Intelligenza Artificiale

Anno Accademico 2022 - 2023

Ricerca Euristica

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

- Introduzione alla Ricerca Euristica
- Ricerca Best First
- Greedy Best-First Search
- Esempi di problemi
- L'Algoritmo A*

Ricerca Euristica

Usa la conoscenza specifica sul problema.

EURISTICA: che aiuta nella scoperta

Nell'algoritmo GENERAL-SEARCH si può usare conoscenza euristica per definire la QUEUING-FUNCTION - che è ciò che determina la scelta del nodo da espandere.

La conoscenza sul problema è rappresentata da una

Funzione di valutazione

che si applica ai nodi dell'albero di ricerca.

$$\mathbf{f:\ nodo} \mapsto \begin{cases} \text{stima della desiderabilità di espandere il nodo;} \\ \text{misura di quanto il nodo è "promettente"} \end{cases}$$

N.B. Normalmente f è una stima del costo della soluzione, quindi viene considerato migliore il nodo n con $f(n)$ minore

Best-First Search (1)

- La QUEUING-FN ordina i nodi dal migliore al peggiore secondo la funzione di valutazione.
- Il nodo scelto per l'espansione è quello la cui valutazione è "migliore" (e.g., se si usa una funzione di costo è il nodo con la ***f*** minore), cioè il nodo apparentemente più promettente.
- Diverse funzioni di valutazione determinano diverse versioni della ricerca BEST-FIRST.

Best-First Search (2)

Negli algoritmi di ricerca euristica, la misura incorpora una stima del costo del miglior cammino dal nodo a una soluzione: conoscenza euristica.

La **ricerca guidata dal costo** si può considerare un caso particolare di ricerca best-first, in cui

$$\boxed{f = g}$$

La funzione di valutazione è $g(n)$: il costo del cammino dallo stato iniziale a n.
Informazione euristica nulla

Best-First Search (3)

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution or
failure

QUEUING-FN ← a function that orders nodes by EVAL-FN
fringe ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem])
loop do
     If EMPTY?(fringe) then return failure
     node ← REMOVE-FRONT(fringe)
     if GOAL-TEST(problem, STATE[node]) then return SOLUTION(node)
     fringe ← QUEUING-FN(fringe, EXPAND(node, OPERATORS(problem)))
end
```

Notazioni

Notazioni che riguardano lo spazio degli stati e costi effettivi:

$\boxed{s, s_0, s_1, s_2, ...}$: stati del problema

$\boxed{s_0}$: stato iniziale

$\boxed{k^*(s_i, s_j)}$: costo del cammino a costo minimo da s_i a s_j (se esiste)

$\boxed{g^*(s_i)} = k^*(s_0, s_i)$: costo del cammino a costo minimo dallo stato inziale a s_i

$\boxed{h^*(s_i)}$: costo effettivo di un cammino a costo minimo da s_i a uno stato obiettivo

$\boxed{f^*(s_i)} = g^*(s_i) + h^*(s_i)$: costo minimo di una soluzione vincolata a passare per s_i

NOTAZIONI

Notazioni che riguardano l'albero di ricerca:

$\boxed{n, n_1, n_2}$: nodi dell'albero di ricerca

Notazioni che riguardano l'albero di ricerca e costi effettivi:

$\boxed{g^*(n)} = g^*(STATE(n))$

$\boxed{h^*(n)} = h^*(STATE(n))$

Notazioni che riguardano l'albero di ricerca e la stima di costi:

$\boxed{g(n)}$: costo del cammino dallo stato della radice dell'albero allo stato di *n*

$$g(n) \geq g^*(n)$$

$\boxed{h(n)}$: **stima di** $h^*(n)$ = stima del costo del cammino a costo minimo dallo stato di n a uno stato obiettivo

h **: FUNZIONE EURISTICA**

Greedy Best-First Search (ricerca golosa)

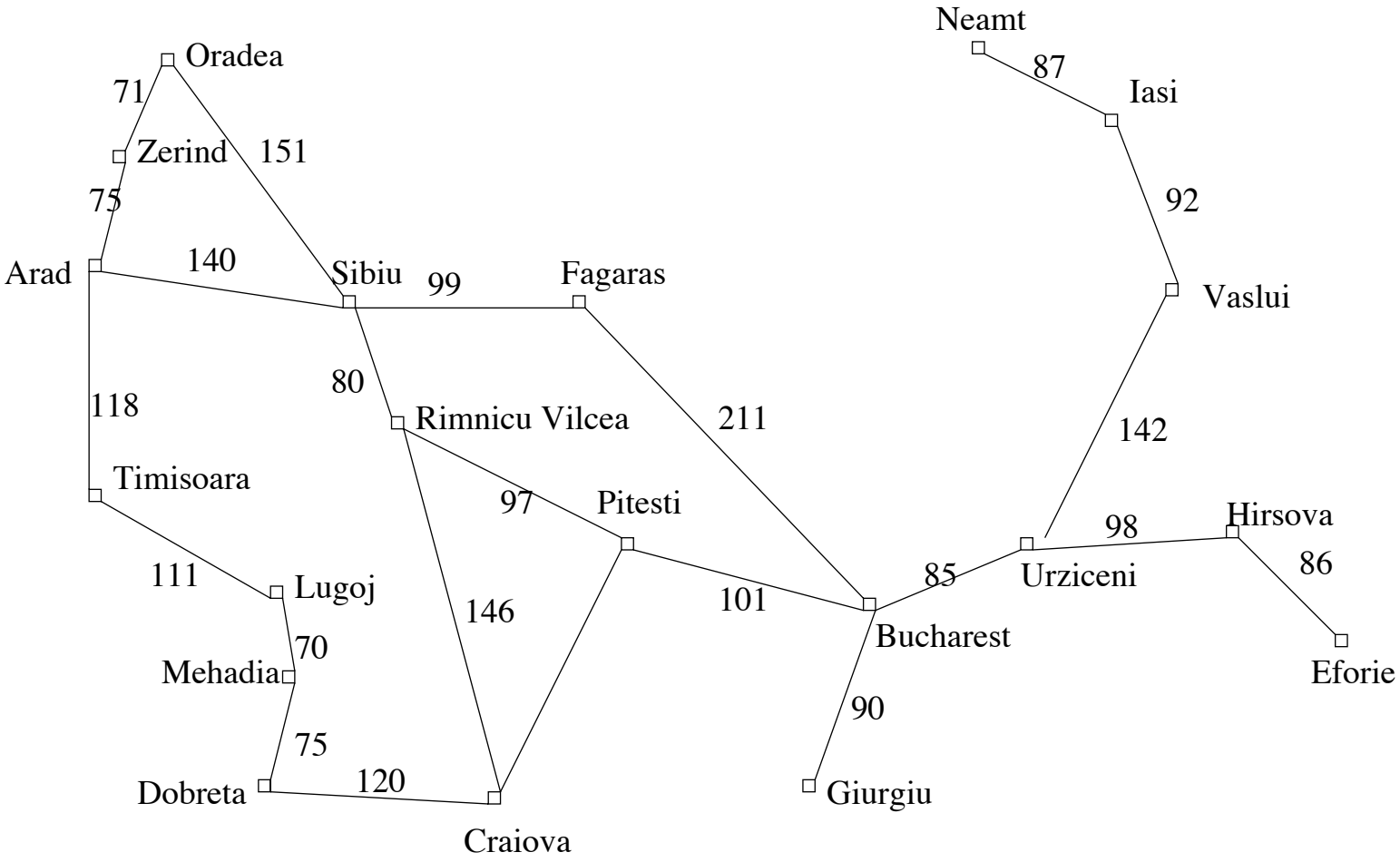
Funzione di valutazione : $\boxed{\text{f} = \text{h}}$

È espanso per primo il nodo ritenuto più vicino a un obiettivo

Si deve avere $h(n) = 0$ se lo stato in n è un obiettivo

La ricerca golosa minimizza il costo stimato per raggiungere l'obiettivo.

Esempio Visita Greedy: Ricerca Itinerario



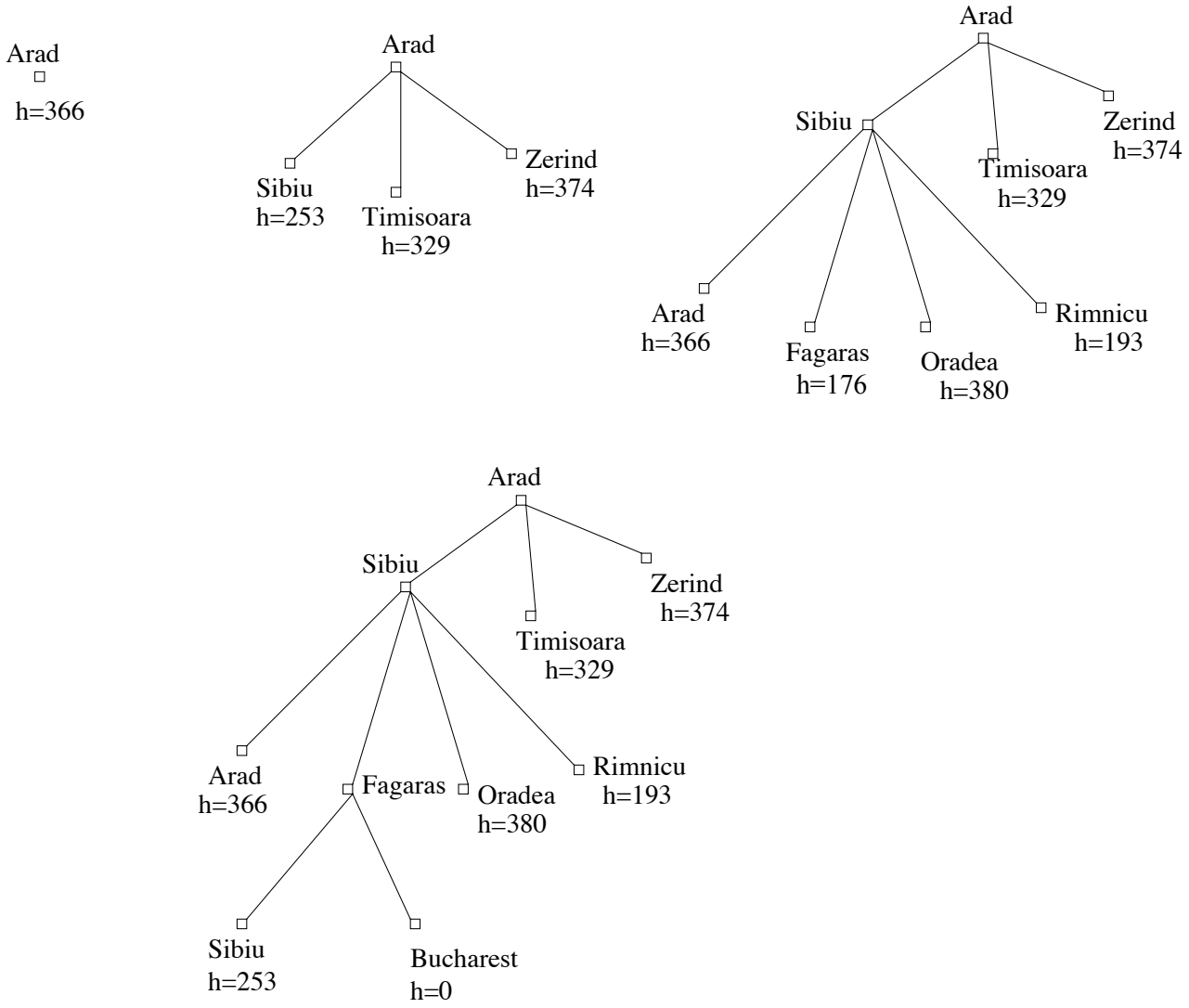
FUNZIONE EURISTICA PER L'ESEMPIO

$h_{SLD}(n)$ = distanza in linea d'aria (Straight Line Distance) dallo stato in n all'obiettivo

SLD da Bucharest

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

La ricerca golosa con $h = h_{SLD}$



La ricerca golosa con $h = h_{SLD}$

Soluzione

Soluzione trovata: Arad - Sibiu - Fagaras - Bucharest
Lunghezza del percorso: 450 Km

Non è una soluzione ottima: Arad - Sibiu - Rimnicu - Pitesti - Bucharest: 418 Km

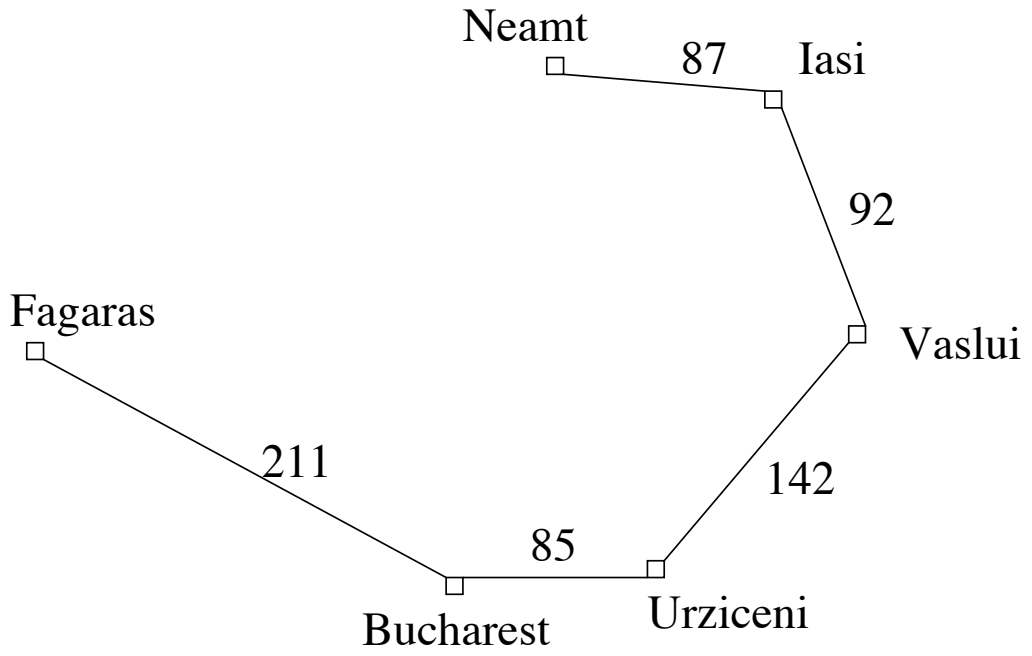
La ricerca golosa non è ottimale

Problema delle false partenze

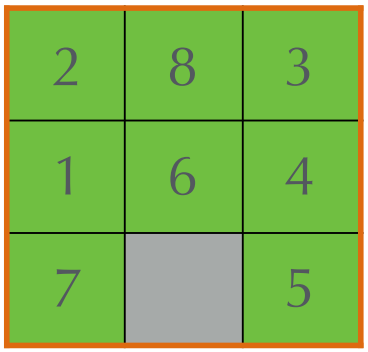
La ricerca golosa non è completa

Esempio: Ricerca di un percorso da Iasi a Fagaras.

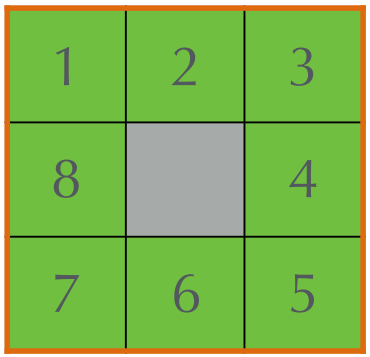
Per raggiungere l'obiettivo ci si deve prima allontanare:



Esempio Visita Greedy: Il Rompicapo a 8 Tasselli



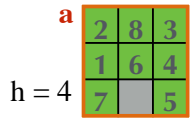
Stato Iniziale



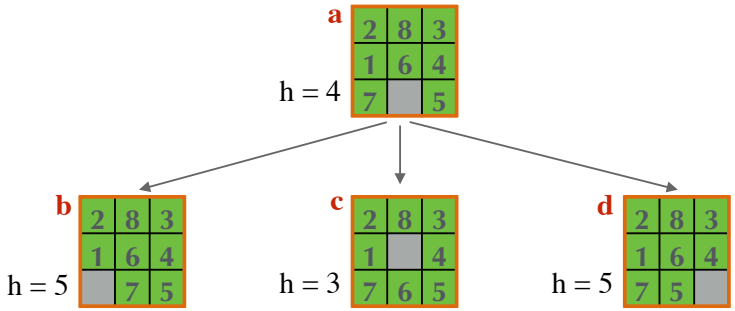
Stato Obiettivo

Funzione euristica **h**: numero di tasselli fuori posto.

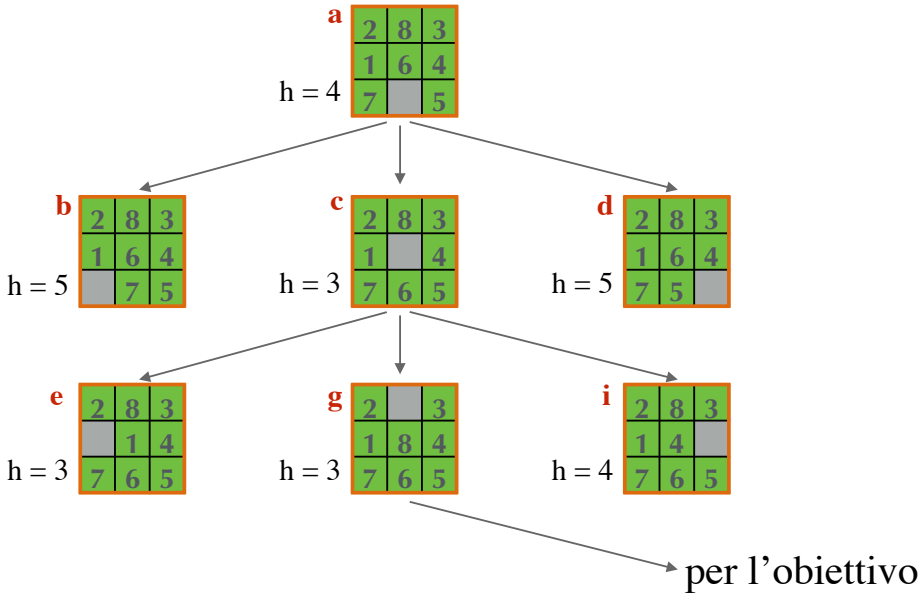
Visita Greedy: Esempio gioco 8



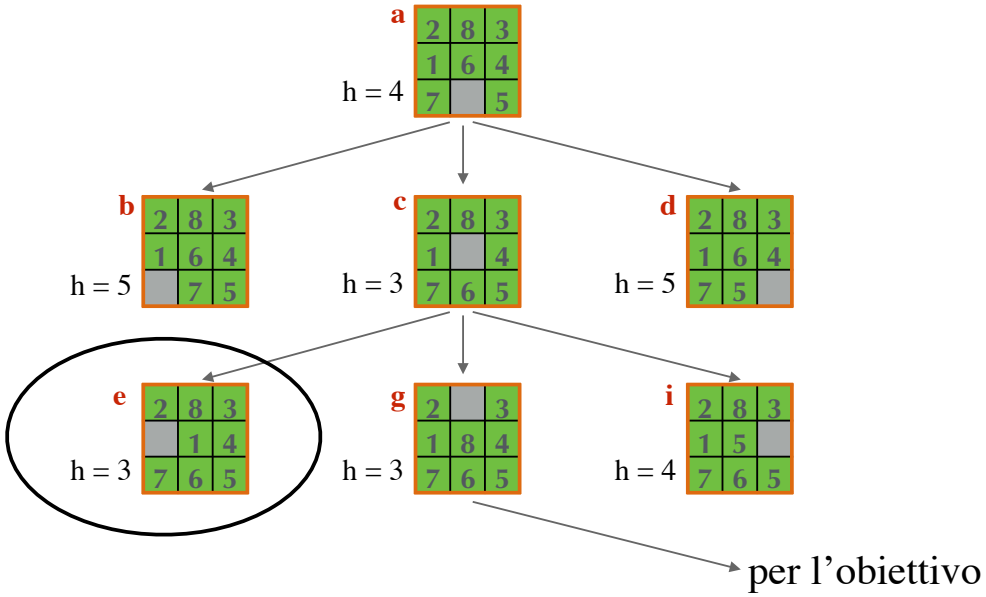
Visita Greedy: Esempio gioco 8



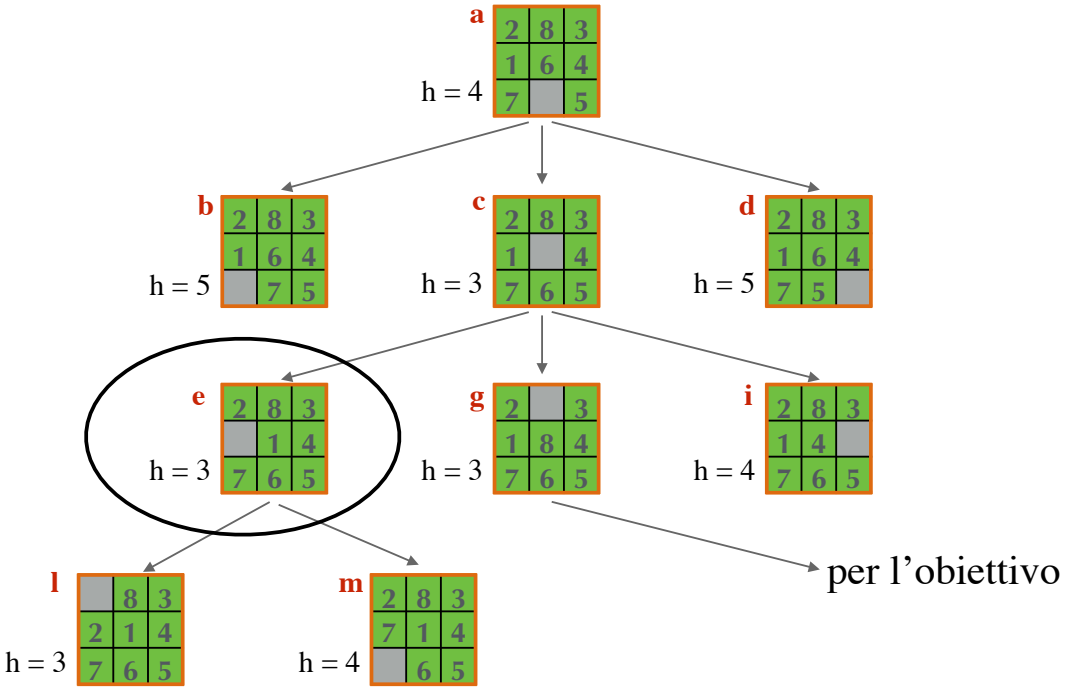
Visita Greedy: Esempio gioco 8



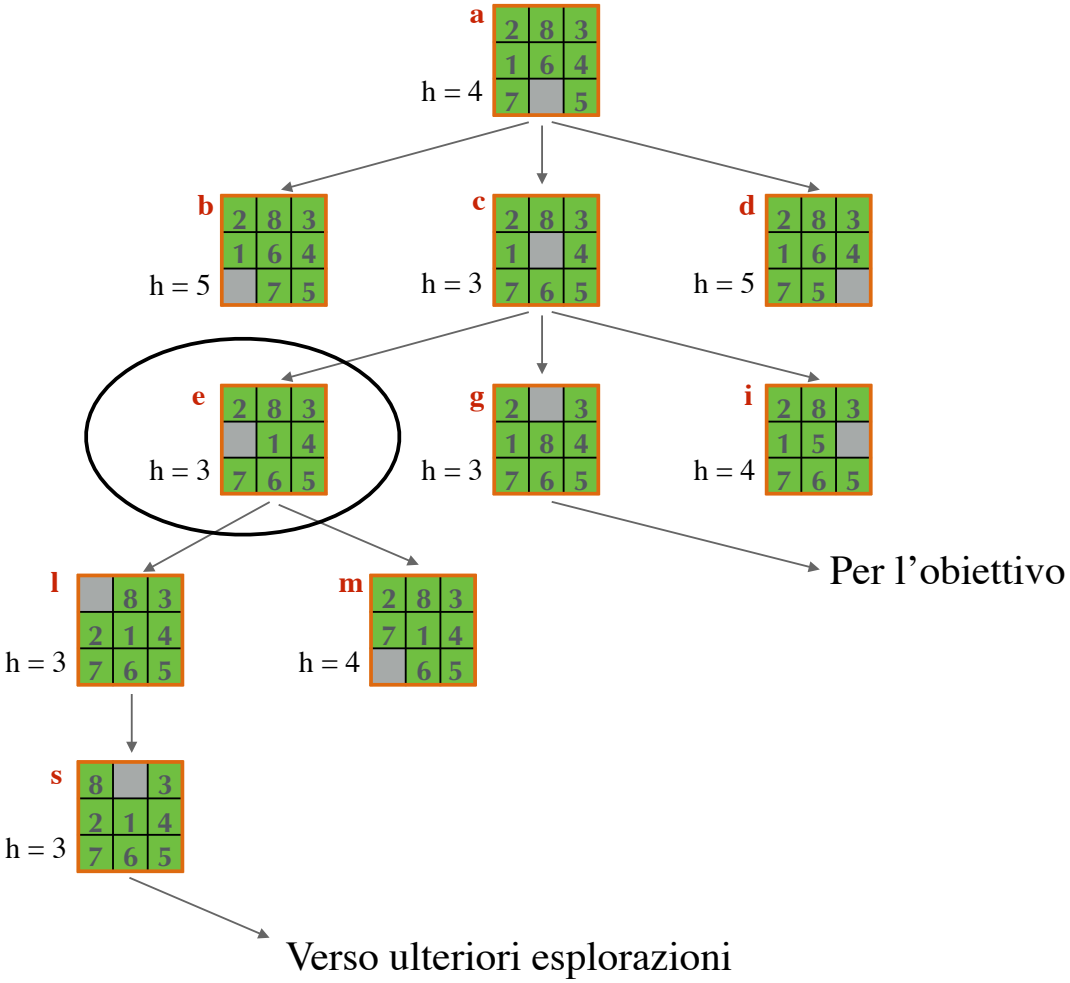
Visita Greedy: Esempio gioco 8



Visita Greedy: Esempio gioco 8



Visita Greedy: Esempio gioco 8



Greedy Search

Non è completa

Potrebbe seguire un cammino infinito senza trovare mai la soluzione (come la ricerca in profondità).

Non è ottimale

(come la ricerca in profondità).

Complessità in tempo $O(b^m)$ (caso peggiore)

m = profondità massima dell'albero di ricerca
b = fattore di ramificazione dell'albero di ricerca

Complessità in spazio $O(b^m)$

perchè deve tenere in memoria tutti i nodi.

Nonostante le apparenze con una buona euristica possiamo avere ottimi risultati

L'algoritmo A*

Ricerca Greedy: minimizza il costo stimato ma non è né ottimale né completa.

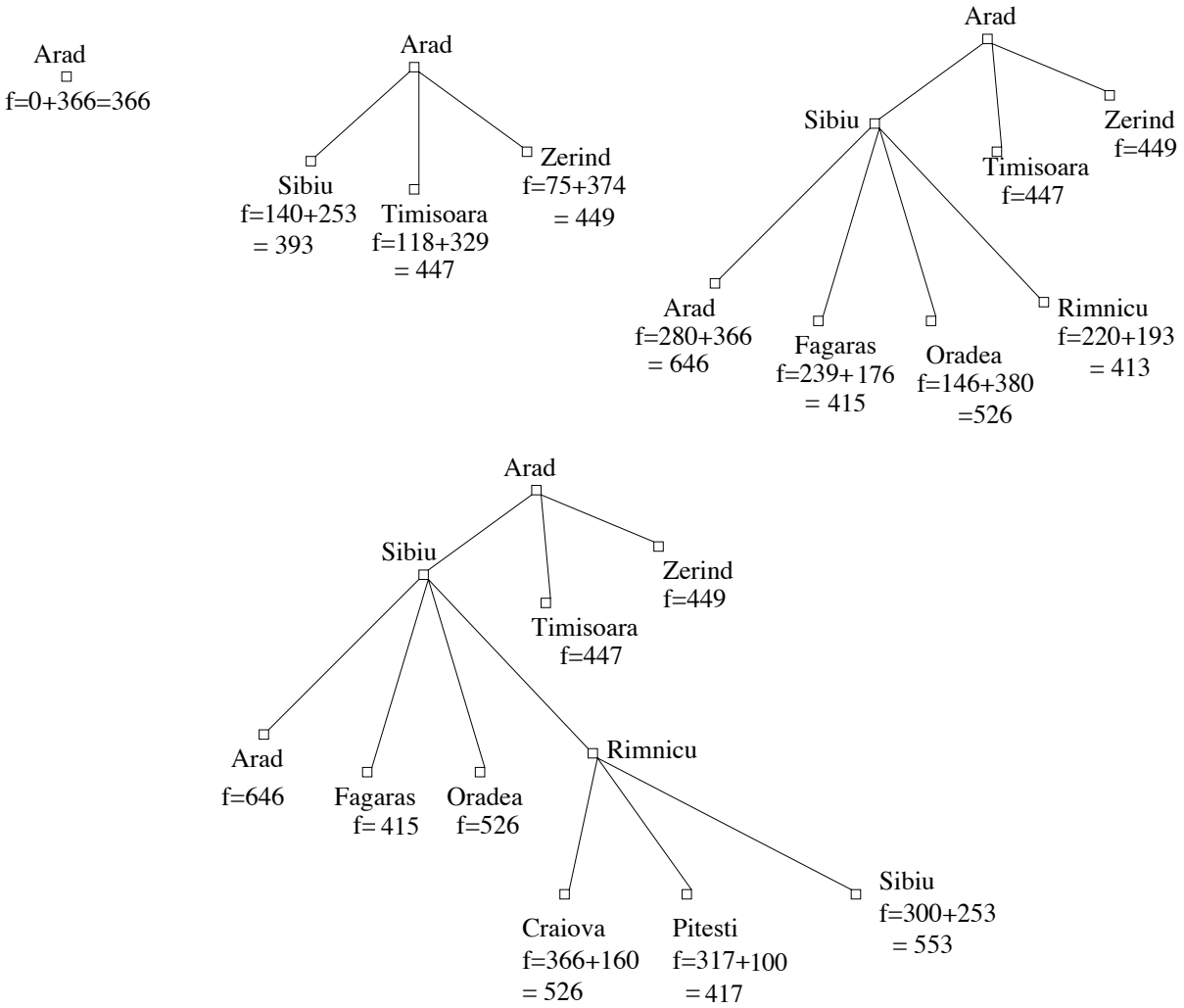
Ricerca guidata dal costo: ottimale e completa ma a volte inefficiente.

Algoritmo A*:

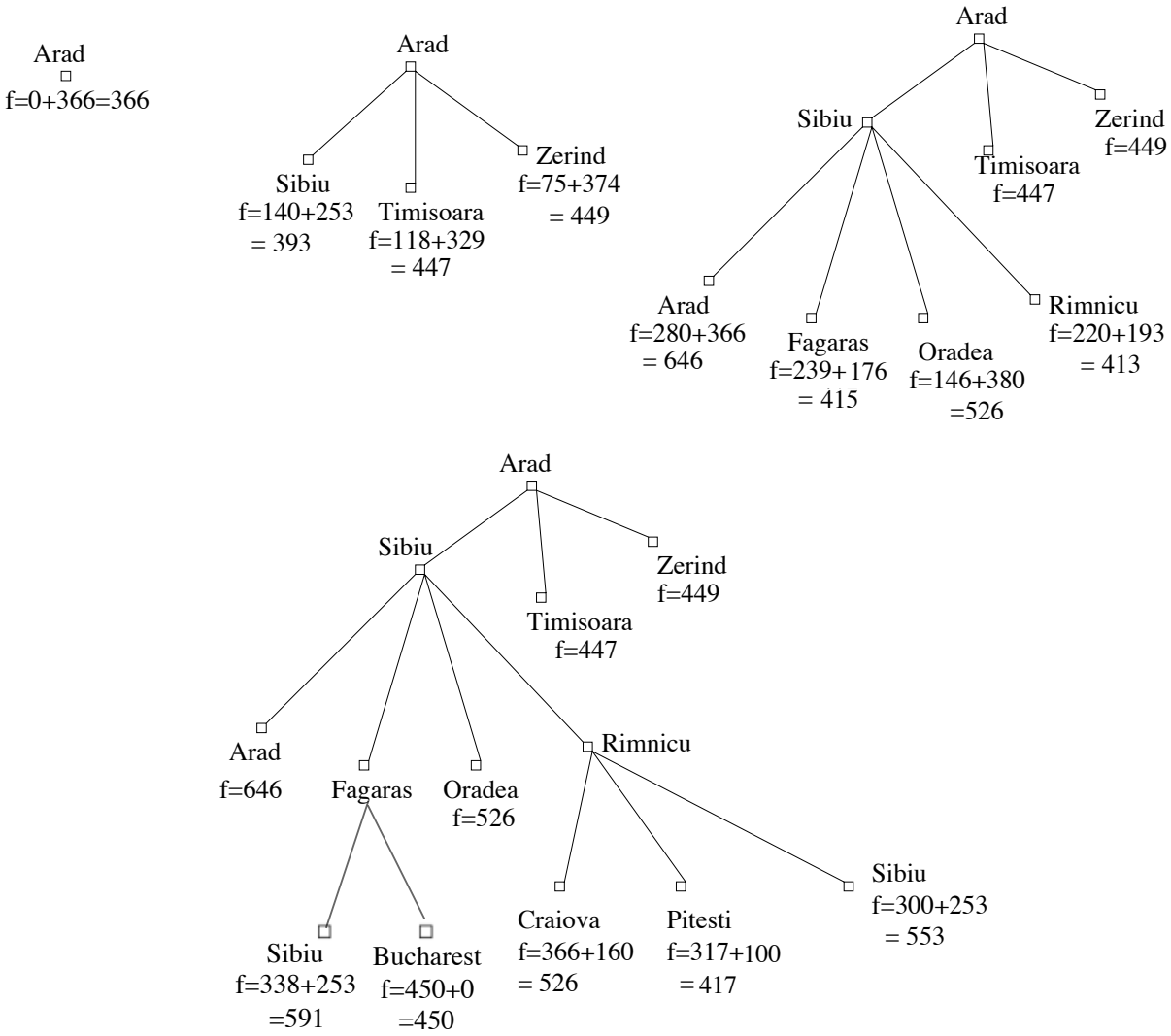
Funzione di valutazione : $f(n) = g(n) + h(n)$

stima del costo del cammino a costo minimo dallo stato iniziale a un obiettivo, vincolato a passare per lo stato di n

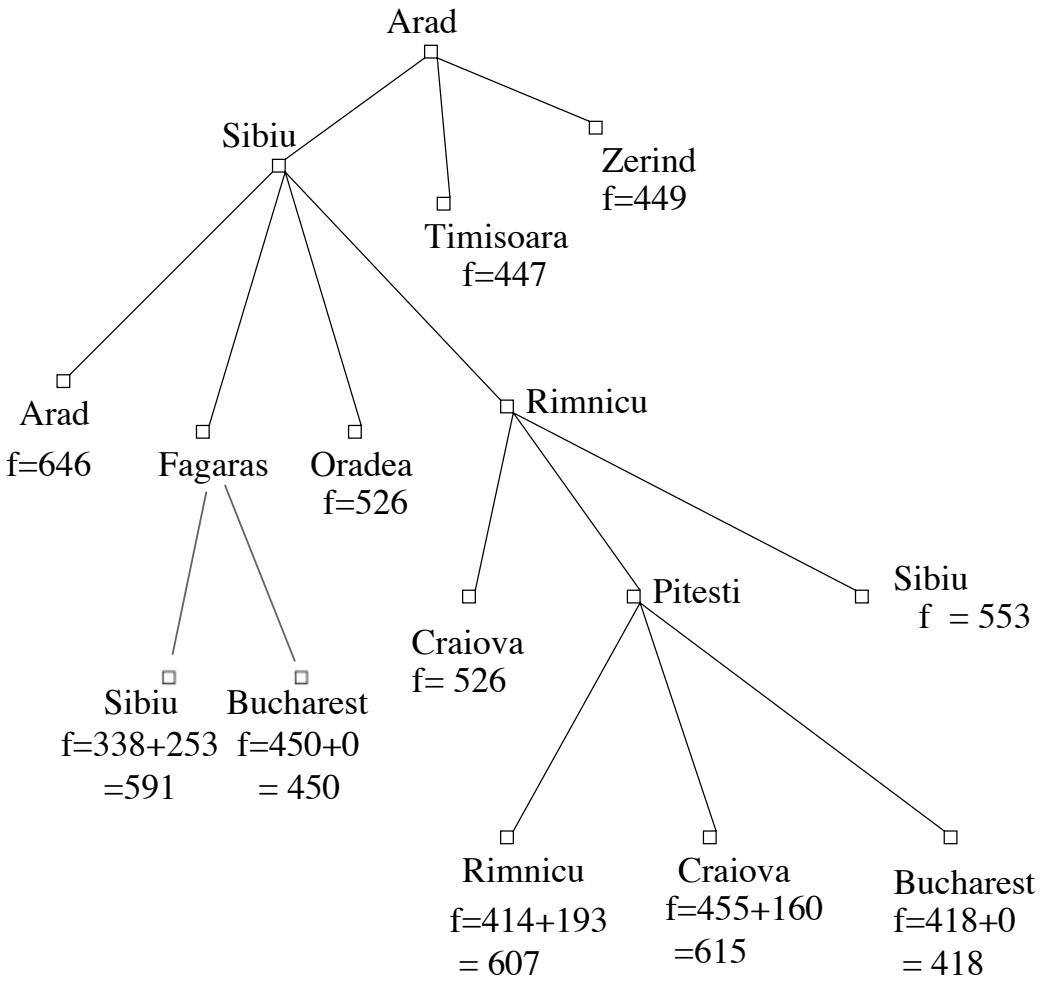
Visita A*: esempio Itinerario



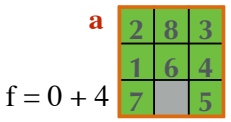
Visita A*: esempio Itinerario



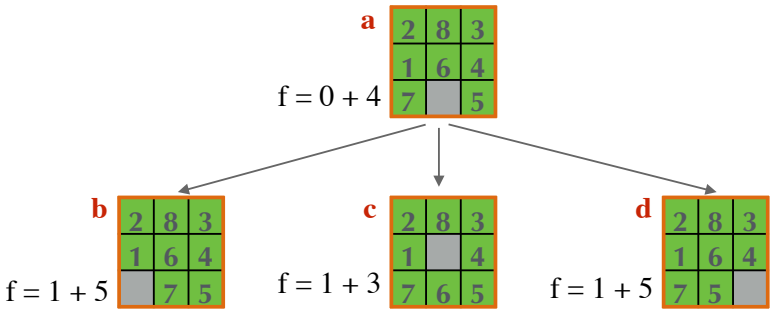
Visita A*: esempio Itinerario



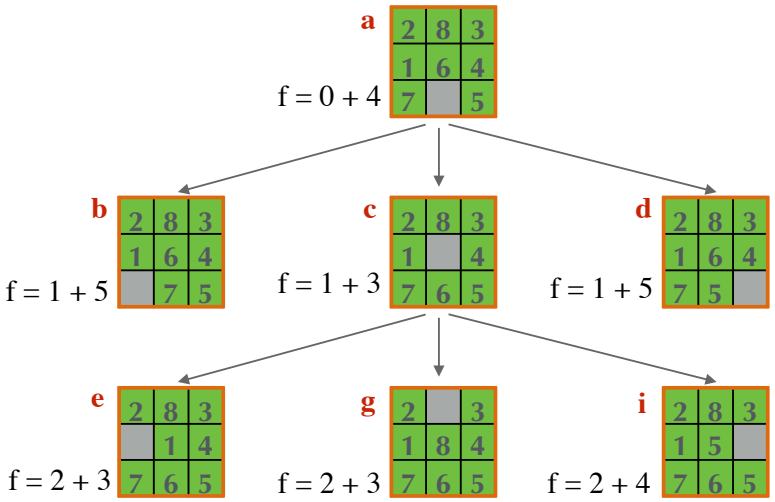
Visita A*: Esempio gioco 8



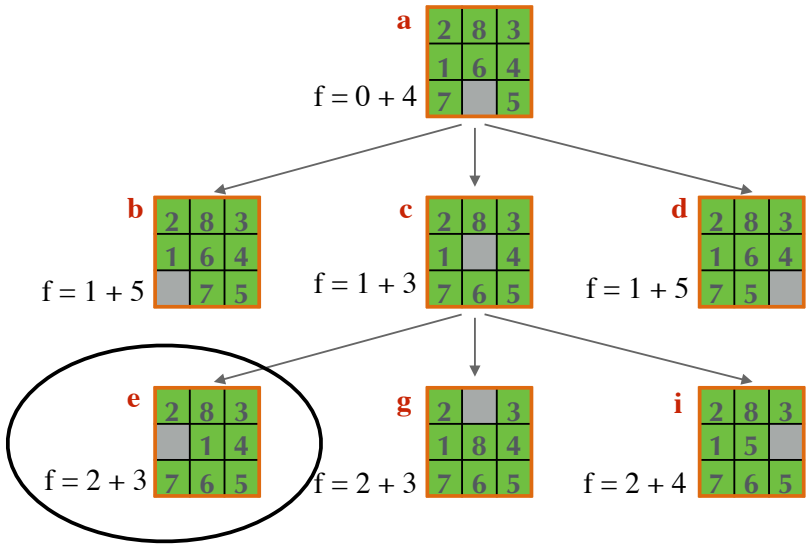
Visita A*: Esempio gioco 8



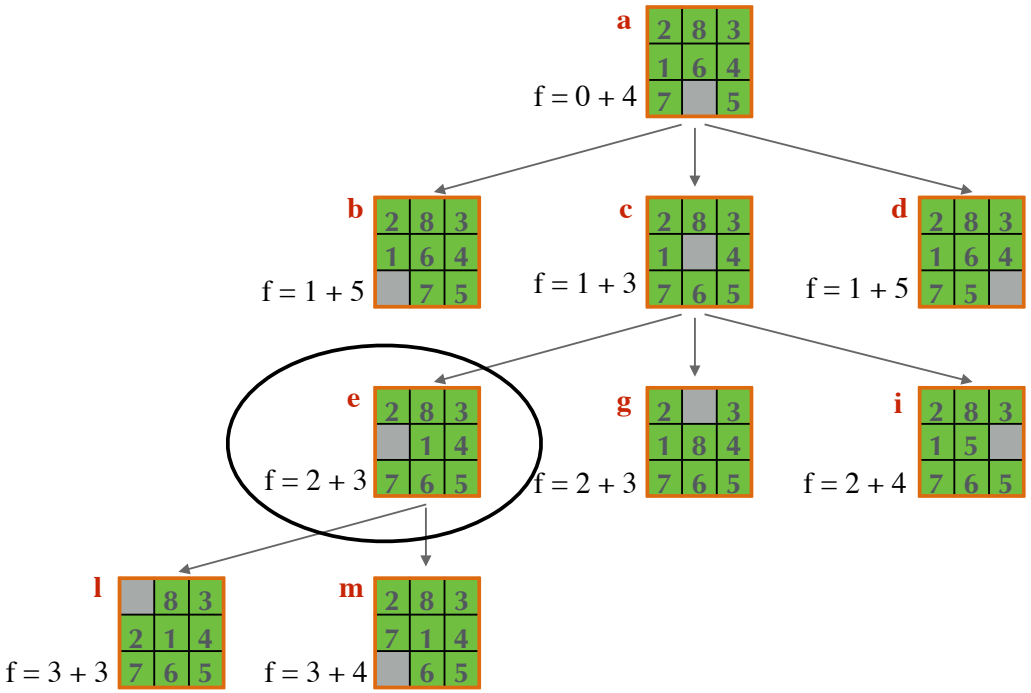
Visita A*: Esempio gioco 8



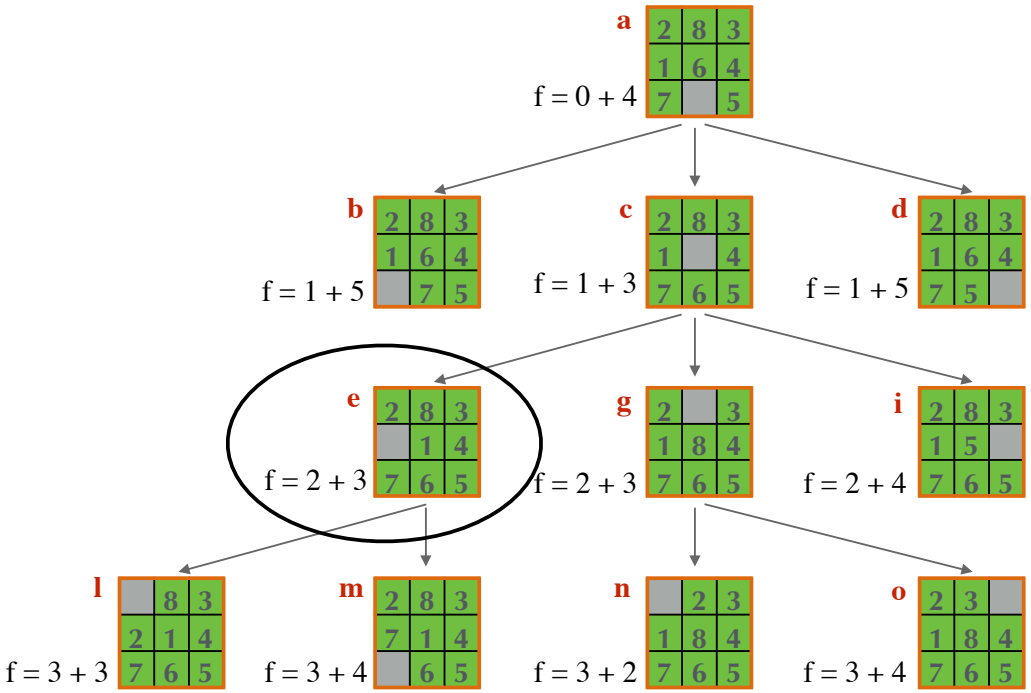
Visita A*: Esempio gioco 8



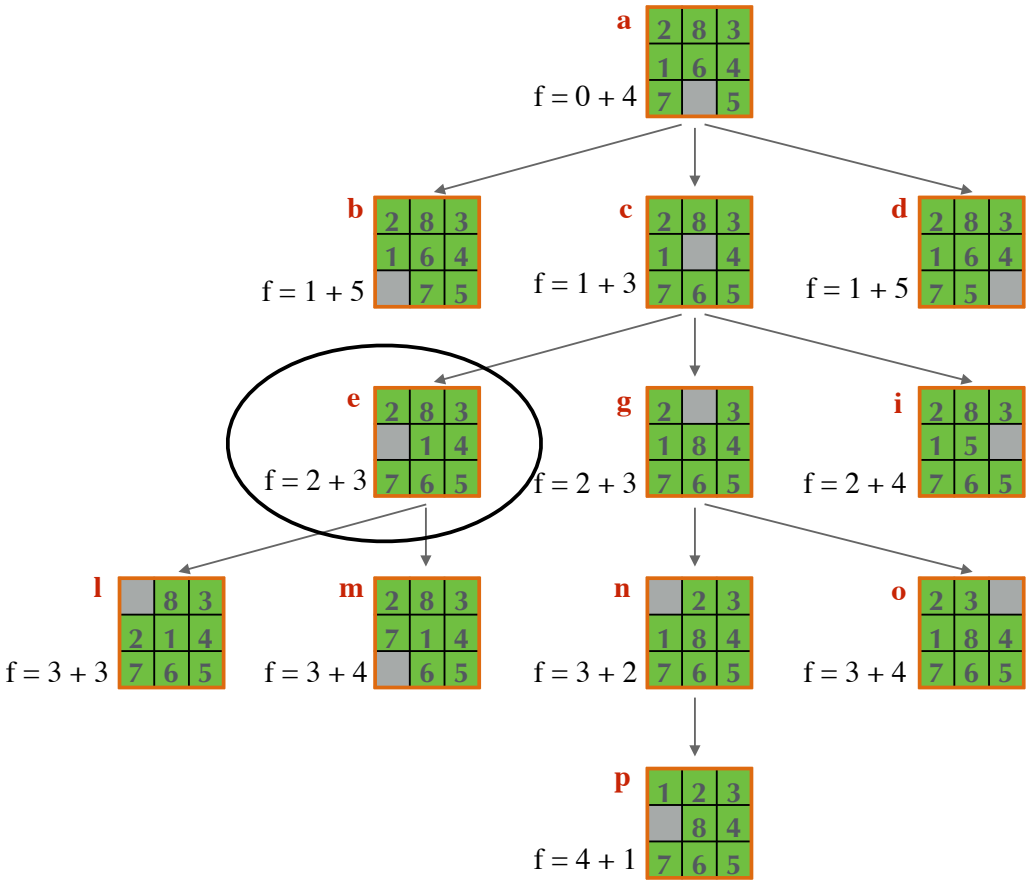
Visita A*: Esempio gioco 8



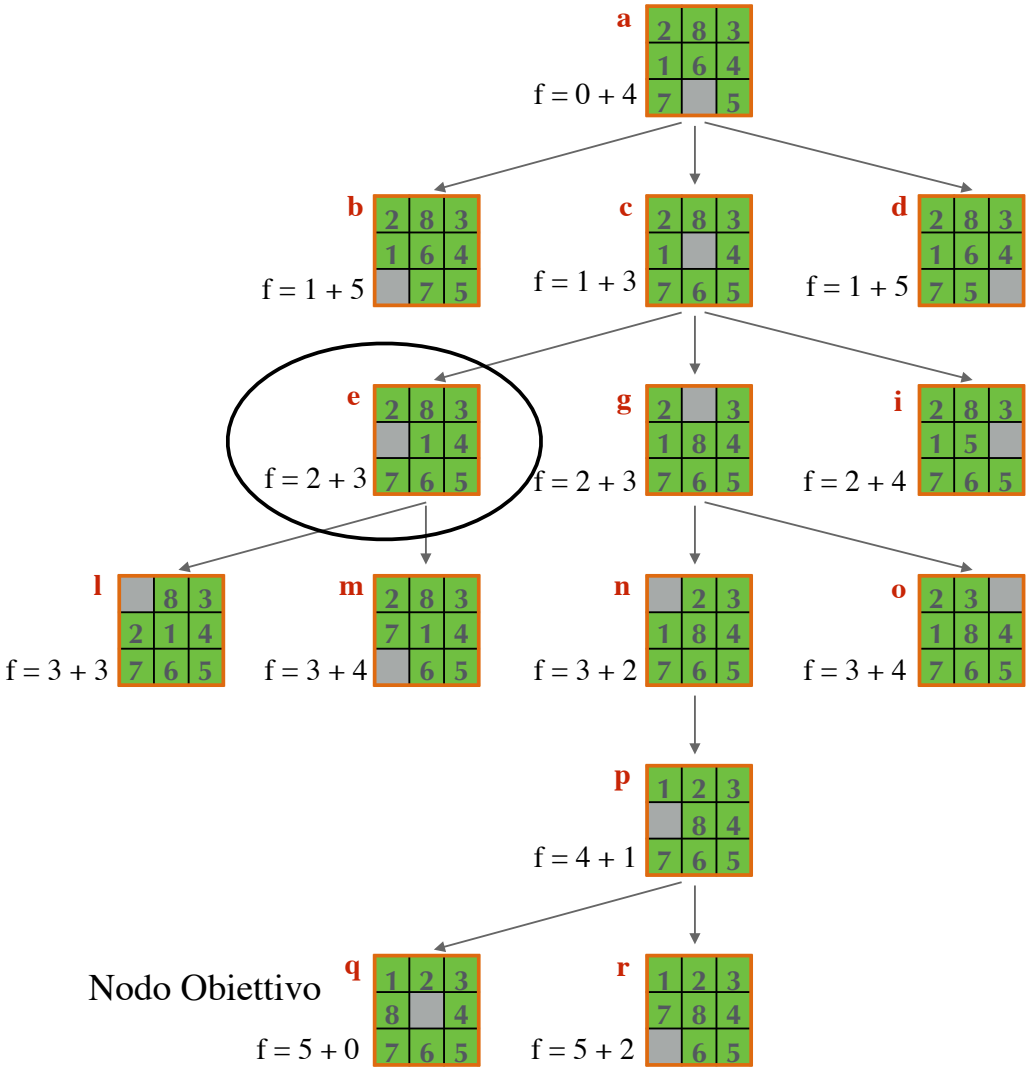
Visita A*: Esempio gioco 8



Visita A*: Esempio gioco 8



Visita A*: Esempio gioco 8



Teorema di Completezza e ottimalità per A*

Supponiamo che:

- ogni nodo del grafo abbia un numero finito di successori.
- tutti gli archi abbiano costi maggiori di una quantità positiva δ.

Ricordando che $h^*(n)$ è il costo effettivo del cammino a costo minimo dallo stato di n a uno stato obiettivo:

Se h è un'euristica ammissibile, cioè se

$$\forall n \quad h(n) \leq h^*(n)$$

(h "ottimista")

allora A* è completo e ottimale

Esempi di Stimatore Ottimistico

In genere non è difficile trovare una funzione h che soddisfi la condizione di limite inferiore citata prima:

- Nei problemi riguardanti la ricerca di itinerari su di un grafo i cui stati rappresentano delle città, possiamo scegliere come funzione h la distanza in linea retta tra una città e una città obiettivo.
- Nel puzzle dell'otto possiamo scegliere per h il numero di tasselli fuori posto, numero che sicuramente costituisce un limite inferiore per il numero di passi necessari per raggiungere l'obiettivo.
- Sempre nel puzzle dell'otto possiamo scegliere per h la Distanza di Manhattan (anch'essa sicuramente ammissibile).
- La ricerca guidata dal costo [f(n) = g(n)] è un caso particolare con h tale che:

$$\forall n \quad h(n) = 0$$

Teorema di Completezza e Ottimalità per A*

Nell'ipotesi che siano soddisfatte le seguenti condizioni sui grafi e su h:

- Ogni nodo del grafo ha un numero finito di successori (se ce ne sono)
- Tutti gli archi nel grafo hanno costi più grandi di una quantità positiva ∂
- Per tutti i nodi n si ha: $h(n) \leq h^*(n)$, ossia h non sovrastima mai il valore reale h^* (una tale funzione è chiamata **euristica ammissibile** o stimatore ottimistico)

e che vi sia un percorso a costo finito dal nodo di partenza, n_0, a un nodo obiettivo, l'algoritmo A* termina sempre con un percorso a costo minimo per il nodo obiettivo (ossia A* è completo e ottimale).

Dimostrazione del Teorema di Completezza e Ottimalità per A*

Ai fini della dimostrazione, procediamo come segue:

A. Enunciamo e dimostriamo il lemma che segue.

B. Dimostriamo il teorema di completezza e ottimalità per A* avvalendoci del lemma suddetto.

Lemma per A*

Ad ogni passo prima della terminazione di A* vi è sempre un nodo, che chiamiamo n*, appartenente alla frontiera (lista APERTA), con le seguenti proprietà:

1. n* è su un percorso ottimo per un nodo obiettivo.
2. A* ha trovato un percorso ottimo fino a n*.
3. $f(n^*) \leq f^*(n_0)$

Prova del Lemma (1)

Dimostriamo il lemma per **induzione matematica**.

Per provare che le conclusioni del lemma valgono ad ogni passo di A* è quindi sufficiente provare che:

1. Esse valgono all'inizio dell'algoritmo (**caso base**).
2. Che se esse valgono al passo m-esimo (prima dell'espansione di un nodo), continueranno a valere al passo (m+1)-esimo (dopo l'espansione del nodo) (**passo induttivo**).

Prova del Lemma (2)

CASO BASE

All'inizio della ricerca, ossia quando proprio il nodo di partenza n_0 è stato selezionato per l'espansione (n_0 è nella lista APERTA), tale nodo è sicuramente su un percorso ottimo per l'obiettivo. A* ha quindi trovato tale percorso ottimo (fino ad n_0).

Anche $f(n^*) \leq f^*(n_0)$ perché $f(n_0) = h(n_0) \leq h^*(n_0) = f^*(n_0)$. Quindi il nodo n_0 può essere il nodo n^* del lemma a questo stadio.

Prova del Lemma (3)

PASSO INDUTTIVO

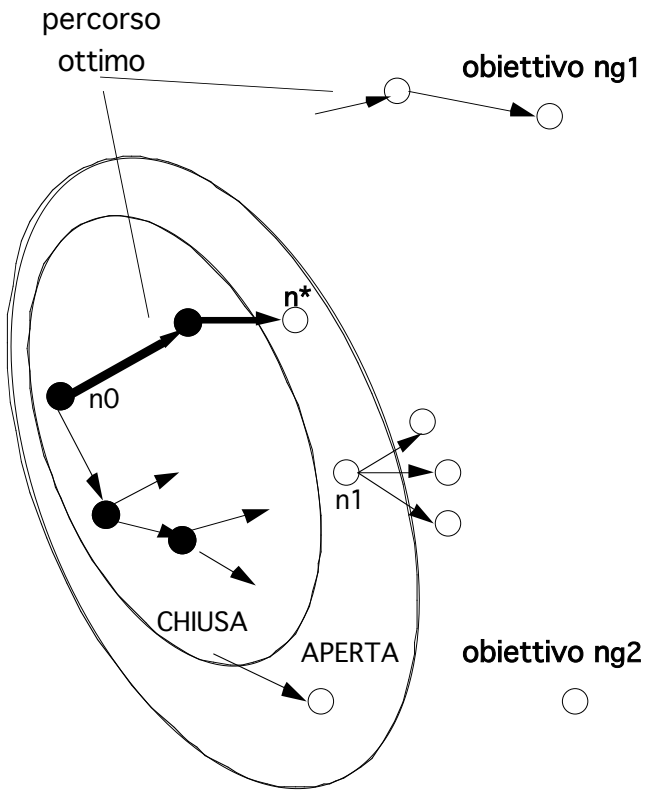
Ipotizziamo la validità delle conclusioni del lemma al passo m-esimo, ossia quando sono stati espansi m nodi (con $m \geq 0$).

Usando queste ipotesi dimostriamo che esse sono ancora vere al passo (m+1)-esimo, ossia quando sono stati espansi m+1 nodi.

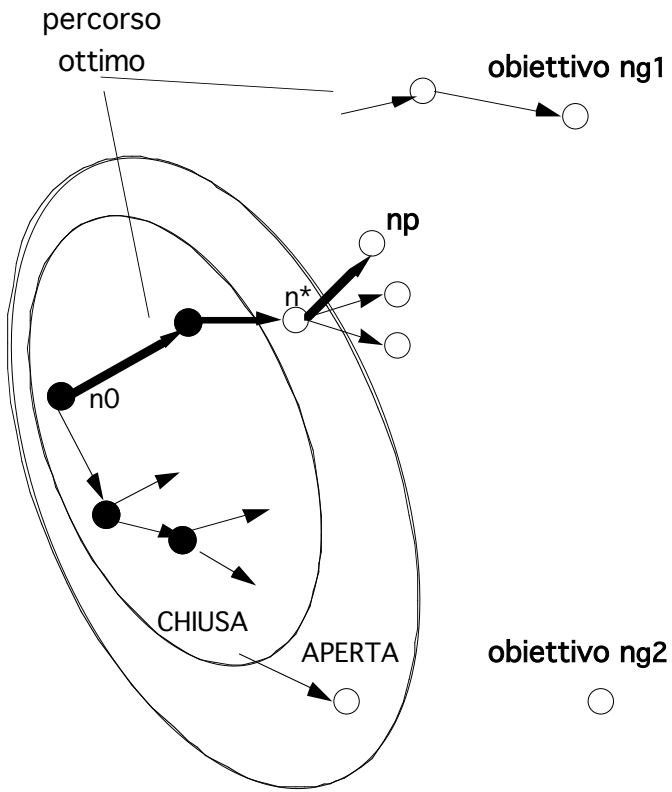
Facciamo riferimento alla figura seguente, in cui è rappresentata la situazione quando il nodo (m+1)-esimo è selezionato per l'espansione.

Prova del Lemma (4)

passo induttivo



(a) A* seleziona n1 come nodo (m+1)-esimo da espandere



(b) A* seleziona n* come nodo (m+1)-esimo da espandere

Prova del Lemma (5)

passo induttivo

- Sia n* il nodo ipotizzato nella lista APERTA, cioè quello che è su un percorso ottimo trovato da A* dopo che ha espanso m nodi.

- Se n* non è selezionato per l'espansione al passo (m+1)-esimo (caso a in figura), n* ha le stesse proprietà di prima, per cui il passo induttivo è dimostrato in questo caso.

- Se n* è selezionato per l'espansione al passo (m+1)-esimo (caso b in figura), tutti i suoi successori saranno inseriti in APERTA, e almeno uno di essi, chiamiamolo n_P, sarà su un percorso ottimo per l'obiettivo. (Questo perché si è ipotizzato che un percorso ottimo passi per n* e debba svilupparsi attraverso uno dei suoi successori.)

Prova del Lemma (6)

- L'algoritmo A* ha quindi trovato un percorso ottimo fino ad n_P poiché, se ci fosse un percorso migliore fino a n_P, tale percorso sarebbe anche un percorso migliore per un nodo obiettivo (contraddicendo la nostra ipotesi secondo la quale non c'è un percorso migliore per l'obiettivo di quello che A* ha trovato attraverso n^*). Il nodo n_P diventa dunque il nuovo n^*.

- Per completare la dimostrazione dobbiamo ancora dimostrare la terza proprietà del lemma: $f(n^*) \leq f^*(n_0)$ applicata al nodo n_P, ossia dobbiamo provare che $f(n_P) \leq f^*(n_0)$.

Prova del Lemma (7)

Per ogni nodo n_P che si trova su un percorso ottimo, e per il quale A* ha già trovato un percorso ottimo, abbiamo:

$f(n_P) = g(n_P) + h(n_P)$ per definizione

$\leq g^*(n_P) + h^*(n_P)$ perché $g(n_P) = g^*(n_P)$ e $h(n_P) \leq h^*(n_P)$

$= f^*(n_P)$ perché $f^*(n_P) = g^*(n_P) + h^*(n_P)$ per def.

$= f^*(n_0)$ perché $f^*(n_P) = f^*(n_0)$ poiché n_P è su un percorso ottimo

[**q.e.d.**]

Prova del Teorema (1)

Dimostriamo che:

1. A* deve terminare se vi è un obiettivo accessibile.

2. A* termina trovando un percorso ottimo per un nodo obiettivo.

Prova del Teorema (2)

Punto 1:

A* deve terminare se vi è un obiettivo accessibile

Supponiamo che A* non termini. In questo caso A* continua ad espandere nodi in APERTA per sempre, ed espanderebbe alla fine i nodi sempre più in profondità nell'albero di ricerca, al di là di qualunque limite finito di profondità prefissato (visto che abbiamo ipotizzato che il grafo su cui effettuare la ricerca abbia un fattore di ramificazione finito).

Dal momento che il costo di ogni arco è maggiore di $\partial > 0$, i valori di g (e quindi di f) di tutti i nodi in APERTA supereranno alla fine il valore $f^*(n_0)$. Ma ciò contraddirebbe il lemma.

Prova del Teorema (3)

Punto 2:
A* termina trovando un percorso ottimo per un nodo obiettivo

L'algoritmo A* può terminare in due modi:

1. Se la lista APERTA è vuota
2. Se raggiunge un nodo obiettivo

Prova del Teorema (4)

Lista APERTA vuota:

Tale situazione può aver luogo solo per grafi finiti non contenenti nessun nodo obiettivo, e il teorema presume di trovare un percorso ottimo per un obiettivo solo se vi è un nodo obiettivo.

(Dunque A* termina trovando un nodo obiettivo)

Prova del Teorema (5)

A* termina trovando un nodo obiettivo:

- Supponiamo che A* termini trovando un nodo obiettivo non ottimo. Chiamiamo questo nodo:

$$n_{G2}$$

- Per questo nodo abbiamo:

$$f(n_{G2}) > f^*(n_0)$$

- Per un obiettivo ottimo n_{G1} ovviamente si ha:

$$f(n_{G1}) = f^*(n_0)$$

Prova del Teorema (6)

- Al momento della terminazione in n_{G2} si ha:
$$f(n_{G2}) > f^*(n_0)$$

- Ma proprio prima della selezione da parte di A* di n_{G2} per l'espansione, per il lemma vi era un nodo n* in APERTA e su un percorso ottimo con
$$f(n^*) \leq f^*(n_0)$$

- Quindi A* non può aver selezionato n_{G2} per l'espansione, perché A* seleziona sempre quel nodo che ha il valore di f più piccolo, e
$$f(n^*) \leq f^*(n_0) < f(n_{G2})$$

Ossia c'è almeno un nodo in frontiera con la f minore di quella di n_{G2}. A* pertanto termina necessariamente su un nodo obiettivo ottimo. [**q.e.d.**]

Influenza di h sulla efficienza

Se due versioni di A*, che possiamo chiamare A_1* e A_2*, differiscono soltanto per il fatto che $h_1 < h_2$ per tutti i nodi non obiettivo, si dice che A_2* è *più informato* di A_1*.

In questo caso vale il seguente teorema:

> Se A_2* è più informato di A_1* allora, al termine delle loro ricerche su qualsiasi grafo che abbia un percorso da n_0 ad un nodo obiettivo, ogni nodo espanso da A_2* sarà stato espanso anche da A_1*.

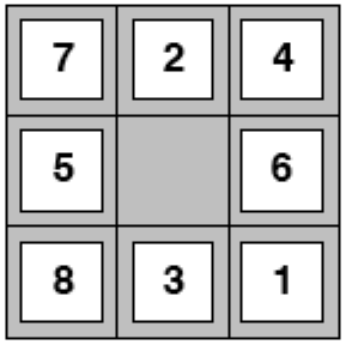
Da questo teorema segue che A_1* espande almeno tanti nodi quanti ne espande A_2*. Quindi l'algoritmo più informato A_2* è normalmente più efficiente.

Due euristiche per il gioco dell'otto

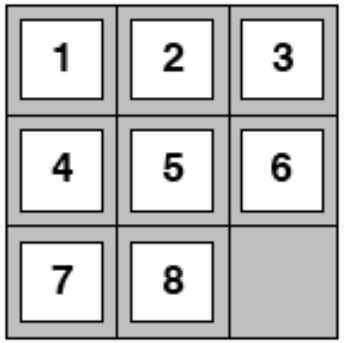
$h_1(n)$: numero di caselle fuori posto

$h_2(n)$: somma delle distanze di ogni casella dalla propria posizione finale

Esempio:



Stato iniziale



Stato obiettivo

$h_1(S) = 6$

$h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$

Due euristiche per il gioco dell'otto

Per ogni n:

$$h_1(n) \leq h_2(n) \leq h^*(n)$$

h_2 **è più informata di** h_1

A* espande ogni nodo n con $f(n) < f^*(s_0)$, cioè ogni nodo n con

$$h(n) < f^*(s_0) - g(n)$$

Quindi ogni nodo espanso da A^* con h_2 è espanso anche con h_1

Due euristiche per il gioco dell'otto

- Per avere un'idea di una buona euristica - confronto tra il costo della ricerca con Iterative Deepening Search (**IDS**) e **A*** con le due euristiche:

d	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73

d	IDS	$A^*(h_1)$	$A^*(h_2)$
14	3473941	539	113
16	−−	1301	211
18	−−	3056	363
20	−−	7276	676
22	−−	18094	1219
24	−−	39135	1641

Numero medio di nodi espansi, su 100 problemi

d = profondità della soluzione

Come inventare funzioni euristiche

h_1 è la h^* per il gioco dell'otto in cui ogni casella può saltare da una posizione a un'altra qualsiasi

h_2 è la h^* per il gioco dell'otto in cui le caselle possono spostarsi anche su caselle occupate

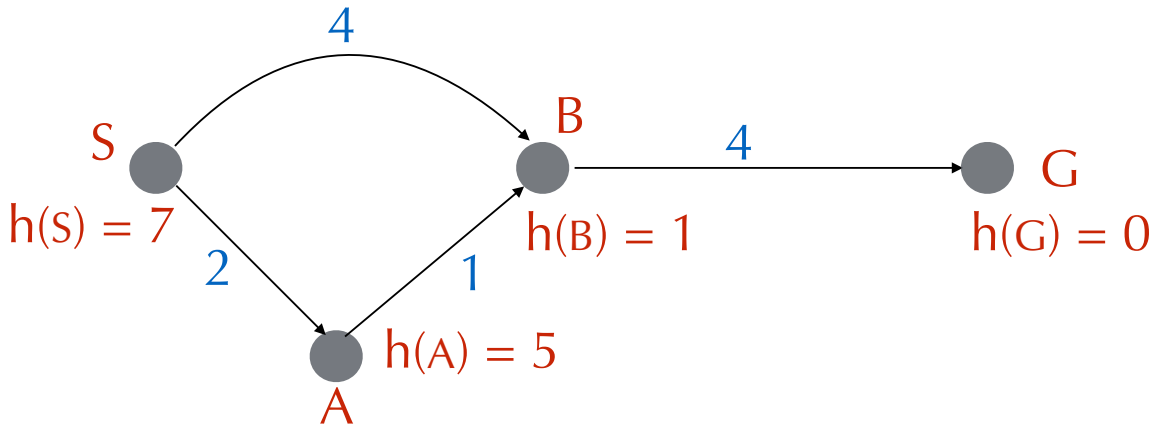
Regola del gioco: una casella si può spostare da A a B se:
A è adiacente a B, e
B è vuota
Relaxed problem : con minori restrizioni sugli operatori

Spesso, il costo effettivo di una soluzione in un relaxed problem è una buona euristica per il problema originario.

Attenzione al costo computazionale della funzione euristica

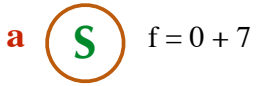
Esercizio per A*

Dato il seguente spazio degli stati, con i valori della funzione euristica **h** indicati in figura:

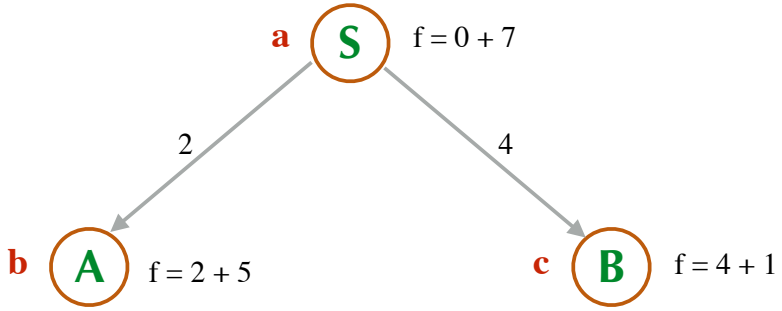


eseguire l'algoritmo **A*** sia in modalità **TREE-SEARCH** sia in modalità **GRAPH-SEARCH** e riportare le soluzioni trovate.

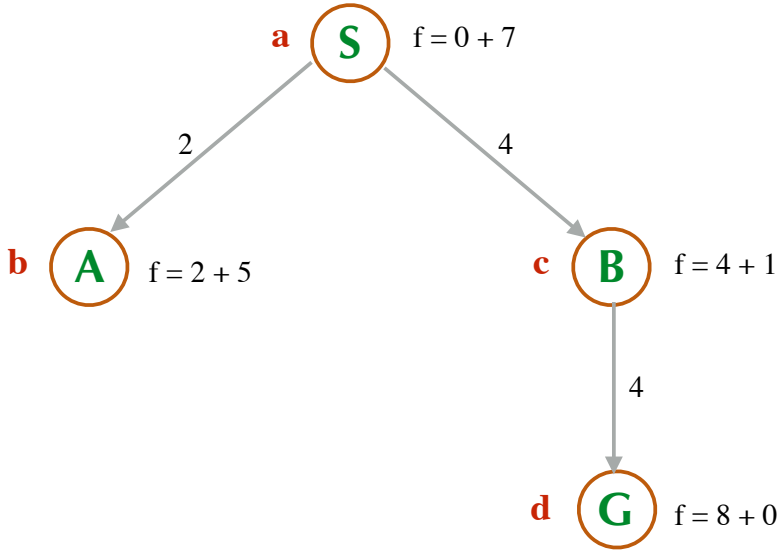
Esercizio per A*: Tree Search



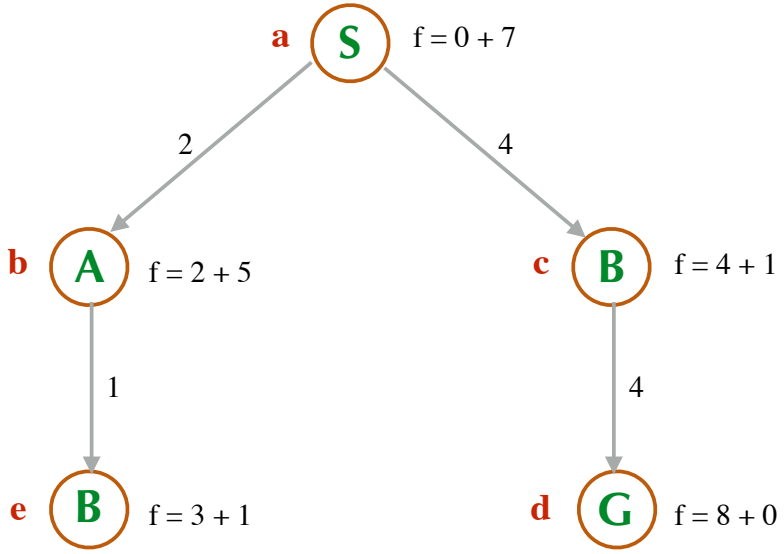
Esercizio per A*: Tree Search



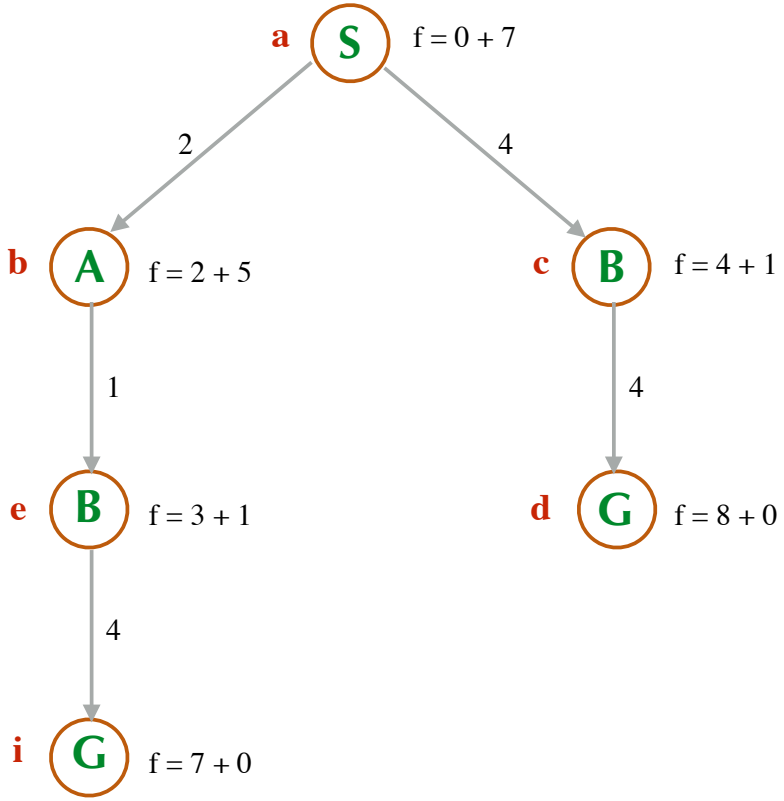
Esercizio per A*: Tree Search



Esercizio per A*: Tree Search



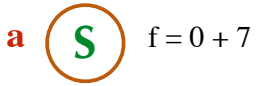
Esercizio per A*: Tree Search



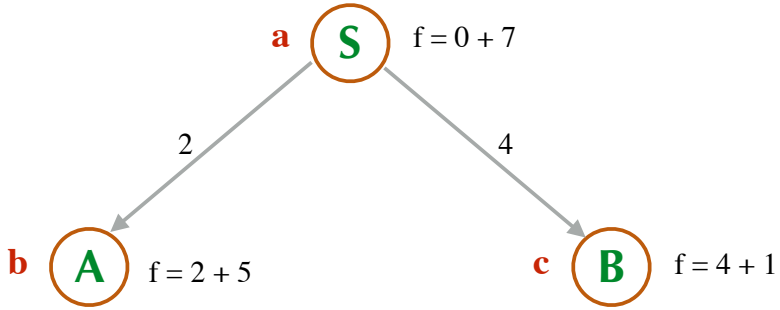
Soluzione: S-A-B-G

Costo: 7

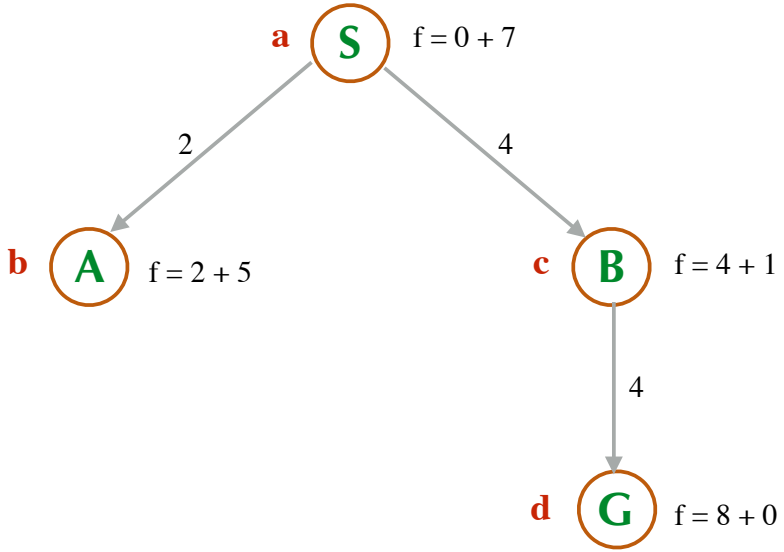
Esercizio per A*: Graph Search



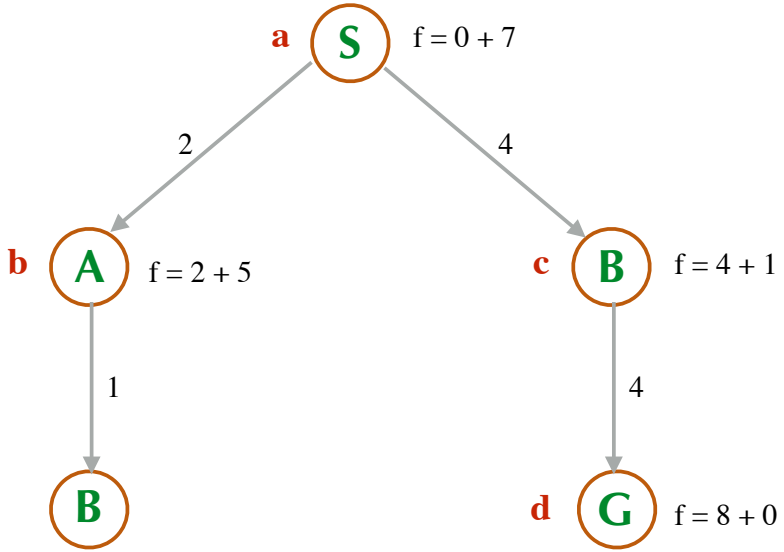
Esercizio per A*: Graph Search



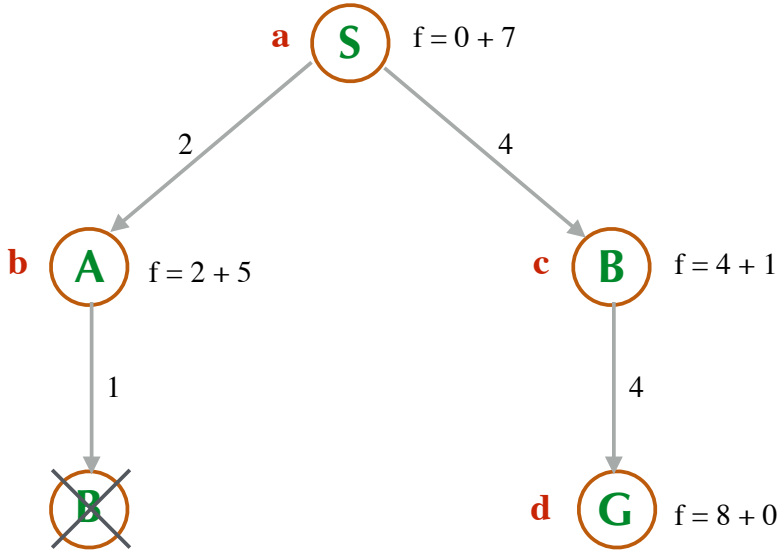
Esercizio per A*: Graph Search



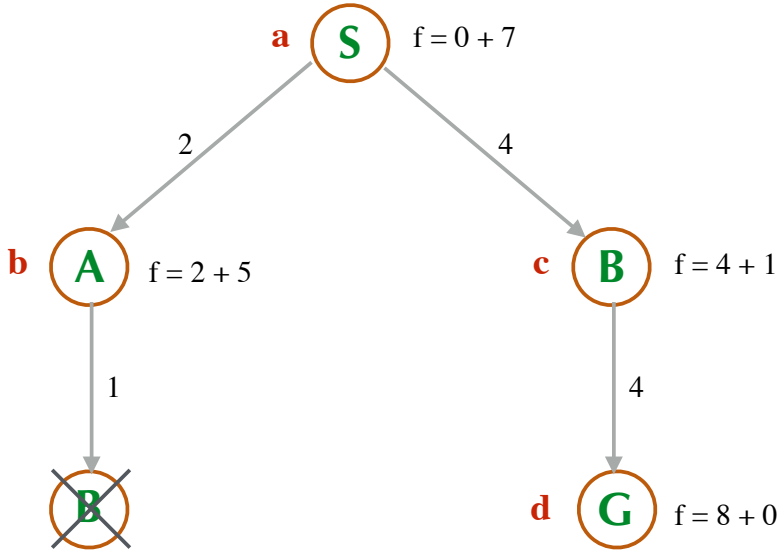
Esercizio per A*: Graph Search



Esercizio per A*: Graph Search



Esercizio per A*: Graph Search



Soluzione: S-B-G

Costo: 8

La Condizione di Consistenza

Consideriamo una coppia di nodi in cui n_j è un successore di n_i

Diciamo che ***h*** obbedisce alla condizione di consistenza se per tutte le coppie di nodi di questo tipo nel grafo di ricerca:

$$h(n_i) \leq c(n_i, n_j) + h(n_j)$$

dove $c(n_i, n_j)$ è il costo dell'arco da n_i a n_j.

La Condizione di Consistenza

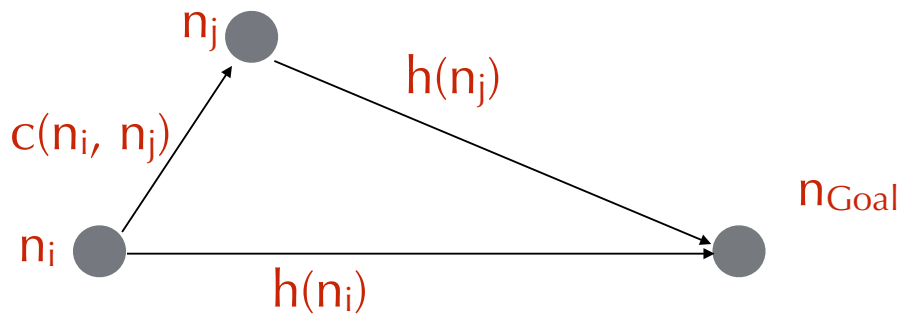
Potremmo anche scrivere:

$$h(n_j) \geq h(n_i) - c(n_i, n_j)$$

Questa condizione afferma che, lungo un qualsiasi percorso, la nostra stima del costo (del percorso rimanente) ottimo fino all'obiettivo non può diminuire di più del costo di un arco lungo quel percorso.

La Condizione di Consistenza
Diseguaglianza Triangolare

La condizione di consistenza può anche essere considerata come un tipo di diseguaglianza triangolare:



$$h(n_i) \leq c(n_i, n_j) + h(n_j)$$

La Condizione di Consistenza
Condizione Monotòna

La condizione di consistenza implica anche che i valori ***f*** dei nodi nell'albero di ricerca siano monotonicamente non-decrescenti man mano che ci allontaniamo dal nodo di partenza.

Dati n_i e n_j, con n_j successore di n_i

Se è soddisfatta la condizione di consistenza, abbiamo:

$$f(n_j) \geq f(n_i).$$

La Condizione di Consistenza
Condizione Monotòna

Per dimostrare questo fatto partiamo con la condizione di consistenza:

$$h(n_j) \geq h(n_i) - c(n_i, n_j)$$

Aggiungiamo $g(n_j)$ in entrambi i membri della diseguaglianza:

$$h(n_j) + g(n_j) \geq h(n_i) - c(n_i, n_j) + g(n_j)$$

Ma $g(n_j) = g(n_i) + c(n_i, n_j)$. Quindi:

$$f(n_j) \geq f(n_i)$$

Per questa ragione la condizione di consistenza (su h) è spesso chiamata condizione monotòna (su f).

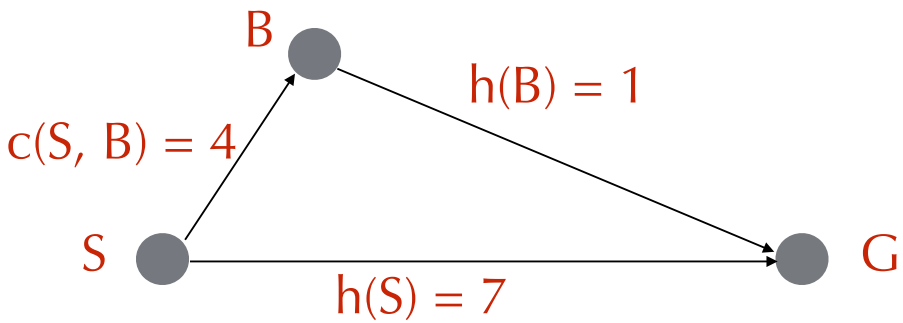
La Condizione di Consistenza
Teorema

Se la condizione di consistenza su h è soddisfatta, allora quando A* espande un nodo n, esso ha già trovato un percorso ottimo per n.

La condizione di consistenza è importante perché, quando essa è soddisfatta, la ricerca su grafo di A* non è differente dalla ricerca su un albero per quanto riguarda la ottimalità.

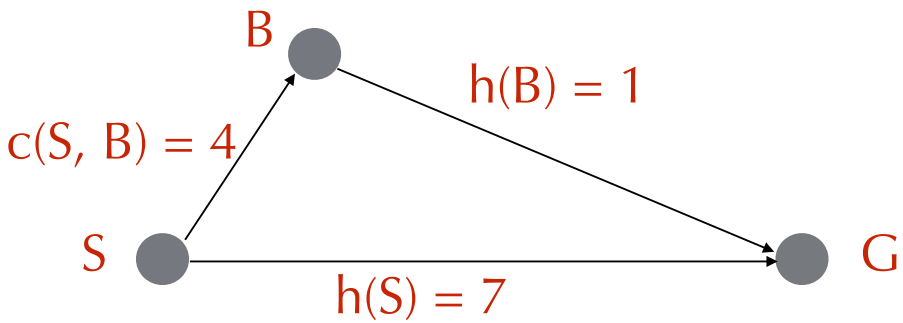
La Condizione di Consistenza
Esempio

Nell'esempio precedente la condizione di consistenza non era soddisfatta:



La Condizione di Consistenza
Esempio

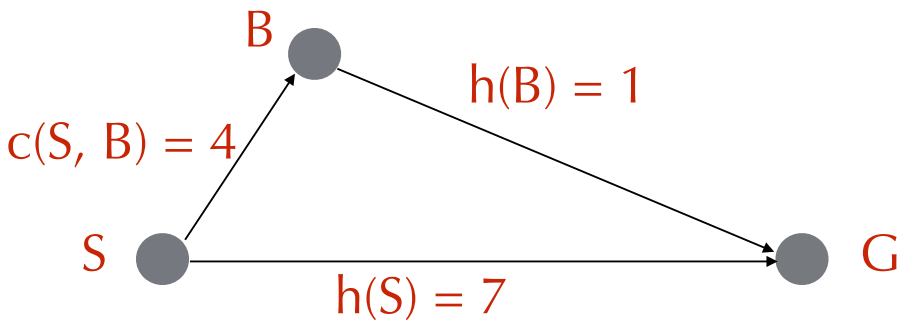
Nell'esempio precedente la condizione di consistenza non era soddisfatta:



$$h(S) \leq c(S, B) + h(B) \ ?$$

La Condizione di Consistenza
Esempio

Nell'esempio precedente la condizione di consistenza non era soddisfatta:

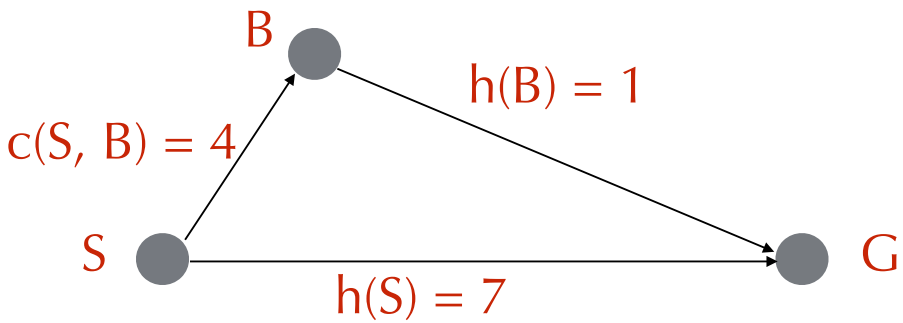


$$h(S) \leq c(S, B) + h(B) \; ?$$

$$7 \leq 4 + 1 \; ?$$

La Condizione di Consistenza
Esempio

Nell'esempio precedente la condizione di consistenza non era soddisfatta:



h(S) ≤ c(S, B) + h(B) ?

7 ≤ 4 + 1 ?

NO!

Sintesi degli Argomenti Trattati nella Lezione

- La **ricerca best-first** è in pratica la **TREE-SEARCH** in cui i nodi con costo minimo (secondo una qualche misura) vengono espansi prima.

- Gli algoritmi best-first usano tipicamente una **funzione euristica** $h(n)$ che stima il costo di una soluzione partendo da un nodo n.

- La **ricerca best-first greedy** o "golosa" espande sempre i nodi con $h(n)$ minima. Il tempo di ricerca di solito decresce rispetto a un algoritmo non informato, ma l'algoritmo non è né ottimale né completo.

- La **ricerca A*** espande i nodi a cui corrisponde un valore minimo di $f(n) = g(n) + h(n)$. A* è completa e ottima se possiamo garantire che $h(n)$ sia **ammissibile** (per la **TREE-SEARCH**) o anche **consistente** (per la versione **GRAPH-SEARCH** che segue il teorema presentato). La complessità spaziale di A* è ancora proibitiva.

- Le prestazioni degli algoritmi di ricerca euristici dipendono dalla qualità della funzione euristica. Si possono talvolta costruire delle buone euristiche ad esempio rilassando la definizione del problema.

Riferimenti

Russell, S., Norvig, P. *Artificial Intelligence - a Modern Approach*, fourth edition, Pearson, 2021.

Nilsson, N.J. *Artificial Intelligence - a New Synthesis*, Morgan Kaufman, 1998.

Hart, P., Nilsson, N. e Raphael, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. Syst. Science and Cybernetics*, SC-4(2), 1968, pp. 100-107.

Hart, P., Nilsson, N. e Raphael, B. "Correction to 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths,'", *SIGART Newsletter*, n. 37, dicembre 1972, pp. 28-29.

Nilsson, N. *Principles of Artificial Intelligence*, San Francisco: Morgan Kaufmann, 1980.