Førsteårsprojekt F2008

Nogle algoritmer på grafer

Peter Sestoft

2008-02-19

Oversigt for i dag

- Definition: graf og orienteret graf
- Repræsentation ved kantlister
- Bredde-først gennemløb
- Dybde-først gennemløb
- Topologisk sortering
- De mysteriøse generiske typer:

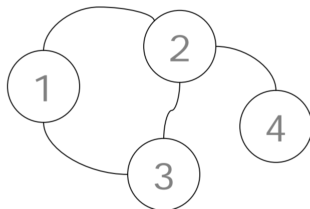
```
class Graph<E extends Edge<N>,
            N extends Node> { ... }
```

- Opgaver

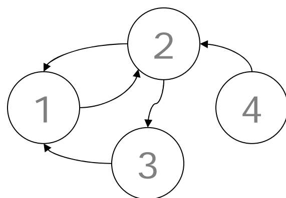
Definition af ikke-orienteret graf

- En graf G = (V, E) består af
  - en mængde V af knuder
  - en mængde $E \subseteq V \times V$ af kanter
- Der er en kant $e = v_1 \rightarrow v_2$ hvis $(v_1, v_2) \in E$
- I en ikke-orienteret graf gælder: $(v_1, v_2) \in E \Leftrightarrow (v_2, v_1) \in E$
- Med andre ord $v_1 \rightarrow v_2 \Leftrightarrow v_2 \rightarrow v_1$

Eksempler



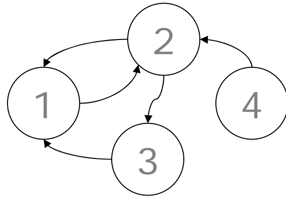
V = { 1, 2, 3, 4 }
E = { (1,2), (2,1), (1,3), (3,1), (2,3), (2,3), (2,4), (4,2) }



V = { 1, 2, 3, 4 }
E = { (1,2), (2,1), (3,1), (2,3), (4,2) }

Repræsentation af graf med kantlister

- For hver knude er der en liste af kanter fra den knude



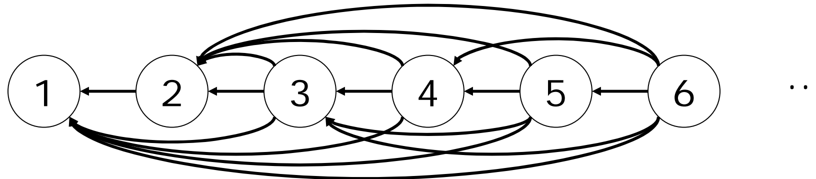
1: [(1,2)]
2: [(2,1), (2,3)]
3: [(3,1)]
4: [(4,2)]

(Kant)relationer i orienterede grafer

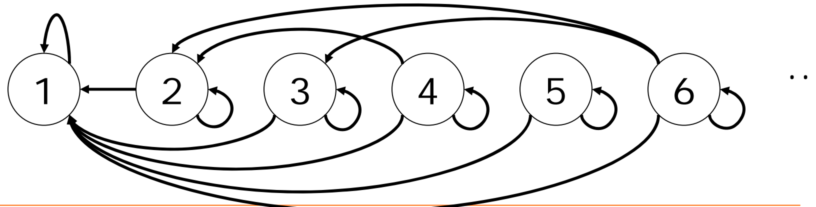
- Kantrelationen er *refleksiv* hvis
  $v_1 \rightarrow v_1$
  for alle knude v_1
- Kantrelationen er *transitiv* hvis
  $v_1 \rightarrow v_2$ og $v_2 \rightarrow v_3 \Rightarrow v_1 \rightarrow v_3$
  for alle knuder v_1, v_2, v_3
- Kantrelationen er *symmetrisk* hvis
  $v_1 \rightarrow v_2 \Rightarrow v_2 \rightarrow v_1$
  for alle knuder v_1, v_2

Eksempler på sære grafer

- Ordning: a → b siger *a større end b*



- Delelighed: a → b siger *a delelig med b*

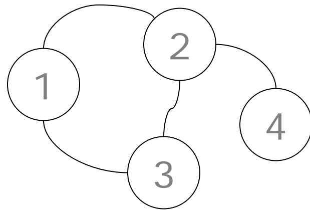


Klasse Graph i vejdataprogrammet bruger kantlisterepræsentation

- Skelner
  - Ikke-orienterede kanter
  - Orienterede kanter (= ensrettet vejstump)
- En Node har et knudenummer, index
- En Edge har to Nodes v1 og v2
- En Graph har for hver knude
  - en ArrayList<Edge> af ikke-orienterede kanter, både til og fra knuden, og af de orientere kanter der udgår fra knuden
  - en ArrayList<Edge> af de orienterede kanter der indgår til knuden

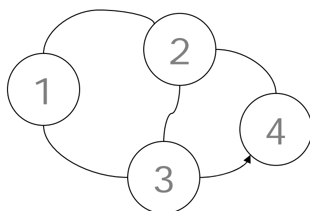
IT University of Copenhagen www.itu.dk

Ikke-orienterede kanter i vejdataprogrammet




```
1:  [(1,2),(1,3)]
    []
2:  [(2,1),(2,3),(2,4)]
    []
3:  [(3,1),(3,2)]
    []
4:  [(4,2)]
    []
```


Blandet ikke-orienterede og orienterede kanter




```
1:  [(1,2),(1,3)]
    []
2:  [(2,1),(2,3),(2,4)]
    []
3:  [(3,1),(3,2),(3,4)]
    []
4:  [(4,2)]
    [(3,4)]
```

Indgående, orienteret kant

Data i klasse Graph<E,N>

- Alle knuder, nummereret fra 1:

```
ArrayList<N> nodes;
```

- NB: **nodes.get(0)** er null
- NB: **nodes.size()** = antal knuder + 1

- Ikke-orienterede samt udgående orienterede kanter:

```
ArrayList<ArrayList<E>> edges;
```

- NB: **edges.get(0)** er den tomme arrayliste
- NB: **edges.size()** = antal knuder + 1

- Orienterede indgående kanter:

```
ArrayList<ArrayList<E>> reverse_edges;
```

Når alle kanter er ikke-orienterede (edge.direction == BOTH)

- Alle kanter til og fra knude **node**:

```
edges.get(node.index)
```

- Opregn alle nabo-knuder:

```
for (Edge<Node> edge : edges.get(node.index))
{
  Node neighbour = edge.getOtherEnd(node);
  ...
}
```

Bredde-først gennemløb (BFS), ikke-orienteret graf

- Lav en *kø* af knuder der skal besøges
- Læg startknuden i køen
- Så længe der er noget i køen
  - Tag knude v fra køen
  - Hvis v ikke er besøgt allerede
    - Marker v besøgt
    - Læg alle v's naboer i køen

Eksempel på bredde-først gennemløb, uorienteret graf

- Goodrich & Tamassia page 314

Bredde-først gennemløb, uorienteret graf

```
HashSet<Node> exploredNodes = new HashSet<Node>();
Queue<Node> toVisit = new LinkedList<Node>();

toVisit.offer(start);
while (!toVisit.isEmpty()) {
  Node v = toVisit.poll();
  if (!exploredNodes.contains(v)) {
    System.out.print(v.index + " ");
    exploredNodes.add(v);
    for (Edge<Node> edge : graph.edges.get(v.index)) {
      Node w = edge.getOtherEnd(v);
      toVisit.offer(w);
    }
  }
}
```

dequeue

enqueue

Dybde-først gennemløb (DFS)

- Lav en *stak* af knuder der skal besøges
- Læg startknuden på stakken
- Så længe der er noget på stakken
  - Tag knude v fra stakken
  - Hvis v ikke er besøgt allerede
    - Marker v besøgt
    - Læg alle v's naboer på stakken

Eksempel på dybde-først gennemløb, uorienteret graf

- Goodrich & Tamassia side 304

Når alle kanter er orienterede (edge.direction != BOTH)

- Alle kanter ud fra knude **node**:

```
edges.get(node.index)
```

- Antal kanter ud fra **node**, dens *outdegree*:

```
edges.get(node.index).size()
```

- Alle kanter ind til knude **node**:

```
reverse_edges.get(node.index)
```

- Antal kanter ind til **node**, dens *indegree*:

```
reverse_edges.get(node.index).size()
```

Topologisk sortering af en *dag*

- Forudsætning: Ingen kredse
- Anvendelser:
  - Rækkefølge af aktiviteter i et byggeprojekt
  - Genberegningsrækkefølge i regneark; jvfr Excels Værktøjer > Revision
- Faktum: I en endelig acyklisk graf er der en knude uden indgående kanter
- Se Goodrich & Tamassia §6.4.4

Algoritme til topologisk sortering

- Gentag indtil ikke flere knuder:
  - Vælg en knude u uden indgående kanter
  - Sæt u i den topologisk ordnede liste
  - Fjern u fra grafen, og fjern alle kanter der udgår fra u
- Hvis grafen indeholder en kreds så standser algoritmen inden alle knuder er behandlet (kredsen mangler nemlig)

Effektiv topologisk sortering

- Lav en HashMap<Node,Integer> med aktuel indgrad for hver knude
- Vedligehold en stak med de knuder der har indgrad = 0
- Gentag indtil stakken er tom
  - Tag en knude u fra stakken
  - Sæt u i den topologiske orden
  - For hver kant fra u til en knude t, tæl t's indgrad ned med 1
  - Hvis t's indgrad bliver 0, læg t på stakken
- Køretid O(|N| + |E|)

Eksempel

- Goodrich & Tamassia side 328

Generiske typer i Edge<N>

- Node er klassen af knuder
- Edge<N> er klassen af kanter som forbinder to knuder af type N
  - Det giver kun mening hvis N er Node eller en subklasse af Node
  - Derfor

```
class Edge<N extends Node> { ... }
```

Generiske typer i Graph<E,N>

- Graph<E,N> er klassen af grafer med kanter af type E og knuder af type N
- Det giver kun mening
  - hvis N er Node eller en subtype af Node, og
  - hvis E er en kant mellem to N'er
- Derfor

```
class Graph<E extends Edge<N>,
            N extends Node> { ... }
```

- Dvs. graf med kanter af type E og knuder af type N, hvor N er en knudetype, og E er en kanttype der forbinder N-knuder

Vejdatasættet som subklasser af Node, Edge, Graph

- Subklasse KrakNode af Node
  - Tilføjer geografiske koordinater
- Subklasse KrakEdge af Edge
  - Tilføjer vejstykkets navn, længde, husnumre, postnumre, køretid, ...
- Der gælder
  - KrakNode extends Node
  - KragEdge extends Edge<KrakNode>
- Og vejdatagrafen har type
  - Graph<KrakEdge, KrakNode>

Opgaver, se seddel 4

- Eksperimenter med bredde-først gennemløb
- Tilføj niveauer til bredde-først
- Implementer dybde-først gennemløb
- Detekter kredse
- Implementer topologisk sortering