PoP øvelsesopaver uge 16-17

Christoffer Øhrstrøm

15. januar 2016

Formålet med disse øvelser er at få emnerne fra PoP (og DMA) til at sidde godt fast. Du kommer bl.a. til at bruge følgende emner:

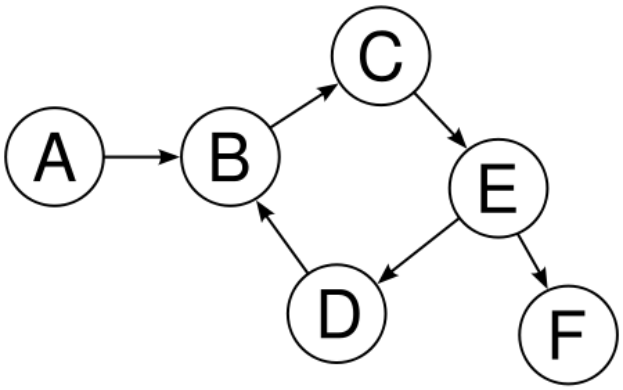
- Datastrukturer (grafer, lister, hash tabeller og prioritetskøer)
- Ind- og udlæsning af filer samt parsing af disse
- Klasser og herunder generiske klasser (nyt stuff)

I DMA har vi studeret grafer, og vi skal her bruge det i to praktiske sammenhænge: Terrorbekæmpelse og flyrejser. Afsnit 1 opsummerer kort graf repræsentationen som kendt fra CLRS. (Det kan springes over, hvis du føler dig fortrolig med grafer). I afsnit 2 skal du udarbejde to klasser, der skal bruges for at arbejde med grafer. De to sidste afsnit (3 og 4) omhandler anvendelsen af grafer med fokus på implementering af udvalgte algoritmer.

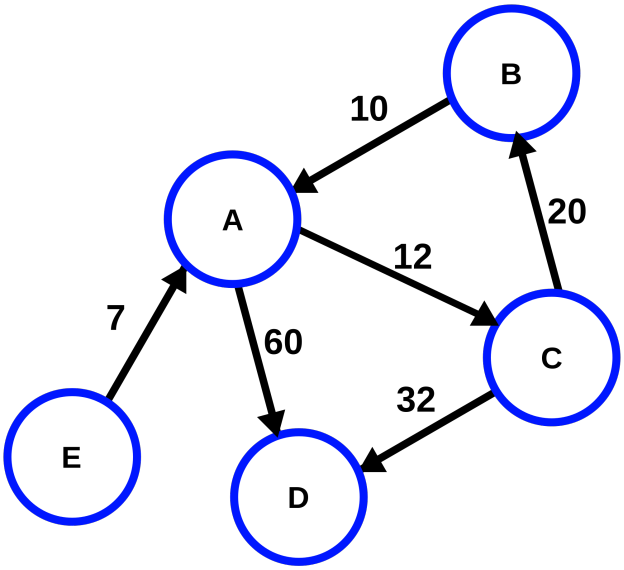
Opgaverne bruger algoritmer for mindst udspændte træer og korteste vej. Bilag A viser Prims algoritme til at finde mindst udspændte træer. Bilag B viser Dijkstras algoritme til at finde korteste vej i en graf. Da både Prims og Dijkstras algoritmer anvender prioritetskøer, kan du læse lidt om implementering af disse i bilag C.

1 Grafer

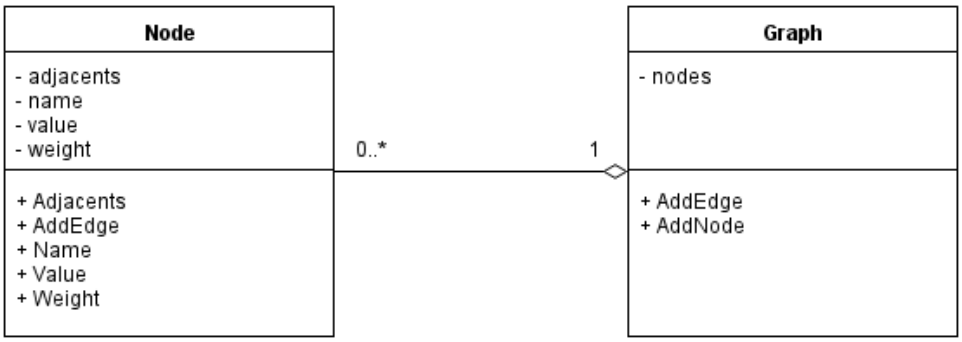
En graf $G = (V, E)$ er repræsenteret ved dens knuder $G.V$ og dens kanter $G.E$. I de følgende opgaver antages det, at grafens kanter er repræsenteret som en adjacency list. Det vil sige, at hver knude v i grafen skal have en liste $v.Adj$, der peger på alle knuder, som v har udgående kanter til. I figur 1 har knuden E de udgående kanter (E, D) og (E, F). Derfor er $E.Adj = [D, F]$. Ofte arbejder man også på vægtede grafer. En vægtet graf er defineret ved en graf, hvor der findes en funktion w, som tillægger en vægt til alle kanter. Figur 2 viser et eksempel på en vægtet graf. Betragt eksempelvis kanten (A, D). Da er $w(A, D) = 60$.



Figur 1: Eksempel på en graf.



Figur 2: Eksempel på en vægtet graf



Figur 3: Klassediagram over forslag til udgangspunkt af design af de to klasser til at arbejde med grafer.

2 Klasser

Du skal nu konstruere to klasser: Node og Graph. Du bestemmer selv, hvilke attributter og metoder, som klasserne skal have, men det kan være en god idè at tage udgangspunkt i UML diagrammet i figur 3. Bemærk at vægten her er repræsenteret ved en attribut i stedet for en funktion, som det bliver gjort i CLRS. De to klasser skal tilsammen kunne håndtere både vægtede og ikke-vægtede grafer samt orienterede og ikke-orienterede grafer.

Node har attributten value. For at gøre Node (og Graph) generisk nok til at kunne håndtere forskellige typer værdier, skal klassen gøres generisk. Dette har vi ikke arbejdet tidligere med i PoP, men du kan lære om det her: `https://msdn.microsoft.com/en-us/library/dd233215.aspx`

3 Terrorbekæmpelse

I data mappen (udleveret sammen med PDF'en) findes der en mappe kaldet *train-bombing*, som indeholder data over mistænkte personer i terrorangrebet i Madrid d. 11. marts 2004 [2]. Hver knude repræsenterer en mistænkt person, og hver kant repræsenterer hvor stærkt et forhold to personer har på en skala fra 1 til 4:

1. Tillid/venskab
2. Forbindelser til Al-Qaeda og Osama Bin Laden
3. Har begge været i militærtræning/krig sammen
4. Har begge deltaget i tidligere terror angreb

Du er blevet hyret af NSA til at analysere dataen og finde ud af, hvilken person der udgør den største trudsel. Den største trudsel skal her forstås som, personen der har det farligste netværk. Du indser, at dette kan løses ved at finde størst udspændte træer for alle personer og vælge personen, hvor summen af kanternes vægt i personens størst udspændte træ er størst mulig. (Der kan potentielt være flere personer, som har den maksimale sum. I så fald vælger du en af dem efter eget valg.) Der er dog et problem: Du kender kun algoritmer til at finde mindst udspændte træer. Din opgave er nu:

1. Indlæs dataen i *nodes.csv* samt *edges.csv* i *train-bombing* mappen
2. Konstruer en graf ud fra dataen
3. Find ud af hvordan du kan beregne størst udspændte træer ved at anvende algoritmer til at beregne mindst udspændte træer
4. Udskriv personen hvor summen af kanternes vægt i personens træ er størst mulig

Den interesserede læser kan finde mere information om dette emne i [1].

4 Flyrejser

I data mappen (udleveret sammen med PDF'en) findes der en mappe kaldet *airports*, som indeholder data over lufthavne og afgange på fly i verdenen [3]. En knude repræsenterer en lufthavn, og en kant mellem to knuder betyder, at der findes mindst èn rute fra en lufthavn til en anden. Hver kant har også en vægt, der angiver antallet af fly fra den ene lufthavn til den anden.

Du er ansat som *forward deployed engineer* i en større IT virksomhed med kunder i hele verdenen. Dit arbejde er at rejse rundt til kunderne og integrere din virksomheds software med kundernes eksisterende systemer. Grundet dit arbejdes natur flyver du umenneskeligt meget, og du er interesseret i at flyve så lidt som muligt. Du kommer så på den fantastiske idè at lave et program, som kan finde den hurtigste rejse fra A til B. For ikke at overkomplificere problemet antager du, at du kommer hurtigst frem ved at vælge en rute med flest mulige fly (kanten med størst vægt). Dette gør problemet til et længste vej problem, men du kender kun algoritmer til korteste vej. Din opgave er nu:

1. Indlæs dataen i *nodes.csv* samt *edges.csv* i *airports* mappen
2. Konstruer en graf ud fra dataen
3. Find ud af hvordan du kan beregne længste vej ved at anvende algoritmer til at beregne korteste vej

4. Udskriv alle lufthavne (inklusiv A og B) på den hurtigste vej fra A til B

Referencer

[1] Brian Hayes. "Connecting the Dots". I: *American Scientist* 94 (2006). URL: http://bit-player.org/wp-content/extras/bph-publications/AmSci-2006-09-Hayes-NSA.pdf.

[2] *Train bombing network dataset*. Maj 2015. URL: http://konect.uni-koblenz.de/networks/moreno_train.

[3] *Us airports network dataset*. Maj 2015. URL: http://konect.uni-koblenz.de/networks/opsahl-usairport.

A Prims algoritme for mindst udspændte træer

Prims algoritme er beskrevet i CLRS side 634-636. Algoritme 1 viser pseudokoden for Prims algoritme. Bemærk at algoritmen bruger Q som en prioritetskø. Se bilag C for kort information om prioritetskøer i F#.

Algorithm 1 MST-Prim(G,w,r)

```
for all u ∈ G.V do
    u.key = ∞
    u.π = NIL
end for
r.key = 0
Q = G.V
while Q ≠ ∅ do
    u = Extract-Min(Q)
    for all v ∈ G.Adj[u] do
        if v ∈ Q and w(u,v) < v.key then
            v.π = u
            v.key = w(u,v)
        end if
    end for
end while
```

B Dijkstras algoritme for korteste vej

Dijkstras algoritme er beskrevet i CLRS side 658-662. Algoritme 4 viser pseudokoden for Dijkstras algoritme. Algoritmen bruger hjælpefunktionen *Initialize-Single-Source*, som kan ses i algoritme 2, og hjælpefunktionen Relax, som kan ses i algoritme 3. Bemærk at algoritmen bruger Q som en prioritetskø. Se bilag C for kort information om prioritetskøer i F#.

Algorithm 2 Initialize-Single-Source(G,s)

```
for all v ∈ G.V do
    v.d = ∞
    v.π = NIL
end for
s.d = 0
```

C Prioritetskøer i F#

Du kan enten lave din egen prioritetskø (hint: brug en hob) eller finde en implementering på nettet. Fx kan du bruge: `https://code.msdn.microsoft.`

Algorithm 3 Relax(u,v,w)

if $v.d > u.d + w(u, v)$ **then**
  $v.d = u.d + w(u, v)$
  $v.\pi = u$
end if

Algorithm 4 Dijkstra(G,w,s)

$S = \varnothing$
$Q = G.V$
while $Q \neq \varnothing$ **do**
  $u =$ Extract-Min(Q)
  $S = S \cup \{u\}$
  for all $v \in G.Adj[u]$ **do**
    Relax(u, v, w)
  end for
end while

`com/Net-Implementation-of-a-d3ac7b9d`. Hvis du vælger selv at implementere din prioritetskø, så anbefales det at kigge i CLRS kapitel 6 (side 151-169).