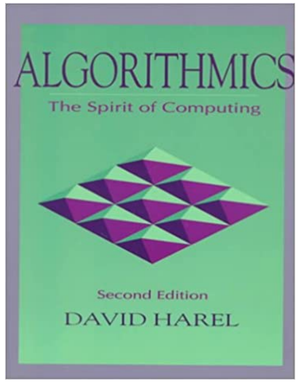
Grafer og bredde-først søgning

Diskret Matematik og Formelle Sprog
Københavns Universitet, marts 2023

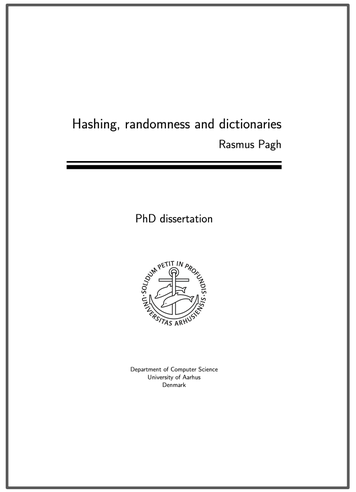
Rasmus Pagh

Slides med lyseblå baggrund er baseret på slides af Kevin Wayne

Lidt om mig



- Født i København i 1975
- Fik første computer i 1985, med 16 kb RAM
- Syntes om matematik i gymnasiet, læste bog om algoritmer
- Læste matematik og datalogi i Århus
- Ph.d-afhandling om hashing-algoritmer i 2002
- Arbejdede på ITU 2002-2020 IT UNIVERSITY OF COPENHAGEN
- På Google Research 2019-2020 Google
- Core researcher på BARC siden 2017
- Professor på DIKU siden 2020



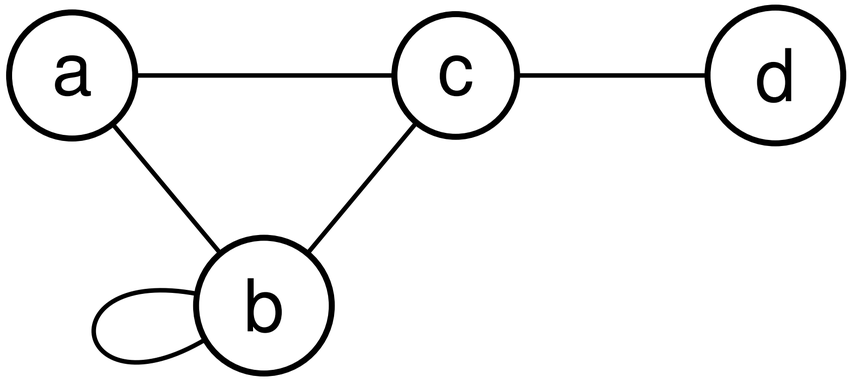
Grafer

- Matematisk set består en graf $G = (V, E)$ af en mængde V af knuder forbundet af en mængde E af kanter.

Eksempel:

$$V = \{a, b, c, d\}$$

$$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}, \{b\}\}$$



Kan fx modellere *venskaber* i et socialt netværk.

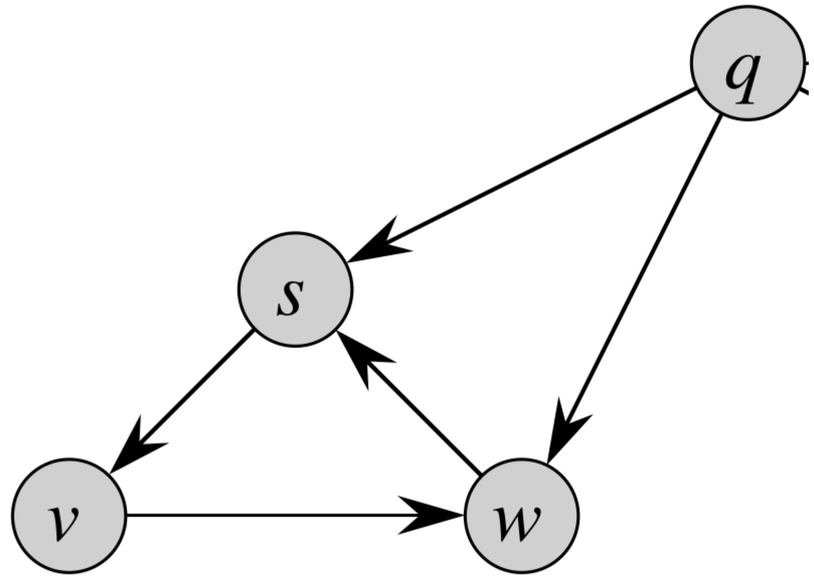
Orienterede grafer

Kanterne er ordnede tupler, dvs. har en retning.

$$G' = (V', E')$$

$$V' = \{s, q, v, w\}$$

$$E' = \{(q, s), (q, v), (w, s), (s, v), (v, w)\}$$

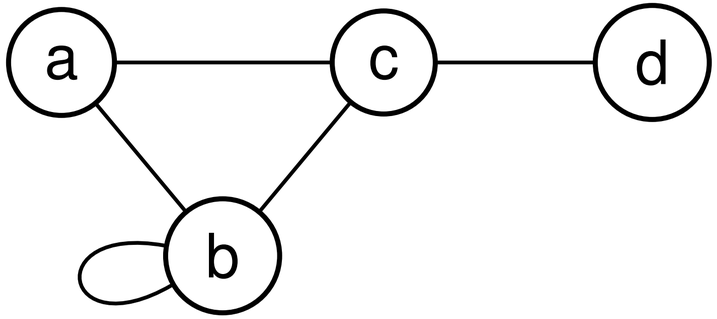


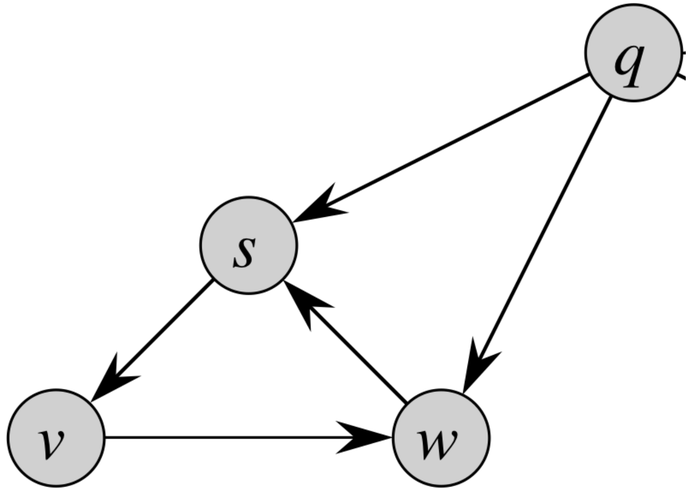
Kan fx modellere *følgere* i et socialt netværk.

Grafbegreber på engelsk og dansk

Se “Noter om grafer” på Absalon og CLRS appendix B.4

ENG	DAN
graph	(uorienteret) graf
node, vertex	knude
edge	kant
arrow	orienteret kant
adjacent, neighbors	naboer
cycle	kreds
path	sti
degree	grad
connected	forbundne
tree	træ
digraph	orienteret graf





Grafer kan modellere mange områder

Anvendelse	Knude	Kant
kommunikation	telefon, computer	et kabel
kredsløb	transistor, register, processor	ledning
mekanik	samling	stav, fjeder
finans	aktie, valuta	transaktion
transport	gadekryds, lufthavn	motorvej, flyforbindelse
internet	class C network	forbindelse
brætspil	spilposition	gyldigt træk
sociale netværk	person	venskab, følger
neurale netværk	neuron	synnapse
proteinnetværk	protein	protein-protein interaktion
molekyler	atom	kemisk binding

Hvad sker der?

Mål for ugen

- Kendskab til terminologi for grafer
- Kendskab til repræsentationer af grafer (tætte og tynde)
- Forståelse af bredde-først søgning og Dijkstra's algoritme

Plan for ugen

Onsdag Repræsentation af grafer (Noter om grafer / CLRS 20.1), bredde-først søgning (Noter om grafer / CLRS 20.2)

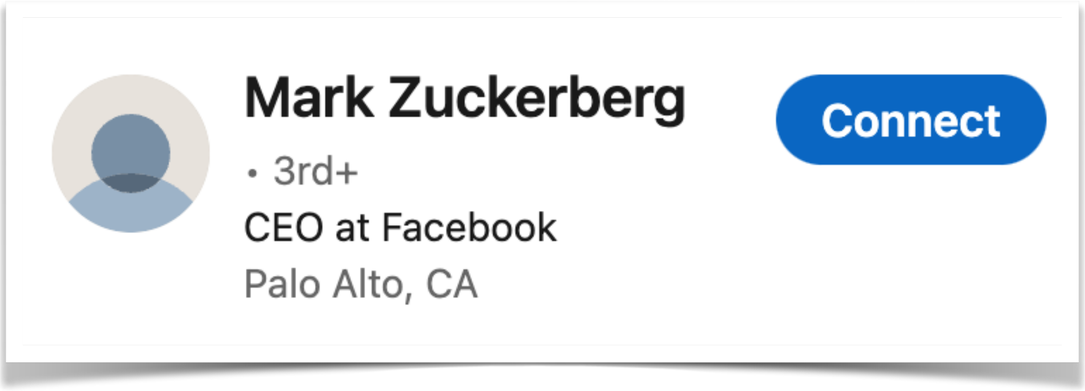
Mandag Korteste vej, Dijkstra's algoritme (Noter om grafer / CLRS 22.0, 22.3)

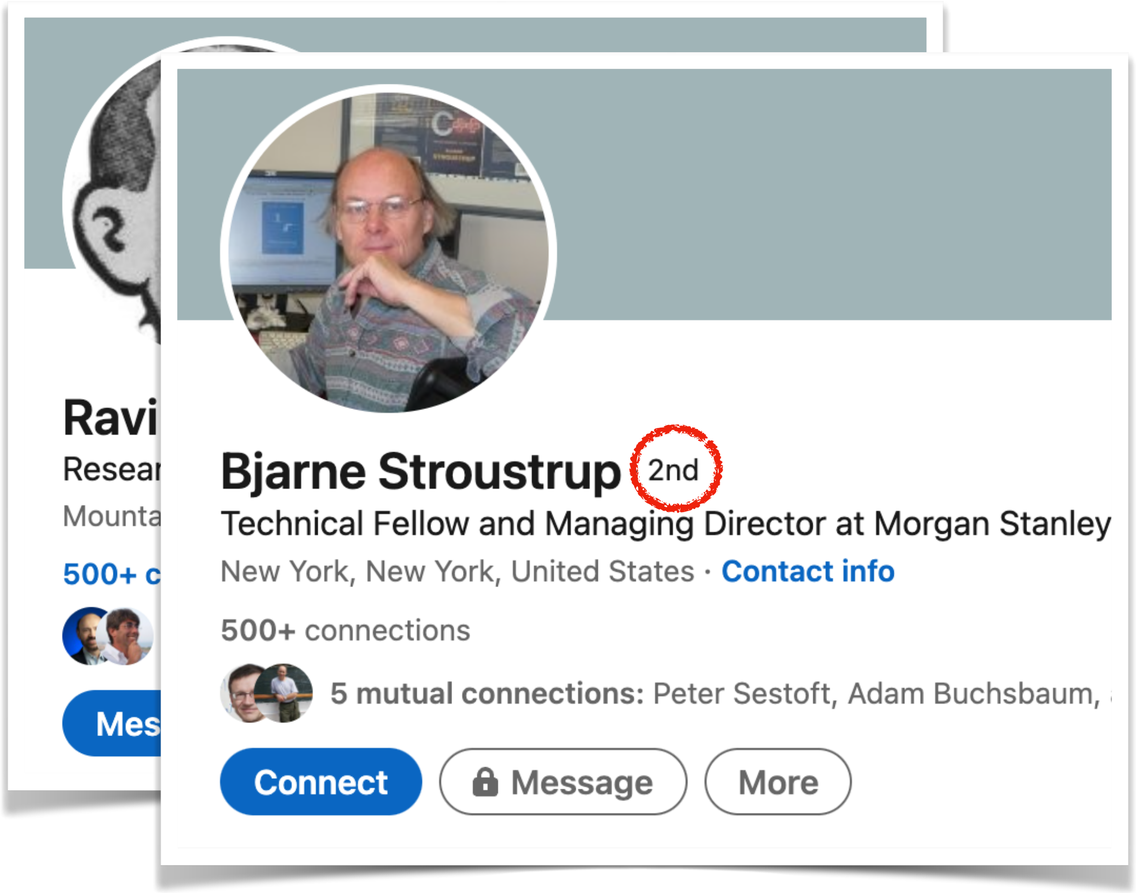
Motiverende case

- Data: En liste af n LinkedIn kontakter

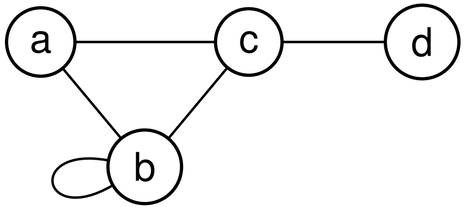
  `[(Rasmus,Ravi),(Rasmus,Peter),(Ravi,Andrei),(Andrei,Jeff),…]`

- Spørgsmål:
  Hvor lang er den korteste forbindelse i LinkedIn-grafen fra Rasmus til Mark Zuckerberg?





Repræsentation 1: Nabolister



Graph.fsx — uge 14 - grafalgoritmer 1

```
2
3   // Graph from slide 2 in lecture, nodes a, b, c, d numbered 0, 1, 2, 3
4   let edges = [ (0,1); (0,2); (1,1); (1,2); (2,3); (1,0); (2,0); (2,1); (3,2) ]
5   let n = 4 // number of nodes
6
7   // Representation using adjacency lists (DA: nabolister)
8   let neighbors_list = Array.create n []
9   for i in 0 .. n-1 do
10     neighbors_list.[i] <- []
11  for e in edges do
12     let i, j = e
13     neighbors_list.[i] <- j :: neighbors_list.[i]
14
15  printfn "Asing adjacency lists:"
16  printfn "Node 0 has neighbors %A" (neighbors_list.[0])
17  printfn "Does node 1 have 3 in its neighbor set? %b" (List.contains 3 neighbors_list.[1])
18
```

Line: 8 | F Sharp | Tab Size: 4 | n

Repræsentation 2: Nabomatricer

Graph.fsx — uge 14 - grafalgoritmer 1

```
20 // Representation using adjacency matrix (DA: nabomatrix)
21 let neighbors_matrix = Array2D.create n n 0
22 for e in edges do
23    let i, j = e
24    neighbors_matrix.[i, j] <- 1
25
26 printfn "Using adjacency matrix:"
27 printfn "Node 0 has neighbors %A" (List.filter (fun x -> neighbors_matrix.[0,x]=1)
   [ 0 .. n-1 ])
28 printfn "Does node 1 have 3 in its neighbor set? %b" (neighbors_matrix.[1,3]=1)
29
```

Line: 15:33 | F Sharp | Tab Size: 4 | i

Repræsentation 3: Hashtabeller i hashtabeller

Øvelse

- Betragt en graf med n knuder og m kanter
- For hver af de tre repræsentationer, hvad er store-O kompleksiteten af følgende:

  1. Tid for at finde alle naboer til knude i
  2. Tid for at afgøre om i og j er naboer
  3. Tid for at lave en liste med alle kanter
  4. Plads til datastrukturen

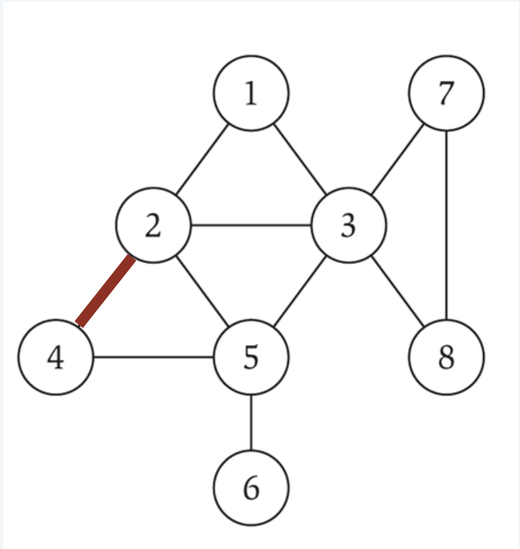
	Naboliste	Nabomatrix	Hashtabel
1			
2			
3			
4			

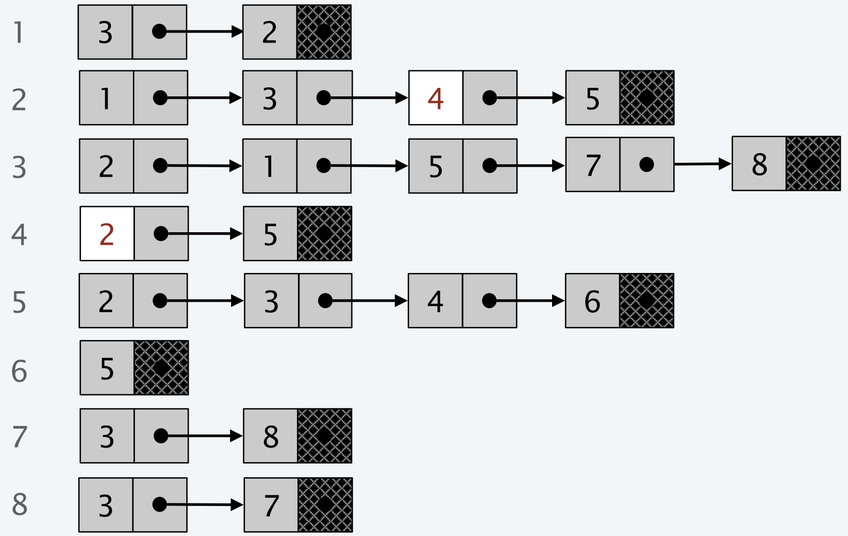
Egenskaber for nabolister

Nabolister. Knude-indekseret tabel af lister.

- To liste-elementer for hver (ikke-orienteret) kant.
- Plads $\Theta(m + n)$.
- At undersøge om (u, v) er en kant tager $O(grad(u))$ tid.
- At finde alle kanter tager $\Theta(m + n)$ tid.

grad = antal naboer til u

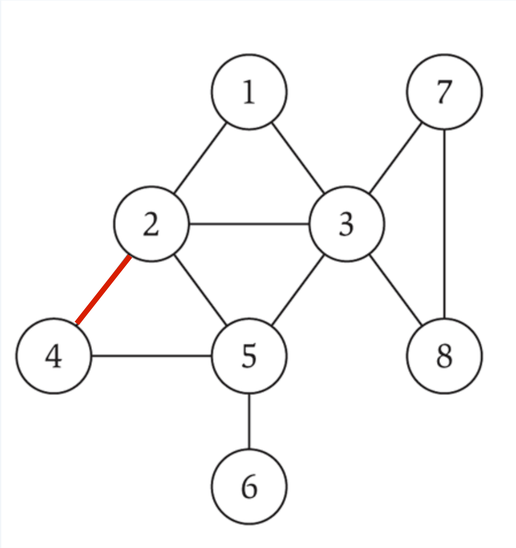




Egenskaber for nabomatrix

Nabomatrix. n-gange-n matrix hvor $A_{uv} = 1$ hvis (u, v) er en kant, $A_{uv} = 0$ ellers.

- To 1ere for hver kant (ikke orienteret).
- Plads proportional med n^2.
- At undersøge om (u, v) er en kant tager $\Theta(1)$ tid.
- At finde alle kanter tager $\Theta(n^2)$ tid.

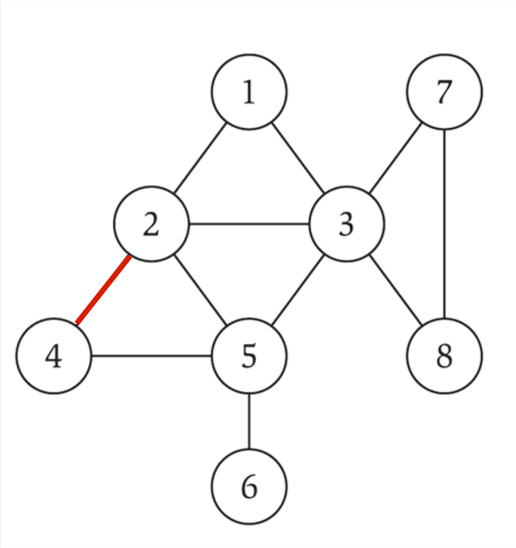


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Egenskaber for hashtabeller i hashtabeller

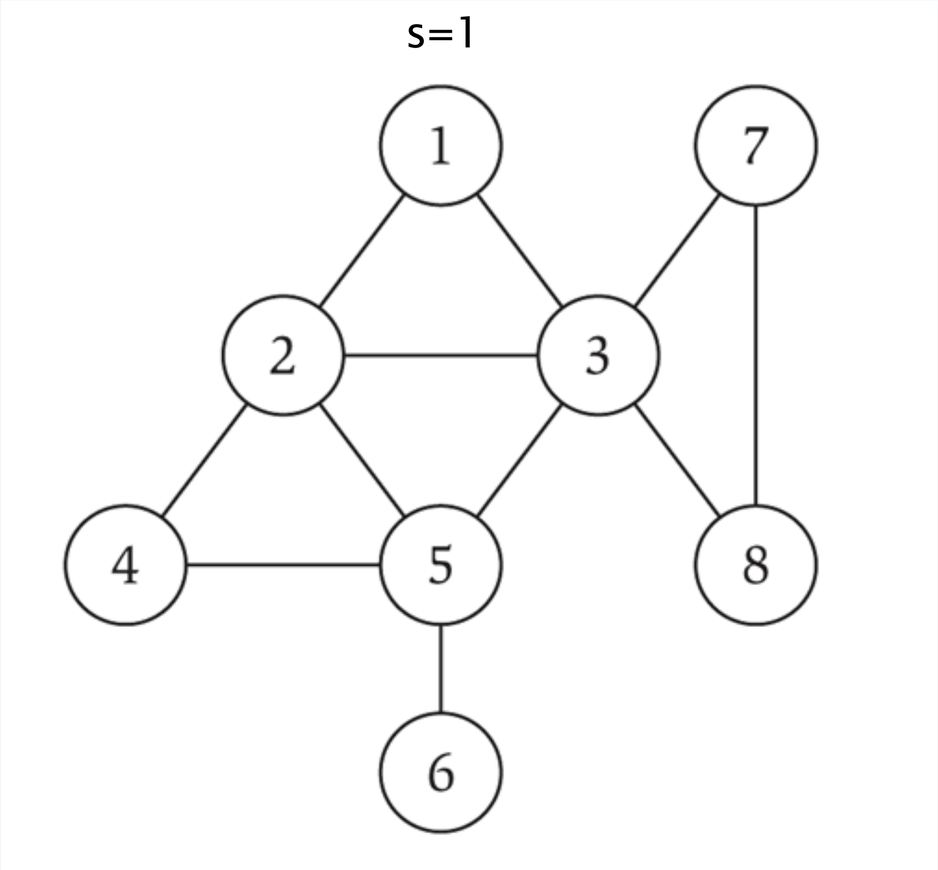
Ordbog A hvor $A[u]$ er et hashset, der indeholder v hvis og kun hvis (u, v) er en kant.

- To nøgler og to elementer i hashsets for hver kant (ikke orienteret).
- Plads $\Theta(m + n)$.
- At undersøge om (u, v) er en kant tager $\Theta(1)$ tid.
- At finde alle kanter tager $\Theta(m + n)$ tid.



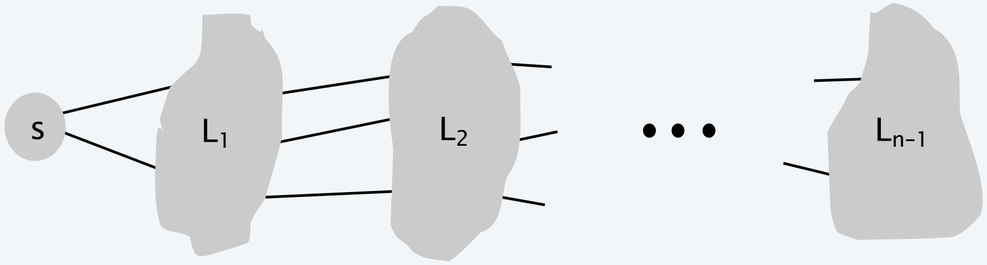
Bredde-først søgning (BFS), et eksempel

Mål. Find korteste vej fra en knude s til alle andre knuder i grafen (hvor en vej findes)



Bredde-først søgning (BFS)

Intuition. Udforsk grafen med udgangspunkt i s, læg nye knuder til i ét "lag" ad gangen.



BFS algoritme.

- $L_0 = \{ s \}$.
- L_1 = alle naboer til L_0.
- L_2 = alle knuder der ikke tilhører L_0 eller L_1, og som har en kant til en knude i L_1.
- …
- L_{i+1} = alle knuder der ikke tilhører et tidligere lag, og som har en kant til en knude i L_i.

Animation: Bredde-først søgning i en labyrint

BFS pseudokode

```
BFS(V, E, s)
  for each u ∈ V − {s}
        u.d = ∞
  s.d = 0
  Q = ∅
  ENQUEUE(Q, s)
  while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
              if v.d == ∞
                    v.d = u.d + 1
                    ENQUEUE(Q, v)
```

Afstand fra s er højst

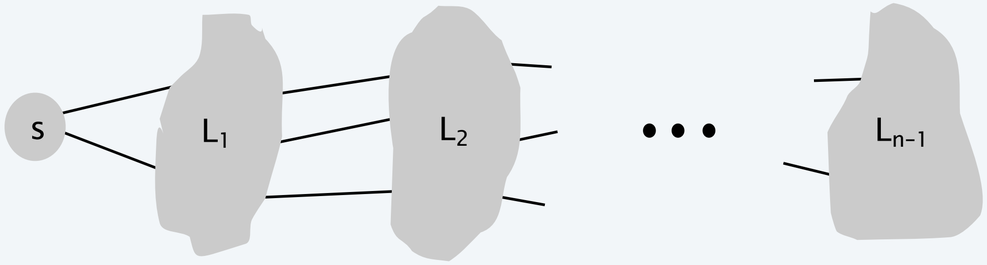
Føj til kø

Tag ud af kø

Naboliste

Bredde-først søgning (BFS)

Intuition. Udforsk grafen med udgangspunkt i s, læg nye knuder til i ét "lag" ad gangen.



BFS algoritme.

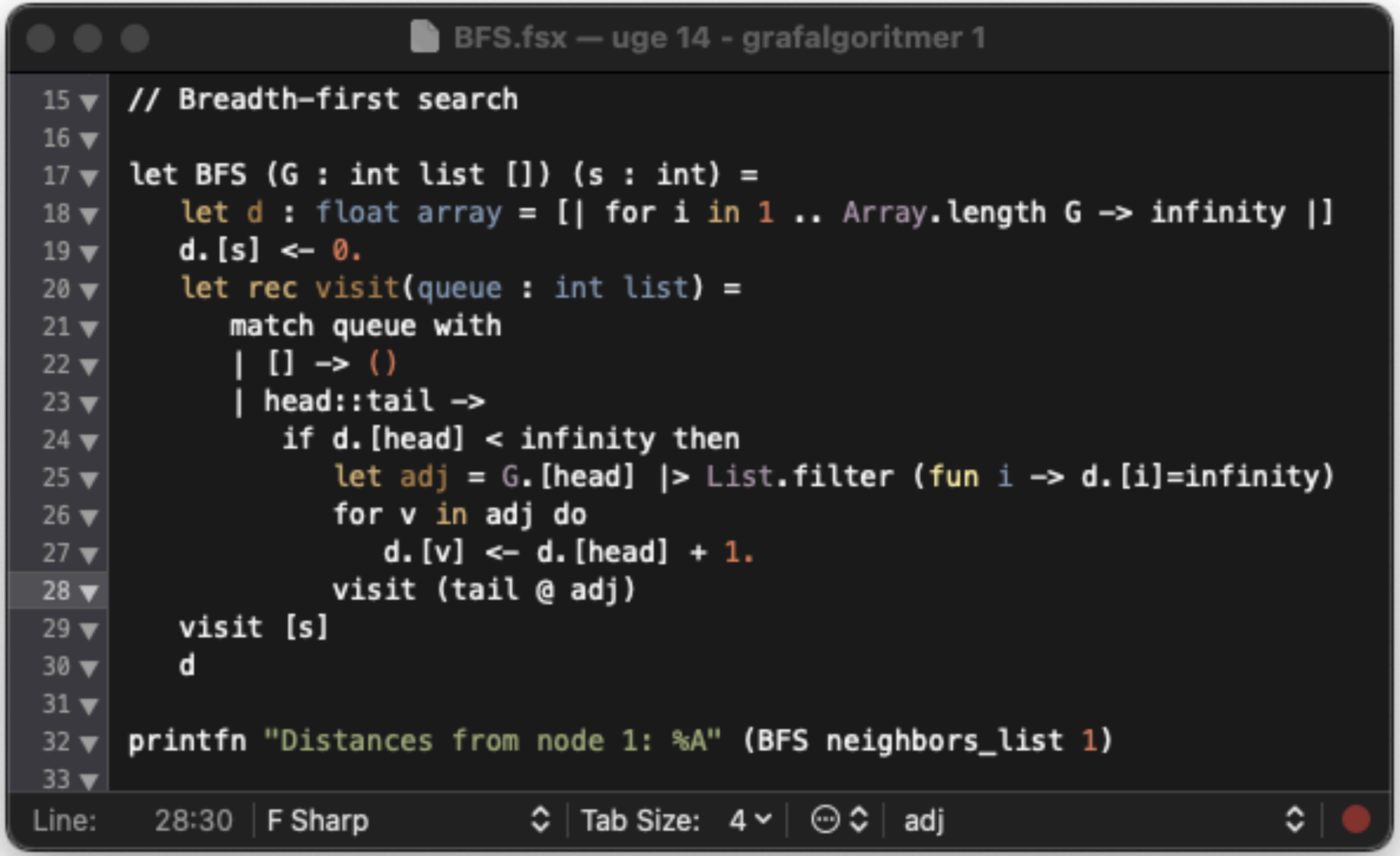
- $L_0 = \{ s \}$.
- L_1 = alle naboer til L_0.
- L_2 = alle knuder der ikke tilhører L_0 eller L_1, og som har en kant til en knude i L_1.
- …
- L_{i+1} = alle knuder der ikke tilhører et tidligere lag, og som har en kant til en knude i L_i.

Teorem. For hvert i består L_i af alle knuder der har afstand præcis i fra s.

Elementerne føjes til køen i rækkefølgen $L_0, L_1, L_2, \ldots$

Der er en sti fra s til t hvis og kun t forekommer i et af lagene.

BFS i F#


BFS.fsx — uge 14 - grafalgoritmer 1
// Breadth-first search
let BFS (G : int list []) (s : int) =
let d : float array = [| for i in 1 .. Array.length G -> infinity |]
d.[s] <- 0.
let rec visit(queue : int list) =
match queue with
| [] -> ()
| head::tail ->
if d.[head] < infinity then
let adj = G.[head] |> List.filter (fun i -> d.[i]=infinity)
for v in adj do
d.[v] <- d.[head] + 1.
visit (tail @ adj)
visit [s]
d
printfn "Distances from node 1: %A" (BFS neighbors_list 1)
Line: 28:30
F Sharp
Tab Size: 4
adj

BFS analyse

Teorem. BFS kører I tid $O(m+n)$ hvis grafen er i naboliste representation.

Bevis.

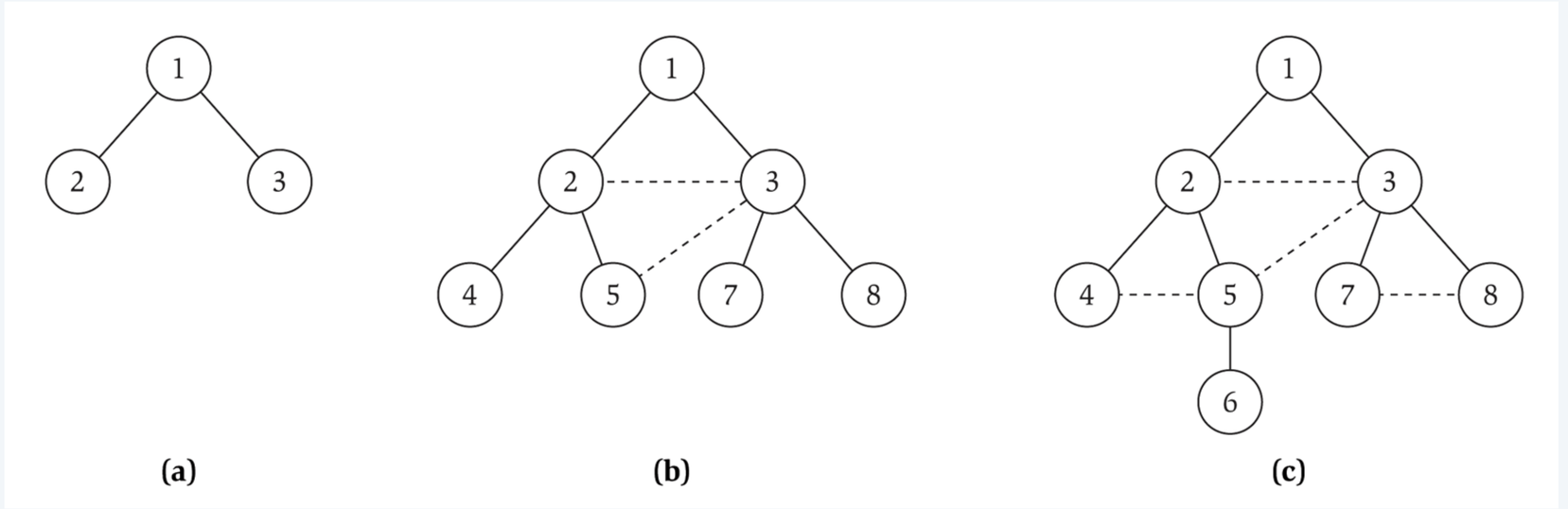
- Nemt at vise $O(n^2)$ køretid:
  - der er højst n nabolister $L[i]$
  - hver knude tilføjes køen højst én gang; while-løkken kører $\leq n$ gange
  - når vi betragter knude u, er der $\leq n$ tilstødende kanter (u, v), og vi bruger tid $O(1)$ for hver af dem

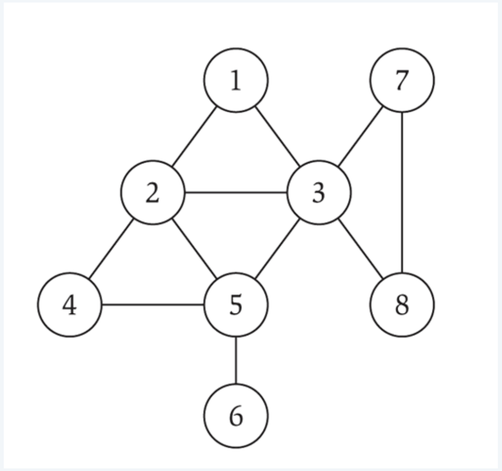
- Kører faktisk i $O(m+n)$ tid:
  - når vi betragter knude u, er der $grad(u)$ tilstødende kanter (u, v)
  - totalt bruger vi tid $\Sigma_{u \in V}\, grad(u) = 2m$ på at løbe gennem kanter. ▪

Hver kant (u, v) tælles præcis to gange i summen: én gang i grad(u) og én gang i grad(v)

```
BFS(V, E, s)
  for each u ∈ V − {s}
        u.d = ∞
  s.d = 0
  Q = ∅
  ENQUEUE(Q, s)
  while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
              if v.d == ∞
                    v.d = u.d + 1
                    ENQUEUE(Q, v)
```

BFS træet

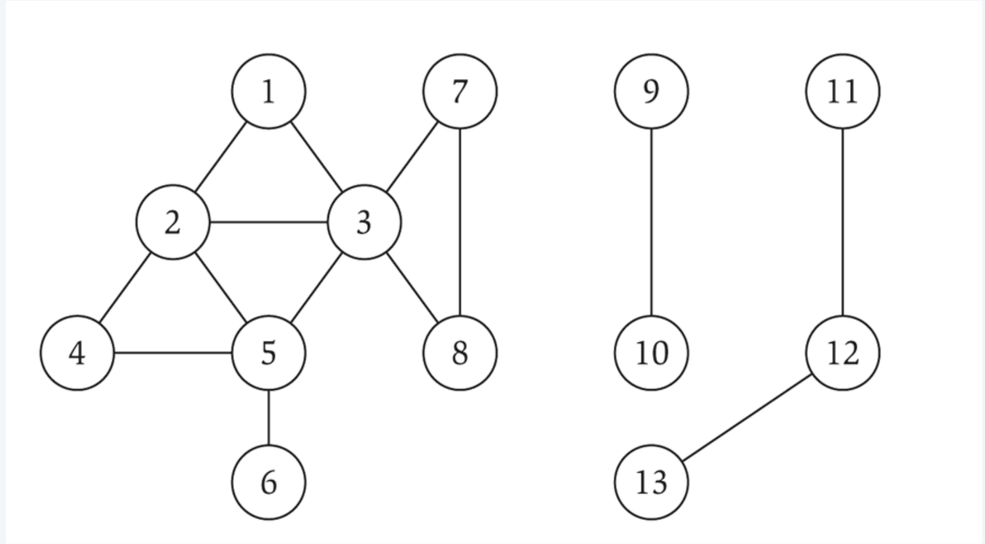




Anvendelse: Sammenhængskomponenter

Sammenhængskomponent for knude s:

Den del af grafen, der kan nås fra s.



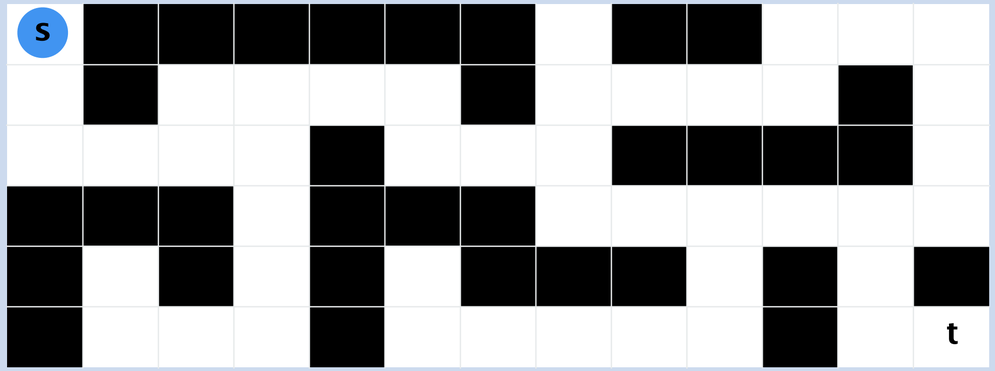
For knude $s = 1$ kan vi finde sammenhængskomponenten med knudemængden $\{1,2,3,4,5,6,7,8\}$ ved at køre BFS og sætte alle knuder, vi besøger, ind i en ordbog.

Alternativ algoritme: Brug datastruktur til disjunkte mængder, lav en union operation for hver kant, og rapportér alle knuder i samme mængde som s.

FOO.BAR CHALLENGE

You have maps of parts of the space station, each starting at a prison exit and ending at the door to an escape pod. The map is represented as a matrix of 0s and 1s, where 0s are passable space and 1s are impassable walls. The door out of the prison is at the top left $(0, 0)$ and the door into an escape pod is at the bottom right $(w-1, h-1)$.

Write a function that generates the length of a shortest path from the prison door to the escape pod, where you are allowed to remove one wall as part of your remodeling plans.



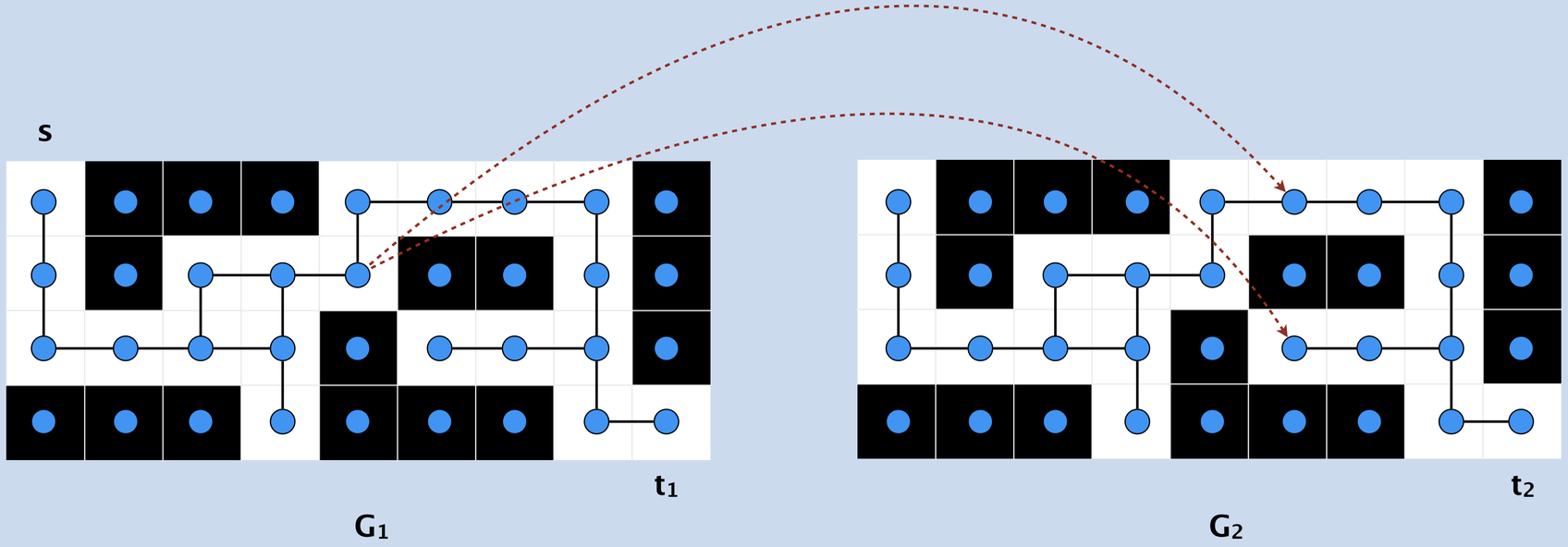
FOO.BAR CHALLENGE

Model maze as a subgraph of grid graph.

- Node for each cell.
- Bidirectional edge between two adjacent open cells (length = 1).

Idea: Use “graph doubling” trick.

- An edge from G_1 to G_2 corresponds to removing a wall (length = 2).
- No edges from G_2 to G_1.



Opsummering

- Grafer har flere effektive repræsentationer, med forskellige egenskaber
  - Nabolister er det mest almindelige, hashtabeller et fleksibelt alternativ
- Bredde-først søgning (BFS) finder korteste veje i en graf med n knuder og m kanter i tid $O(n + m)$

Næste uge: Korteste veje når kanter har forskellige længder ("vægte")