

Generazione e risoluzione di labirinti

Anna Spanò - 27/03/24

licensed under CC BY 4.0

Contenuti

- Caratteristiche di un buon labirinto
- Struttura dati
- Generazione: RandomDFS
- Risoluzione: A* search
 - Dimostrazione ammissibilità
 - Ottimalità con coerenza
 - Applicazione a un labirinto
- Riferimenti

Caratteristiche di un buon labirinto

- Ha un bordo ben definito
- Ogni punto deve poter essere raggiungibile
- non è banale

struttura dati

visivamente e ...

rappresentazione “visiva” → **griglia 2D NxM**

4 movimenti possibili → ← ↓ ↑

N colonne					
0	1	2	...	N-1	M righe
N	N+1				
:					
:					
:					
N(M-1)				NH-1	

... internamente

struttura interna → grafo non orientato come lista di adiacenza
“potenziale”

- vertici = caselle
- archi = passaggi tra le caselle
- vettore di $N*M$ vertici
- ogni vertice ha 4 adiacenti, a parte i vertici al bordo → Bordo ben definito dalla connettività ✓

complessità spaziale:

- $O(N*M)$ vettore di vertici
- $O(N*M)$ lista di adiacenza “potenziale” come vettore di vettori
(grado di ogni vertice ≤ 4)

Generazione: RandomDFS

DFS ...

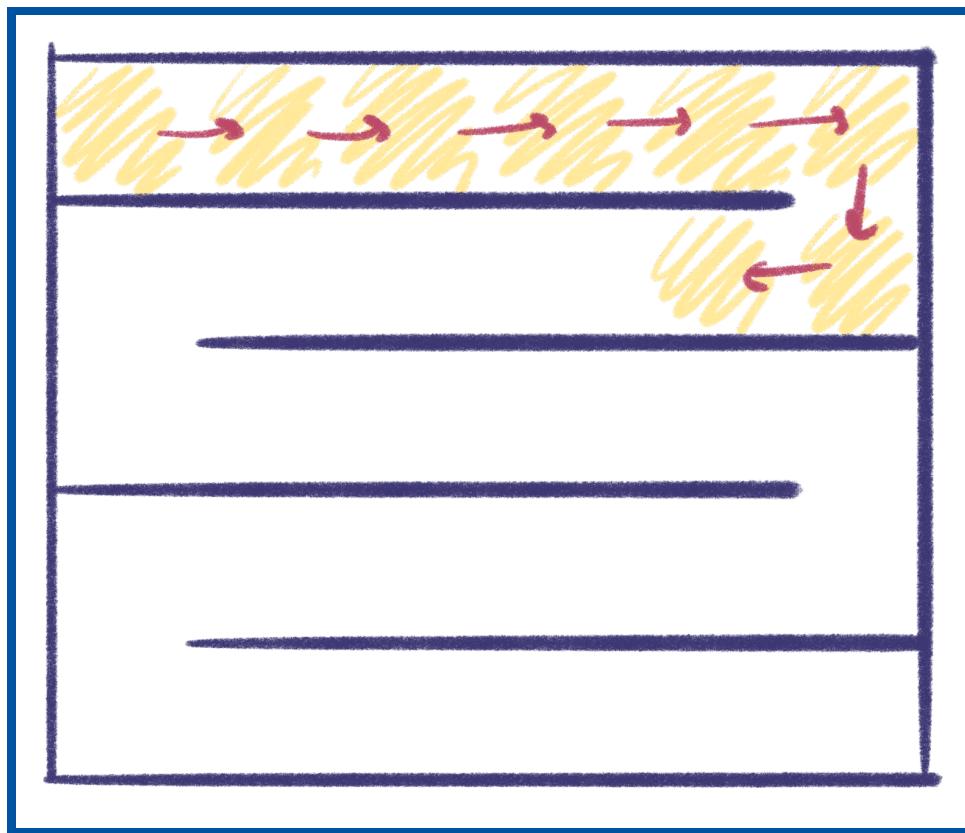
- si utilizza la capacità del Depth First Search di generare un albero di copertura → Ogni punto è raggiungibile ✓
- DFS seleziona gli archi “da aprire” → lista di adiacenza “vera” del labirinto è quella “potenziale” senza muri
- → Complessità spaziale rimane sempre la stessa per ragioni “visive”

... randomizzato

- DFS non randomizzato crea un unico lungo corridoio in base a come è inizializzata la lista (vd sotto) **RandomDFS**:
- scelgo un vertice adiacente a caso non ancora visitato e lo visito
- appena arrivo a un punto dove non ci sono più mosse disponibili “chiudo” il vertice e faccio il backtracking
- A causa della scelta random è meglio usare uno stack esplicito dove salvare i vertici “in progress”/Grey invece di una chiamata ricorsiva

DFS non randomizzato

lista di adiacenza = $\rightarrow \leftarrow \downarrow \uparrow$



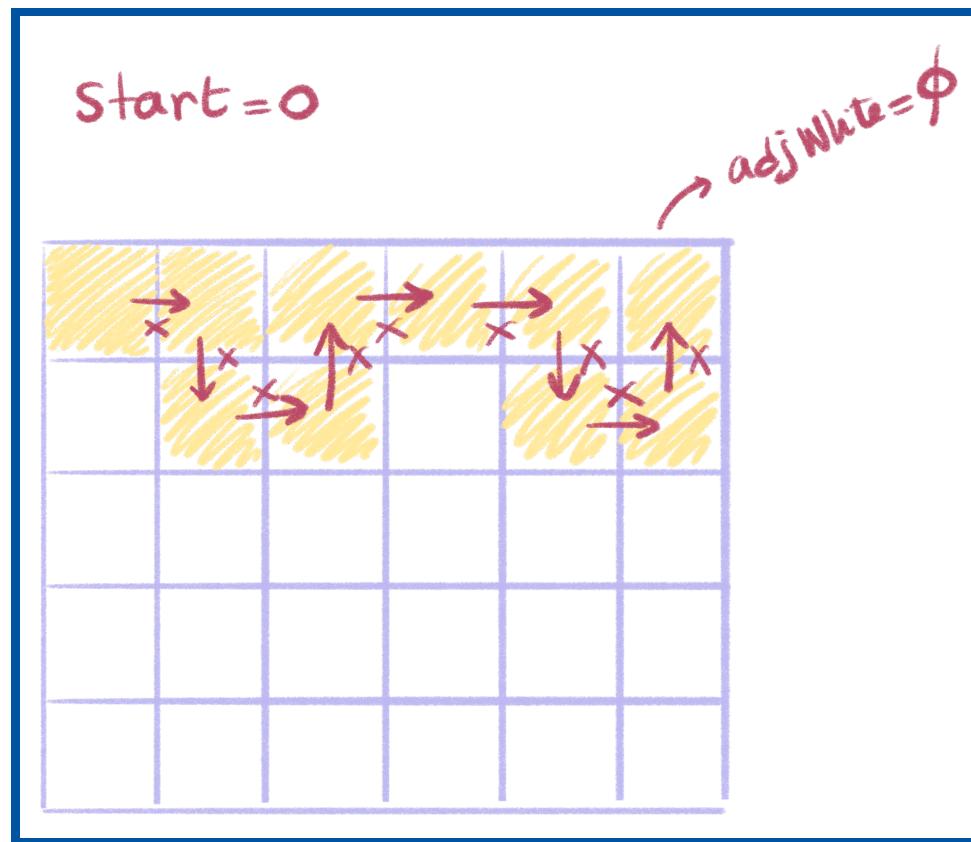
pseudocodice

```
1 RandomDFS (G,start)
2     Stack S;
3     S.push(start); //O(1)
4
5     while (S non vuoto):
6         u = S.top(); //O(1)
7         u.col=GREY;
8         inizializzare vettore vertici AdjDisponibili/AdjWhite //O(4)=O(1)
9         if (AdjWhite non vuoto):
10             selezionare vertice random in AdjWhite;
11             random.parent=u;
12             breakWall(u,random); //O(4)=O(1)
13             S.push(random);
14         else //nessun movimento possibile
15             u.col=BLACK;
16             estrarre vertice u da S; //O(1)
```

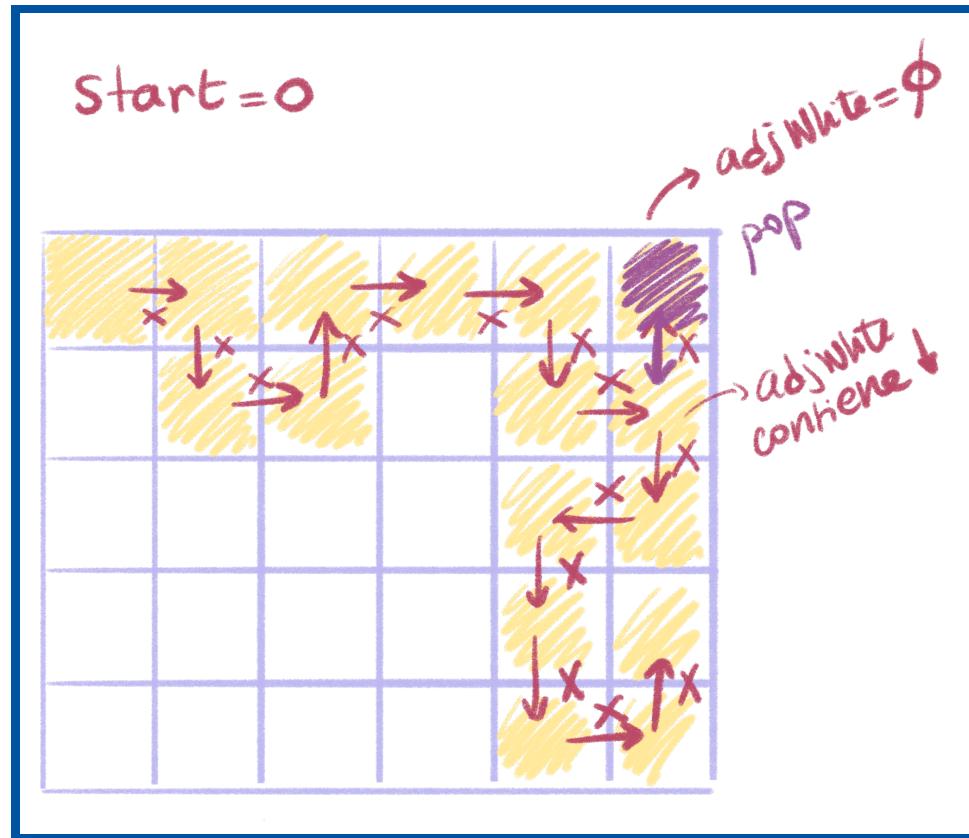
WORST CASE:

- **Complessità spaziale:** $O(|V|) = O(N \cdot M) \leftarrow$ caso “serpentone”
- **Complessità temporale:** $O(|V| + |E|) = O(N \cdot M) \leftarrow$ al massimo 4 adiacenti possibili per la scelta per ogni vertice

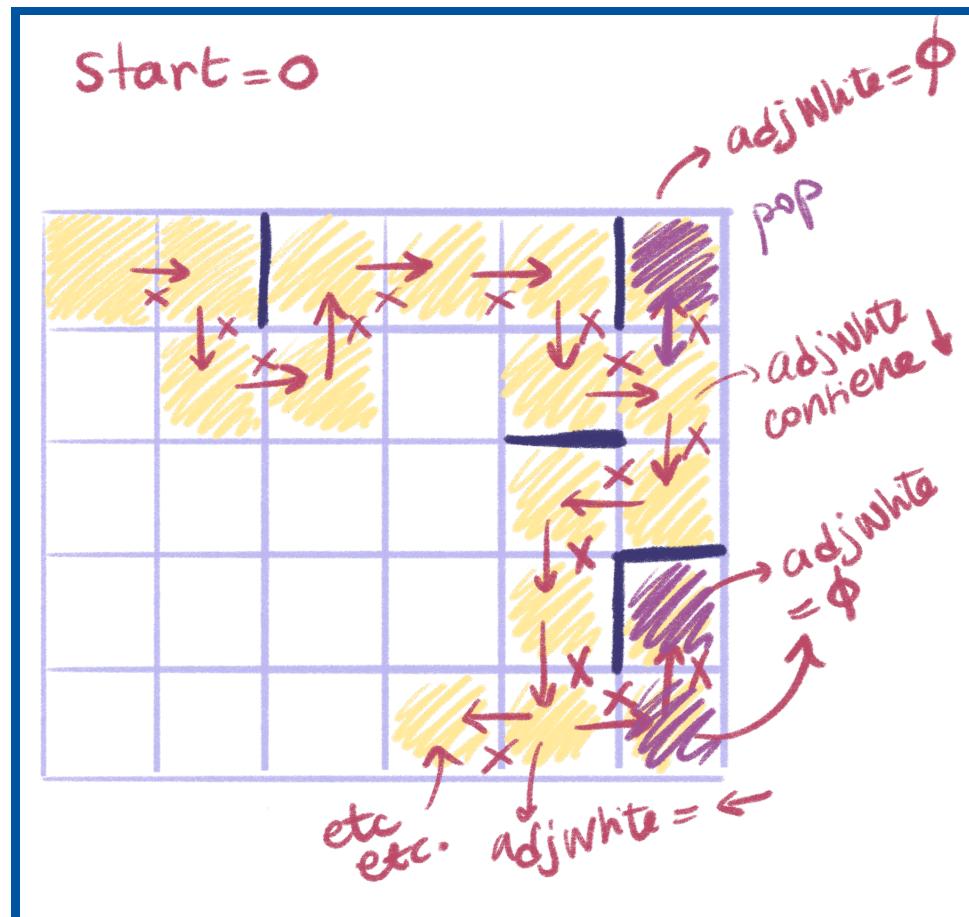
Esempio



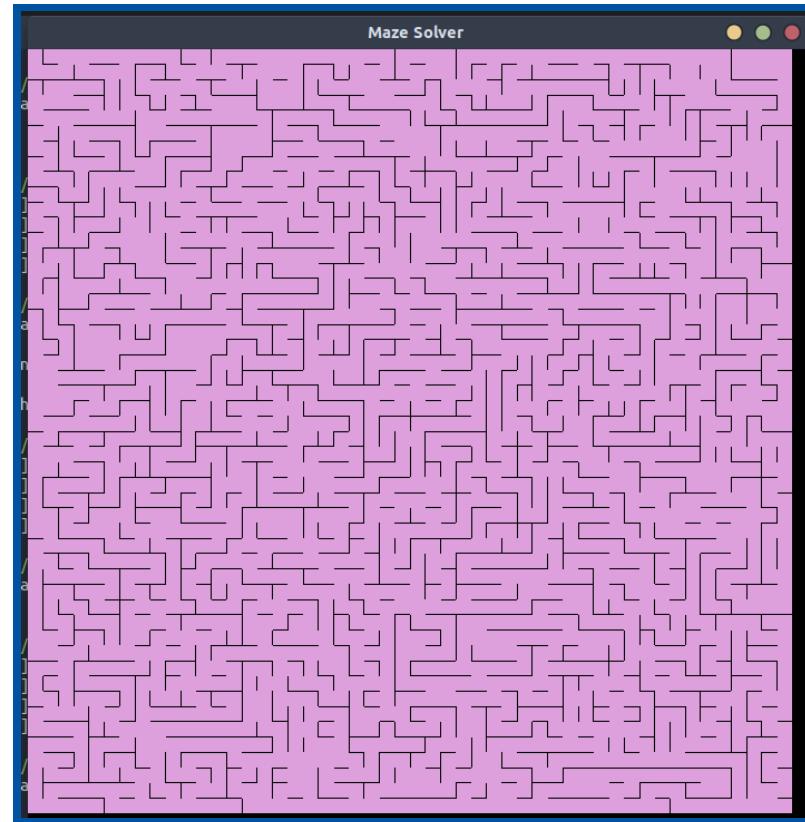
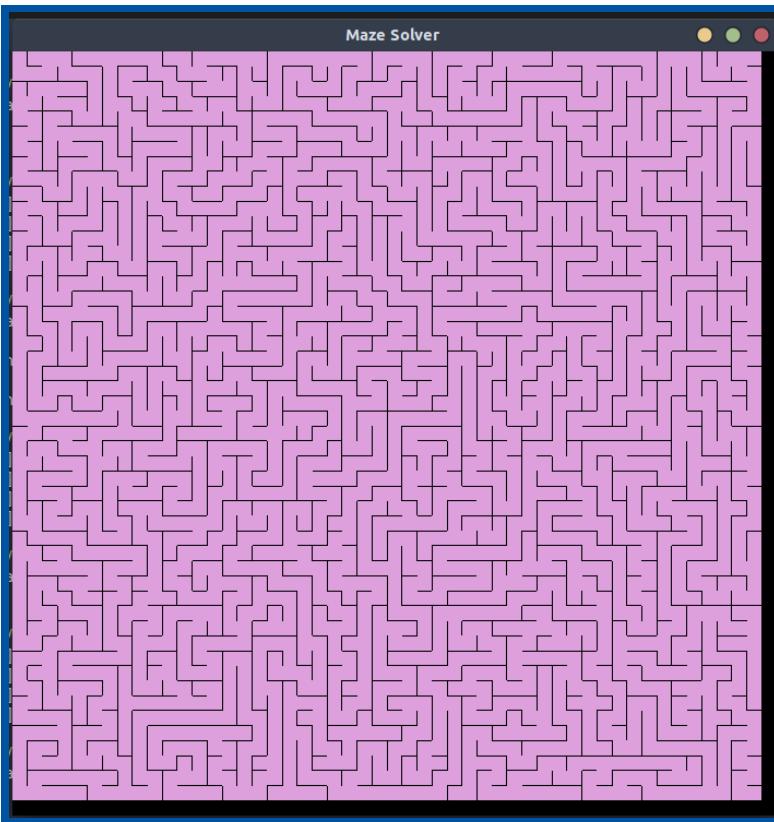
Esempio



Esempio



same DFS generated maze without and with loops



Risoluzione: A*

search

A* search

- Algoritmo del 1968
- In realtà è una classe di algoritmi **euristici** che ha come parametro l'euristica scelta
- Lavora su grafi orientati pesati → labirinto: **tutti i pesi sono 1**
- Come Dijkstra, A* **minimizza** una funzione $\tilde{f}(n)$
- ma **INFORMED SEARCH** vs **UNINFORMED SEARCH**
(Dijkstra,BFS,DFS)
- A* ottimo sotto euristica:
 - **Ammissibile:** $\tilde{h}(n) \leq h(n) \rightarrow$ A* trova lo shortest path
 - **Coerente (Consistent):** $h(m, n) + \tilde{h}(m) \leq \tilde{h}(n) \rightarrow$ A* esplora meno vertici rispetto ad altri algoritmi ammissibili

pseudocodice

```
1 A*(G, start, goal) //N.B. necessita di un goal!
2     for v in V: //O(N*M)
3         v.gTilde=inf;
4         v.parent=null;
5
6     start.gTilde=0;
7     start.fTilde=h(start,goal);
8
9     OPEN={start}; //minHeap
10    start.col=GREY;
11
```

```

10 while(OPEN non vuoto):
11     estraggo u da OPEN con fTilde minimo; //log(N)
12     if u=GOAL: exit;
13     for adj in adjList(u) //con edge=OPEN
14         costo = u.gTilde + w(u,adj) //u.gTilde = costo "so far" di u
15
16         // se adj non ancora espanso
17         // o il nuovo costo passando per u è minore del precedente
18         if (adj.col=WHITE or costo < adj.gTilde):
19             adj.gTilde = costo;
20             adj.fTilde = adj.gTilde + h(adj,goal);
21             adj.parent = u;
22             if adj not in OPEN //adj not GREY, se è BLACK lo riapro
23                 adj.col=GREY
24                 OPEN.add(adj); //log(N) Binomial heap/O(1) Fibonacci Heap
25             else OPEN.update(adj); //log(N)
26             u.col=BLACK; //add to CLOSED list

```

Complessità worst case:

- **Spaziale:**
 - $O(|V| + |E|) = O(N*M)$ per adj list
 - $O(|V|) = O(N*M)$ per il MinHeap (in generale $\leq |V|$)

```

10 while(OPEN non vuoto):
11     estraggo u da OPEN con fTilde minimo; //log(N)
12     if u=GOAL: exit;
13     for adj in adjList(u) //con edge=OPEN
14         costo = u.gTilde + w(u,adj) //u.gTilde = costo "so far" di u
15         if (adj.col=WHITE or costo < adj.gTilde):
16             adj.gTilde = costo;
17             adj.fTilde = adj.gTilde + h(adj,goal);
18             adj.parent = u;
19             if adj not in OPEN //adj not GREY, se è BLACK lo riapro
20                 adj.col=GREY
21                 OPEN.add(adj); //log(N) Binomial heap/O(1) Fibonacci Heap
22             else OPEN.update(adj); //log(N)
23             u.col=BLACK; //add to CLOSED list

```

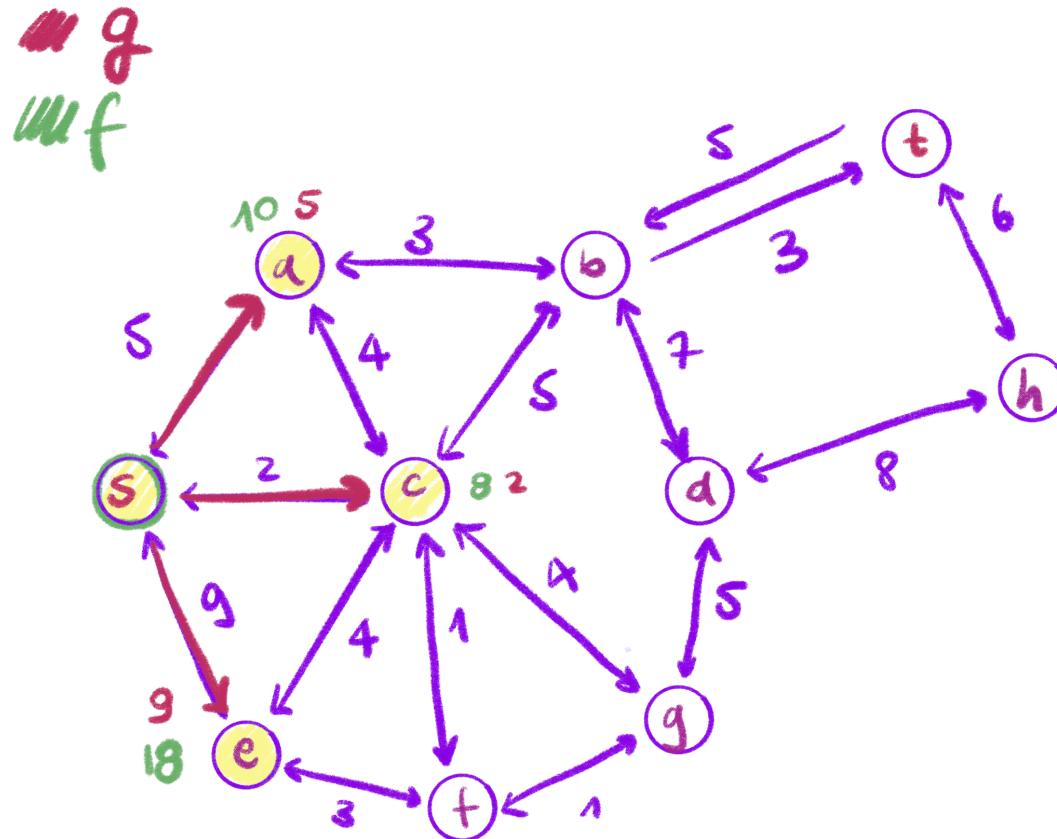
- **Temporale:**

- $O(|V|)$ resetMaze
- $O(|V| \log |V|)$ per il pop in OPEN
- $O(|E| \log |V|)$ per il push/update in OPEN considerando che ogni vertice chiuso non viene riaperto con un'euristica coerente
→ ogni vertice viene aggiunto una sola volta
- $O(|E| \log |V| * f(|V|))$ per il calcolo di $h(n,m)$, generalmente $O(1)$
- **TOTALE** $O((|V| + |E| * f(|V|)) * \log |V|) = O(N * M * \log(N * M))$

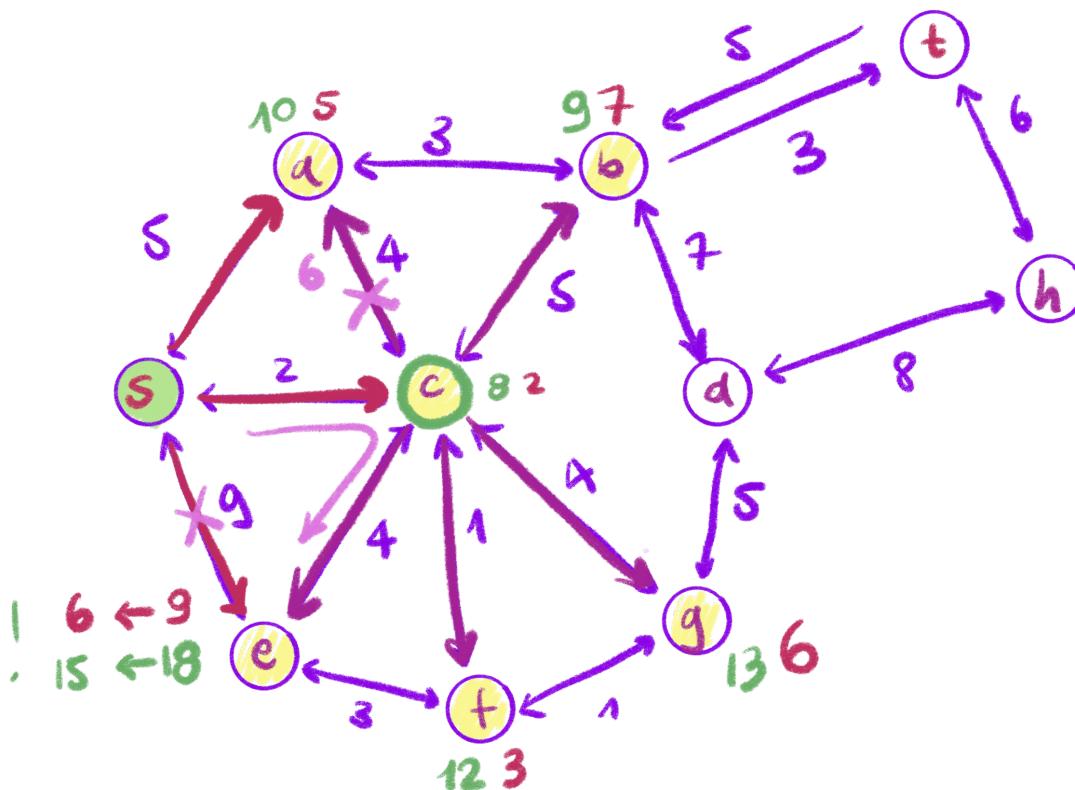
NOTA IMPORTANTE: Essendo stato applicato a un labirinto, ossia una struttura t.c. tutti i punti sono raggiungibili tra loro, non è stato necessario aggiungere l'ultimo pezzo del codice, i.e.

```
// Open set is empty but goal was never reached  
    return failure
```

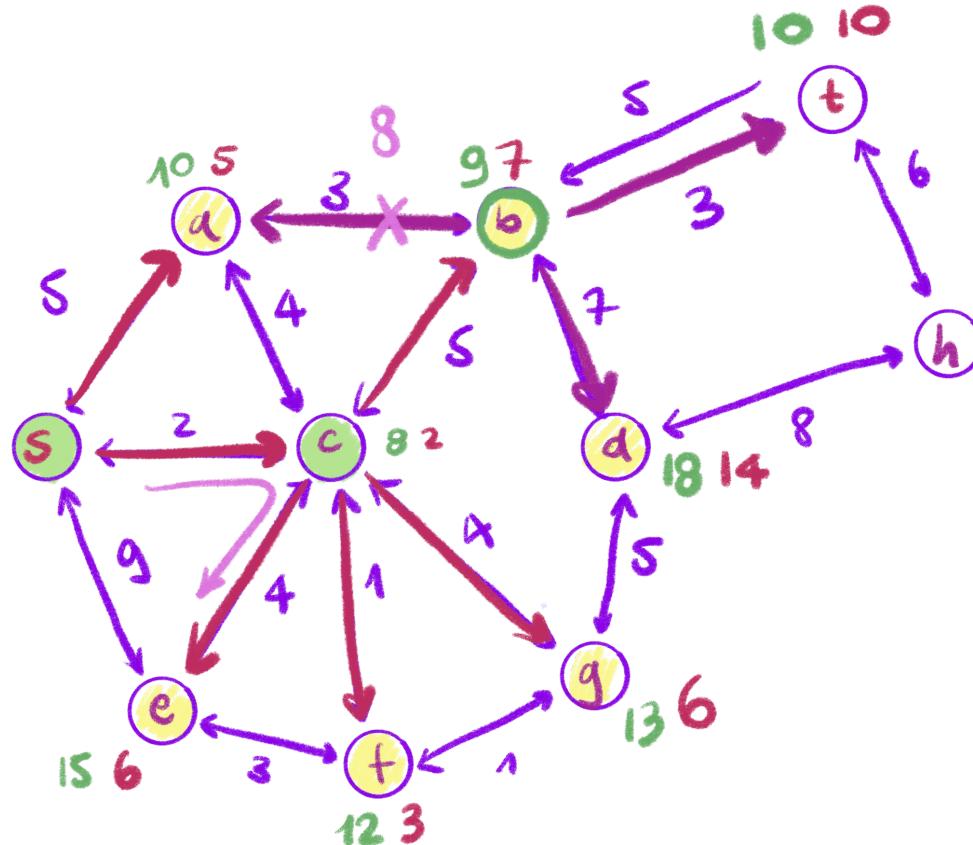
Esempio A*



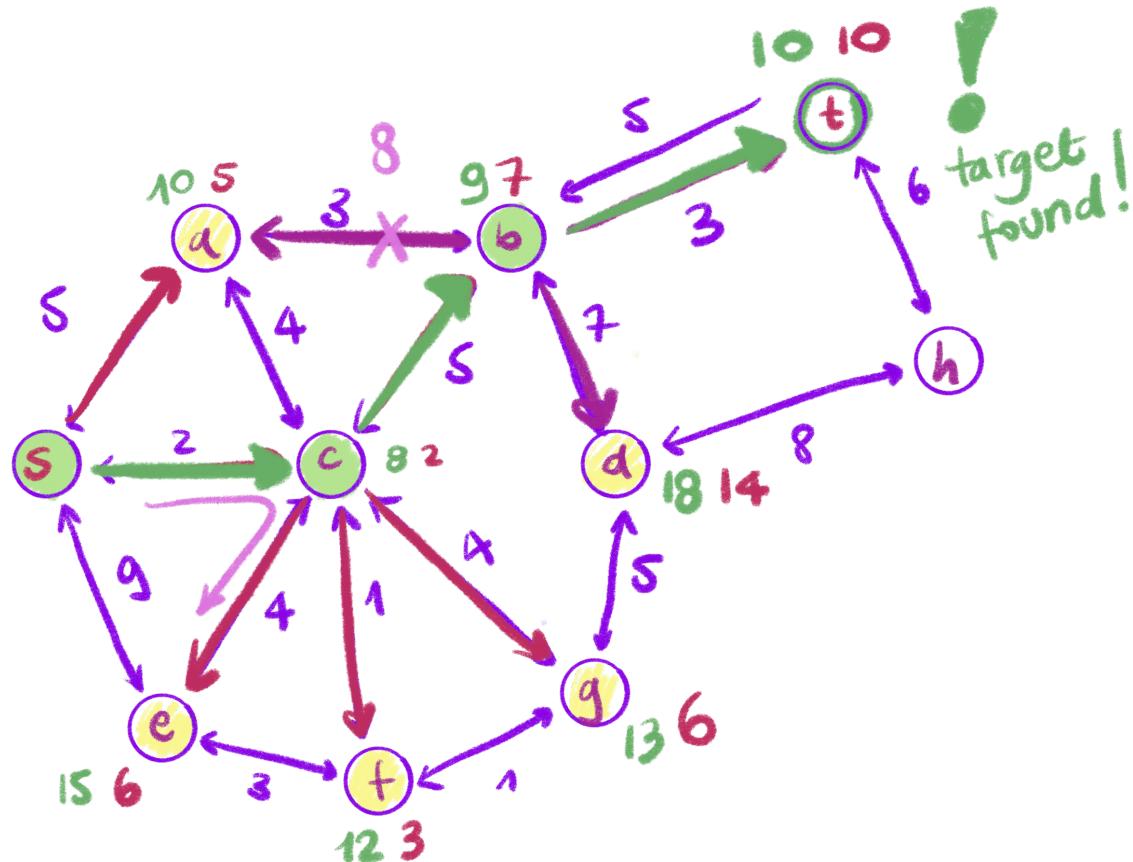
Esempio A*



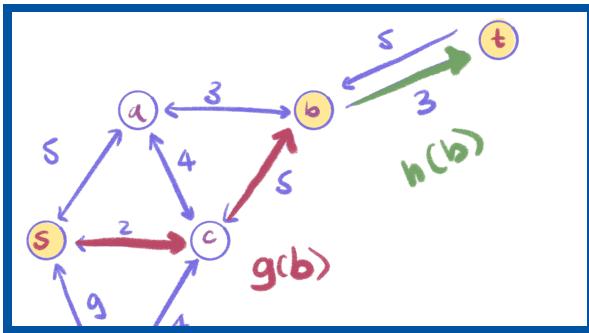
Esempio A*



Esempio A*



Definizione della funzione f



- $P^*(m, n) \equiv$ percorso ottimo (di costo minimo) tra m ed n
- $g(n) \equiv$ costo di $P^*(s,n)$
- $h(n) \equiv$ costo di $P^*(n,t)$
- $f(n) \equiv$ costo totale di $P^*(s,t)$ vincolato attraverso n ($f(s) = h(s)$)
costo di $P^*(s,t)$ non vincolato)

$$\rightarrow f(n) = g(n) + h(n)$$

Definizione della funzione di stima \tilde{f}

$$f(n) = g(n) + h(n) \rightarrow \text{stimato } \tilde{f}(n) = \tilde{g}(n) + \tilde{h}(n)$$

- $\tilde{g}(n) \equiv$ stima del costo di $P^*(s,n)$, è preso come il costo minore trovato "so far" dall'algoritmo, quindi $\tilde{g}(n) \geq g(n)$
 - $\tilde{h}(n) \equiv$ stima del costo di $P^*(n,t)$, è **estraibile dal problema** p.e. nel caso in cui si considerasse una mappa, potrebbe essere la distanza in linea d'aria
- Si dimostra che se $\tilde{h}(n) \leq h(n)$, ossia se la distanza minima è sottostimata, allora A* è ammissibile:

Dimostrazione di ammissibilità di A*

A* è ammissibile se trova il percorso ottimo (quindi di costo minore) da s a t per ogni δ grafo, i.e. ogni grafo i cui archi abbiano peso maggiore o uguale a $\delta > 0$

Lemma

Per ogni n NON CHIUSO e per ogni percorso ottimo $P^*(s,n)$, esiste n' su P^* APERTO t.c. $\tilde{g}(n') = g(n')$

Corollario

Supponendo $\tilde{h}(n) \leq h(n) \forall n$ e supponendo che A* non abbia terminato, allora per ogni percorso ottimo $P^*(s,t)$ esiste n' su P^* APERTO t.c. $\tilde{f}(n') \leq f(s)$ con $f(s)$ costo reale del percorso ottimo $P^*(s,t)$.

TEOREMA

Se $\tilde{h}(n) \leq h(n) \forall n$, allora A* è ammissibile.

Dimostrazione

P.A. A* termina in t con $\tilde{f}(t) = \tilde{g}(t) > f(s)$.

N.B. $f(s) = h(s) = g(t)$ costo $P^*(s,t)$ unconstrained

Per il corollario, prima della terminazione esiste n' APERTO su P^* t.c.
 $\tilde{f}(n') \leq f(s) < \tilde{f}(t)$. \rightarrow n' dovrebbe essere espanso prima di t \rightarrow
impossibile che A* abbia terminato

Ottimalità di A* sotto euristica coerente

Euristica coerente (consistent)

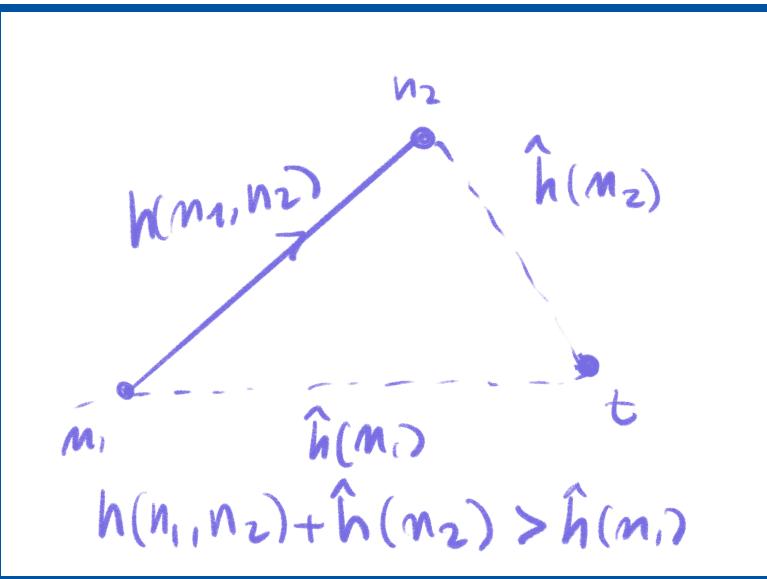
Per ogni nodo m ed n vale la diseguaglianza triangolare

$$h(m, n) + \tilde{h}(m) \geq \tilde{h}(n)$$

con $h(m, n)$ distanza lungo P^* .

Considerando la distanza in linea d'aria d

$$\begin{aligned} h(m, n) &\geq d(m, n) \rightarrow \\ h(m, n) + \tilde{h}(m) &\geq d(m, n) + \tilde{h}(m) \geq \tilde{h}(n) \end{aligned}$$



Si può dimostrare che Euristica coerente $\rightarrow A^*$ espande meno nodi rispetto a un algoritmo ammissibile A

$\rightarrow N(A^*, G_s) \leq N(A, G_s)$ dove N numero di nodi espansi.

Vale = sse A espande gli stessi nodi di A^*

Infatti per esempio A^* con $\tilde{h} = 0$ si riconduce a Dijkstra

Per dimostrarlo si nota che sotto euristica coerente se un nodo n è già stato CHIUSO \rightarrow è stato già trovato il $P^*(s,n) \rightarrow \tilde{g}(n) = g(n) \rightarrow$ n non verrà mai riaperto.

A* nei labirinti

Euristica

Come buona euristica è naturale considerare la **Manhattan distance (L_1)** in 2D:

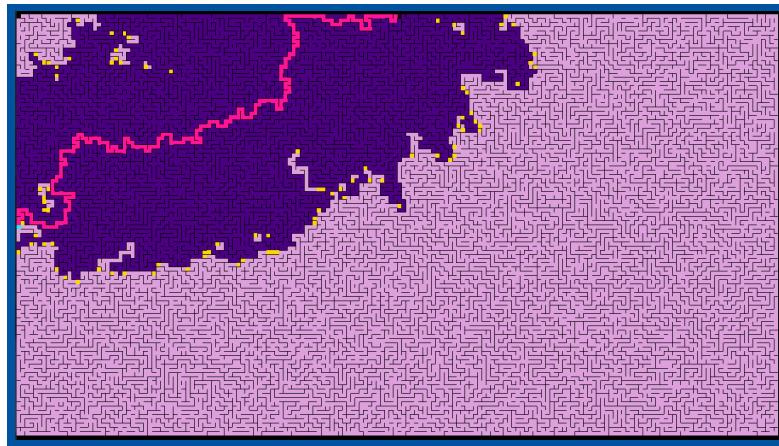
$$|x_1 - x_2| + |y_1 - y_2|$$

in quanto è la distanza che si avrebbe nel caso in cui non ci fossero muri.

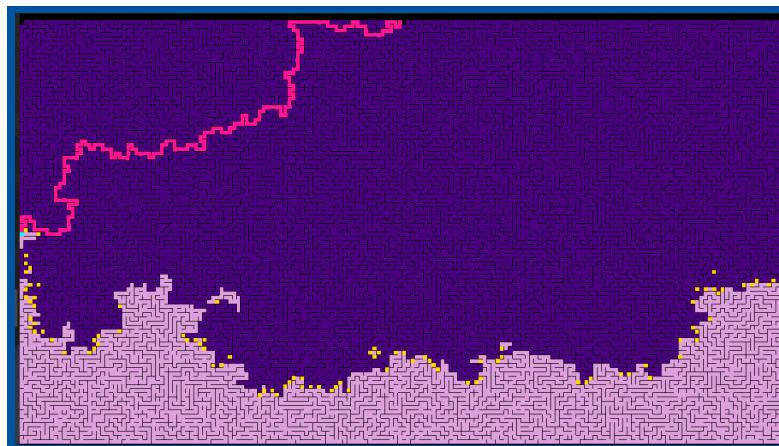
Si dimostra che questa metrica è coerente.



A* vs...



... BFS/Dijkstra



Riferimenti

[https://web.archive.org/web/20160322055823/http://
ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf](https://web.archive.org/web/20160322055823/http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf)

<http://theory.stanford.edu/~amitp/GameProgramming/>