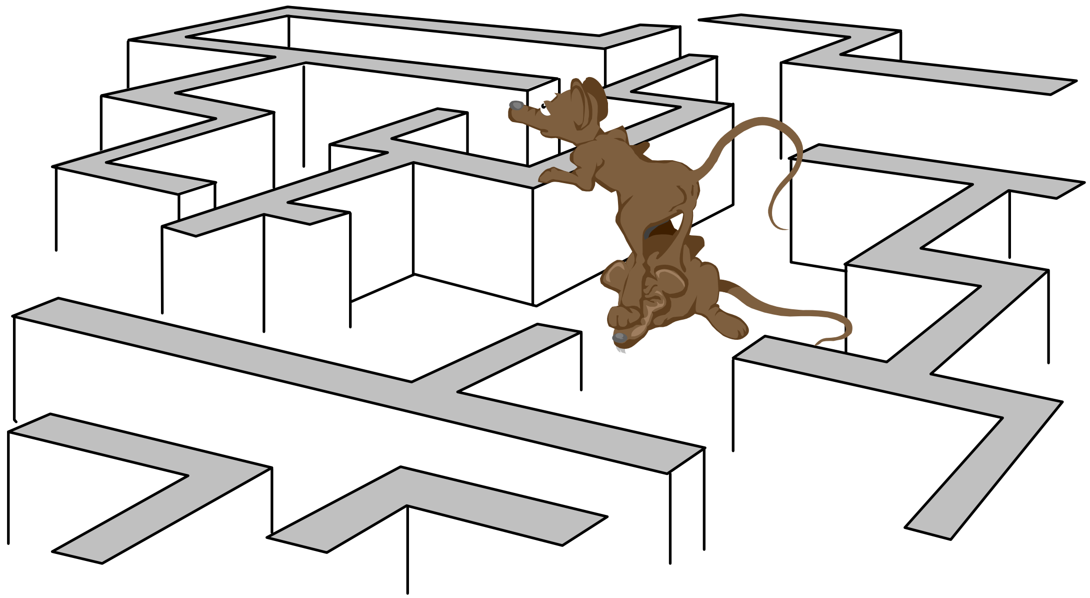
Algoritmi di Ricerca



Contenuto

- **Algoritmi non informati**
  - Nessuna conoscenza sul problema in esame
- **Algoritmi euristici**
  - Sfruttano conoscenze specifiche sul problema
- **Giochi**
  - Quando la ricerca è ostacolata da un “agente ostile” (l’avversario)

Algoritmi non informati

Non richiedono nessuna conoscenza specifica relativa al problema (approccio *brute-force*)

Vantaggio: ***generalità***

Noi vedremo:

- **Ricerca in ampiezza**
- **Ricerca in profondità**
- **Iterative deepening**

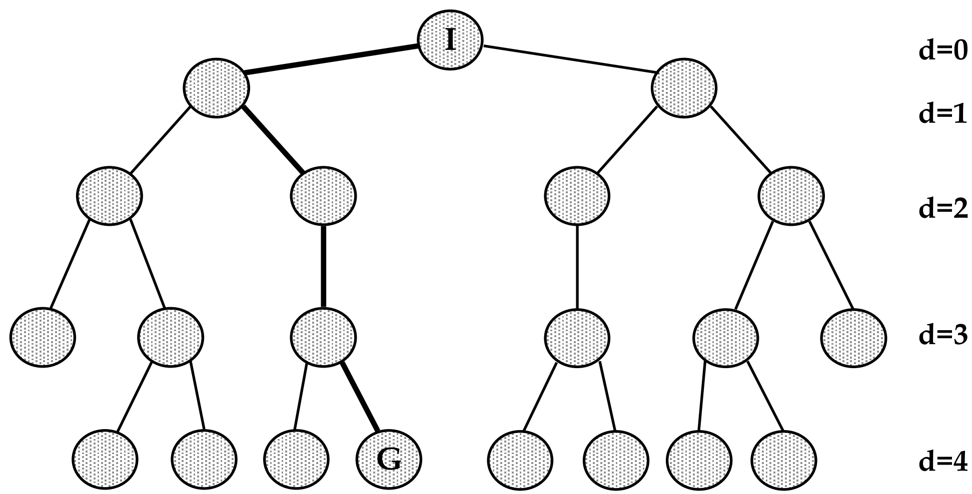
Problemi di ricerca

I problemi di ricerca sono spesso formulati in termini di *alberi*.

I = nodo iniziale

G = nodo "goal"

d = profondità



Obiettivo: ***cercare un percorso che, partendo dal nodo iniziale I, arrivi al nodo finale G.***

Algoritmi di ricerca

Lo spazio di ricerca (albero) è raramente memorizzato completamente in memoria ma, in genere, è disponibile una procedura che, dato in input un nodo qualsiasi, restituisce la lista dei suoi successori:

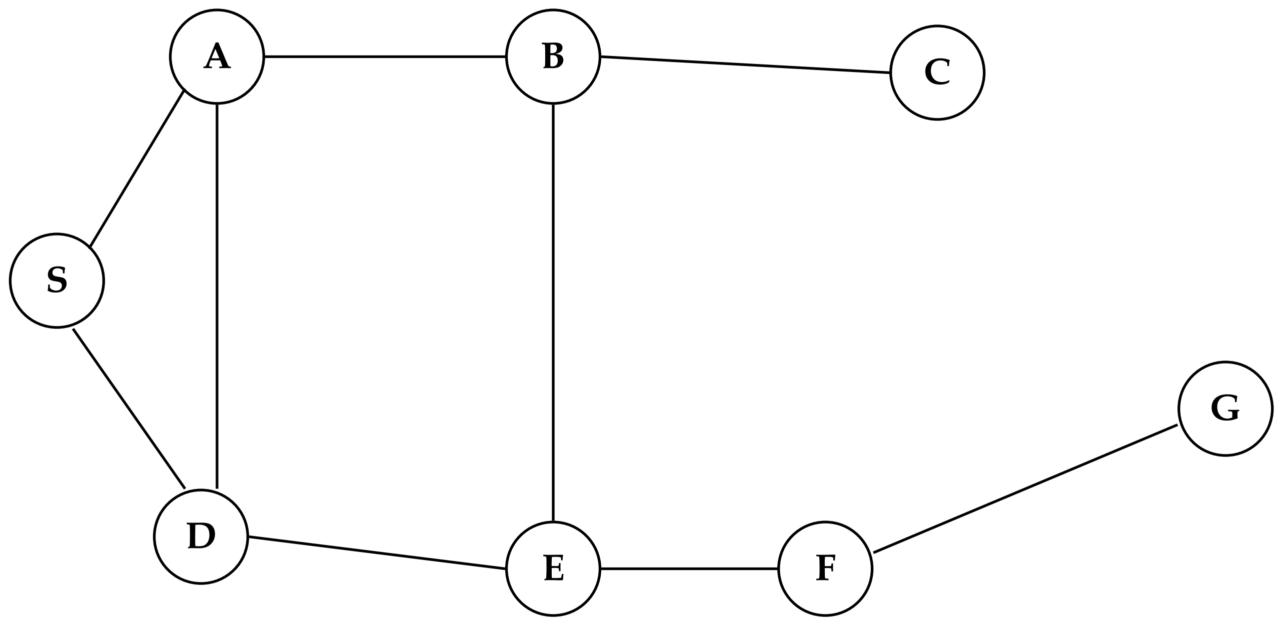
$$\texttt{succ(n)} \longrightarrow \texttt{n}_1\texttt{,...,n}_k$$

Un algoritmo di ricerca si caratteriza da:

- **Input: descrizione dei nodi I, e test per G**
- **Output: sequenza (legale) di nodi a partire da quello iniziale fino ad un nodo goal**

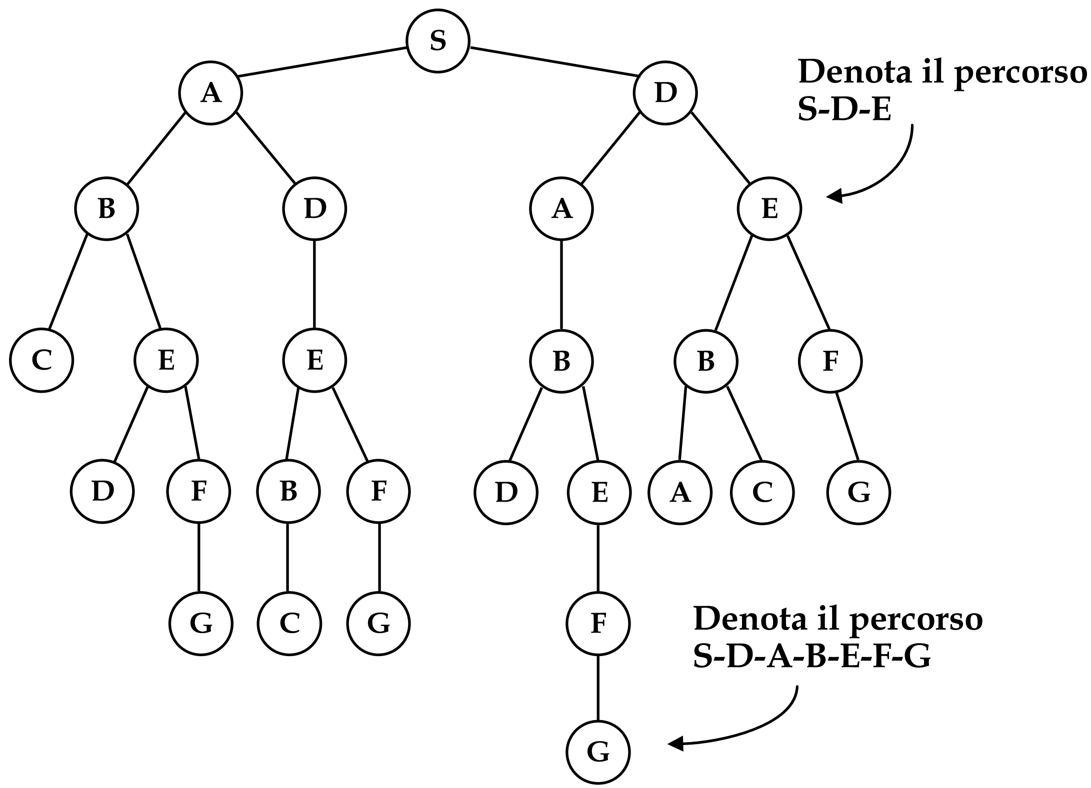
Esempio

Trovare un percorso che partendo dal nodo S arrivi al nodo G.



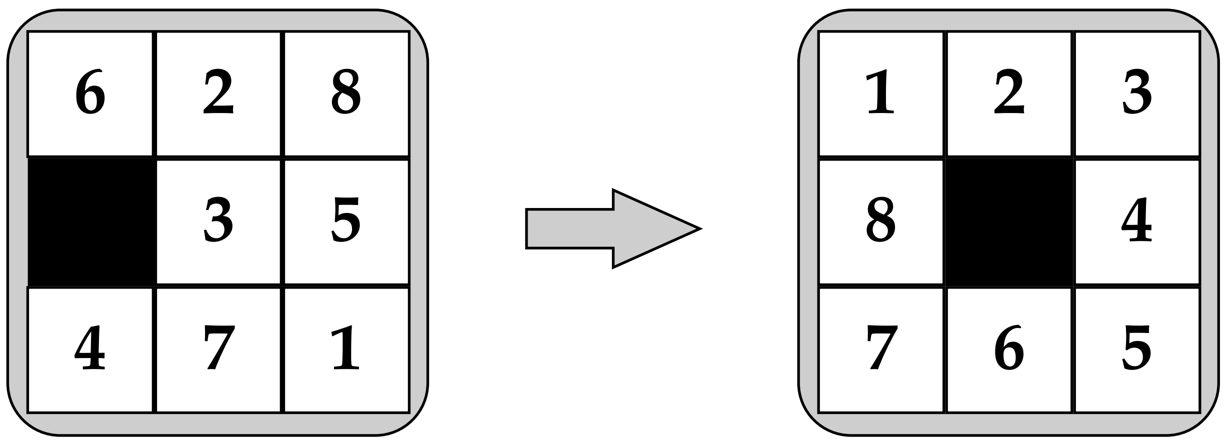
L'albero di ricerca

Il problema si può trasformare in uno di ricerca su albero.



Il gioco dell'8

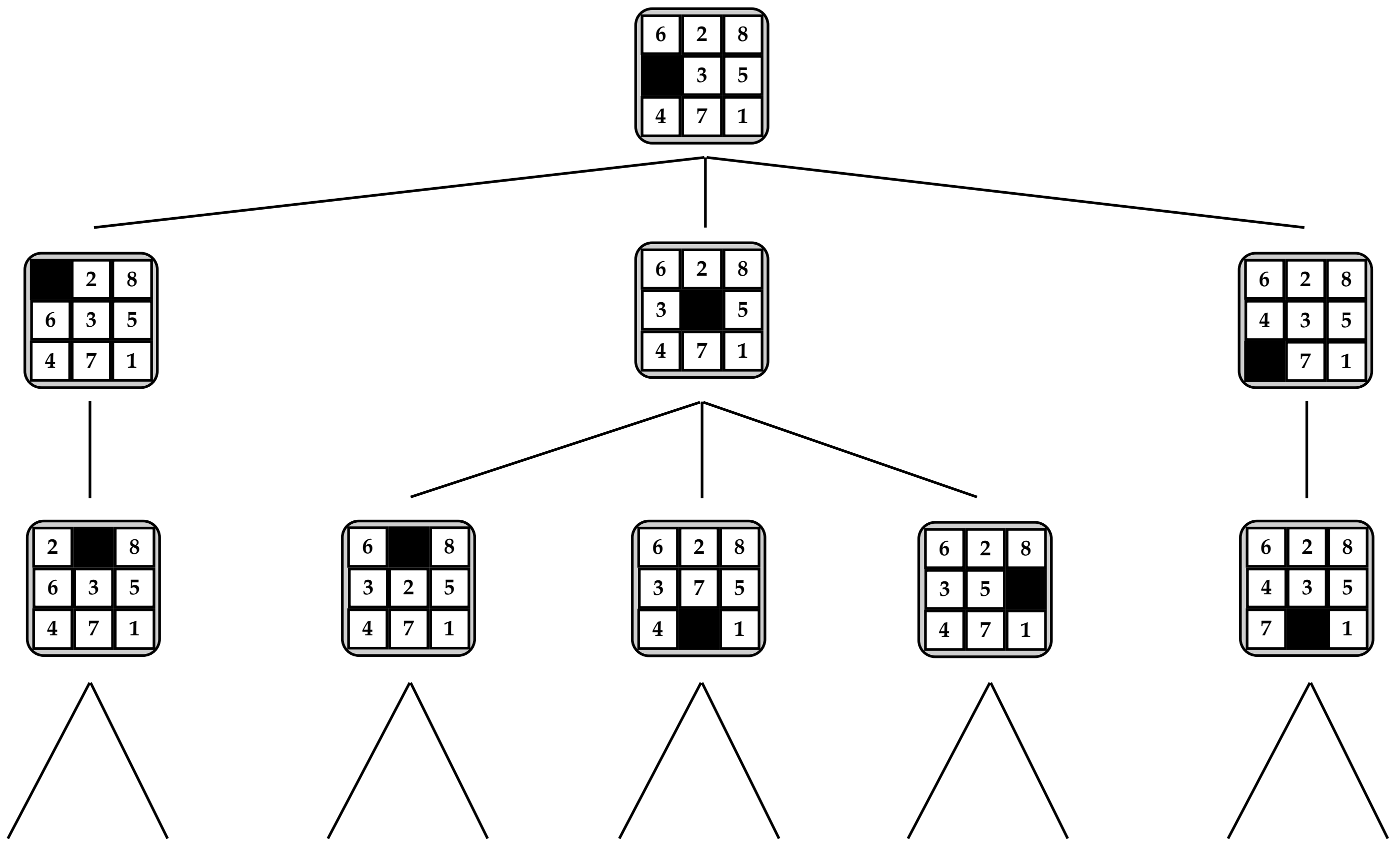
Spostare le tessere numerate in modo da passare dalla configurazione iniziale a quella finale



Configurazione iniziale

Configurazione finale

L'albero di ricerca

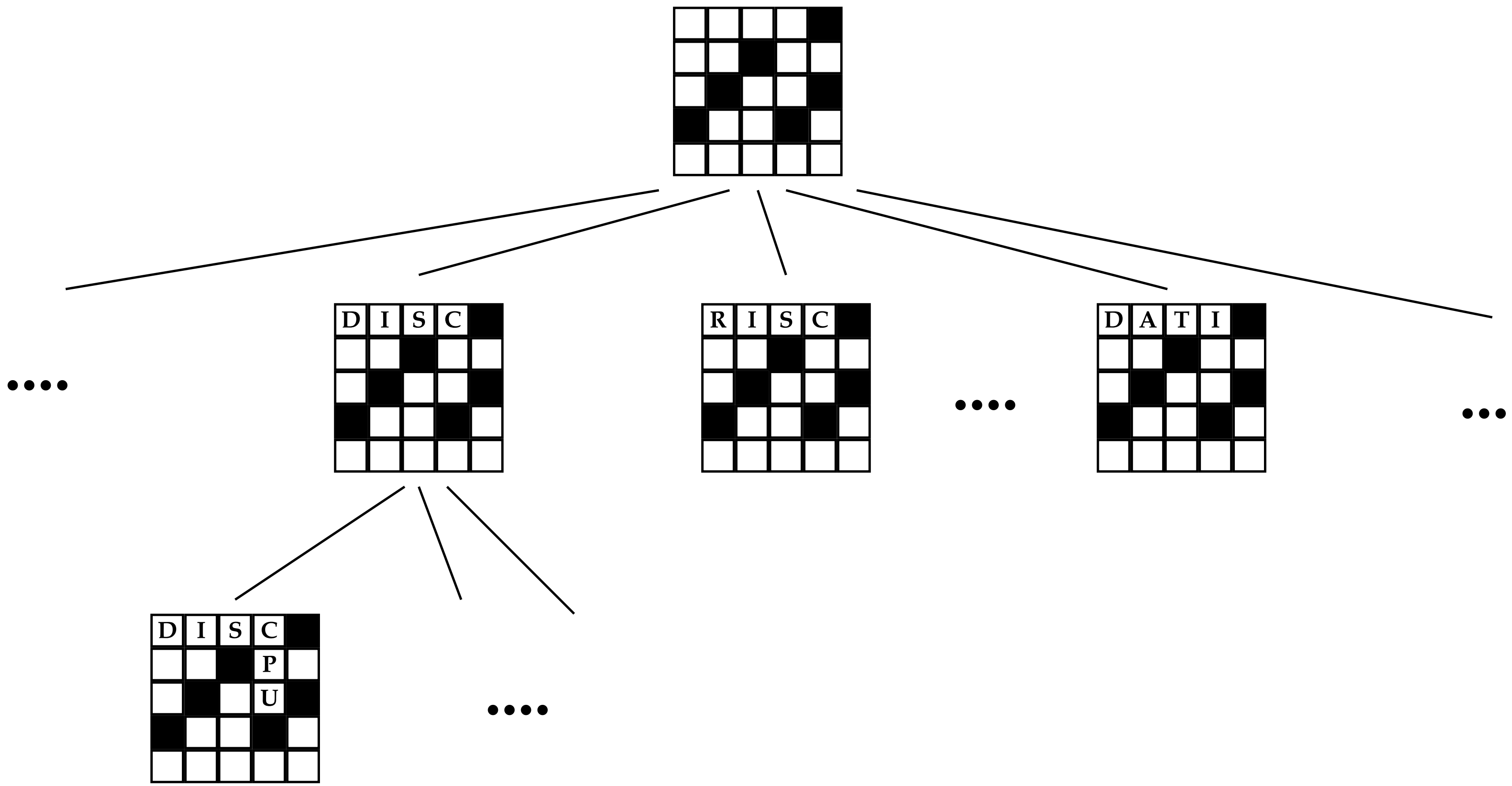


Cruciverba

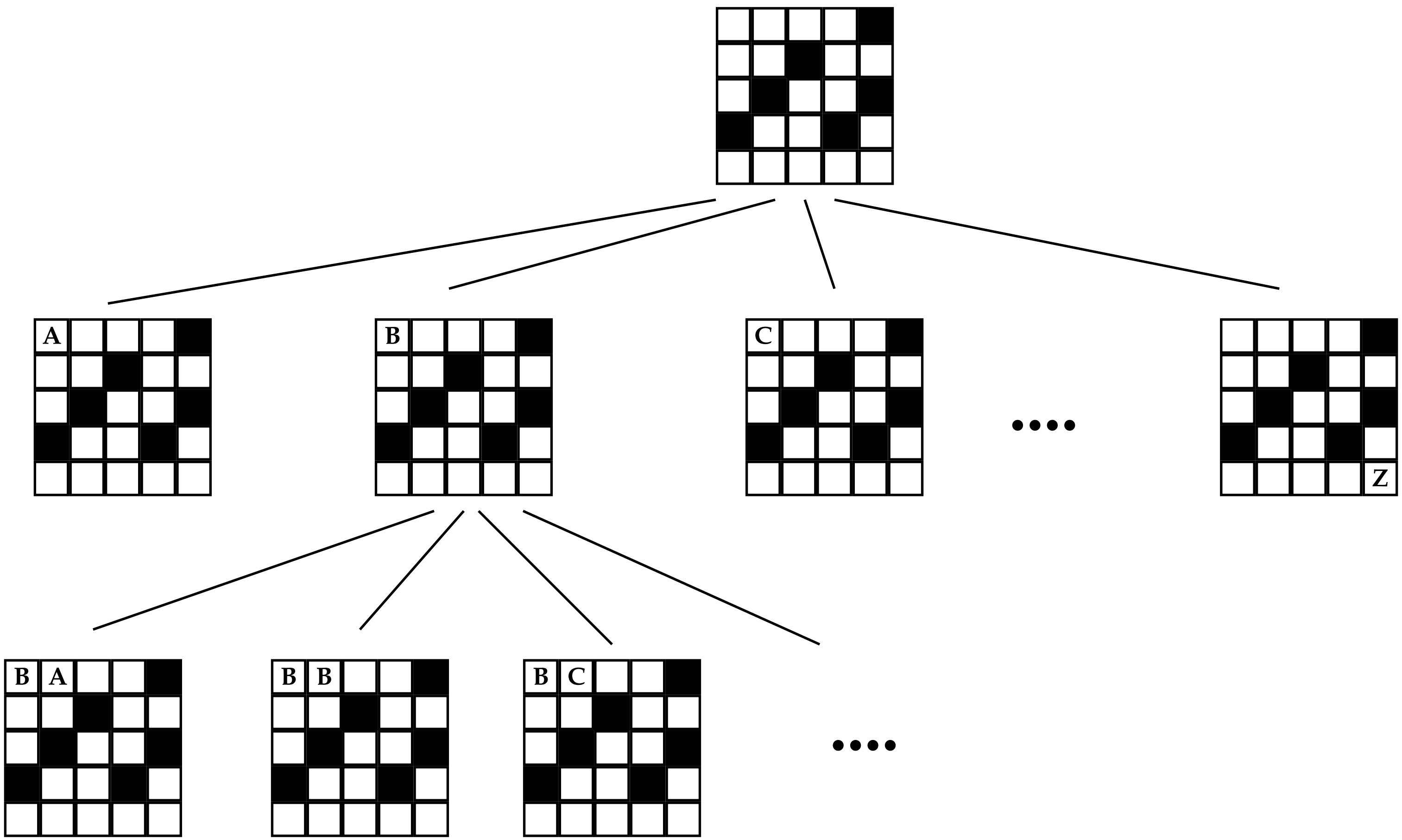
Ci sono 26^{19} modi possibili per riempire le caselle del cruciverba, ma soltanto una piccolissima parte è “legale”, cioè costituite da parole di senso compiuto

R	I	S	C	
O	F		D	O
M		C	R	
	T	P		P
M	O	U	S	E

L'albero degli stati (1)



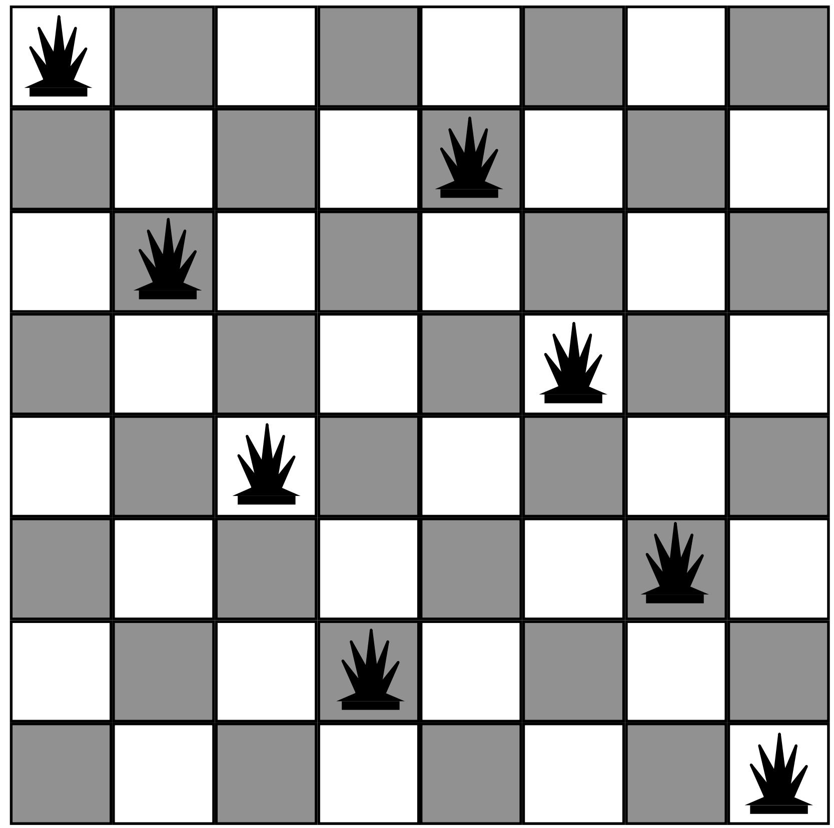
L'albero degli stati (2)



Il problema delle *n* regine

Disporre *n* regine su una scacchiera *n* x *n* in modo che nessuna possa attaccare le altre.

Questa è una soluzione?



Valutazione

Ci sono 4 criteri per valutare un algoritmo di ricerca:

- **Completezza:**
  - **l’algoritmo troverà una soluzione, se esiste?**
- **Ottimalità:**
  - **l’algoritmo troverà la soluzione “migliore”, quando ve n’è più d’una?**
- **Complessità temporale:**
  - **quanto tempo richiederà il trovare una soluzione?**
- **Complessità spaziale:**
  - **quanta memoria richiederà l’algoritmo?**

Una procedura di ricerca

1) Sia *L* la lista dei nodi "iniziali". In ogni istante di tempo *L* conterrà tutti i nodi che non sono ancora stati esaminati dal programma

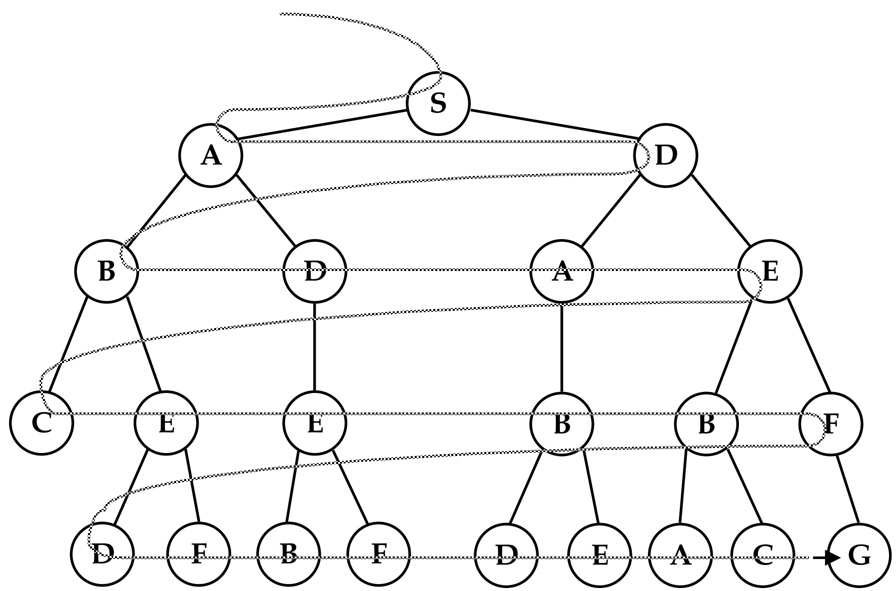
2) Se *L* è vuota, ESCI con FALLIMENTO. Altrimenti, prendi un nodo *n* da *L*

3) Se *n* è un nodo "goal", allora ESCI e restituisci in output *n* ed il percorso dal nodo iniziale fino a *n*

4) Altrimenti, cancella *n* da *L* ed aggiungi a *L* tutti i nodi discendenti di *n*, etichettando ciascuno di essi con il percorso dal nodo iniziale

5) Ritorna al passo 2)

Ricerca in ampiezza
Esempio



Ricerca in ampiezza (FIFO)

1) **Sia *L* la lista dei nodi "iniziali"**
2) **Sia *n* il primo nodo di *L*. Se *L* è vuota, ESCI**
3) **Se *n* è un nodo "goal", allora ESCI e restituisci in output *n* ed il percorso dal nodo iniziale fino a *n***
4) **Altrimenti, cancella *n* da *L* ed aggiungi <u>*alla fine*</u> di *L* tutti i nodi discendenti di *n*, etichettando ciascuno di essi con il percorso dal nodo iniziale**
5) **Ritorna al passo 2)**

Ricerca in ampiezza
Completezza e Ottimalità

Poiché l'algoritmo procede livello dopo livello, è:

- ***Completo* (troverà sempre una soluzione se ce n'è almeno una)**

- ***Ottimale* (troverà sempre la soluzione migliore, cioè meno profonda)**

Ricerca in ampiezza
Complessità temporale

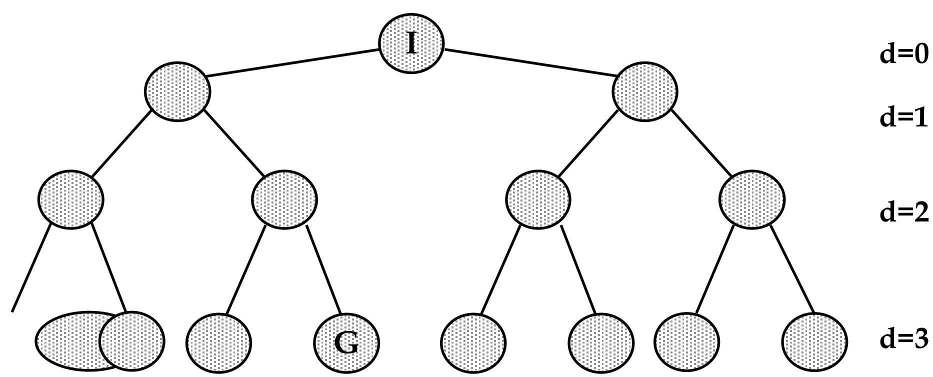
Ipotesi semplificative:

- **l'albero ha un fattore di ramificazione uniforme (*b*)**
- **profondità dell'albero *d***
- **esiste un solo nodo goal, ed è a livello *d***

Esempio:

***b*=2**

***d*=3**

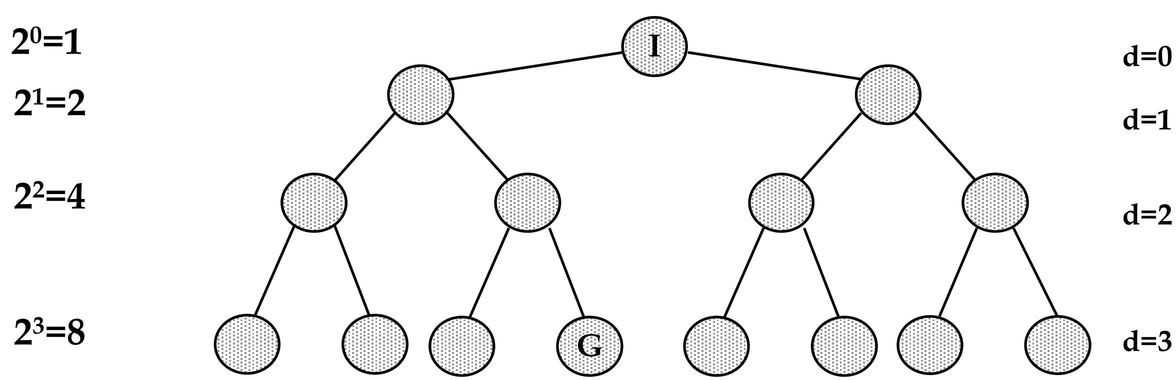


Ricerca in ampiezza
Complessità temporale

Il tempo di ricerca verrà misurato in termini del numero di nodi esaminati dall'algoritmo

Osservazione:

– **il numero di nodi a livello *d* è b^d**



Ricerca in ampiezza
Complessità temporale

Il numero di nodi non-terminali da esaminare per raggiungere il nodo goal al livello *d* è:

$$1 + b + b^2 + \ldots + b^{d-1} = \frac{b^d - 1}{b - 1}$$

Il numero *medio* di nodi terminali (a livello *d*) da esaminare è:

$$\frac{1 + b^d}{2}$$

Sommando:

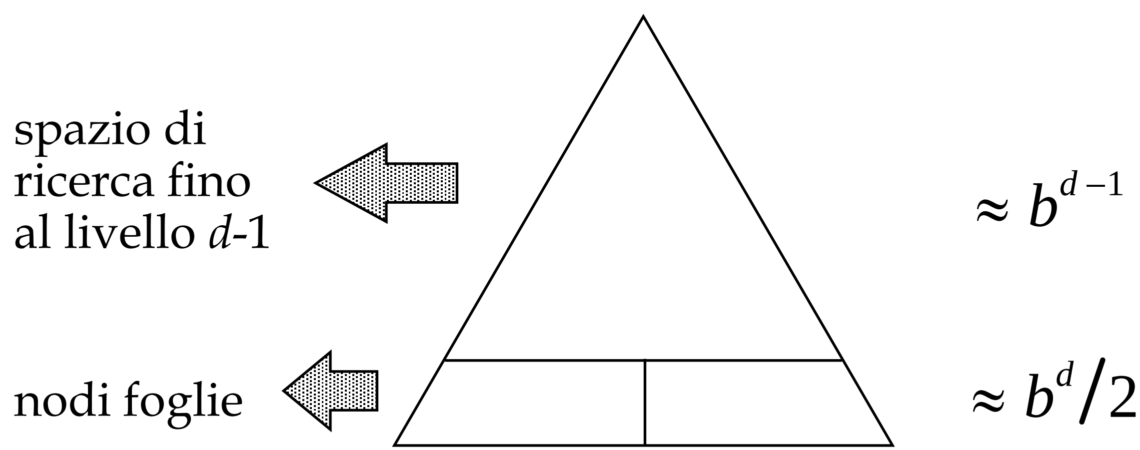
$$\frac{b^{d+1} + b^d + b - 3}{2(b-1)}$$

Ricerca in ampiezza
Complessità temporale

Per *d* molto grande, il tempo medio di ricerca diventa circa $b^d/2$, ovvero:

$$O(b^d)$$

che è circa il tempo speso per esaminare le foglie dell'albero.



Ricerca in ampiezza
Complessità spaziale

Prima di esaminare il primo nodo a livello *k+1*, l'algoritmo deve memorizzare tutti i nodi a livello *k* (ovvero b^k).

Di conseguenza, la quantità di memoria richiesta per raggiungere il nodo goal a livello *d*, sarà almeno:

$$b^{d-1}$$

cioè:

$$O(b^d)$$

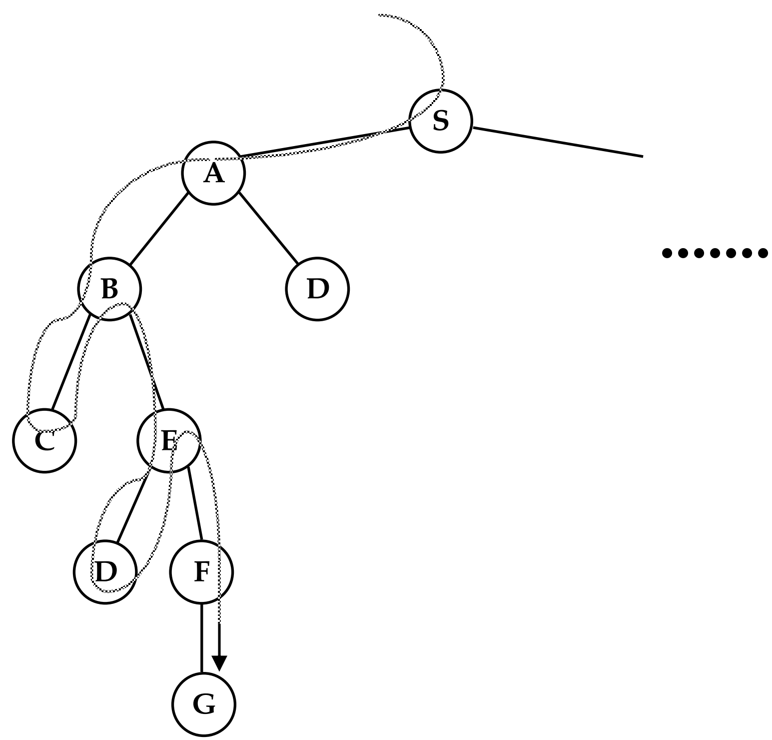
Ricerca in ampiezza
Esempio

Supponiamo di dover effettuare una ricerca su un albero con b=10, il programma elabora 1000 nodi al secondo, e richiede 100 byte per nodo.

Livello	Nodi	Tempo	Memoria
2	111	0.1 sec	11 Kb
6	10^6	18 min	111 Mb
10	10^{10}	128 giorni	11 Gb
14	10^{14}	3500 anni	11.111 Tb

NB: 1 Tb - 1000 miliardi di byte

Ricerca in profondità
Esempio



Ricerca in profondità (LIFO)

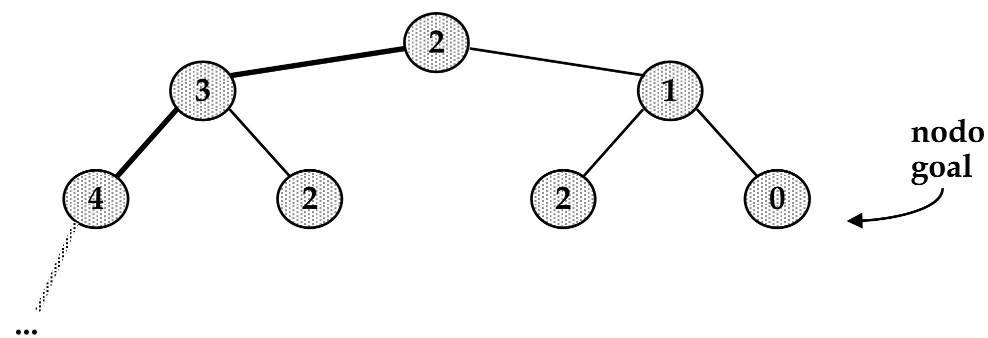
1) **Sia *L* la lista dei nodi "iniziali"**
2) **Sia *n* il primo nodo di *L*. Se *L* è' vuota, ESCI**
3) **Se *n* è un nodo "goal", allora ESCI e restituisci in output *n* ed il percorso dal nodo iniziale fino a *n***
4) **Altrimenti, cancella *n* da *L* ed aggiungi *all'inizio* di *L* tutti i nodi discendenti di *n*, etichettando ciascuno di essi con il percorso dal nodo iniziale**
5) **Ritorna al passo 2)**

Ricerca in profondità
Completezza

Se l'albero di ricerca ha una profondità infinita, l'algoritmo non è detto che trovi una soluzione (se ne esiste una). In altri termini:

L'algoritmo non è completo

Esempio: (albero per provare che 2 è un intero)

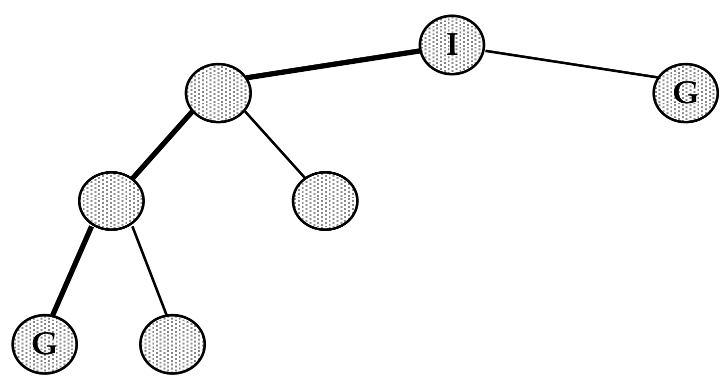


Ricerca in profondità
Ottimalità

Se il problema ammette più di una soluzione, l'algoritmo non è detto che trovi quella a profondità minima. In altri termini

L'algoritmo non è ottimale

Esempio:



Ricerca in profondità
Complessità temporale

- **Nel caso migliore, il nodo goal si trova alla estrema sinistra dell'albero. Qundi il numero di nodi da esaminare è:**

$$d+1$$

- **Nel caso peggiore è alla estrema destra. In questo caso il numero di nodi è**

$$1+\ldots+b^d = \frac{b^{d+1}-1}{b-1}$$

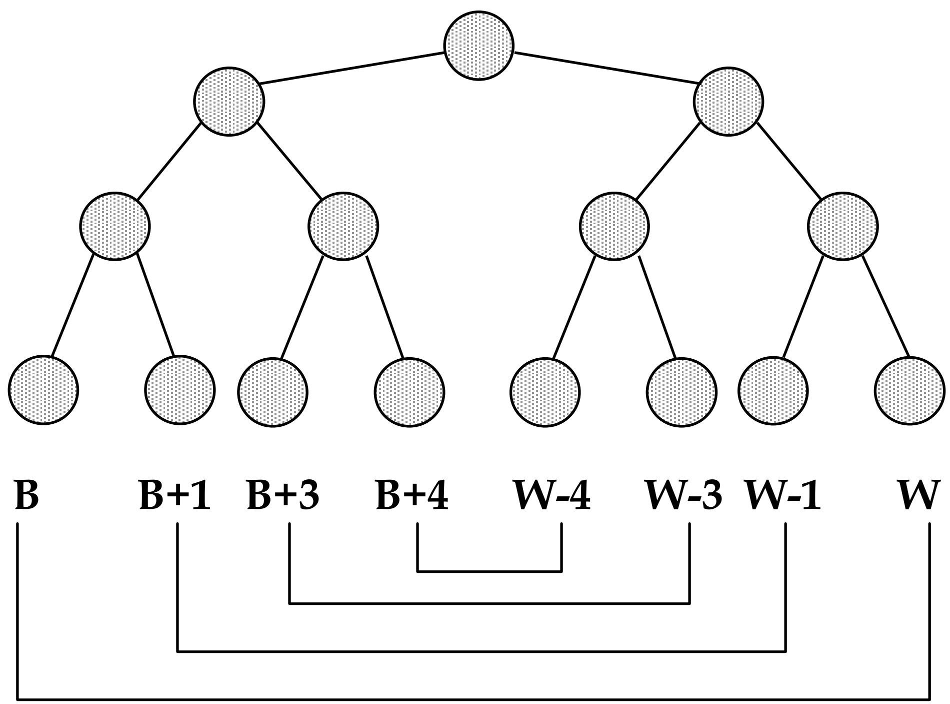
- **Mediando:**

$$\frac{b^{d+1}+bd+b-d-2}{2(b-1)}$$

Osservazione

E' lecito fare la media?

Sia *B* il caso migliore, e *W* il caso peggiore.

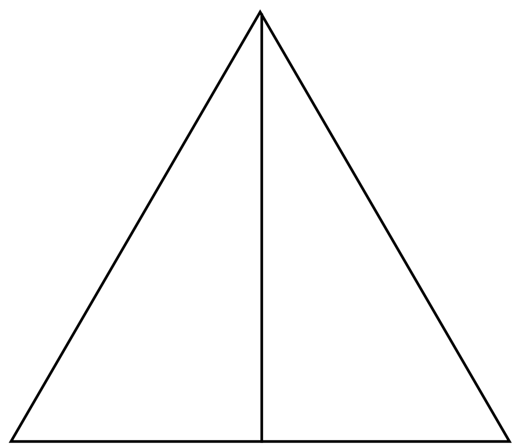


Ricerca in profondità
Complessità temporale

Per d molto grande, il tempo medio di ricerca diventa circa $b^d/2$, ovvero:

$$O(b^d)$$

che è circa il tempo speso per esaminare metà albero.



$\approx b^d/2$

Ricerca in profondità
Complessità spaziale

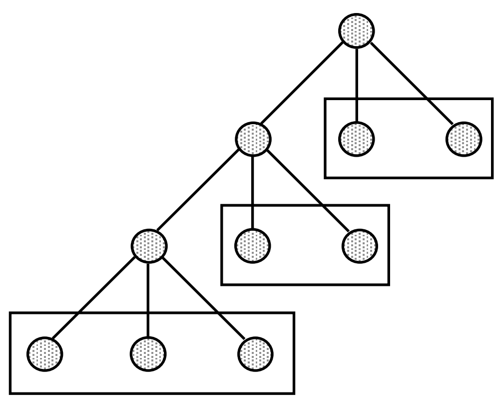
La massima quantità di memoria richiesta è

$$d(b-1)+1$$

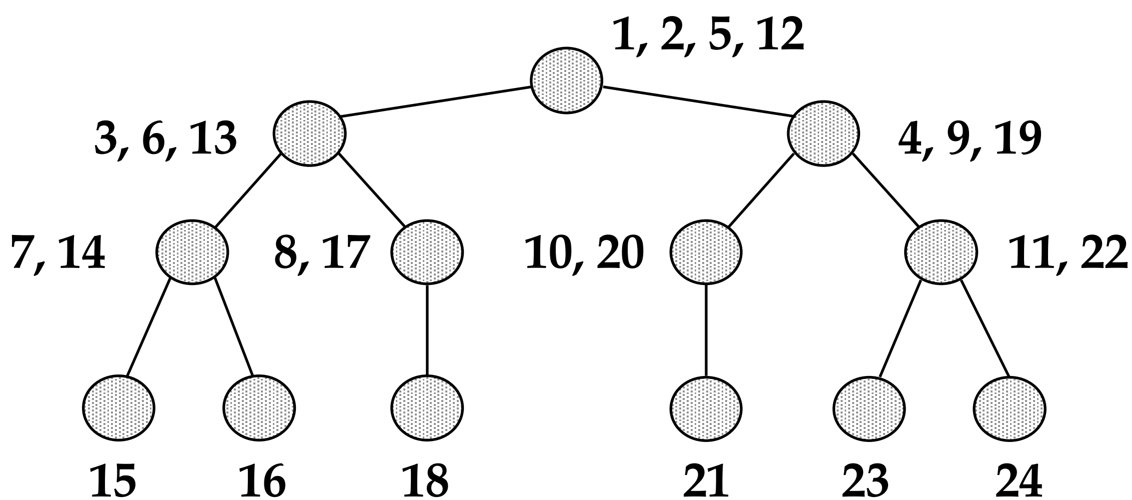
e quindi:

$$O(db)$$

Esempio



Iterative deepening
Esempio



NB: I numeri rappresentano l'ordine di espansione dei nodi

Iterative deepening

1) **Poni C=0; C rappresenta il livello massimo**
2) **Sia *L* la lista dei nodi "iniziali"**
3) **Sia *n* il primo nodo di *L*. Se *L* è vuota, incrementa C e ritorna al passo 2)**
4) **Se *n* è un nodo "goal", allora ESCI e restituisci in output *n* ed il percorso dal nodo iniziale fino a *n***
5) **Altrimenti, cancella *n* da *L.* Se il livello di *n* è minore di C, aggiungi *all'inizio* di *L* tutti i nodi discendenti di *n*, etichettando ciascuno di essi con il percorso dal nodo iniziale**
6) **Ritorna al passo 3)**

Iterative deepening
Completezza e Ottimalità

Poiché l'algoritmo procede livello dopo livello, è:

- ***Completo*** **(troverà sempre una soluzione se ce n'è almeno una)**

- ***Ottimale*** **(troverà sempre la soluzione migliore, cioè meno profonda)**

Iterative deepening
Complessità temporale

Nell'iterazione finale (a livello *d*), il numero medio di nodi da esaminare è

$$\frac{b^{d+1} + bd + b - d - 2}{2(b-1)}$$

In ciascuna iterazione precedente si saranno esaminati *(b^{j+1}-1)/(b-1)*, dove *j* è la profondità dell'albero. In totale:

$$\sum_{j=0}^{d-1} \frac{b^{j+1}-1}{b-1} = \frac{1}{b-1}\left[b\sum_{j=0}^{d-1} b^{j} - \sum_{j=0}^{d-1} 1 \right] = \frac{1}{b-1}\left[b\left(\frac{b^{d}-1}{b-1} \right) - d \right] =$$

$$\frac{b^{d+1} - bd - b + d}{(b-1)^{2}}$$

Iterative deepening
Complessità temporale

Sommando, si ottiene

$$\frac{b^{d+2} + b^{d+1} + b^2 d + b^2 - 4bd - 5b + 3d + 2}{2(b-1)^2}$$

che, per *d* grande, è dominato da:

$$\frac{(b+1)b^{d+1}}{2(b-1)^2}$$

che è

$$O(b^d)$$

Iterative deepening
Complessità spaziale

Poiché ad ogni livello l'algoritmo effettua una ricerca in profondità, la quantità massima di memoria richiesta è:

$$d(b-1)+1$$

che è

$$O(db)$$

Tabella riassuntiva

	Ampiezza	Profondità	It. Deep.
Completo?	**Sì**	**No**	**Sì**
Ottimale?	**Sì**	**No**	**Sì**
Tempo	b^d	b^d	b^d
Spazio	b^d	bd	bd