Avgjørelsesproblemer, P og NP

Hva er avgjørelsesproblemer?

- Avgjørelsesproblem er et problem hvor svaret er kun JA eller NEI.
- F.eks, Hvis vi har "Sorted" så vil avgjørelsesproblemet i "Sorted" beskrives om en liste er sortert eller ikke (JA for sortert, NEI for ikke sortert)
- Et annet eksempel, Hvis vi har "Reachability" som vil si at vi har et par av noder i en graf, så vil avgjørelsesproblemet i "Reachability" beskrives om det finnes en sti mellom to par av noder i en graf.
- En "instans" av et problem er inputtet som er gitt

Reachability	
Instans:	En graf G og to noder v, w
Spørsmål:	Finnes det en sti fra v til w i G?

Sorted	
Instans:	En liste L
Spørsmål:	Er L sortert?

- JA-instans: Dvs, instanser som fører til output JA. Tar vi inn en sortert liste som ser slik ut [1,2,3,5,6,9] så tilsvarer dette en JA-instans av Sorted.
- NEI-instans: Dvs, instanser som fører til output NEI. Tar vi inn en usortert liste som ser slik ut [9,10,2,3,4] så tilsvarer dette en NEI-instans av Sorted.

Kompleksitetsklasser

- Man ønsker å klassifisere problemer etter hvor vanskelige de er å løse
- Vi har følgende to klasser som utgjør avgjørelsesproblemer: "P" og "NP"
- P: Løsninger kan bli effektivt beregnet. Enkle problemer som er lette å beregne
- NP: Løsninger kan bli effektivt verifisert. Vanskelige problemer som er lette å verifisere, men vanskelig å beregne.
- "Effektivt" vil angi polynomisk tid altså(O(n^k) hvor k > 0)

Kompleksitetsklassen P

- Problemer som kan løses i polynomisk tid. Dette betyr problemer der vi vet at det finnes en algoritme som er i O(n^k) for et tall (k > 0).

- alle problemer vi har sett algoritmer for er i P
  - sorteringsalgoritmene løser

Sort	
Instans:	To lister L_1 og L_2
Spørsmål:	Består L_2 av elementene fra L_1 i sortert rekkefølge?

  - minimale spenntrær:

MST-k	
Instans:	En graf G, et tall k
Spørsmål:	Finnes det et spenntre i G som koster mindre enn k?

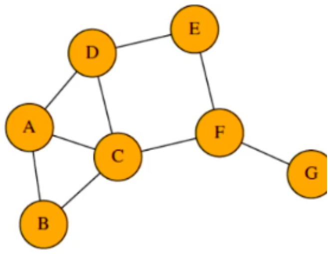
Løsning vs verifisering

Dette er for angi et spesifisere verifisering i henhold til avgjørelsesproblemer.

Hva betyr å "verifisere" avgjørelsesproblemer?

- En algoritme som verifiserer et problem, tar instansen og et sertifikat som input. Output blir JA hvis instansen er en JA-instans

Reachability	
Instans:	En graf G og to noder v, w
Spørsmål:	Finnes det en sti fra v til w i G?



- Instans: Grafen som er vist til høyre og erstatter "v" og "w" med "A" og "G".
- Mulig sertifikat: En sti fra "A" til "G": A,C,F,G
- Verifiseringsalgoritme: Sjekk om sertifikatet er en sti fra "A" til "G" i grafen.
- Det å "verifisere" avgjørelsesproblemer, dreier seg om å ha en sertifikat for å verifisere om instansen gir Output avhengig av spørsmålet som er blitt vist. F.eks så kunne "Sorted" ha sertifikat [1,2,3,4,5] og vi hadde hatt instans [1,2,3,4,5] også sjekker vi om instansen tilsvarer sertifiktatet, hvor vi i dette tilfelle vil få en JA-instans.

Kompleksitetsklassen NP

- Problemer som kan verifiserer i polynomisk tid
- Ekvivalent definisjon: Problemer som kan løses av ikke-deterministiske algoritmer i polynomisk tid
- Så et problem "L" er i "NP" hvis det finnes en algoritme "V" som tar som en instans av "L" og et sertifikat som input og outputter JA for JA-instanser innen polynomisk tid altså (O(n^k) for et tall k >0).

Kompleksitetsklassen NP: Eksempler

- Reachability er i NP, siden verifikasjonen (sjekk om sertifikatet er en sti fra v til w i G) kan gjøres i polynomisk tid

Algorithm 1: Reachability Verifier

Input: En graf G, to noder v, w, og en liste L av noder

Output: JA hvis L er en sti fra v til w i G

```
Procedure Reachability-Verifier(G, v, w, L)
    n = L.length
    if L[0] ≠ v or L[n − 1] ≠ w then
        return NEI
    for i from 1 to n − 1 do
        if (L[i − 1], L[i]) not edge in G then
            return NEI
    return JA
```

Reachability	
Instans:	En graf G og to noder v, w
Spørsmål:	Finnes det en sti fra v til w i G?

P er i NP

- Hvis vi kan løse i polynomisk tid, så kan vi også verifisere i polynomisk tid! Ved kompleksitetsklassen P, så husker vi at vi har problemer som er lette å beregne i polynomisk tid. Ved NP derimot så trengte vi en sertifikat og en instans for å verifisere at instansen er JA- eller NEI. Men hvis vi kan løse problemer via P, så kan vi verifisere problemene i NP.
- P er altså en delmengde av NP.
- Verifiseringsalgoritmen kan da bare løse problemet, og trenger ingen sertifikat. Dermed så kan vi bare bruke P-algoritmen som en verifiseringsalgoritme fremfor et sertifikat som var nødvendig i NP.

Algorithm 2: Verifying by solving

Input: En graf G, to noder v, w, og en liste L av noder

Output: JA hvis L er en sti fra v til w i G

```
Procedure Reachability-Verifier2(G, v, w, L)
    BFS(G,v)
    return w.visited
```

- Gjennom det som er blitt nevnt over, kan det finnes problemer i NP som ikke er i P. Det vet vi ikke, altså om det finnes problemer i NP som ikke kan løses i polynomisk tid som finnes i P.

"Vanskelige" problemer i NP

- NP-komplette problemene vil ansees som problemer som er de vanskeligste problemene i NP
- Eksempler på diverse vanskelige oppgaver under:

Hamiltonsykel	
Instans:	En graf G
Spørsmål:	Finnes det en sykel i G som besøker alle noder nøyaktig en gang?

Knapsack	
Instans:	En mengde objekter med hver sin vekt og verdi, og to tall s og t
Spørsmål:	Finnes det en mengde objekter som som tilsammen er verdt mer enn t og veier mindre enn s?

Sudoku	
Instans:	Et ufullstendig fylt ut $n \times n$ Sudoku brett
Spørsmål:	Har inputbrettet en gyldig løsning?