Grafalgoritmer 1

Douglas Wikström
KTH Stockholm
popup-help@csc.kth.se

Oriktade och riktade grafer

Definition. En **oriktad** graf består av en mängd noder V och en mängd kanter E, där en kant är ett **oordnat** par av noder.

Definition. En **riktad** graf består av en mängd noder V och en mängd kanter E, där en kant är ett **ordnat** par av noder.

Ibland kan det finnas både riktade och oriktade kanter i samma graf.

Allmänt om grafer

- Noder kallas även hörn.
- En graf har loopar om någon nod har en kant till sig själv.
- En graf har multipla kanter om det finns flera kanter (i samma riktning) mellan samma par av noder.
- Noder och/eller kanter kan vara märkta, dvs det kan finnas funktioner $f : V \to S$ and $h : E \to T$, för mängder S och T.
- I en kantviktad graf är $T = \mathbb{R}$, och oftast är $T = \mathbb{R}^+$.

Grafrepresentationer

En graf $G = (V, E)$ med $V = \{1, \ldots, n\}$ och $|E| = m$.

Grannmatris. G representeras av en 0/1-matris $A = (a_{ij})$ där $a_{ij} \neq 0 \iff (i, j) \in E$.

Snabb åtkomst, enkel att ändra, men kräver $O(n^2)$ minne. Kanter kan märkas genom att spara värden eller pekare i matrisen.

Grannlistor. G representeras av en vektor av längd n, där varje element i innehåller en lista på alla grannar till i.

Kräver $O(m)$ minne, men åtkomst är långsammare. Kanter kan märkas genom att spara värden eller pekare med listelementen.

Ett varningens ord

Inom matematiken och teoretiska datalogin är det oftast klart från sammanhanget vilken sorts grafer man talar om, dvs ordet *graf* används utan närmare bestämning.

Var noga med att själva veta vilket som avses i en algoritm. Många saker gäller för alla grafer, men problem som är lätt i en oriktad graf kan vara svårt i en riktad.

Ett varningens ord i den här kursen

Innan ni börjar på en uppgift:

- Tänk igenom vilken sorts graf som avses (riktad/oriktad, vikter på noder och kanter, multipla kanter, loopar).
- Tänk på vad ni kommer att behöva göra med grafen när en algoritm körs (märka om noder/kanter, ta bort noder/kanter, uppslagning, tilläggning, etc), eftersom det påverkar hur grafen bör representeras.

Minsta spännande träd

Problem. Givet en **oriktad, viktad, och sammanhängande** graf, hitta ett delträd av lägsta vikt som innehåller alla noder.

Hur gör vi detta?

Ett lemma om minsta spännande träd

Lemma. Låt $G = (V, E)$ vara ett en graf, låt $W \subset V$, och låt $e = (u, v)$ vara en kant av minsta vikt sådan att $u \in W$ och $v \in V \setminus W$. Då finns ett MST som innehåller (u, v).

Bevis. Om e inte är med i ett MST (V, E_T), så bildas en cykel om vi lägger till e. Cykeln innehåller en kant $f = (u', v')$ sådan att $u' \in W$ och $v' \in V \setminus W$ eftersom (V, E_T) är ett MST. Således är $(V, \{e\} \cup E_T - f)$ ett MST.

Kruskal's algoritm (1956)

Input: En oriktad viktad graf $G = (V, E)$.
Output: Minsta spännande träd för G.

KRUSKAL(G)

```
Sortera kanterna e_1, ..., e_m ∈ E så att
w(e_i) ≤ w(e_{i+1}), i = 1, ..., m − 1.
E_T ← ∅
for i = 1 to m
    if (V, E_T ∪ {e_i}) är acyclisk
        E_T ← E_T ∪ {e_i}
return (V, E_T)
```

Kruskal's algoritm (1956)

Input: En oriktad viktad graf $G = (V, E)$.
Output: Minsta spännande träd för G.

KRUSKAL(G)

(1) Sortera kanterna $e_1, \ldots, e_m \in E$ så att
(2) $w(e_i) \leq w(e_{i+1})$, $i = 1, \ldots, m-1$.
(3) $E_T \leftarrow \emptyset$
(4) **for** $i = 1$ **to** m
(5) **if** $(V, E_T \cup \{e_i\})$ är acyclisk
(6) $E_T \leftarrow E_T \cup \{e_i\}$
(7) **return** (V, E_T)

Korrekthet följer från lemmat. Det är oklart hur vi kollar om en cykel skapas, men det kan göras så att sorteringen blir det dyra steget, dvs körtiden är $O(m \log n)$.

Prim's algoritm (Jarník 1930, Prim 1957)

Input: En oriktad viktad graf $G = (V, E)$.
Output: Minsta spännande träd för G.

PRIM(G)

(1) $V_T \leftarrow \{1\}$, $E_T \leftarrow \emptyset$
(2) **while** $V_T \neq V$
(3) Hitta $(u, v) \in E \cap (V_T \times V \setminus V_T)$ med minimal $w(u, v)$.
(4) $E_T \leftarrow E_T \cup \{(u, v)\}$
(5) $V_T \leftarrow V_T \cup \{v\}$
(6) **return** (V, E_T)

Prim's algoritm (Jarník 1930, Prim 1957)

Input: En oriktad viktad graf $G = (V, E)$.
Output: Minsta spännande träd för G.

$\textsc{Prim}(G)$

(1) $V_T \leftarrow \{1\}$
(2) $p(u) \leftarrow \perp$, $c(u) \leftarrow \infty$ för $u \in V$
(3) $c(1) \leftarrow 0$, $H \leftarrow \textsc{MinHeap}_c(V)$
(4) **while** $V_T \neq V$
(5) $\quad v \leftarrow \textsc{GetHeapMin}(H)$
(6) $\quad V_T \leftarrow V_T \cup \{v\}$
(7) $\quad$ **foreach** $(v, u) \in E$ with $u \notin V_T$
(8) $\quad\quad$ **if** $w(v, u) < c(u)$
(9) $\quad\quad\quad c(u) \leftarrow w(v, u)$, $p(u) \leftarrow v$
(10) $\quad\quad\quad \textsc{UpdateHeap}(H, u)$
(11) **return** (V, E_T) där E_T är inducerad av $p(\cdot)$

Prim's algoritm (Jarník 1930, Prim 1957)

- Vi gör n extraktioner från heapen.
- Vi gör $2m$ updateringar av värden i heapen.

Med vanlig heap blir körtiden $O((n+m)\log n)$ eftersom extraktion/uppdatering kostar $O(\log n)$.

Med Fibonacci heap blir körtiden $O(m + n\log n)$, eftersom extraktion kostar $O(\log n)$ och uppdatering kostar $O(1)$ (amorterade kostnader).

Topologisk sortering

Problem. Givet är en **riktad acyklisk graf** $G = (V, E)$. Sortera noderna så att varje nod kommer före dess grannar som kan nås med ut-kanter.

Problem (alternativ formulering). En riktad acyklisk graf kan ses som en partiell ordning på noderna. Finn en linjär utvidgning av denna ordning.

Hur löser vi detta problem?

Kahn (1962)

Input: En riktad acyklisk graf $G = (V, E)$.
Output: Noderna i en topologisk ordning.

```
KAHN(G)
    Q ← QUEUE({u ∈ V : deg_in(u) = 0})
    i ← 1
    while Q ≠ ∅
        u ← POP(Q)
        ord(u) ← i
        i ← i + 1
        foreach (u, v) ∈ E
            E ← E − (u, v)
            if deg_in(v) = 0
                PUSH(Q, v)
    return ord
```

DFS-lösning

Input: En riktad acyklisk graf $G = (V, E)$.

Output: Noderna i en topologisk ordning.

$\textsc{TopoDFS}(G)$

(1) $\text{vis}(u) = \textit{false}$ och $\text{ord}(u) = \infty$ för $u \in V$

(2) $i = n$

(3) **foreach** $u \in V$

(4) **if** $\text{vis}(u) = \textit{false}$

(5) $i \leftarrow \textsc{InnerDFS}(u, i, \text{vis}, \text{ord})$

(6) **return** ord

$\textsc{InnerDFS}(u, i, \text{vis}, \text{ord})$

(1) $\text{vis}(u) \leftarrow \textit{true}$

(2) **foreach** $(u, v) \in E$

(3) **if** $\text{vis}(v) = \textit{false}$

(4) $i \leftarrow \textsc{InnerDFS}(v, i, \text{vis}, \text{ord})$

(5) $\text{ord}(u) \leftarrow i$

(6) **return** $i - 1$

DFS-lösning (med cykelkontroll)

```
InnerDFS(u, i, vis, ord)
    vis(u) ← true
    foreach (u, v) ∈ E
        if vis(v) = false
            i ← InnerDFS(v, i, vis, ord)
        else if ord(v) = ∞
            return ⊥
    ord(u) ← i
    return i − 1
```

Starkt sammanhängande komponenter

Problem. Givet är en **riktad graf** $G = (V, E)$. Två noder $u, v \in V$ är i samma starkt sammanhängande komponent om det finns en väg från u till v och en väg från v till u.

(Kanter får återanvändas och vägarna behöver inte besöka samma noder i båda riktningarna.)

Hur löser vi detta problem?

Starkt sammanhängande komponenter

1. Sortera noderna topologiskt.
2. Vänd på alla kanter.
3. Kör DFS på noderna enligt topologisk ordning. Alla noder som besöks i en DFS ingår i samma starkt sammanhängande komponent.

Starkt sammanhängande komponenter

Varför blir svaret korrekt?

Lemma. Låt C och C' vara två olika starkt sammanhängande komponenter i en graf $G = (V, E)$ och låt $(u, v) \in E^\top \cap (C' \times C)$. Då gäller $\min_{w \in C}\{\text{ord}(w)\} \leq \min_{w' \in C'}\{\text{ord}(w')\}$.

Här betyder $E^\top = \{(v, u) \mid (u, v) \in E\}$. Notera att grannmatrisen för E är transponat till grannmatrisen för E.

Starkt sammanhängande komponenter

Input: En riktad graf $G = (V, E)$.

Output: De starkt sammanhängande komponenterna i G.

$\mathrm{SCC}(G)$

(1) vis$(u) = \mathit{false}$ för $u \in V$

(2) $S = \emptyset$, ord $\leftarrow \textsc{TopoDFS}(G)$

(3) **for** $i = 1$ **to** n

(4) $\quad S \leftarrow S \cup \textsc{CollectDFS}(\mathrm{ord}^{-1}(i), \mathrm{vis}, \emptyset)$

(5) **return** S

$\textsc{CollectDFS}(u, \mathrm{vis})$

(1) **if** vis$(u) = \mathit{true}$

(2) $\quad$ **return** $\emptyset$

(3) **else**

(4) $\quad$ vis$(u) = \mathit{true}$, $C \leftarrow \emptyset$

(5) $\quad$ **foreach** $(u, v) \in E^{\top}$

(6) $\quad\quad C \leftarrow C \cup \textsc{CollectDFS}(v, \mathrm{vis})$

(7) **return** C

Kortaste avstånd från given nod med icke-negativa vikter

Problem. Låt $G = (V, E)$ vara en **riktad viktad graf** med **icke-negativa** vikter och $s \in V$. Hitta för varje v vägen från s till v med lägst totalvikt.

Hur gör vi detta?

Kortaste avstånd från given nod med icke-negativa vikter

Abstrakt idé 1. Hitta för $d = 1, 2, 3, \ldots$ alla nod(er) v som kan nås från s genom en väg av vikt högst d. Låt $p(v)$ vara en granne till s som rekursivt ger en väg som uppfyller denna garanti.

Kortaste avstånd från given nod med icke-negativa vikter

Abstrakt idé 1. Hitta för $d = 1, 2, 3, \ldots$ alla nod(er) v som kan nås från s genom en väg av vikt högst d. Låt $p(v)$ vara en granne till s som rekursivt ger en väg som uppfyller denna garanti.

Konkret idé 1. Tänk på en kant av vikt a som en kedja av längd a, och kör BFS från s, där varje nod får en pekare till sin förälder.

Kortaste avstånd från given nod med icke-negativa vikter

Abstrakt idé 1. Hitta för $d = 1, 2, 3, \ldots$ alla nod(er) v som kan nås från s genom en väg av vikt högst d. Låt $p(v)$ vara en granne till s som rekursivt ger en väg som uppfyller denna garanti.

Konkret idé 1. Tänk på en kant av vikt a som en kedja av längd a, och kör BFS från s, där varje nod får en pekare till sin förälder.

Abstrakt idé 2. Börja med en mängd $S = \{s\}$. Upprepa $n - 1$ gånger: lägg till S den nod $v \notin S$ som ligger närmast s. Låt $p(v)$ vara en granne i S som rekursivt ger en väg som uppfyller denna garanti.

Kortaste avstånd från given nod med icke-negativa vikter

Abstrakt idé 1. Hitta för $d = 1, 2, 3, \ldots$ alla nod(er) v som kan nås från s genom en väg av vikt högst d. Låt $p(v)$ vara en granne till s som rekursivt ger en väg som uppfyller denna garanti.

Konkret idé 1. Tänk på en kant av vikt a som en kedja av längd a, och kör BFS från s, där varje nod får en pekare till sin förälder.

Abstrakt idé 2. Börja med en mängd $S = \{s\}$. Upprepa $n - 1$ gånger: lägg till S den nod $v \notin S$ som ligger närmast s. Låt $p(v)$ vara en granne i S som rekursivt ger en väg som uppfyller denna garanti.

Konkret idé 2. Det räcker att hålla koll på grannarna till S. Välj som v den granne till S som har lägst $D(s, u) + w(u, v)$ för något $u \in S$ och sätt $p(v) = u$, där $D(s, u)$ är avstånd från s till u. Uppdatera grannarna till S med grannarna till v.

Dijkstra (1959)

Input: En riktad graf $G = (V, E)$ med icke-negativa vikter $w(\cdot)$, och en nod $u \in V$.

Output: Kortaste vägen från u till varje annan nod i V.

DIJKSTRA$(G, w(\cdot), s)$

(1) $p(u) \leftarrow \perp$, $d(u) \leftarrow \infty$ och $\text{vis}(u) = \textit{false}$ för $u \in V$.
(2) $d(s) \leftarrow 0$
(3) **for** $i = 1$ **to** n
(4) Hitta en nod u med $\text{vis}(u) = \textit{false}$ och minimal $d(u)$.
(5) $\text{vis}(u) \leftarrow \textit{true}$
(6) **if** $d(u) = \infty$ **then return** $\perp$
(7) **foreach** $(u, v) \in E$ with $\text{vis}(v) = \textit{false}$
(8) **if** $d(u) + w(u, v) < d(v)$
(9) $d(v) \leftarrow d(u) + w(u, v)$
(10) $p(v) \leftarrow u$
(11) **return** $p(\cdot)$

Dijkstra (1959)

En nod u med minimal $d(u)$ kan hittas precis som i Prim's algoritm och körtiden beror på samma sätt av hur detta implementeras.

Kortaste avstånd från given nod

Vad händer om vikterna är negativa?

Kortaste avstånd från given nod (negativa vikter)

Input: En graf G med viktade kanter och en startnod s
Output: Minsta avståndet från en nod s till alla andra noder i G

$\textsc{BellmanFord}(G, w(\cdot), s)$

```
p(u) ← ⊥ och d(u) ← ∞ för u ∈ V.
d(s) ← 0
for i = 1 to n − 1
    foreach (u, v) ∈ E
        if d(u) + w(u, v) < d(v)
            d(v) ← d(u) + w(u, v)
            p(v) ← u
return (d(·), p(·))
```

Att upptäcka negativa cykler

Input: En graf G med viktade kanter och en startnod s, och en Bellman-Ford lösning $(d(\cdot), p(\cdot))$.

Output: *true* eller *false* beroende på om G har en negativ cykel eller inte.

$\textsc{NegCycles}(G, w(\cdot), s, d(\cdot), p(\cdot))$

```
(1)  foreach (u, v) ∈ E
(2)      if d(u) + w(u, v) < d(v)
(3)          return true
(4)  return false
```

Kortaste

Lemma. Efter i iterationer i for-loopen gäller:

- Om $d(u) \neq \infty$ så finns en väg av längd $d(u)$ från s till u.
- Om det finns en väg från s till u med högst i kanter, så är $d(u)$ längden av en kortaste väg från s till u med högst i kanter.