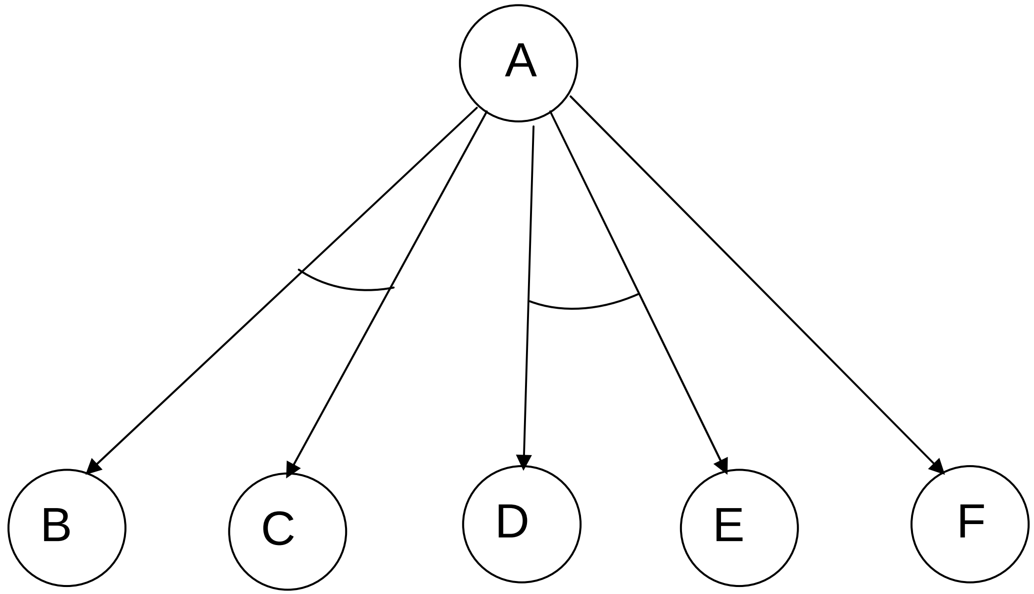
Riduzione a sottoproblemi e grafi AND/OR

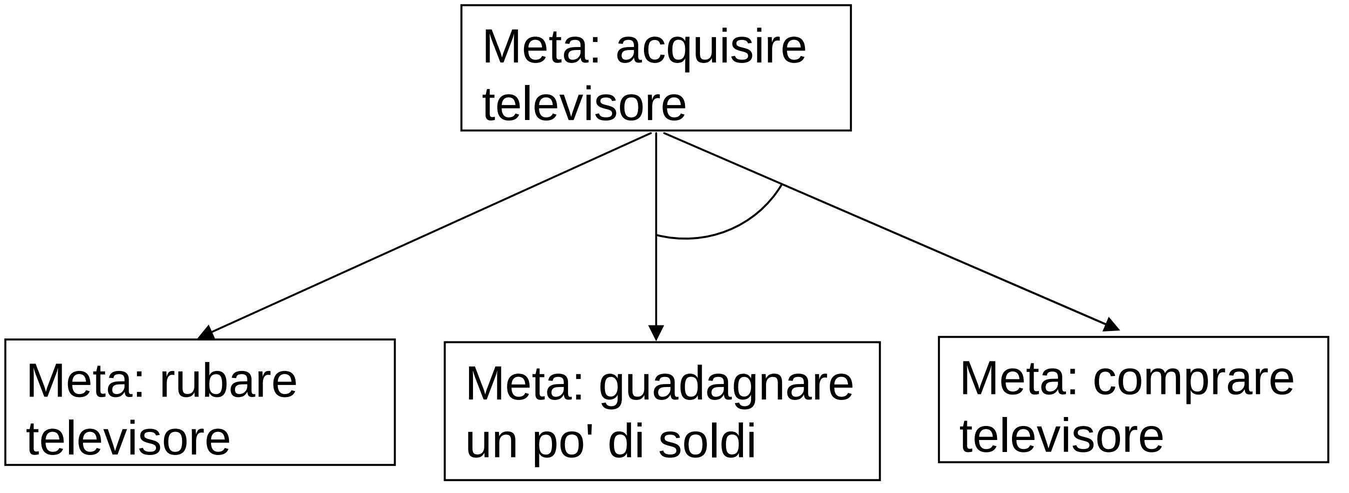
Lo scopo della riduzione a sottoproblemi è arrivare a sottoproblemi con soluzione ovvia (si risolvono con 1 passo nello spazio degli stati o la soluzione è conosciuta).

Si usa rappresentare la *r*. a *s*. mediante grafi particolari:



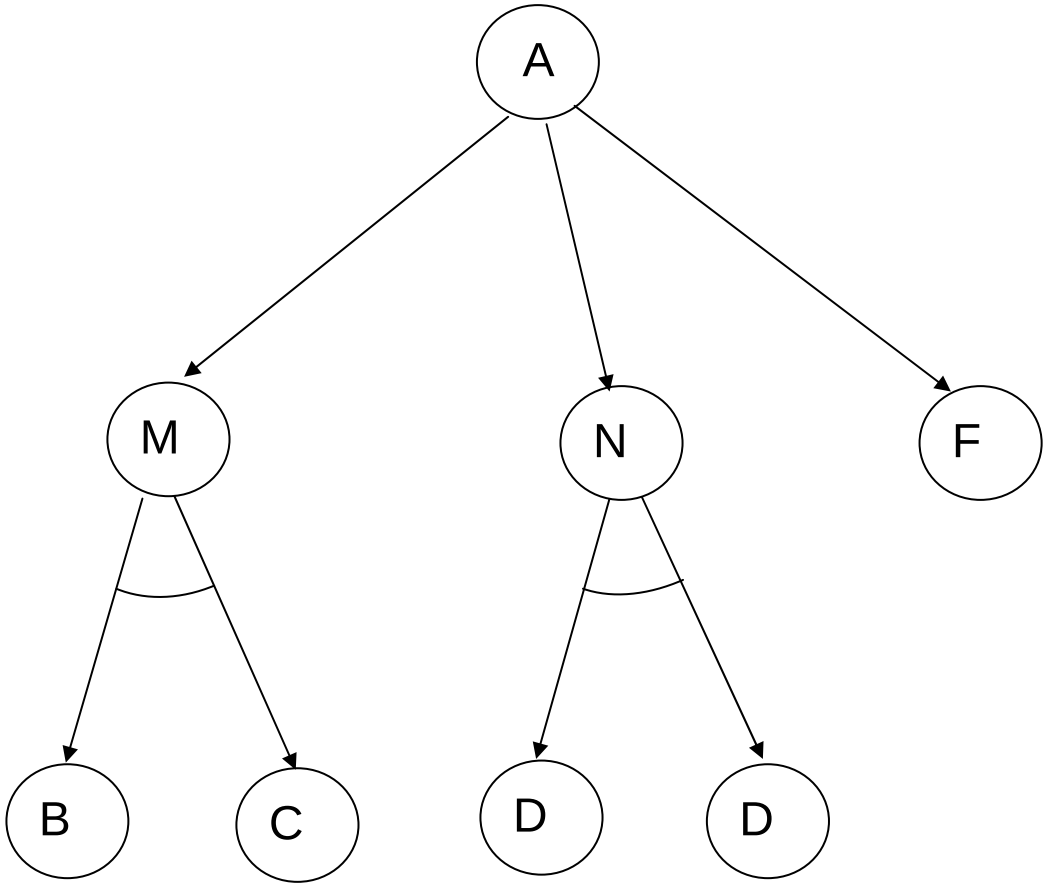
Il grafo AND/OR mostra gli insiemi alternativi di sottoproblemi di *A*.

Esempio:



Ci si può ricondurre allo schema più generale creando (eventualmente) nodi ausiliari.

Nell’esempio seguente *A* assume il significato di “la soluzione del problema”, *M* ed *N* sono nodi "fittizi".



Scopo del processo di ricerca su un grafo AND/OR è mostrare che il nodo di partenza è risolto.

Definizione di nodo risolto (ricorsiva):

1. i nodi terminali sono risolti (poiché sono associati a problemi primitivi);

2. se un nodo non terminale ha successori OR, esso è risolto se e solo se almeno uno dei suoi successori è risolto;

3. se un nodo non terminale ha successori AND, esso è risolto se e solo se *tutti* i suoi successori sono risolti.

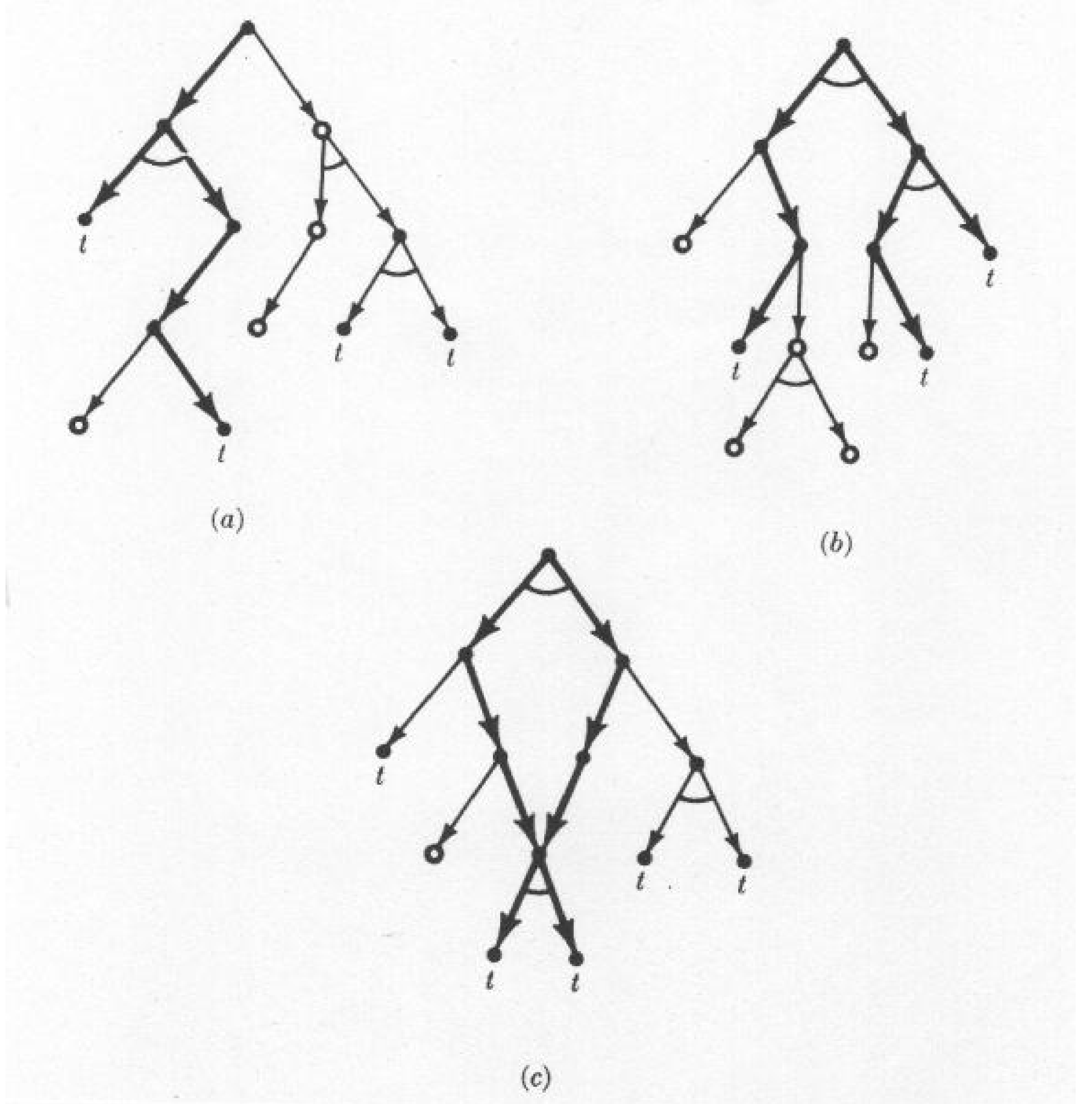
Si definisce *grafo risolutivo* il sottografo di nodi risolti che dimostra (secondo la precedente definizione) che il nodo di partenza è risolto.

In grafo AND/OR un nodo privo di successori è detto insolubile.

Definizione (ricorsiva):

1. i nodi non terminali privi di successori sono insolubili;
2. se un nodo non terminale ha successori OR, è insolubile se e solo *se tutti* i suoi successori sono insolubili;
3. se un nodo non terminale ha successori AND, è insolubile se e solo se almeno *uno* dei suoi successori è insolubile.

Esempi di grafi AND/OR e di grafi risolutivi. Il grafo (c) ha più di una soluzione:



I nodi con ○ sono insolubili.

In questo caso l’operatore di successione Γ viene applicato a una descrizione di un problema e produce l’insieme di descrizione di problemi successori

(cioè l’applicazione di Γ consiste nell’applicazione di *tutti* gli operatori di riduzione ammissibili).

Esempio di applicazione della riduzione a sottoproblemi: l'integrazione simbolica

Si dispone di una tabella di integrali più semplici:

$$\int u\,du = \frac{u^2}{2}$$

$$\int \sin u\,du = -\cos u$$

$$\int a^u du = a^u \log_a e$$

$$etc.$$

Si vuole integrare una funzione *I(x)* rispetto a *x*, cioè:

$$\int I(x)dx$$

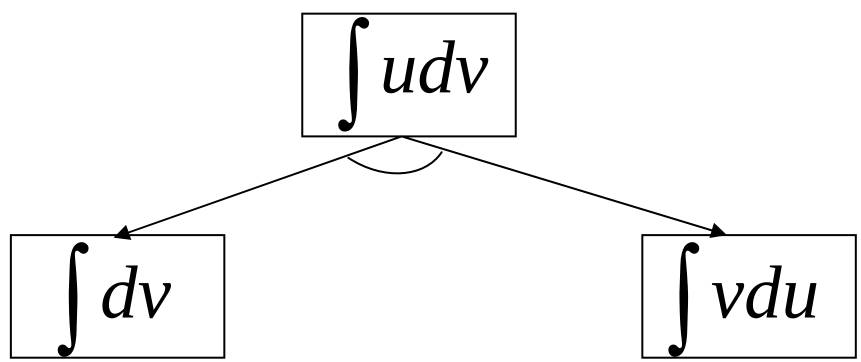
Si dispone di operatori di riduzione a sottoproblemi:

- regola di integrazione per parti
- regola di decomposizione di una somma
- regole di trasformazione con sostituzioni algebriche o trigonometriche
- ecc.

Regola di integrazione per parti:

$$\int u\,dv = u\int dv - \int v\,du$$

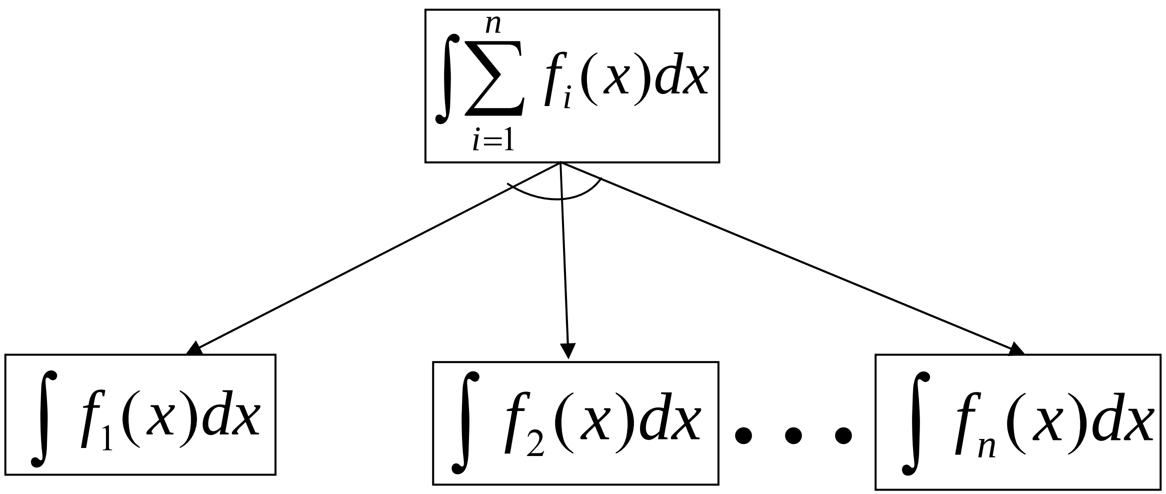
Quindi il grafo corrispondente è:



Regola di decomposizione:

$$\int \sum_{i=1}^{n} f_i(x)dx = \sum_{i=1}^{n} \int f_i(x)dx$$

Il grafo:



Regola di fattorizzazione:

$$\int kf(x)dx = k\int f(x)dx$$

Il grafo:

$$\begin{array}{c}\boxed{\int kf(x)dx}\\ \big\downarrow\\ \boxed{\int f(x)dx}\end{array}$$

Regole di sostituzione:

un'espressione viene sostituita con un'altra (creano nodi OR)

1. Sostituzioni algebriche; esempio:

$$\int \frac{x^2 dx}{(2+3x)^{\frac{2}{3}}} \rightarrow \int \frac{1}{9}\left(z^6 - 4z^3 + 4\right) dz$$

usando

$$z^2 = (2+3x)^{\frac{2}{3}}$$

2. Sostituzioni trigonometriche; esempio:

$$\int \frac{dx}{x^2\sqrt{25x^2+16}} \rightarrow \int \frac{5}{16}\cot\theta\csc\theta d\theta$$

usando

$$x = \frac{4}{5}\tan\theta$$

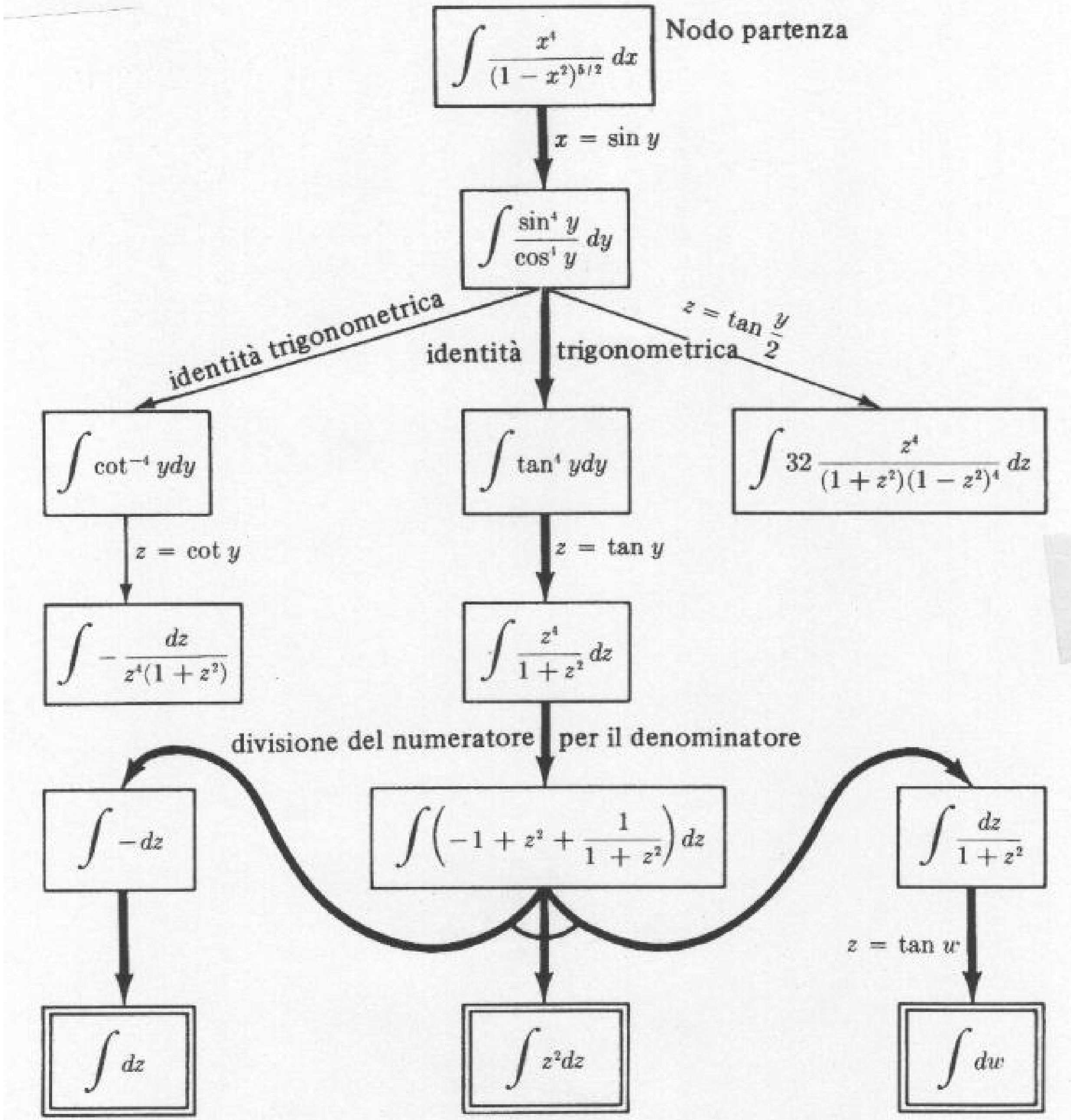
3. Divisione del numeratore per il denominatore; esempio:

$$\int \frac{z^4 dz}{z^2+1} \rightarrow \int \left(z^2 - 1 + \frac{1}{1+z^2} \right) dz$$

4. Completamento del quadrato; esempio:

$$\int \frac{dx}{\left(x^2 - 4x + 13\right)} \rightarrow \int \frac{dx}{\left[\left(x-2\right)^2 + 9\right]^2}$$

Quello che segue è un esempio di un grafo di ricerca AND/OR per un problema di integrazione.



Dal grafo risolutivo (in grassetto) e dalla tabella degli integrali elementari:

$$\int \frac{x^4}{\left(1-x^2\right)^{\frac{5}{2}}}\,dx = \arcsin x + \frac{1}{3}\tan^3\left(\arcsin x\right) - \tan(\arcsin x)^3$$

Pianificazione nella riduzione a sottoproblemi.

Un problema di ricerca nello spazio degli stati è definito dalla terna (S, F, G) dove:

- S = stato/i iniziale/i
- F = insieme di operatori applicati
- G = stato/i finale/i

Se si trova una opportuna successione di stati "*chiave*" $g_1, g_2, \ldots, g_n$ il problema si può ridurre nell'insieme di problemi definiti dalle terne

$$(S, F, \{g_1\}), (\{g_1\}, F, \{g_2\}), \ldots, (\{g_n\}, F, G)$$

dove $\{g_1\} \equiv G_1$, $\{g_2\} \equiv G_2$, ecc.

Come specificare gli insiemi chiave?

Nei problemi reali spesso è facile identificare "*passi cruciali*" per la soluzione (operatori "*chiave*").

Ad esempio, nella torre di Hanoi, è un passo cruciale "*muovi il disco C al piolo 3*".

Sia *f* nell'insieme F un operatore chiave nel problema definito da *(S,F,G)*.

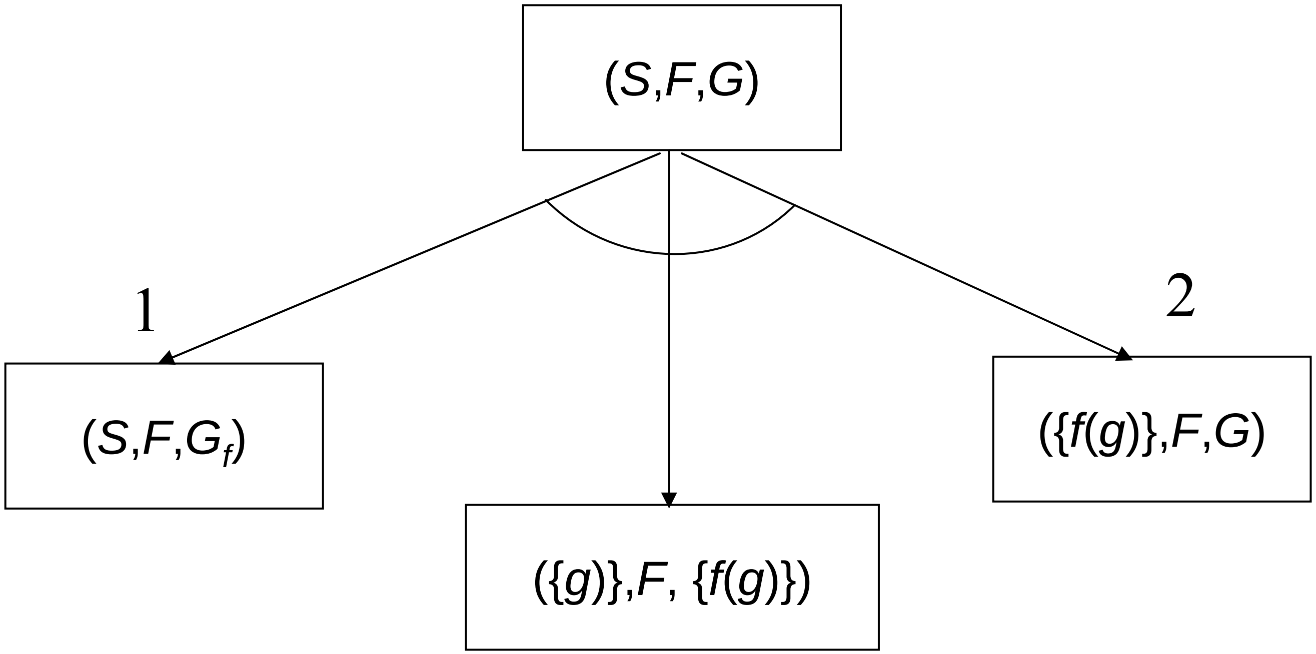
Conviene allora cercare un cammino fino a uno stato in cui *f* è applicabile.

Sia G_f l'insieme degli stati a cui *f* è applicabile. Abbiamo identificato allora il sottoproblema *(S,F,* G_f*)*.

Ora se *g* di G_f è uno stato chiave, $(\{g\}, F, \{f(g)\})$, dove $f(g)$ è lo stato che si ottiene applicando *f* a *g*, è un problema primitivo (basta applicare *f* !).

Ci rimane quindi il sottoproblema descritto da $(f(g), F, G)$ per arrivare alla soluzione.

Pertanto quando si può identificare uno stato chiave, si può usare lo schema:



primitivo, quindi già risolto

Metodo delle differenze (o *analisi mezzi-fini*)

Per trovare gli aspiranti operatori chiave si usa un metodo basato sulle *differenze*.

Una differenza per *(S,F,G)* è una lista parziale dei motivi per cui il criterio di stato finale (l'insieme G) non è soddisfatto dai membri di S.

Gli aspiranti operatori chiave sono quelli che si applicano a una differenza (nel senso che "*rimuovono*" la differenza),

Esempio:

Problema della scimmia e delle banane.

Lo stato è descritto da una lista di 4 elementi (**w**, x, **y**, z) dove:

1. w = posizione orizzontale della scimmia (vettore bidimensionale;

2. x = 1 o 0, a seconda che la scimmia si trovi rispettivamente sulla cassa o a terra;

3. y = posizione orizzontale della cassa (vettore bidimensionale);

4. z = 1 o 0, a seconda che la scimmia rispettivamente abbia o non abbia preso le banane;

E gli operatori:

$$(\mathbf{w}, 0, \mathbf{y}, z) \xrightarrow{goto(u)} (\mathbf{u}, 0, \mathbf{y}, z)$$

$$(\mathbf{w}, 0, \mathbf{w}, z) \xrightarrow{pushbox(v)} (\mathbf{v}, 0, \mathbf{v}, z)$$

$$(\mathbf{w}, 0, \mathbf{w}, z) \xrightarrow{climbox} (\mathbf{w}, 1, \mathbf{w}, z)$$

$$(\mathbf{c}, 1, \mathbf{c}, z) \xrightarrow{grasp} (\mathbf{c}, 1, \mathbf{c}, 1)$$

Lo stato iniziale è: $(\boldsymbol{a}, 0, \boldsymbol{b}, 0)$;
gli operatori: $F = \{f_1, f_2, f_3, f_4\}$;
il goal: $G = (\mathbf{c}, 1, \mathbf{c}, 1)$.

Il problema è dunque descritto da

$$\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0), F, G\}$$

ovvero, poiché F non cambia:

$$\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0), G\}$$

Il motivo per cui lo stato iniziale non soddisfa lo stato meta è che l'ultimo elemento non è 1.

Occorre quindi applicare (“grasp”) come operatore chiave, per ridurre questa differenza.

Il problema si riduce a

$$\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0), G_{f_4}\} \quad \text{e} \quad (\{f_4(s_1)\}, G)$$

dove:

G_{f_4} è l'insieme di stati in cui si può applicare f_4

s_1 è lo stato di G_{f_4} ottenuto dalla soluzione di $\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0), G_{f_4}\}$.

Consideriamo il primo sottoproblema e consideriamo le differenze tra:

$$(\boldsymbol{a}, 0, \boldsymbol{b}, 0) \quad \text{e} \quad G_{f_4}$$

Esse sono:

1. la cassa non si trova in ***c***
2. la scimmia non si trova in ***c***
3. la scimmia non è sulla cassa

Gli operatori che eliminano queste differenze sono:

- f_2 pushbox(***c***)
- f_1 goto(***c***)
- f_3 climbbox

Si applicano a turno questi operatori e si producono coppie di sottoproblemi. Se si applica il primo, f_2, si ottiene:

(1-1) $(\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0)\}, G_{f_2})$
(1-2) $(\{f_2(s_{11})\}, G_{f_4})$

dove s_{11} è ottenuto dalla soluzione della (1-1).

Consideriamo ancora il primo sottoproblema (1-1): per poter spingere la cassa la scimmia deve essere in ***b***. L’operatore che annulla questa differenza è:

f_1 goto(***b***)

Applicando questo, si ottengono i due sottoproblemi:

(1-11) $(\{(\boldsymbol{a}, 0, \boldsymbol{b}, 0)\}, G_{f_1})$

(1-12) $(\{f_1(s_{111})\}, G_{f_2})$

Il primo dei due è primitivo (basta applicare f_1), ovvero, formalmente, ($\boldsymbol{a}$, 0, $\boldsymbol{b}$, 0) è nel dominio di f_1.

Si passa a considerare (1-12), tenendo conto del fatto che, applicando f_1, $f_1(s_{111})$ è in realtà (***b***, 0, ***b***, 0) e quindi:

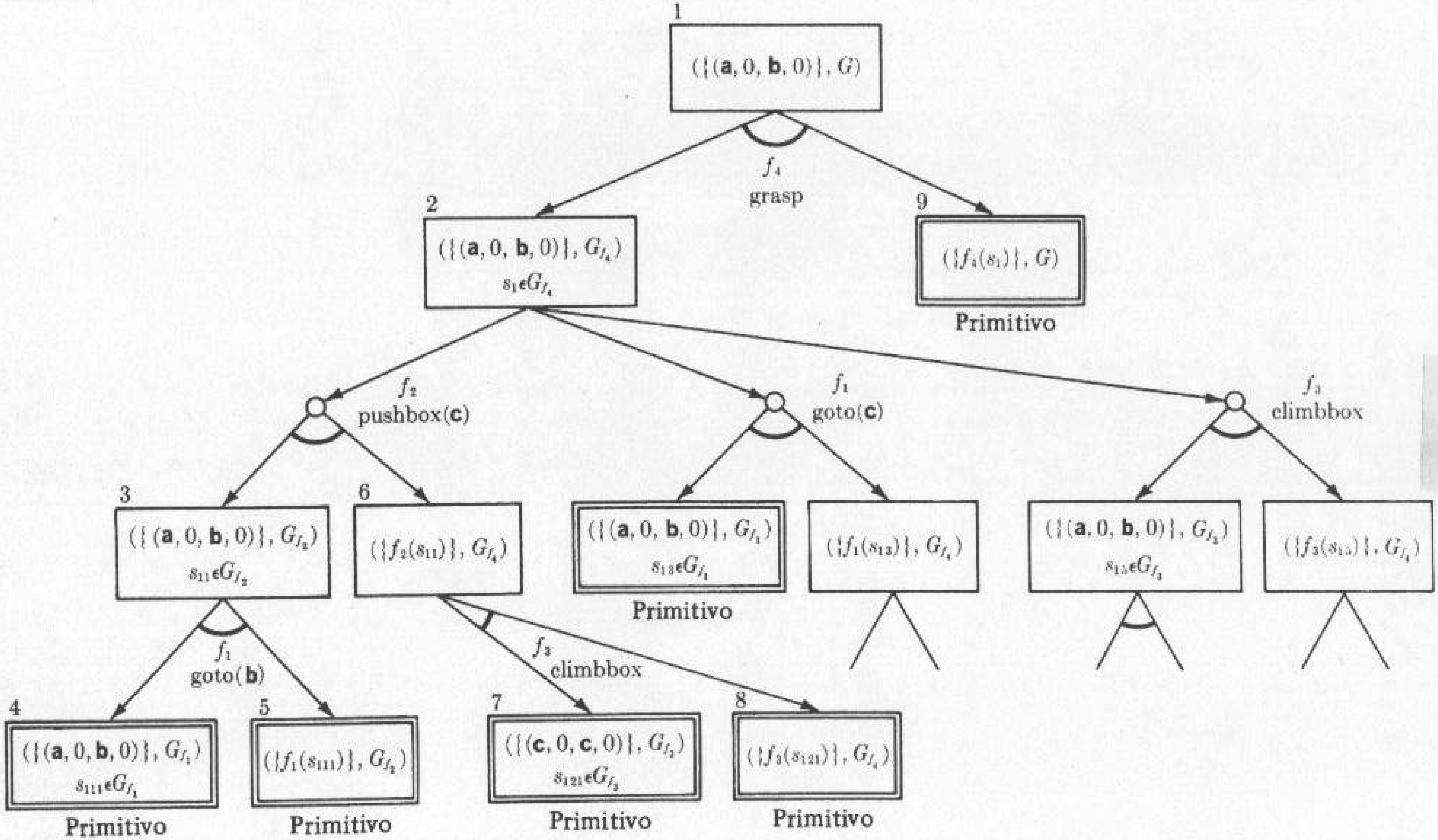
$$(\{(\boldsymbol{b}, 0, \boldsymbol{b}, 0)\}, G_{f_2})$$

Quest'ultimo è primitivo (è sufficiente applicare pushbox(***c***) $\equiv f_2$): infatti (***b***, 0, ***b***, 0) è nel dominio di f_2.

Si procede così per gli altri rami.

La soluzione complessiva può essere resa meglio dal seguente grafo AND/OR che corrisponde al procedimento usato.

– *Grafo AND/OR per il problema della scimmia e delle banane*



Il metodo appena illustrato è la base della strategia di ricerca detta

analisi mezzi-fini.

Il processo di analisi può essere fondato su tabelle che, per ogni operatore, evidenziano la differenza (o le differenze) che l'operatore elimina. Queste liste sono solitamente ordinate rispetto a una qualche priorità.

Inoltre queste liste devono includere le condizioni che devono essere soddisfatte per l’applicabilità delle regole.

Su questo metodo sono basati i sistemi quali il GPS (General Problem Solver), sviluppato da Newell & Ernst.

Ricerca su grafi AND/OR

Si rammentino le definizioni:

Nodo risolto:

1. i nodi terminali sono risolti (poiché sono associati a problemi primitivi);
2. se un nodo non terminale ha successori OR, esso è risolto se e solo se almeno uno dei suoi successori è risolto;
3. se un nodo non terminale ha successori AND, esso è risolto se e solo se *tutti* i suoi successori sono risolti.

Nodo insolubile:

1. i nodi non terminali privi di successori sono insolubili;
2. se un nodo non terminale ha successori OR, è insolubile se e solo *se tutti* i suoi successori sono insolubili;
3. se un nodo non terminale ha successori AND, è insolubile se e solo se almeno *uno* dei suoi successori è insolubile.

Lo schema generale di risoluzione è:

1. si associa un nodo di partenza alla descrizione del problema iniziale;

2. si calcolano insiemi di successori del nodo di partenza applicando i possibili operatori di riduzione a sottoproblemi. Sia Γ l'operatore combinato che calcola tutti i successori di un nodo; ancora chiameremo *espansione* di un nodo il processo di applicazione di Γ al nodo (si ricordi che se si genera più di un insieme di successori AND ogni insieme non unitario va raggruppato sotto un nodo OR intermedio);

3. si predispongono *puntatori* da ogni nodo successore al nodo genitore. Questi vengono utilizzati nel processo di etichettatura dei nodi risolti e insolubili, e indicano il grafo risolutivo dopo la terminazione;

4. si continua il processo di espansione dei nodi e di predisposizione dei puntatori finché si può etichettare il nodo di partenza come risolto o insolubile.

Ricerca in ampiezza (per gli alberi):

1. porre il nodo *s* di partenza in una lista di nome OPEN;

2. rimuovere il primo nodo di OPEN e porlo in una lista di nome CLOSED, chiamandolo *n*;

3. espandere il nodo *n*, generandone tutti i successori; porre questi alla *fine* di OPEN predisponendo puntatori ad *n*. Se non ci sono successori, etichettare *n* come insolubile e proseguire, altrimenti andare a 8.

4. applicare il procedimento di etichettatura dei nodi insolubili all'albero di ricerca;

5. se il nodo di partenza è etichettato come insolubile, uscire con un fallimento, altrimenti continuare;

6. rimuovere da OPEN tutti i nodi con antenati insolubili (questo passo ci permette di evitare lo sforzo superfluo di tentare di risolvere problemi insolubili);

7. andare a 2;

8. se alcuni dei successori sono nodi terminali, etichettarli come risolti e proseguire, altrimenti andare a 2;

9. applicare il procedimento di etichettatura dei nodi risolti all'albero di ricerca;

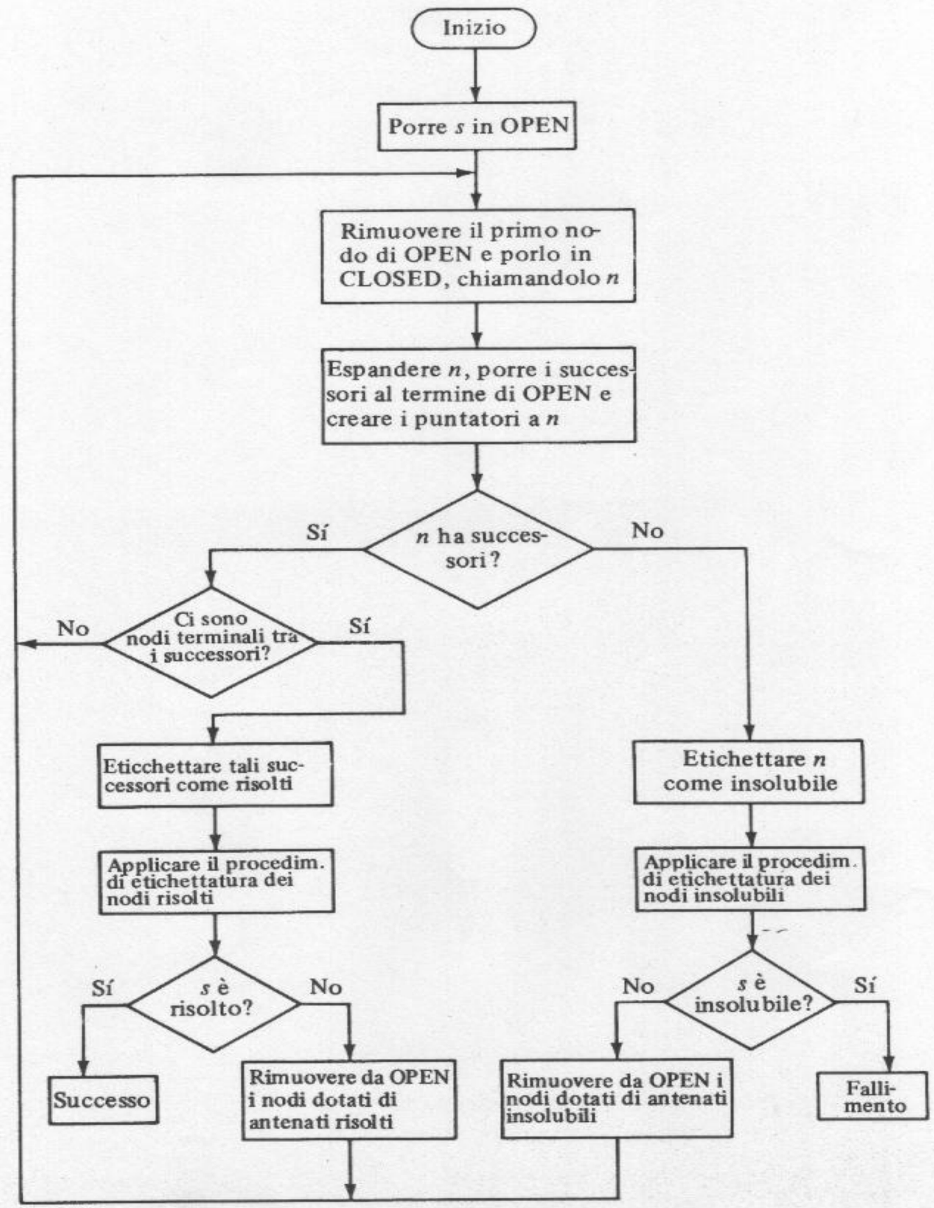
10. se il nodo di partenza è etichettato come risolto, uscire con l'albero risolutivo che verifica ciò; altrimenti continuare;

11. rimuovere da OPEN tutti i nodi risolti o con antenati risolti (questo passo ci permette di evitare lo sforzo superfluo di risolvere un problema in più di un modo);

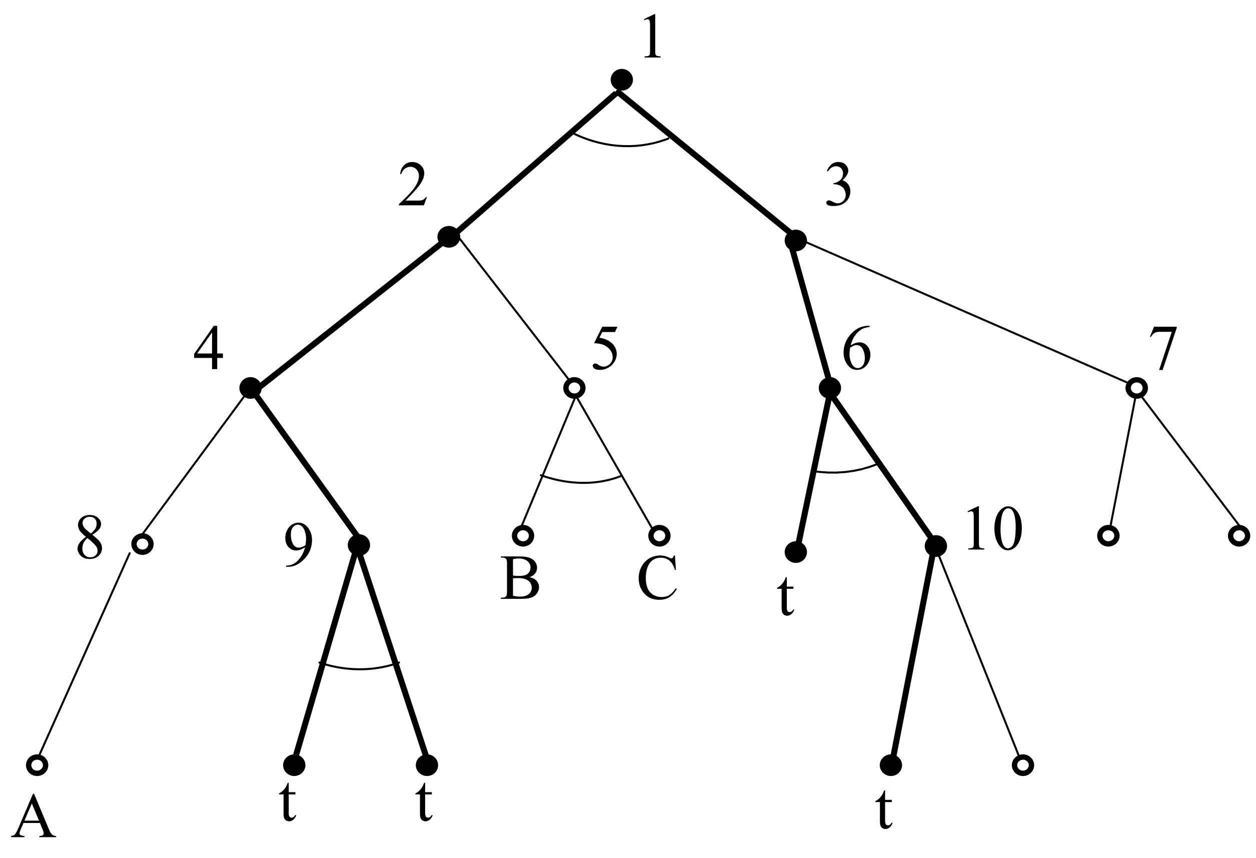
12. andare a 2.

Può essere utile in questo caso lo schema a blocchi di questo algoritmo:

– *Schema a blocchi della procedura di ricerca in ampiezza per gli alberi AND/OR*



L'esempio che segue mostra l'ordine di espansione dei nodi nella ricerca in ampiezza in un albero AND/OR.



Ricerca in profondità (per gli alberi):

Definizioni:

Come per gli alberi ordinari, si definisce la *profondità* di un nodo in un albero AND/OR come segue:

- La profondità del nodo di partenza è zero
- La profondità di ogni altro nodo è uguale alla profondità del suo genitore più 1.

Algoritmo:

avvertenza: i nodi non terminali che si trovano oltre un limite di profondità vengono etichettati come insolubili.

1. porre il nodo *s* di partenza in una lista di nome OPEN;

2. rimuovere il primo nodo di OPEN e porlo in una lista di nome CLOSED, chiamandolo *n*;

3. se la profondità di *n* è uguale al limite di profondità etichettare *n* come insolubile e andare a 5, altrimenti proseguire;

4. espandere il nodo *n*, generandone tutti i successori. Porre questi all'inizio di OPEN (in ordine arbitrario) e predisporre puntatori a *n;* se non ci sono successori etichettare *n* come insolubile e continuare, altrimenti andare a 9;

5. applicare all'albero di ricerca il procedimento d'etichettatura dei nodi insolubili;

6. se il nodo di partenza è stato etichettato come insolubile, uscire con un fallimento; altrimenti proseguire;

7. rimuovere da OPEN tutti i nodi con antenati insolubili;

8. andare a 2;

9. se alcuni dei successori sono nodi terminali etichettarli come risolti e proseguire, altrimenti andare a 2;

10. applicare all'albero di ricerca il procedimento di etichettatura dei nodi risolti;

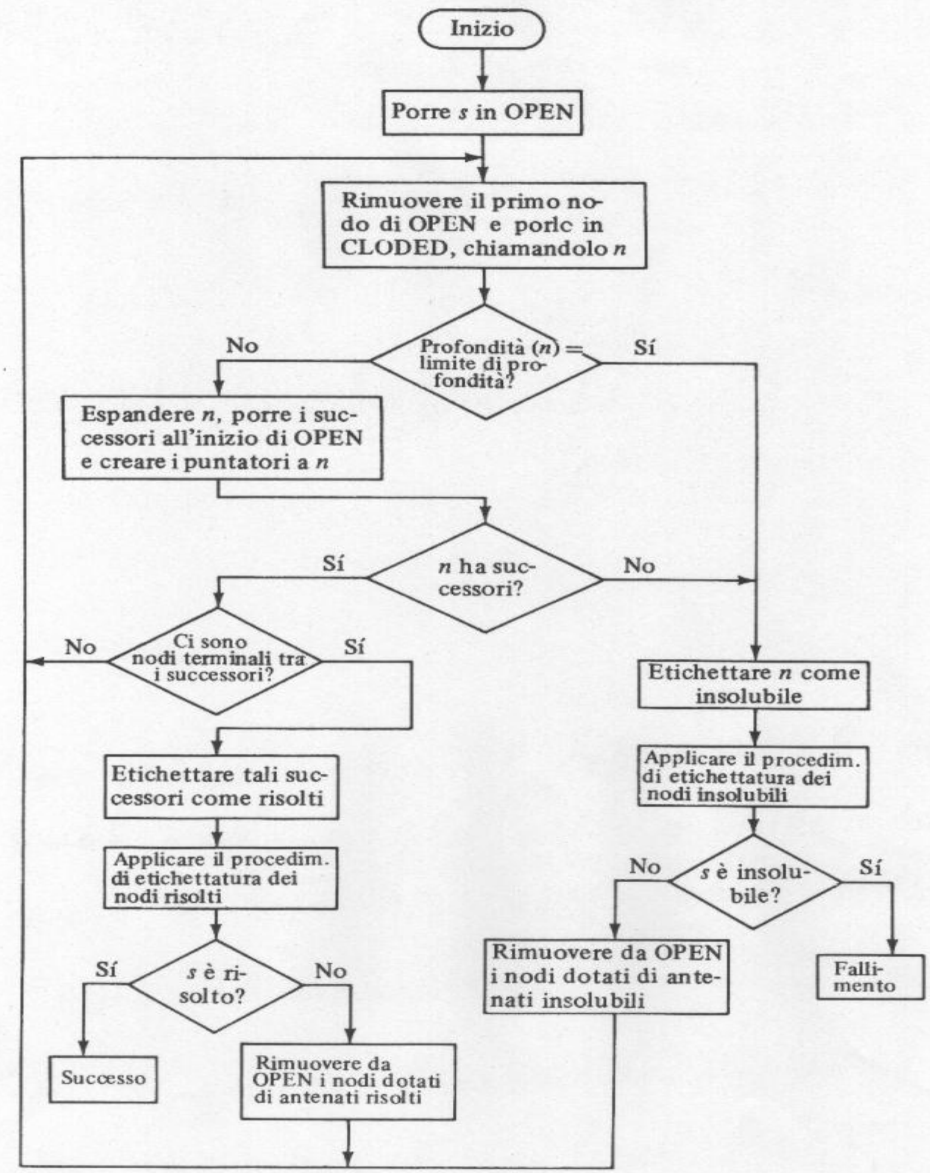
11. se il nodo di partenza viene etichettato come risolto, uscire con l'albero risolutivo che verifica ciò; altrimenti proseguire;

12. rimuovere da OPEN i nodi risolti o con antenati risolti;

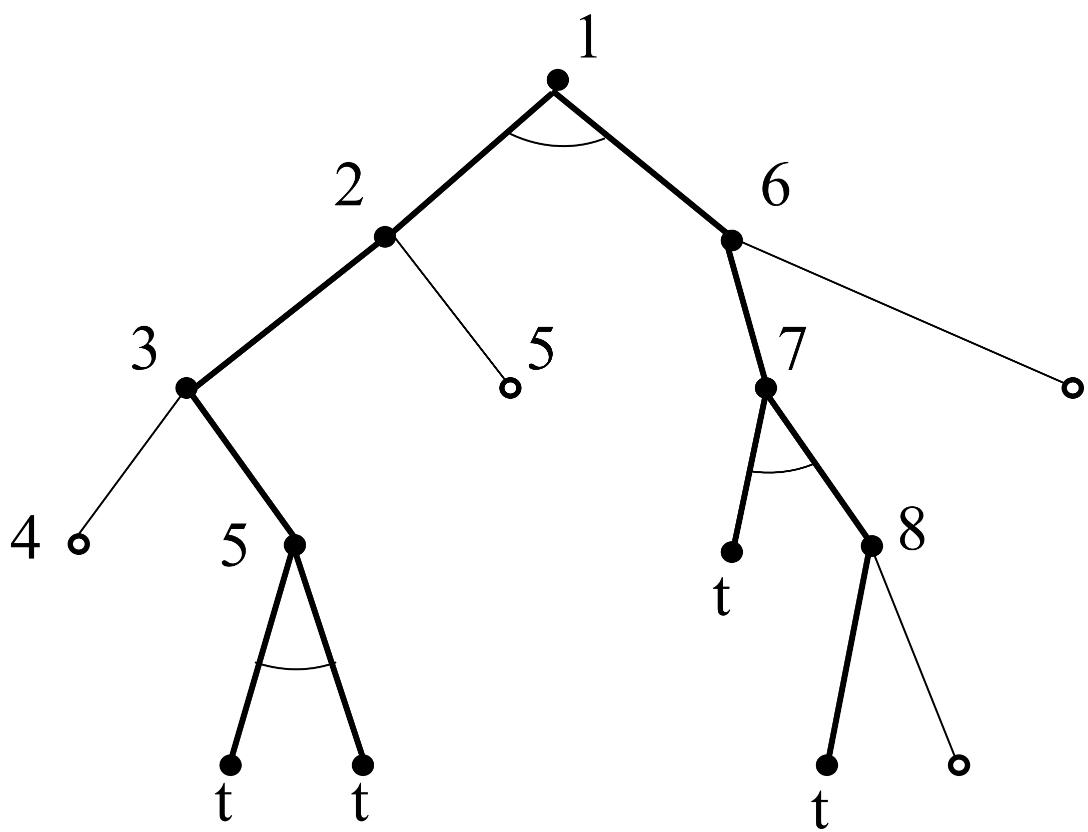
13. andare a 2.

Ecco lo schema a blocchi dello stesso algoritmo:

– *Schema a blocchi di una procedura di ricerca in profondità per gli alberi AND/OR*



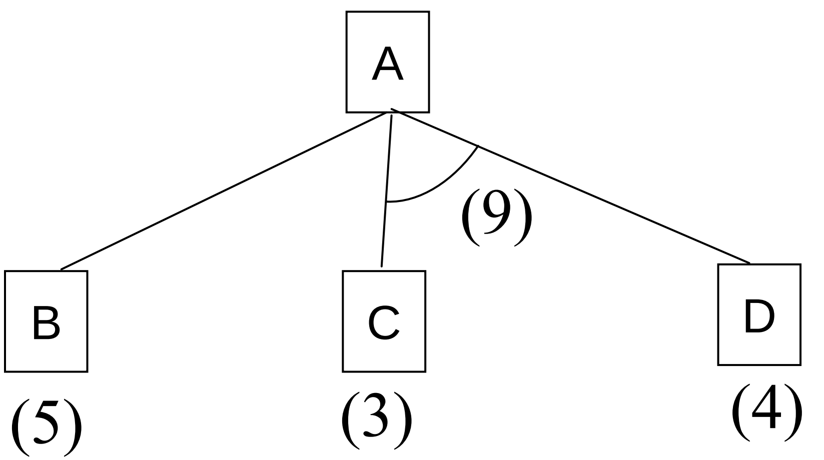
L'esempio che segue mostra l'ordine di espansione dei nodi nella ricerca in profondità (limite di profondità = 4).



Ricerca su grafi AND/OR

Per trattare grafi AND/OR occorre un algoritmo come , ma in più capace di trattare gli archi AND. Infatti è inadeguato.

Esempio: si abbia il grafo:

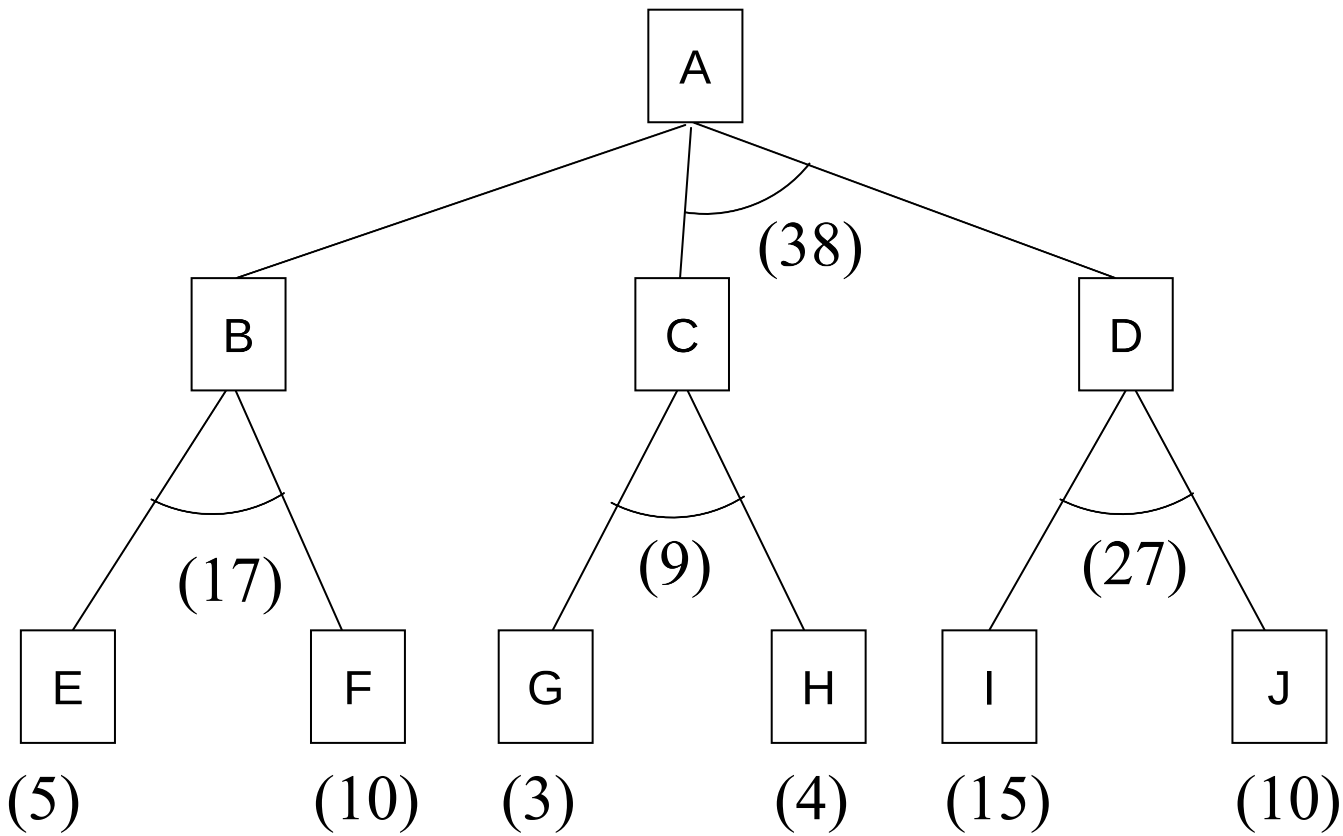


In parentesi sia il valore di $\hat{f}$ per quel nodo e gli archi abbiano costo pari ad 1 (NB: negli archi AND i costi si sommano).

Se si guarda localmente il nodo da espandere è C (con $\hat{f} = 3$). Tenendo conto del ramo AND, il costo totale è 9 (cioè D+C+2), mentre se si passa per B il costo è 5+1=6.

La scelta del nodo da espandere dipende non solo dal valore di $\hat{f}$ in quel nodo, ma anche dal fatto che quel nodo faccia o meno parte dell'attuale cammino ottimo a partire dal nodo iniziale.

Al passo successivo il problema diventa più evidente:



G, con $\hat{f} = 3$, sembra il nodo più promettente. Fa parte dell'arco costituito da G-H, con costo=9 (quindi ancora promettente). Ma per usarlo occorre usare anche I-J con costo 27.

Quindi il cammino che passa per B e arriva a E-F ha un costo minore (in B il costo vale 18) ed è quindi ancora in gioco.

Per la ricerca su un grafo AND/OR occorre:

- Attraversare il grafo a partire dal nodo iniziale seguendo il cammino ottimo attuale; accumulare l'insieme dei nodi presenti sul cammino e non ancora espansi

- Prendere uno dei nodi non espansi ed espanderlo. Aggiungere i successori al grafo e calcolare $\hat{f}$ (si usa solo h)

- Cambiare la stima di $\hat{f}$ del nodo appena espanso (per tener conto dei successori). Propagare all'indietro questo cambiamento. Ad ogni nodo visitato mentre si risale il grafo, decidere quale dei suoi archi successori è il più promettente e marcarlo come parte dell'attuale cammino migliore.

N.B:

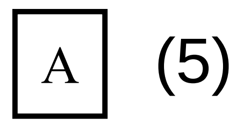
- l’attuale cammino ottimo può cambiare

- la propagazione della stima all’indietro non era necessaria in A^*, perché si esaminavano solo i nodi non espansi. Qui, invece, i nodi espansi devono essere riesaminati.

L’algoritmo prende il nome di AO^*

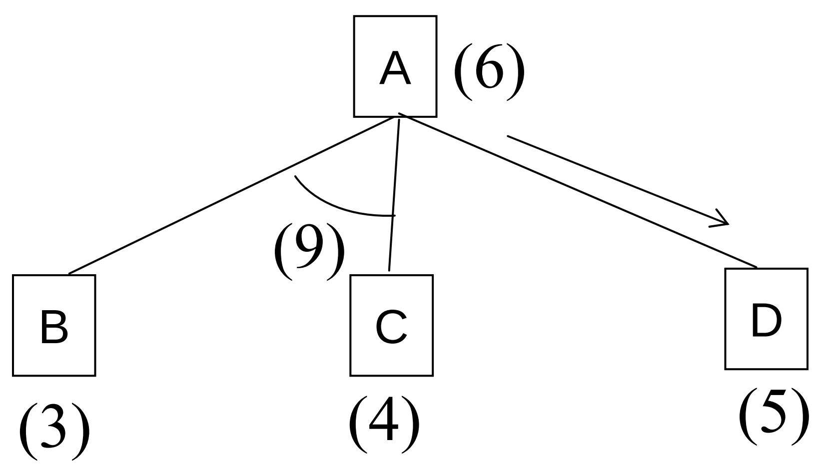
Esempio di funzionamento

Prima del passo 1



Il nodo di partenza fa parte del cammino ottimo. Viene espanso.

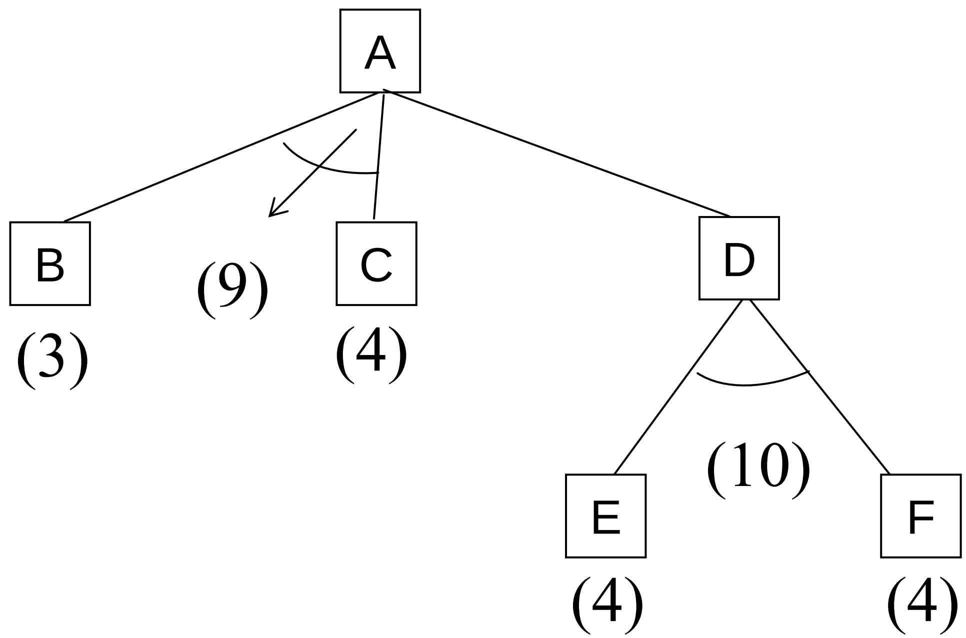
Prima del passo 2



L'arco verso D (costo 5+1=6) viene marcato come il più promettente. Infatti il ramo B-C ha costo 9 (=3+4+1+1).

Si espande D:

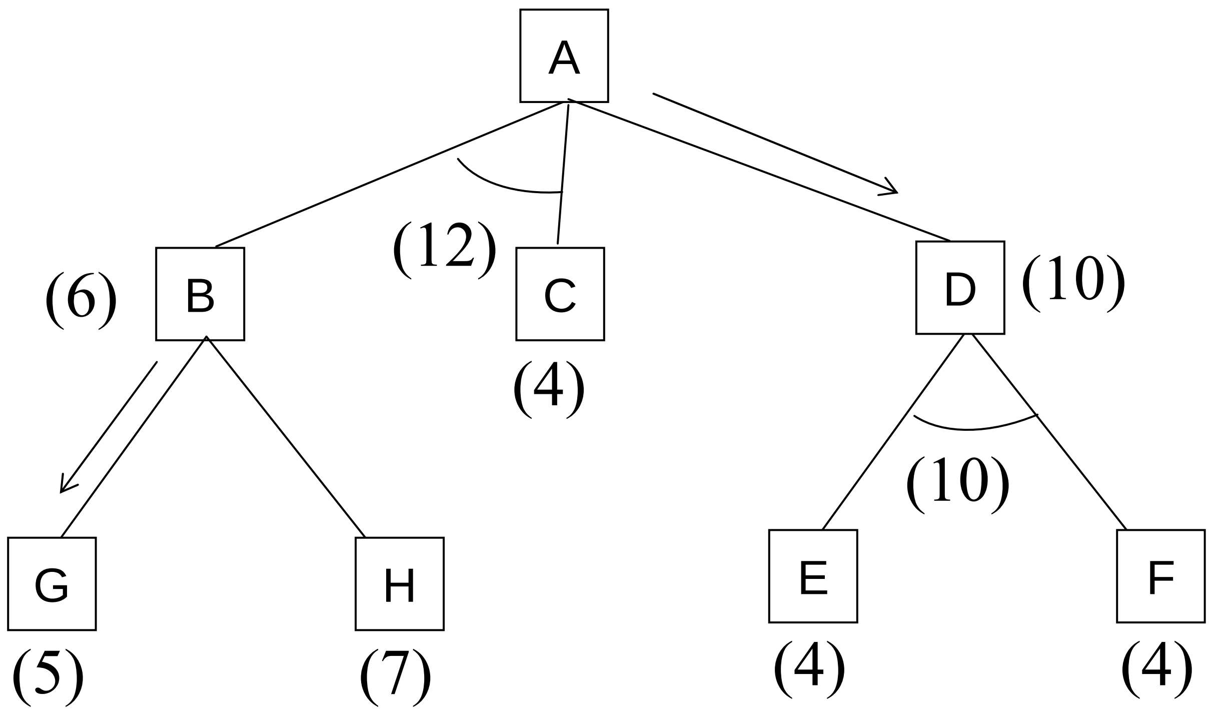
Prima del passo 3



Poiché si produce l’arco AND verso E e F con costo 10, si aggiorna a 10 il valore del costo di D. Si torna indietro ancora di un passo e si nota a questo punto che l’arco AND costituito da B-C è ora più conveniente (9 contro 10).

L’attuale cammino ottimo deve essere cambiato e diventa quello verso B-C. Si attraversa questo ramo e si incontrano B e C da espandere.

Si comincia con l’esplorare B. Espandendo B si ottiene:



Da B si va a G con costo 6(=5+1), si va ad H con costo 8(=7+1).

Si sceglie di andare verso G. A questo punto si propaga indietro questo costo, aggiornando il valore di B che diventa 6 e poi quello del ramo B-C che diventa 12(=6+4+1+1).

L’arco verso D diventa di nuovo il cammino migliore. A questo punto si procede espandendo E oppure F.

Si procede così finché non si trova una soluzione ovvero si verifica che non c’è una soluzione (si trovano vicoli ciechi).

Il concetto di «costo».

Nella ricerca negli spazi degli stati è stata usata una funzione di valutazione euristica (stima il costo di un cammino ottimo tra un nodo e l’obiettivo).

Per gli alberi AND/OR occorre introdurre il concetto di costo di un albero risolutivo radicato in un dato nodo.

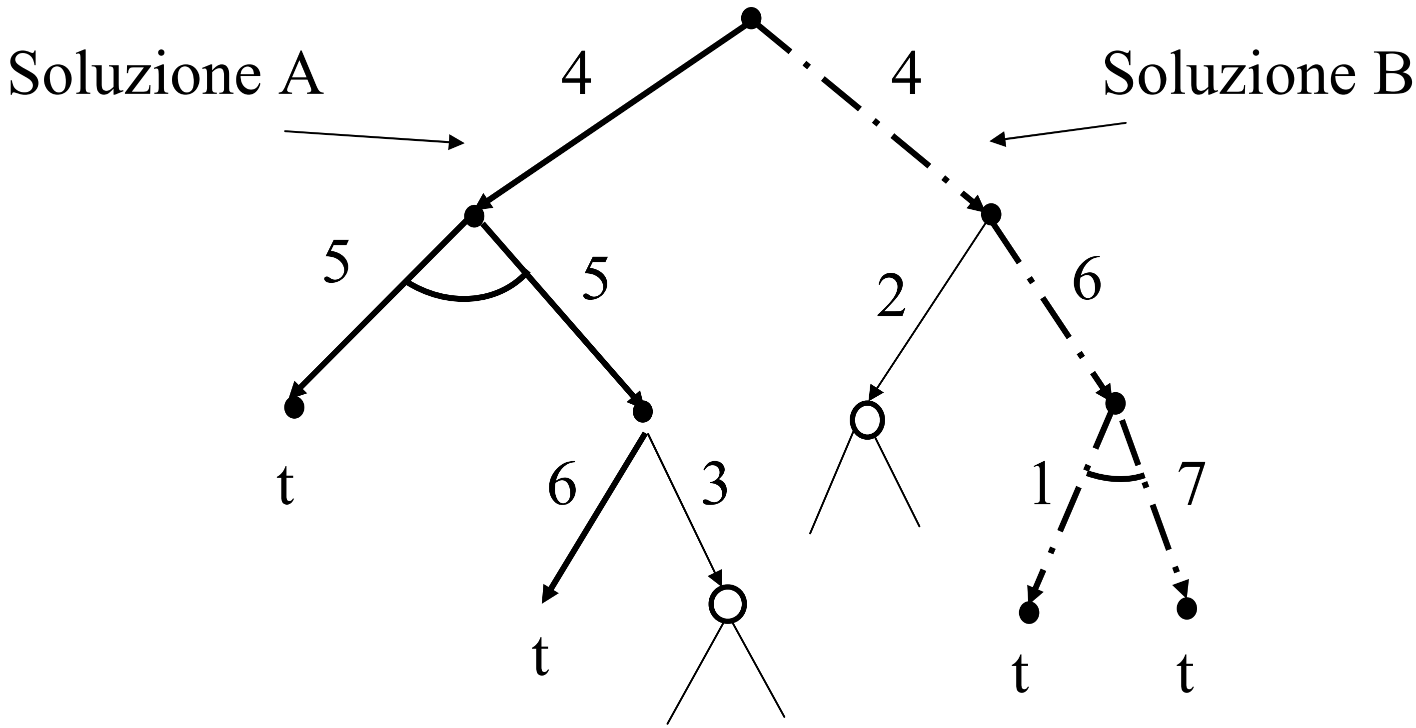
Iª definizione:

Costo complessivo: somma dei costi di tutti gli archi nell'albero risolutivo

IIª definizione:

Costo massimo: costo del cammino a massimo costo nell'albero risolutivo. Un cammino radicato in n è una successione di nodi da un nodo iniziale n a un nodo finale n_k di cui uno è successore dell'altro.

Ecco due alberi risolutivi e i loro costi:



Soluzione A
Costo complessivo = 20
Costo massimo = 15

Soluzione B
Costo complessivo = 18
Costo massimo = 17

Si cerca un albero risolutivo a minimo costo all'interno dell'intero albero AND/OR implicito (*albero risolutivo* ≡ *sottoalbero*). Sarà detto *albero risolutivo ottimo*.

Sia $h(s)$ il costo di un albero risolutivo ottimo radicato nel nodo di partenza s.

Se $c(n_i, n_j)$ è il costo dell'arco fra n_i ed il successore n_j , valgono le seguenti definizioni:

1. se n è un nodo terminale (corrispondente a un problema primitivo)

$$h(n = 0)$$

2. se n è un nodo non terminale con successori OR n_1, n_2, …, n_k,

$$h(n) = \min_i [c(n, n_i) + h(n_i)]$$

3. se n è un nodo non terminale con successori AND n_1, n_2, …, n_k,

$$h(n) = \sum_{i=1}^{k} [c(n, n_i) + h(n_i)]$$

per il costo complessivo

$$h(n) = \max_i [c(n, n_i) + h(n_i)]$$

per il costo massimo

Naturalmente è indefinito se n è un nodo insolubile.

Anche per l'albero AND/OR si farà uso del concetto di *costo stimato*, cioè si introduce una funzione $\hat{h}(n)$ che è una stima di $h(n)$, costo di un albero risolutivo ottimo radicato in n. $\hat{h}$ è una *funzione euristica*.

Ad ogni passo, all’estremo dell’albero di ricerca ci possono essere nodi, detti foglie, che ricadono in uno di questi tre casi :

1. nodi già identificati come terminali dal processo di ricerca;
2. nodi già identificati come non terminali privi di successori dal processo di ricerca;
3. nodi i cui successori non sono ancora stati generati dal processo di ricerca.

Quindi per ogni nodo *n* dell'albero di ricerca, la funzione $\hat{h}(n)$ può essere così definita:

1. se *n* è un nodo foglia:
   1. se *n* è stato identificato come terminale $\hat{h}(n) = 0$;
   2. se *n* è stato identificato come non terminale privo di successori, $\hat{h}(n)$ è indefinito;
   3. se *n* è un nodo i cui successori non sono ancora stati generati $\hat{h}(n)$ è una stima euristica del costo $h(n)$ di un albero risolutivo ottimo radicato nel nodo *n*; tale stima si deve basare sulla conoscenza euristica disponibile sul dominio del problema rappresentato dall'albero AND/OR.

2. se *n* è un nodo non foglia con successori OR n_1, n_2, …, n_k:

$$\hat{h}(n) = \min_i \left[c(n, n_i) + \hat{h}(n_i) \right]$$

3. Se *n* è un nodo non foglia con successori AND n_1, n_2, …, n_k,

$$\hat{h}(n) = \sum_{i=1}^{k} \left[c(n, n_i) + \hat{h}(n_i) \right]$$

per il costo complessivo

$$\hat{h}(n) = \max_i \left[c(n, n_i) + \hat{h}(n_i) \right]$$

per il costo massimo

Si introduce ancora il concetto di *albero risolutivo potenziale per s*, cioè un sottoalbero radicato in s che potrebbe divenire la parte superiore di un albero risolutivo completo.

Ad ogni passo della ricerca si estrae l'albero risolutivo potenziale τ_0 radicato in s che si ***stima*** essere la parte superiore di un albero risolutivo ottimo radicato in s.

I valori di usati come stime sono basati su queste definizioni:

1. il nodo di partenza appartiene a τ_0
2. se il nodo *n* appartiene a τ_0, allora:
    1. se il nodo *n* ha successori OR $n_1, n_2, \ldots, n_k$ nell'albero di ricerca il successore con il minimo valore di $\lfloor c(n,n_i) + \hat{h}(n_i) \rfloor$ appartiene a τ_0 (eventuali conflitti sono risolti arbitrariamente);
    2. se il nodo *n* ha successori AND nell'albero di ricerca, tutti questi successori appartengono a τ_0.

In somma, la strategia che verrà usata sarà quella di estendere l'albero risolutivo potenziale più promettente (e non il nodo più promettente).

Ma con quale criterio? Un buon criterio è scegliere un nodo la cui espansione ha la maggiore provabilità di ***refutare*** l'ipotesi che τ_0 sia veramente la parte superiore di un albero risolutivo ottimo (in modo da tagliare al più presto i rami inutili!).

Algoritmo AO^*

1. porre il nodo di partenza s in una lista di nome OPEN e calcolare $\hat{h}(s)$;

2. determinare l'albero risolutivo potenziale τ_0 , che si ritiene essere la parte superiore dell'albero risolutivo ottimo radicato in s, usando i valori di ai nodi dell'albero di $\hat{h}$ ricerca;

3. scegliere una foglia di τ_0 che si trovi in OPEN e porla in CLOSED, chiamandola *n*;

4. se *n* è un nodo terminale etichettarlo come risolto e continuare, altrimenti andare a 9;

5. applicare a τ_0 il procedimento di etichettatura dei nodi risolti;

6. se il nodo di partenza è etichettato come risolto uscire con τ_0 come albero risolutivo, altrimenti continuare;

7. rimuovere da OPEN i nodi con antenati risolti;

8. andare a 2;

9. applicare a *n* l'operatore di successione Γ, generandone tutti i successori. Se non ci sono successori etichettare *n* come insolubile e continuare, altrimenti andare a 14;

10. applicare a il procedimento di etichettatura dei nodi insolubili;

11. se il nodo di partenza è etichettato come insolubile uscire con un fallimento, altrimenti continuare;

12. rimuovere da OPEN i nodi con antenati insolubili;

13. andare a 2;

14. porre i successori in OPEN e predisporre i puntatori a *n*. Calcolare i valori di $\hat{h}$ per i successori e ricalcolare i valori di $\hat{h}$ per *n* e i suoi antenati;

15. andare a 2.

Esempio (sulla falsariga di quello già analizzato. Anche qui i costi degli archi valgono 1. Accanto ad ogni nodo c'è un valore di $\hat{h}$. I nodi terminali hanno $\hat{h} = 0$).

– La successione degli alberi di ricerca prodotti da un algoritmo di ricerca ordinata