Föreläsning
Datastrukturer (DAT037)

Nils Anders Danielsson

2015-11-20

Idag

Grafer:

- Terminologi.
- Datastrukturer.
- Topologisk sortering.
- Kortaste vägen.
  - Bredden först-sökning.
  - Dijkstras algoritm.

(Vi får se vad vi hinner.)

Hashtabeller, uppdatering

Förra föreläsningens analyser av insättning i hashtabeller gäller om:

1. `rehash` implementeras på ett optimerat sätt (vi behöver inte testa om elementen redan finns i hinkarna):
   - Tidskomplexitet: $\Theta(kapacitet + n)$.

Hashtabeller, uppdatering

Förra föreläsningens analyser av insättning i hashtabeller gäller om:

1. `rehash` implementeras på ett optimerat sätt (vi behöver inte testa om elementen redan finns i hinkarna):
   - Tidskomplexitet: $\Theta(kapacitet + n)$.
2. Lastfaktorn är $\geq c$ (en konstant):
   - Lastfaktorn $= n/kapacitet \geq c$.
   - $kapacitet = O(n)$.
   - $\Theta(kapacitet + n) = \Theta(n)$.

Grafer

Grafer

Grafer kan representera:

- Nätverk.
- Beroenden.
- ...

Grafer

Givet en graf kan man ställa olika frågor:

- Nätverk.
  - Hur tar man sig från A till B? Snabbast? Billigast?
  - Vilken rutt har störst bandbredd?
- Beroenden.
  - Vad måste göras först?
- ...

Terminologi

Terminologi

Varning

Kan variera från författare till författare.

Terminologi

- Graf: $G = (V, E)$.
- V: Ändlig mängd av noder.
- E: Kanter/bågar.
- Riktad graf: $E \subseteq V \times V$.
- Oriktad graf: $E \subseteq \{\, U \subseteq V \mid 1 \leq |U| \leq 2 \,\}$.
- Viktad graf: $E \subseteq V \times V \times W$ eller $E \subseteq \{\, U \subseteq V \mid 1 \leq |U| \leq 2 \,\} \times W$ (där W kan vara $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, …).
- I en *multigraf* kan det finnas flera kanter från u till v.

Terminologi

- Direkta efterföljare till u: $\{ v \mid (u, v) \in E \}$.
- Direkta föregångare till v: $\{ u \mid (u, v) \in E \}$.
- Ingrad: Antalet direkta föregångare.
- Utgrad: Antalet direkta efterföljare.

Begreppen definieras på motsvarande sätt för oriktade grafer/multigrafer/viktade grafer.

Terminologi

- Väg: $v_1 \to v_2 \to ... \to v_n$.
- Längd: $n - 1$.
- Vägar kan ha längd 0.
- Enkel väg: Alla noder distinkta (utom möjligtvis v_1 och v_n).
- Loop: Kant från nod till sig själv.

Terminologi

För riktade grafer:

- Cykel: Väg av längd ≥ 1 från v till v.
- Enkel cykel: Cykel som är enkel väg.
- (Riktad) acyklisk graf/DAG: Graf utan cykler.

För oriktade grafer:

- (Enkel) cykel:
  Enkel väg av längd ≥ 3 från v till v.

Terminologi

För oriktade grafer:

- Sammanhängande:
  Finns väg från varje nod till varje annan nod.

För riktade grafer:

- Starkt sammanhängande:
  Finns väg från varje nod till varje annan nod.
- Svagt sammanhängande:
  Finns väg från varje nod till varje annan nod, om man även får följa kanter baklänges.

Terminologi

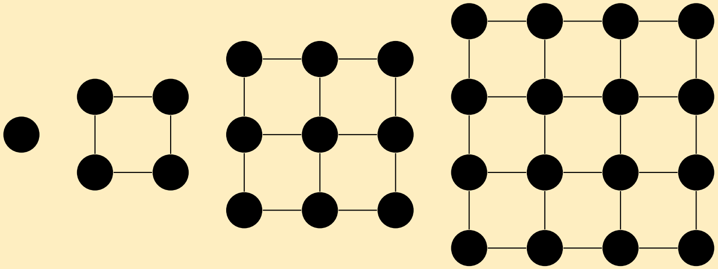
Komplett graf:

- Inga loopar.
- I övrigt så många kanter som möjligt.

Terminologi

- Tät graf: Många kanter (kanske $|E| \sim |V|^2$).
- Gles graf: Få kanter (kanske $|E| \lesssim |V|$).

Gäller $|E| = \Theta(|V|^2)$ för följande klass av grafer?



...

Data-strukturer

Grannmatriser

- Kvadratisk matris med $|V|^2$ element.
- Elementen kan t ex vara `true`/`false`.
- Tar stor plats om grafen är gles.
- Gå igenom en nods direkta efterföljare: $\Theta(|V|)$.
- Gå igenom alla noders direkta efterföljare: $\Theta(|V|^2)$.
- Avgöra om det finns en kant från u till v: $\Theta(1)$.
- Antagande ovan: Känner till noders index.
- Kan använda avbildning (map): nod $\mapsto$ index.

Grannlistor

En variant:

- Array av storlek $|V|$...
- ...innehållandes oordnade listor med direkta efterföljare.
- Ibland också listor med direkta föregångare.
- Gå igenom en nods direkta efterföljare: $O(|V|)$.
- Gå igenom alla noders direkta efterföljare: $\Theta(|V| + |E|)$.
- Avgöra om det finns en kant från u till v: $O(|V|)$.

Grannlistor

En annan variant:

- Ett objekt per nod.
- Avbildning från noder till nodobjekt.
- Objekten innehåller grannlistor med pekare till andra objekt.

Hur stor plats tar en array med grannlistor? (Anta att etiketter/vikter tar liten plats.)

- $\Theta(|V|)$.
- $\Theta(|E|)$.
- $\Theta(|V| + |E|)$.
- $\Theta(|V|^2)$.

Topologisk sortering

Topologisk sortering

Definition:

- Total ordning av V.
- Om det finns en väg från v_1 till v_2 så är $v_1 < v_2$.

Exempel:

- Förkunskapskrav $\Rightarrow$ giltig ordning av kurser.

Topologisk sortering

- Ointressant för oriktade grafer.
- Cykel $\Rightarrow$ ingen topologisk sortering.
- DAGs (riktade acykliska grafer) kan alltid sorteras topologiskt.
- Tillräckligt villkor för att vara cyklisk: Grafen innehåller minst en nod, och alla noder har ingrad > 0.

Topologisk sortering: enkel algoritm

```
r = new empty list

while V ≠ ∅ do
  if any v ∈ V with indegree(v) = 0 then
    r.add-last(v)
    remove v from G
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Topologisk sortering: enkel algoritm

```
r = new empty list

while V ≠ ∅ do
  if any v ∈ V with indegree(v) = 0 then
    r.add-last(v)
    remove v from G
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Kan vi undvika radering?

Topologisk sortering: enkel algoritm (2)

```
r = new empty list
d = map from vertices to their indegrees
    // null for nodes in r.

repeat |V| times
  if d[v] == 0 for some v then
    r.add-last(v)
    d[v] = null
    for each direct successor v' of v do
      decrease d[v'] by 1
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Grafrepresentation

Pseudokod: Behöver mer information för tidskomplexitetsanalys.

Grafrepresentation (den här gången):

- Noder numrerade $0, 1, ..., |V| - 1$.
- Array `adjacent` med $|V|$ positioner.
- `adjacent[i]` innehåller grannlista (länkad lista) för nod `i`.

`r`: dynamisk array, `d`: array.

Pseudokod

Gärna mer detaljer och bättre namn på tenta.
Exempel:

```
// indegree är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
indegree = new array of size |V|

// Initialisera indegree.
for i in [0,...,|V|-1] do
  indegree[i] = 0
for i in [0,...,|V|-1] do
  for each direct successor j of i do
    indegree[j]++
```

Pseudokod

Gärna mer detaljer och bättre namn på tenta.
Exempel:

```
// indegree är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
indegree = new array of size |V|

// Initialisera indegree.
for i in [0,...,|V|-1] do                     O(|V|) ggr
  indegree[i] = 0                               O(1)
for i in [0,...,|V|-1] do
  for each direct successor j of i do
    indegree[j]++
```

Pseudokod

Gärna mer detaljer och bättre namn på tenta.
Exempel:

```
// indegree är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
indegree = new array of size |V|

// Initialisera indegree.
for i in [0,...,|V|-1] do                 O(|V|) ggr
  indegree[i] = 0                           O(1)
for i in [0,...,|V|-1] do                 O(|V|)
  for each direct successor j of i do     O(|E|) ggr
    indegree[j]++                           O(1)
```

Topologisk sortering: enkel algoritm (2)

```
r = new empty list                           O(1)
d = map from vertices to their indegrees     O(|V| + |E|)
    // null for nodes in r.

repeat |V| times                             O(|V|) ggr
  if d[v] == 0 for some v then                 O(|V|)
    r.add-last(v)                              O(1)
    d[v] = null                                O(1)
    for each direct successor v' of v do     O(|E|) ggr
      decrease d[v'] by 1                      O(1)
  else
    raise error: cycle found                   O(1)

return r  // Nodes, topologically sorted.    O(1)
```

Totalt: $O(|V|^2 + |E|) = O(|V|^2)$.

Topologisk sortering med kö

```
r = new empty list
d = map from vertices to their indegrees
q = queue with all nodes of indegree 0

while q is non-empty do
  v = q.dequeue()
  r.add-last(v)
  for each direct successor v' of v do
    decrease d[v'] by 1
    if d[v'] = 0 then
      q.enqueue(v')

if r.length() < |V| then
  raise error: cycle found

return r  // Nodes, topologically sorted.
```

Analysera värstafallstidskomplexiteten.

- $\Theta(|V|)$.
- $\Theta(|E|)$.
- $\Theta(|V| + |E|)$.
- $\Theta(|V|^2)$.

Bonusövning

Vad händer om man använder en stack istället för en kö?

Topologisk sortering med kö

```
r = new empty list                             Θ(1)
d = map from vertices to their indegrees       Θ(|V| + |E|)
q = queue with all nodes of indegree 0         O(|V|)

while q is non-empty do                        O(|V|) ggr
  v = q.dequeue()                                Θ(1)
  r.add-last(v)                                  Θ(1)
  for each direct successor v' of v do         O(|E|) ggr
    decrease d[v'] by 1                          Θ(1)
    if d[v'] = 0 then                            Θ(1)
      q.enqueue(v')                              Θ(1)

if r.length() < |V| then                       Θ(1)
  raise error: cycle found                     Θ(1)

return r  // Nodes, topologically sorted.      Θ(1)
```

Kortaste vägen

Kortaste vägen

Kostnad av väg $v_1, ..., v_n$:

$$\sum_{i=1}^{n-1} c_{i,i+1}$$

I oviktad graf: $n - 1$.

Kortaste vägen

Kortaste vägen-problem:

- Givet två noder u och v,
  hitta en kortaste väg från u till v.
- Givet nod u, för varje nod v,
  hitta en kortaste väg från u till v.
- Hitta kortaste vägen
  från varje nod till varje annan.

Oviktade grafer: bredden först-sökning

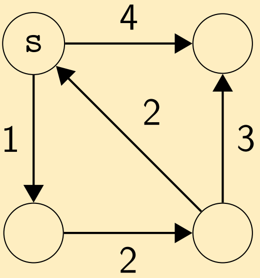
```
d = new array of size |V|, initialised to ∞
p = new array of size |V|, initialised to null
q = new empty queue

q.enqueue(s)
d[s] = 0

while q is non-empty do
  v = q.dequeue()
  for each direct successor v' of v do
    if d[v'] = ∞ then
      d[v'] = d[v] + 1
      p[v'] = v
      q.enqueue(v')

return (d, p)
```

Oviktade grafer: bredden först-sökning

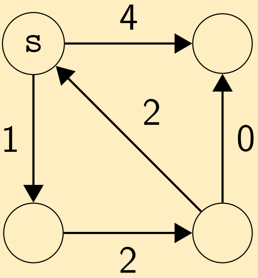
```
d = new array of size |V|, initialised to ∞     O(|V|)
p = new array of size |V|, initialised to null  O(|V|)
q = new empty queue

q.enqueue(s)
d[s] = 0

while q is non-empty do                       O(|V|) ggr
  v = q.dequeue()
  for each direct successor v' of v do        O(|E|) ggr
    if d[v'] = ∞ then
      d[v'] = d[v] + 1
      p[v'] = v
      q.enqueue(v')

return (d, p)
```

Fungerar algoritmen för viktade grafer (om `+ 1` byts ut mot `+ vikten av kanten från v till v'`)? Testa!

A:



B:



Viktade grafer

Viktade grafer: Dijkstras algoritm

- Observation: Kan behöva uppdatera kostnader flera gånger.
- Antagande: Inga negativa vikter.
- Grundidé:
  - Anta att vi redan känner till kortaste vägen till vissa noder.
  - Beräkna avståndet till alla de här nodernas direkta efterföljare (utom noderna själva).
  - Det kortaste av de här avstånden måste vara korrekt.

```
d    = new array of size |V|, initialised to ∞
p    = new array of size |V|, initialised to null
k    = new array of size |V|, initialised to false
d[s] = 0

repeat until no unknown node v' satisfies d[v'] < ∞

  v = one of the unknown nodes v' with smallest d[v']
  k[v] = true

  for each direct successor v' of v do
    if (not k[v']) and d[v'] > d[v] + c(v,v') then
       d[v'] = d[v] + c(v,v')
       p[v'] = v

return (d, p)
```

Viktade grafer: Dijkstras algoritm

- Enkel implementation:
  $O(|E| + |V|^2) = O(|V|^2)$.
- För täta grafer med $|E| = \Theta(|V|^2)$:
  linjär i grafernas storlek.

(Antagande: Viktoperationer tar konstant tid.)

```
d    = empty map from node indices (by default ∞)
p    = empty map from node indices
k    = empty set of node indices
q    = new empty priority queue
d[s] = 0
q.insert(s, 0)

while q is non-empty do
  v = q.delete-min()
  if v not in k then
    insert v into k
    for each direct successor v' of v do
      if (v' not in k) and d[v'] > d[v] + c(v,v') then
         d[v'] = d[v] + c(v,v')
         p[v'] = v
         q.insert(v', d[v'])

return (d, p)
```

Viktade grafer: Dijkstras algoritm

Med prioritetskö (binär heap eller leftistheap):

- Om `d`, `p` och `k` är arrayer:

$$O(|V| + 2|E| \log |E|) = O(|V| + |E| \log |V|).$$

- Med vissa andra avbildnings- och mängddatastrukturer:

$$O(|E| \log |V|).$$

- För täta grafer med $|E| = \Theta(|V|^2)$:

$$O(|V|^2 \log |V|).$$

Viktade grafer: Dijkstras algoritm

Kan också använda `decrease-key`:

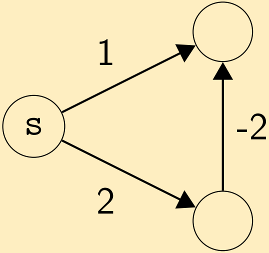
- Dubbletter i kön undviks.
- $O(|E| \log |V|)$.
- Krångligare (?), verkar ofta vara långsammare.

Giriga algoritmer

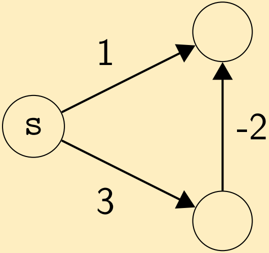
- Girig algoritm: Varje steg baserat på det som verkar bäst ”just nu”.

Ger Dijkstras algoritm rätt svar för följande grafer?

A:



B:



Sammanfattning

- Definition.
- Datastrukturer.
- Topologisk sortering.
- Kortaste vägen.

Nästa vecka:

- Minsta uppspännande träd.
- Djupet först-sökning.
- Dugga.