NAVIGAZIONE DI ROBOT

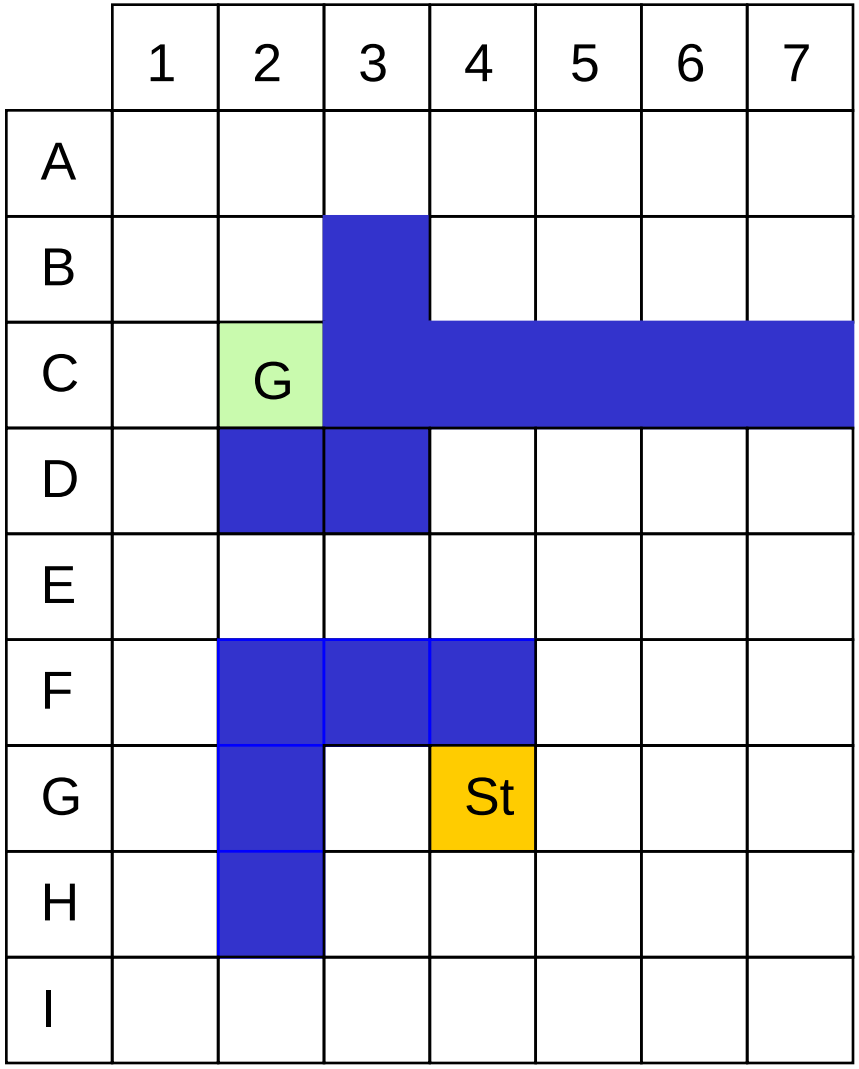
- La navigazione di robot implica la pianificazione del movimento
  - trovare il cammino più corto dalla posizione iniziale alla posizione finale

- Se la mappa è nota si può usare l'algoritmo A*
  - Per ogni nodo n calcoliamo
    - g(n): costo del cammino percorso
    - h'(n): distanza stimata dal goal
  - Espandiamo quindi i nodi in ordine crescente di

$$f(n) = g(n) + h'(n)$$

- Trova sempre il cammino più corto

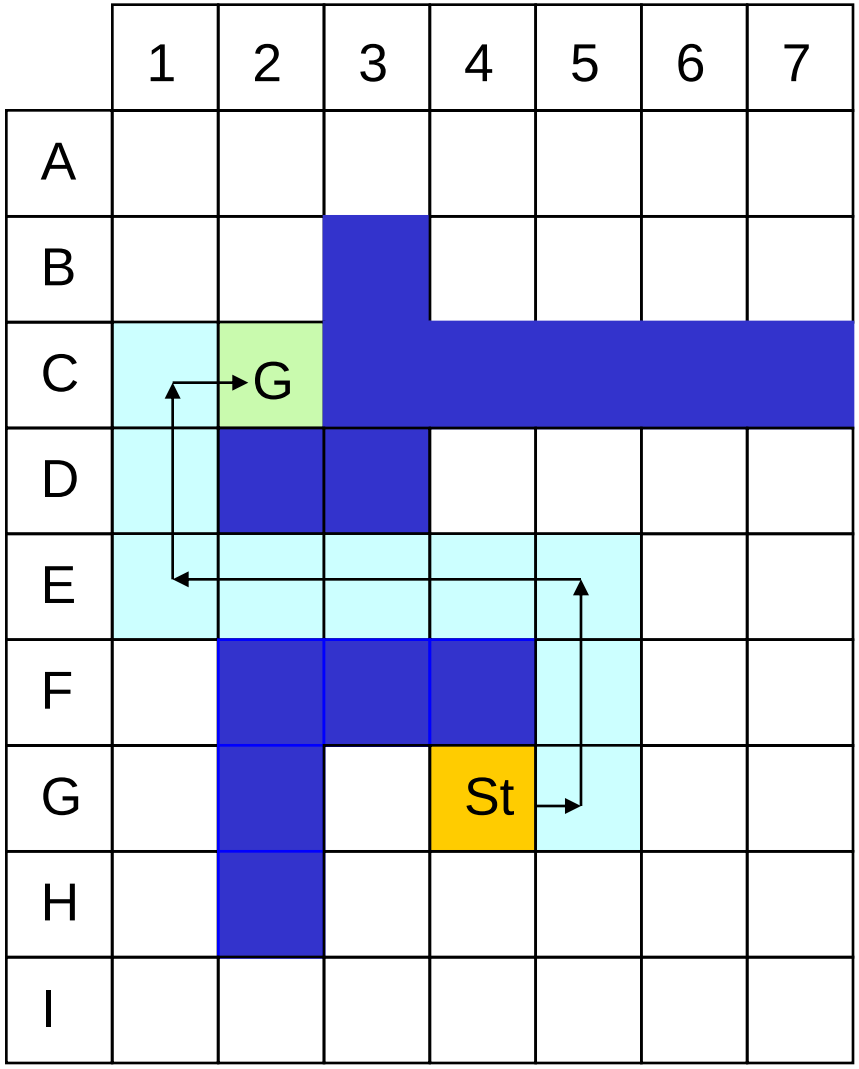
NAVIGAZIONE DI ROBOT



St:Start node

G: Goal node

NAVIGAZIONE DI ROBOT



St:Start node

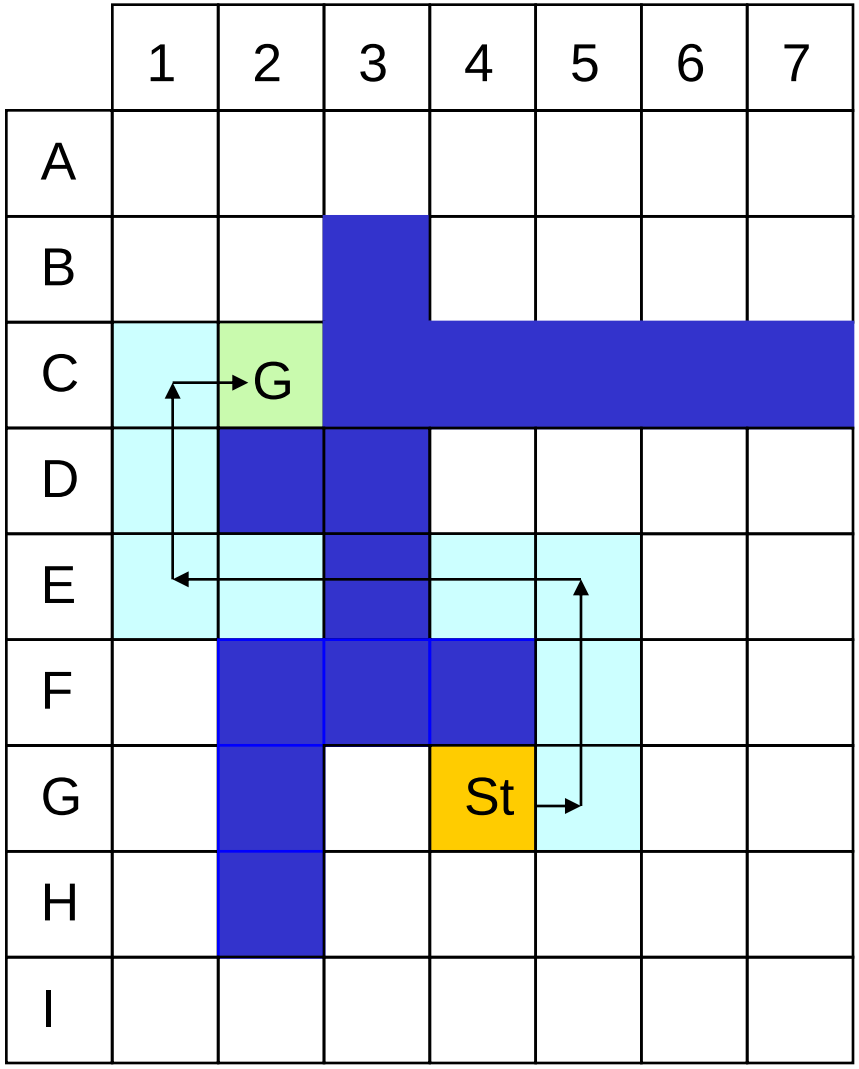
G: Goal node

RIPIANIFICAZIONE CON A*

- Se il dominio è parzialmente noto, si pianifica tramite A*, e in caso di cambiamenti del mondo, è necessario **ripianificare**.

- Tramite A* si deve ripianificare da zero:
  - inefficiente su grandi domini in cui i cambiamenti avvengono di frequente
  - molte risultati della precedente pianificazione si possono riusare.

NAVIGAZIONE DI ROBOT

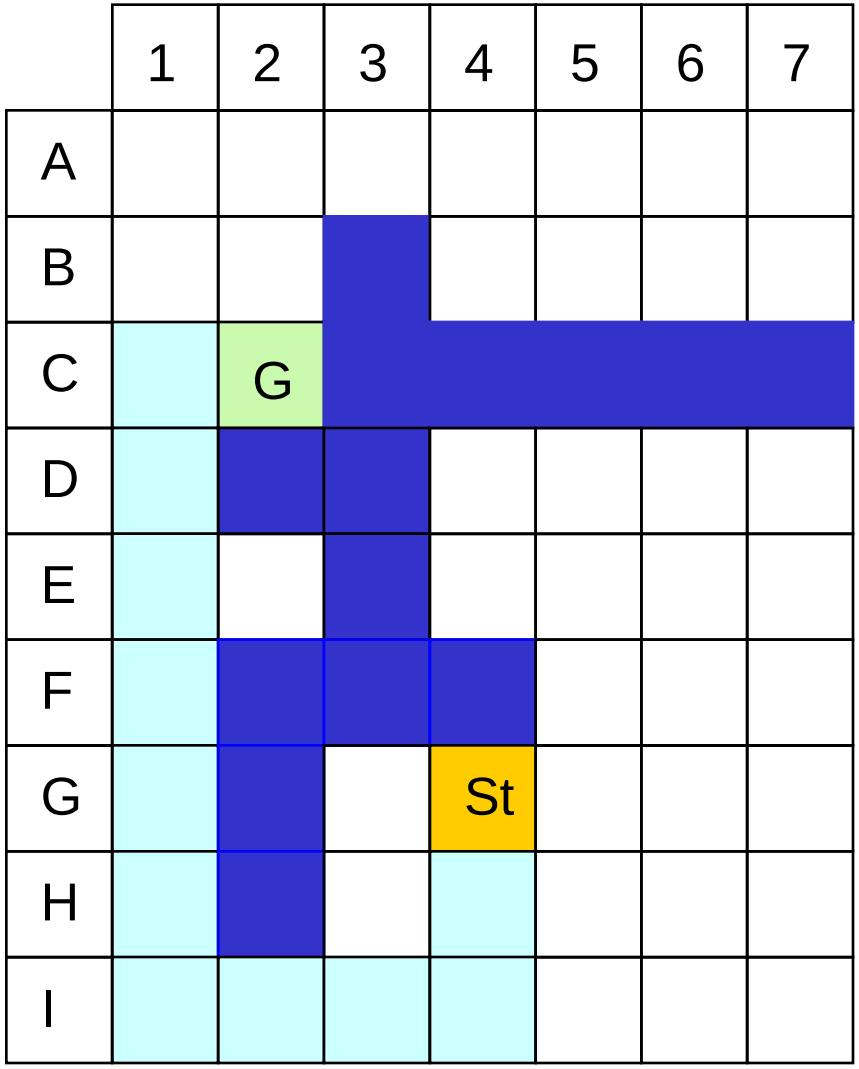


St:Start node

G: Goal node

PIANO NON PIU' VALIDO

NAVIGAZIONE DI ROBOT



St:Start node

G: Goal node

NUOVO PIANO RICALCOLATO DA ZERO

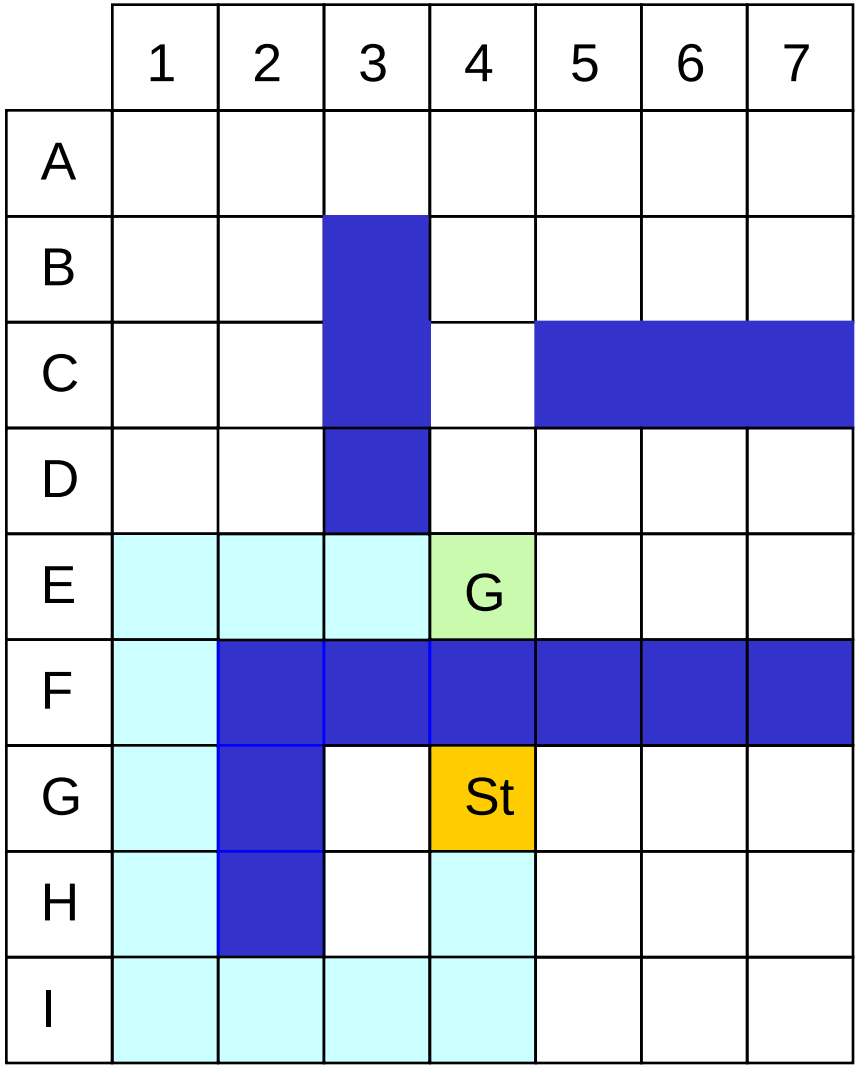
LIFELONG PLANNING A*

- Lifelong planning A* ripianifica più efficientemente di A* perché ri-utulizza parti del piano precedente

- Perché riutilizzare parti del piano
  - Perché i cambiamenti sono in genere piccoli
  - Per migliorare l'efficienza

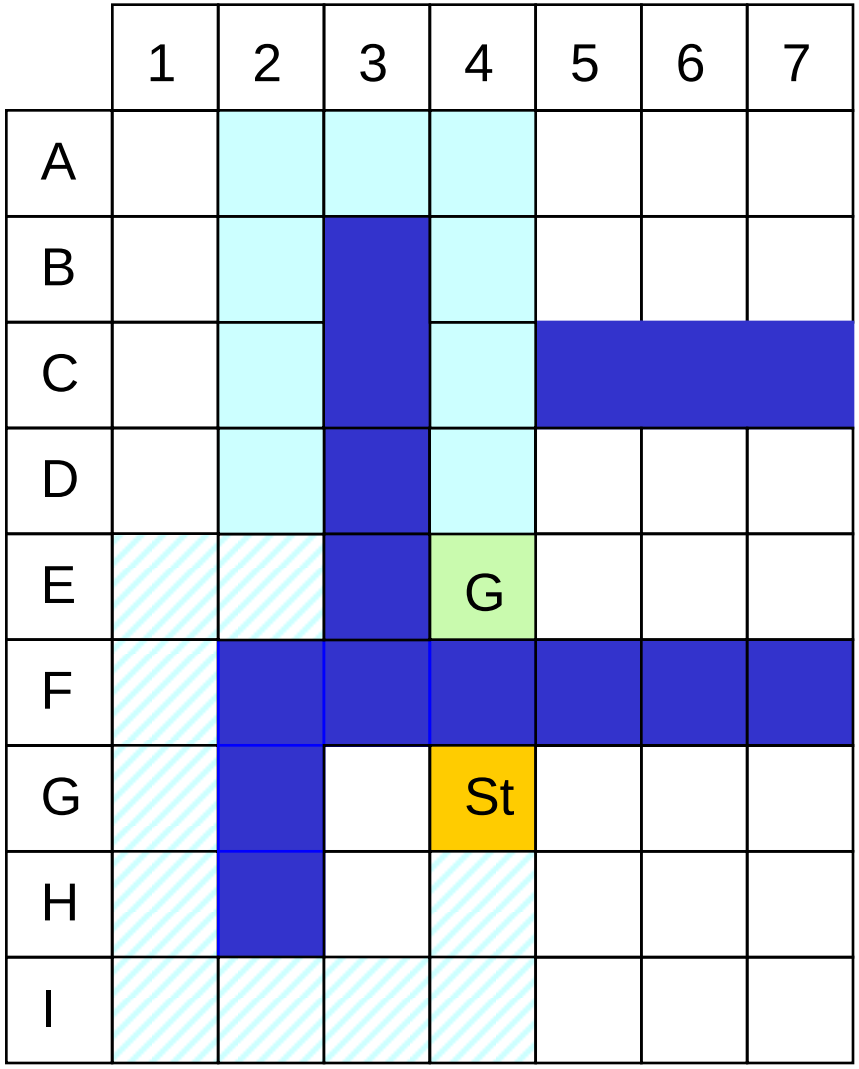
- Quali parti del piano si possono riutilizzare?
  - quelle che non sono modificate

LIFELONG PLANNING A*



Piano di partenza

LIFELONG PLANNING A*



Piano di partenza

Riutilizzo 10 mosse

LIFELONG PLANNING A*: algoritmo

Variabili:

LPA* mantiene una distanza dallo start g(n) come shortest path dallo start al nodo n

$$g(n) = \begin{cases} 0 & \text{if } n = start \\ \min_{n' \in pred(n)} (g(n) + c(n',n)) & \text{otherwise} \end{cases}$$

LPA* mantiene una seconda distanza dallo start rhs(n) con un passo di look ahead

$$rhs(n) = \begin{cases} 0 & \text{if } n = start \\ \min_{n' \in pred(n)} (g(n') + c(n',n)) & \text{otherwise} \end{cases}$$

LIFELONG PLANNING A*: algoritmo

procedure CalculateKey(s)
    return [min(g(s); rhs(s)) + h(s); min(g(s); rhs(s))];

procedure Initialize()
U = ∅
for all s ∈ S rhs(s) = g(s) = ∞;
rhs(s_{start}) = 0;
U.Insert(s_{start}; [h(s_{start}); 0]);

procedure UpdateVertex(u)
if (u ≠ s_{start}) rhs(u) = $\min_{s' \in pred(u)}$ (g(s') + c(s'; u));
if (u ∈ U) U.Remove(u);
if (g(u) ≠ rhs(u)) U.Insert(u; CalculateKey(u));

LIFELONG PLANNING A*

```
procedure CalculateKey(s)
        return [min(g(s); rhs(s)) + h(s); min(g(s); rhs(s))];
procedure Initialize()
f02g U = ;;
f03g for all s 2 S rhs(s) = g(s) = 1;
f04g rhs(sstart) = 0;
f05g U.Insert(sstart; [h(sstart); 0]);
procedure UpdateVertex(u)
f06g if (u 6= sstart) rhs(u) = mins02pred(u)(g(s0) + c(s0; u));
f07g if (u 2 U) U.Remove(u);
f08g if (g(u) 6= rhs(u)) U.Insert(u; CalculateKey(u));
procedure ComputeShortestPath()
f09g while (U.TopKey() _<CalculateKey(sgoal) OR rhs(sgoal) 6= g(sgoal))
f10g u = U.Pop();
f11g if (g(u) > rhs(u))
f12g g(u) = rhs(u);
f13g for all s 2 succ(u) UpdateVertex(s);
f14g else
f15g g(u) = 1;
f16g for all s 2 succ(u) [ fug UpdateVertex(s);
procedure Main()
f17g Initialize();
f18g forever
f19g ComputeShortestPath();
f20g Wait for changes in edge costs;
f21g for all directed edges (u; v) with changed edge costs
f22g Update the edge cost c(u; v);
f23g UpdateVertex(v);
```

D* LITE

- I robot si muovono su una mappa (grafo) parzialmente nota
- Quando viene rilevato un nuovo ostacolo i pesi sugli archi del grafo cambiano.
- D* lite