Föreläsning 2. Kortaste vägar i grafer.

Problem: KORTASTE VÄGAR

Den enklaste varianten är om vi inte har *kantvikter* och kortaste väg är en väg med så få kanter som möjligt.

Indata: En riktad graf G och en startnod s.
Mål: Kortaste väg från s till alla andra noder.
Metod: Sök igenom grafen enligt Bredden-Först-Sökning (BFS)

Algoritm

BFS tar en graf och går igenom hörnen ett i taget. Algoritmen börjar i starthörnet s och tar sedan grannarna till s, grannarna till grannarna till s etc. En kö används för att hålla reda på ordningen i vilken hörnen besöks. Det oviktade avståndet från s beräknas samtidigt.

```
BFS(V, E, s)
(1)   d[u] ← ∞ för alla u ∈ V
(2)   d[s] ← 0
(3)   Q ← {s}
(4)   while Q ≠ ∅
(5)       u ← Dequeue(Q)
(6)       foreach granne v till u
(7)           if d[v] = ∞
(8)               d[v] ← d[u] + 1
(9)               p[v] = u
(10)              Enqueue(Q, v)
```

Tidskomplexitet: $O(|V| + |E|)$

Grafer med kantvikter

Om vi har kantvikter som längden på en väg räknas som summan av kantvikterna längs vägen får vi ett svårare problem. Vi har två fall:
1. Alla kantvikter är ≥ 0.
2. Vi tillåter kantvikter < 0.
Det första fallet är betydligt enklare än det andra. Det första kan lösas med Dijkstras algoritm.

Dijkstras algoritm

```
DIJKSTRA(G = ⟨V, E⟩,s, t ∈ V, l : E → R)
(1)     foreach u ∈ V
(2)         Märk u med d[u] = ∞
(3)     foreach u så att (s, u) ∈ E
(4)         Märk u med d[u] = w(s, u)
(5)     Märk s med d[s] = 0
(6)     S ← {s}
(7)     while t ∉ S
(8)         Välj u ∉ S med d[u] minimal.
(9)         S ← S ∪ {u}
(10)        foreach v ∉ S så att (u, v) ∈ E
(11)            if d[v] > d[u] + w(u, v)
(12)                d[v] ← d[u] + w(u, v)
(13)                p[v] ← u
(14)    return Märkningen hos t
```

G antas representerad i form av grannlistor.
När algoritmen är klar markerar $d[v]$ kortaste avståndet från s till v. Kortaste vägen fås baklänges som $p[v], p[p[v]], ...$

Tidskomplexitet:
Den inledande märkningen tar $O(|V|)$ tid. Att hitta hörnet som S ska utvidgas med tar $O(|V|)$ tid, och detta utförs högst $|V|$ gånger. För varje kant i grafen kommer högst en uppdatering ske.
Totalt: $O(|V| + |V|^2 + |E|) = O(|V|^2)$
(eftersom $|E| \in O(|V|^2)$)

Korrekthetsbevis

Vi antar för enkelhets skull att alla noder kan nås från s. Låt $\delta[v]$ betyda kortaste vägen från s till v och $\delta_S[v]$ betyda kortaste vägen som går genom noder i S, förutom v.
Vi använder följande invariant för den yttersta slingan:

Alla noder u i S är märkta med $d[u] = \delta[u]$. Dessutom är alla $v \notin S$ med $d[v] < \infty$ märkta med $d[v] = \delta_S[v]$. Noderna märkta med $d[v] = \infty$ går inte att nå med en S-stig.

Invarianten är sann innan slingan börjar köras. Antag att den är sann i ett steg i algoritmen. I nästa steg läggs u till. Vi måste visa att $d[u] = \delta[u]$. Antag att $d[u] \neq \delta[u]$. Då måste $d[u] > \delta[u]$. I den kortaste vägen från s till u finns en första nod x som ligger utanför S. Då gäller $\delta[u] = \delta[x] + \delta[x, u]$ där $\delta[x, u]$ är kortaste avståndet mellan x och u. Eftersom kantvikterna är positiva är $\delta[x, u] \geq 0$. Men $\delta[x] = \delta_S[x] = d[x]$ enligt antagandet. Då gäller $\delta[u] = d[x] + \delta[x, u] \geq d[x] \geq d[u]$. Men det är omöjligt.

Vi lägger nu till u till S. Kalla den nya mängden S'. Vi märker om noderna utanför S'. Vi måste visa att $d[v] = \delta_{S'}[v]$ för de ommärkta noderna. Vi antar att $v \notin S'$ är granne till u och att v har märkningen $d[v]$ före en eventuell ommärkning. Om det finns en kortaste väg till v i S' som inte innhåller u så måste $d[v] = \delta_{S'}[v] \leq \delta_S[u] + w(u, v)$ och noden v kommer inte att märkas om. Om $d[v] = \delta_{S'}[v] = \delta_S[v]$ så är $\delta_{S'}[s, v] \leq \delta[s, u] + w(u, v) = d[u] + w(u, v)$ Noden kommer inte att märkas om utan behåller märkningen $d[v] = \delta_S[s, v] = \delta_{S'}[s, v]$.

Om $\delta_{S'}[v] < \delta_S[v]$ så måste kortaste S'-vägen gå över u. Men då måste u vara sista noden före v eftersom alla kantvikter är positiva. Då får vi $\delta_{S'} = d[u] + w(u, v)$ och v får märkningen $d[v] = \delta_{S'}$.

När man går ur slingan har man beräknat korrekt kortaste avstånd $d[t]$ från s till t.

Negativa kantvikter tillåtna

Om en graf har negativa kantvikter men **inga negativa cykler** så finns kortaste vägar. Men de kan inte hittas med Dijkstras algoritm. Mer avancerad algoritm krävs.

Om en graf har negativa cykler finns följande val.
1. Acceptera att kortaste avstånd inte kan beräknas och förkasta grafen.
2. Leta efter kortaste stigar istället för vägar. Ingen effektiv algoritm som gör detta är känd.
3. Leta efter kortaste vägar med begränsning att de får innehålla högst k kanter. Detta går att göra relativt enkelt. (Men är det det man är ute efter?)

Bellman-Fords algoritm

Följande algoritm löser problemet enligt fall 1 ovan. Vi antar att varje nod kan nås från s. Målet är då att hitta kortaste vägar från s till varje annan nod.

```
BELLMAN-FORD(G = ⟨V, E⟩, s ∈ V, w : E → R)
(1)   foreach u ∈ V
(2)       d[u] ← ∞
(3)   d[s] ← 0
(4)   for i = 1 to |V| − 1
(5)       foreach (u, v) ∈ E
(6)           if d[v] > d[u] + w(u, v)
(7)               d[v] ← d[u] + w(u, v)
(8)               p[v] ← u
(9)   foreach (u, v) ∈ E
(10)      if d[v] > d[u] + w(u, v)
(11)          return Negativ cykel!
(12)  return d, p
```

Komplexitet

Inledande märkningen tar $O(|V|)$ steg. Yttre slingan tar $O(|V|)$ steg. Inre slingan tar $O(|E|)$ steg. Det ger tillsammans $O(|V||E|)$ steg. Den sista kontrollen tar $O(|E|)$ steg. Totalt får vi $O(|V||E|) = O(|V|^3)$. Det är sämre än Dijkstras algoritm.

Korrekthetsanalys (skiss)

Vi antar först att grafen inte har negativa cykler. Då existerar kortaste vägar. Det verkliga kortaste avståndet från s till x kallar vi $\delta[x]$. Vi låter $\delta_k[x]$ betyda det kortaste avståndet om vi bara får använda vägar med högst k st kanter. För alla x och k gäller $d[x] \leq \delta_k[x]$ och $\delta[x] = \delta_{n-1}[x]$ där n är antalet noder.
Vi använder följande invariant för yttre slingan:

Efter iteration k gäller för alla noder u att $d[u] \leq \delta_k[u]$

Påståendet är sant för $k = 0$. Vi antar att det är sant för k och visar att det då är sant för $k + 1$ också. Om u kan nås med en väg som innehåller högst $k + 1$ kanter finns det en kortaste sådan väg. Antag att noden före u på en sådan väg är x. Då måste $\delta_{k+1}[u] = \delta_k[x] + w(x, u)$. Om inte $d[x]$ har hunnit bli ändrat redan i iteration $k + 1$ är $d[x] = \delta_k[x]$ och $d[u]$ kommer att sättas till $\delta_{k+1}[u] = \delta_k[x] + w(x, u)$. Sedan kan $d[u]$ förstås sänkas under iterationen. Om $d[x]$ redan

hunnit märkas om gäller $d[x] < \delta_k[x]$. Då kommer och enligt antagandet är $d[u]$ att få ett värde $d[u] \leq d[x] + w(x,u) < \delta_k[x] + w(x,u) \leq \delta_{k+1}[u]$. I vilket fall kommer $d[u] \leq \delta_{k+1}[u]$ när iterationen är klar. Efter $n-1$ iterationer får vi $d[u] \leq \delta_{n-1}[u] = \delta[u]$.

Låt oss kalla en kant (u,v) med $p[v] = u$ för *märkt*. Det går att se att de märkta kanterna hela tiden kommer att bilda ett träd rotat i s och med kanter riktade ut från s. Genom att jämföra δ och d längs vägarna i detta träd finner vi att $\delta[u] > d[u]$ är omöjligt. Alltså måste $d[u] = \delta[u]$ för alla u. Vi ser också att $d[v] > d[u] + w(u,v)$ är omöjligt eftersom det skulle ge $\delta[v] > \delta[u] + w(u,v)$. Så algoritmen *godkänner* grafen.

Antag nu att grafen innehåller en negativ cykel med noderna $u_1, u_2, ..., u_r$. När algoritmen är klar måste det finnas u, v med $d[v] > d[u] + w(u,v)$. Om det inte gjorde det så skulle vi ha $d[u_{i+1} \leq d[u_i] + w(u_i, u_{i+1})$ och $d[u_1] \leq d[u_r] + w(u_r, u_1)$ för $i = 1, 2, ..., r$. Då skulle summan $(d[u_2] - d[u_1]) + (d[u_3] - d[u_2]) + ... + (d[u_1] - d[u_r])$ vara både 0 och $w(u_1, u_2) + ... + w(u_r, u_1)$. Eftersom cykeln är negativ är det omöjligt och det måste finnas i med $d[u_{i+1}] > d[u_i] + w(u_i, u_{i+1})$. Den olikheten kommer sista delen av algoritmen att upptäcka.