

Busca de caminhos em grafos

Algoritmos de Dijkstra e A*

Prof. Marcelo de Souza

`marcelo.desouza@udesc.br`





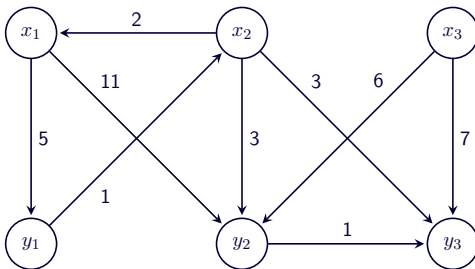
Leitura obrigatória:

- Capítulo 4 de [Goldbarg and Goldbarg \(2012\)](#) – Caminhos.
- Capítulo 4 de [Kleinberg and Tardos \(2006\)](#) – Algoritmos gulosos.

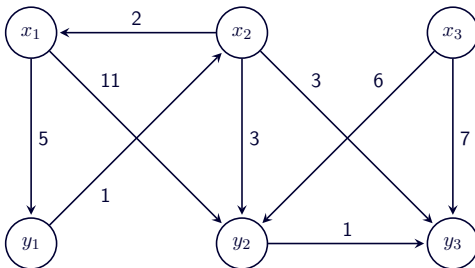
Leitura complementar:

- Capítulo 14 de [Goodrich et al. \(2014\)](#) – Algoritmos em grafos.
- Capítulo 15 de [Preiss \(2001\)](#) – Grafos e algoritmos em grafos.

- Um grafo ponderado possui valores numéricos associados às arestas.
- Em muitas aplicações, o peso de uma aresta denota o custo da mesma.
- **Aplicações:**
 - **Tráfego:** arestas são vias cujo peso é a distância ou o tempo de viagem.
 - **Comunicação de dados:** arestas são links entre computadores com o respectivo tempo de transmissão.



- **Problema:** descobrir o menor caminho entre os vértices s e t .
 - Um grafo não ponderado é aquele com todos os pesos iguais a 1. Neste caso, o menor caminho é o que passa pelo menor número de arestas.
 - **Exemplo:** para chegar de x_1 a y_3 no grafo abaixo existem dois caminhos $[(x_1, y_2, y_3)]$ e $[(x_1, y_1, x_2, y_3)]$. Apesar de passar por mais arestas, o segundo caminho é menor (custo total menor).
- **Variante:** descobrir o menor caminho entre os vértices s e t .
 - Essa variante é chamada de **árvore de caminhos mínimos**.

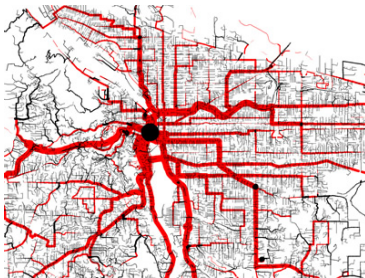


Algoritmo de Dijkstra



Conceitos

- Proposto por Edsger Dijkstra (veja [Dijkstra \(1959\)](#)), o algoritmo calcula a árvore de caminhos mínimos a partir de um vértice de origem s .
 - $d(u)$ é a menor distância encontrada para chegar em u a partir de s .
 - $p(u)$ é o vértice predecessor de u no menor caminho encontrado para chegar em u a partir de s .
 - Q é uma fila de prioridades com os vértices do grafo e $d(u)$ como prioridade.



Algoritmo de Dijkstra



Pseudocódigo

Algorithm: Dijkstra(Vertex s)

$Q \leftarrow V$

Set $d(v) = \infty$ and $p(v) = -1$ for each $v \in V$

$d(s) = 0$

while $Q \neq \emptyset$ **do**

$u \leftarrow \text{extract-min}(Q)$

for each v *adjacent to* u **do**

if $d(v) > d(u) + w(u, v)$ **then**

$d(v) \leftarrow d(u) + w(u, v)$

$p(v) \leftarrow u$

Algoritmo de Dijkstra



Pseudocódigo

Algorithm: Dijkstra(Vertex s)

$Q \leftarrow V$

Set $d(v) = \infty$ and $p(v) = -1$ for each $v \in V$

$d(s) = 0$

while $Q \neq \emptyset$ **do**

$u \leftarrow \text{extract-min}(Q)$

for each v *adjacent to* u **do**

if $d(v) > d(u) + w(u, v)$ **then**

$d(v) \leftarrow d(u) + w(u, v)$

$p(v) \leftarrow u$

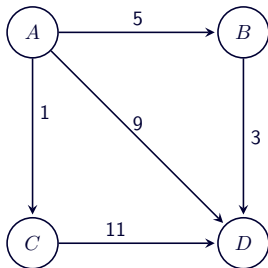
- Sempre que um elemento é extraído de Q , foi descoberto o menor caminho até ele (predecessor e custo).
- Quando todos os elementos foram removidos, todos os caminhos mínimos de s a todos os demais vértices foram determinados.

Algoritmo de Dijkstra



Exercício de aplicação

- Considere o grafo apresentado abaixo. Simule a execução do algoritmo de Dijkstra para encontrar os caminhos mínimos de A para todos os demais vértices.





- **Complexidade:**

- O pré-processamento é executado em $O(n)$.
- O laço externo executa para cada vértice do grafo – $O(n)$.
- A extração de Q custa $O(\log n)$ usando um heap binário.
- O laço interno executa exatamente m vezes (todos os arcos).
- Cada atualização de vértices implica em atualizar Q , o que custa $O(\log n)$.
- Complexidade total: $O(m \log n)$.

- **Limitações:**

- O algoritmo não funciona com pesos negativos.
- Ineficiente para calcular o caminho simples entre um par de vértices.

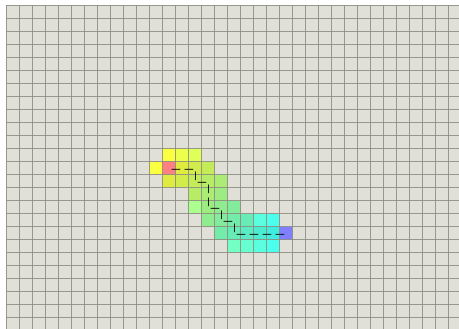
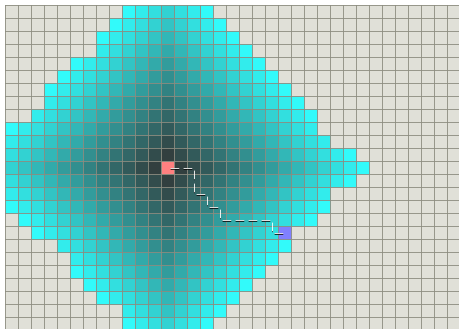


- Algoritmo de Dijkstra gasta processamento para determinar todos os caminhos mínimos a partir de s .
- Caso apenas o caminho de s a t seja de interesse, podemos parar o algoritmo quando t é removido de Q e retornar o caminho encontrado.
- Existe forma mais eficiente?
 - Sim, podemos guiar a busca em direção ao vértice de destino.
 - **Algoritmo A*!**
- O A* é o algoritmo mais utilizado para busca de caminhos em grafos.
- Veremos a aplicação do A* para grafos **não-ponderados**.
 - A aplicação para grafos ponderados pode ser feita com ajustes simples.
- Aplicações populares: jogos, inteligência artificial, roteamento.

Busca de caminho entre um par de vértices



Dijkstra *versus* A*





- A fila de prioridades usa uma função $f(v)$ para determinar a prioridade do vértice v .
- A função é composta por dois componentes: $f(v) = g(v) + h(v)$.
 - $g(v)$ é a função de custo, dada pela distância s a v .
 - $h(v)$ é a função heurística, dada pela **estimativa de distância** de v a t .
- A busca é guiada pelo custo em se chegar ao vértice selecionado ($g(v)$), mas também o quão próximo ele está do objetivo ($h(v)$).

- Dado um grafo não-ponderado, podemos escrever o algoritmo A* como:

Algorithm: A* (Vertex s , Vertex t)

$Q \leftarrow \{s\}$

Set $p(v) = -1$ and v not marked for each $v \in V$

while $Q \neq \emptyset$ **do**

$u \leftarrow \text{extract-min}(Q)$

if $u = t$ **then**

return *path from s to t*

for *each v adjacent to u* **do**

if v is not marked **then**

 Set v marked

$p(v) \leftarrow u$

 Add v to Q

Busca A^*



Função heurística

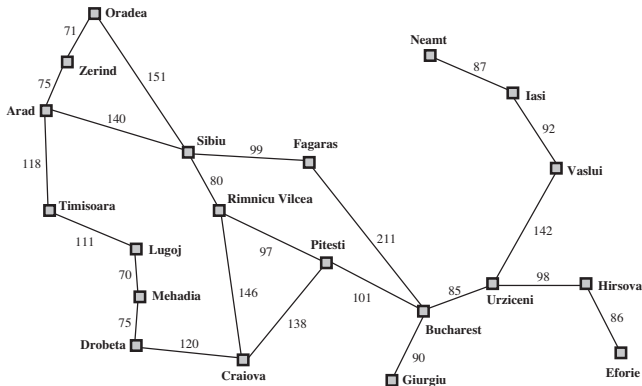
- Esta função estima a distância de um vértice até o vértice de destino.
- É sensível ao contexto, isto é, ao problema que está sendo resolvido.

Busca A*

Função heurística



- Esta função estima a distância de um vértice até o vértice de destino.
- É sensível ao contexto, isto é, ao problema que está sendo resolvido.
- **Exemplo:** dado um grafo que representa as cidades de um país, encontra o menor caminho desde uma cidade até outra.

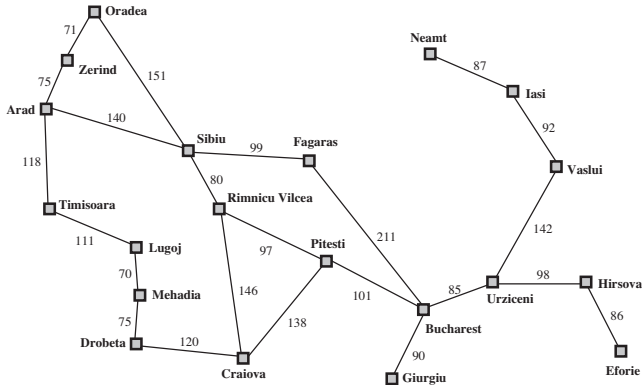


Busca A*

Função heurística



- Consideremos a cidade de origem *Sibiu*, com destino a *Bucharest*.
- $g(v)$ é o custo de sair de *Sibiu* até v .
- $h(v)$ é uma estimativa de distância de v até *Bucharest*.

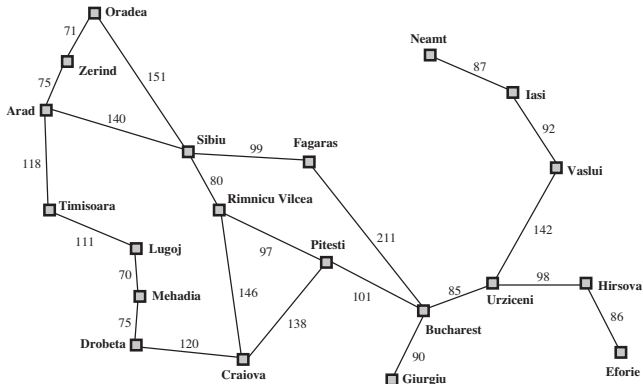


Busca A*

Função heurística



- Neste caso, $h(v)$ pode ser a distância em linha reta de v até *Bucharest*.



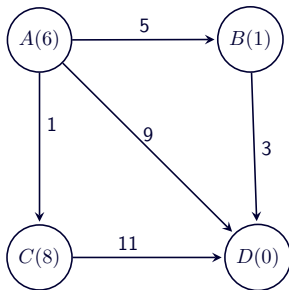
- O algoritmo A* é ótimo, desde que a função heurística seja **admissível**.
- Uma função admissível é aquela que **nunca superestima** o custo para alcançar o objetivo.
 - Ou seja, o custo estimado nunca é superior ao custo real.

Busca A*



Exercício de aplicação

- Considere o grafo apresentado abaixo. O valor numérico nos vértices representa a função heurística para o mesmo. Simule a execução do algoritmo A* para encontrar o caminho mínimo de A a D .





- Para árvores de caminhos mínimos em grafos não-ponderados:
 - Busca em largura.
- Para pesos negativos:
 - Bellman-Ford.
- Para caminhos entre todo par de vértices:
 - Floyd-Warshall.
 - Algoritmo de Johnson.

Exercício

Busca de caminhos



1. Crie um programa que receba a descrição de um grafo dirigido e ponderado (vértices, arcos, pesos) e permita a busca de caminhos no grafo. O usuário poderá selecionar um vértice de origem s e verificar o caminho mínimo de s até todos os demais vértices. O usuário também poderá informar um par de vértices e checar o menor caminho entre eles. Para isso, implemente os algoritmos de busca de Dijkstra e A*.



- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.
- Goldbarg, M. and Goldbarg, E. (2012). *Grafos: Conceitos, algoritmos e aplicações*. Elsevier.
- Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2014). *Data structures and algorithms in Java*. John Wiley & Sons, 6th edition.
- Kleinberg, J. and Tardos, É. (2006). *Algorithm Design*. Pearson Education India.
- Preiss, B. R. (2001). *Estruturas de dados e algoritmos: padrões de projetos orientados a objetos com Java*. Campus.