Rettede og Vektede Grafer

I. GRAFTERMINOLOGI

II. GRAF ADT OG IMPLEMENTASJON

III. GRAF TRAVERSERING: DFS OG BFS

IV. RETTEDE GRAFER (DIGRAPHS)

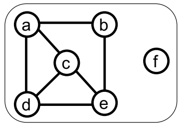
- **terminologi**
- **ADT og implementasjoner**
- **DFS/BFS av DIGRAPH**
- **transitiv tillukning**
- **DAG og topologisk sortering**

V. KORTESTE STI (SSSP) I VEKTEDE GRAFER

Ford-Bellman algoritme
Dijkstra's SSSP (Single Source Shortest Path) algoritme

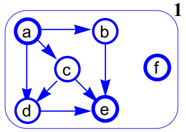
Rettet Graf (DiGraph)

URETTET: en ikke-rettet kant (u,v) = to rettede kanter $u \rightarrow v$ og $v \rightarrow u$

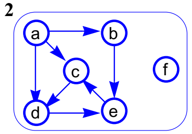


RETTET: hver kant (u, v) er et ordnet (rettet) par $u \rightarrow v$.

1



2



sti : en sekvens $n_1, n_2 \ldots n_k$ av noder slik at $(n_i, n_{i+1}) \in E$

syklus: enkel sti (hver node 1 gang) men $n_1 = n_k$

sammenhengende graf : det finnes en sti mellom alle par av noder

Er v oppnåelig fra u ?

Finn alle v oppnåelige fra u.

Er G sterkt sammenhengende ?

Er G asyklisk ?

sti : en sekvens av fra-til kanter ... : ade (ikke eda)

rettet syklus: rettet enkel sti ... **2**: cde

kilde/sluk : en node uten noen inngående / utgående kanter **1**: a/e

oppnåelig (eng: reachable) : en node v kan nåes fra u dersom det finnes en rettet sti fra u til v
e oppnåelig fra a, men ikke omvendt

sterkt sammenhengende graf : enhver node u er oppnåelig fra enhver annen node v
hverken **1** eller **2**

DAG : rettet, asyklisk graf – ingen (rettede) sykler
1 er DAG, men ikke **2**

transitiv tillukning

package jdsl.graph.api;

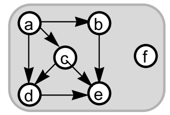
Graf ADT

*(kan ha **både** rettede og ikke-rettede kanter)*

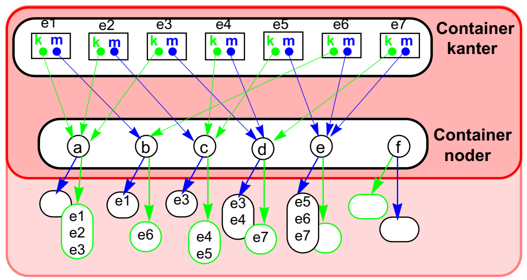
```
interface InspectableGraph extends InspectablePositionalContainer {

  int numVertices(); int numEdges();
  Enumeration vertices();
  Enumeration edges();
/** # rettede og ikke-rettede nabokanter */
  int degree(Vertex v);
  Vertex[] endVertices(Edge e)
  Vertex opposite(Vertex v, Edge e);
/** nabonoder langs alle kanter
    inn–, utgående samt ikke-rettede */
  Enumeration adjacentVertices(Vertex v)
/** rettede og ikke-rettede nabokanter */
  Enumeration incidentEdges(Vertex v)

  Enumeration unDirectedEdges();
  Enumeration directedEdges();
  int outDegree(Vertex v);
  int inDegree(Vertex v);
  Vertex origin(Edge e)
    Vertex destination(Edge e)
  boolean isDirected(Edge e)
  Enumeration outAdjacentVertices(Vertex v)
  Enumeration inAdjacentVertices(Vertex v)
  Enumeration outIncidentEdges(Vertex v)
  Enumeration inIncidentEdges(Vertex v)
}
```

```
interface Graph extends ModifiableGraph {
  Vertex insertVertex(Object o);
  Edge insertEdge(Vertex u, Vertex v, Object o);

  Object removeEdge(Edge e)
  Object removeVertex(Vertex v)
  Edge insertDirectedEdge(Ver v, Ver u, Obj O)
}

interface ModifiableGraph extends InspectableGraph
  void makeUndirected(Edge e)

  void reverseDirection(Edge e)
  /** gir retning til en ikke-rettet kant */
  void setDirectionTo/From(Edge e, Vertex v)
```


Implementasjon av Graph



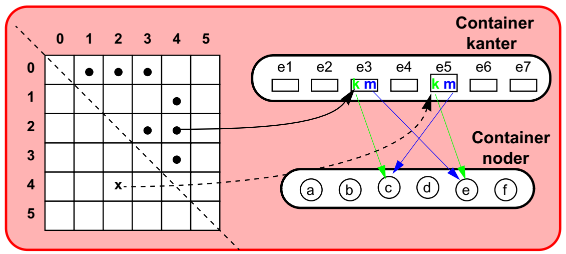
Kant-Liste



Nabo-Liste

utvid ikke-rettede implementasjoner slik at kant-klassen
- *skiller mellom* **Vertex origin** *og* **Vertex destination**, *og*
- *har et attributt* **boolean isDirected**

Nabo-Matrise

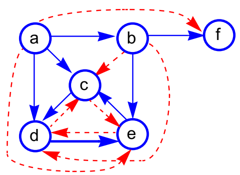


Implementasjoner av Graph

n : antall noder **m** : antall kanter

kompleksitet operasjon	Kant-Liste	Nabo-Liste	Nabo-Matrise
numVertices(), numEdges()	1	1	1
vertices() / edges() un/directedEdges()	n / m	n / m	n / m
degree(v) in/outDegree(v)	1	1	1
endVertices(e), opposite(v,e) origin(e), destination	1	1	1
adjacentVertices(v)in/outAdjacentVertices(v)	m	deg v	n
incidentEdges(v) in/outIncidentEdges(v)	m	deg v	n
insertVertex(o)	1	1	n^2
removeVertex(v)	m	deg v	n^2
insertEdge(v,u,o)inserDirectedEdge(v,u,o)	1	1	1
removeEdge(e)	1	1	1
reverseDirection(e), makeUndirected(e), ...	1	1	1
areAdjacent(v,u)	m	min(deg u,v)	1

Transitiv tillukning



IterertDFS(Graph G) O(**n * DFS**)
for hver node v ε V
DFS(v) ogutvid G med kant (v,u)
for hver u i DFS-tre til v

node	kant til	lagt til
a	b c d	e f
b	e f	c d
c	d	e
d	e	c
e	c	d
f		

Kant-Liste $O(n^2 * m)$
Nabo-Liste $O(n^2 + nm)$
Nabo-Matrise $O(n^3)$

FloydWarshall(Graph G) O(n^3) **med nabomatrise**
enummerer V : $v_1, v_2, \ldots, v_n$ (vilkårlig)
$G_0 = G$
for k = 1,2,...,n
$G_k = G_{k-1}$
for hvert tallpar a ≠ b, a,b ≠ k
if G_{k-1}.**areAdjacent**(v_a, v_k) og G_{k-1}.**areAdjacent**(v_k, v_b)
legg kant (v_a, v_b) til G_k

i en rettet graf:
G_{k-1} har kant (v_a, v_i) og (v_i, v_b)

1 2 3 4 5 6
a b c d e f

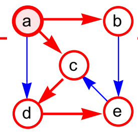
$G_a = G_0$ = { ab, ac, ad, cd, de, ec, bf }
$G_b = G_a$ ∪ { ae, af }
$G_c = G_b$ ∪ { ed } (ad)
$G_d = G_c$ ∪ { ce }
$G_e = G_d$ ∪ { bc, bd, dc } (ac)
$G_f = G_e$

Kant-Liste $O(n^3 * m)$
Nabo-Liste $O(n^3 * deg)$
Nabo-Matrise $O(n^3)$

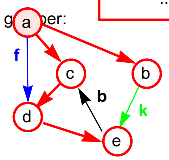
DFS på en rettet graf

9.16 (9.12) DFS traversering av en rettet graf G fra en node a:
a) besøker alle noder *oppnåelige* fra a
b) gir et utspennende tre, DFS-treet, for delgrafen oppnåelig fra a

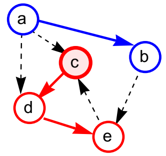
```
DFS(u)              // opptil n rekursive kall
  merk–u
  for hver kant e ε outIncidentEdges(u)
     v = opposite(u,e)
     if ( ! merket(v) )
        // merk e rød
        ..... DFS(v)
```



– Traverserte kanter som ikke er med i DFS-treet kan deles i tre grupper:
- **fram**-kanter fra v til en etterfølger node i DFS-treet
- **tilbake**-kanterfra v til en forgjenger node i DFS-treet
- **kryss**-kanter fra v til en urelatert node i DFS-treet



– Iterert DFS: 'for hver node v utfør DFS(v)' kan gi en skog:



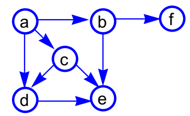
– BFS for rettede grafer har tilsvarende egenskaper til BFS for ikke-rettede grafer (etterlater kun tilbake- og kryss-kanter)

kompleksitet	kant-liste	nabo-liste	nabo-matrise
outIncidentEdges(v)	O(m)	O(deg)	O(n)
DFS	**O(n * m)**	**O(n + m)**	**O(n * n)**
siden m=O(n*n) får vi	$O(n^3)$	$O(n^2)$	$O(n^2)$

DFS på rettet graf gir opphav til O(**n+m)** algoritme for å :
- finne alle noder oppnåelig fra en gitt node

Iterert DFS gir O(**n(n+m))** algoritmer for å :
- avgjøre om G er sterkt sammenhengende; (mulig også i O(n+m))
- lage transitiv tilluking G* av G

DAG-Directed Acyclic Graph



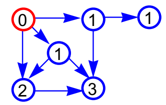
rettet asyklisk graf
- *arv i et OO-språk*
- *forkrav til kurs*
- *planlegging av avhengige aktiviteter*

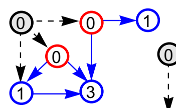
1	2	3	4	5	6
a	**b**	**f**	**c**	**d**	**e**
a	**c**	**d**	**b**	**e**	**f**
....					
a	d	c	e	b	f

Topologisk ordning av en graf G er en enummerering av noder $v_1, v_2 \ldots v_n$ slik at hvis (v_i, v_j) ε E så er i < j. (dermed, hvis det finnes en sti $v_i \ldots v_j$ så er i < j)

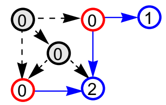
9.21. En rettet graf kan sorteres topologisk hvis og bare hvis den er asyklisk.

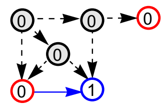
```
Queue TS(Graph G)                          O(nk)
  Q, R = empty Queue                       O(n+k)
  for hver node v ε V                      O(n2)
  { in(v) = G.inDegree(v)
    if (in(v) == 0) Q.enqueue(v) }
  while (! Q.isEmpty() )
  { h = Q.dequeue()
    for hver v ε G.outAdjacentVertices(h)
    { in(v) = in(v) – 1
      if (in(v)==0) Q.enqueue(v) }
    R.enqueue(h) }
```

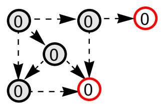




R	a	b	c	d	e	f	Q
	0	1	1	2	3	1	a
a	x	0	0	1	3	1	cb
a c	x	0	x	0	2	1	bd
a c b	x	x	x	0	1	0	d f
a c b d	x	x	x	x	0	0	f e
a c b d f e							







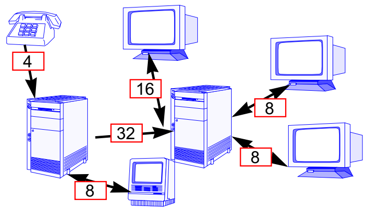
9.22 Har grafen en rettet sykel vil TS oppdage dette. Hvordan?.
-> TS gir en O(n+m) algoritme som sjekker om en rettet graf er

Vektede Grafer

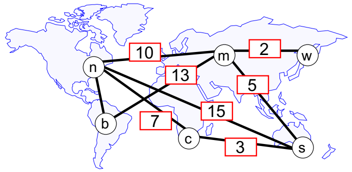
en graf der hver kant har et vekt-attributt

- vektene er vanligvis Totalt Ordnet (typisk heltall)
  - man designer en Comparator for sammenlikning av kanter mht. vekt
  - tilleggs antakelser om vekter (f.eks. > 0, 0, etc.)

NETTVERK KAPASITET



AVSTAND



- Implementasjon bruker det faktum at 'Edge implements Position' – kanter lagrer objekter med vekt

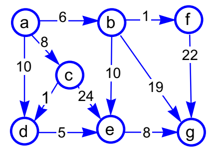
- I tillegg til vanlige graf-problemer, spør man i forbindelse med vektede grafer for eksempel om
  - *hva er **korteste** sti fra u til v, dvs sti fra u til v med minste sum av vekter?*
  - *hva er **minste** utspennende tre, dvs hvor sum av vekter på kanter i treet er minst?*
  - *..... minste / korteste / billigste ..... ?*

Korteste sti (single-source shortest-paths) :

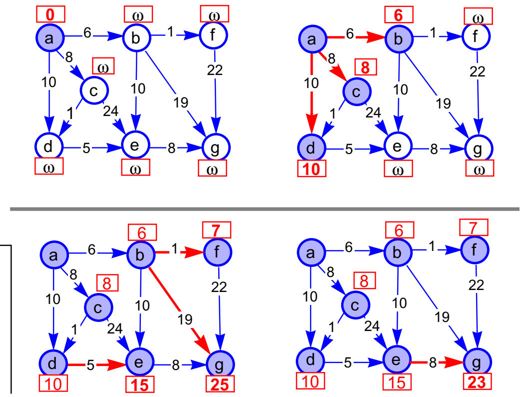
Finn (lengden av) korteste sti fra a til en bestemt/alle andre node(r)

Ford-Bellman SS-SP : for each node v : D(v)= ω // ω er et maksimalt tall (her ω > n)
D(s) = 0; // s er startnoden
for (i=1; i<n; i++) // n antall noder, m antall kanter
for each kant (u,v)
if (D(u) + vekt(u,v) < D(v)) D(v) = D(u) + vekt(u,v)

O(n*m)



graf	vekter
ikke-rettet	1
rettet	vilkårlige

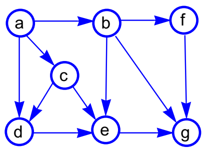


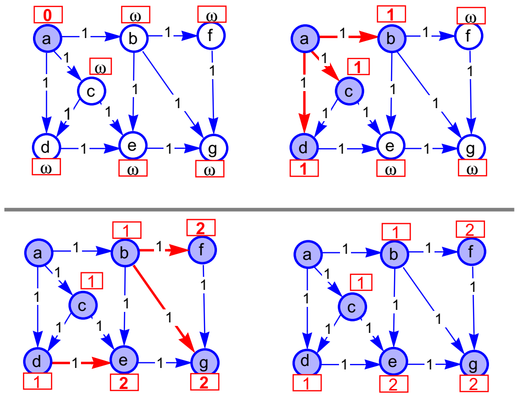
10.3 Etter Ford-Bellman (G,a):
- hvis det finnes en kant (u,v) med D(u) + vekt(u,v) < D(v), så har G en negativ sykel
- ellers D(v) korrekt for alle noder v

Korteste sti i ikke-vektet graf ...BFS

Finn sti fra a til enhver annen node, som har minst antall kanter (alternativt: kantvekter=1)

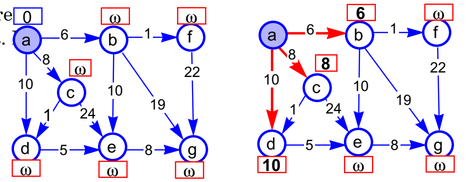
Ford-Bellman **BFS** : for each node v : D(v)= ω // ω er et maksimalt tall (her ω > n)
D(s) = 0; // s er startnoden
for (i=1; i<n; i++) // n er antall noder i grafen G **O(n*m)**
for each kant (u,v)
if (D(u) + 1 < D(v)) D(v) = D(u) + 1

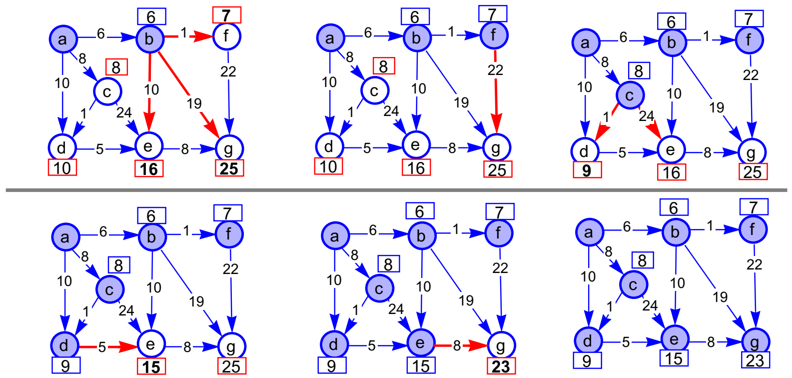




Korteste sti (SS-SP) : Dijkstra algoritme (vekt(e) ≥ 0)

initialiser D(a)=0 og D(v)=ω for alle andre
sett alle noder i en PriorityQueue Q mht. D
while (! Q.isEmpty())
v = Q.removeMinElement()// Greedy
for hver z ε G.outAdjacentVertices(v)
if (D(v)+vekt(v,z) < D(z))
D(z)= D(v) + vekt(v,z)
oppdater Q // z kan få ny nøkkel

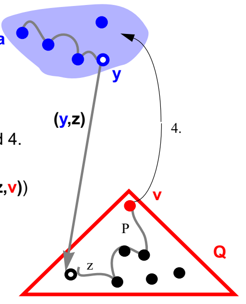




Dijkstra's SS-SP

```
1. initialiser D(a)=0 og D(v)=ω for alle andre v
2. sett alle noder i en PriorityQueue Q mht. D
3. while ( ! Q.isEmpty() )                    // n
4.    v = Q.removeMinElement()
5.    for hver z ε G.outAdjacentVertices(v)// k: Nabo-Liste
6.       if ( D(v)+vekt(v,z) < D(z) )
7.       Q.replaceKey(z, D(v) + vekt(v,z) )
```

PriorityQueue	skal bruke Locator !!!	
heap	**O**((n+m) log n)	**O**(n^2 log n)
usortert sekvens	**O**(n*n+m)	**O**(n^2)
sortert sekvens	**O**(n +m*n)	**O**(n^3)



Løkke Invariant : **alle noder x som har blitt fjernet fra Q har korrekt D(x)**
– holder før inngangen siden ingen node ble fjernet.
– for å vise at den opprettholdes i løkken, må vise at den holder etter 4.

10.1 Ved 4. er D(v) = d(a,v) – lengden av korteste sti fra a til v.

a) Anta ikke og la **v** være **den første** node for hvilken **D(v) > d(a,v)** ved 4.
b) Dvs. **korteste sti P** a–v er kortere enn **D(v)**
c) La **z** være første noden på P som fortsatt er i Q (**d(a,v) = d(a,z)+d(z,v)**)
d) og la **y** være **z**'s umiddelbar forgjenger på P med en P-kant = **(y,z)**

a) → **e)** **D(y) = d(a,y)**
4. → **f)** **D(v) ≤ D(z)**
d) → **g)** **D(z) ≤ D(y) + vekt(y,z) = d(a,y) + vekt(y,z)**
siden (y,z) er med i korteste sti P a—v, finns det ikke en kortere sti a—z enn
h) **d(a,z) = D(y) + vekt(y,z) = D(z)**

Men da:

D(v) ≤^{f} D(z) =^{h} d(a,z) ≤ d(a,z) + d(z,v) =^{c} d(a,v) – motsier a) D(v) > d(a,v)

Oppsummering

- *Rettede Grafer*
  - *– Graf ADT*
  - *– implementasjoner*
- ***Algoritmer***
  - ***DFS*** *– forskjeller fra ikke-rettet tilfelle*
  - ***transitiv tillukking***
    - *– n * DFS*
    - *– Floyd-Warshall : nabo-matrise!*
  - ***topologisk sortering***
    - *DAG*
- *Vektede Grafer*
  - ***korteste sti***
    - *– Ford-Bellman algoritme : O(n*m)*
    - *– Dijkstra algoritme : O((n+m) log n)*