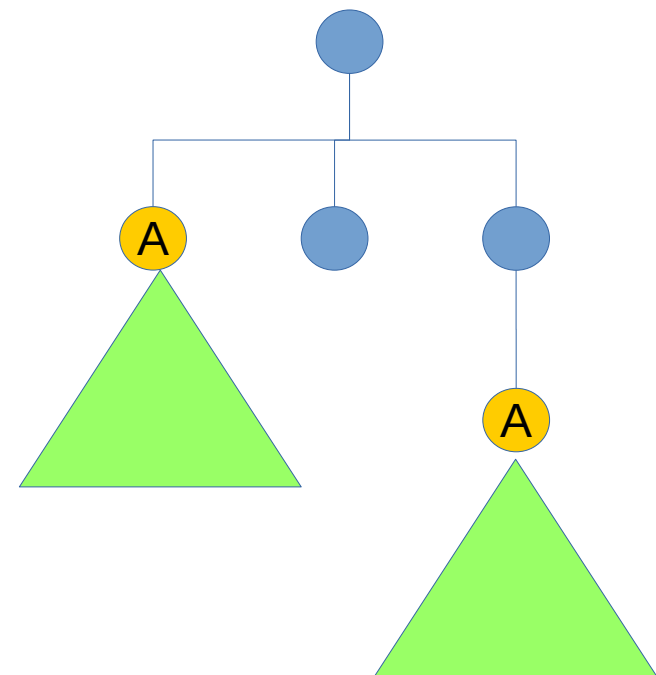
```
function RICERCA-ALBERO(problema) returns una soluzione, o il fallimento
  inizializza la frontiera usando lo stato iniziale di problema
  loop do
    if la frontiera è vuota then return fallimento
    scegli un nodo foglia e rimuovilo dalla frontiera
    if il nodo contiene uno stato obiettivo then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

function RICERCA-GRAFO(problema) returns una soluzione, o il fallimento
  inizializza la frontiera usando lo stato iniziale di problema
  inizializza al vuoto l'insieme esplorato
  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
    aggiungi il nodo all'insieme esplorato
    espandi il nodo scelto, aggiungendo i nodi risultanti alla frontiera
    solo se non è nella frontiera o nell'insieme esplorato
```

Figura 3.7 Una descrizione informale degli algoritmi di ricerca su albero e di ricerca su grafo. Le parti di RICERCA-ALBERO evidenziate in grassetto corsivo sono le aggiunte necessarie per gestire gli stati ripetuti.

Grafi di ricerca

- In molti problemi di ricerca lo stesso stato può essere raggiunto tramite percorsi differenti, causando una ripetizione di parte dell'esplorazione
- Marcare i nodi già visitati e controllare sempre se i nodi da esplorare sono per caso già stati visitati rende (molto) più efficiente la ricerca



Problemi dell'attraversamento del fiume

Capra, cavolo e lupo

- Un uomo deve traghettare un lupo, una pecora e un cavolo. Ha una barca in cui può far salire uno solo dei tre per volta ma non deve lasciare su di una riva insieme:
 - Pecora e cavolo
 - Lupo e pecora



Rappresentazione degli stati

- Siano **a** e **b** le due rive
- Possiamo rappresentare ogni elemento della scena con una **variabile** (U, C, P e L per uomo, cavolo, pecora e lupo) che vale a oppure b, a seconda della riva su cui si trova
- Non ci interessa rappresentare il transito da una riva all'altra, immaginiamolo istantaneo
- Stato iniziale: $\langle U=a, C=a, P=a, L=a \rangle$ (o semplicemente $\langle a,a,a,a \rangle$)
- Stato target: $\langle U=b, C=b, P=b, L=b \rangle$ ($\langle b,b,b,b \rangle$)
- Azioni? Una? Più di una? Precondizioni? Effetti?
- Disegnate lo spazio degli stati

Missionari e cannibali

- Un po' più complesso del precedente
- Ci sono 3 missionari e 3 cannibali sulla riva di un fiume che tutti vogliono attraversare
- Vi è una sola barca che può contenere 1 o 2 persone alla volta
- La barca necessita di una persona che la porti da una riva all'altra, quindi se trasporta due persone all'andata ed altre persone rimangono sulla riva di partenza, almeno una dovrà tornare indietro per prendere gli altri
- Lo scopo è fare attraversare tutte e 6 le persone senza mai lasciare su una delle due rive più cannibali che missionari

Rappresentazione

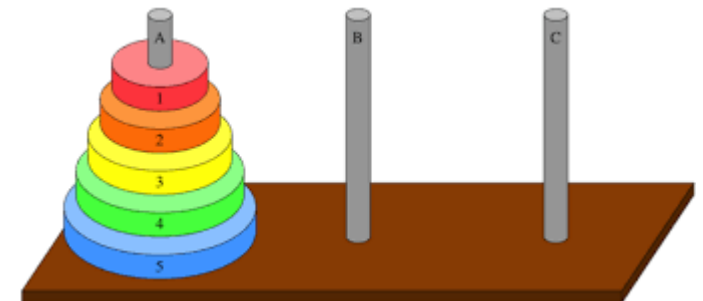
- Siano a e b la riva di partenza e di arrivo
- Sono state proposte diverse rappresentazioni, fra cui:
 - $\langle Ma, Ca, Mb, Cb, B \rangle$: numero di missionari/cannibali sulla riva a e b e riva su cui si trova la barca
 - $\langle Ma, Ba, B \rangle$: numero di missionari/cannibali sulla riva di partenza, Barca vale 0 se su b e 1 se su a
- Stato di partenza? Di arrivo? Azioni?

Torre di Hanoi

Scopo del gioco: spostare la torre dal primo al terzo piolo usando il secondo come appoggio. Due dischi possono essere sovrapposti solo se quello sotto è più grande di quello sopra.

Supponiamo per facilità di avere solo 3 dischi e di voler usare algoritmi di ricerca di una soluzione.

- 1) Dare una rappresentazione degli stati, e in particolare di stato iniziale stato goal
- 2) Definire le azioni con cui calcolare i successori
- 3) Disegnare i primi tre livelli dell'albero di ricerca (3 livelli oltre la radice). Evitate di rappresentare stati ripetuti
- 4) Provare ad applicare gli algoritmi di ricerca studiati per trovare una soluzione
- 5) Per ognuno calcolare il numero di nodi generati



Missionari e cannibali

- Siano le due rive $r1$ (quella di partenza) ed $r2$ (quella a cui tutti desiderano passare)
- Catturiamo i possibili stati tramite un vettore $\langle A, B, C \rangle$:
 - A = numero di missionari su $r1$
 - B = numero di cannibali su $r1$
 - C = numero di barche su $r1$
- Stato di partenza?
- Costruire il grafo delle possibili transizioni di stato
- Provare ad applicare le strategie di ricerca non informata studiate per trovare una soluzione
- Fare un confronto

Missionari e cannibali

