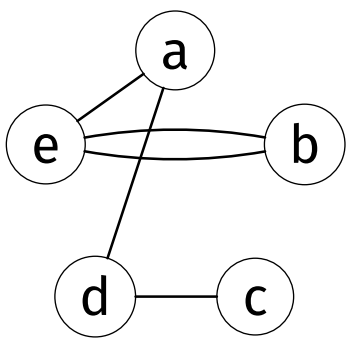
Grafer

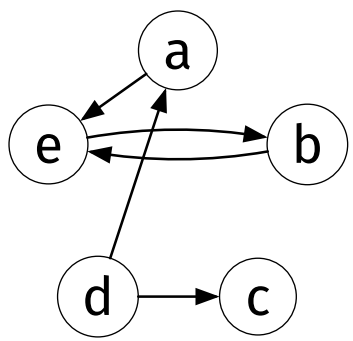
Grafrepresentation

Grafrepresentation

En graf är en samling *hörn* V och en samling *kanter* E:



Oriktad graf



Riktad graf

$$V = \{a, b, c, d, e\}, \;\; E = \{ae, eb, be, da, dc\}$$

De tre vanligaste sätten att representera grafer:

Kantlista

$$(0,3)$$
$$(0,4)$$
$$(1,4)$$
$$(2,3)$$

Grannlistor

$$0 \to 3,4$$
$$1 \to 4$$
$$2 \to 3$$
$$3 \to 0,2$$
$$4 \to 0,1$$

Grannmatris

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Vad blir minneskomplexiteten m.a.p. $|V|$ och $|E|$?

Kantlista

$(0, 3)$

$(0, 4)$

$(1, 4)$

$(2, 3)$

Grannlistor

$0 \to 3, 4$

$1 \to 4$

$2 \to 3$

$3 \to 0, 2$

$4 \to 0, 1$

Grannmatris

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Tidskomplexiteten för att avgöra om två hörn är grannar?

Kantlista

$$(0,3)$$
$$(0,4)$$
$$(1,4)$$
$$(2,3)$$

Grannlistor

$$0 \to 3,4$$
$$1 \to 4$$
$$2 \to 3$$
$$3 \to 0,2$$
$$4 \to 0,1$$

Grannmatris

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Tidskomplexiteten för att hitta alla grannar till ett hörn?

Kantlista

$$(0,3)$$
$$(0,4)$$
$$(1,4)$$
$$(2,3)$$

Grannlistor

$$0 \to 3,4$$
$$1 \to 4$$
$$2 \to 3$$
$$3 \to 0,2$$
$$4 \to 0,1$$

Grannmatris

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Hur kan dessa se ut om vi har en riktad/viktad graf?

Kantlista

$$(0, 3)$$
$$(0, 4)$$
$$(1, 4)$$
$$(2, 3)$$

Grannlistor

$$0 \to 3, 4$$
$$1 \to 4$$
$$2 \to 3$$
$$3 \to 0, 2$$
$$4 \to 0, 1$$

Grannmatris

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Grafsökning

Det finns två huvudsakliga metoder för att utforska en graf

Bredden först

- Vi utforskar de närmsta hörnen först, och går sedan vidare till hörn med avstånd 2, 3... från ursprungshörnet.
- Implementeras med en *kö*.

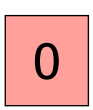
Djupet först

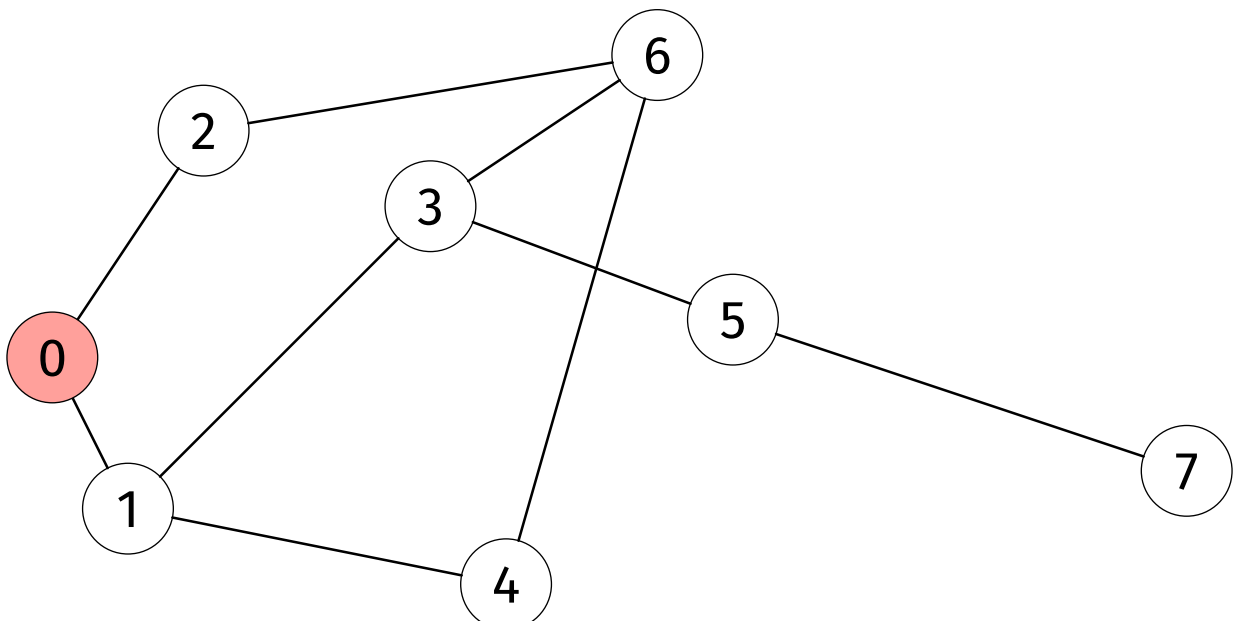
- Vi går så lång in i grafen som möjligt tills vi när en återvändsgränd, då backtrackar vi.
- Implementeras med en *stack* eller rekursion.

Bredden-först-sökning

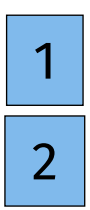
Breadth first search (BFS)

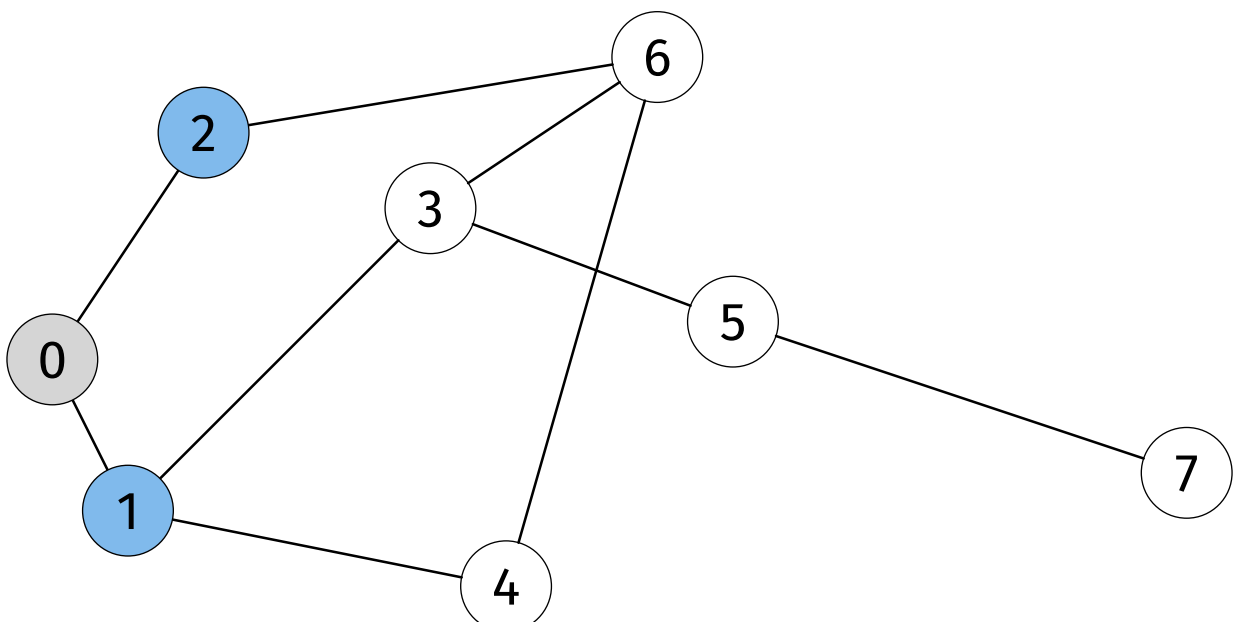
Bredden-först-sökning



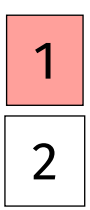


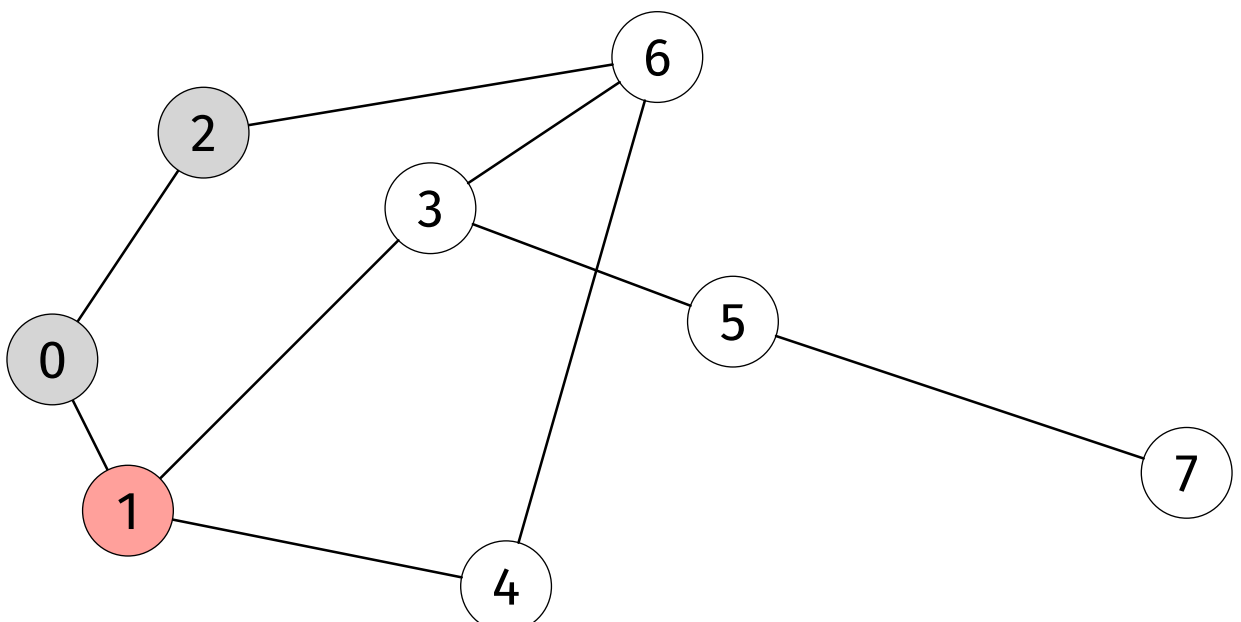
Bredden-först-sökning



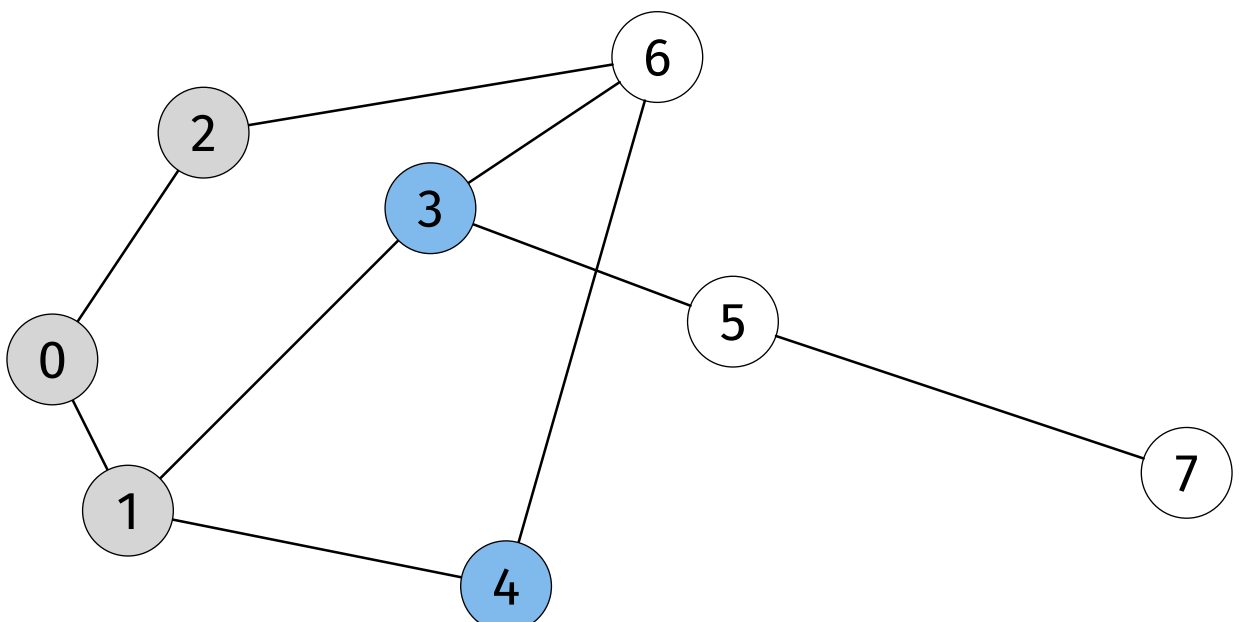


Bredden-först-sökning

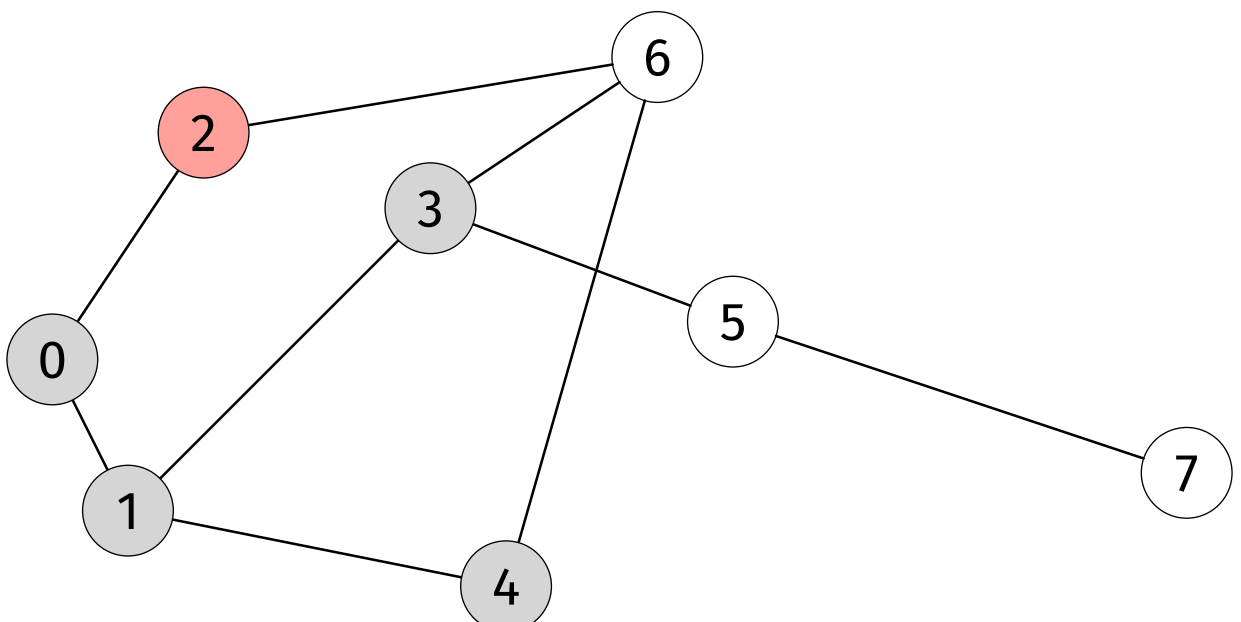




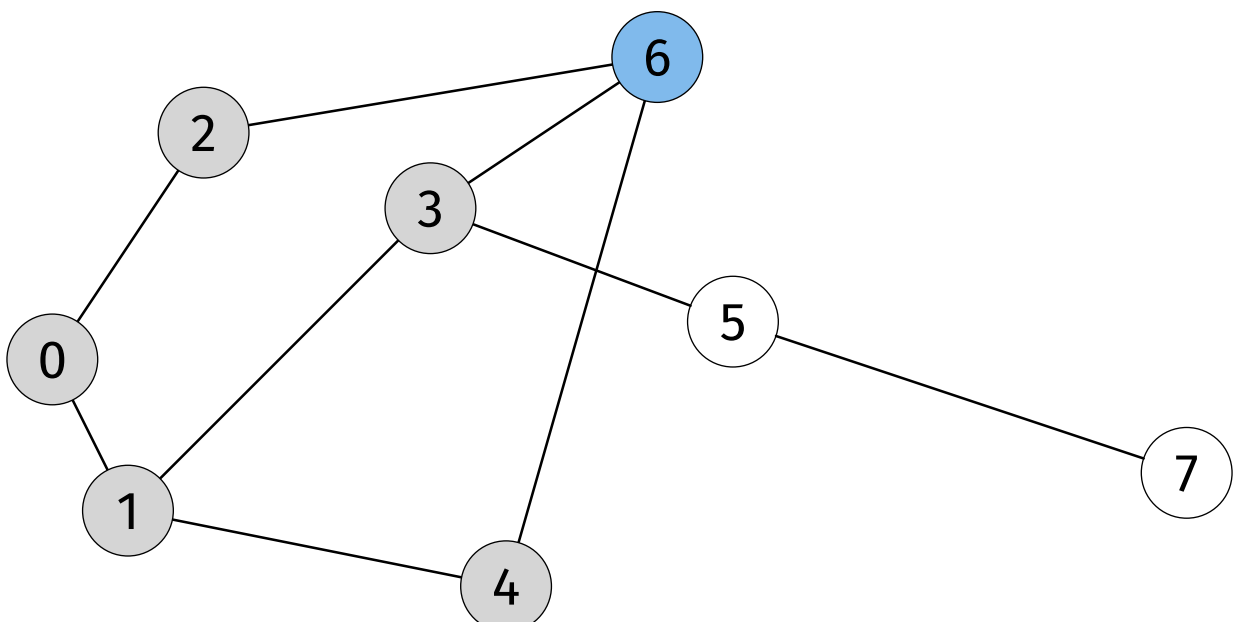
Bredden-först-sökning



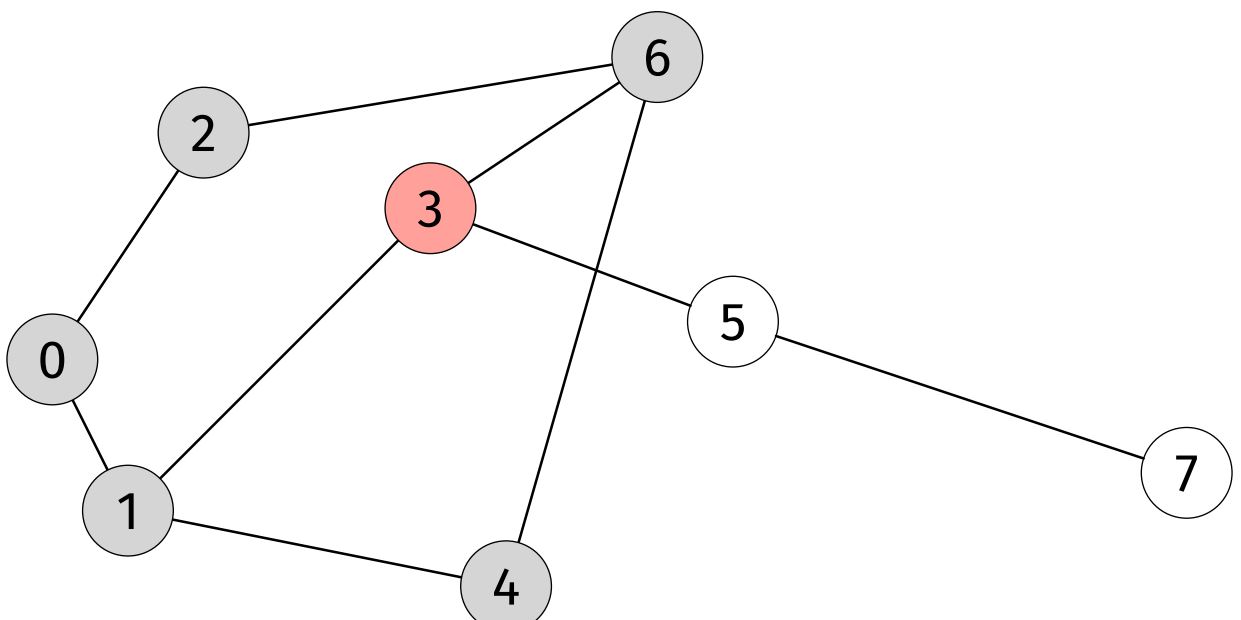
Bredden-först-sökning



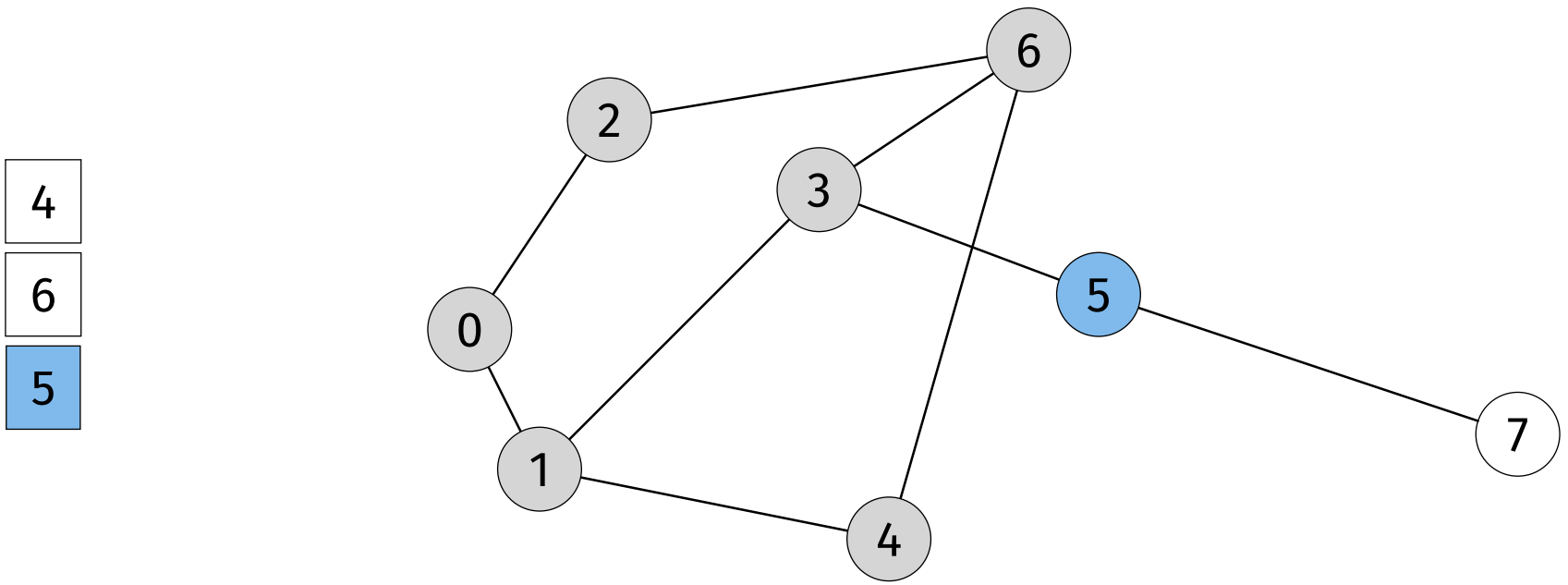
Bredden-först-sökning



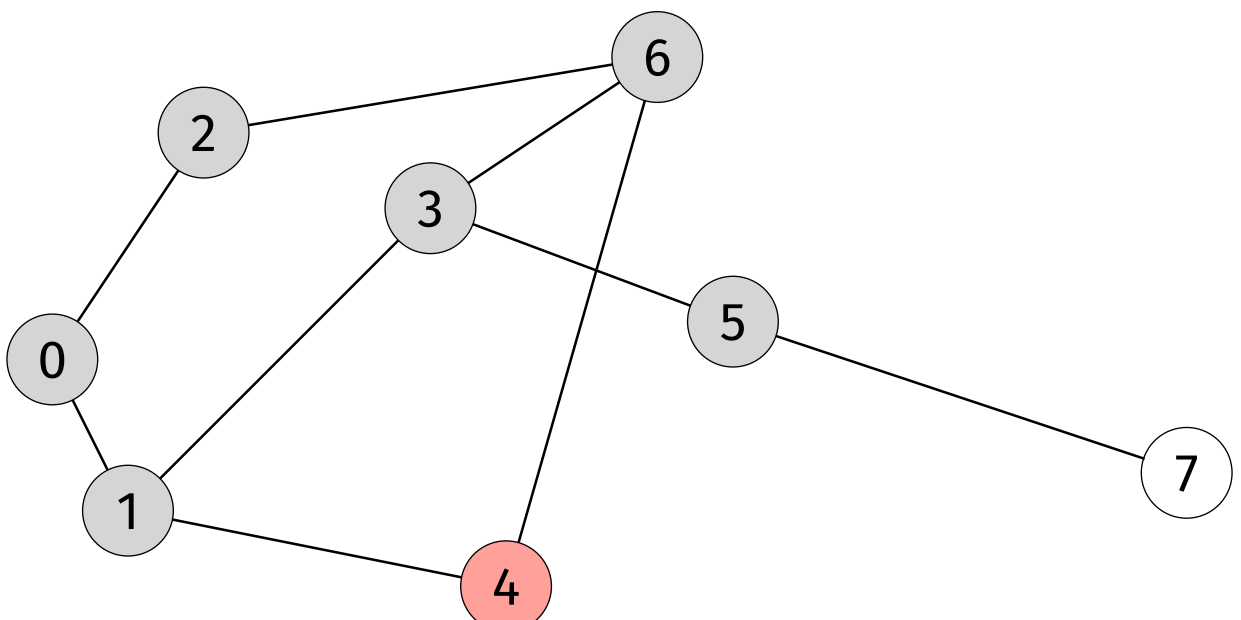
Bredden-först-sökning



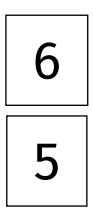
Bredden-först-sökning

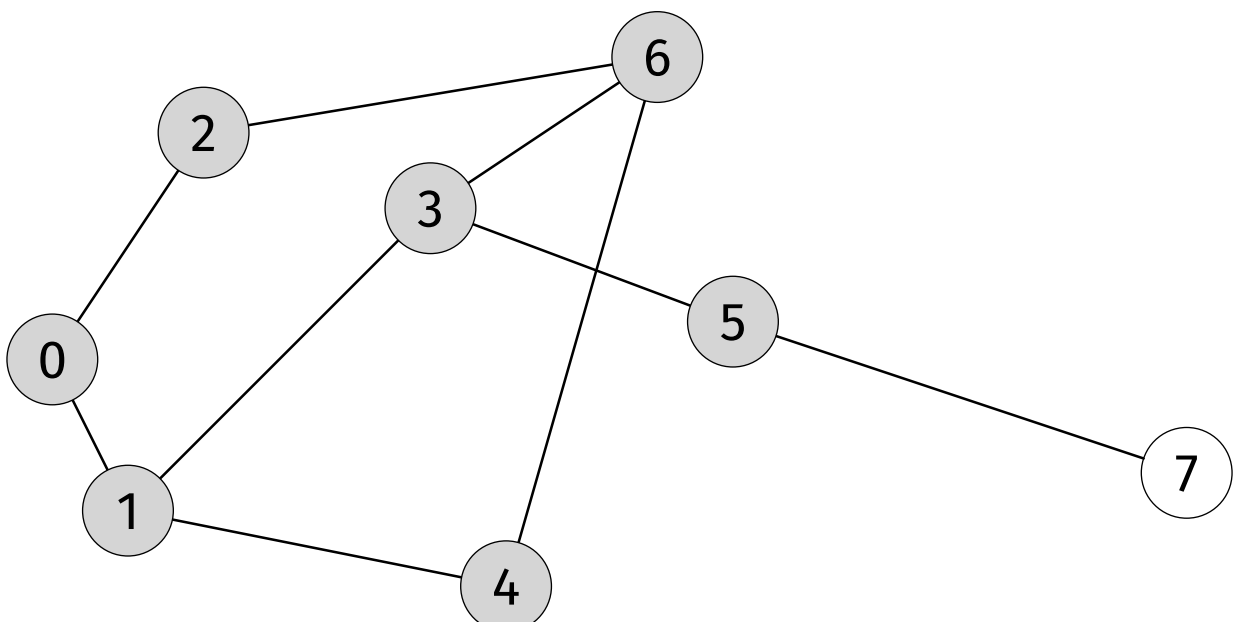


Bredden-först-sökning

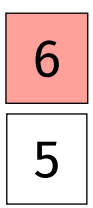


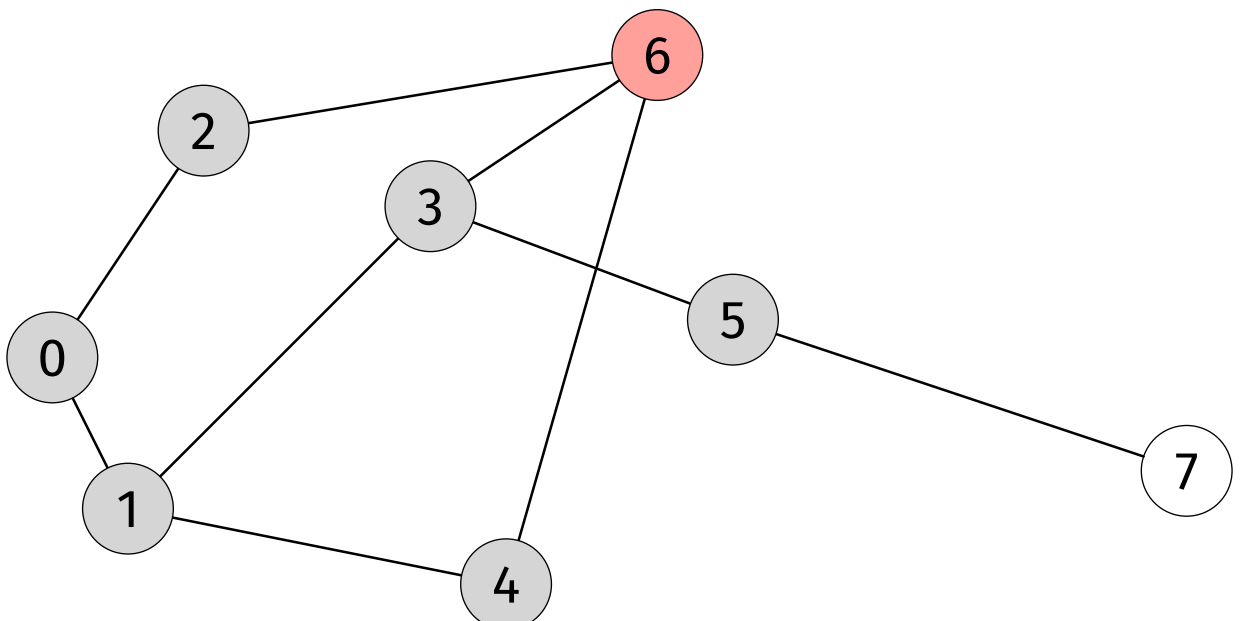
Bredden-först-sökning





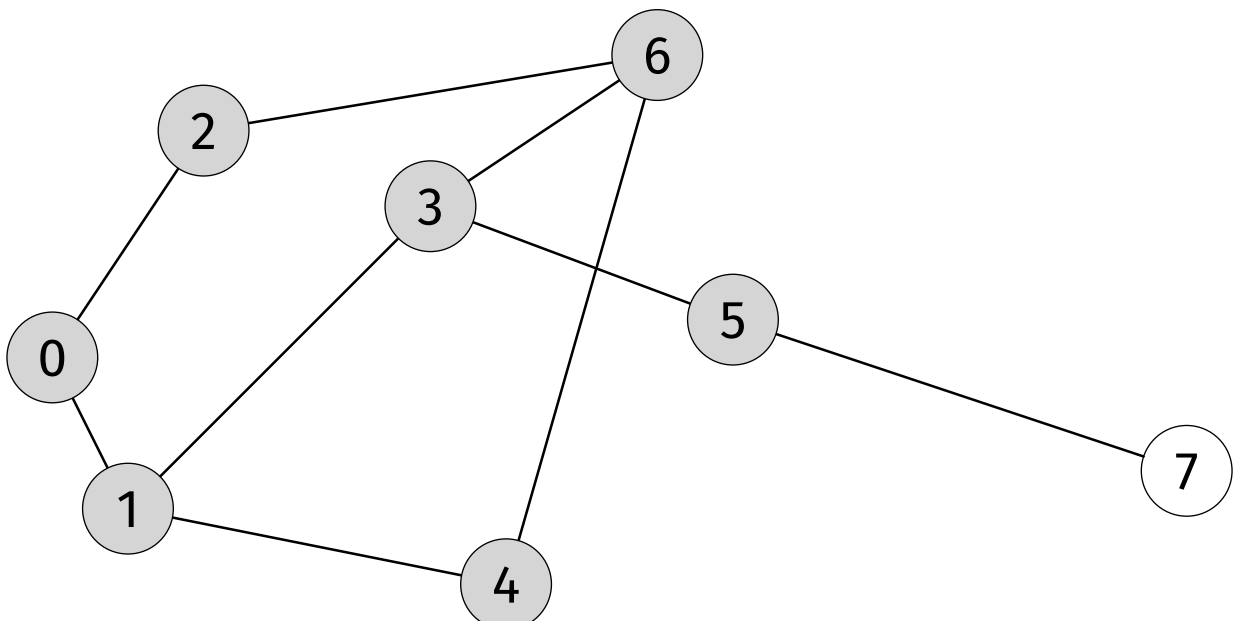
Bredden-först-sökning





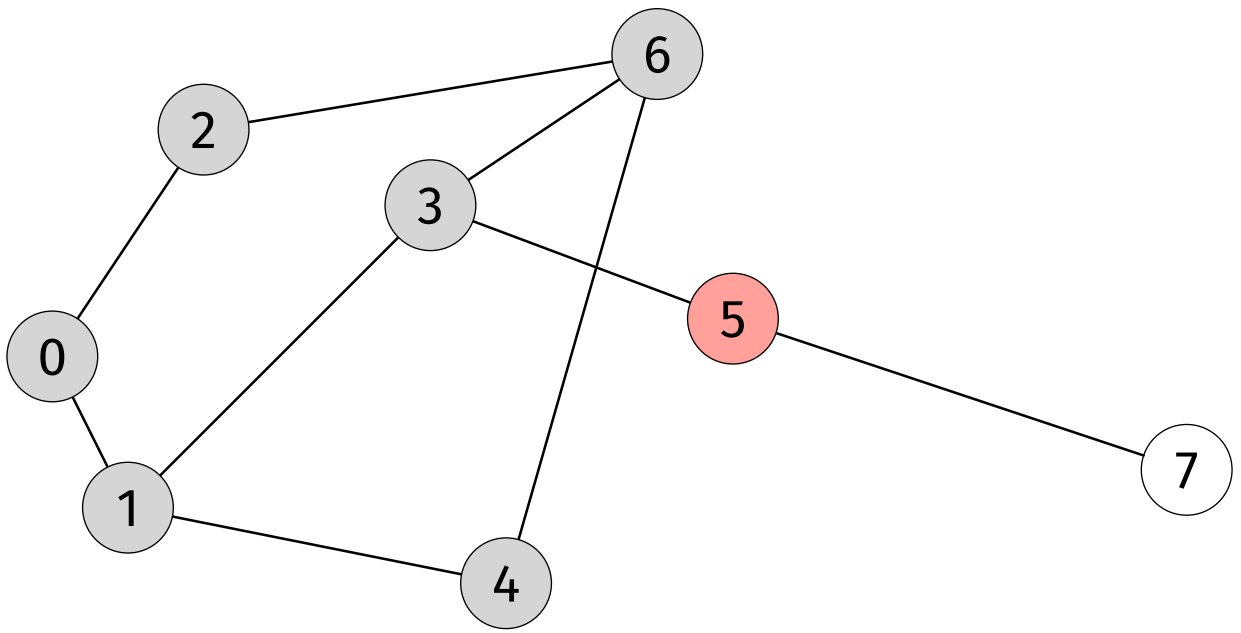
Bredden-först-sökning

5



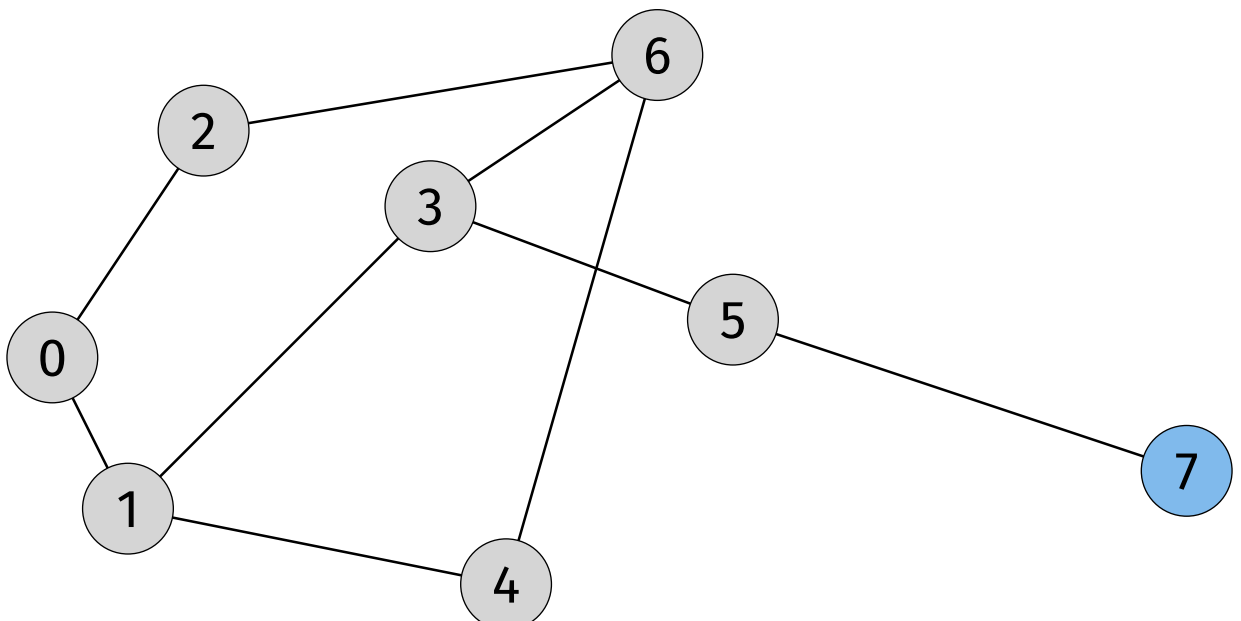
Bredden-först-sökning

5



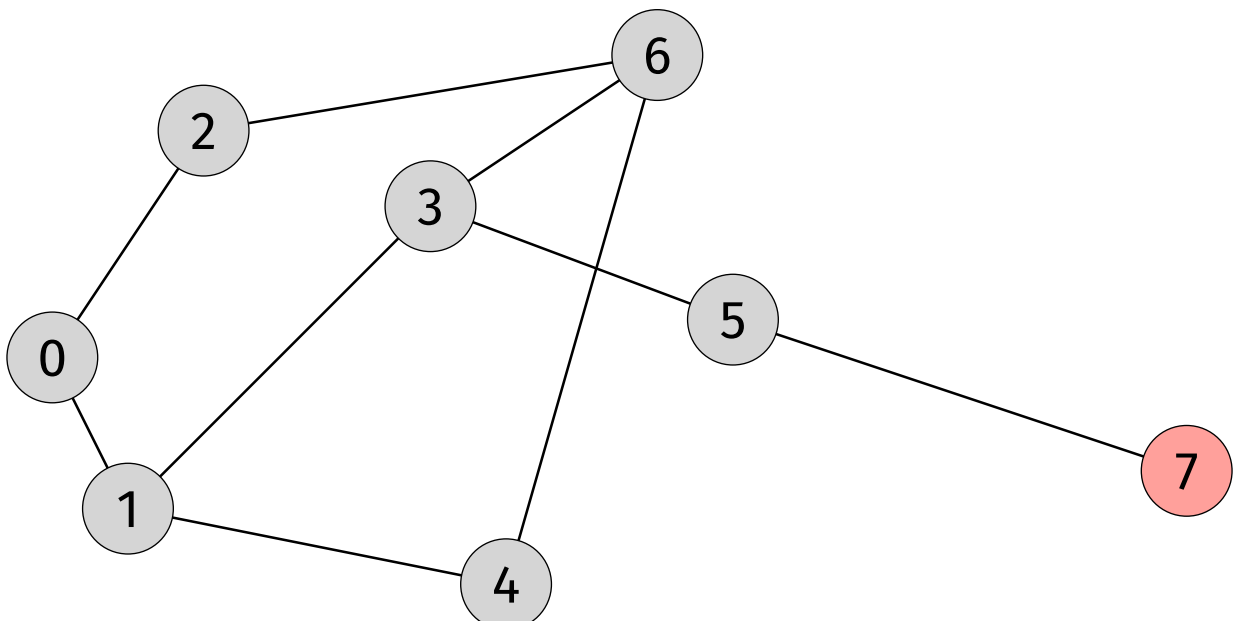
Bredden-först-sökning

7

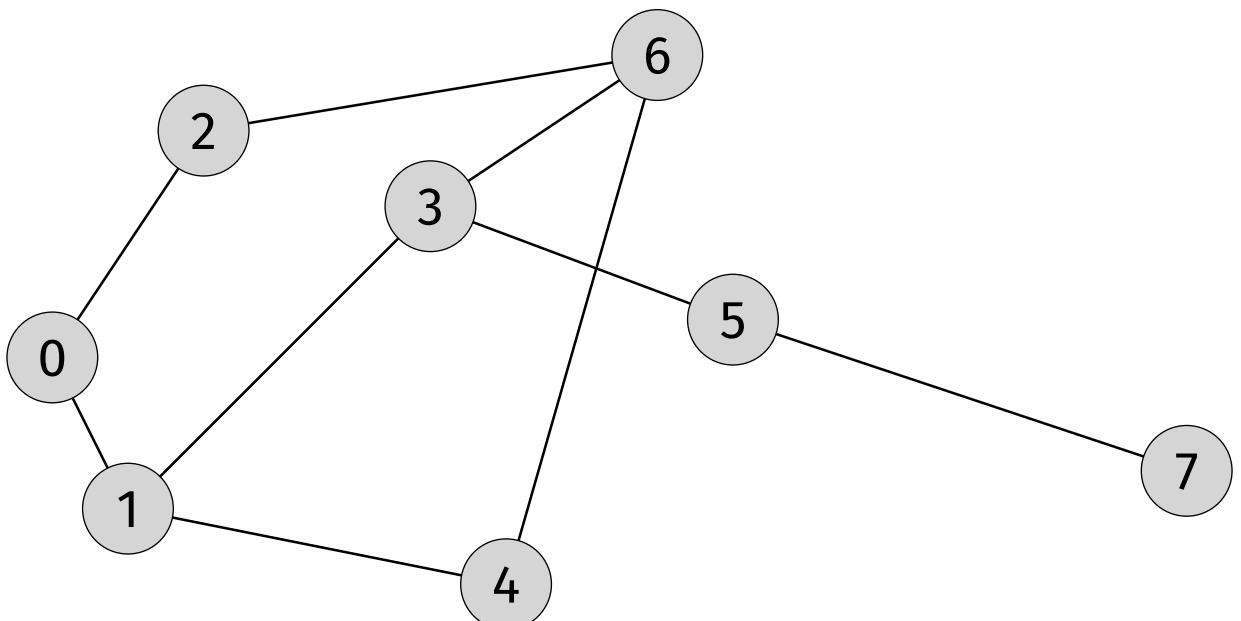


Bredden-först-sökning

7



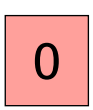
Bredden-först-sökning

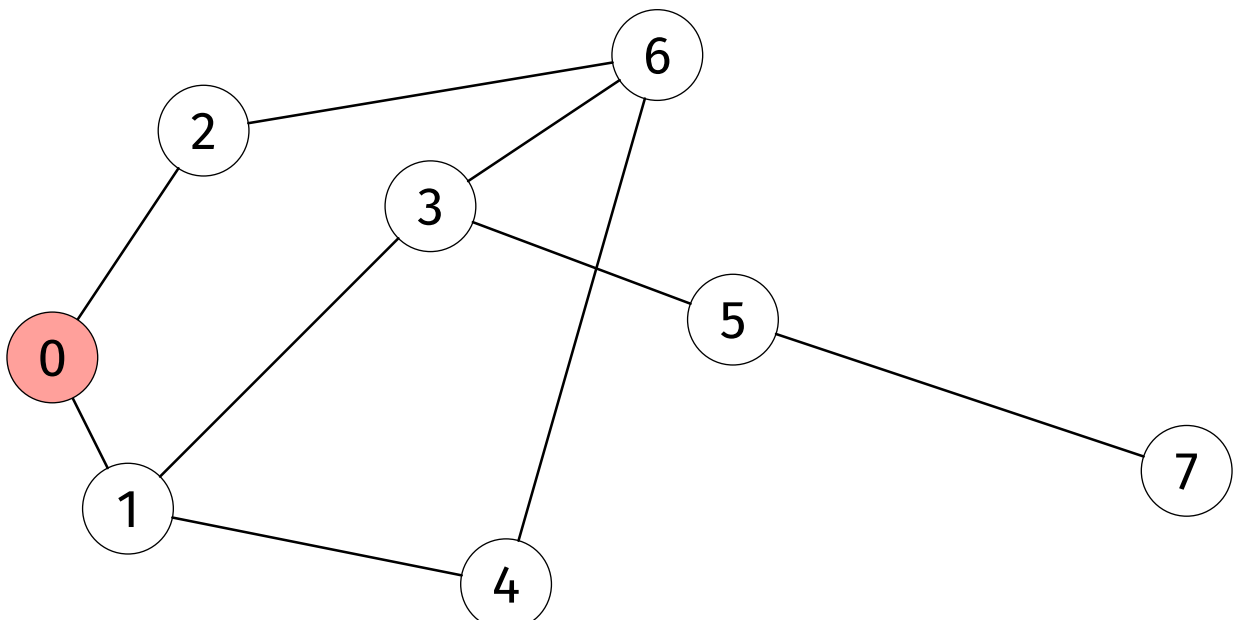


Djupet-först-sökning

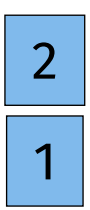
Depth first search (DFS)

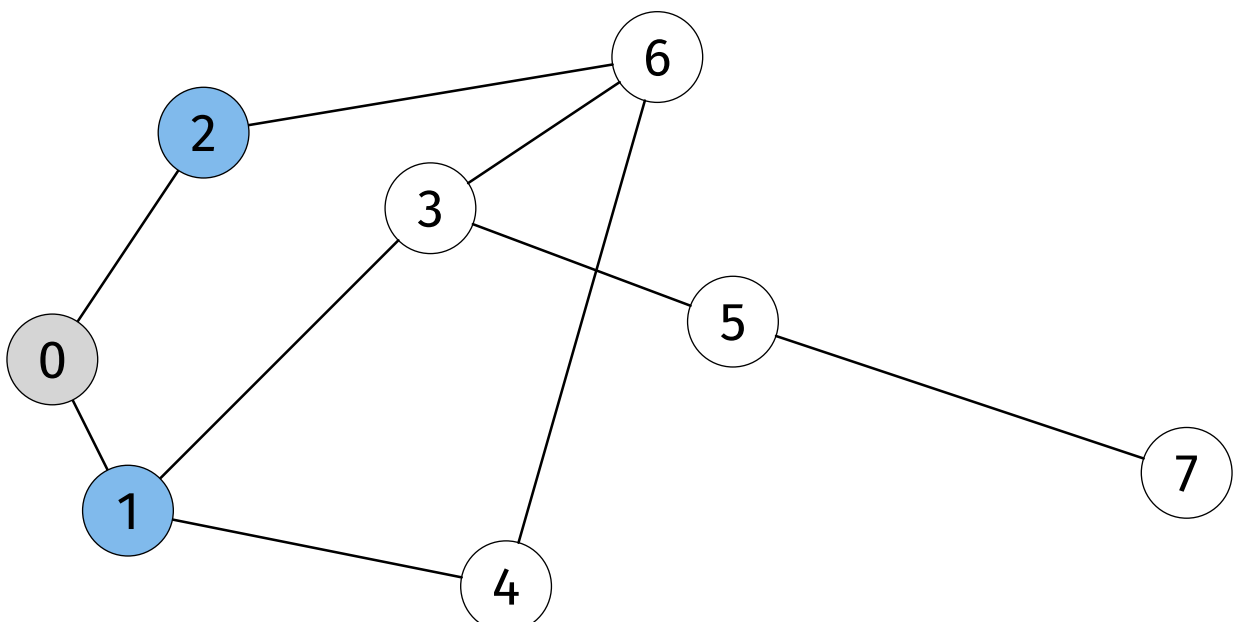
Djupet-först-sökning



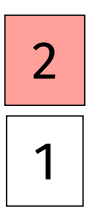


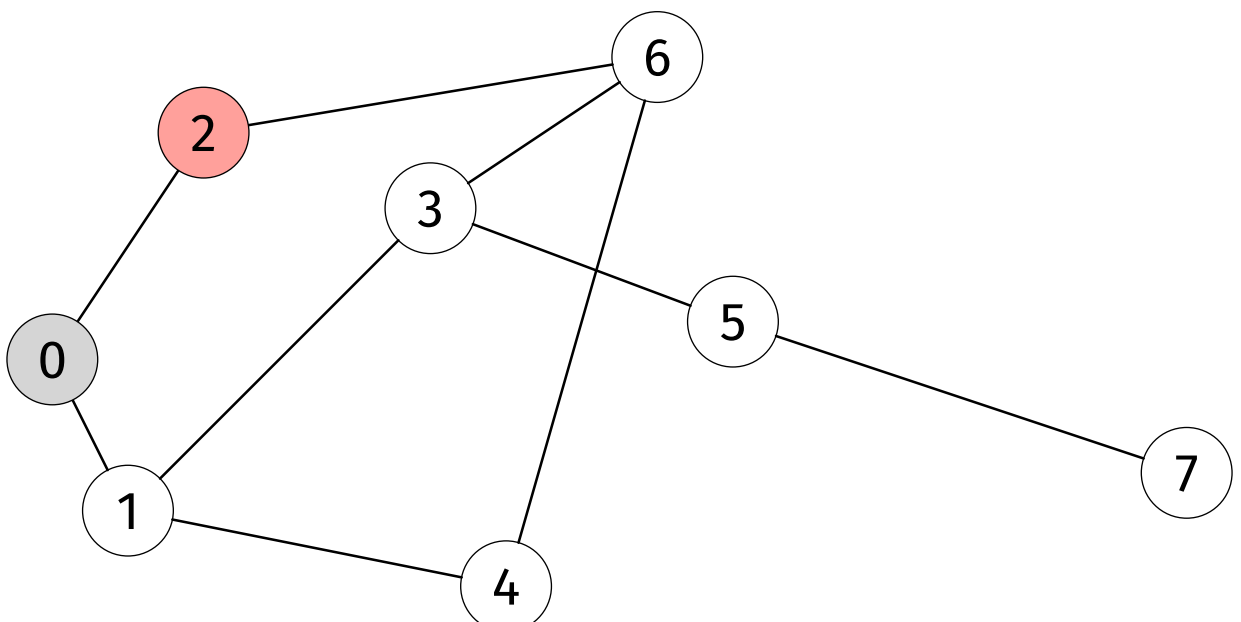
Djupet-först-sökning



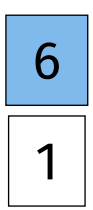


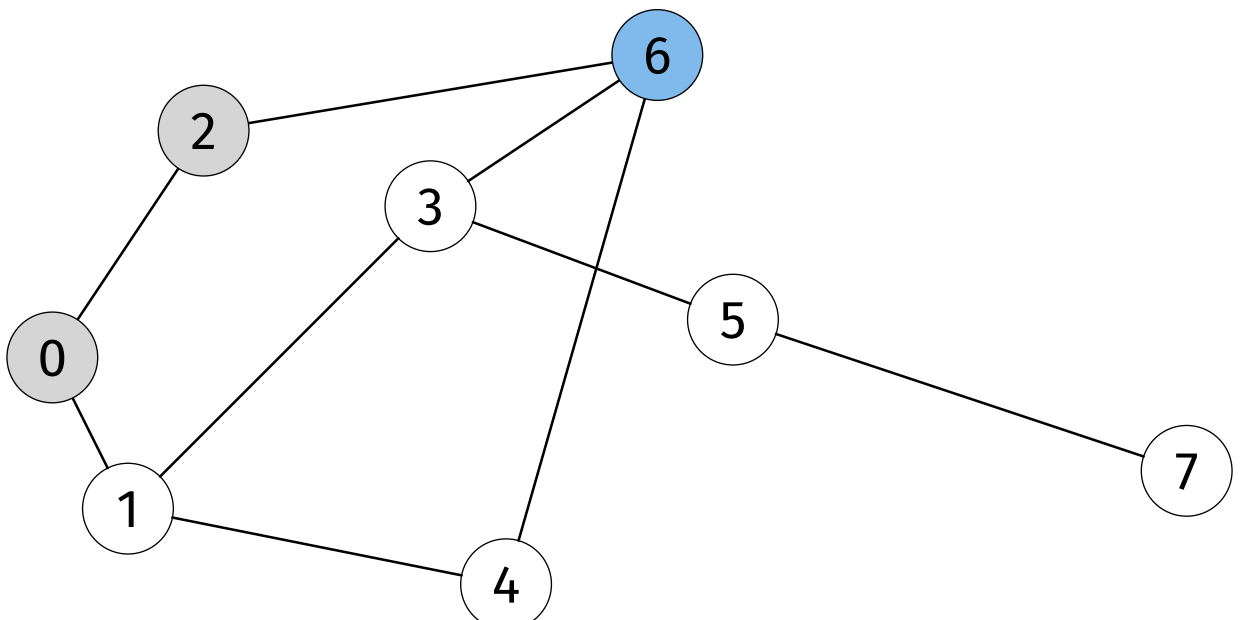
Djupet-först-sökning



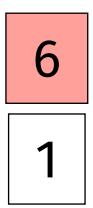


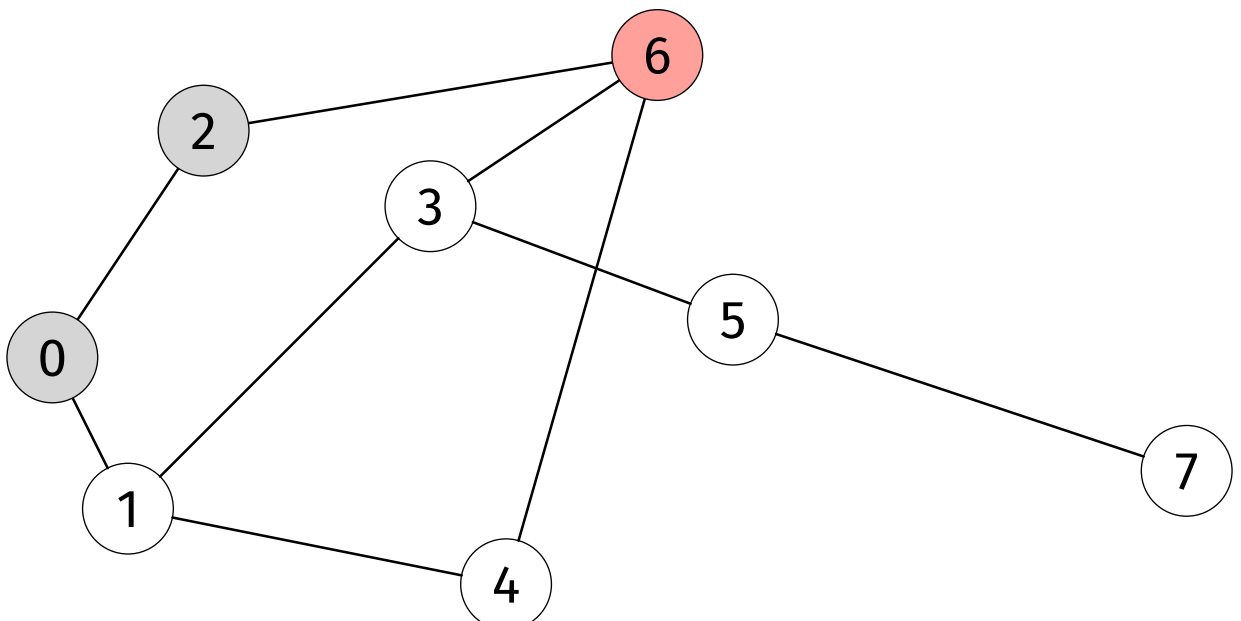
Djupet-först-sökning



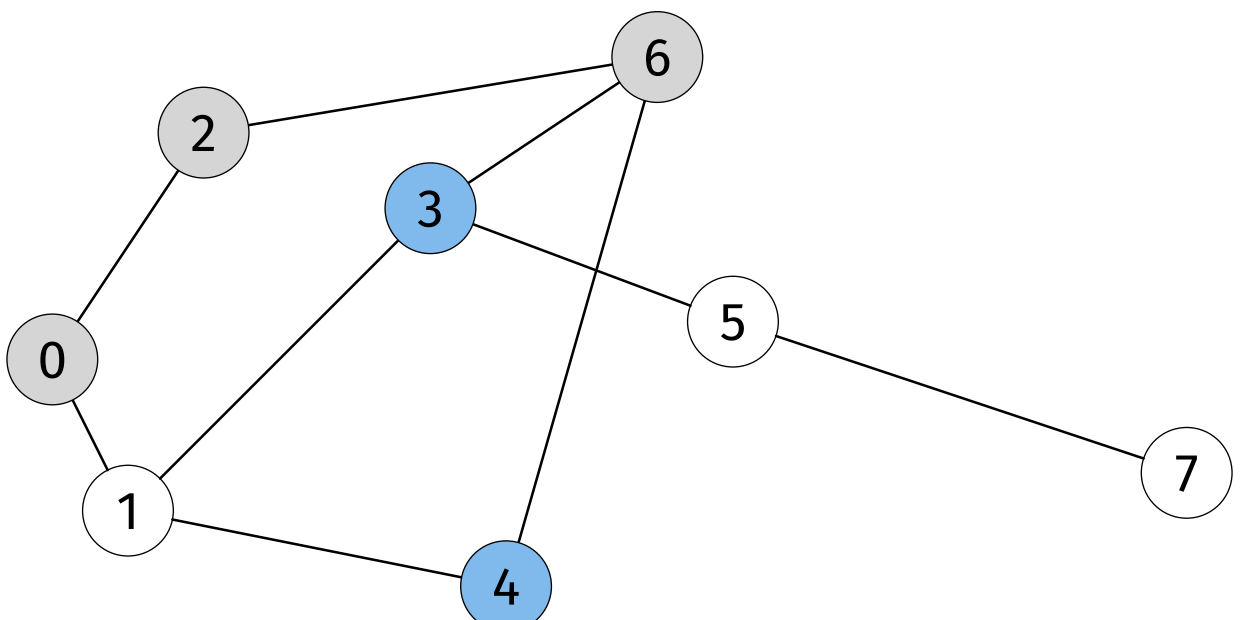


Djupet-först-sökning

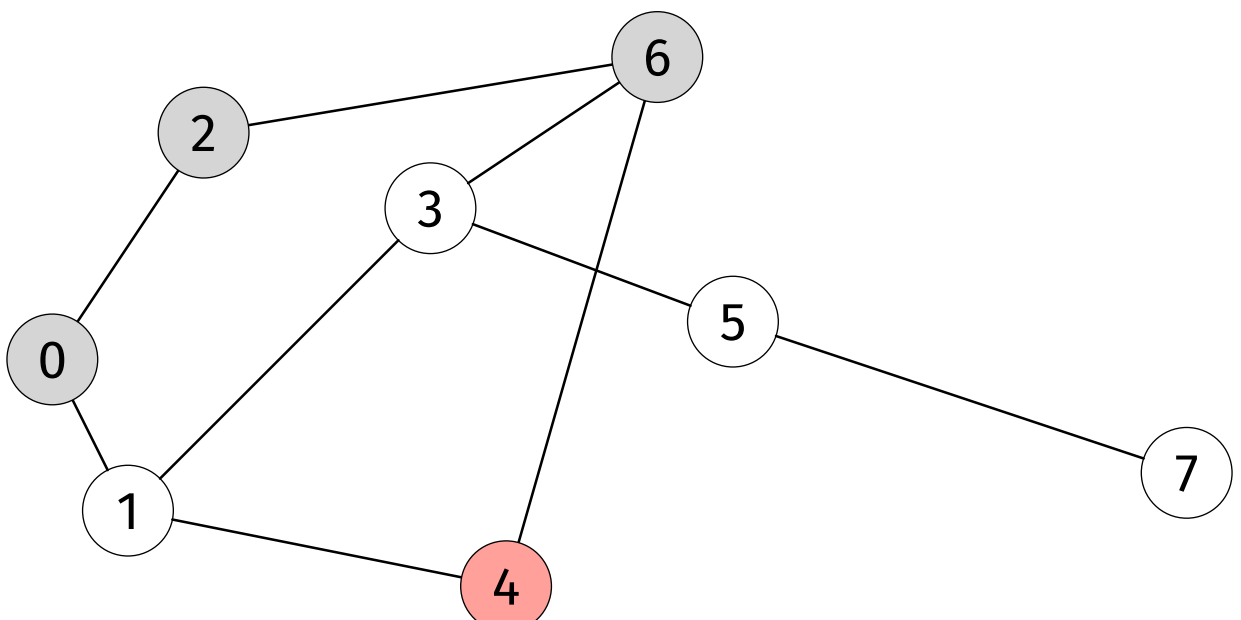




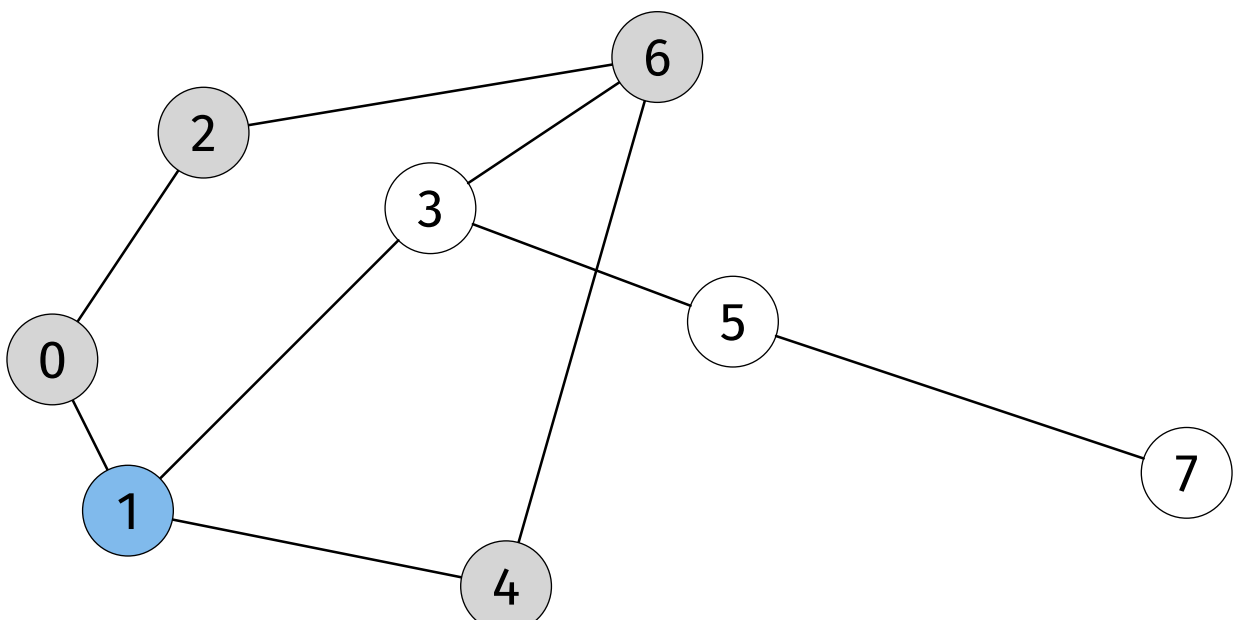
Djupet-först-sökning



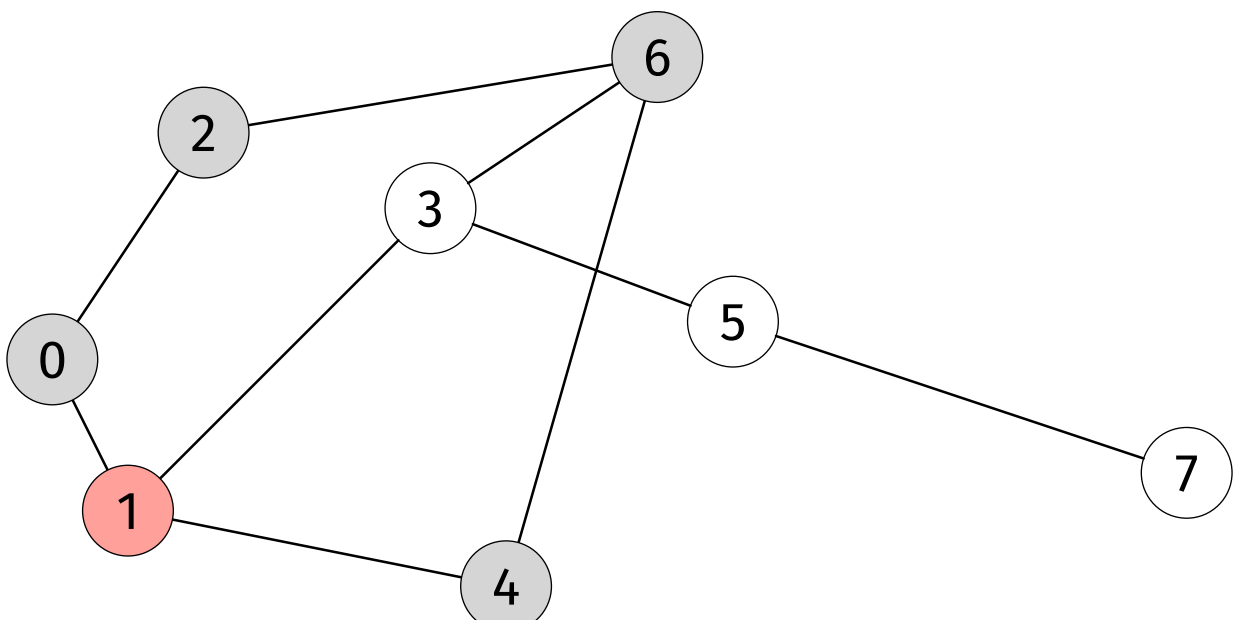
Djupet-först-sökning



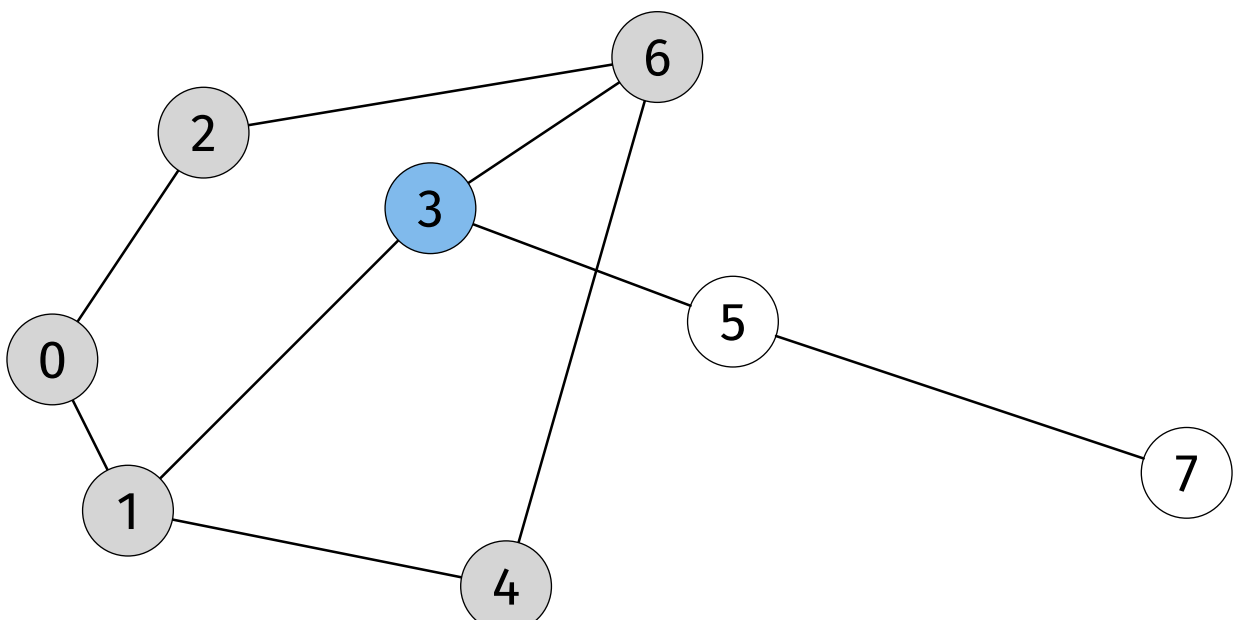
Djupet-först-sökning



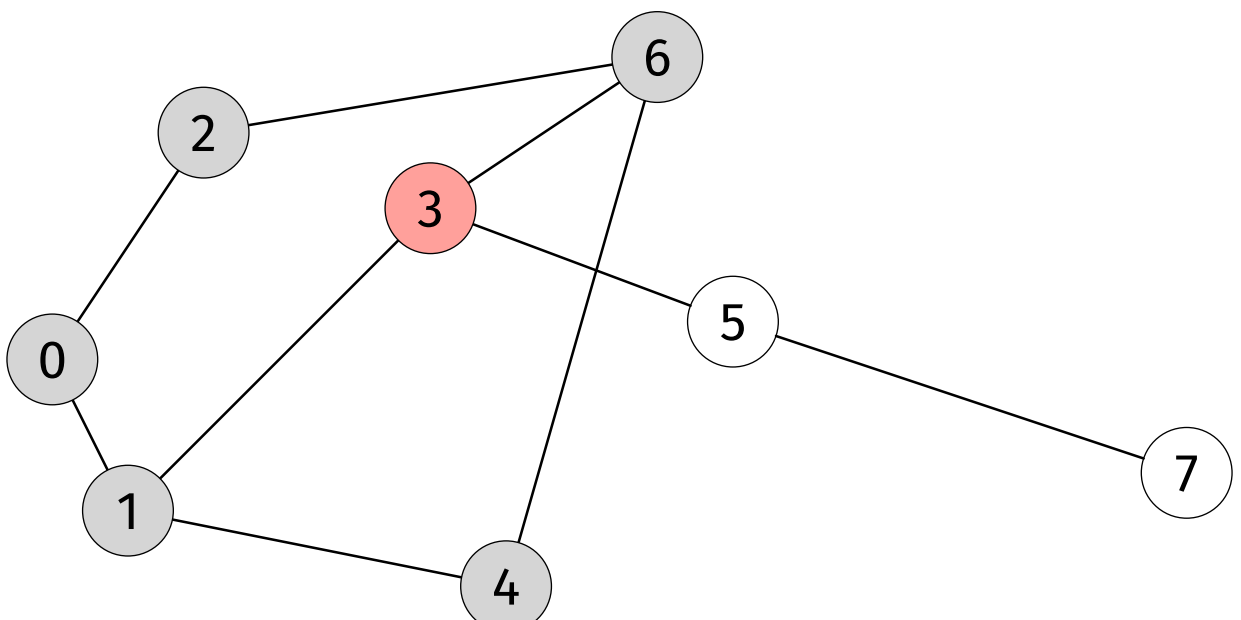
Djupet-först-sökning



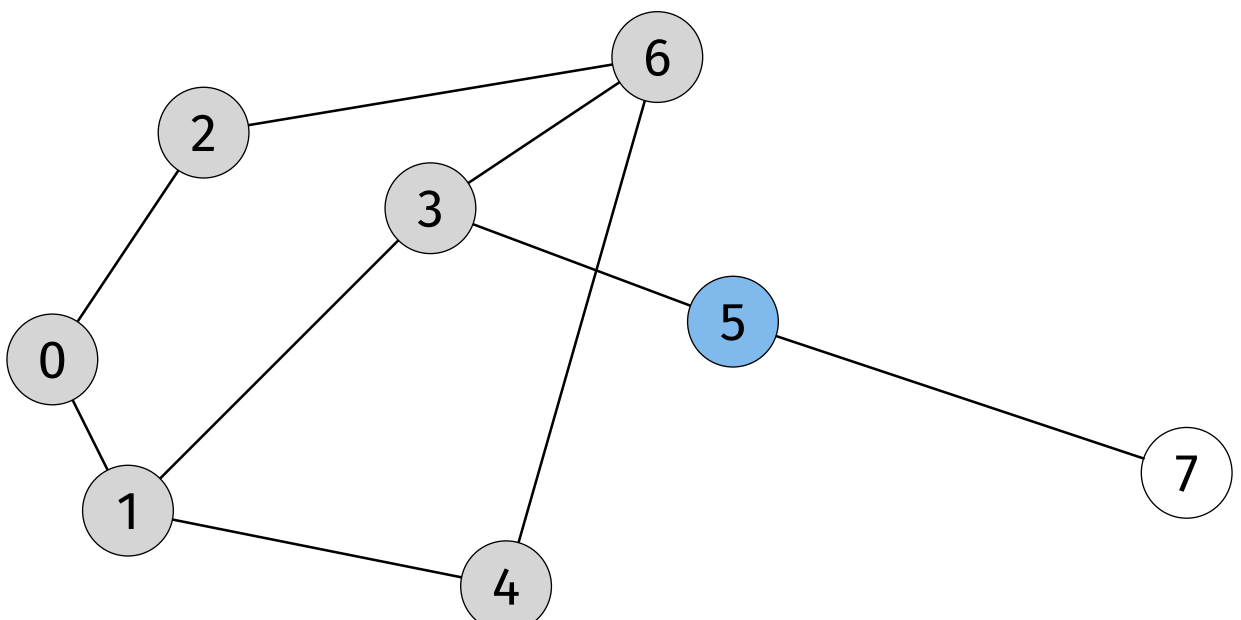
Djupet-först-sökning



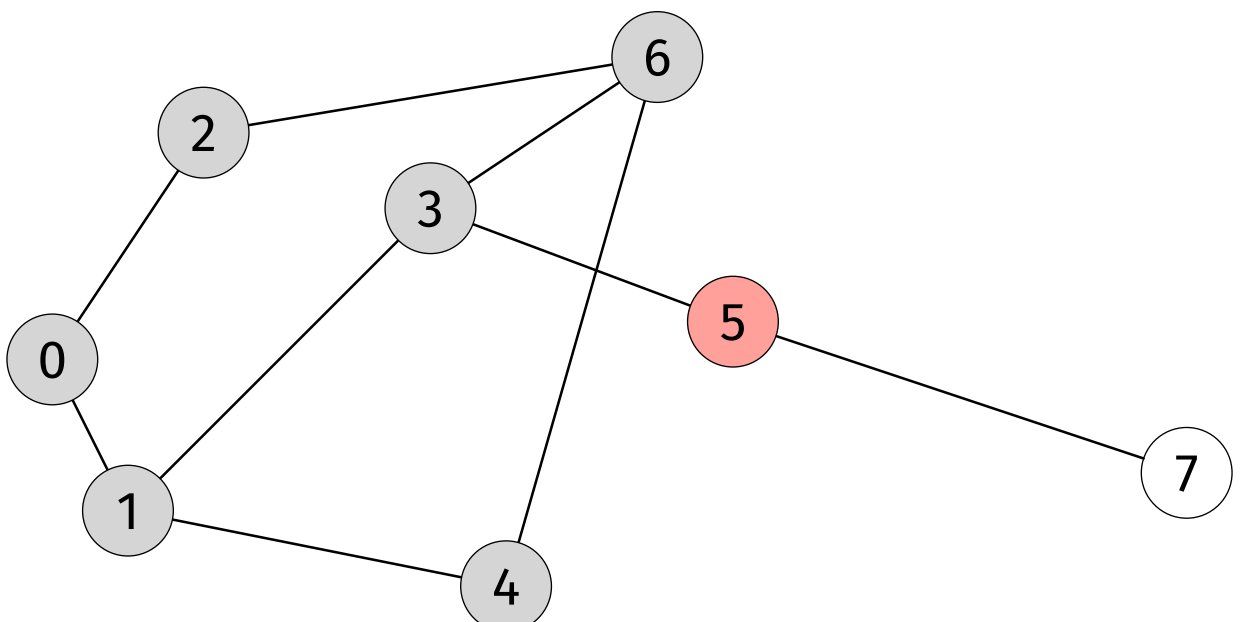
Djupet-först-sökning



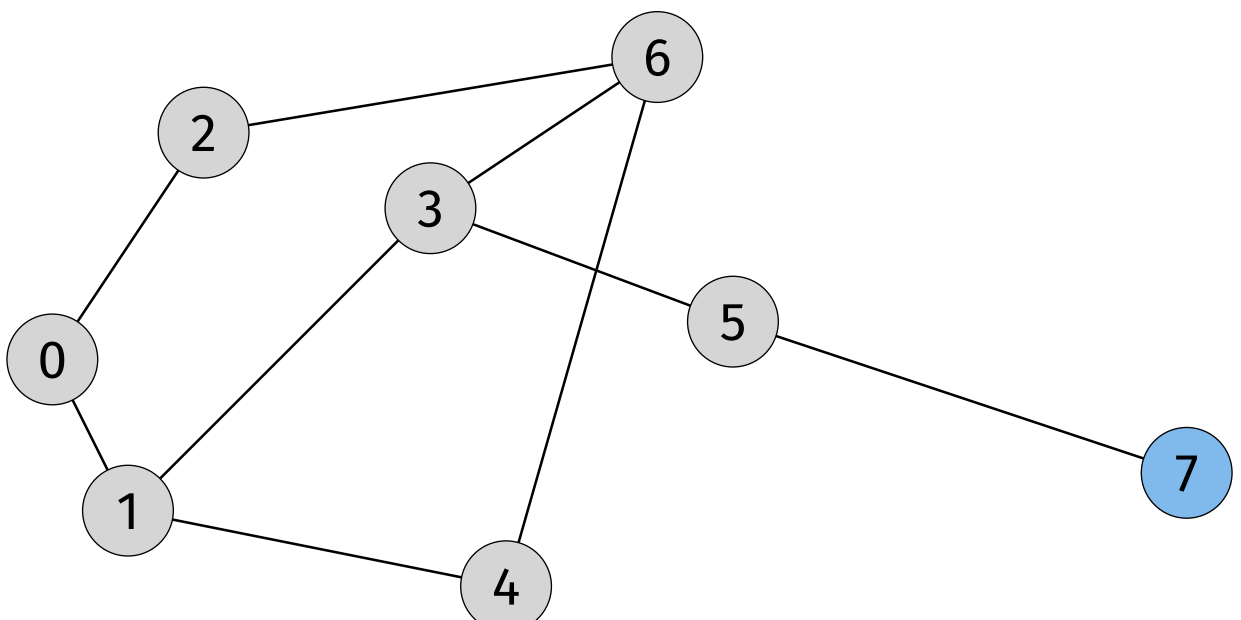
Djupet-först-sökning



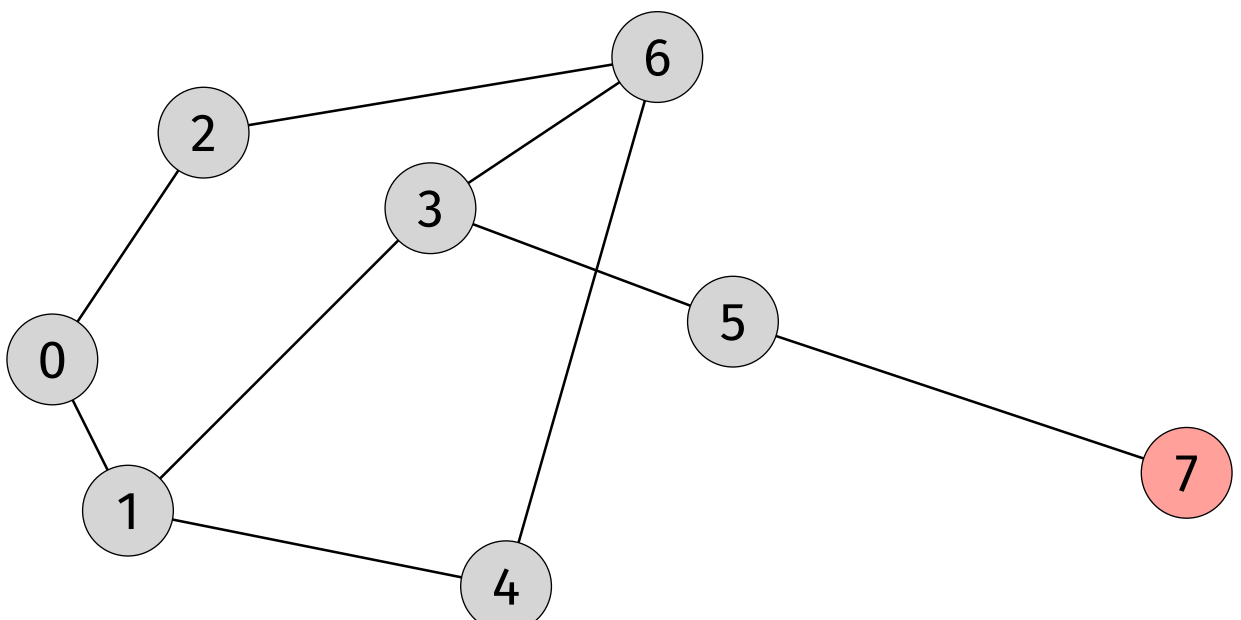
Djupet-först-sökning



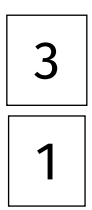
Djupet-först-sökning

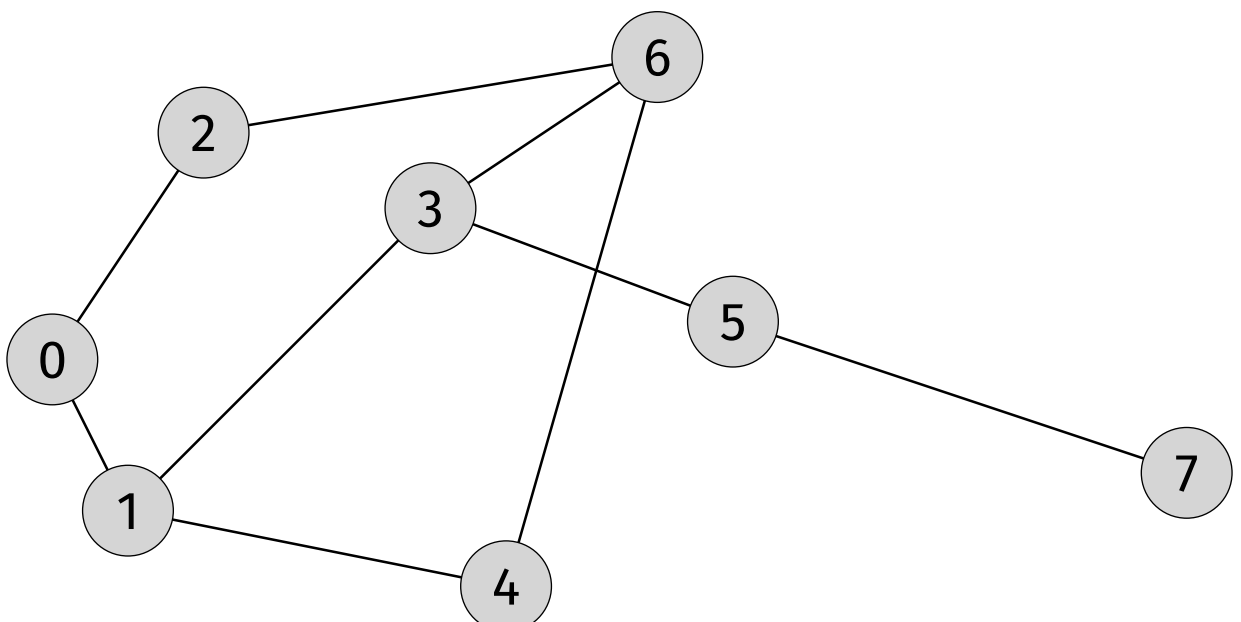


Djupet-först-sökning

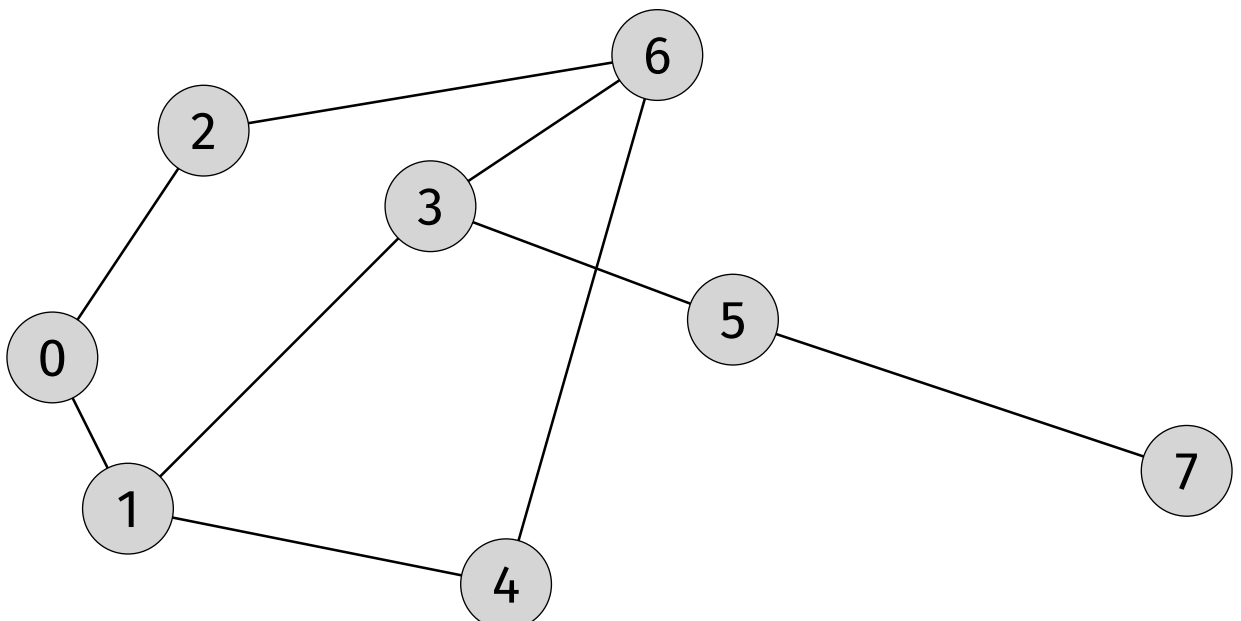


Djupet-först-sökning





Djupet-först-sökning



Problem:

Hitta kortaste stigen mellan två noder i en viktad graf

Dijkstras algoritm

- Spara avståndet till varje hörn (börja med ∞)
- Sätt avståndet till starthörnet till 0
- Medans det fortfarande finns obesökta hörn:
  - Välj ett obesökt hörn v med minsta avstånd och markera som besökt
  - (Om v är målet är vi klara)
  - Uppdatera grannarna till v om avståndet blir mindre via v
  - (Om vi vill konstruera kortaste stigen, sätt v som föregångare)
- Tidskomplexitet beror på datastruktur, en fibonacci heap ger $O(E + V \log V)$

- Negativa kanter (Bellman-Ford)

Kortaste stigen - Dijkstras algoritm

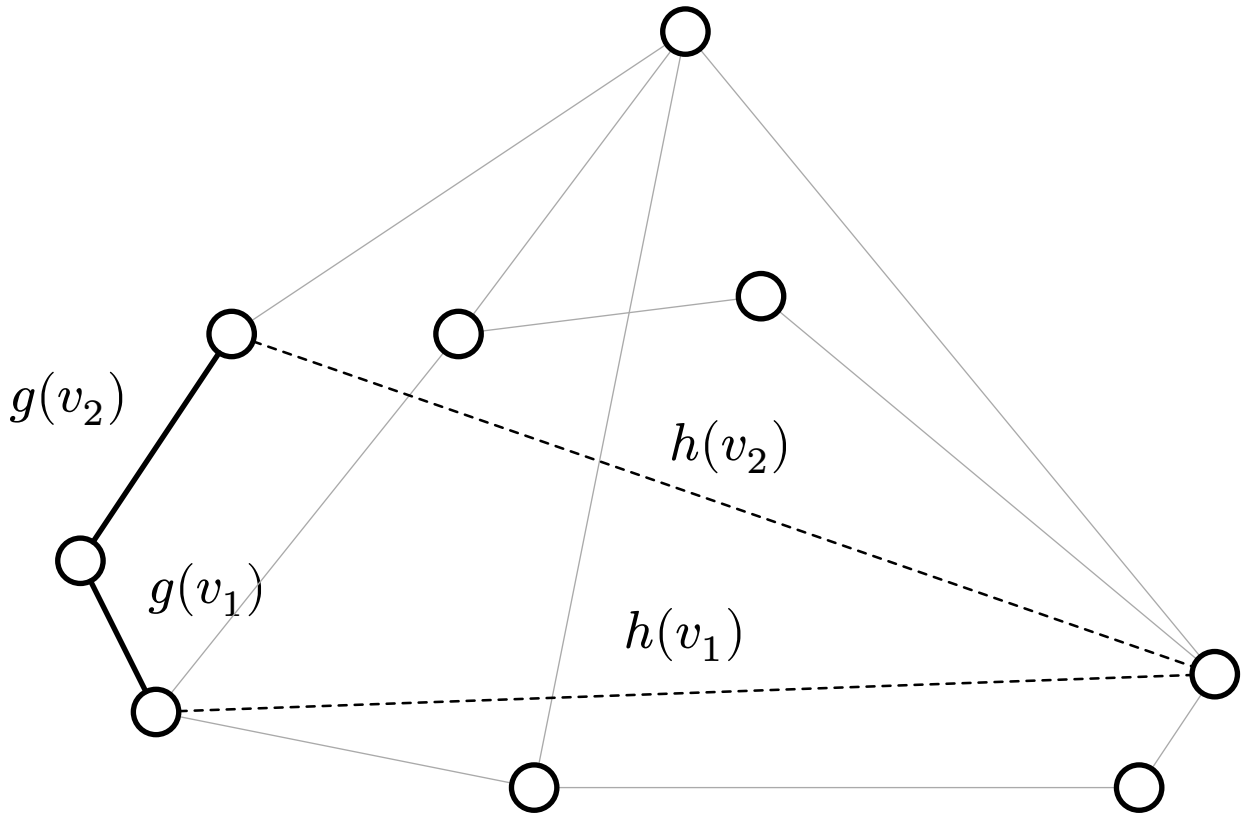

```
def dijkstra(graph, start, end):
    for vertex in graph.vertices:
        vertex.distance = math.inf
        vertex.visited = False
    start.distance = 0;

    while graph.has_unvisited():
        current = graph.closest_unvisited()
        current.visited = True
        if current == end:
            return end.distance
        for edge in graph.edges[current]:
            neighbour = edge.target
            new_distance = current.distance + edge.weight
            neighbour.distance = min(neighbour.distance, new_distance)
```

A* search

- Vi använder en heuristik/uppskattning av avståndet till målet.
- Vi väljer mellan alternativa stigar med hjälp av en kostnadsfunktion.

$$\underbrace{f(v)}_{\text{Kostnad}} = \underbrace{g(v)}_{\text{Avståndet från start}} + \underbrace{h(v)}_{\text{Uppskattat avstånd till målet}}$$

Kortaste stigen – A* search



$h(v)$ kan t.ex. vara ”fågelavståndet” till målet.

Kortaste stigen mellan *alla* hörn

Problem:

Hitta kortaste vägen mellan *alla* par i en viktad graf

Floyd-Warhalls algoritm

- Vi lägger ett tillåtet ”mellanhörn” i taget
- Vi tar fram en rekursion för funktionen $FW(i, j, k)$ som är kortaste avståndet mellan hörnen i och j där hörnen $\{1, 2, ..., k\}$ är tillåtna mellanhörn
- Tidskomplexitet: $O(V^3)$

En stig mellan i och j går antingen via k eller inte:

$$FW(i,j,k) = \min \begin{cases} FW(i,j,k-1) \\ FW(i,k,k-1) + FW(k,j,k-1) \end{cases}$$

$$FW(i,j,0) = \begin{cases} w & \text{om } (i,j,w) \in E \\ \infty & \text{annars} \end{cases}$$

All pairs shortest path - Floyd-Warshall


```
def floyd_warshall(graph):
  n = len(graph.vertices)
  distance = [[math.inf] * n] * n

  for vertex in graph.vertices:
    distance[vertex][vertex] = 0

  for edge in graph.edges:
    distance[edge.start][edge.end] = edge.weight

  for k in graph.vertices:
    for a in graph.vertices:
      for b in graph.vertices:
        distance[a][b] = min(distance[a][b], distance[a][k] + distance[k][b])
```

Träd

Minimala spännande träd

- *Träd* – en graf utan cykler
- *Spännande träd* – ett träd med alla hörn från V och bara kanter från E
- *Minimalt spännande träd* – ett spännande träd som minimerar den totala kantvikten

Problem:

Hitta det minimala delträdet som spänner upp en graf

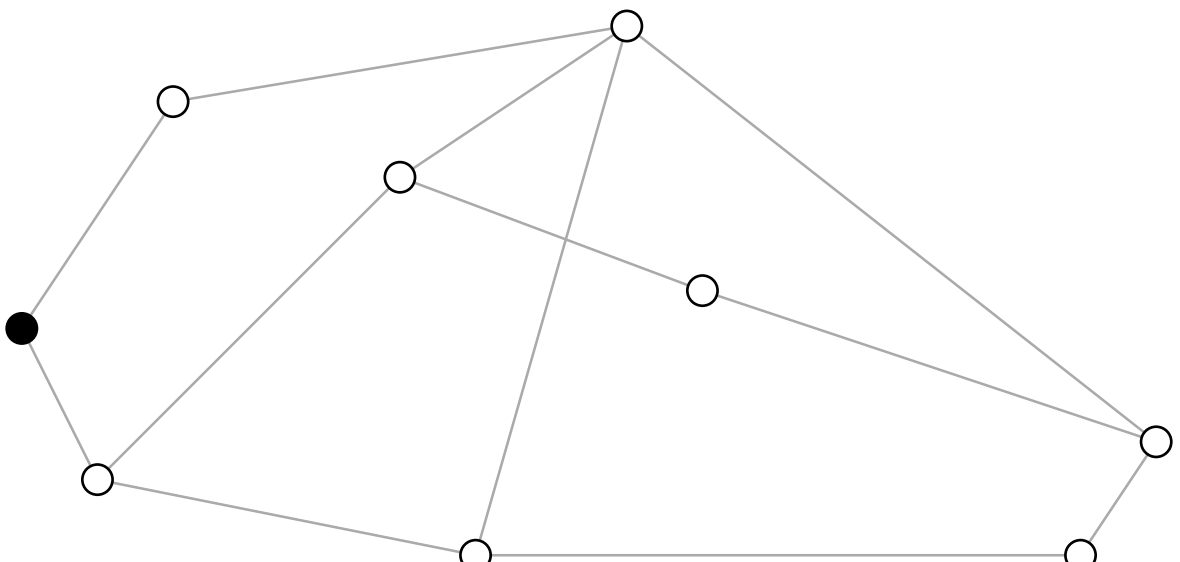
Prims algoritm

- Lägg till ett godtyckligt hörn i trädet
- Tills alla hörn är i trädet:
  - Hitta den kant e med minsta vikt som ansluter ett nytt hörn v till trädet
  - Lägg till e och v i trädet

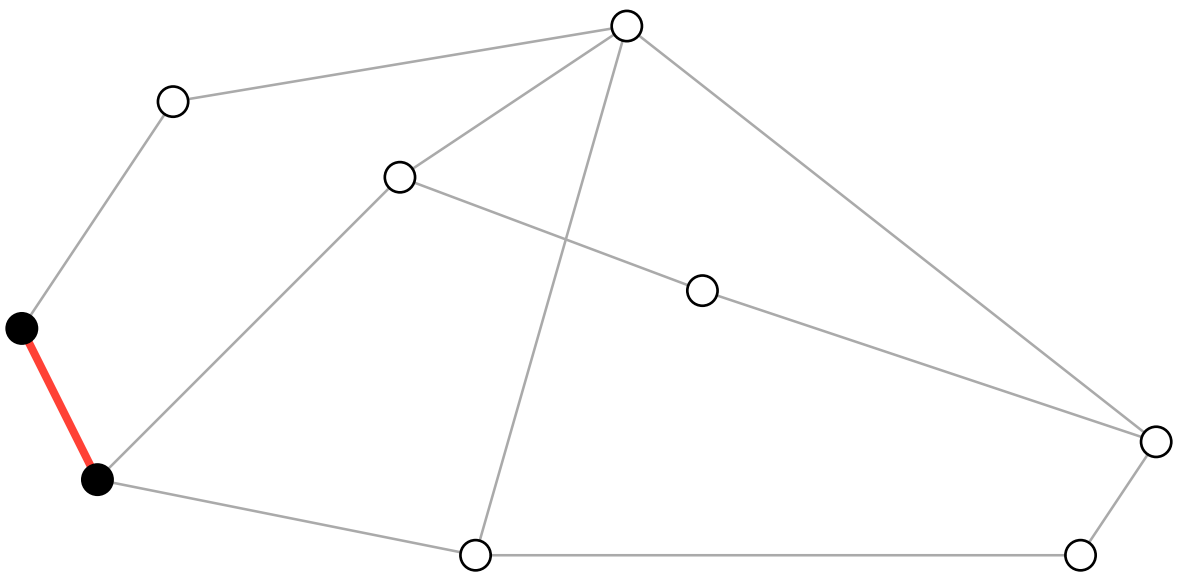
Tidskomplexitet beror på datastruktur:

- Grannmatris/grannlista $O(V^2)$
- Kantlista/grannlista + binary heap $O(V \log E)$
- Kantlista/grannlista + fibonacci heap $O(E + V \log V)$

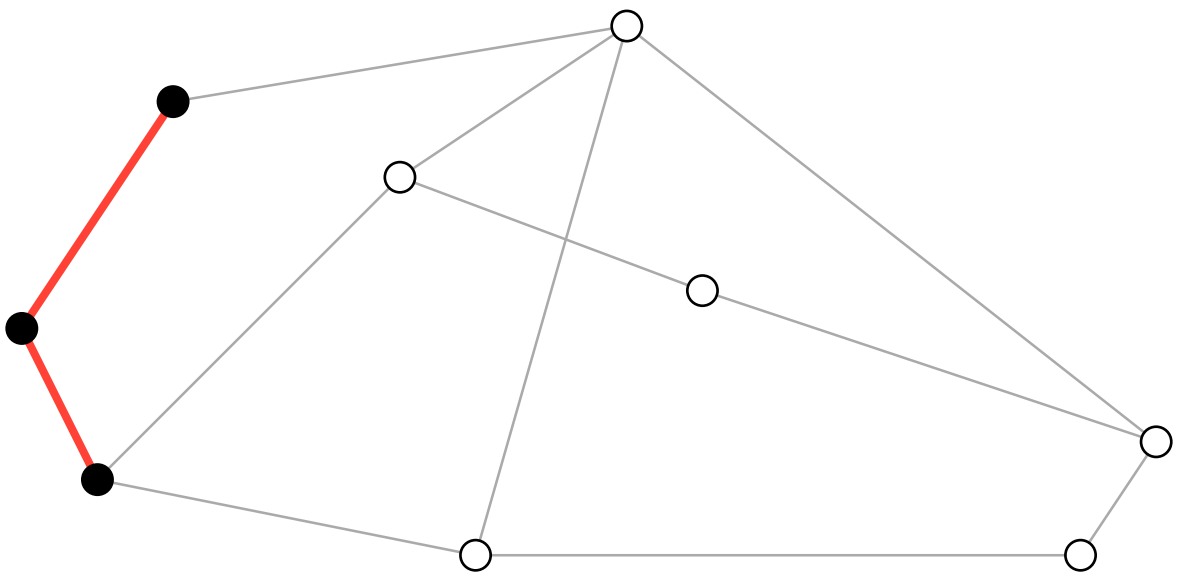
Minimala spännande träd - Prims algoritm



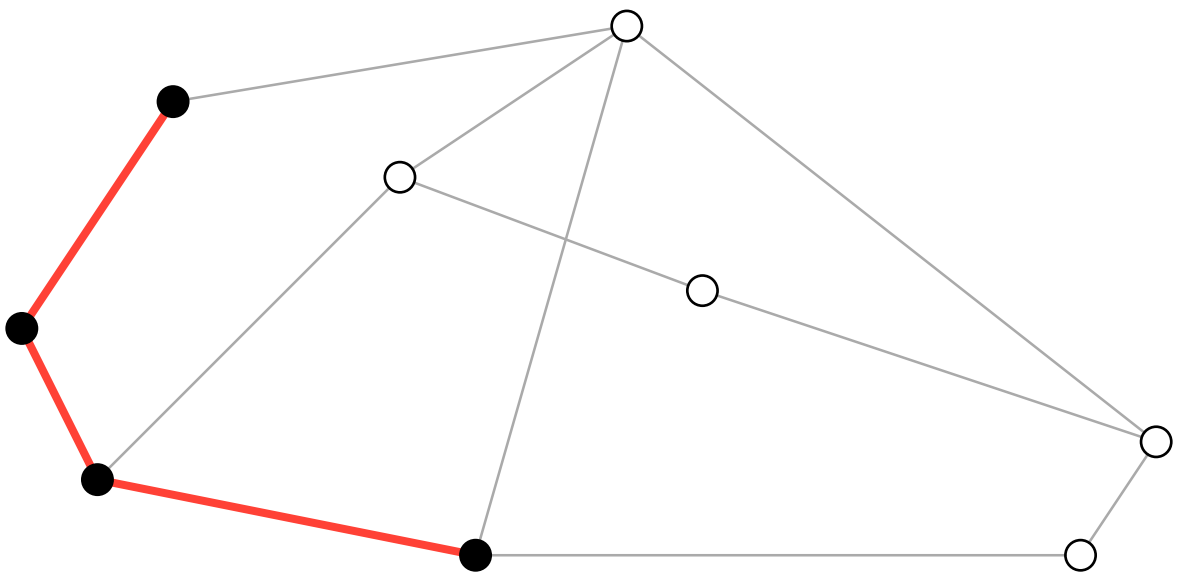
Minimala spännande träd - Prims algoritm



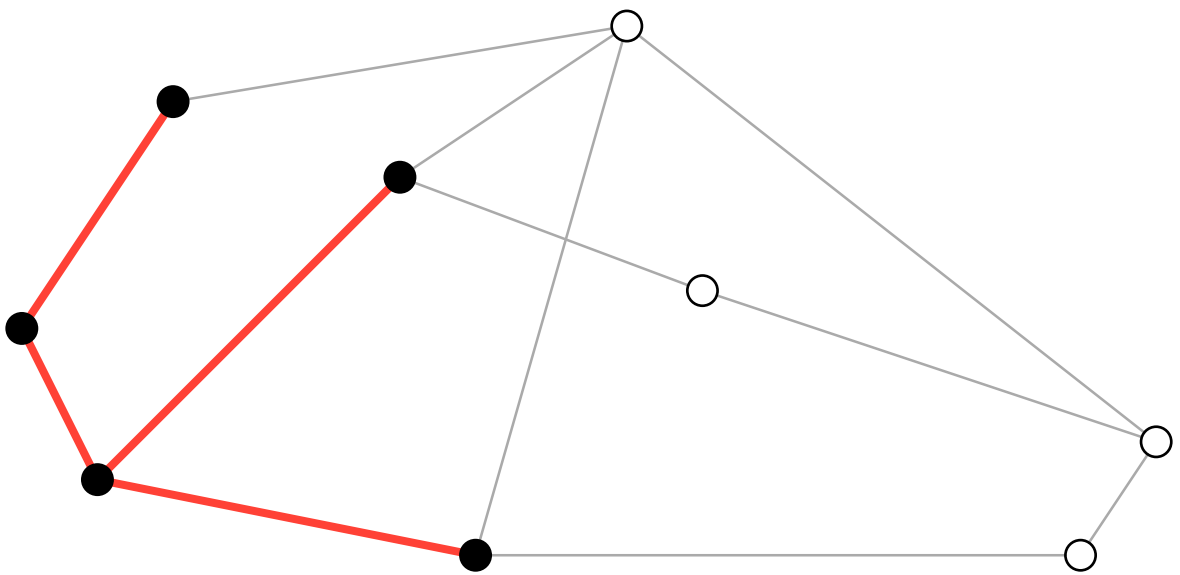
Minimala spännande träd - Prims algoritm



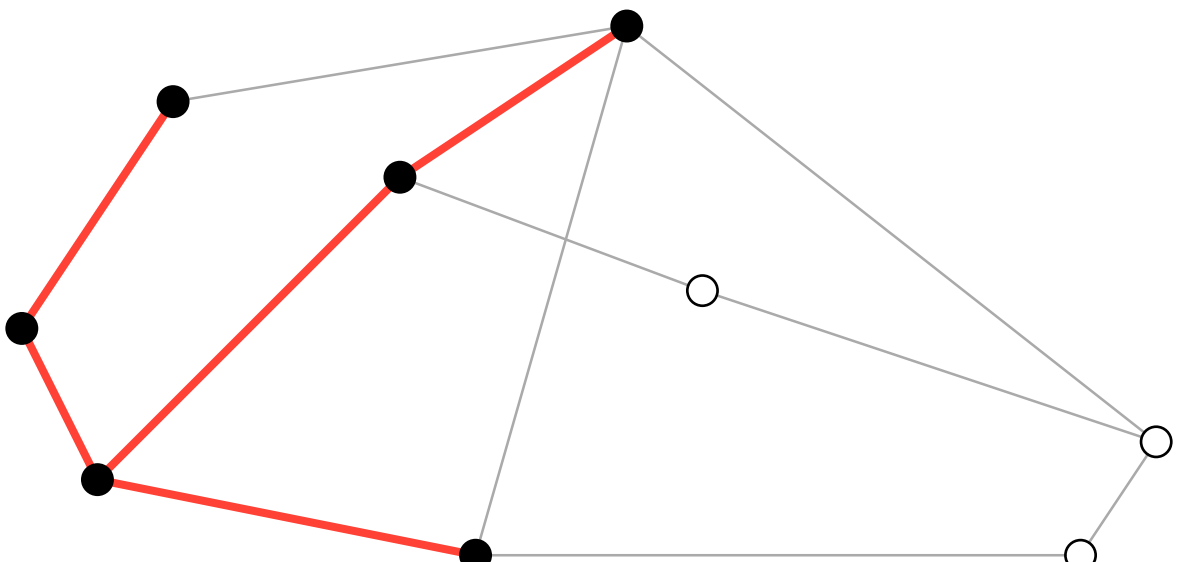
Minimala spännande träd - Prims algoritm



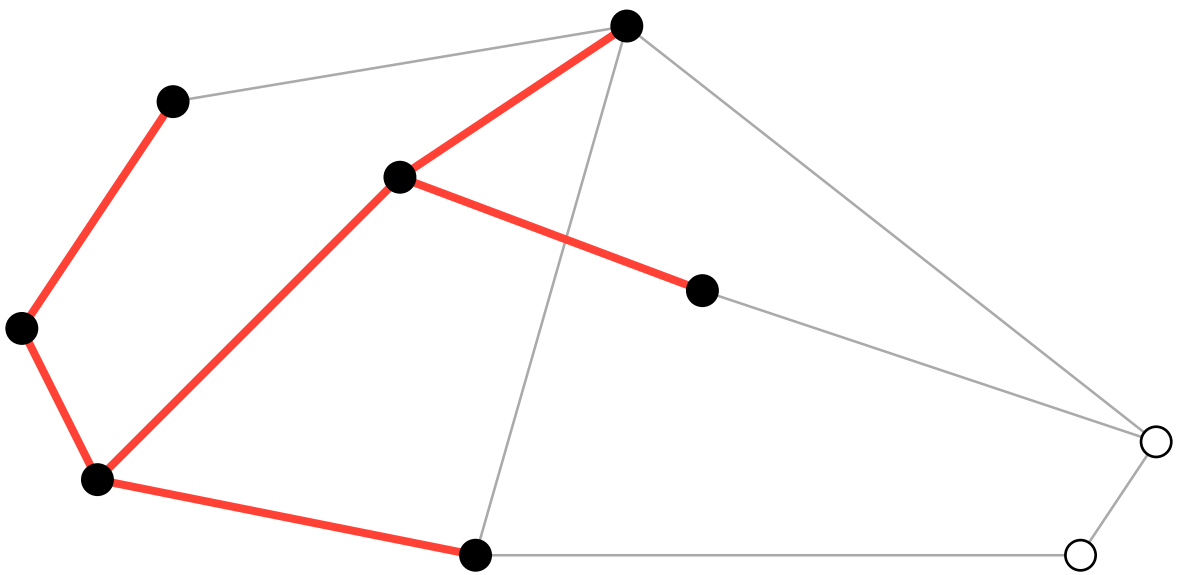
Minimala spännande träd - Prims algoritm



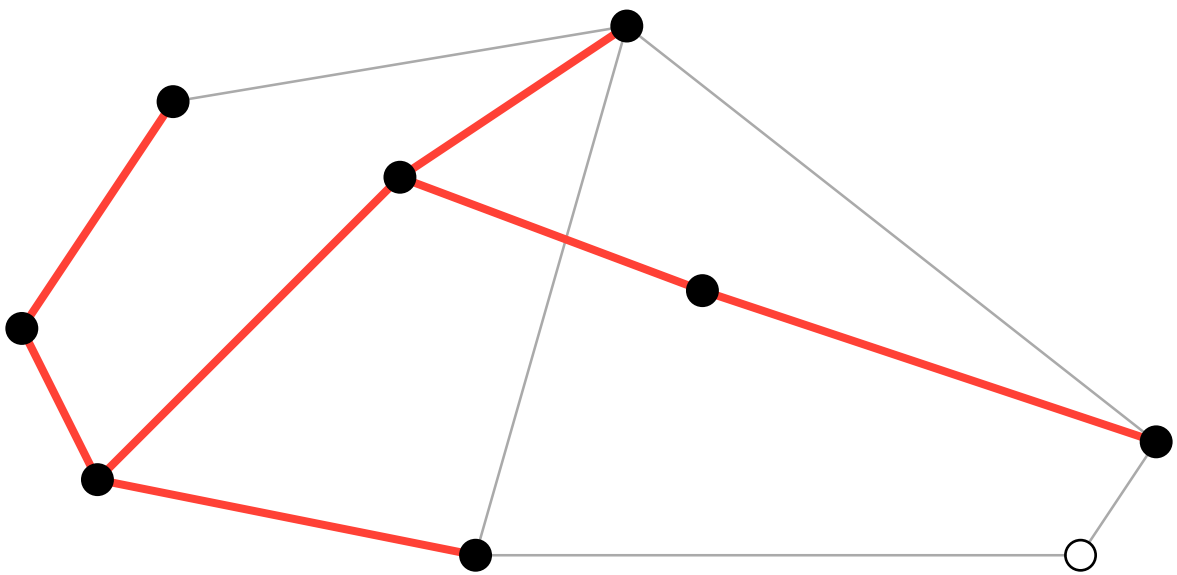
Minimala spännande träd - Prims algoritm



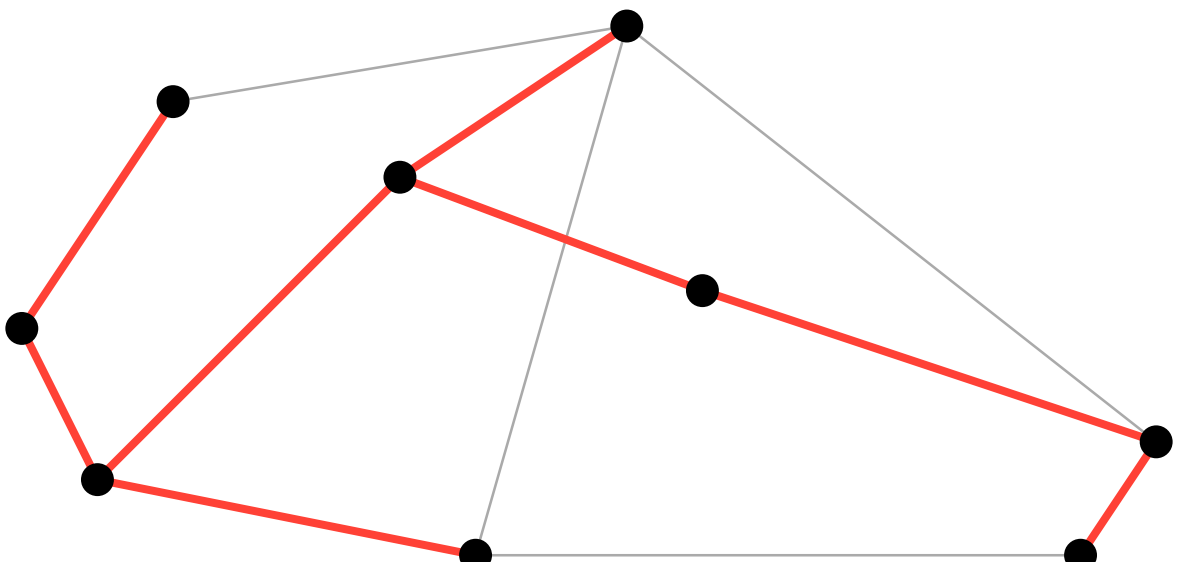
Minimala spännande träd - Prims algoritm



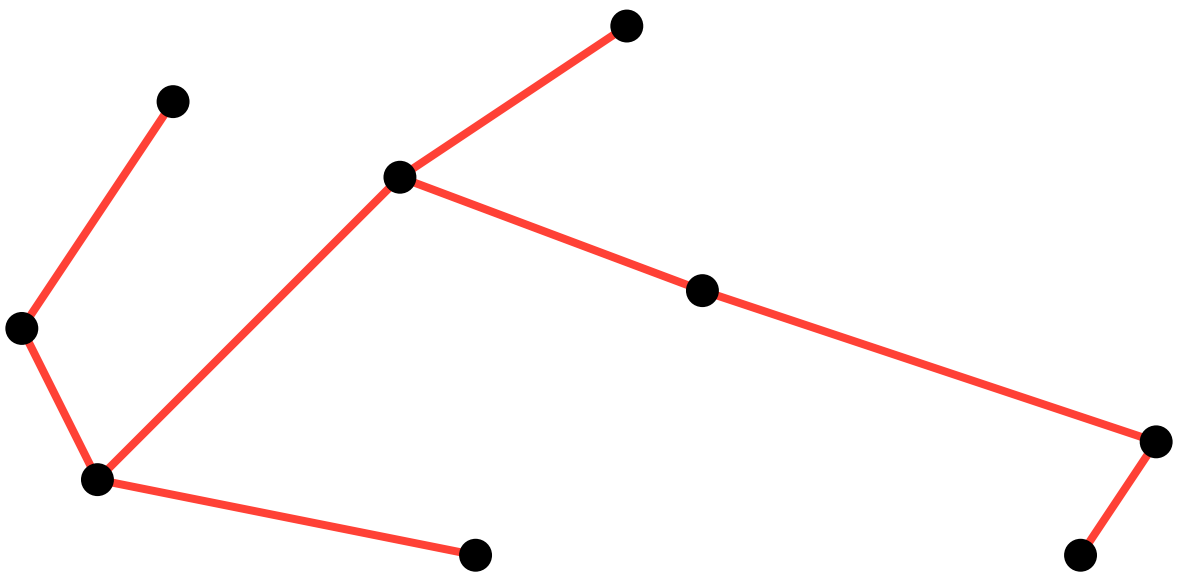
Minimala spännande träd - Prims algoritm



Minimala spännande träd - Prims algoritm



Minimala spännande träd - Prims algoritm



Kruskals algoritm

- Lägg hela tiden till kanten med minsta vikt så att inga cykler uppstår
- Fungerar även för osammanhängande grafer
- Ger en minimal spännande *skog*
- Kan implementeras med datatstrukturen *Union-Find*

Potenser av grannmatriser

- En grannmatris A kan kan tolkas som en matris av element A_{ij} som ger antalet stigar från i till j av längd ett
- Om vi tar A till en potens n så är elementet $[A^n]_{ij}$ antalet möjliga vandringar från i till j med n steg

Intressant om grannmatriser

Potenser av grannmatriser

- Matrismultipliktion görs oftast i vanlig algebra, dvs kroppen $(\mathbb{R}, +, \cdot)$
- Vi kan använda andra algebror, t.ex. *min-plus algebra*: $(\mathbb{R} \cup \{\infty\}, \min, +)$
- Om vi då har en viktad graf med avstånd så är $[A^n]_{ij}$ avståndet för den kortaste vandringen från i till j med n steg
- Floyd-Warshall beräknar $\min_{k=1}^{N} A^k$ med dynamisk programmering (för en $N \times N$-matris A och elementvis $\min$)

NP-svåra grafproblem

Det finns flera NP-svåra problem för grafer:

- Graffärgning
  - *Kan en graf (V, E) hörnfärgas med k färger så att inga hörn som delar en kant har samma färg?*
- Hamiltoncykel
  - *Finns det en cykel som passerar varje hörn exakt en gång?*
- Handelsresandeproblemet
  - *Vad är den kortaste cykeln som passerar varje hörn exakt en gång?*

Läxa

Kattis-uppgifter