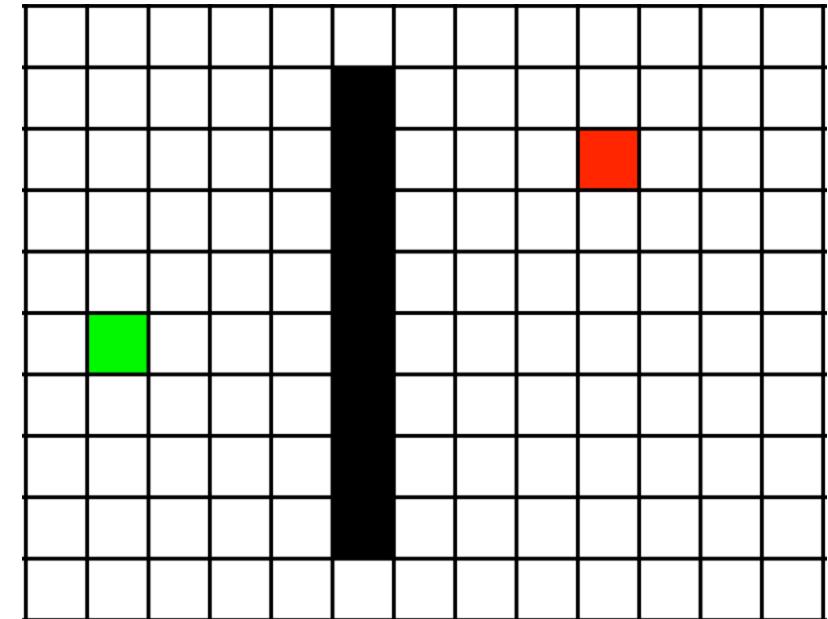
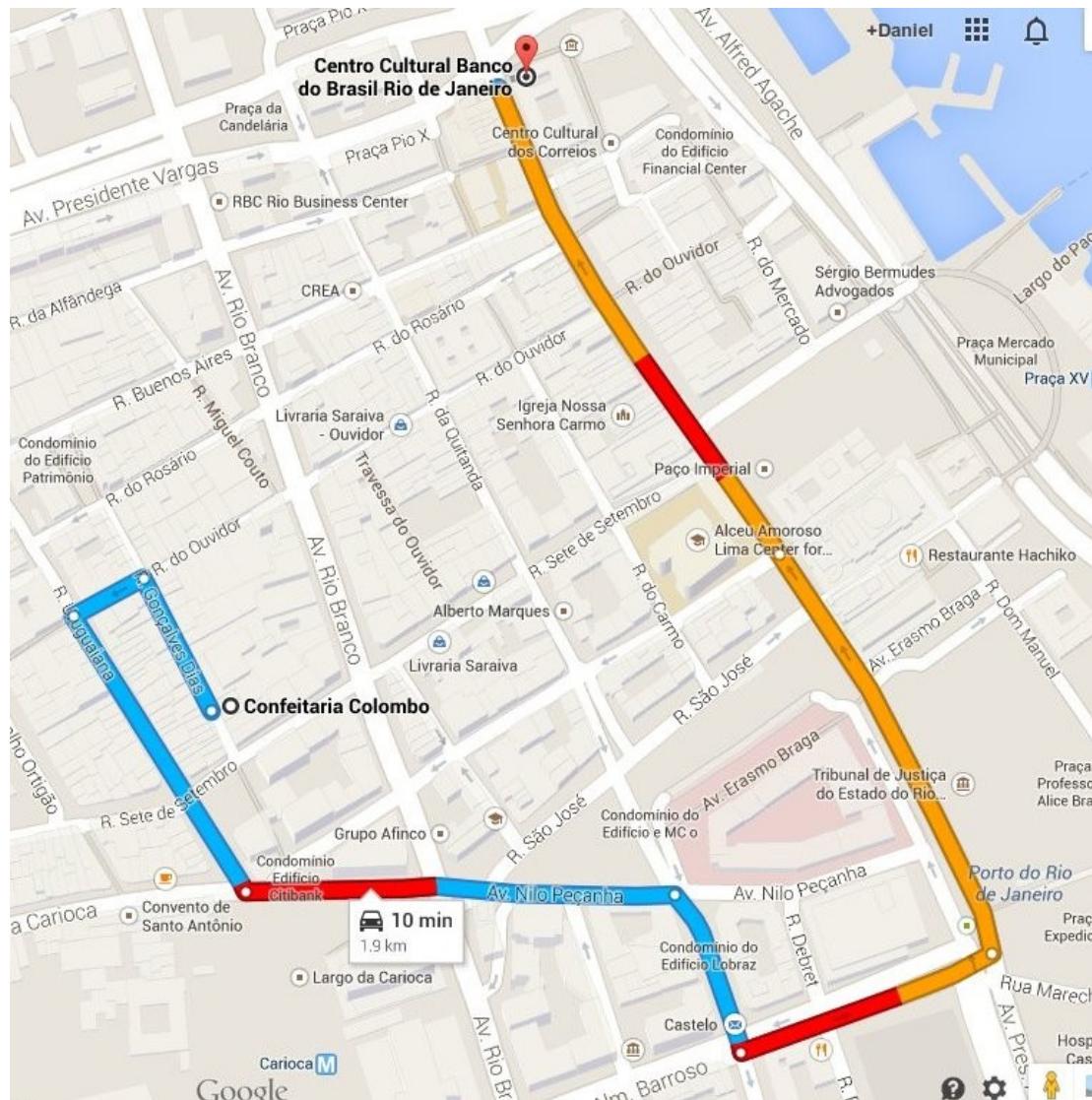


Grafos - Aula 10

Roteiro

- Problema do labirinto
(path finding)
- Busca informada
- Best-first search
- A*
- Heurística admissível

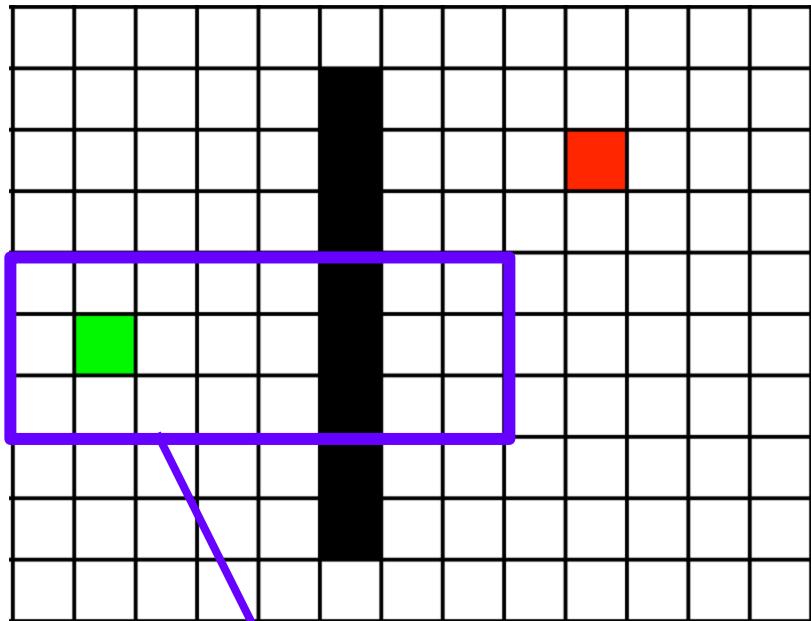
Problema do Labirinto (Path finding)



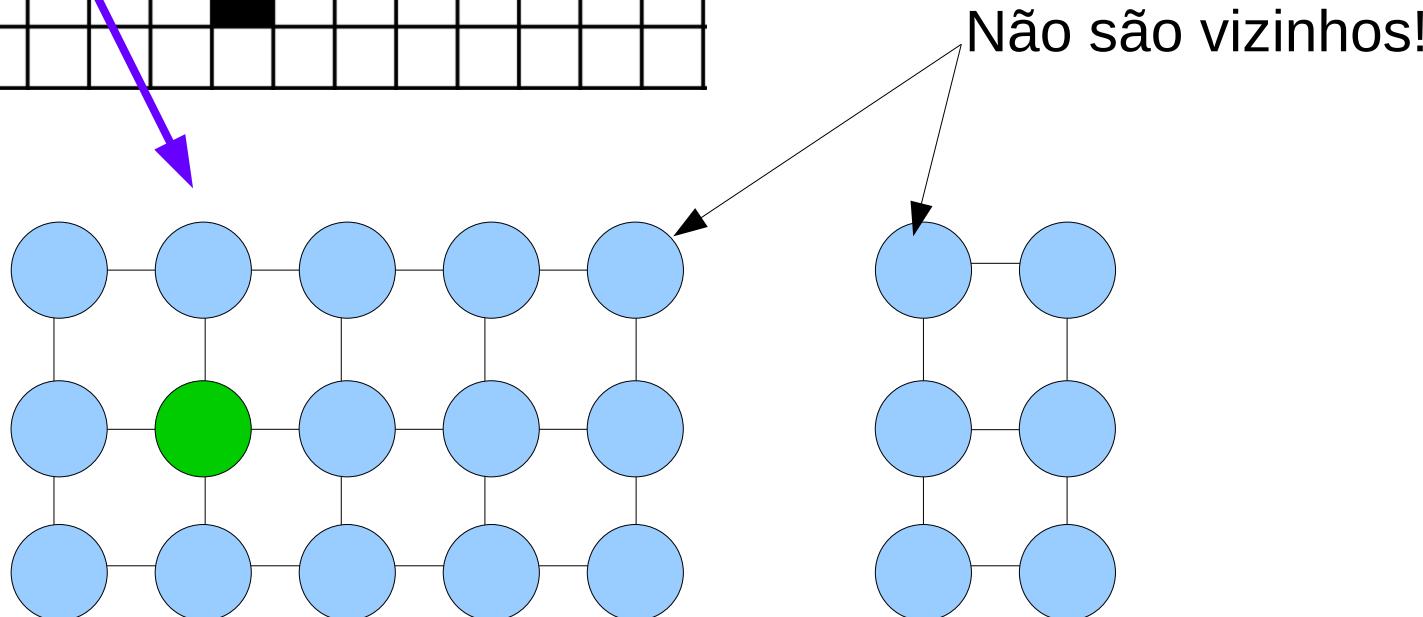
- **Problema:** Encontrar caminho mais curto em um labirinto
- Representação com grafos (direção, pesos)

BFS ou Dijkstra to the rescue!

De Grids para Grafos

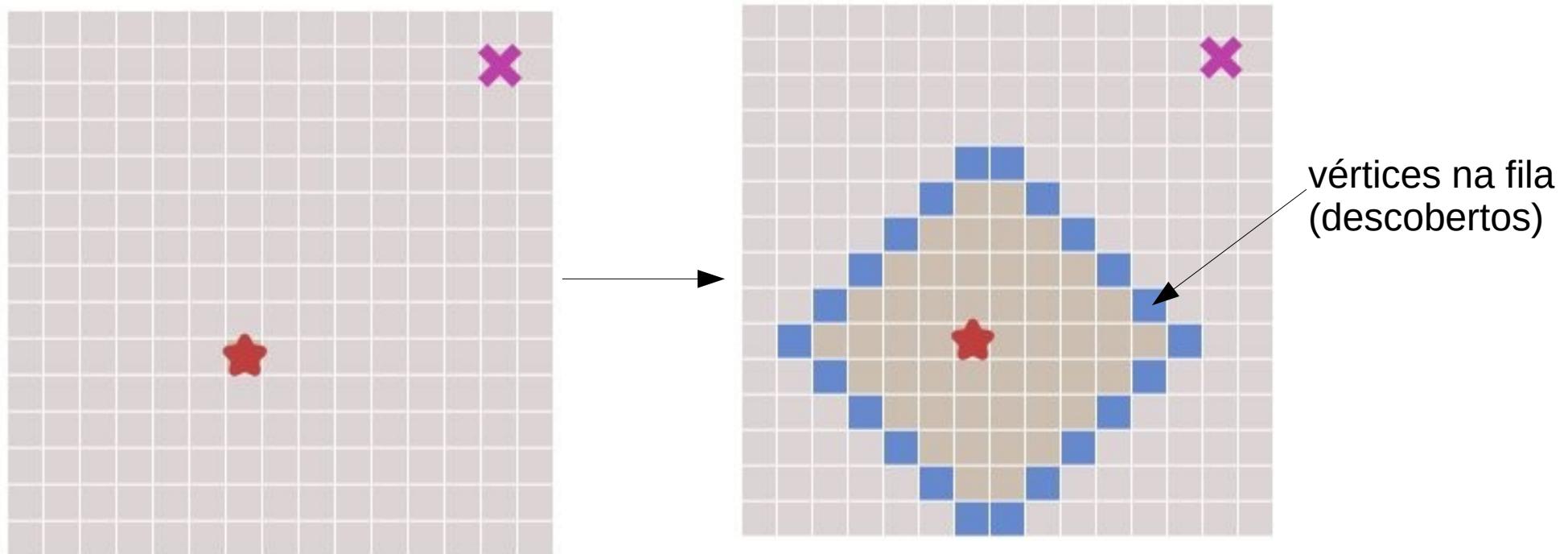


- Cada posição no grid se torna um vértice do grafo
- Arestras entre dois vértices posições são vizinhas



Problema com BFS e Dijkstra

- Expandem em “todas as direções” até encontrar destino
- Exploram muitos vértices desnecessários



Meio ineficiente (na prática)!

Algoritmos Informados

- BFS e Dijkstra são algoritmos de busca *não-informados*
 - não conhecem nada sobre destino

Não podem fazer melhor!

- Em muitos contextos, existe informação sobre o destino da busca

Exemplos?

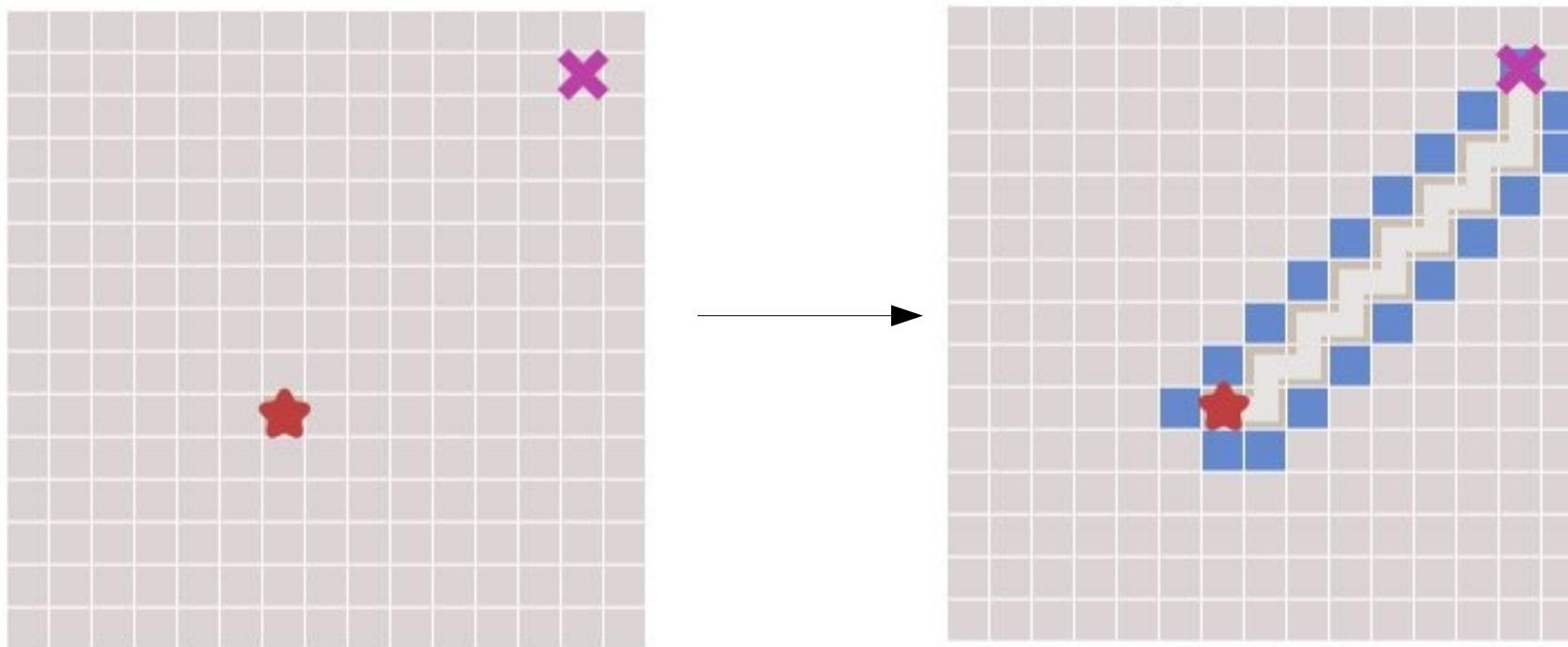
► Locais em uma cidade:
coordenadas (lat., long.)

- Grafos onde vértices possuem informação
 - que pode facilitar navegação

Como explorar isto?

Exemplo

- Grid em duas dimensões
- Coordenada do vértice está disponível em cada vértice
 - Coordenada do vértice destino é conhecida
- Algoritmo de busca pode utilizar estas informações



Bem mais eficiente!

Best-first Search

■ Ideia

- 1) utilizar informação dos vértices para estimar distância até destino
- 2) ordem de exploração é definida pela estimativa da distância (menor primeiro)
- Precisa de uma função para estimar distância
 - $f(a_u, a_v) = \text{estimativa da distância entre } u \text{ e } v$ usando a informação a_u e a_v disponível
 - $f(a_u, a_v)$ não precisa ser exata, pode ser alguma heurística (aproximação)
- *Best-first Search*
 - Como a BFS mas explora na ordem de menor estimativa de distâncias (*best-first*)

Algoritmo Best-first Search

- Baseado na estrutura do algoritmo de Dijkstra

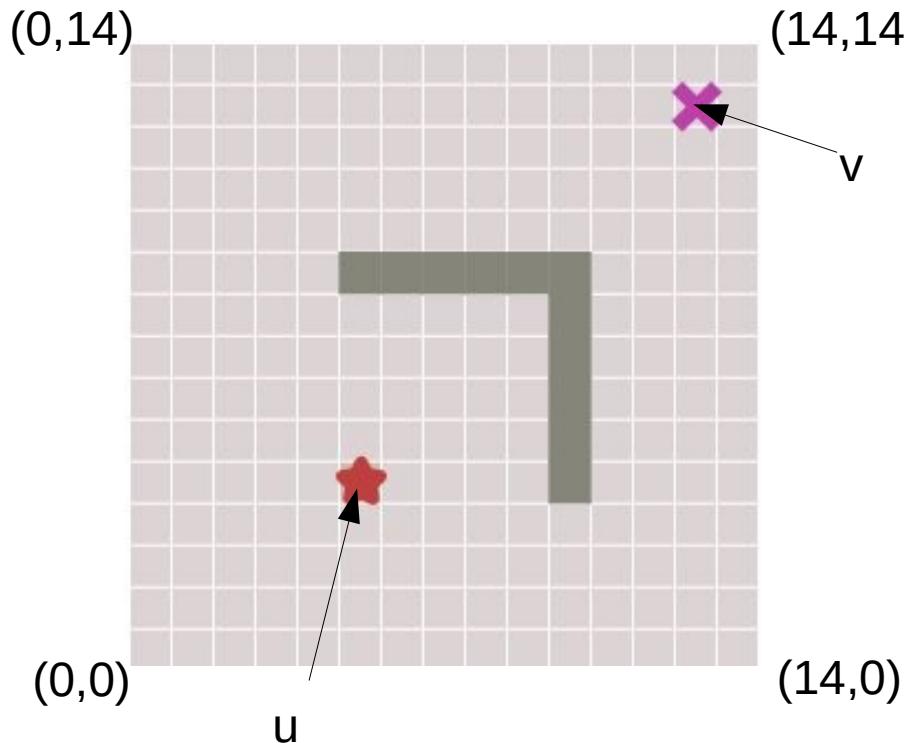
```
1. Best_first_search(G,s,d)
2. Para cada vértice v
3.   dist[v] = infinito ← Estimativa de distância
4.   define conjunto S = Ø // vazio entre v e d (destino)
5.   dist[s] = heuristica(s,d) ← Função para estimar
6. Enquanto S != V
7.   Selecione u em V-S, tal que dist[u] é mínima
8.   Se u == d, pare
9.   Adicione u em S
10.  Para cada vizinho v de u faça
11.    Se dist[v] == infinito
12.      dist[v] = heuristica(v,d) ← Atualizar a heurística
13.      pai[v] = u para o vértice v
```

- Explora vértices na ordem dada pela heurística

Funciona? Encontra menor caminho?

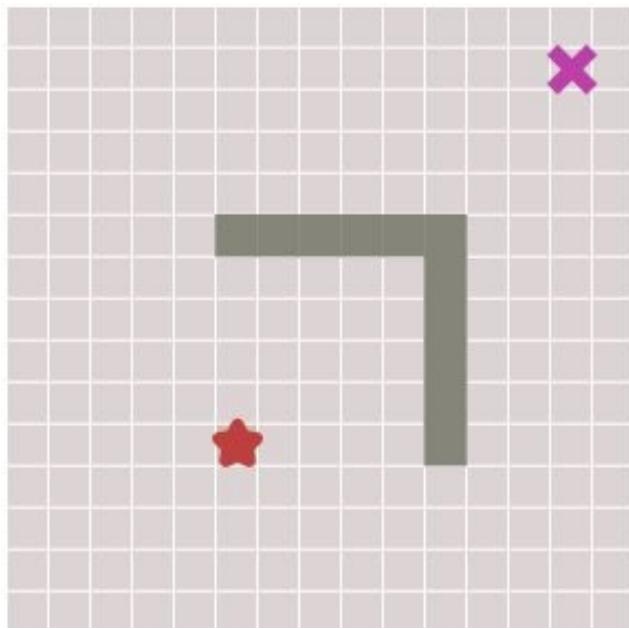
Exemplo de Heurística

- Grafo construído a partir de um grid
 - cada vértice é um célula do grid
 - cada vértice possui uma coordenada (x, y) no grid
- Distância de Manhattan entre vértices u e v
 - $f(u,v) = |x_u - x_v| + |y_u - y_v|$



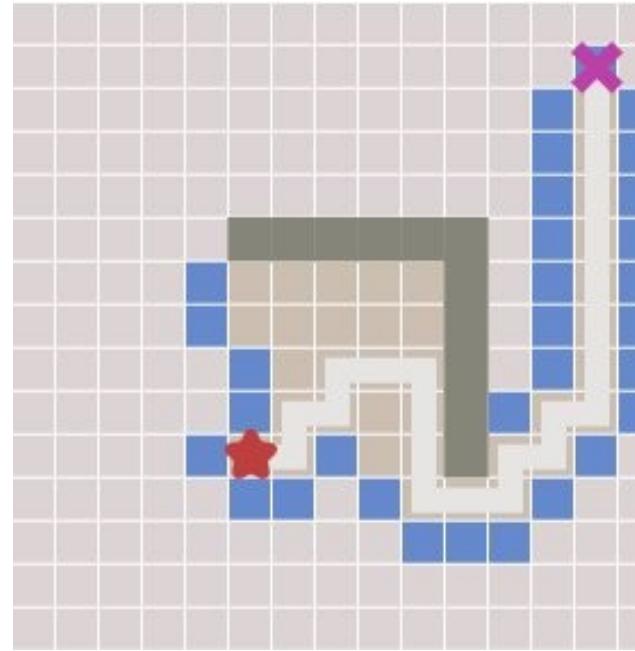
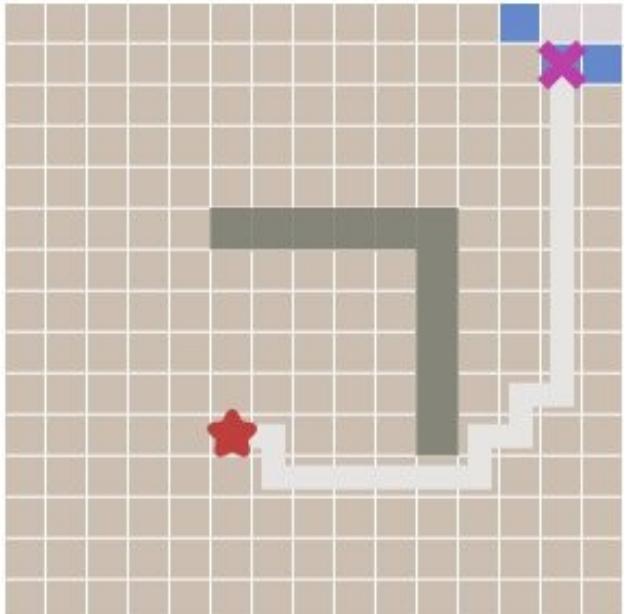
- Exemplo
- $f(u,v) = |5 - 13| + |4 - 13| = 17$
- Calculada a partir das coordenadas do vértices (e não do grafo)

Exemplo Best-First Search



Best-first

BFS



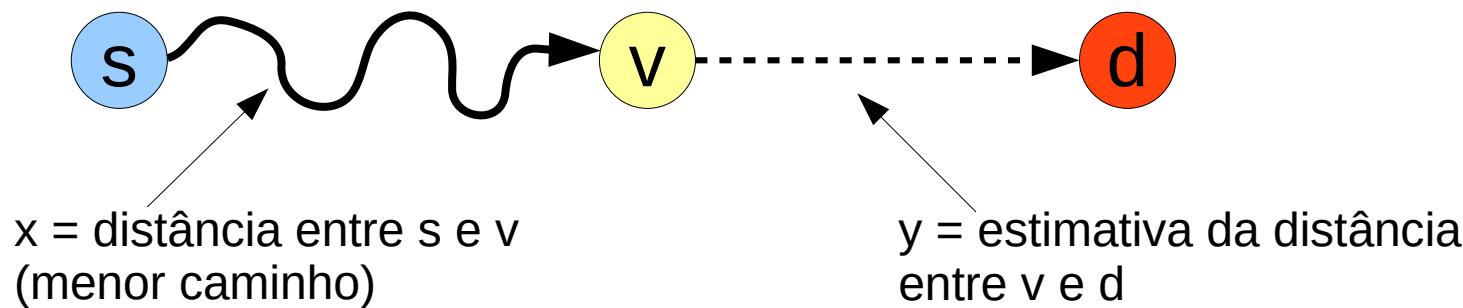
- Best-first search não garante menor caminho
- Best-first mais eficiente que BFS
- Tradeoff: eficiência x optimalidade

Melhorando Best-First

- **Problema:** Best-first não utiliza o que foi descoberto, confia muito na heurística

Ideia: Usar o que já foi descoberto!

- Considere s origem, d destino, e v um vértice qualquer já descoberto



- $x + y$: estimativa da distância entre s e d passando por v
- Explorar baseado nesta estimativa
- Se $y = 0$, estimativa é idêntica ao algoritmo de Dijkstra!

Algoritmo A*

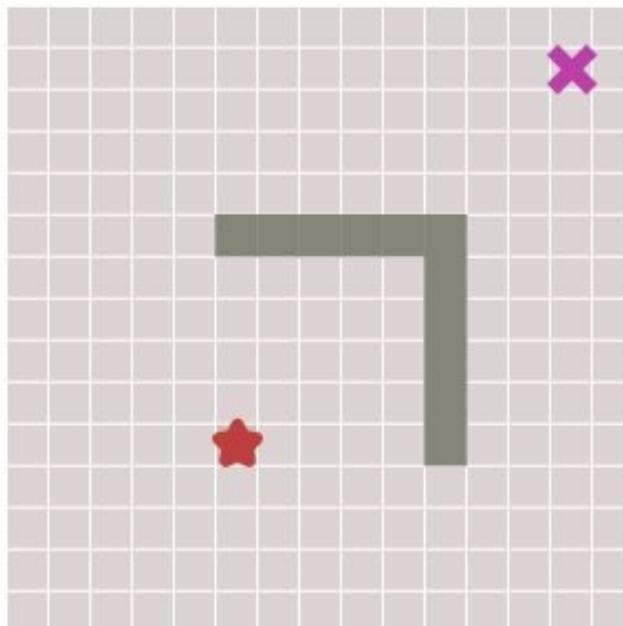
- Muito famoso (início anos 70) e muito utilizado, diversas aplicações em problemas de busca em IA
 - ex. jogos, como xadrez e Go
- Generaliza Dijkstra e possui propriedades importantes (que veremos)
- Implementação demanda dois vetores de distâncias
 - $\text{dist_s}[v]$: distância de s até v
 - $\text{dist}[v]$: estimativa de distância de s até d passando por v
- $\text{dist_s}[v]$: mesmo vetor mantido pelo alg. de Dijkstra
- $\text{dist}[v]$: define ordem de exploração dos vértices

Algoritmo A*

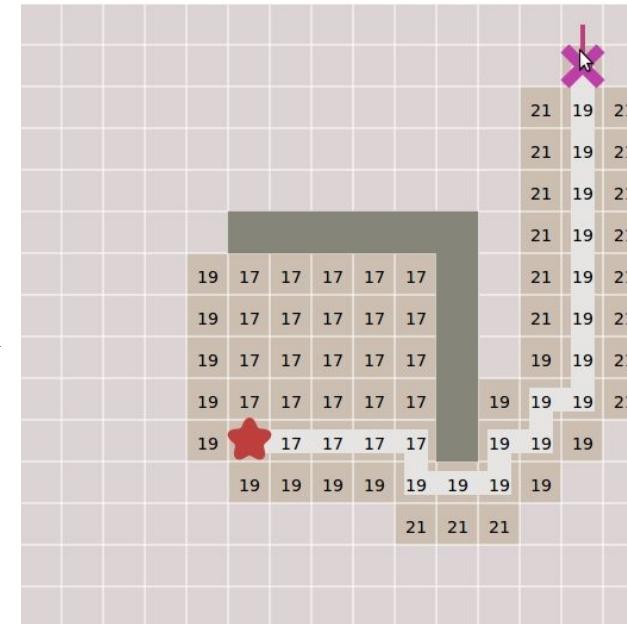
```
1. A_star(G,s,d)
2. Para cada vértice v
3.     dist_s[v] = dist[v] = infinito
4. Define conjunto S = {} // vazio
5. dist_s[s] = 0
6. dist[s] = 0 + heuristica(s,d)
7. Enquanto S != V
8.     Selecione u em V-S, tal que dist[u] é mínima
9.     Se u == d, pare
10.    Adicione u em S
11.    Para cada vizinho v de u faça
12.        Se dist_s[v] > dist_s[u] + w(u,v) então
13.            dist_s[v] = dist_s[u] + w(u,v)
14.            dist[v] = dist_s[v] + heuristica(v,d)
15.            pai[v] = u
```

- Extrai mínimo utilizando estimativa de distância
- Atualiza distância até v e estimativa até d via v

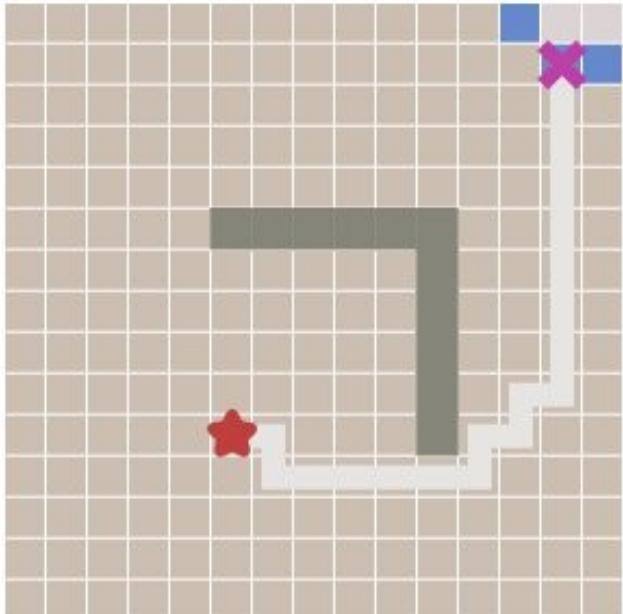
Exemplo do A*



A*



BFS

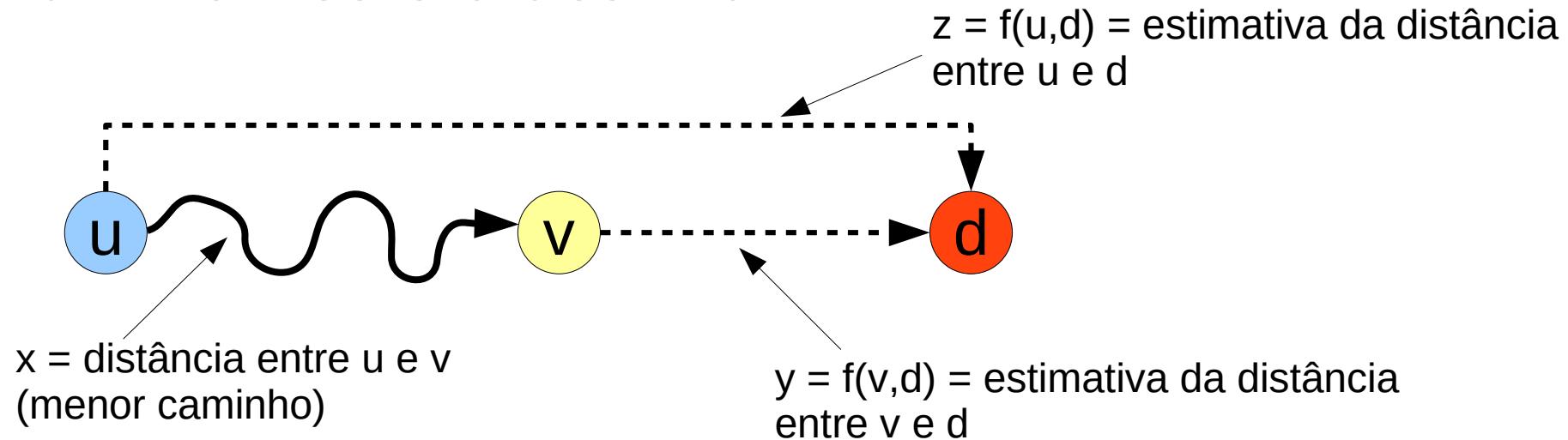


- A* encontrou caminho mínimo
- Explorou muito menos que BFS

Maravilha sempre?

Heurística Admissível

- Propriedade da função que estima distância entre um vértice e o destino



- função f é dita admissível se $z \leq x + y$
 - $f(u,d) \leq \text{dist}(u,v) + f(v,d)$ para todos vértices do grafo
 - heurística é conservadora, nunca sobre estima a distância real no grafo

Propriedades do A*

- Teorema: A* retorna o caminho mínimo se heurística é admissível
 - ótima notícia, mas nem tanto

Como saber se heurística é admissível?

- Exemplo de mapas e caminhos (vértices são pontos no mapa e distância é comprimento do caminho)
 - $f(u,v) = \text{distância Euclideana entre as coordenadas no mapa dos vértices } u \text{ e } v$
 - f é admissível: distância Euclideana é sempre menor que comprimento do menor caminho
- Em geral, difícil encontrar heurísticas admissíveis
- A* pode falhar em encontrar caminho mínimo
 - mas é mais eficiente que Dijkstra (*tradeoff*)

Explorando Buscas

- Website: Red Blob Games - Amit Patel
 - “I explore visual and interactive ways of explaining math and computer algorithms, especially those used in computer games. I want to learn by playing with things.”
- A* search:
<http://www.redblobgames.com/pathfinding/a-star/introduction.html>

**Explore este (e outros) recursos
deste website!**

* imagens dos slides foram obtidas em exemplos deste website