09. Minimale spenntrær

Forelesning 9

Her har vi en graf med vekter på kantene, og ønsker å bare beholde akkurat de kantene vi må for å koble sammen alle nodene, med en så lav vektsum som mulig. Erke-eksempel på grådighet: Velg én og én kant, alltid den billigste lovlige.

Pensum

- ☐ Kap. 19. Data structures for disjoint sets: Innl., 19.1 og 19.3
- ☐ Kap. 21. Minimum spanning trees

Læringsmål

- [I1] Forstå *skog*-implementasjonen av *disjunkte mengder*, inkl. CONNECTED-COMPONENTS
- [I2] Vite hva *spenntrær* og *minimale spenntrær* er
- [I3] Forstå GENERIC-MST
- [I4] Forstå hvorfor *lette kanter* er *trygge kanter*
- [I5] Forstå MST-KRUSKAL
- [I6] Forstå MST-PRIM

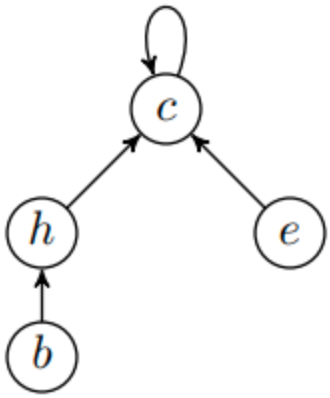
📚Disjunkte mengder

Disjunkte mengder kommer til nytte når vi ønsker å finne spanntrær i en graf. Her kommer litt nyttig terminologi.

- Delgraf: graf som består av en delmengde av nodene og kantene til den opprinnelige grafen.
- Spenngraf: en dekkende delgraf, graf med samme nodesett som originalgrafen. (uvanlig begrep på norsk)
- Spennskog: en asyklisk spenngraf. (uvanlig begrep på norsk)
- Spenntrær: en sammenhengende spennskog, altså $|E| = |V| - 1$.

Disjunkte mengder er en effektiv måte å gruppere noder inn i sammenkoblede komponenter ved å introdusere en

“parent” til hver node. Noden på topp peker deretter på seg selv.



I tillegg har hver node en rang som beskriver hvor høyt opp i hierarkiet den ligger. I andre tilfeller kan denne komme til nytte, men i tilfellet med minimale spanntrær ønsker vi kun å gruppere nodene inn i komponenter. Denne rangen brukes for å merge to komponenter (for å bestemme hvilken node som dominerer). Relevant funksjonalitet fra pensom er:

- $MakeSet$ - Lag et sett med ett element av en node. Setter parent til seg selv, og rang til 0. (I konstant tid $O(1)$)
- $FindSet$ - Finn representanten (øverste parent) til mengden som inneholder den en gitt node, ved amortisert analyse i $O(\alpha(n))$ tid.
- $Union$ - Merge to sett sammen, hvor settet med høyest rang på øverste parent dominerer den andre. Hvis begge er like, velges en vilkårlig.
- $Link$ - Lenker et sett til et annet, hvor settet med høyest rang på øverste parent dominerer den andre. (Kalt av $Union$)

$$m \text{ operasjoner: } \mathrm{O}(m \cdot \alpha(n))$$

$$\alpha(n) = \begin{cases} 0 & \text{hvis } 0 \leq n \leq 2\,, \\ 1 & \text{hvis } n = 3\,, \\ 2 & \text{hvis } 4 \leq n \leq 7\,, \\ 3 & \text{hvis } 8 \leq n \leq 2047\,, \\ 4 & \text{hvis } 2048 \leq n \leq 16^{512}\,. \end{cases}$$

- $ConnectedComponents$ - Danne sammenkoblede komponenter av en graf. Dette kan gjøres i tilnærmet $O(V + E)$ tid (variasjon, basert på amortisert kjøretid av $FindSet$).
- $SameComponents$ - Finn ut om to noder er i samme komponent, i tilnærmet $O(1)$ tid (variasjon, basert på amortisert kjøretid av $FindSet$).

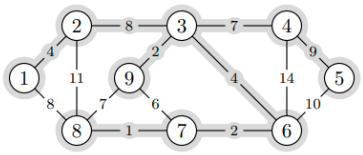
💻Generisk MST

Kanter kan ha vekter som betyr at vi til tider ønsker minimale eller maksimale spenntrær. Realistisk eksempel kan være bygging av veier mellom byer. Et minimalt spenntre er spenntreet som knytter sammen nodene med minst vekt mulig, og er et praktfult eksempel på en grådig algoritme.

Input: En urettet graf $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ og en vektfunksjon $w : \mathrm{E} \to \mathbb{R}$.

Output: En asyklisk delmengde $\mathrm{T} \subseteq \mathrm{E}$ som kobler sammen nodene i V og som minimerer vektsummen

$$w(\mathrm{T}) = \sum_{(u,v) \in \mathrm{T}} w(u, v) .$$



Delmengde $\mathrm{T} \subseteq \mathrm{E}$ som spenner over V og minimerer $\sum_{e \in \mathrm{T}} w(e)$

Eksempelet i forelesning bruker kantkontraksjon. Her vil hvert alternativ gi en ny, mindre delinstans. Dette passer godt til dynamisk programmering, men det er mye enklere å løse problemet på grådig vis.

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

Pensum bygger videre på dette prinsippet gjennom to algoritmer som opererer på litt forskjellig vis:

- Kruskal: en kant med minimal vekt blant de gjenværende er trygg så lenge den ikke danner sykler
- Prim: bygger ett tre gradvis; en lett kant over snittet rundt treet er alltid trygg

💻Kruskals algoritme

Kruskals algoritme fungerer ved å splitte opp grafen i to skoger som man jobber med; et tre man bygger til et MST, og den andre som er en skog av disjunkt mengder.

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  create list of edges in G.E
5  sort edge list by w
6  for each edge (u, v) in edge list
7      if FIND-SET(u) ≠ FIND-SET(v)
8          A = A ∪ {(u, v)}
9          UNION(u, v)
10 return A
```

Operasjon	Antall	Kjøretid
MAKE-SET	V	O(1)
Sortering	1	O(E lg E)
FIND-SET	O(E)	O(lg V)
UNION	O(E)	O(lg V)

Totalt: $O(E \lg V)$

$$|E| < |V|^2 \implies \lg|E| < 2\lg|V| \implies \lg E = O(\lg V)$$

💻Prims algoritme

Prims algoritme kan implementeres på forskjellige måter, men pensum dekker en metode lik BFS ved hjelp av en min-prioritets-kø. Prioriteten er vekten på den letteste kanten mellom noden og treet.

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = ∅
 6  for each u ∈ G.V
 7      INSERT(G, u)
 8  while Q ≠ ∅
 9      u = EXTRACT-MIN(Q)
10      for each v ∈ G.Adj[u]
11          if v ∈ Q and w(u, v) < v.key
12              v.π = u
13              v.key = w(u, v)
14              DECR-KEY(Q, v, w(u, v))
```

Operasjon	Antall	Kjøretid
BUILD-MIN-HEAP	1	O(V)
EXTRACT-MIN	V	O(lg V)
DECREASE-KEY	E	O(lg V)

Totalt: O(E lg V)