STRATEGIE INFORMATE

- L'intelligenza di un sistema non è misurabile in termini di capacità di ricerca, ma nella capacità di utilizzare conoscenza sul problema per eliminare il pericolo dell'esplosione combinatoria.

- Se il sistema avesse un qualche controllo sull'ordine nel quale vengono generate le possibili soluzioni, sarebbe allora utile disporre questo ordine in modo che le soluzioni vere e proprie abbiano un'alta possibilità di comparire prima.

- L'intelligenza, per un sistema con capacità di elaborazione limitata consiste nella saggia scelta di cosa fare dopo.....".

STRATEGIE INFORMATE

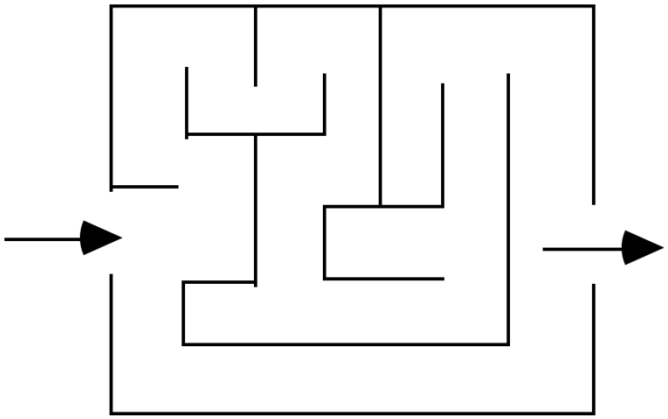
- **Newell-Simon**:
  - I metodi di ricerca precedenti in uno spazio di profondità d e fattore di ramificazione b risultano di complessità proporzionale a b^d per trovare un goal in una delle foglie.
  - Questo è inaccettabile per problemi di una certa complessità. Invece di espandere i nodi in modo qualunque utilizziamo conoscenza euristica sul dominio (funzioni di valutazione) per decidere quale nodo espandere per primo.

STRATEGIE INFORMATE

- Le funzioni di valutazione **danno una stima computazionale dello sforzo** per raggiungere lo stato finale.
- In particolare:
  - Il tempo speso per valutare mediante una funzione euristica il nodo da espandere deve corrispondere a una riduzione nella dimensione dello spazio esplorato.
  - Trade-off fra tempo necessario nel risolvere il problema (livello base) e tempo speso nel decidere come risolvere il problema (meta-livello).
  - Le ricerche non informate non hanno attività di meta-livello.

- PROBLEMI:
  - Come trovare le funzioni di valutazione corrette, cioè come si fa a valutare bene quale è il nodo più "promettente"?
  - È difficile caratterizzare numericamente la conoscenza empirica sul problema.
  - Non sempre la scelta più ovvia è la migliore.

ESEMPIO



- Scelta euristica: muoversi sempre per ridurre la distanza dall'uscita.
- Nel caso di questo labirinto sceglieremmo la strada più lunga.

ESEMPIO: GIOCO DEL FILETTO

- La distanza dal goal è stimata in base al numero di caselle fuori posto:

stato

1	2	3
7	8	4
6		5

goal

1	2	3
8		4
7	6	5

  - Sinistra: distanza 2
  - Destra: distanza 4
  - Alto: distanza 3

- Non è detto che la distanza stimi esattamente il numero di mosse necessarie

1	2	3
7		4
6	8	5

Distanza=3 ma in realtà sono necessarie 4 mosse per arrivare alla soluzione.

Ricerca Best First

Usa una ***funzione di valutazione*** che calcola un numero che rappresenta la desiderabilità relativa all'espansione di nodo.

Best-first significa scegliere come nodo da espandere quello che sembra più desiderabile.

QueuingFn = inserisce I successori in ordine decrescente di desiderabilità.

Casi particolari:

- ***ricerca greedy*** (golosa)
- ***ricerca A****

STRATEGIE INFORMATE

- BEST-FIRST: Cerca di muoversi verso il massimo (minimo) di una funzione che “stima” il costo per raggiungere il goal.

- Greedy:
  - Sia L una lista dei nodi iniziali del problema, ordinati secondo la loro distanza dal goal (nodi più vicini al goal precedono quelli più lontani);
  - Sia n il primo nodo di L. Se L è vuota fallisci;
  - Se n è il goal fermati e ritorna esso più la strada percorsa per raggiugerlo
  - Altrimenti rimuovi n da L e aggiungi a L tutti i nodi figli di n con label la strada percorsa partendo dal nodo iniziale. Poi ordina tutta la lista L in base alla stima della distanza relativa dal goal. Ritorna al passo 2

Una realizzazione della ricerca best-first

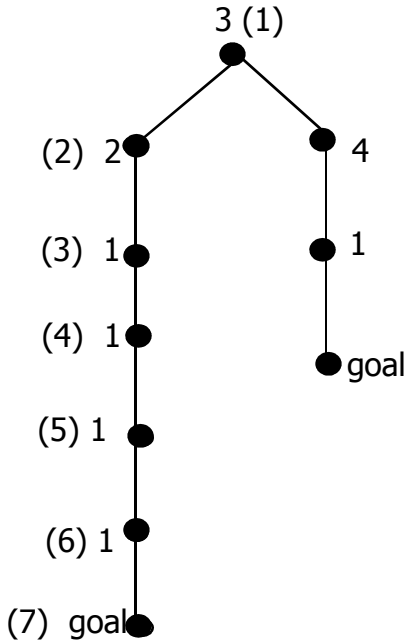
function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

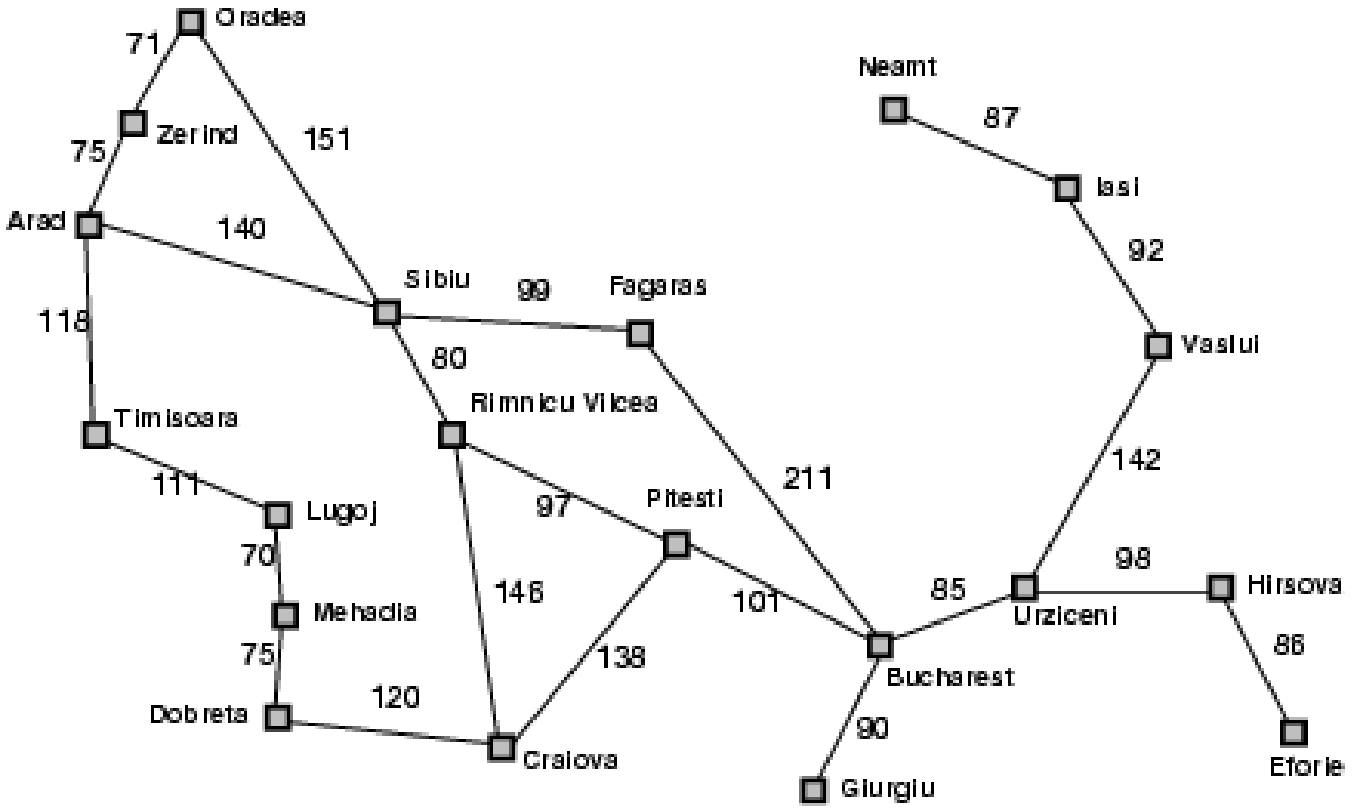
usa l'algoritmo di ricerca generale e la **funzione di valutazione *EvalFn***

RICERCA LUNGO IL CAMMINO MIGLIORE

- La ricerca best-first non è detto che trovi la soluzione migliore, ovvero **il cammino migliore** per arrivare alla soluzione (si pensi all'esempio del labirinto).
- Questo perché la tecnica best-first cerca di trovare il più presto possibile un nodo con distanza 0 dal goal senza curarsi di trovare quel nodo che ha profondità più bassa.



Romania con costo in km

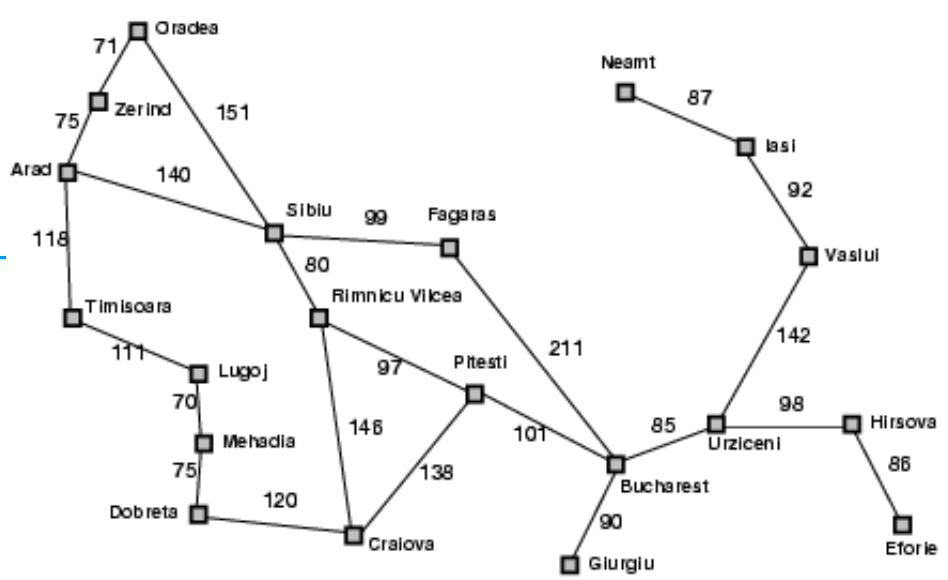


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Ricerca greedy best-first

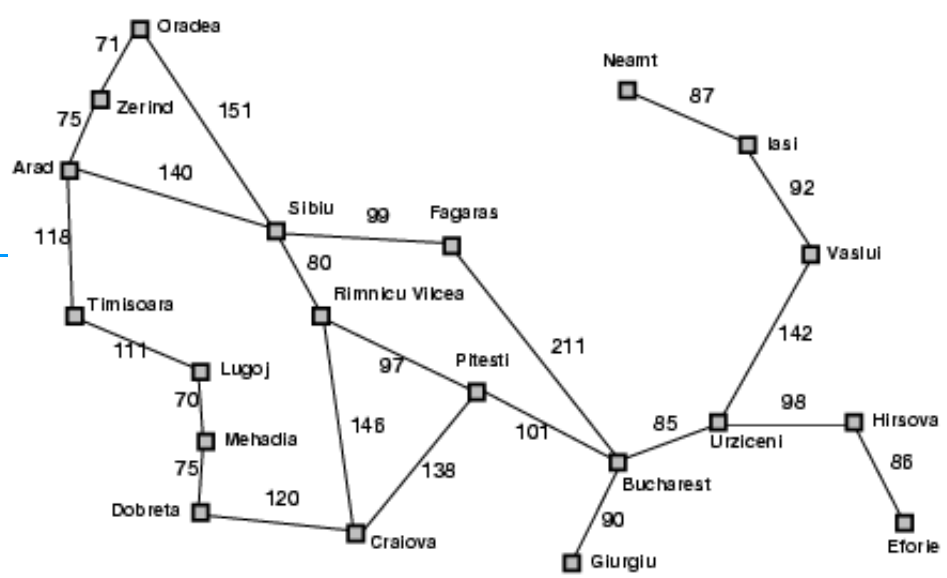
- Funzione di valutazione $f(n) = h(n)$ (euristica)
- = stima del costo da n al *goal*
- e.g., $h_{SLD}(n)$ = distanza in linea d'aria fra n e Bucharest
- La ricerca greedy best-first espande il nodo che sembra piu' vicino al goal

Greedy best-first: esempio



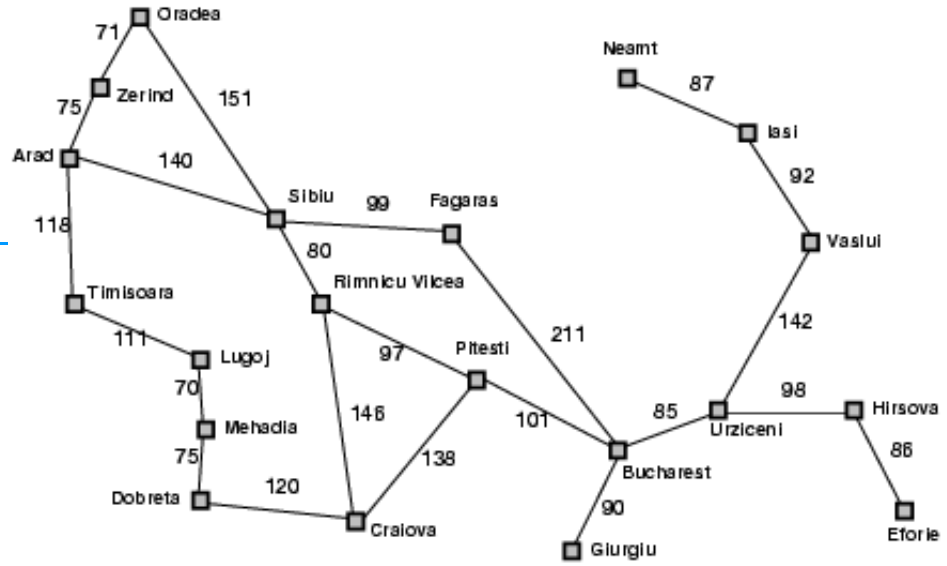
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first esempio

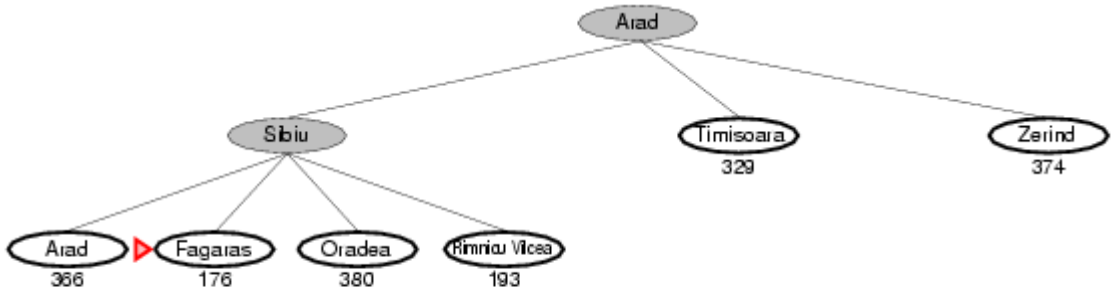


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

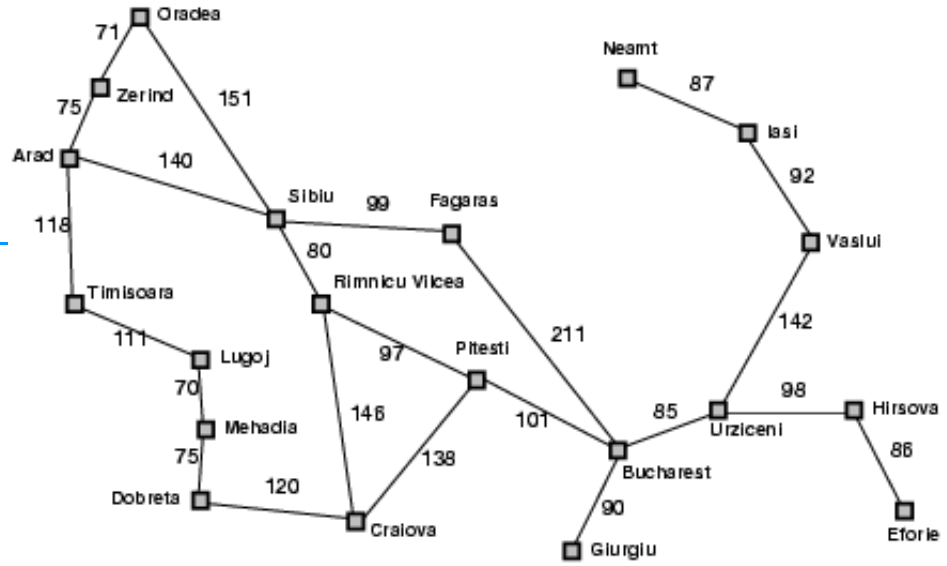
Greedy best-first esempio



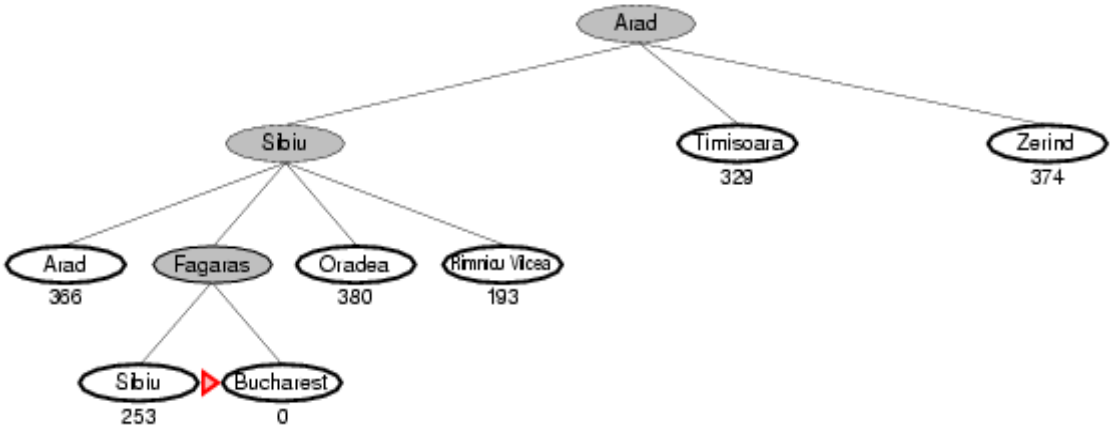
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Greedy best-first esempio



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



STRATEGIE INFORMATE: problemi di questo approccio

- La ricerca di questo tipo non è ottimale e può essere incompleta (presenta gli stessi difetti della ricerca in profondità).
- Nel caso peggiore, se abbiamo profondità d e fattore di ramificazione b il numero massimo di nodi espansi sarà b^d. (complessità temporale).
- Poiché, inoltre, tiene in memoria tutti i nodi la complessità spaziale coincide con quella temporale.
- Con una buona funzione euristica, la complessità spaziale e temporale possono essere ridotte sostanzialmente.

Algoritmo A*

- Invece di considerare solo la distanza dal goal, considera anche il "costo" nel raggiungere il nodo N dalla radice.

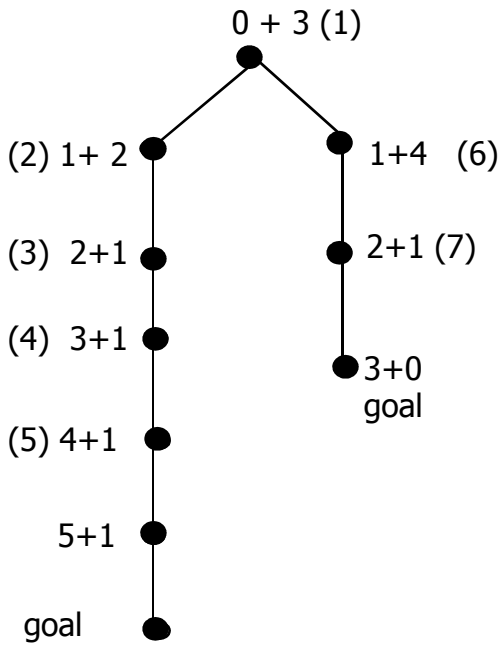
- Espandiamo quindi i nodi in ordine crescente di
  $f(n) = g(n) + h'(n)$

- Dove g(n) è la profondità del nodo e h'(n) la distanza dal goal.

- Scegliamo come nodo da espandere quello per cui questa somma è più piccola.

Algoritmo A*

- In pratica cerchiamo di combinare i vantaggi della ricerca in profondita' (efficienza) e della ricerca a costo uniforme (ottimalità e completezza).

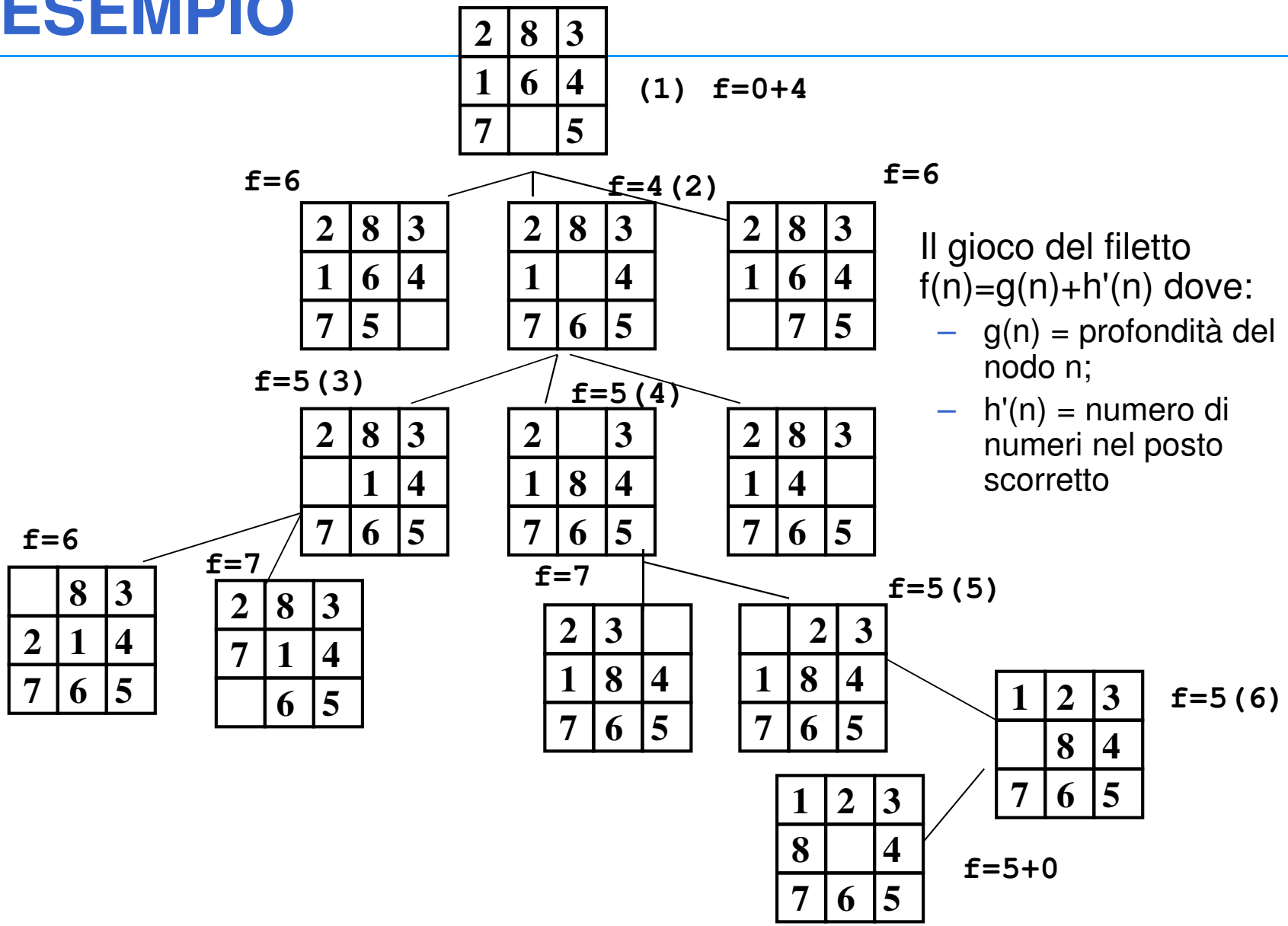


Algoritmo A*

- Sia L una lista dei nodi iniziali del problema.
- Sia n il nodo di L per cui g(n) + h'(n) è minima. Se L è vuota fallisci;
- Se n è il goal fermati e ritorna esso più la strada percorsa per raggiungerlo
- Altrimenti rimuovi n da L e aggiungi a L tutti i nodi figli di n con label la strada percorsa partendo dal nodo iniziale. Ritorna al passo 2

- Nota:
  - ancora una volta l'algoritmo non garantisce di trovare la strada ottima. Nell'esempio, se il nodo con label 5 fosse stato il goal sarebbe stato trovato prima di quello sulla strada destra (ottimale).

ESEMPIO



Il gioco del filetto
f(n)=g(n)+h'(n) dove:

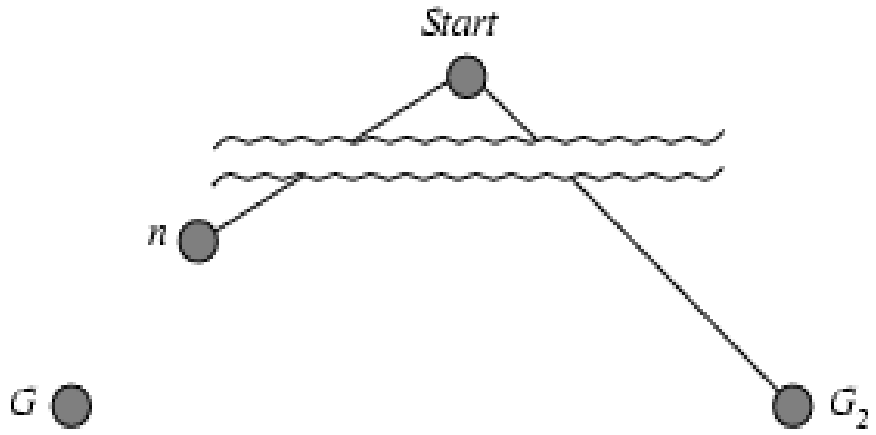
- g(n) = profondità del nodo n;
- h'(n) = numero di numeri nel posto scorretto

ALGORITMO A*

- A* non garantisce di trovare la soluzione ottima (dipende dalla funzione euristica).
- Si supponga di indicare con h(n) la vera distanza fra il nodo corrente e il goal.
- La funzione euristica h'(n) è ottimistica se abbiamo sempre che
  h'(n) <= h(n).
- Tale funzione euristica è chiamata ammissibile.
- Teorema:
  - Se h'(n) <= h(n) per ogni nodo, allora l'algoritmo A* troverà sempre il nodo goal ottimale (in caso di alberi -TREE-SEARCH).
- Ovviamente l'euristica perfetta h' = h è sempre ammissibile.
- Se h' = 0 otteniamo sempre una funzione euristica ammissibile $\Rightarrow$ ricerca breadth-first (ovviamente la depth-first non lo sarà mai).

Ottimalita` di A^* (dimostrazione)

- Si supponga di avere generato un goal sub-ottimo *G2 e* di averlo inserito nella coda. Sia *n* un nodo non espanso nella coda tale che *n* e' nella strada piu' breve verso il goal ottimo *G*.



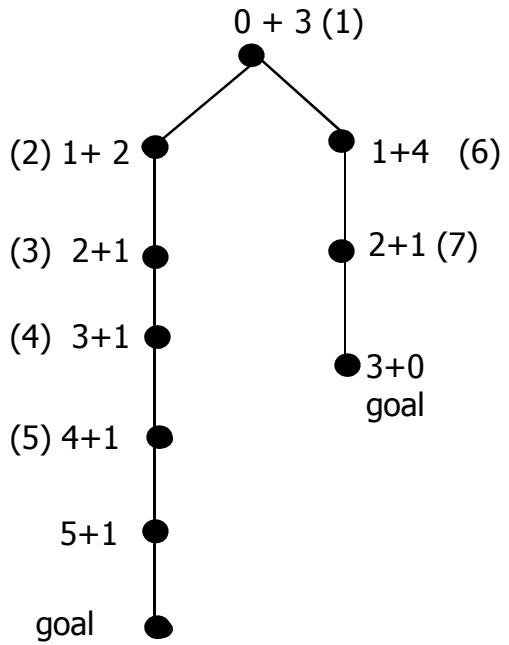
Ottimalita` di A^* (dimostrazione - cont)

- $f(G_2) = g(G_2)$ poiche' $h(G_2) = 0$
- $g(G_2) > g(G)$ poiche' G_2 e' sub-ottima
- $f(G) = g(G)$ poiche' $h(G) = 0$
- $f(G_2) > f(G)$ da sopra

- $h'(n) \leq h(n)$ poiche' h' e' ammissibile
- $g(n) + h'(n) \leq g(n) + h(n) = f(G)$ (n e' sulla strada di G)
- $f(n) \leq f(G)$

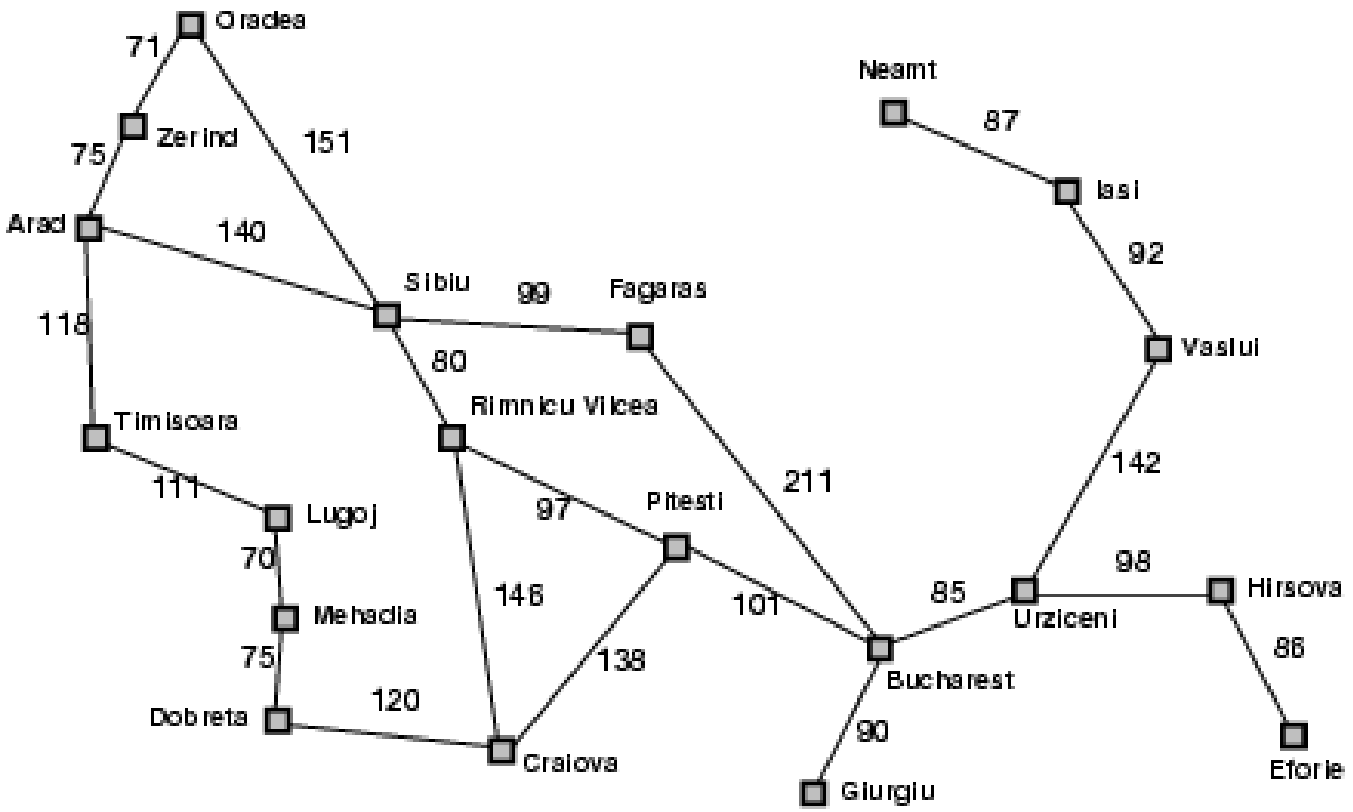
quindi $f(G_2) > f(n)$, e A^* non selezionera' mai G_2 per l'espansione.

ESEMPIO: RICERCA CON FUNZIONE EURISTICA AMMISSIBILE



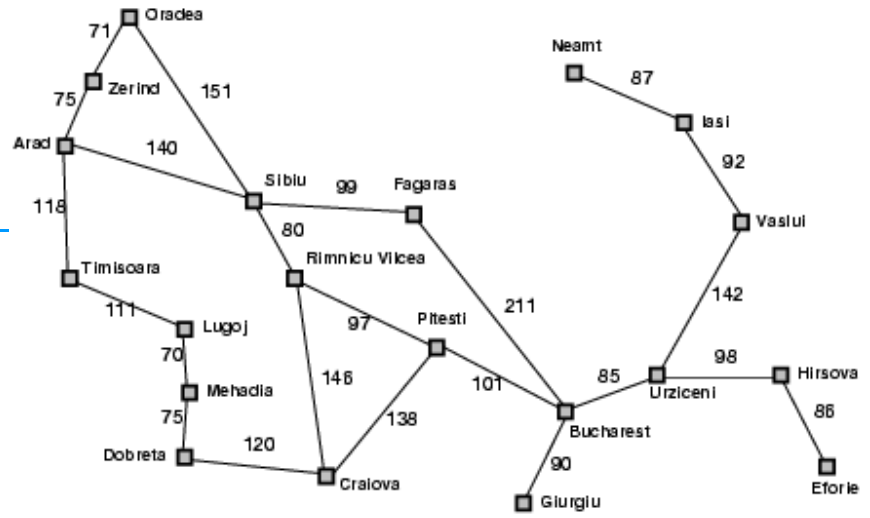
- La scelta precedente per il gioco del filetto (contare le tessere fuori posto h') è sempre ammissibile perché ogni mossa può ridurlo al più di 1.

Romania con costo in km

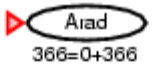


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

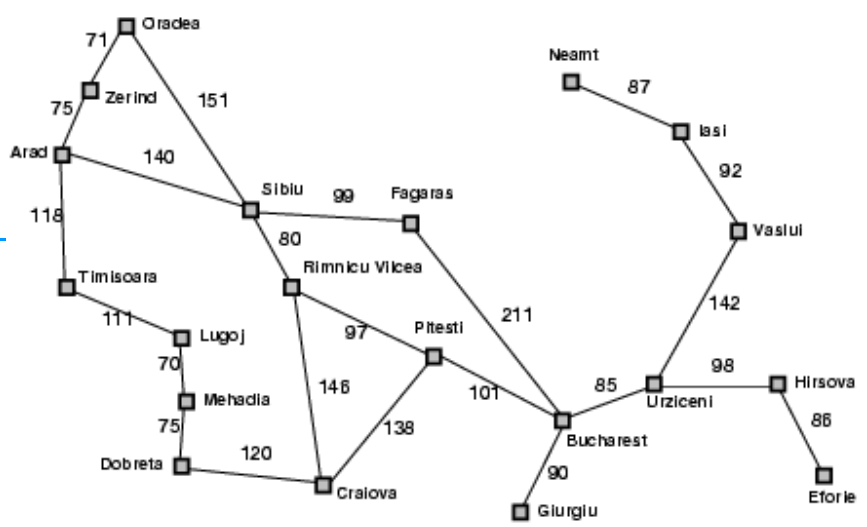
A* esempio



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

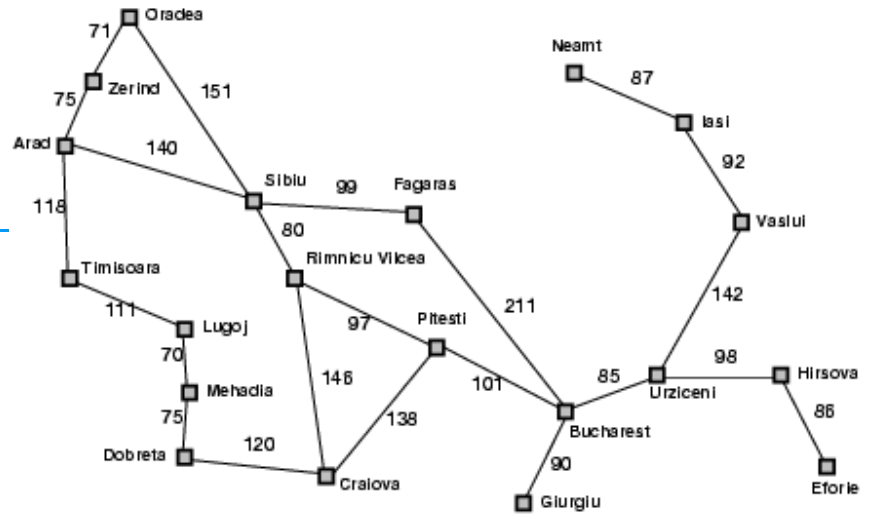


A* esempio

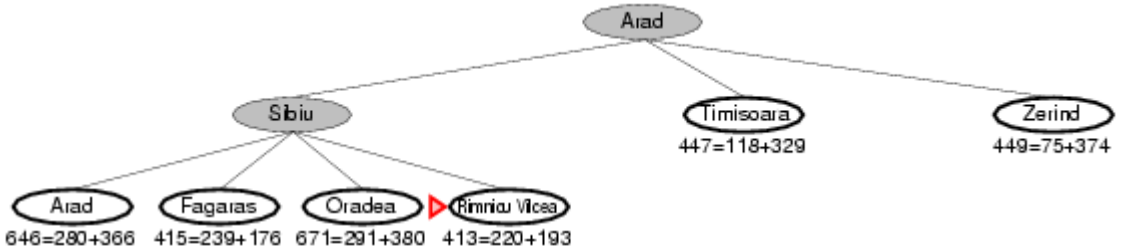


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

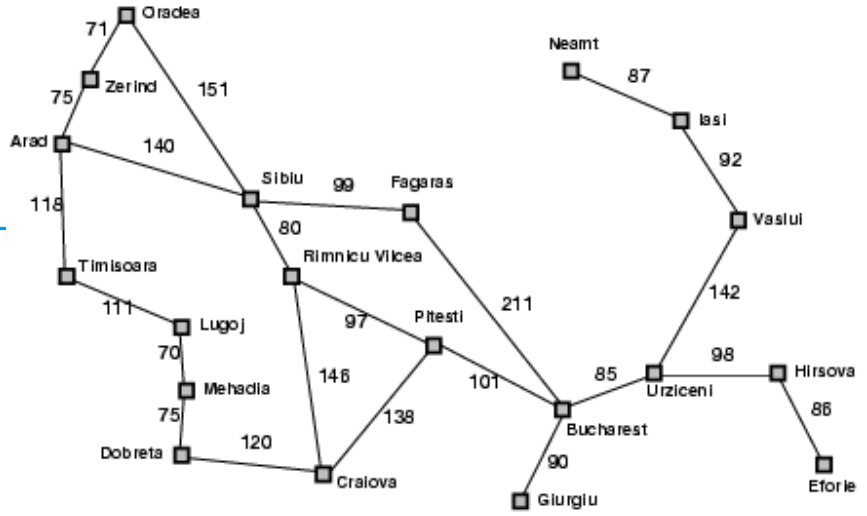
A* esempio



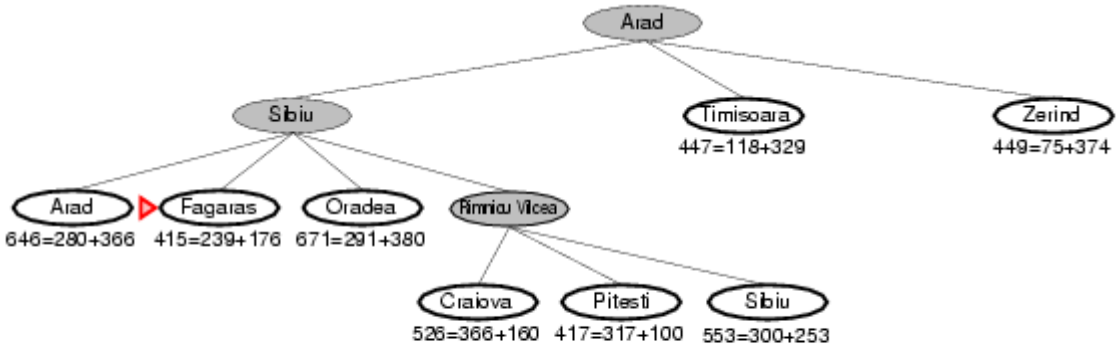
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



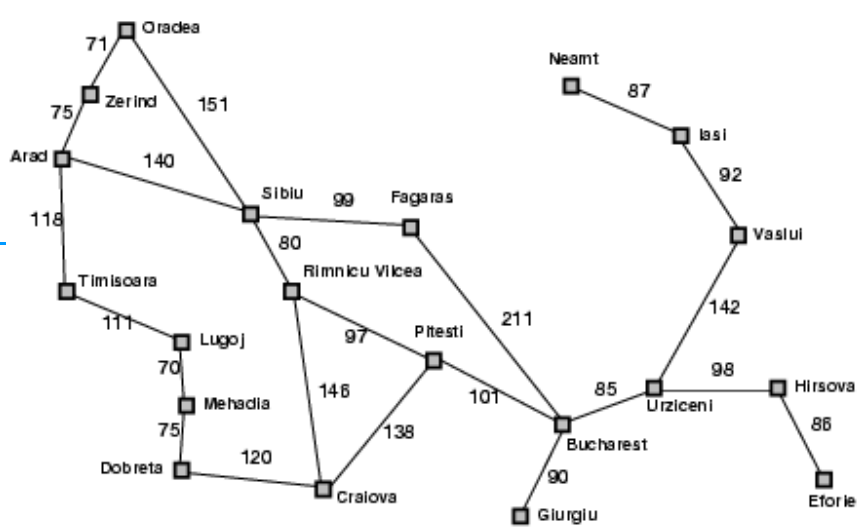
A* esempio



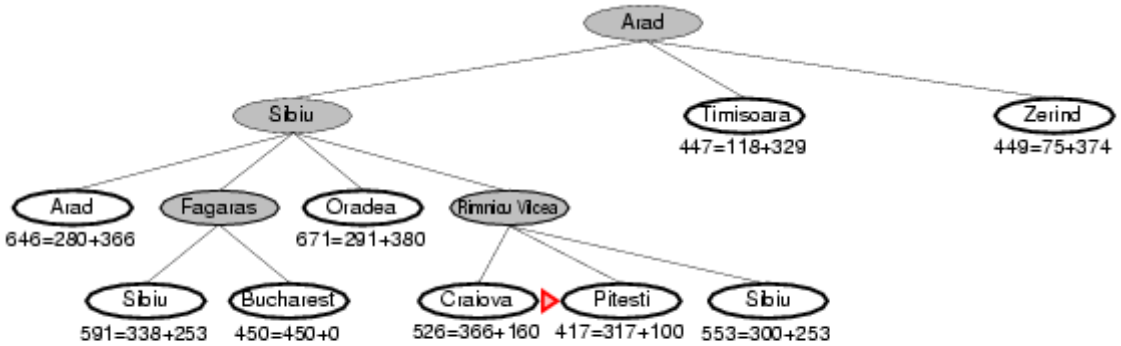
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



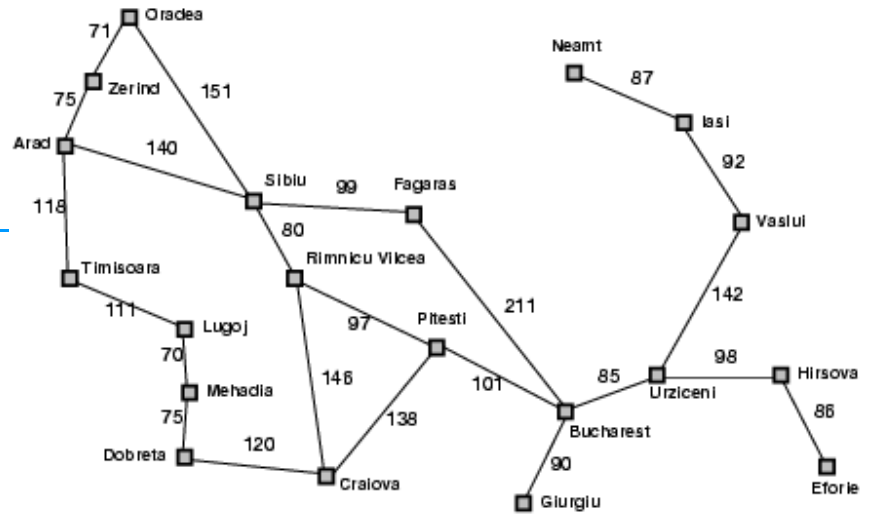
A* esempio



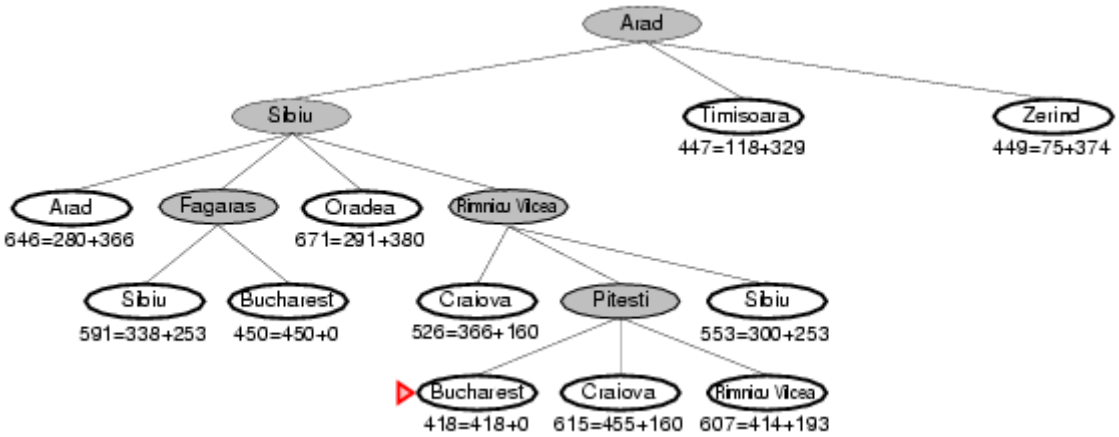
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



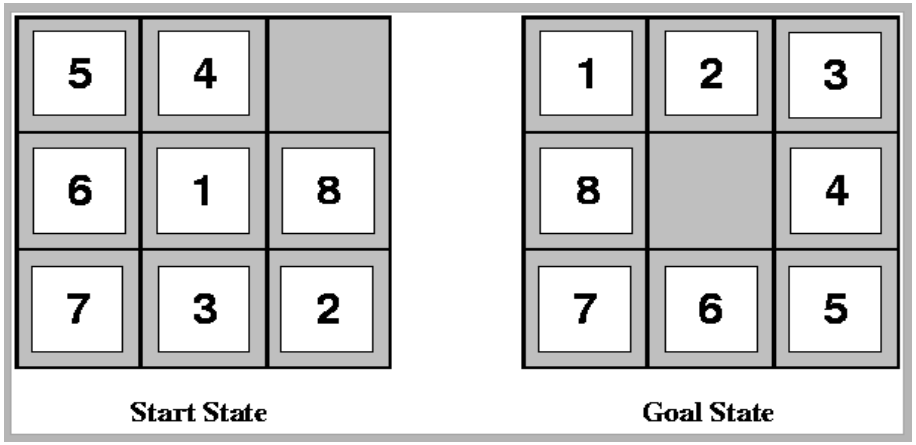
A* esempio



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



FUNZIONI EURISTICHE AMMISSIBILI



E' possibile definire differenti funzioni euristiche. Ad esempio:

h_1= numero di tessere che sono fuori posto (h_1= 7)

h_2= la somma delle distanze dalle posizioni che le tessere devono assumere nella configurazione obiettivo. La distanza è una somma delle distanze orizzontali e verticali (**distanza di Manhattan**). Le tessere da 1 a 8 nello stato iniziale danno una distanza h_2= 2+3+3+2+4+2+0+2 = 18

Sono entrambe ammissibili

FUNZIONI EURISTICHE AMMISSIBILI

- Poiché h'<=h'*<=h la migliore è h'*.
- È meglio usare una funzione euristica con valori più alti, a patto che sia ottimista.
- Come inventare funzioni euristiche?
- Spesso il costo di una soluzione esatta per un problema rilassato è una buona euristica per il problema originale.
- Se la definizione del problema è descritta in un linguaggio formale è possibile costruire problemi rilassati automaticamente.
- A volte si puo' utilizzare il massimo fra diverse euristiche.
- h'(n) = max(h1(n),h2(n)…hm(n)).

ESEMPIO: GIOCO DEL FILETTO

- Descrizione: una tessera può muoversi dal quadrato A al quadrato B se A è adiacente a B e B è vuoto.
- Problemi rilassati che rimuovono alcune condizioni:
- Una tessera può muoversi dal quadrato A al quadrato B se A è adiacente a B.(distanza di manhattan)
- Una tessera può muoversi dal quadrato A al quadrato B se B è vuoto.
- Una tessera può muoversi dal quadrato A al quadrato B (tesserine fuori posto).

DA ALBERI A GRAFI

- Da alberi a grafi:
  - Abbiamo assunto fin qui che lo spazio di ricerca sia un albero e non un grafo. Non è quindi possibile raggiungere lo stesso nodo da strade diverse.
  - L'assunzione è ovviamente semplicistica: si pensi al gioco del filetto, ai missionari e cannibali ecc.
  - Come si estendono gli algoritmi precedenti per trattare con i grafi?

Stati ripetuti: Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Ricerca in grafi ed algoritmo A*

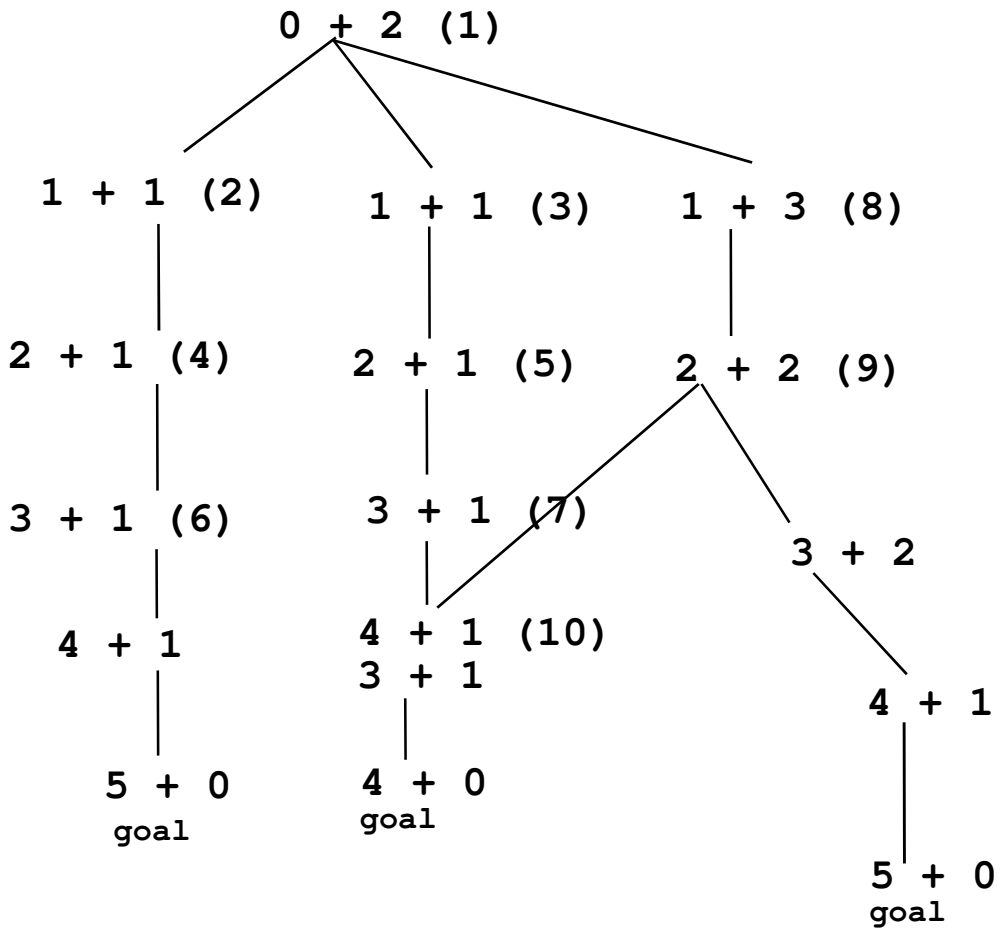
- Due liste:
  - Nodi espansi e rimossi dalla lista per evitare di esaminarli nuovamente (nodi chiusi);
  - Nodi ancora da esaminare (nodi aperti).

- A* che cerca in un grafo invece che in un albero.
- Modifiche:
  - Il grafo puo' diventare un albero con stati ripetuti.
  - Aggiunta della lista dei nodi chiusi e assunzione che g(n) valuta la distanza minima del nodo n dal nodo di partenza.

ALGORITMO A* PER GRAFI

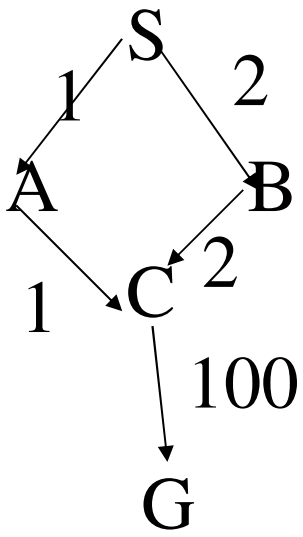
- Sia La una lista aperta dei nodi iniziali del problema.
- Sia n il nodo di La per cui g(n) + h'(n) è minima. Se La è vuota fallisci;
- Se n è il goal fermati e ritorna esso più la strada percorsa per raggiungerlo
- Altrimenti rimuovi n da La, inseriscilo nella lista dei nodi chiusi Lc e aggiungi a La tutti i nodi figli di n con label la strada percorsa partendo dal nodo iniziale. Se un nodo figlio è già in La, non riaggiungerlo, ma aggiornalo con la strada migliore che lo connette al nodo iniziale. Se un nodo figlio è già in Lc, non aggiungerlo a La, ma, se il suo costo è migliore, aggiorna il suo costo e i costi dei nodi già espansi che da lui dipendevano.
- Ritorna al passo 2

ESEMPIO


0 + 2 (1)
1 + 1 (2)
1 + 1 (3)
1 + 3 (8)
2 + 1 (4)
2 + 1 (5)
2 + 2 (9)
3 + 1 (6)
3 + 1 (7)
3 + 2
4 + 1
4 + 1 (10)
3 + 1
4 + 1
5 + 0
goal
4 + 0
goal
5 + 0
goal

Algoritmo A*: considerazioni

- E se non controllassimo la lista dei nodi chiusi? Se in essa conservassimo solo gli stati e la utilizzassimo solo per evitare l'espansione dei nodi in essa contenuti?
- Saremmo piu' efficienti, ma…
- Potremmo non trovare la strada migliore:
- Esempio: euristica h*: A=100, C=90, S=0, G=0, B=1



S espansione Chiusi S Aperti B(3) A (101)
B espansione Chiusi B S Aperti A(101) C(94)
C espansione Chiusi B S C Aperti A(101) G(104)
A espansione Chiusi B S C A Aperti C eliminato quindi G (104)
G goal ma no migliore strada

Algoritmo A* (considerazioni - consistenza)

- In questo caso troveremmo la strada che passa per B (non e' quella ottima)
- Il motivo e' che a causa della stima troppo ottimista di B, C e' espanso prima di trovare la strada alternativa che passa per A.
- Ulteriore condizione su h: consistenza (o monotonicita`)
- Definizione: una euristica e' consistente se per ogni nodo *n*, ogni successore *n'* di *n* generato da ogni azione *a*,

  h(n) = 0 se lo stato corrispondente coincide con il goal

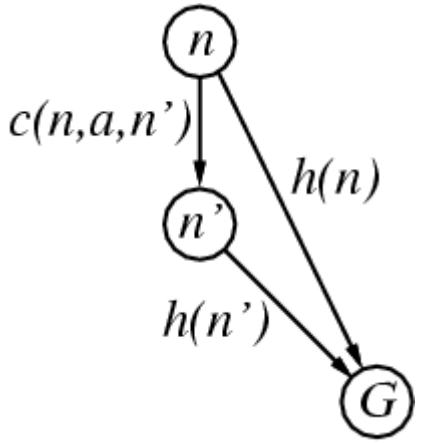
  $h(n) \leq c(n,a,n') + h(n')$

- Con la monotonicita` ogni nodo e' raggiunto prima mediante la strada migliore.

Euristica Consistente

- Se *h* e' consistente abbiamo che:

f(n') = g(n') + h(n')
= g(n) + c(n,a,n') + h(n')
≥ g(n) + h(n)
= f(n)



- i.e., *f(n)* non decresce mai lungo un cammino.

- Theorema : Se *h(n)* e' consistente, A* usando `GRAPH-SEARCH` e' ottimale

ALGORITMI COSTRUTTIVI e ALGORITMI DI RICERCA LOCALE

- **Gli algoritmi costruttivi** (visti fino ad ora) **generano** una soluzione ex-novo aggiungendo ad una situazione di partenza (vuota o iniziale) delle componenti in un particolare ordine fino a "costruire" la soluzione completa.

- **Gli algoritmi di ricerca locale** partono da una soluzione iniziale e iterativamente cercano di rimpiazzare la soluzione corrente con una "migliore" in un intorno della soluzione corrente. In questi casi non interessa la "strada" per raggiungere l'obiettivo.

- Diversi problemi (ad esempio, il problema delle 8 regine) hanno la proprietà che la descrizione stessa dello stato contiene tutte le informazioni necessarie per la soluzione (il cammino è irrilevante).
- In questo caso gli algoritmi con miglioramenti iterativi spesso forniscono l'approccio più pratico.
- Ad esempio, possiamo cominciare con tutte le regine sulla scacchiera e poi spostarle per ridurre il numero di attacchi.

RICERCA LOCALE

- La ricerca locale è basata sull'esplorazione iterativa di "soluzioni vicine", che possono migliorare la soluzione corrente mediante modifiche locali.

- *Struttura dei "vicini" (neighborhood).* Una struttura dei "vicini" è una funzione *F* che assegna a ogni soluzione *s* dell'insieme di soluzioni *S* un insieme di soluzioni *N(s)* sottoinsieme di S.

- La scelta della funzione *F* è fondamentale per l'efficienza del sistema e definisce l'insieme delle soluzioni che possono essere raggiunte da s in un singolo passo di ricerca dell'algoritmo.

- Tipicamente è definita implicitamente attraverso le possibili mosse.

- La soluzione trovata da un algoritmo di ricerca locale non è detto sia ottima globalmente, ma può essere ottima rispetto ai cambiamenti locali.

RICERCA LOCALE (continua)

- Massimo (minimo) locale
  - Un massimo locale è una soluzione s tale che per qualunque s' appartenente a N(s), data una funzione di valutazione f, $f(s) \geq f(s')$.
  - Quando risolviamo un problema di massimizzazione (minimizzazione) cerchiamo un massimo (minimo) globale s_{opt} cioè tale che per qualunque s, $f(s_{opt}) > f(s)$.
  - Più è largo il neighborhood più è probabile che un massimo locale sia anche un massimo globale (ma ovviamente esiste un problema di complessità computazionale). Quindi, più è largo un neighborhood più aumenta la qualità della soluzione

- L'algoritmo base a cui ci si riferisce è detto "iterative improvement" (miglioramentio iterativo):
  - Tipicamente, tale algoritmo parte con una soluzione di tentativo generata in modo casuale o mediante un procedimento costruttivo e cerca di migliorare la soluzione corrente muovendosi verso i vicini. Se fra i vicini c'è una soluzione migliore, tale soluzione va a rimpiazzare la corrente. Altrimenti termina in un massimo (minimo) locale.

ALGORITMO PIU' NOTO: HILL CLIMBING

- Un modo semplice per capire questo algoritmo è di immaginare che tutti gli stati facciano parte di una superficie di un territorio. L'altezza dei punti corrisponde alla loro funzione di valutazione.

- Dobbiamo muoverci cercando le sommità più alte tenendo conto solo dello stato corrente e dei vicini immediati.

- Hill climbing si muove verso i valori più alti (o bassi) che corrispondono a massimi (o minimi) locali

- Problema: l'algoritmo può incappare in un minimo (massimo) locale di scarsa qualità:
  - Ha il limite di essere un metodo locale che non valuta tutta la situazione, ma solo gli stati abbastanza vicini a quello corrente.

- L'algoritmo non gestisce un albero di ricerca, ma solo lo stato e la sua valutazione.

PROBLEMI

- Massimi locali:
  - Stati migliori di tutti i vicini, ma peggiori di altri stati che non sono nelle vicinanze. Il metodo ci spinge verso il massimo locale, peggiorando la situazione.

- Pianori o Altopiani:
  - Zone piatte dello stato di ricerca in cui gli stati vicini hanno tutti lo stesso valore. In quale direzione muoversi (scelta casuale)?

- Crinali:
  - È una zona più alta di quelle adiacenti verso cui dovremmo andare, ma non possiamo andarci in modo diretto. Dobbiamo quindi muoverci in un'altra direzione per raggiungerlo.

- Una possibile (semplice) soluzione: ripartire con una nuova ricerca da una soluzione di partenza random o generata in modo costruttivo. Si salva poi la soluzione migliore dopo una serie di tentativi (bound dovuto al tempo di CPU o numero di iterazioni).

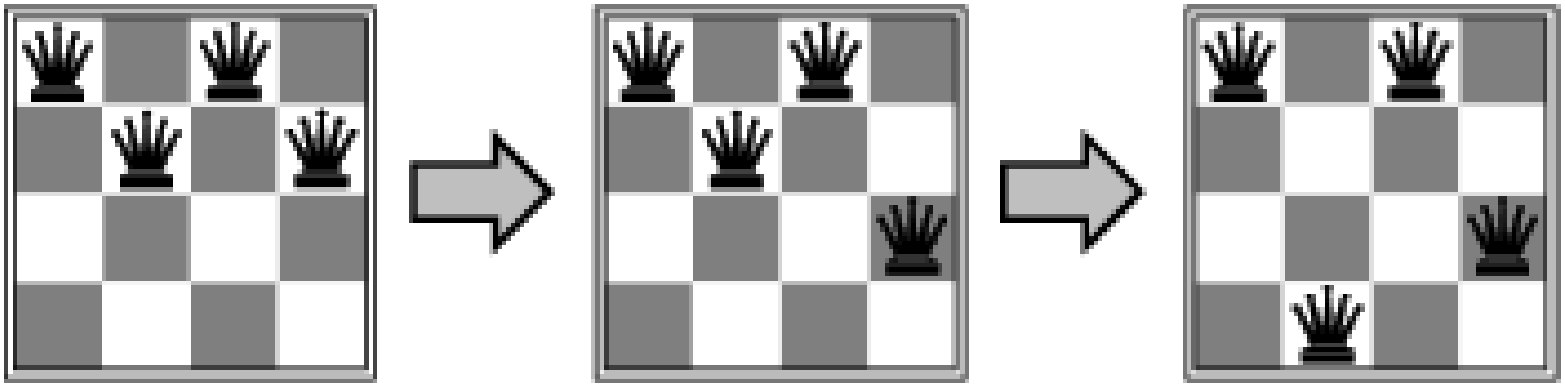
Hill-climbing

- "Like climbing Everest in thick fog with amnesia"
  Si noti che l'algoritmo non tiene traccia dell'albero di ricerca.

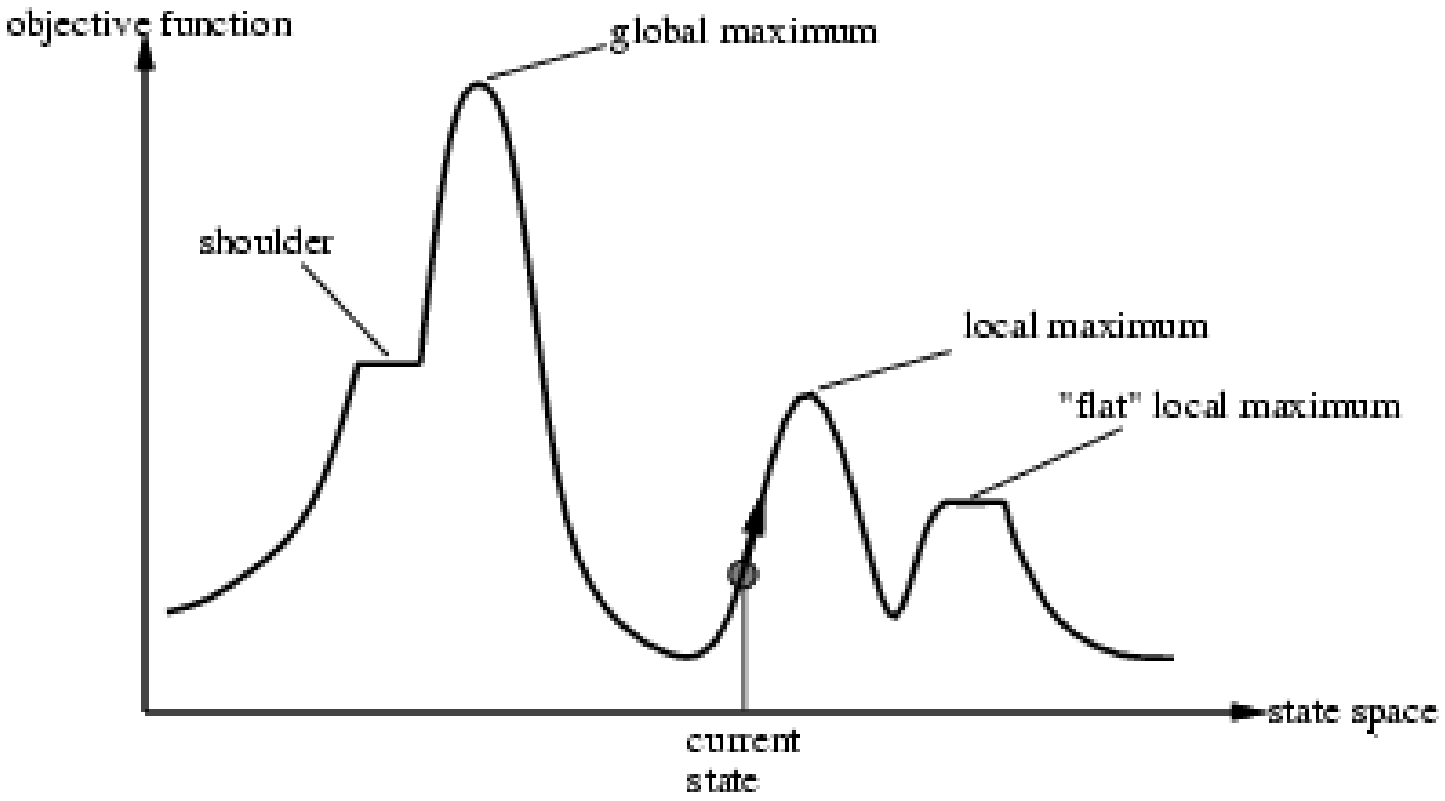
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Esempio: N- regine

- Inserire n regine in una scacchiera in modo che non si attacchino.



Hill-climbing



META EURISTICHE

- Si definiscono meta-euristiche l'insieme di algoritmi, tecniche e studi relativi all'applicazione di criteri euristici per risolvere problemi di ottimizzazione (e quindi di migliorare la ricerca locale con criteri abbastanza generali).

- Ne citiamo alcuni:
- **ANT colony optimization**
  - ispirata al comportamento di colonie di insetti.
  - Tali insetti mostrano capacità globali nel trovare, ad esempio, il cammino migliore per arrivare al cibo dal formicaio (algoritmi di ricerca cooperativi)

- **Tabu search**
  - La sua caratteristica principale è l'utilizzo di una memoria per guidare il processo di ricerca in modo da evitare la ripetizione di stati già esplorati.

META EURISTICHE (continua)

- **Algoritmi genetici**

  - Sono algoritmi evolutivi ispirati ai modelli dell'evoluzione delle specie in natura. Utilizzano il principio della selezione naturale che favorisce gli individui di una popolazione che sono più adatti ad uno specifico ambiente per sopravvivere e riprodursi.

  - Ogni individuo rappresenta una soluzione con il corrispondente valore della funzione di valutazione (*fitness*). I tre principali operatori sono: *selezione*, *mutazione* e *ricombinazione*.

META EURISTICHE (Continua)

- **Simulated Annealing**
  - Cerca di evitare il problema dei massimi locali, seguendo in pratica la strategia in salita, ma ogni tanto fa un passo che non porta a un incremento in salita.
  - Quando rimaniamo bloccati in un massimo locale, invece di cominciare di nuovo casualmente potremmo permettere alla ricerca di fare alcuni passi in discesa per evitare un massimo locale.
  - Ispirato al processo di raffreddamento dei metalli.
  - In pratica, ci suggerisce di esaminare, ogni tanto, nella ricerca un nodo anche se sembra lontano dalla soluzione.

Ricerca in grafi AND/OR

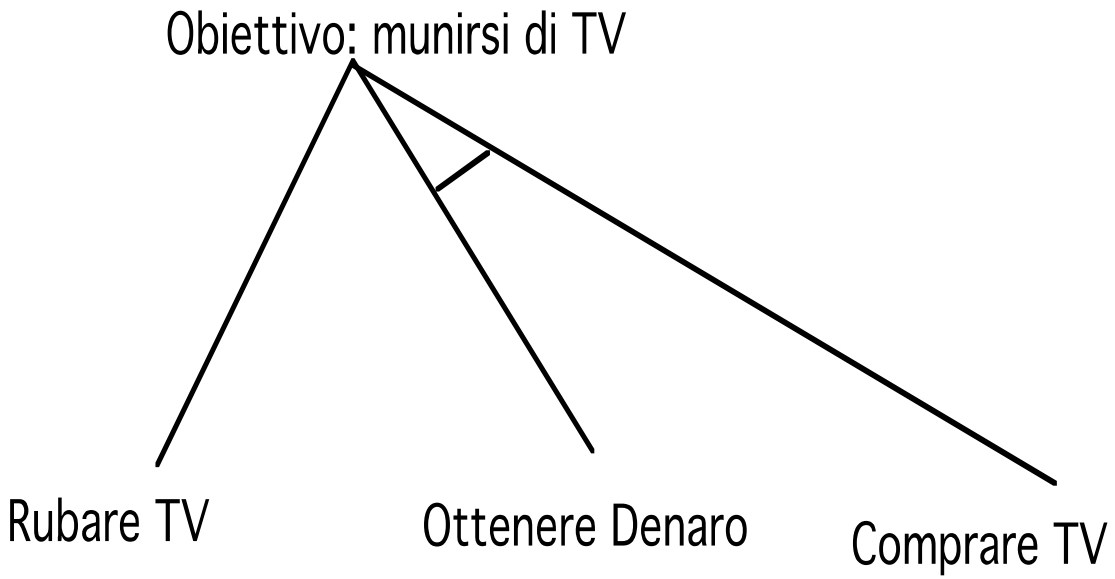
- Fino a questo momento abbiamo discusso strategie di ricerca per grafi OR nei quali vogliamo trovare un singolo cammino verso la soluzione.

- Il grafo (o albero) AND/OR risulta appropriato quando si vogliono rappresentare problemi che si possono risolvere scomponendoli in un insieme di problemi più piccoli che andranno poi tutti risolti.

- $\Rightarrow$ arco AND che deve puntare a un qualunque numero di nodi successori che si devono tutti risolvere per risolvere il nodo AND stesso.

- Dal nodo AND possono anche partire rami OR che indicano soluzioni alternative.

ESEMPIO



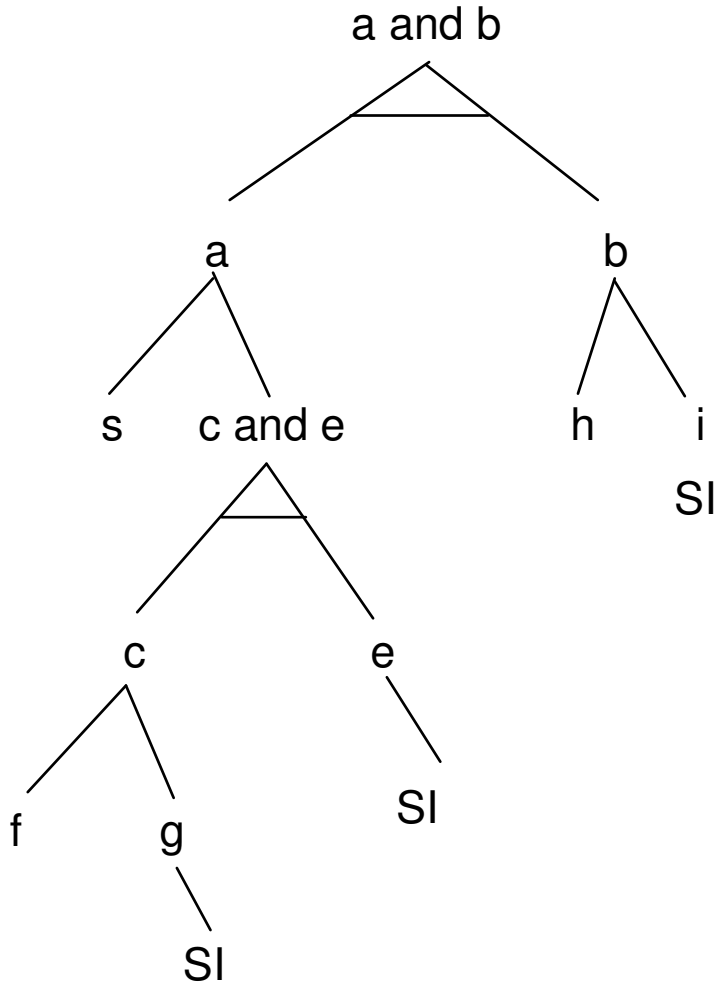
- L'algoritmo di ricerca va opportunamente modificato per essere in grado di trattare gli archi AND.

TRASFORMAZIONE DI ALBERI AND/OR IN ALBERI DI RICERCA:

- g.
- e.
- i.

- s → a.
- c and e → a.
- f → c.
- g → c.
- h → b.
- i → b.

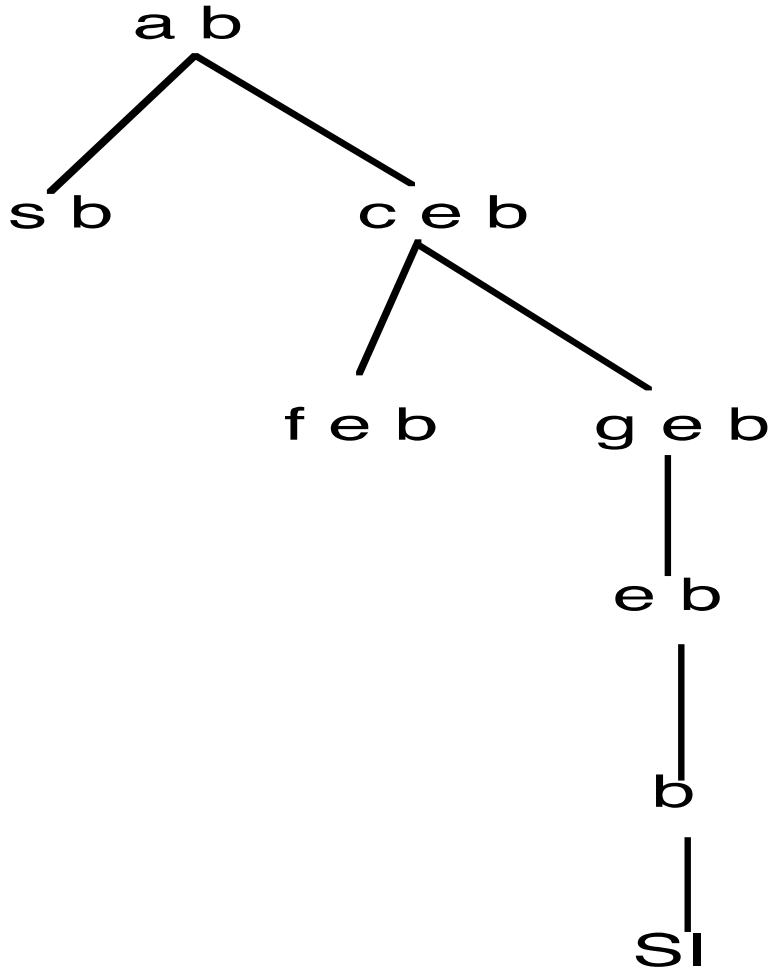
- Goal: a and b



TRASFORMAZIONE DI ALBERI AND/OR IN ALBERI DI RICERCA:

Nota:

- Non ha alcuna influenza, al fine del raggiungimento della soluzione, l'ordine di espansione dei goal in AND.

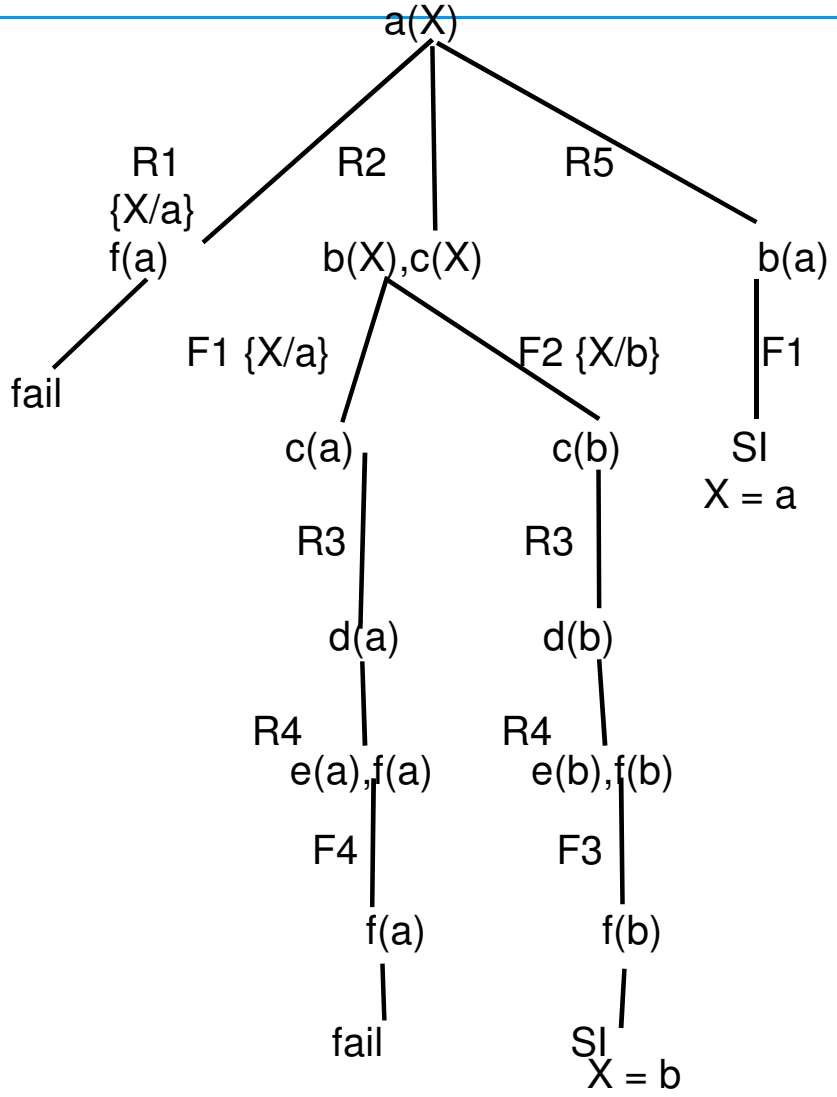


ALTRO ESEMPIO:

- **FATTI:**
  - F1: b(a).
  - F2: b(b).
  - F3: e(b).
  - F4: e(a).
  - F5: f(b).

  - R1: f(a) $\rightarrow$ a(X).
  - R2: b(X) and c(X) $\rightarrow$ a(X).
  - R3: d(Y) $\rightarrow$ c(Y).
  - R4: e(X) and f(X) $\rightarrow$ d(X).
  - R5: b(a) $\rightarrow$ a(a).
- **GOAL: a(X)**

ALBERO DI RICERCA BACKWARD:



ALTRE TECNICHE EURISTICHE

- SCELTA IN BASE AGLI OPERATORI
  - (regole, task inseriti in un'agenda):

- A) Insita nel controllo e influenzata IMPLICITAMENTE dall'utente:
  - Prolog: ordine testuale delle regole
  - OPS: strategie in base alla forma dell'antecedente (LEX MEA)

- B) Influenzata ESPLICITAMENTE dall'utente:
  - Attribuire alle regole dei valori di priorità rappresentati come numeri interi; in questo modo l'interprete selezionerà sempre la regola applicabile a più alta priorità.

ESEMPIO:

R1: IF <la temperatura è maggiore di 100 gradi>,
AND <la pressione ha valore p1>
THEN <attiva la valvola .....>
AND <segnala una situazione di allarme all'operatore> --- priorità 10

R2: IF <la pressione ha valore p1>
THEN <attiva la procedura di funzionamento normale PROC5> --- priorità 5.

- PRIORITÀ DINAMICHE:
  - a) lo stato della memoria di lavoro;
  - b) la presenza di particolari regole applicabili;
  - c) la precedente esecuzione di regole correlate;
  - d) l'andamento dell ' esecuzione per particolari scelte fatte precedentemente.

META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA

- PROBLEMI:
  - 1) risulta particolarmente complesso individuare il corretto valore di priorità;
  - 2) l'attribuzione dei valori di priorità è oscura: diventa dunque carente nel sistema la capacità di spiegazione.
  - 3) la modificabilità della base di conoscenza diminuisce: controllo e conoscenza sul dominio non separati; le regole presentano dipendenze implicite.

- META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA
- Può risolvere i problemi precedentemente citati (vasta applicazione non solo per il controllo).

- meta-regola:
  - MR1 "Usa regole che esprimono situazioni di allarme prima di regole che esprimono situazioni di funzionamento normale".

META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA

- ESEMPI: Sistemi Esperti: Molgen, Neomycin, Teiresias.

- VANTAGGI:
  - a) il sistema è maggiormente flessibile e modificabile; un cambiamento nella strategia di controllo implica semplicemente il cambiamento di alcune meta-regole.
  - b) la strategia di controllo è semplice da capire e descrivere.
  - c) potenti meccanismi di spiegazione del proprio comportamento.

TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

- Tale meta-conoscenza ha tre scopi fondamentali:
  - a) sono utilizzate meta-regole per definire strategie di controllo;
  - b) sono utilizzati "modelli" per descrivere regole di livello oggetto;
  - c) sono utilizzati metodi descrittivi espliciti per descrivere la rappresentazione utilizzata e dunque poterla modificare.

- CONTROLLO:
- Meta-Regola 1
  - **IF** <ci sono regole che **menzionano**.....>
  - **AND** <ci sono regole che **menzionano**.....>.
  - **THEN** <le prime dovrebbero essere eseguite
  - **prima** delle seconde> (FC=0.4).

- Meta-Regola 2
  - **IF** <il paziente è.....>
  - **AND** <ci sono regole che **menzionano** ...>
  - **THEN** <è certo che nessuna di esse
  - **sarà utile** ulteriormente> (FC=1).

TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

- meta-regola 1: ordine parziale sulle regole di livello oggetto;
- meta-regola 2: utilità di certe regole di livello oggetto;
  - NOTA: La sintassi delle meta-regole e` identica a quella di livello oggetto
- **NUOVE FUNZIONI:**
  - **MENTIONS: serve per referenziare le regole di livello oggetto;**
  - **UTILITY e BEFORE: servono per influenzare la strategia di controllo.**

TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

FUNZIONAMENTO:

- Trova le regole di livello oggetto applicabili (L);
- Trova le meta-regole associate (L');
- Valuta ogni meta-regola di L' e determina se è applicabile per alcune regole di L;
- Ordina o toglie elementi da L in accordo ai criteri delle meta-regole;
- Esegue ogni regola rimasta in L nell'ordine ottenuto.

META-CONOSCENZA PER IL CONTROLLO UNA NUOVA ARCHITETTURA

- **Così si espande anche l'interprete:**