

08. Traversering av grafer

Forelesning 8

Vi traverserer en graf ved å besøke noder vi vet om. Vi vet i utgangspunktet bare om startnoden, men oppdager naboene til dem vi besøker. Traversering er viktig i seg selv, men danner også ryggraden til flere mer avanserte algoritmer.

Pensum

- ☐ Innledningen til del VI
- ☐ Kap. 20. Elementary graph algorithms: Innl. og 20.1–20.4
- ☐ Appendiks E i pensumheftet

Læringsmål

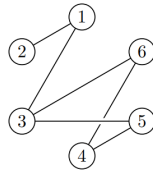
- [H₁] Forstå hvordan grafer kan implementeres
- [H₂] Forstå BFS, også for å finne *korteste vei uten vektor*
- [H₃] Forstå DFS og *parentesteoremet*
- [H₄] Forstå hvordan DFS *klassifiserer kanter*
- [H₅] Forstå TOPOLOGICAL-SORT
- [H₆] Forstå hvordan DFS kan *implementeres med en stakk*
- [H₇] Forstå *traverseringstrær* (som *bredde-først-* og *dybde-først-trær*)
- [H₈] Forstå *traversering med vilkårlig prioritetskø*

Grafrepresentasjoner

Det finnes flere måter å representere en graf i minne. I pensum er nabomatriser og nabolister i fokus. Representasjonene har noen forskjeller, og man sier ofte at nabomatriser er raskere, men de tar mer plass i minne.

Nabomatriser har en fordel siden det er mulig å slå opp kanter i grafen direkte, men i hovedsak bruker vi nabolister. Dessuten er det andre faktorer som veier inn på hvilken representasjon vi bruker, og som nevnt finnes det mange andre representasjoner.

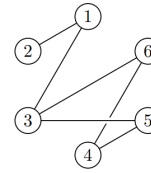
	1	2	3	4	5	6
1		1	1			
2	1					
3	1				1	1
4					1	1
5			1	1		
6			1	1		



Egnet til raske oppslag; ikke så egnet til traversering

16

1	→	2	3
2	→	1	
3	→	6	1
4	→	6	5
5	→	3	4
6	→	3	4



Kompakt; egnet til traversering; ikke så egnet til raske oppslag

23

Nabomatriser egner seg til direkte oppslag. Nabolister egner seg til traversering. Nabolister tar også mindre plass dersom grafen har få kanter – men ikke ellers!

Traversering av grafer

Traversering av en graf kan kjøres på mange forskjellige måter, og pensum har tidligere vært innom traversering av et trær. Målet med å traversere en graf er å unngå å besøke samme node to ganger, selv om grafen har sykler. Pensum nevner to enkle måter å traversere grafer på;

- Slett noden fra grafen
- Marker noder i grafen med farger, nåværende node som grå, besøkte som svart, og ubesøkte som hvite.

TRAVERSE'(G, u)

```

1  print  $u$ 
2   $u.color = \text{GRAY}$ 
3  for each  $v \in G.Adj[u]$ 
4      if  $v.color == \text{WHITE}$ 
5          TRAVERSE'( $G, v$ )
6   $u.color = \text{BLACK}$ 
```

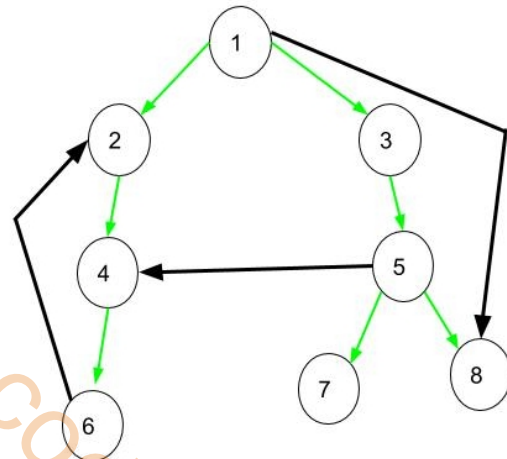
Depth-first search

Depth-first search er en traverseringsalgoritme som går så dypt igjennom kantene i grafen som mulig før den backtracker og fortsetter i bredden. Dette skjer fordi depth-first search besøker alle nodene i grafen, og for hver node besøker alle nodene som er koblet til den nåværende noden. På denne måten vil den alltid nå “bunnen” før den fortsetter i bredden.

- Kompleksitet på $O(V + E)$ siden algoritmen besøker hver node og hver kant.

I depth-first search er det vanlig å klassifisere nodene. Vi har følgende klasser:

- Tre-kanter: kanter som er i dybde-først skogen, altså når du møter en hvit node.
- Bakoverkanter: kanter til en forhjenger i skogen, altså når du møter en grå node.
- Forkanter: kanter som ikke er en del av DFS-skogen.
- Krysskanter: kanter som er ikke har noen ancestor/descendant relasjon



Dessuten er det interessant å merke seg at i dynamisk programmering utfører vi implisitt DFS på delproblemene.

Breadth-first search

Breadth-first search er en traverseringsalgoritme som går så bredt igjennom kantene i grafen som mulig før den fortsetter i dybden. Den kan også brukes til å finne korteste vei fra en node til alle. Breadth-first search bruker en kø over noder den ønsker å besøke, og kan på denne måten sikre seg at den søker i bredden først.

- Kompleksitet på $O(V + E)$ siden algoritmen besøker hver node og hver kant.

Topological sort

Topological sort er en traverseingsmetode på en DAG som gir nodene en viss rekkefølge; foreldre kommer før barn.

Det finnes flere måter å implementere topological sort på, og forelesning tar en kjapp titt på en versjon som fjerner "sources" fra grafen. En DAG har minst en "source", en node uten noen inn-kanter.

Typisk implementasjon av topological sort bruker en algoritme lik DFS med en Stack. Her kjører man DFS over grafen, deretter sortere rangere etter synkende finish-tid.

- Kompleksitet på $O(V + E)$ siden algoritmen besøker hver node og hver kant.

<https://jun.codes/>