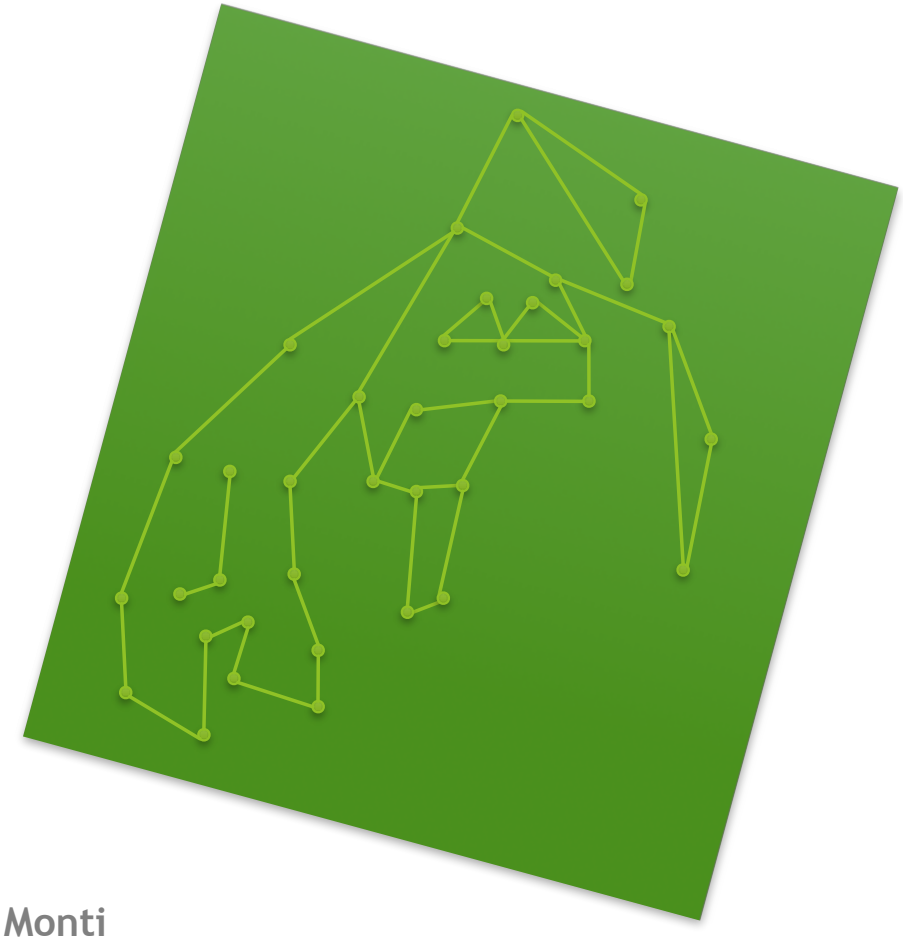
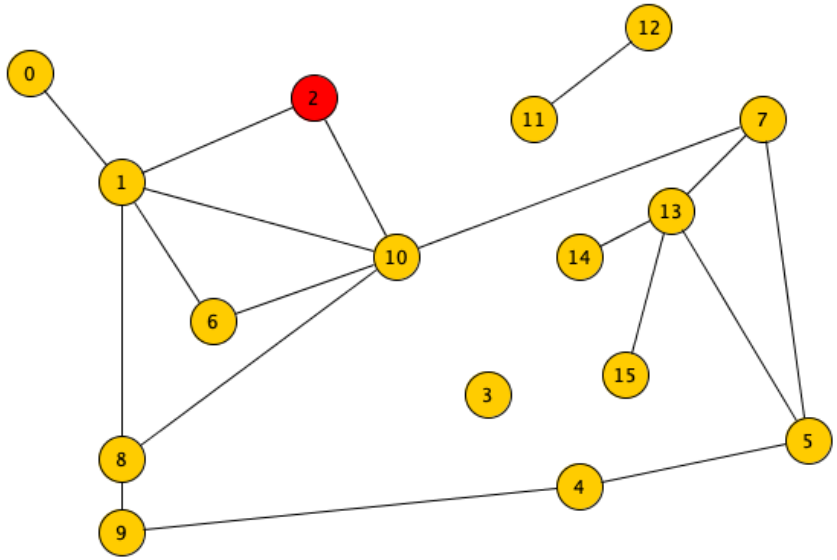
Attività di Alternanza Scuola-Lavoro
Roma, 8 Marzo 2022

Il problema dei cammini minimi



Angelo Monti
Professore Associato del Dipartimento di Informatica
Sapienza Università di Roma

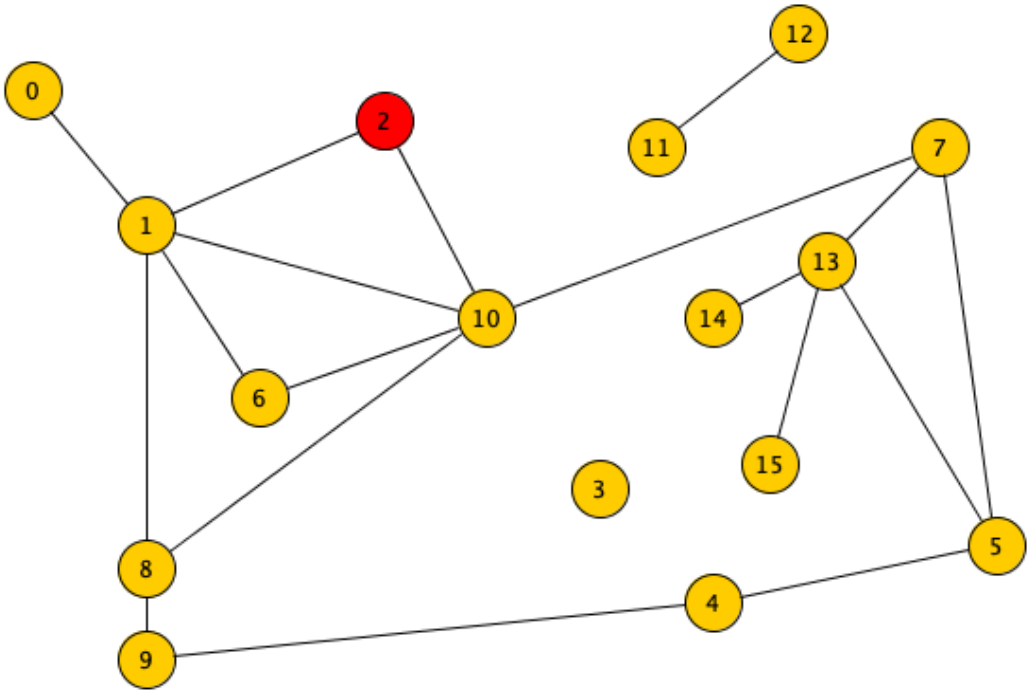
Il calcolo dei cammini minimi in G da un dato nodo sorgente



Tra due nodi in G possono esserci più cammini o nessuno

- tra il nodo 2 ed il nodo 11 non c'è alcun cammino
- tra il nodo 2 ed il nodo 5 ci sono diversi cammini:
  - 2—>10—>6—>1—>8—>9—>4—5
  - 2—>1—>8—>9—>4—>5
  - 2—>10—>7—>13—>5
  - 2—>10—>7—>5

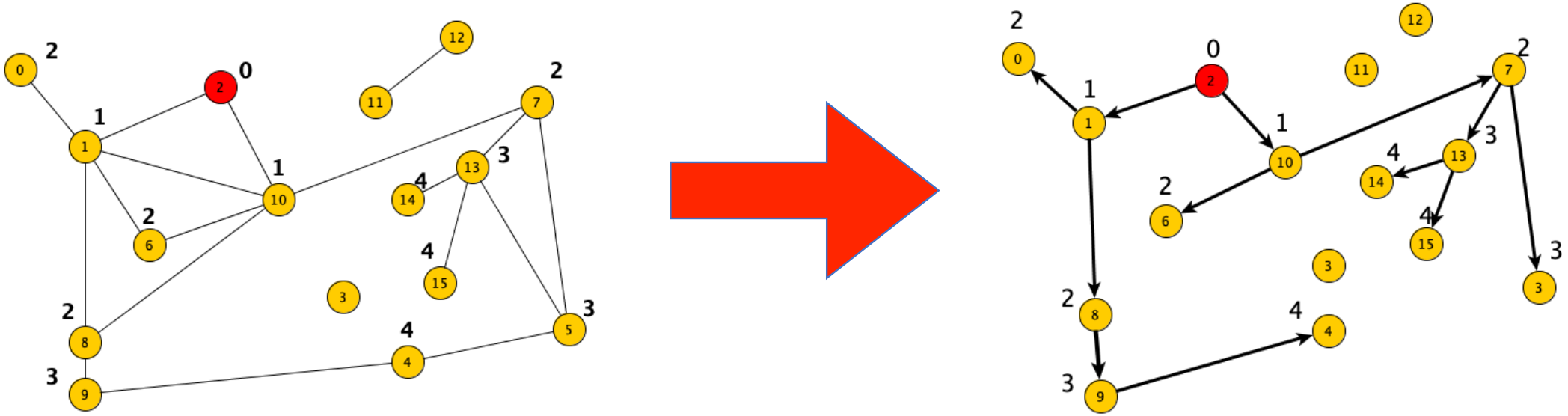
- tra il nodo 2 ed il nodo 5 ci sono diversi cammini:
  - 2—>10—>6—>1—>8—>9—>4—5
  - 2—>1—>8—>9—>4—>5
  - 2—>10—>7—>13—>5
  - 2—>10—>7—>5



Dati due nodi a e b di un grafo G, definiamo **cammino minimo** di a da b il cammino che porta da a a b attraversando il numero minimo di archi.

Dati due nodi a e b di un grafo G definiamo **distanza** di a da b il numero di archi del cammino minimo tra a e b

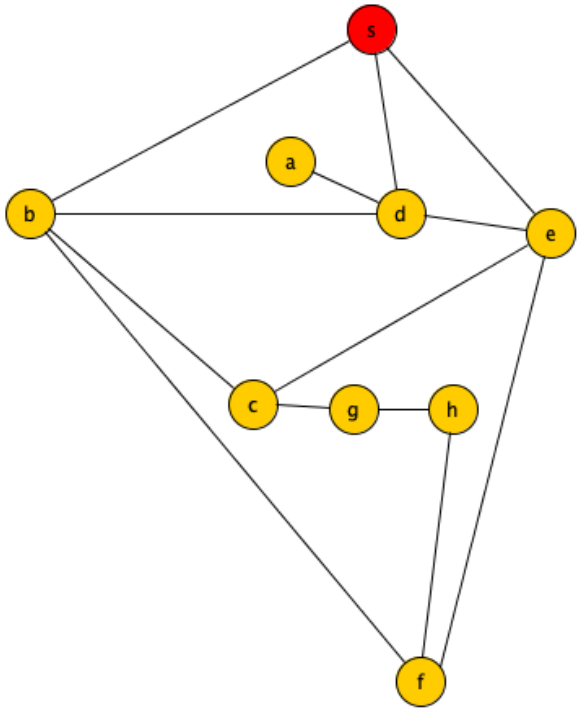
Problema: dato un grafo G ed un suo nodo x vogliamo calcolare i cammini minimi che portano da x a tutti gli altri nodi raggiungibili

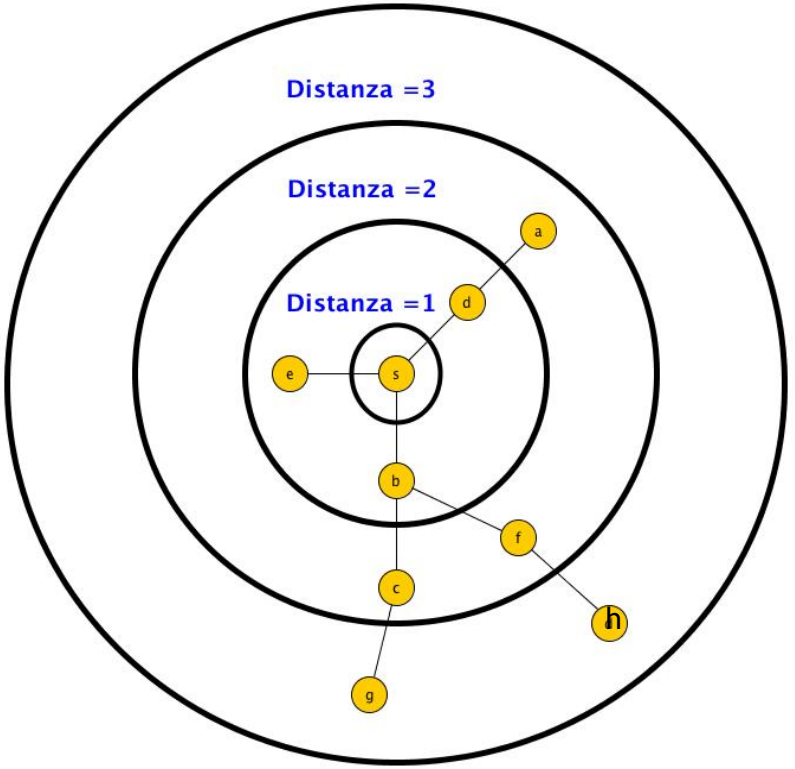


I cammini minimi dal nodo sorgente 2 agli altri nodi del grafo

Soluzione al problema:

- proponiamo quella che è nota come visita in ampiezza del grafo (in acronimo BFS, Breadth First Search).
- Con questa visita i nodi del grafo sono esplorati partendo da quelli a distanza 1 dalla sorgente s. Poi si visitano quelli a distanza 2 e così via.
- L'algoritmo **visita** tutti i nodi a distanza k prima di passare a quelli a distanza $k+1$.





- Per effettuare questo tipo di visita manteniamo in una coda i nodi visitati i cui adiacenti non sono stati ancora esaminati.
- Ad ogni passo, preleviamo il primo nodo dalla coda, esaminiamo i suoi adiacenti e se scopriamo un nuovo nodo lo visitiamo e lo aggiungiamo alla coda.

Algoritmo per la ricerca in ampiezza (BFS)

1. **Metti in coda il nodo di partenza**
2. **Togli dalla coda un nodo x (nella prima iterazione è il nodo sorgente) ed esploralo alla ricerca di nuovi nodi da visitare**
   A. **metti in coda tutti i vicini del nodo x non ancora visitati**
   B. **se la coda è vuota la visita termina**
   C. **altrimenti ripeti il passo 2**

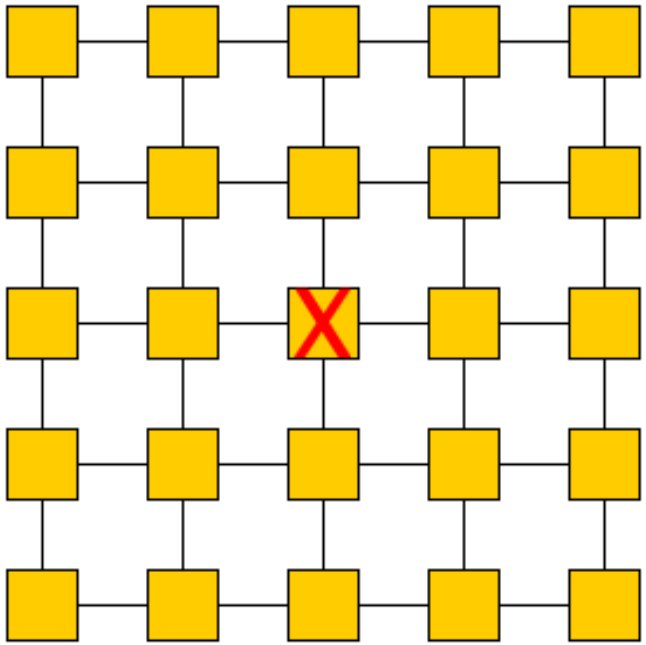
Algoritmo BFS

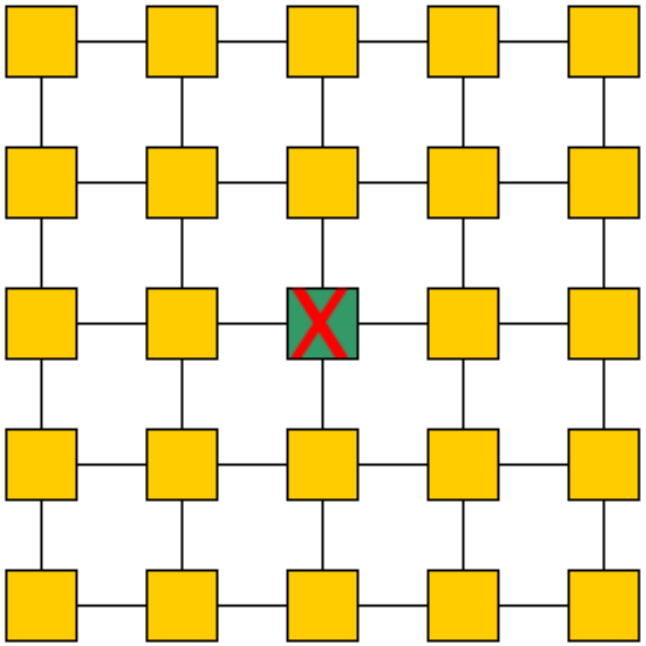
1. Metti in coda il nodo di partenza
2. Togli dalla coda un nodo x (nella prima iterazione è il nodo sorgente) ed esploralo alla ricerca di nuovi nodi da visitare
    1. metti in coda tutti i vicini del nodo x non ancora visitati
    2. se la coda è vuota termina
    3. altrimenti ripeti il passo 2

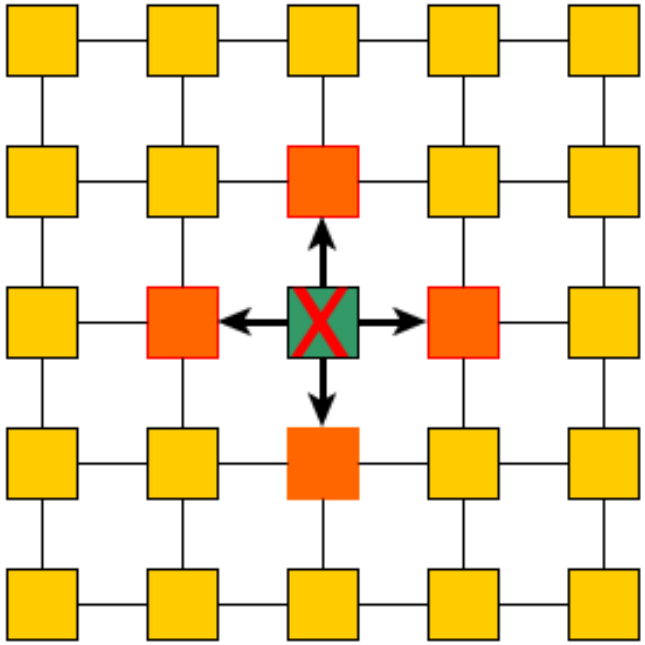
Nota che nel corso della BFS i nodi possono trovarsi in uno di tre stati:

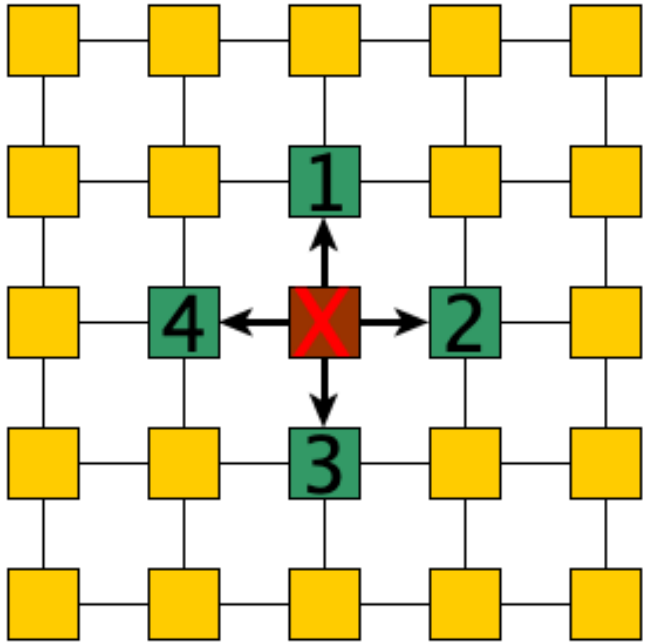
- non esplorati
- in corso di visita (sono in attesa nella coda)
- visitati (sono stati estratti dalla coda)

Illustreremo il procedimento su di una griglia (vale a dire un grafo in cui ogni nodo (tranne quelli ai bordi) hanno esattamente 4 vicini (nord, est, sud e ovest).

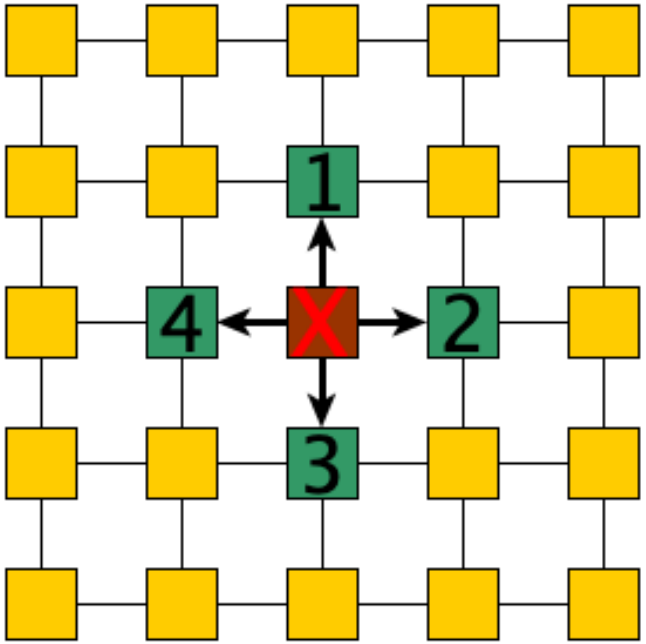


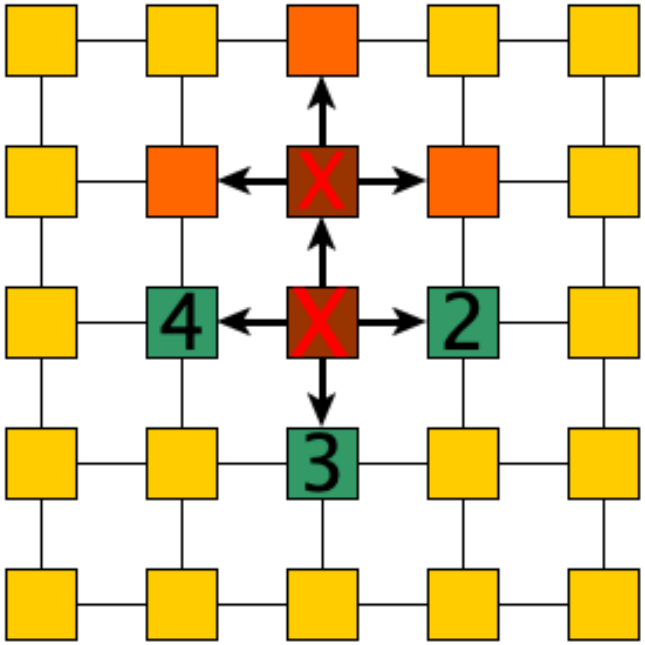


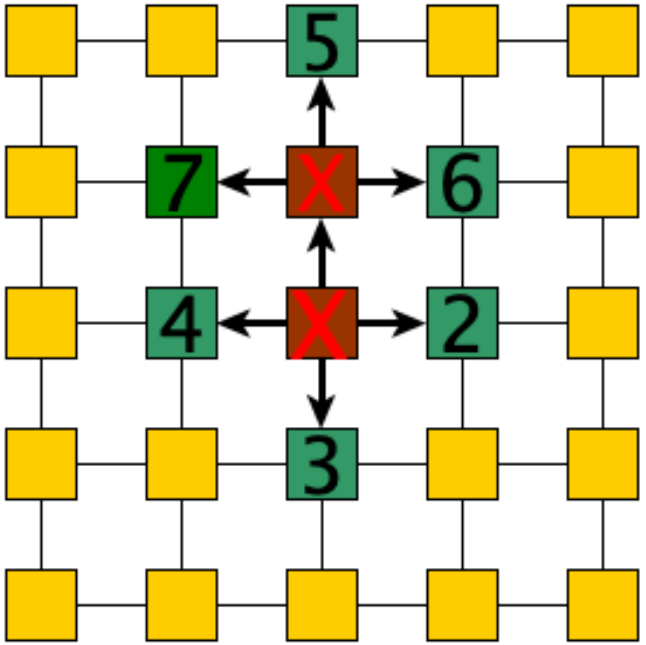




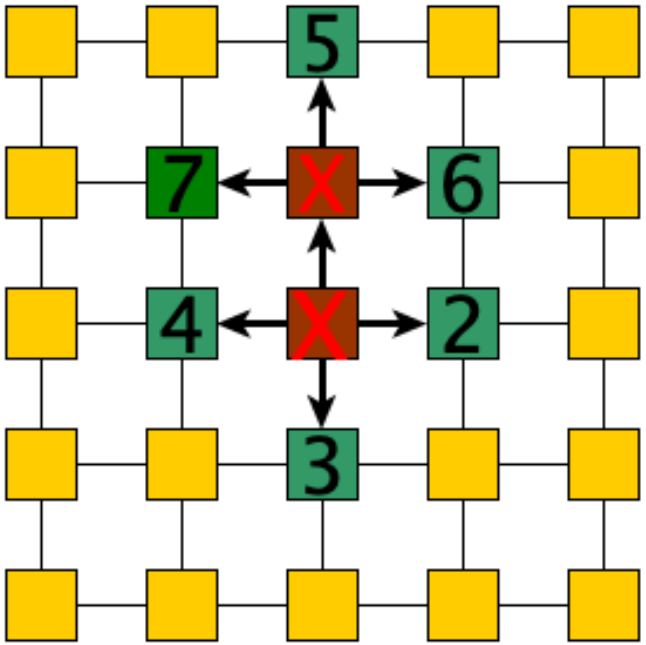
Prof. **Angelo Monti**

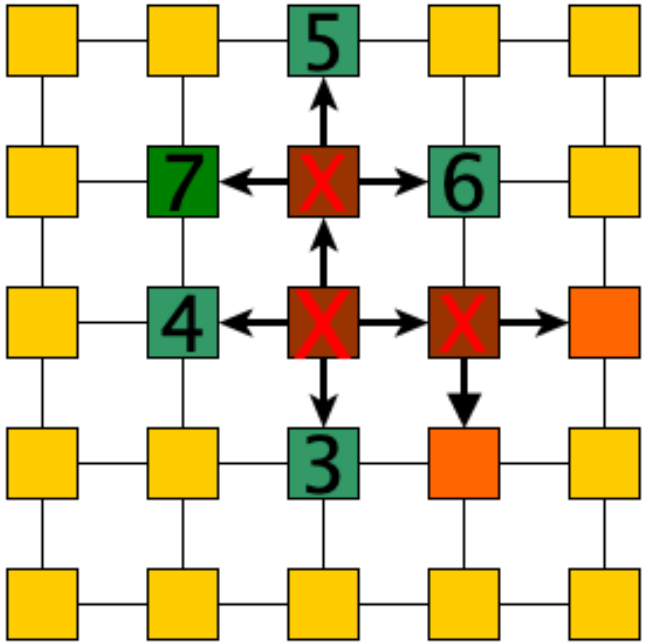


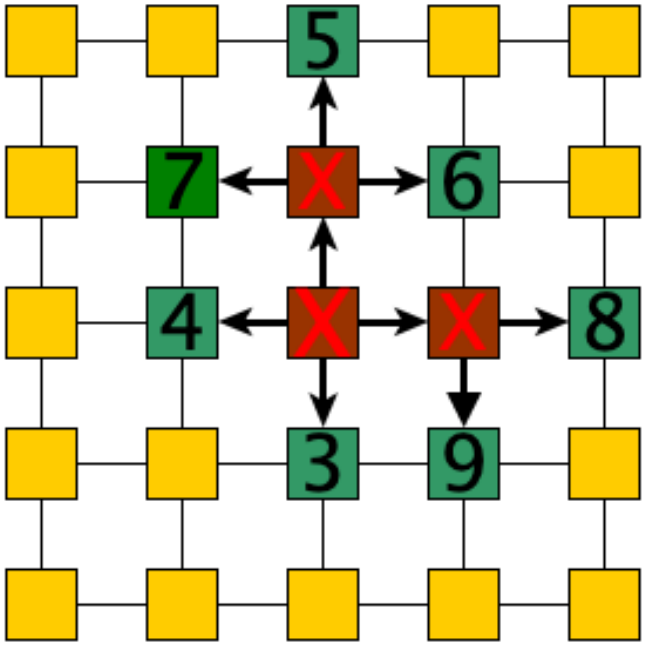




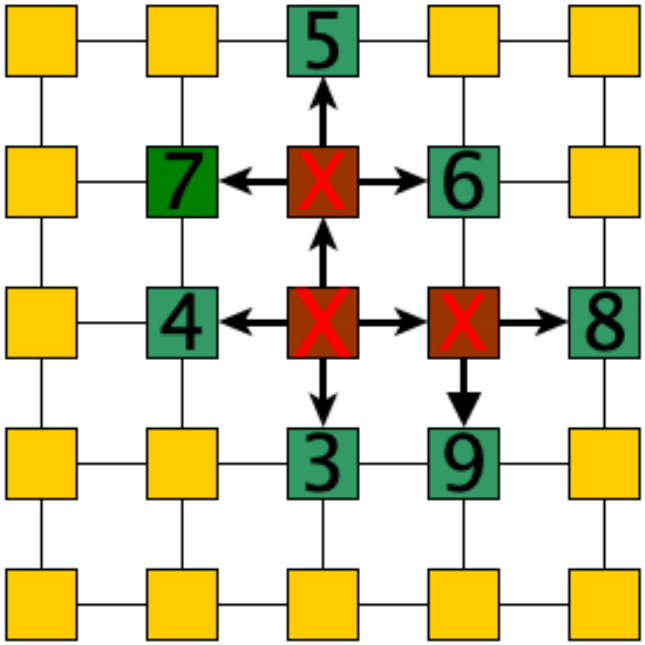
Prof. **Angelo Monti**



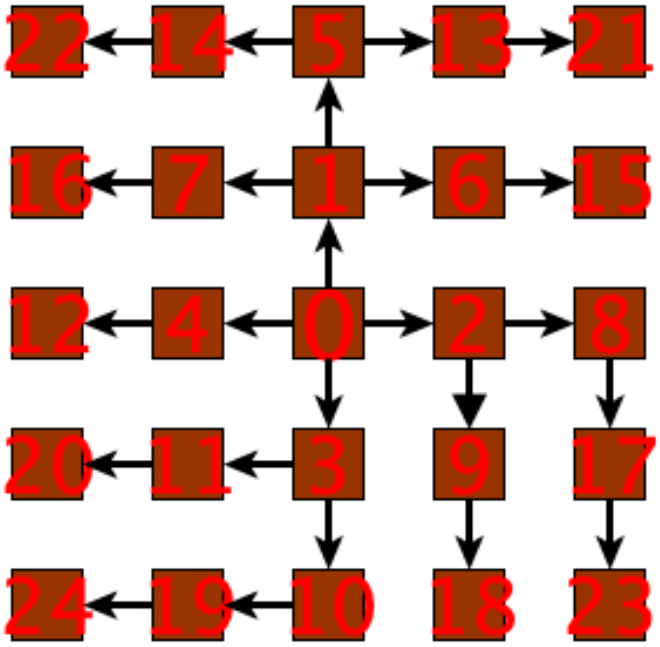




Prof. **Angelo Monti**

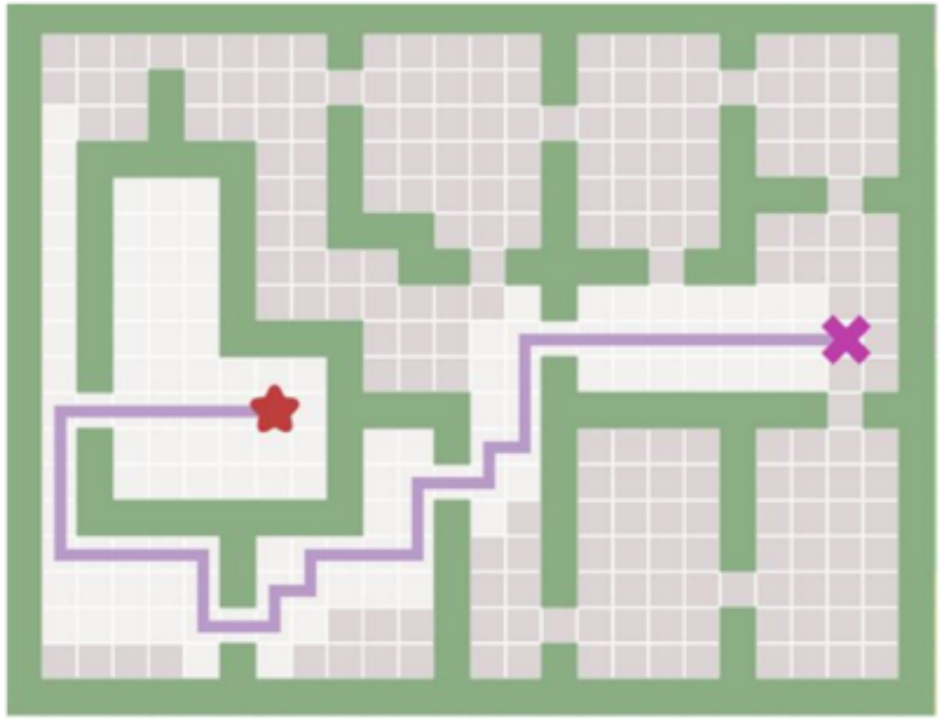

5
7
X
6
4
X
X
8
3
9

.....

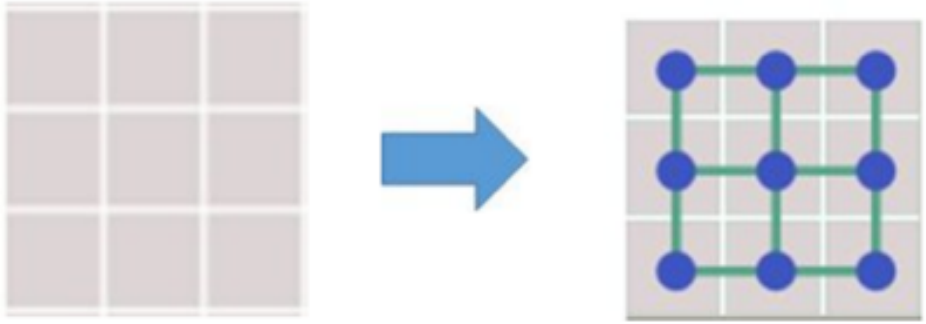

22
14
5
13
21
16
7
1
6
15
12
4
0
2
8
20
11
3
9
17
24
19
10
18
23

Problema 2: trovare il cammino minimo che permette di uscire da un labirinto

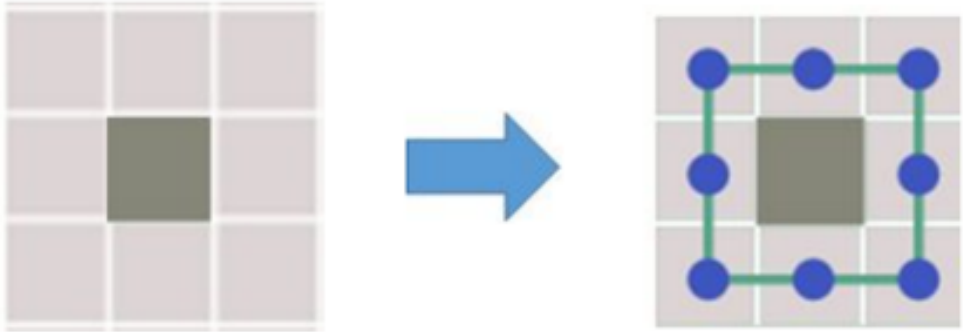
Più in generale date due posizioni ★ e ✖ nel piano vogliamo il cammino minimo che porta da ★ a ✖



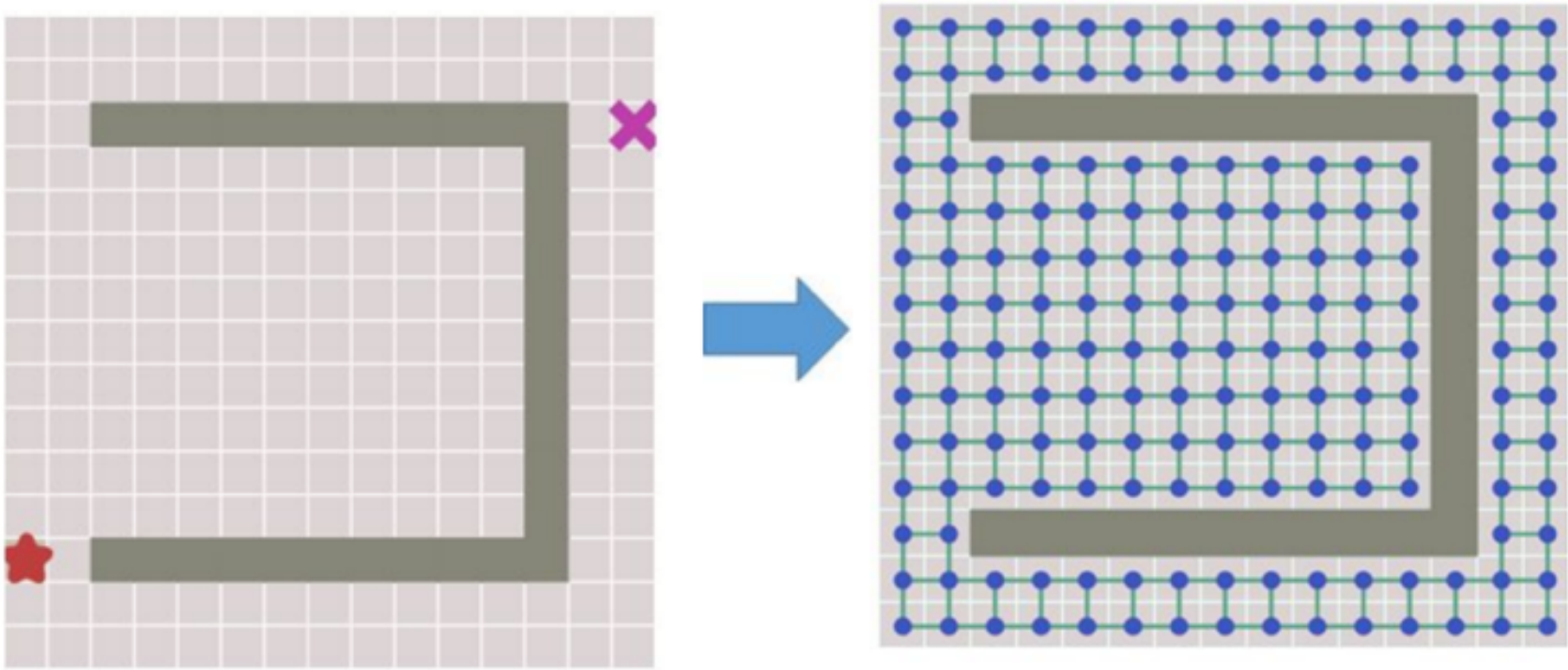
- Ogni cella del labirinto può essere rappresentata con un nodo



- I muri non sono connessi a nulla



- Un esempio di labirinto:

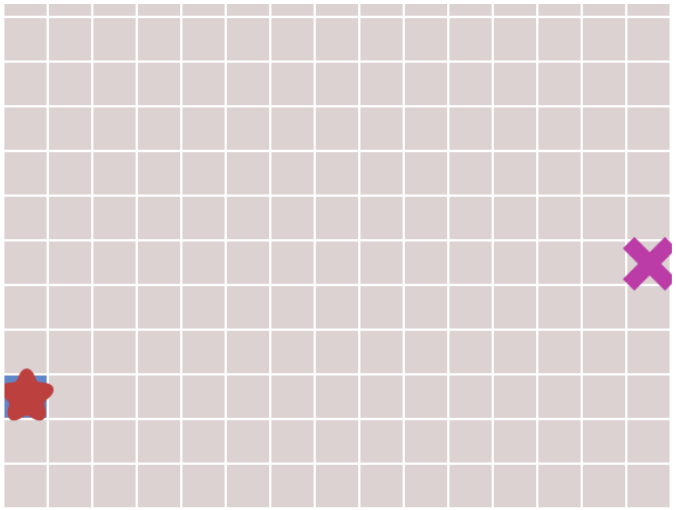


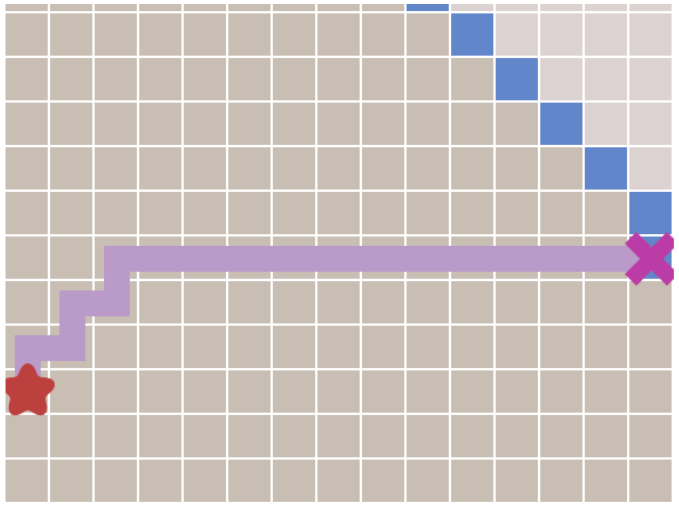
Soluzione: posso usare la BFS che usa come nodo di partenza ★ e termina la visita appena giunge a visitare il nodo ✕

Algoritmo BFS modificato

1. **Metti in coda il nodo di partenza ★**
2. **Togli dalla coda un nodo x ed esploralo alla ricerca di nuovi nodi da visitare**
   - **A. metti in coda tutti i vicini del nodo x non ancora visitati**
   - **B. se tra questi nodi c'è il nodo ✕ termina.**
   - **C. se la coda è vuota termina (in questo caso il nodo ✕ non è raggiungibile)**
   - **D. altrimenti ripeti il passo 2**

1. **Metti in coda il nodo di partenza ★**
2. **Togli dalla coda un nodo x ed esploralo alla ricerca di nuovi nodi da visitare**
    1. **metti in coda tutti i vicini del nodo x non ancora visitati**
    2. **se tra questi nodi c'è il nodo ✕ termina.**
    3. **se la coda è vuota termina**
    4. **altrimenti ripeti il passo 2**





celle non visitate
celle visitate
celle nella coda

Non visita tutte le celle ma prima di raggiungere la cella che interessa calcola "inutilmente"il cammino di molte altre celle che non hanno a che fare col cammino minimo da ★ a ✕

Come velocizzare la BFS per trovare prima il cammino minimo da ★ a ✕ ?

BFT è un algoritmo utilissimo per tante applicazioni ma esplora lo spazio di ricerca ugualmente in tutte le direzioni

IDEA:

- nell'estrarre i nodi dalla coda diamo la precedenza a quelli che sono "*più vicini*" al nodo destinazione ✕.
- Tra i vari nodi presenti nella coda scegliamo di esplorare quello che ha ***distanza stimata*** minima dalla nostra destinazione ✕

1. **Metti in coda il nodo di partenza ★**
2. **estrai dalla coda il nodo x che ha distanza stimata da ★ ed esploralo alla ricerca di nuovi nodi da visitare**
    1. **metti in coda tutti i vicini del nodo x non ancora visitati**
    2. **se tra questi nodi c'è il nodo ✕ termina.**
    3. **se la coda è vuota termina**
    4. **altrimenti ripeti il passo 2**

Quest'algoritmo è noto come Greedy Best First Search

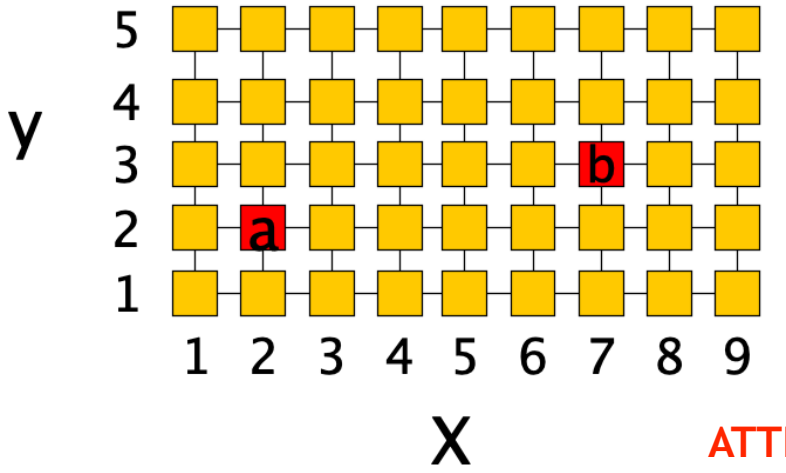
Come stimo la distanza tra due celle (x, y) e (x', y') sulla griglia?

poiché nella griglia posso muovermi solo in orizzontale e/o verticale la distanza da percorrere può essere stimata come

$$|x - x'| + |y - y'|$$

dove $|x - x'|$ è il numero di celle che devo attraversare in orizzontale

mentre $|y - y'|$ è il numero di celle che debbo attraversare in verticale

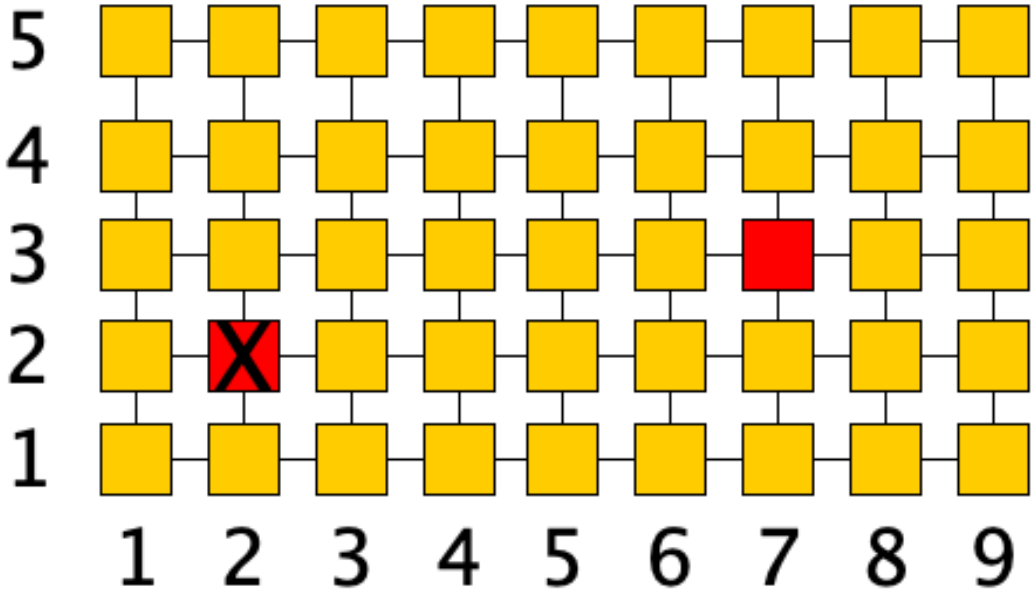


a=(2,2)
b=(7,3)

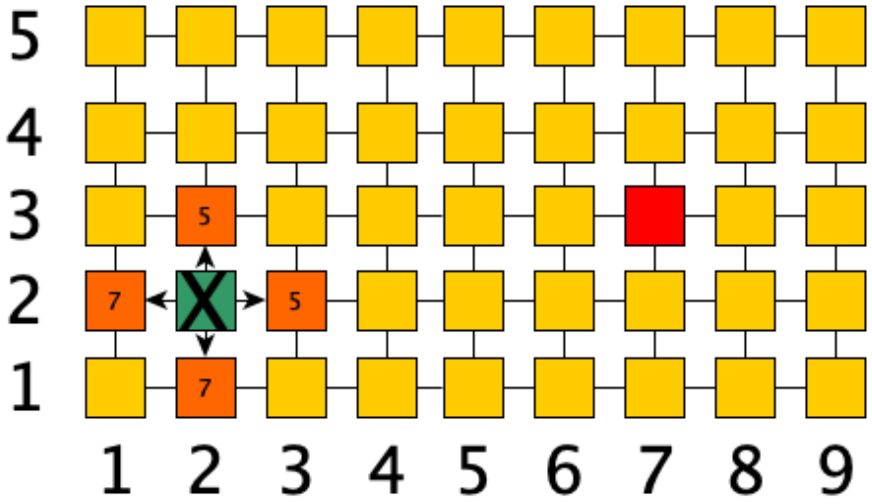
distanza(a,b)=|2-7|+|2-3|=5+1=6

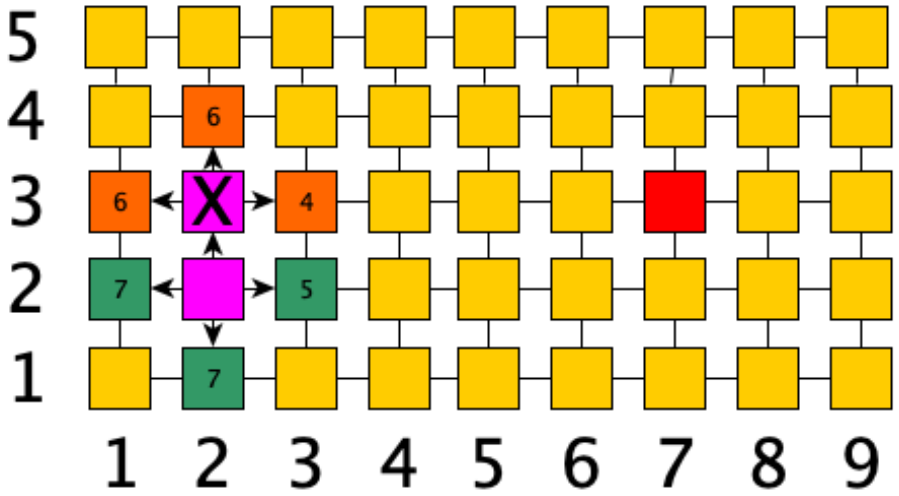
ATTENZIONE: la distanza calcolata è solo una stima in quanto non si tiene conto dell'eventuale presenza di un ostacolo sul percorso da a e b

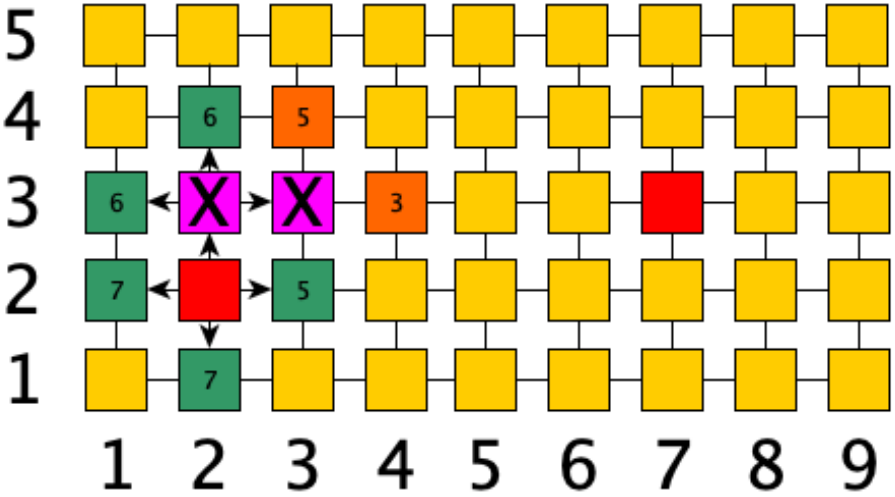
Esempio di applicazione:

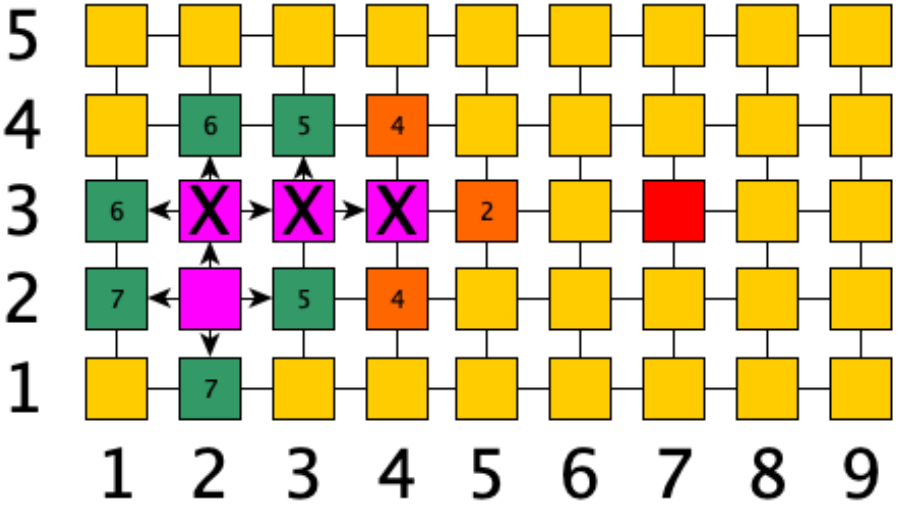


dovrebbe trovare un cammino di costo 6

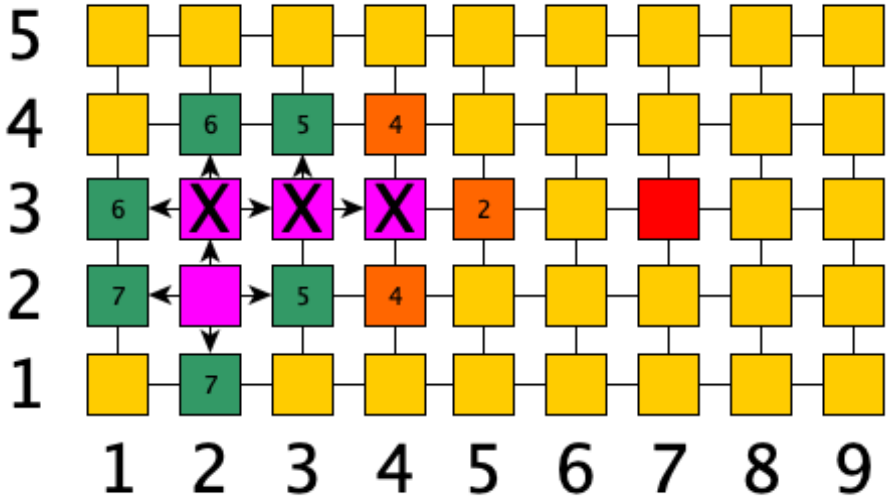


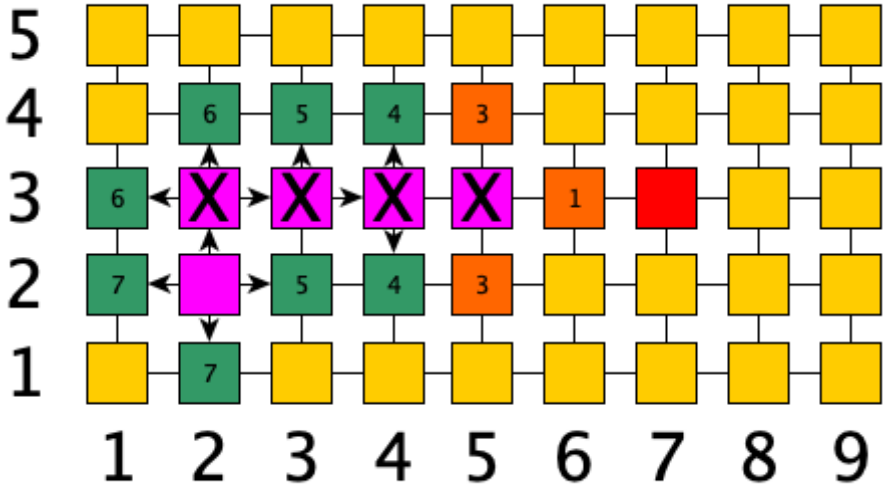


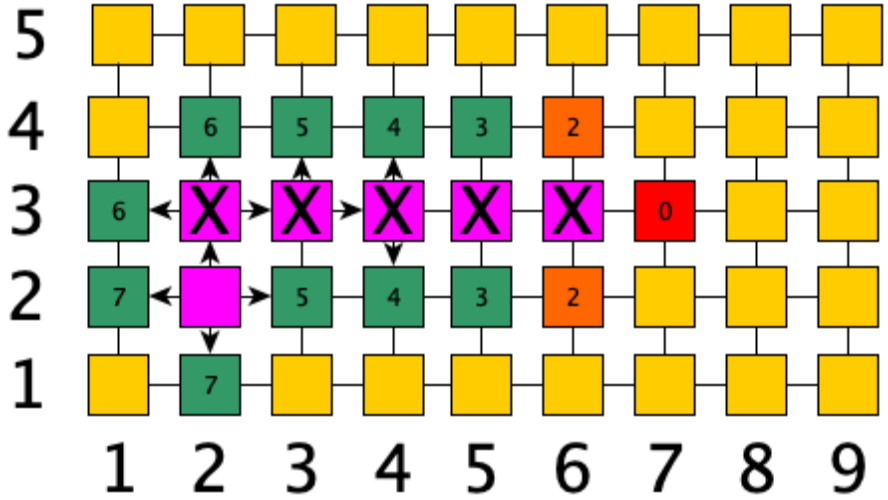


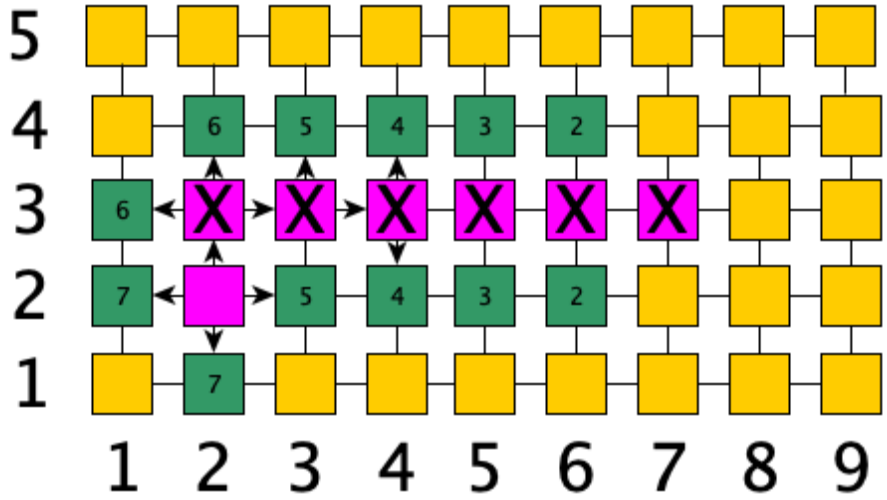


Prof. **Angelo Monti**



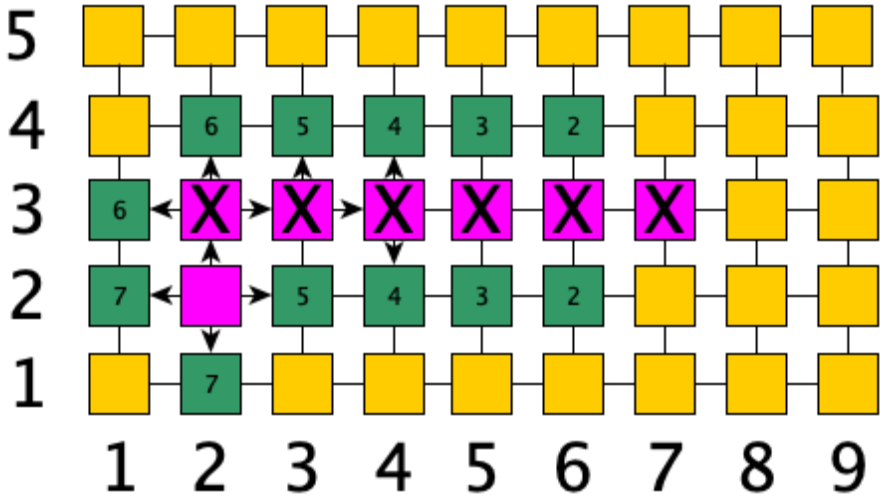




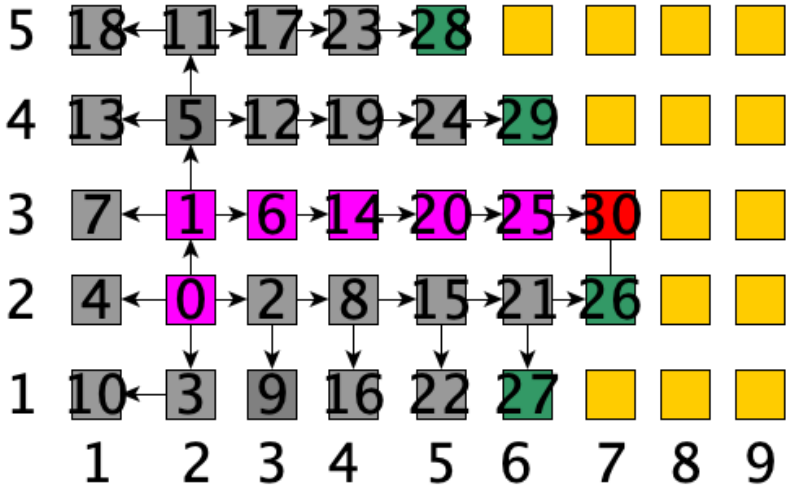


Prof. **Angelo Monti**

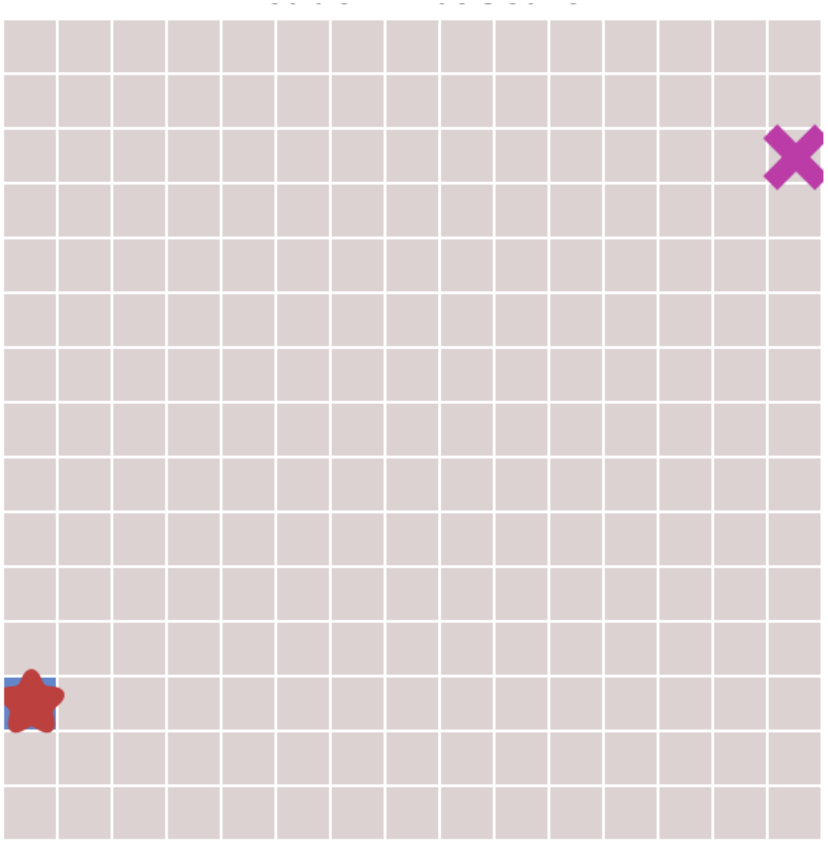
Greedy best first search

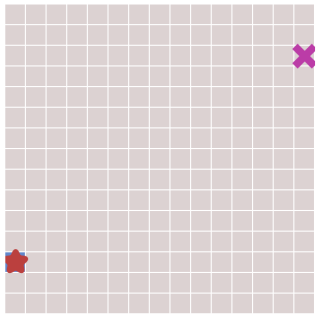


Bread first search

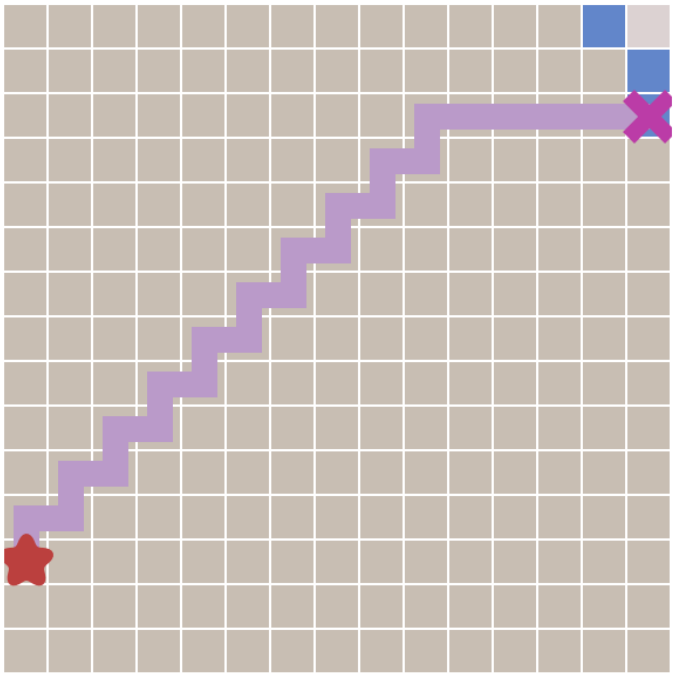


ESEMPIO più significativo del fatto che il nuovo algoritmo trova il cammino in molto meno tempo visitando molti meno nodi:

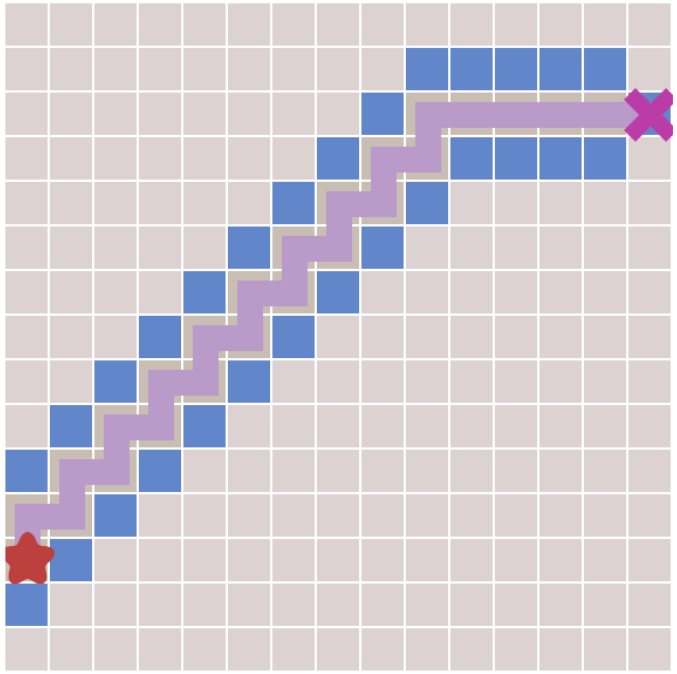




Breadth First Search

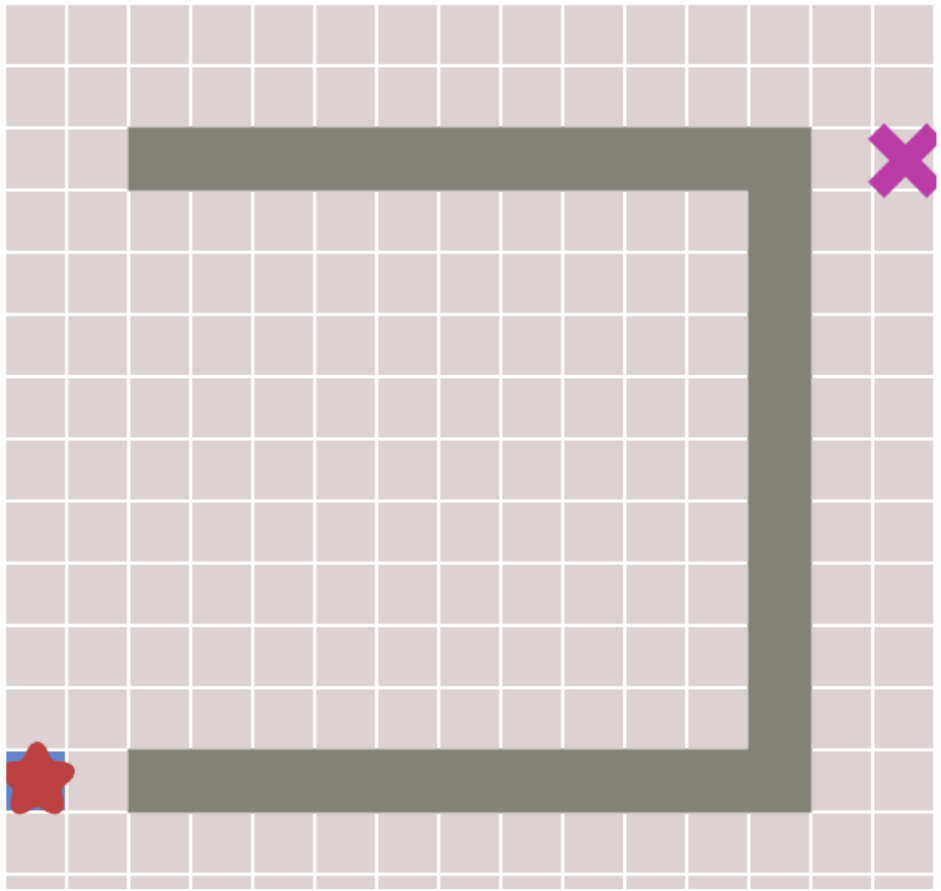


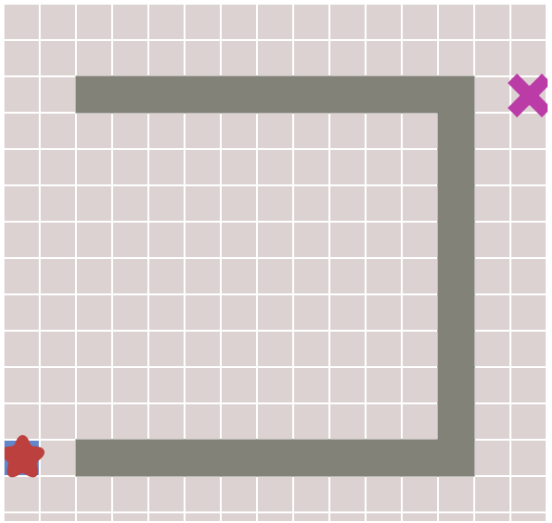
Greedy Best-First Search



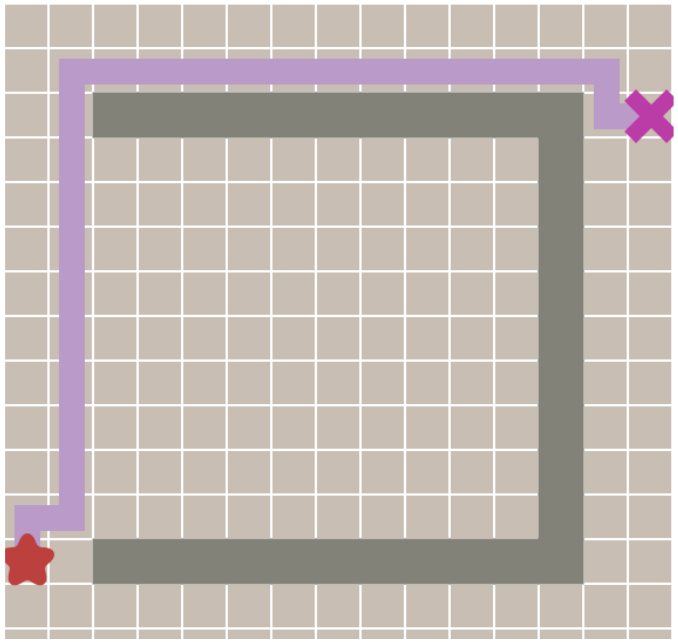
celle non visitate
celle visitate
celle nella coda

Attenzione: non sempre l'algoritmo Greedy best-first-search trova il cammino minimo!

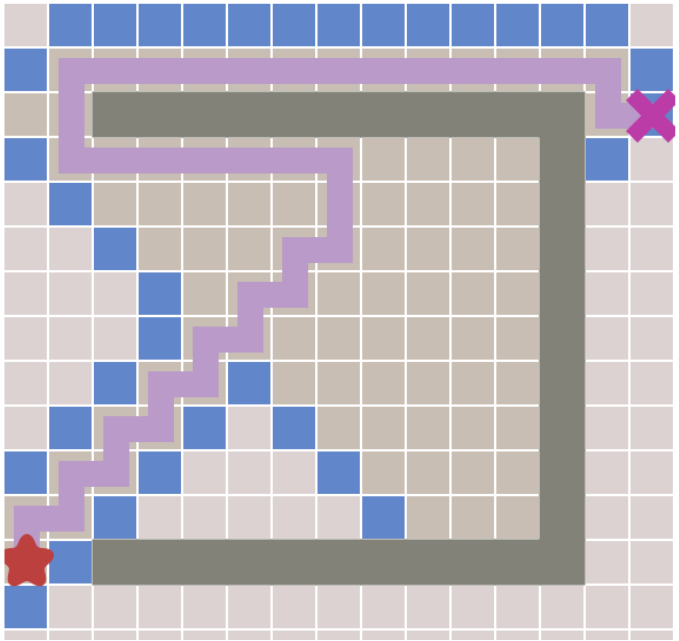




Breadth First Search



Greedy Best-First Search



celle non visitate
celle visitate
celle nella coda

Prof. **Angelo Monti**