

Øvingsforelesning 9: Dynamisk programmering

Benjamin Bjørnseth

Gruppeøvinger

- Onsdag 14:15 - 15:45, F3 / R8
- Torsdag 16:15 - 17:45, F1
- Gamle eksamensoppgaver

Dagens forelesning

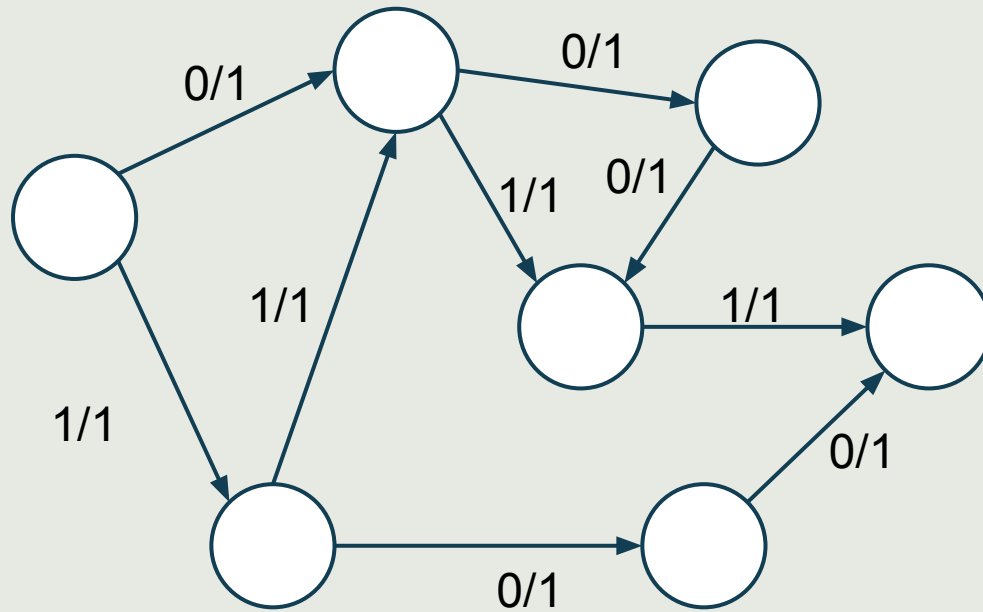
- Neste ukes øving
- Dynamisk programmering
- Forrige øving

Øving 9 - Teori

- Maks flyt
- Oppgave 1 og 2: Generelt
 - Bevaring av flyt
 - Residualnettverk
 - Flytforøkende sti
 - Max-flow min-cut

Oppgave 3 - Oppdatering av maks flyt

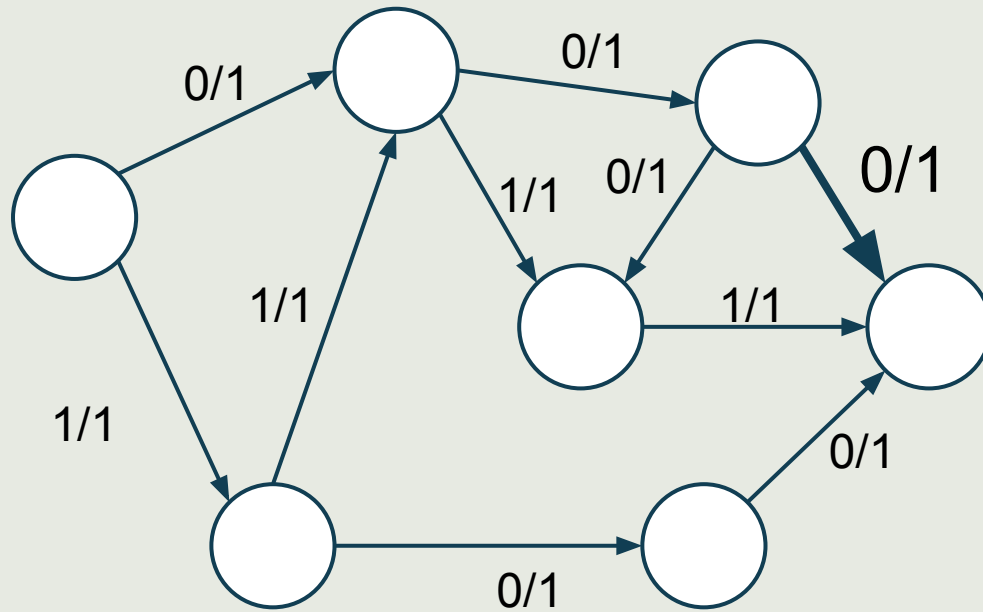
- Alle kanter har kapasitet 1
- $A(s, t)$ finner flytforøkende sti



$$|f| = 15$$

Oppgave 3 - Oppdatering av maks flyt

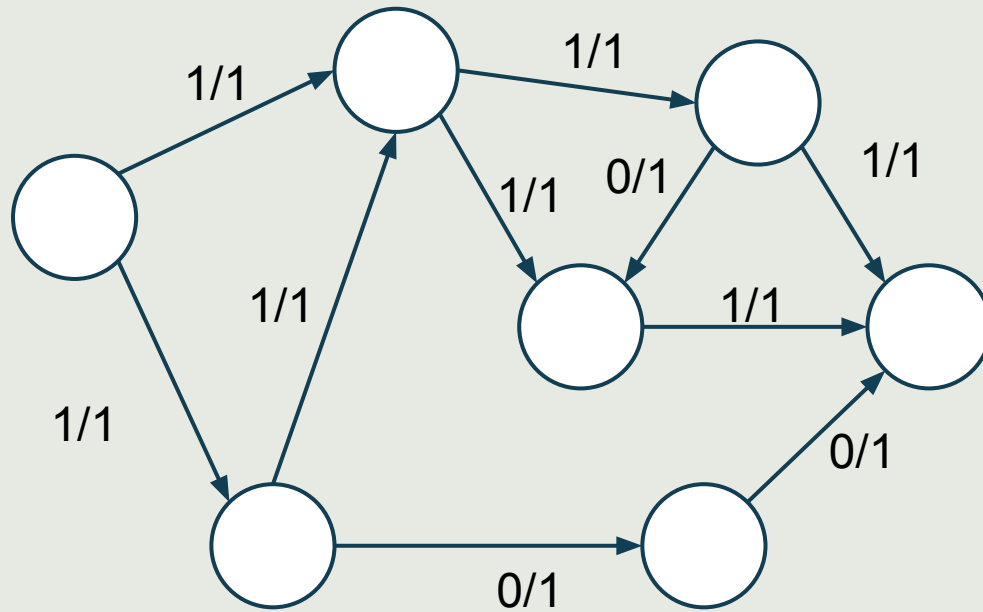
Hvordan bruke A til å finne ny maks flyt?



$$|f| = ?$$

Oppgave 3 - Oppdatering av maks flyt

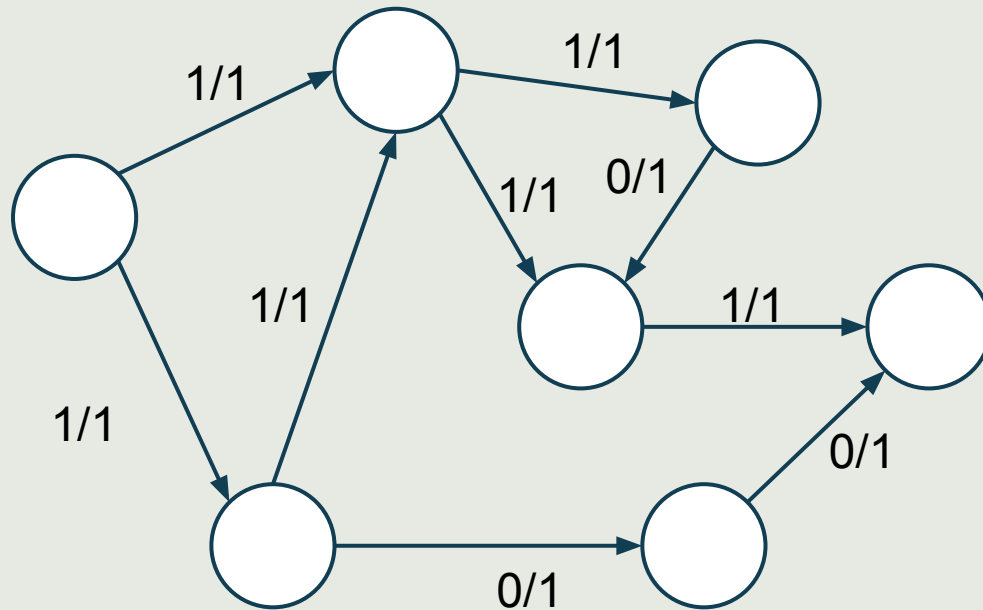
- Alle kanter har kapasitet 1
- $A(s, t)$ finner flytforøkende sti



$$|f| = 16$$

Oppgave 3 - Oppdatering av maks flyt

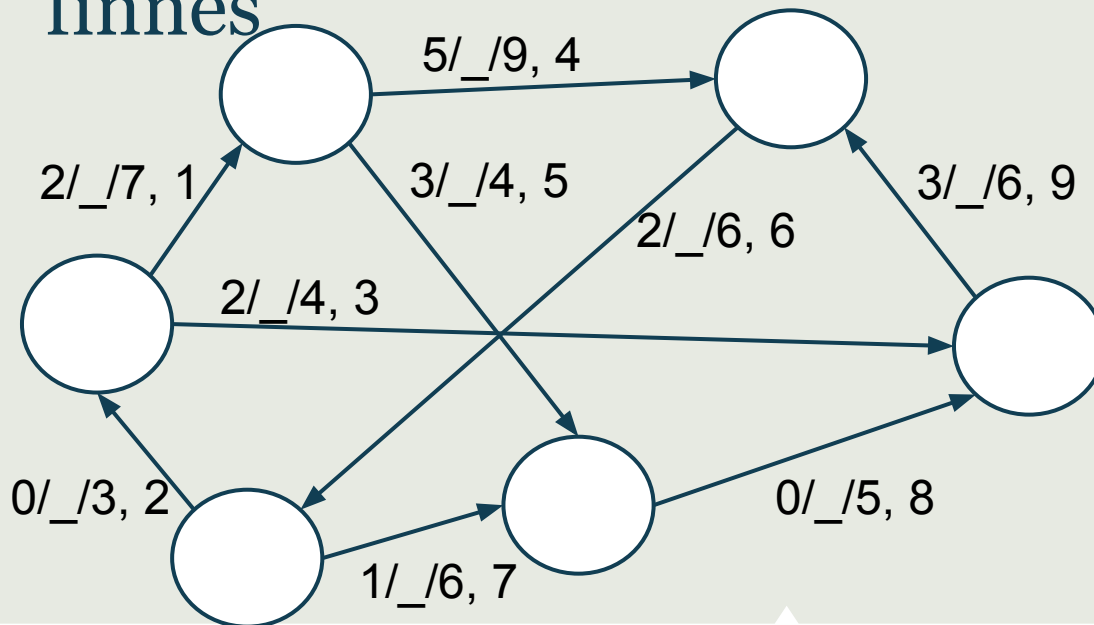
- Hvordan bruke A til å finne ny maks flyt?



$|f| = ?$

Oppgave 4 - Sirkulasjon

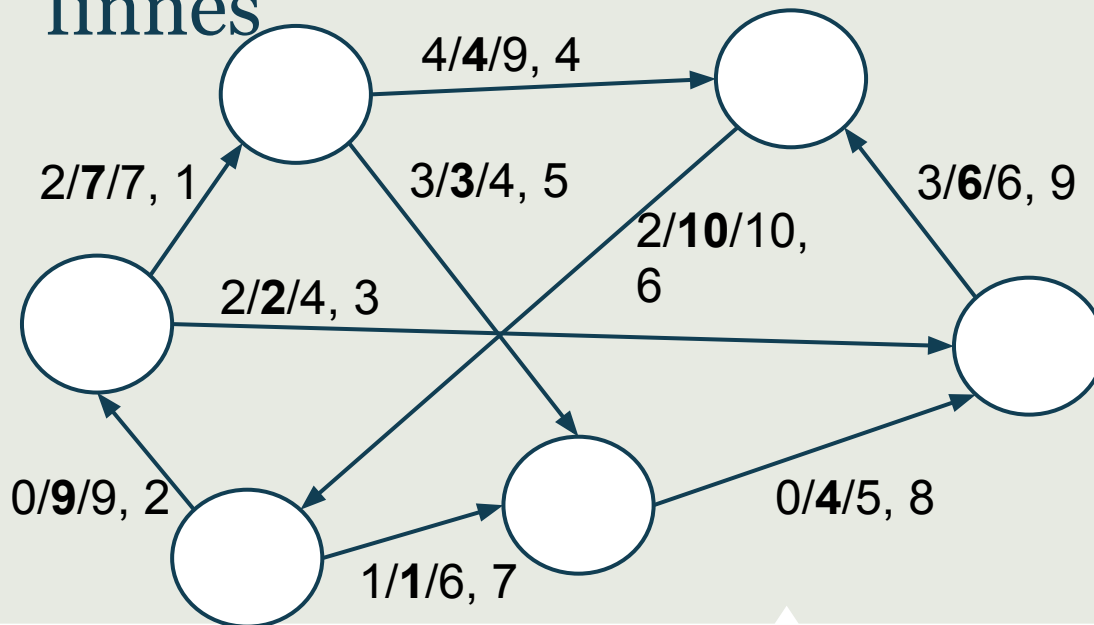
- Modifisert flytproblem:
 - Nedre grense på kantflyt
 - Flytbevaring i alle noder
- $A(G)$ finner min. kost sirkulasjon om det finnes



Notasjon:
<min>/<flow>/<max>,
<cost per flow unit>

Oppgave 4 - Sirkulasjon

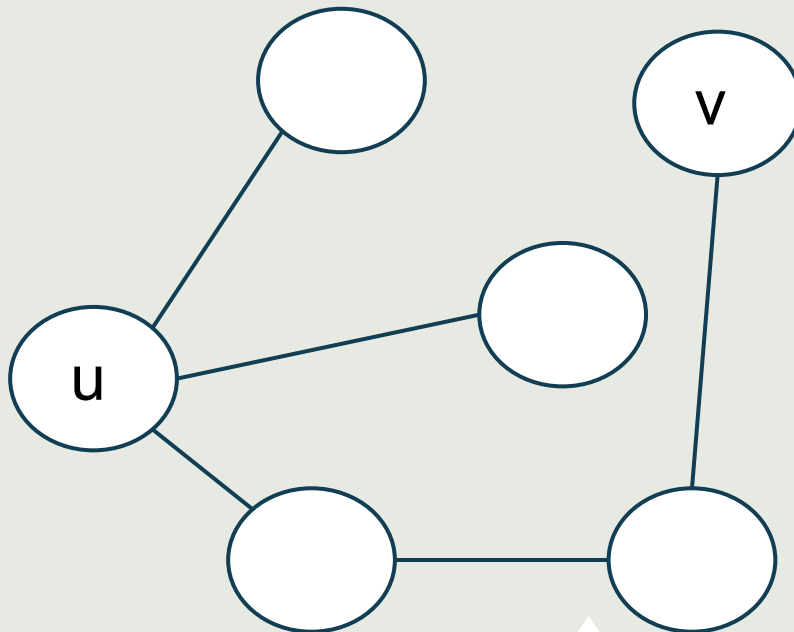
- Modifisert flytproblem:
 - Nedre grense på kantflyt
 - Flytbevaring i alle noder
- $A(G)$ finner min. kost sirkulasjon om det finnes



Sirkulasjon finnes,
Minimum kost: 215

Oppgave 4 - Sirkulasjon

- Har urettet graf $G' = (V', E')$
- Skal avgjøre om det finnes vei mellom to noder.
- Hvordan bruke sirkulasjonalgoritmen A ?



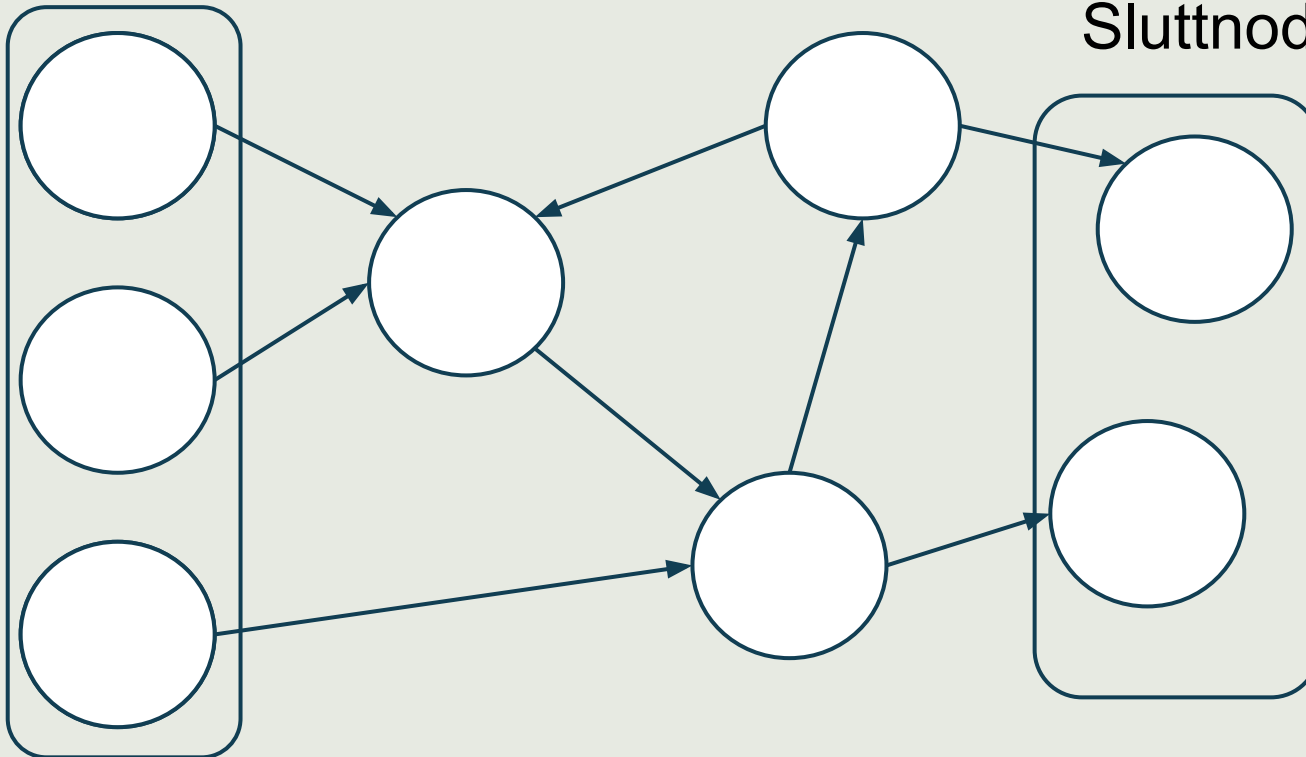
Øving 9 - Praksis

- Skumlehulen
- Problem: Gitt en graf, hvor mange ikke-overlapende stier finnes fra en mengde startnoder til en mengde sluttnoder?

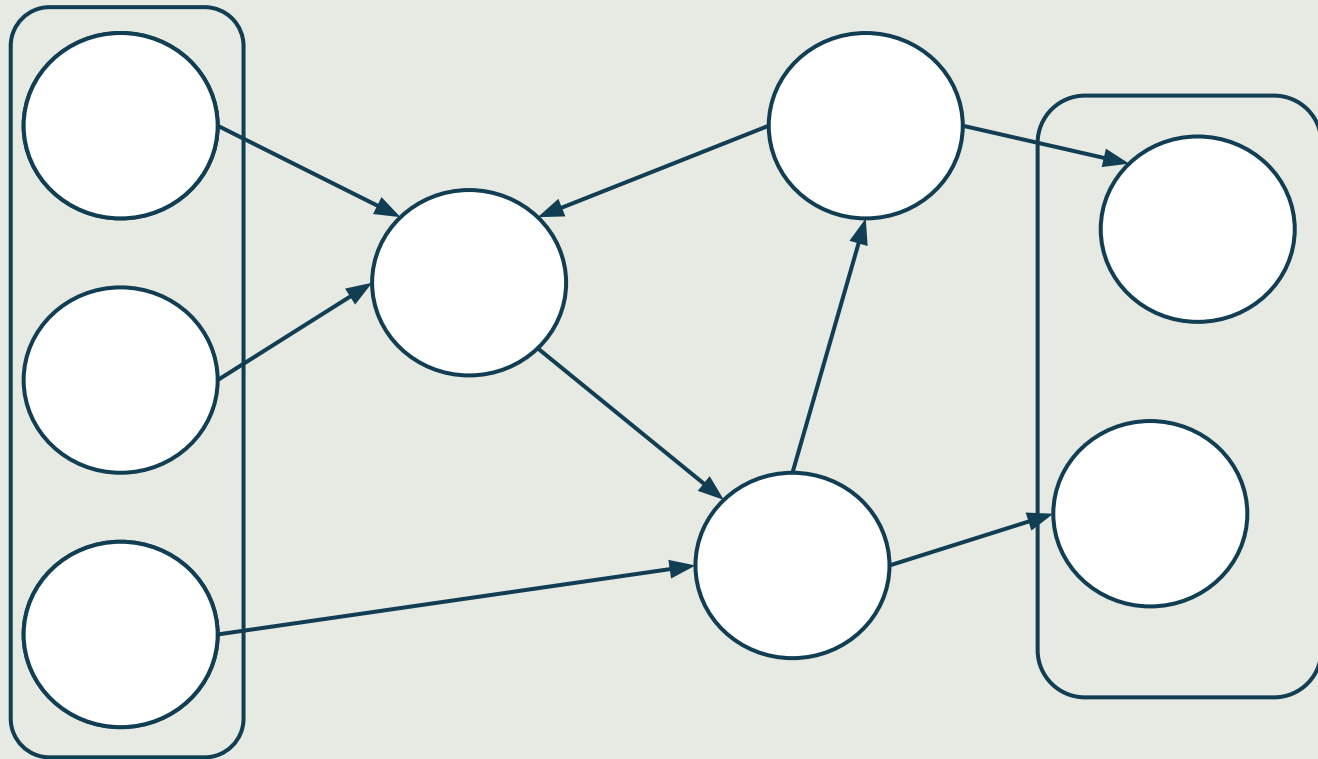
Øving 9 - Praksis

Startnoder

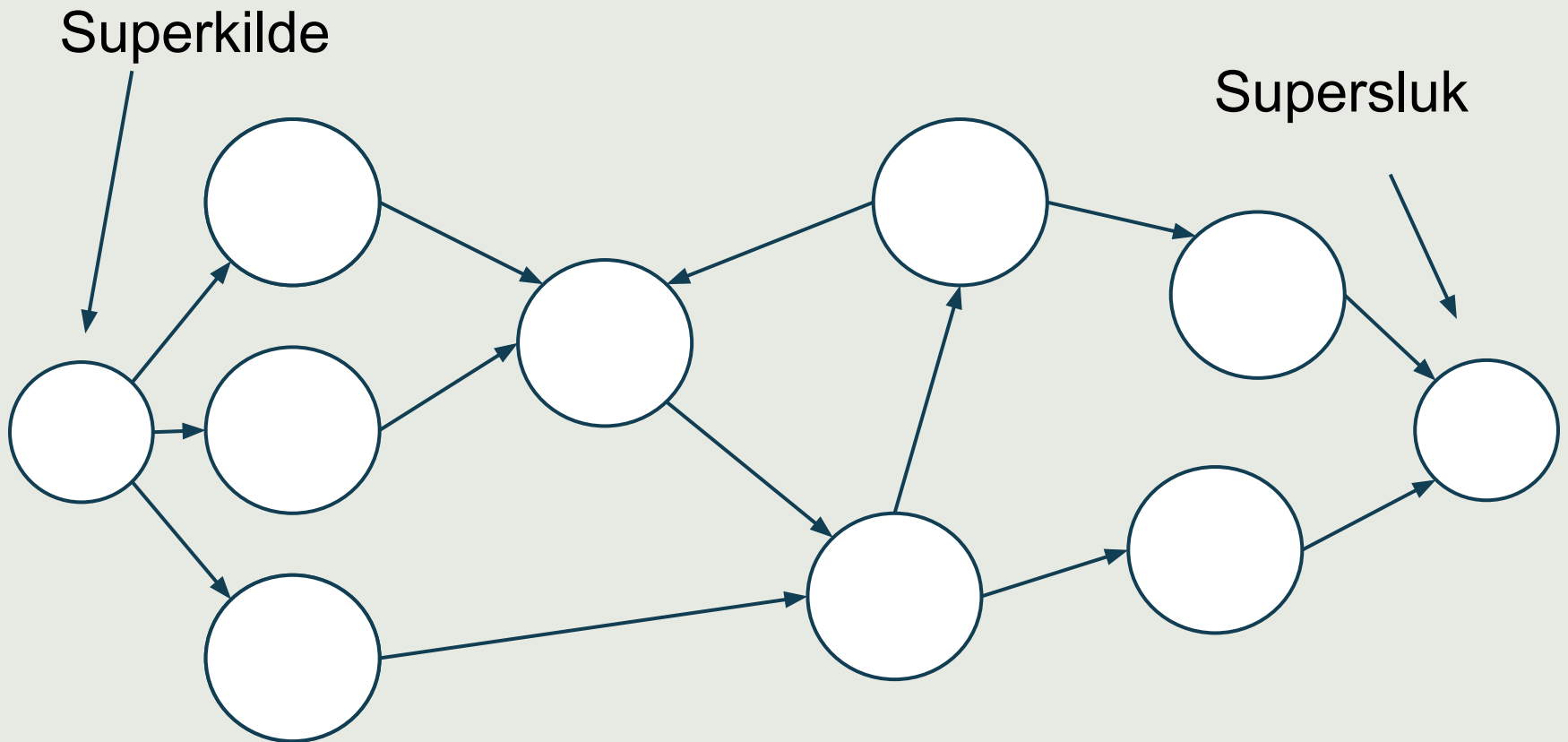
Sluttnoder



Øving 9 - Praksis

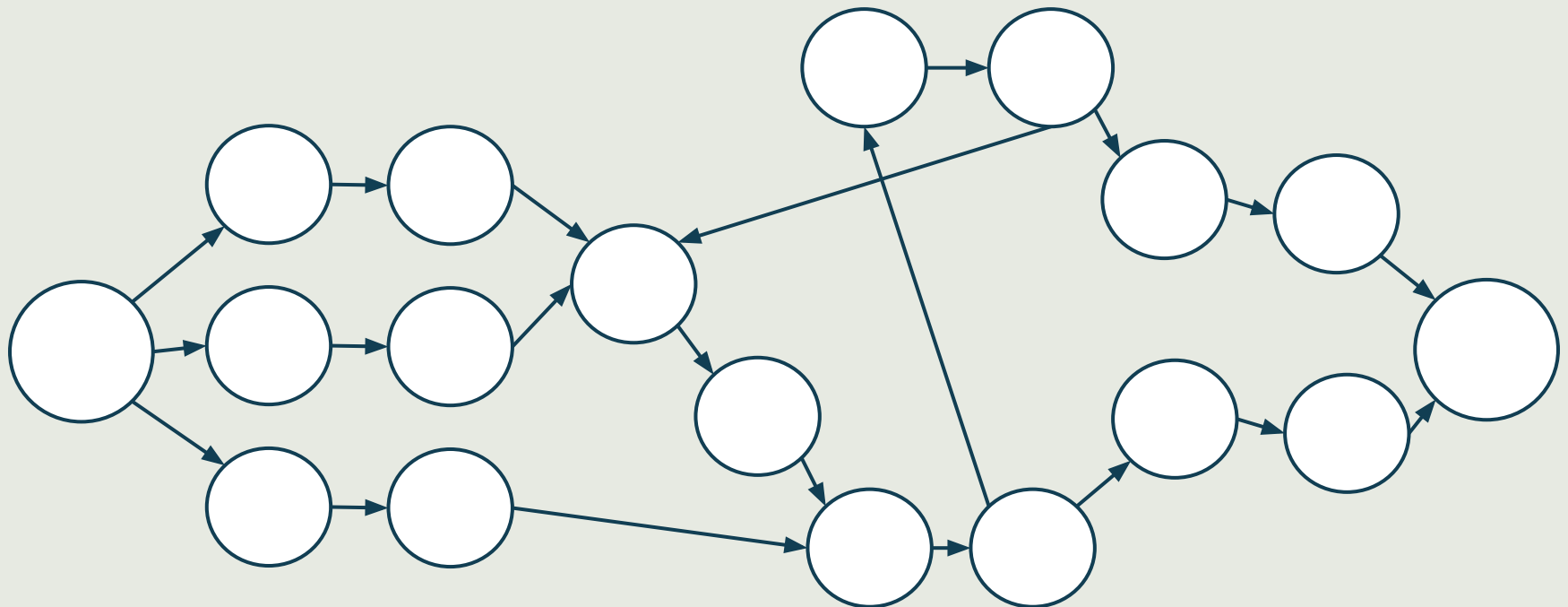


Øving 9 - Praksis



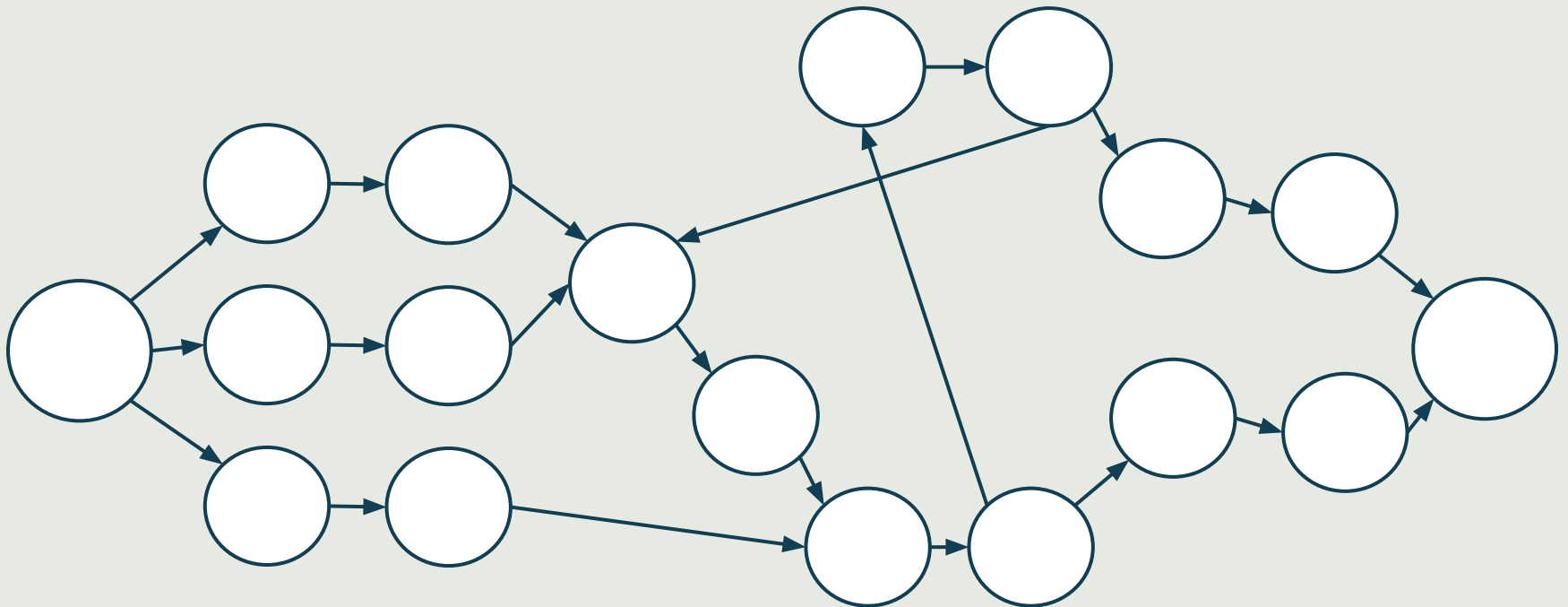
Øving 9 - Praksis

Splitt nodene



Øving 9 - Praksis

Sett kapasitet til 1



Øving 3 - Praksis

- Rammeverket bidrar med:
 - Innlesing av graf, startnoder og sluttnoder
 - Algoritme for å finne flytforøkende sti
- Dere må lage
 - Algoritme for å utvide grafen
 - Algoritme som finner maks flyt

Dynamisk programmering

Program

- Hva er dynamisk programmering?
- Når kan vi bruke DP?
- Hvordan bruke DP?
- Når kan vi ikke bruke DP?
- Om bevis

Dynamisk programmering

- Python, Ruby, Javascript,... ? (nei)
- Generell problemløsningsstrategi
 - Dynamisk: Løsning varierer med beslutning
 - Programmering: Skjematisk planlegging
- Brukes for å løse optimaliseringsproblemer
- To tegn på anvendbarhet
 - Optimal substruktur
 - Overlappende og uavhengige delproblemer

Optimal substruktur

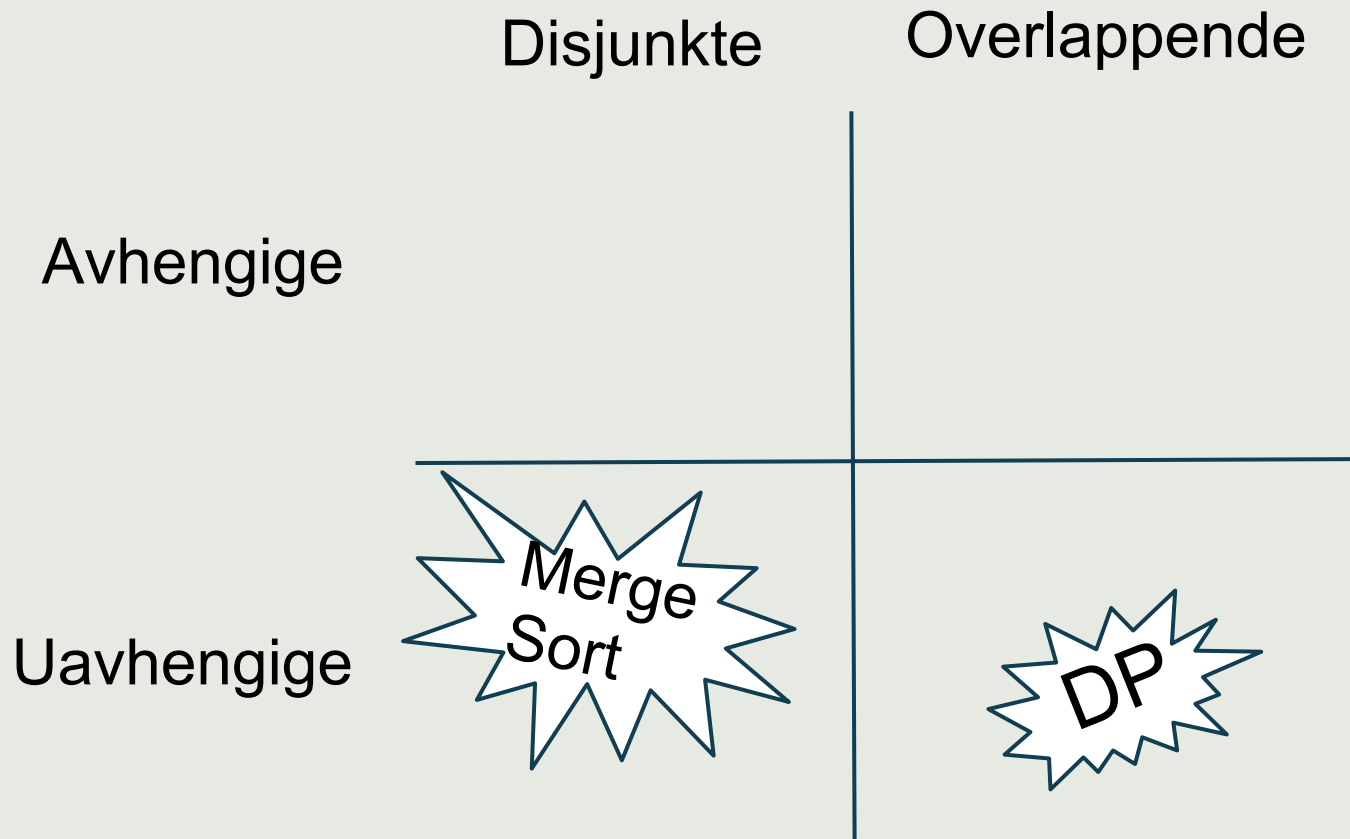
- Optimal løsning består av optimal løsning av mindre delproblem
- Dette gjør at vi kan bruke rekursjon

Overlappende delproblemer

- I DP: Tar vare på midlertidige løsninger
- Kun nyttig hvis delproblemene overlapper

Overlappende og uavhengig?

- To punkter på forskjellige akser

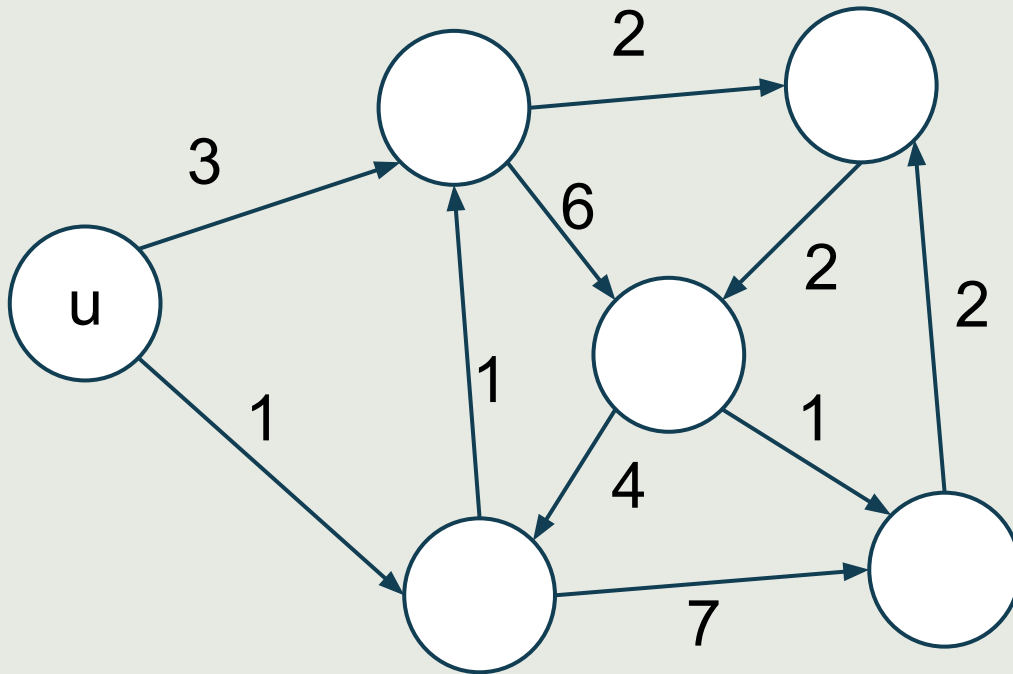


Virker kjent?

- Har sett eksempler tidligere på DP
 - Bellman(!)-Ford
 - Floyd-Warshall

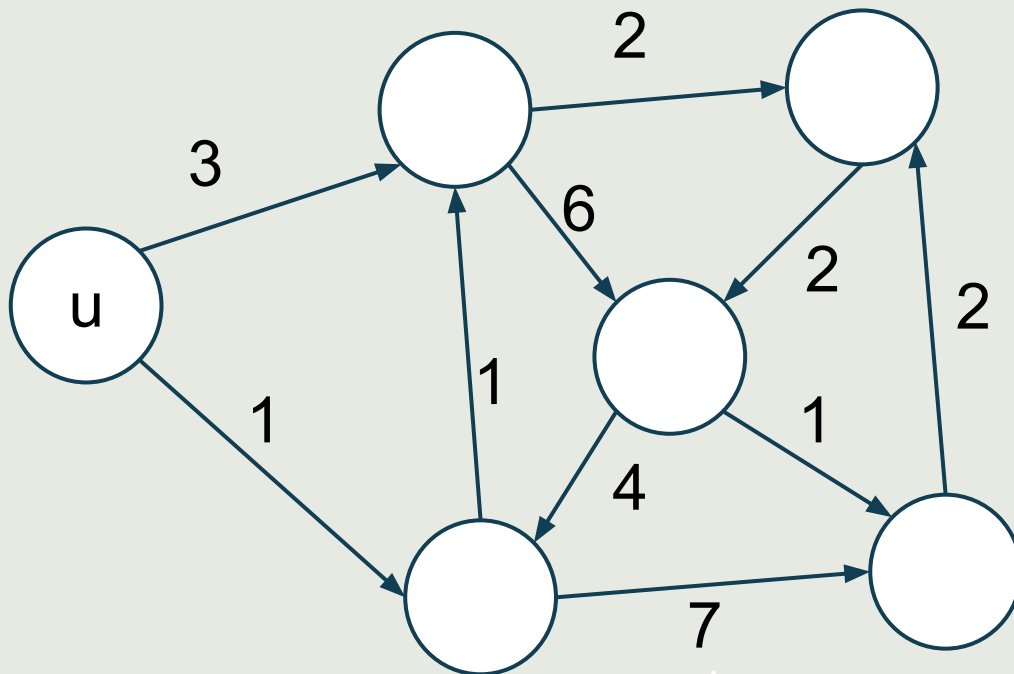
Bellman-Ford

- Korteste vei i graf fra u til resten.



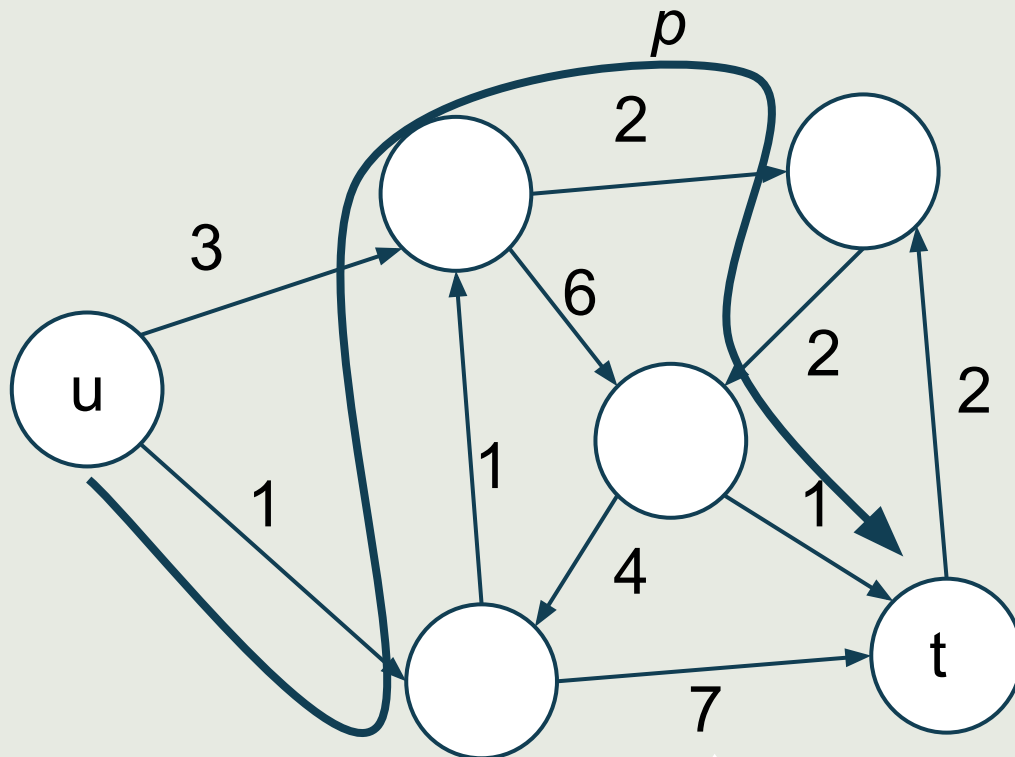
Bellman-Ford

- Optimal substruktur



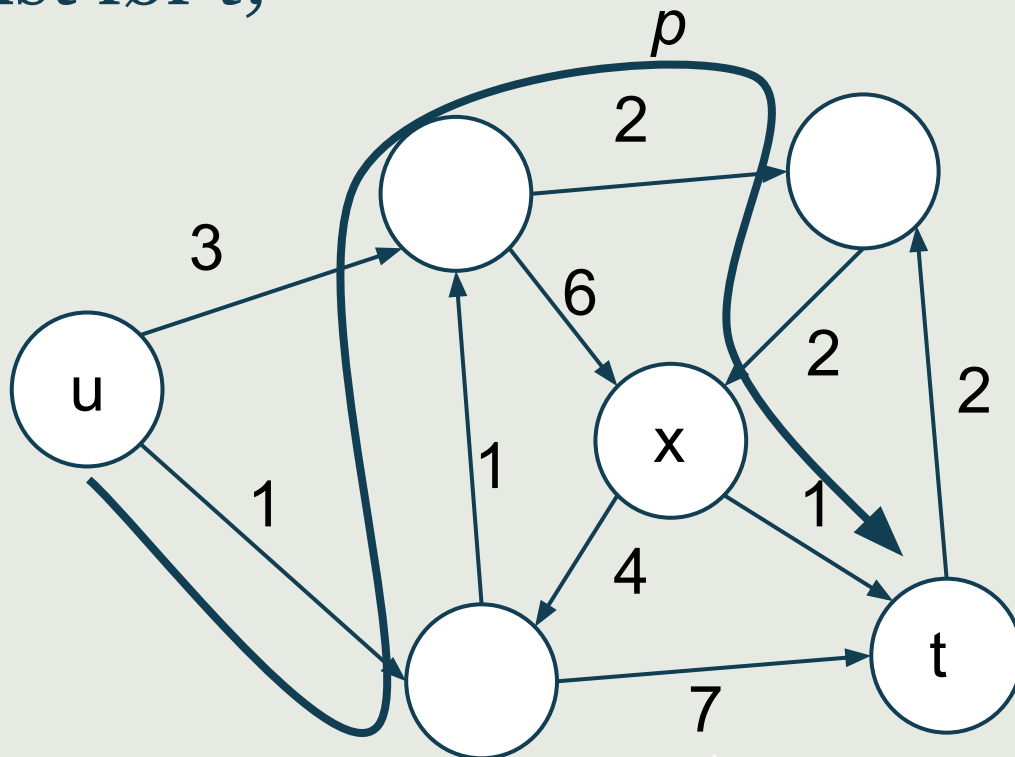
Bellman-Ford

- Optimal substruktur
Hvis korteste vei til t er p ,



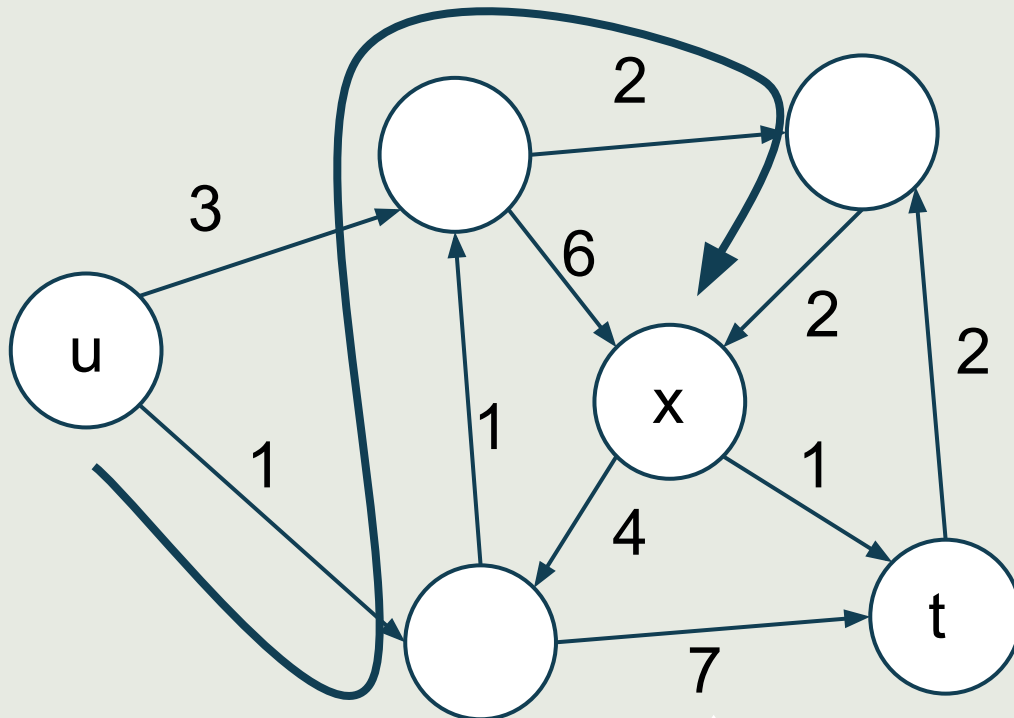
Bellman-Ford

- Optimal substruktur
Hvis korteste vei til t er p , og p går innom x sist før t ,



Bellman-Ford

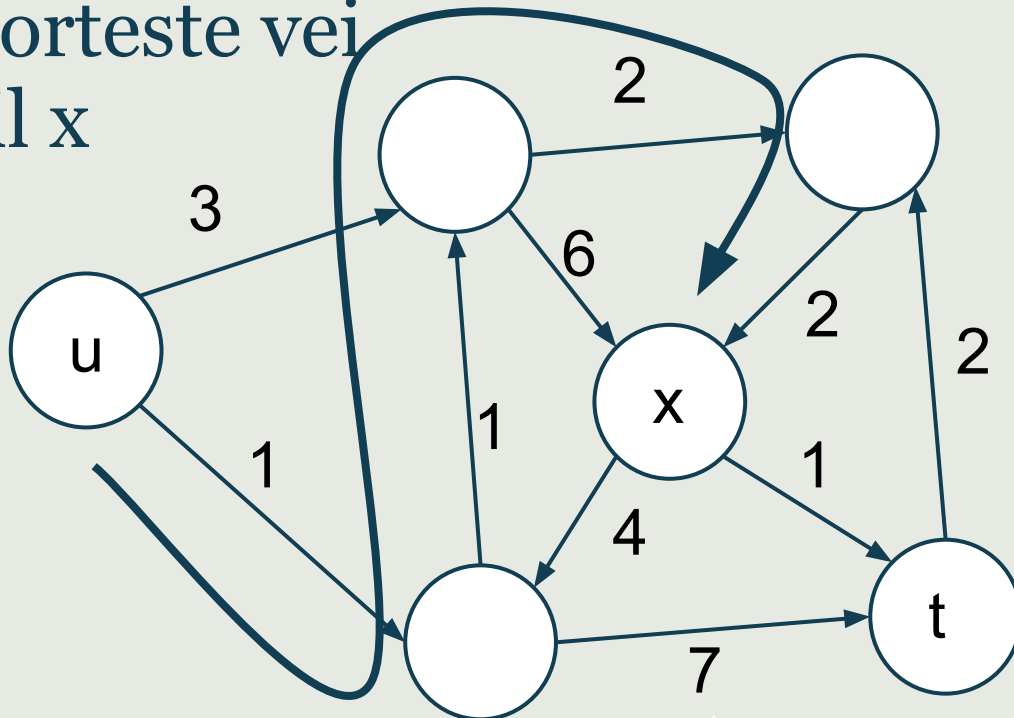
- Optimal substruktur
Hvis korteste vei til t er p , og p går innom x sist før t , så må delen av p frem til x



Bellman-Ford

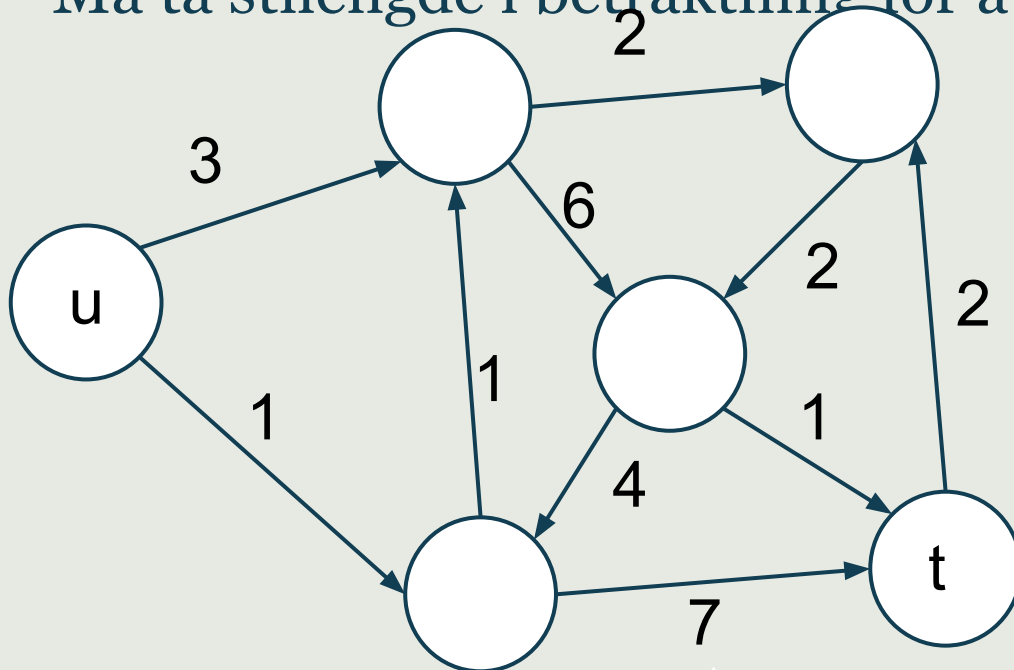
- Optimal substruktur

Hvis korteste vei til t er p , og p går innom x sist før t , så må delen av p frem til x være korteste vei til x



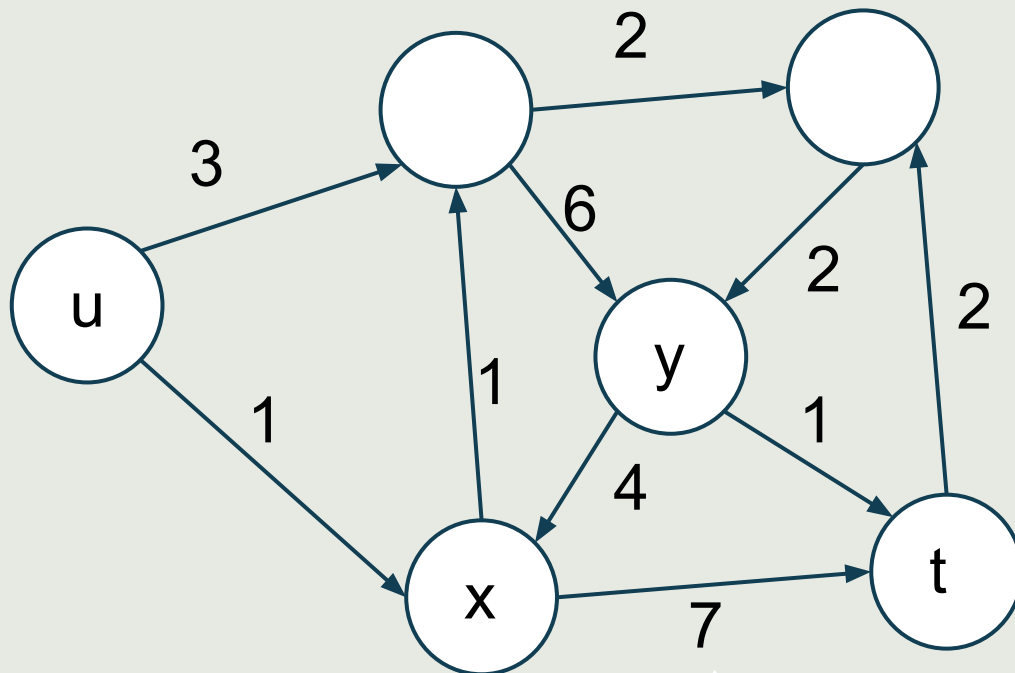
Bellman-Ford

- Optimal substruktur
 - Rekursiv løsning: sjekk min. kost for alle noder m/ kant inn
 - Må ta stilengde i betraktning for å håndtere sykler



Bellman-Ford

- Overlappende delproblemer
 - Korteste vei til en node sjekkes flere ganger.



Floyd-Warshall

- Korteste veier mellom alle par av noder i en graf
- Optimal substruktur
 - En korteste vei mellom node i og j gjennom kun noder $\{1, \dots, k\}$ består av korteste veier innom noder $\{1, \dots, k-1\}$
- Overlappende delproblemer
 - Korteste veier mellom forskjellige par (i, j) kan være interessert i samme korteste veier innom noder $\{1, \dots, k-1\}$

Generell fremgangsmåte

1. Beskriv optimal substruktur
2. Formuler rekursiv løsning
3. Beregn verdien av optimal løsning
4. Hvis ønskelig, finn optimal løsning

Eksempel: Maksimal strengdelingssum

- Har en vilkårlig lang streng av heltall
 - 1235982349780659892003482349859892381
- Ønsker å dele opp i segmenter
 - Segmentverdi må passe i 32-bits signed integer
 - Del slik at summen blir høyest mulig
- Eksempel
 - 1234554321 => 1234554321
 - 5432112345 => 543211234 + 5 = 543211239
 - 1212121212 => 1 + 2121212121 + 2 = 2121212124

Bruke DP?

- Skal dele opp 12309304239402384023492
- Optimal substruktur?
- Overlappende delproblemer?

Verktøykasse

- Definer terminologi for formell håndtering
- s = strengen vi skal dele opp
- n = lengden av s
- $s[:i]$ = delen av strengen opp til indeks i
- $m[i]$ = maksimal strengdelingssum for $s[:i]$
- $v(s)$ = tallverdien av s
- $f(s)$ = funksjon som returnerer $v(s)$ hvis s har en tillatt verdi, og $-\infty$ ellers

Steg 1 - Optimal substruktur

- Dersom $v(s) \leq 2^{31} - 1$, er ingen oppdeling gunstigst
 - $x + y < 10^{(\text{floor}(\log(y)) + 1)} * x + y$
- Hvis ikke, må vi velge et punkt å dele på

Steg 1 - Optimal substruktur

- Anta optimal oppdeling er i punkt j
Optimal løsning for s består av
optimal løsning for $s[:j]$
 - $m[n] = m[j] + f(j, n)$
- Bevises ved å anta det motsatte

Steg 1 - Optimal substruktur

- Hvordan finne j ?
- $n - 10 \leq j \leq n - 1$
- Må finne $m[n-10], m[n-9], \dots, m[n-1]$

Overlappende delproblemer

- For å finne $m[n]$, trenger vi $m[n-1], \dots, m[n-10]$
- For å finne $m[n-1]$, trenger vi $m[n-2], \dots, m[n-11]$
- \Rightarrow Overlappende delproblemer!
 - DP har noe for seg

Steg 2 - Rekursiv løsning

$$m[i] = \begin{cases} f(s[:i]) & ; f(s[:i]) \neq -\infty \\ \max_{1 \leq k \leq 10} \{ f(s[i-k : i]) + m[i - k] \} & ; \text{ellers} \end{cases}$$

Steg 3 - Beregn optimal verdi

- $m[i]$ avhenger av $m[j < i]$
- For $i < 10$, vet vi at base case gjelder
- Beregn fra bunnen og opp

Steg 3 - Beregn optimal verdi

- Hjelpesfunksjon:

```
MAX = 2**31 - 1
```

```
NEGINF = -1e40
```

```
def f(s):
```

```
    t = int(s)
```

```
    return t if t <= MAX else NEGINF
```

Steg 3 - Beregn optimal verdi

```
def string_split_sum(s):  
    n = len(s)  
    m = [0] * (n+1)
```

Steg 3 - Beregn optimal verdi

```
def string_split_sum(s):  
    n = len(s)  
    m = [0] * (n+1)  
    #base case  
    for i in range(10):  
        m[i] = int(s[:i])
```

Steg 3 - Beregn optimal verdi

```
def string_split_sum(s):  
    # ...  
    for i in range(10, n+1):  
        m[i] = -1  
        for k in range(1, 11):  
            # Beregn maksimal m[i]-verdi,  
            # basert på rekursjonen
```

Steg 3 - Beregn optimal verdi

```
def string_split_sum(s):  
    # ...  
    for i in range(10, n+1):  
        m[i] = -1  
        for k in range(1, 11):  
            t = f(s[i-k:i]) + m[i-k]  
            m[i] = m[i] if m[i] > t else t  
    print m[n-1]
```

Steg 4 - Finn optimal løsning

- Ta vare på optimalt kutt ved hver delstring

Steg 4 - Finn optimal løsning

```
def stringsplitsum(s):  
    splits = [0] * (n+1)  
    # ...  
    # for i ...  
        best_k = 0  
        # for k ...  
            # if must_update:  
                best_k = k  
        splits[i] = i - k
```

Steg 4 - Finn optimal løsning

- Eksempelverdi:

121212121212 => $[0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 0, 1, 11]$

- For $s[:12]$, er det beste å dele opp i 

Steg 4 - Finn optimal løsning

- Eksempelverdi:

121212121212 => [0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 11]

indeg 0

- For $s[:12]$, er det beste å dele opp i $s[:11]$
- indeg 12
-

Steg 4 - Finn optimal løsning

- Eksempelverdi:

121212121212 => $[0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 0, 1, 11]$

- For $s[:12]$, er det beste å dele opp i $s[:11]$, $s[11:12]$
-

Steg 4 - Finn optimal løsning

- Hvordan bruke `splits`?

```
def print_optimal_splits(splits, n, s):  
    # ...
```

Steg 4 - Finn optimal løsning

- Hvordan bruke `splits`?

```
def print_optimal_splits(splits, n, s):  
    def rec_helper(j):  
        # ...  
    rec_helper(n-1)
```

Steg 4 - Finn optimal løsning

- Hvordan bruke `splits`?

```
def print_optimal_splits(splits, n, s):  
    def rec_helper(j):  
        if splits[j] > 0:  
            # ...  
        else:  
            print s[:j]  
    rec_helper(n-1)
```

Steg 4 - Finn optimal løsning

- Hvordan bruke `splits`?

```
def print_optimal_splits(splits, n, s):  
    def rec_helper(j):  
        if splits[j] > 0:  
            rec_helper(splits[j])  
            print '+', s[splits[j]:j]  
        else:  
            print s[:j]  
    rec_helper(n-1)
```

Steg 4 - Finn optimal løsning

121212121212

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]



Steg 4 - Finn optimal løsning

12121212121 2

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]



Steg 4 - Finn optimal løsning

12121212121 2

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]



Steg 4 - Finn optimal løsning

1 2121212121 2

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]



Steg 4 - Finn optimal løsning

1 2121212121 2

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]



Ferdig!

Steg 4 - Finn optimal løsning

1 + 2121212121 + 2 = 212121212124

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]

Steg 4 - Finn optimal løsning

1 + 2121212121 + 2 = 212121212124

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 11]

Optimal verdi

Utvis aktpågivenhet og varsomhet!

- Delproblemer må være uavhengige
 - Kan gjøre beregninger uten å ta hensyn til tidligere valg
- Eksempelvis strengdelingssum
 - Deler alltid strengen i to separate deler

Optimal oppdeling av venstre avhenger ikke av dette

- (Merk: total løsning avhenger av delproblem)

Eksempel: Optimale søketrær

- Eksempel i forelesning
- Rekursiv oppdeling:
 $k_{i\dots j}$ beregnes fra av $k_{i\dots r-1}$, $k_{r+1\dots j}$
- Hvert delproblem kan regnes ut uavhengig av det andre
- For hvert valg av r , får vi disjunkte problem
 - Men mange valg gir overlapp

Eksempel: Korteste sti

- Så tidligere på Bellman-Ford

Eksempel: Korteste sti

- Så tidligere på Bellman-Ford
- Dekomponerte en korteste sti i

$$u \overset{p}{\rightsquigarrow} t \implies u \overset{p'}{\rightsquigarrow} x \rightarrow t$$

- (Sti regnes her som en vei uten sykler)

Eksempel: Korteste sti

- Så tidligere på Bellman-Ford
- Dekomponerte en korteste sti i

$$u \overset{p}{\rightsquigarrow} t \implies u \overset{p'}{\rightsquigarrow} x \rightarrow t$$

- (Sti regnes her som en vei uten sykler)
- Gir dette uavhengige delproblemer?

Eksempel: Korteste sti

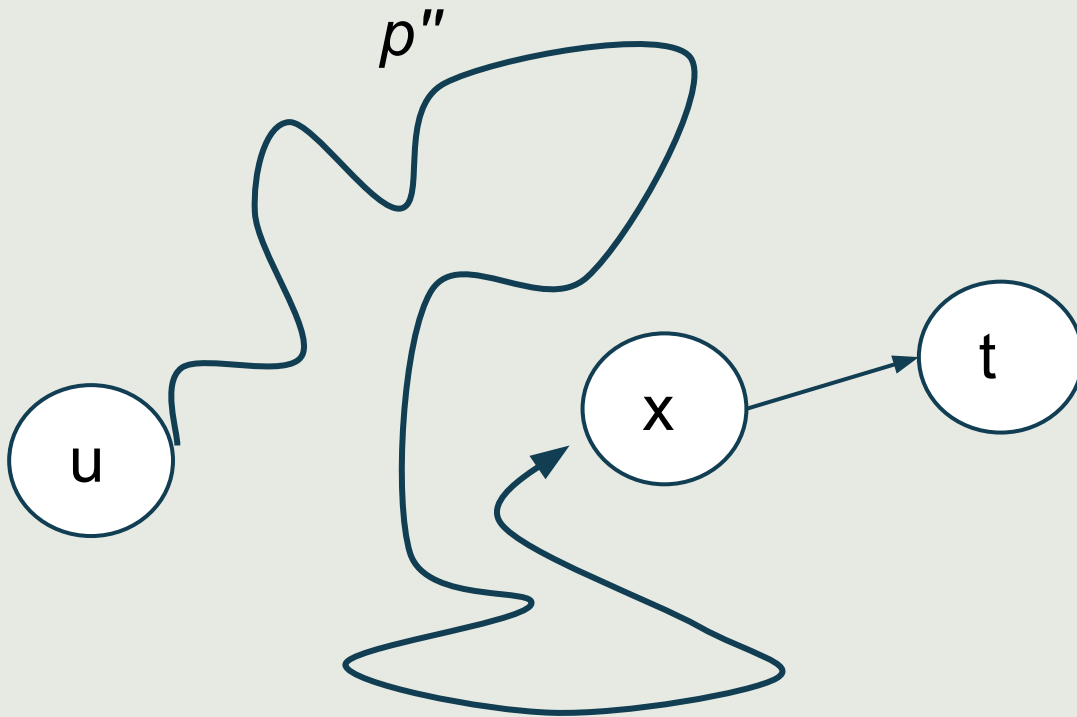
- Så tidligere på Bellman-Ford
- Dekomponerte en korteste sti i

$$u \overset{p}{\rightsquigarrow} t \implies u \overset{p'}{\rightsquigarrow} x \rightarrow t$$

- (Sti regnes her som en vei uten sykler)
- Gir dette uavhengige delproblemer?
- La oss se på korteste vei fra u til x , p''

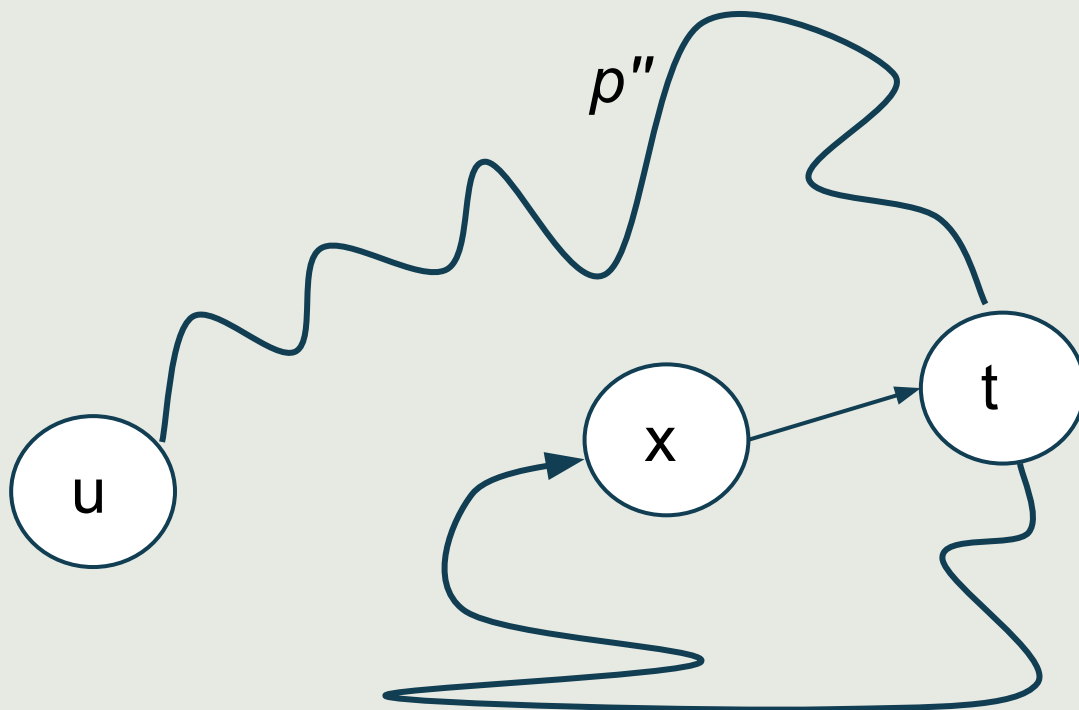
Eksempel: Korteste sti

- p'' vil ikke gå gjennom t



Eksempel: Korteste sti

- p'' vil ikke gå gjennom t
- Hvis den gjorde det...



Eksempel: Korteste sti

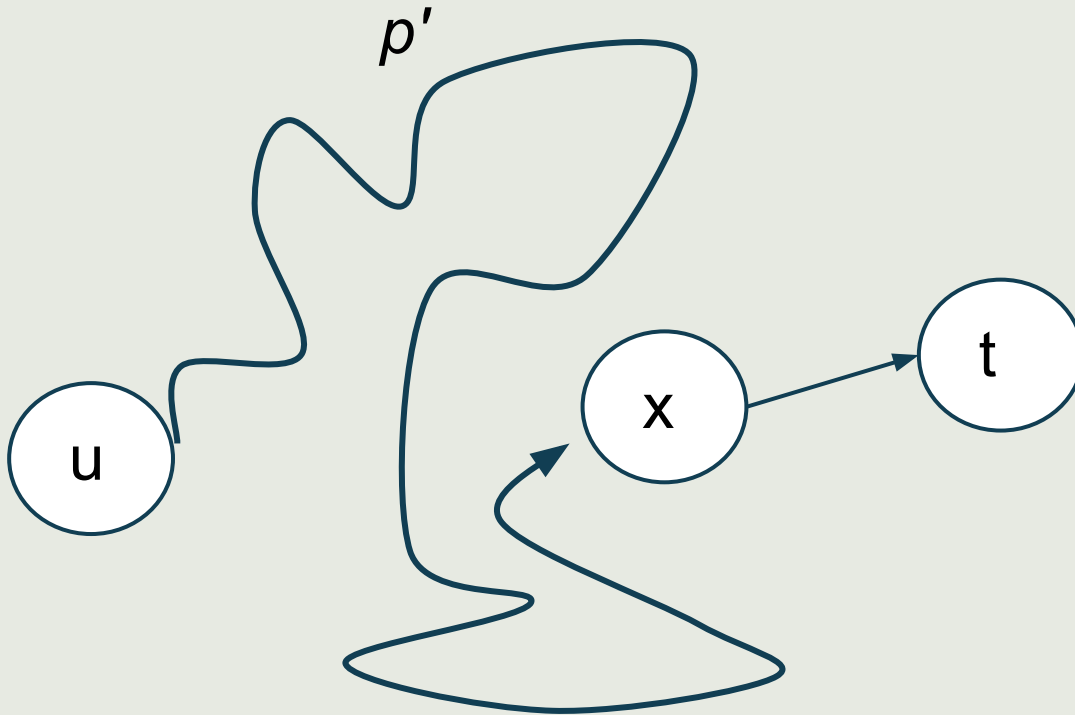
- p'' vil ikke gå gjennom t
- Hvis den gjorde det ville ikke p vært korteste sti til t likevel
- p'' bevarer sti-egenskapen hvis satt sammen med kanten fra x til t

Eksempel: Korteste sti

- p'' vil ikke gå gjennom t
- Hvis den gjorde det ville ikke p vært korteste sti til t likevel
- p'' bevarer sti-egenskapen hvis satt sammen med kanten fra x til t
- Dermed kan vi si at $p' = p''$
 - Hvis ikke hadde ikke p vært kortest

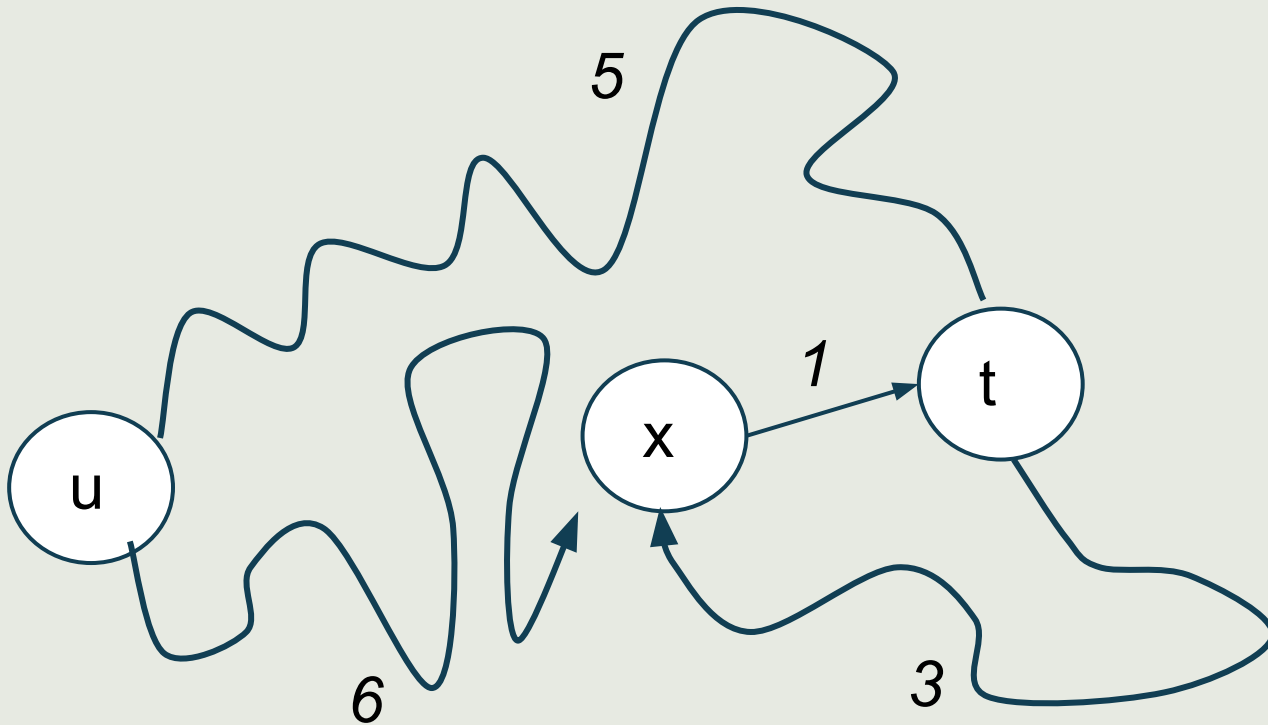
Eksempel: Lengste sti

- Hva med lengste sti?



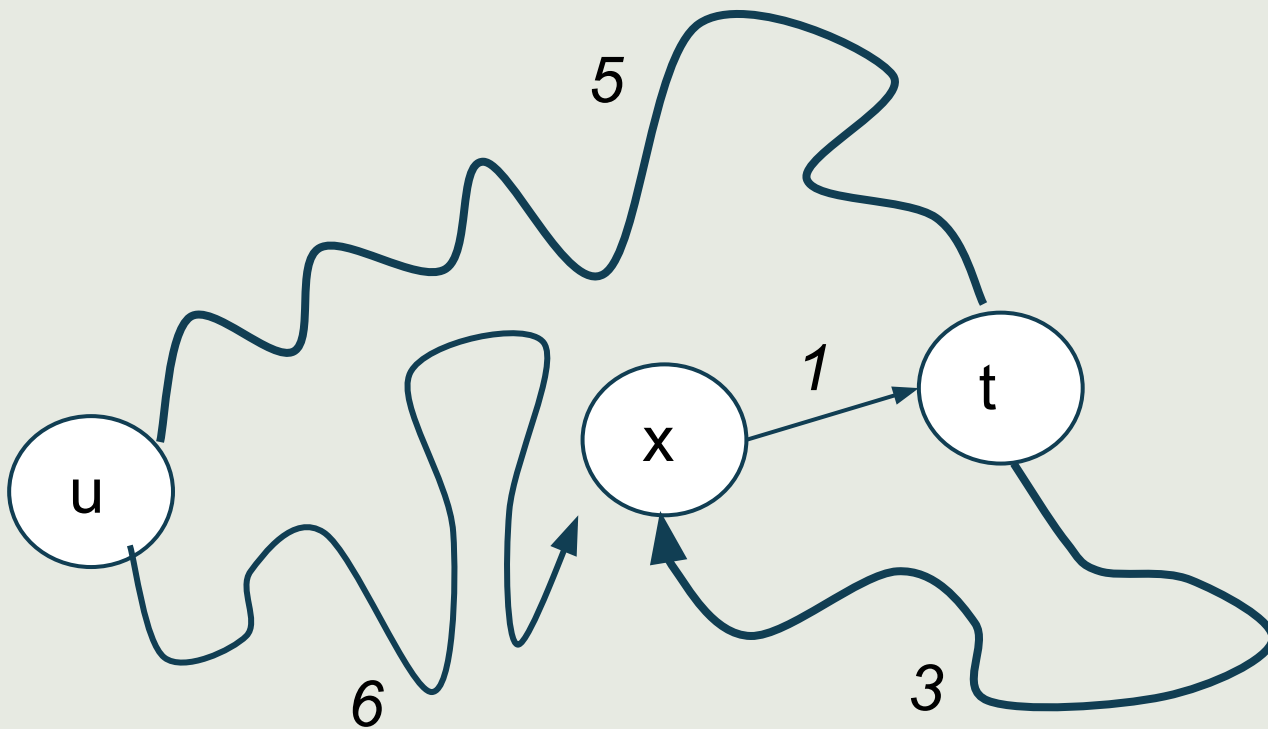
Eksempel: Lengste sti

- Hva med lengste sti?
- Ser igjen på p'' , lengste sti til x



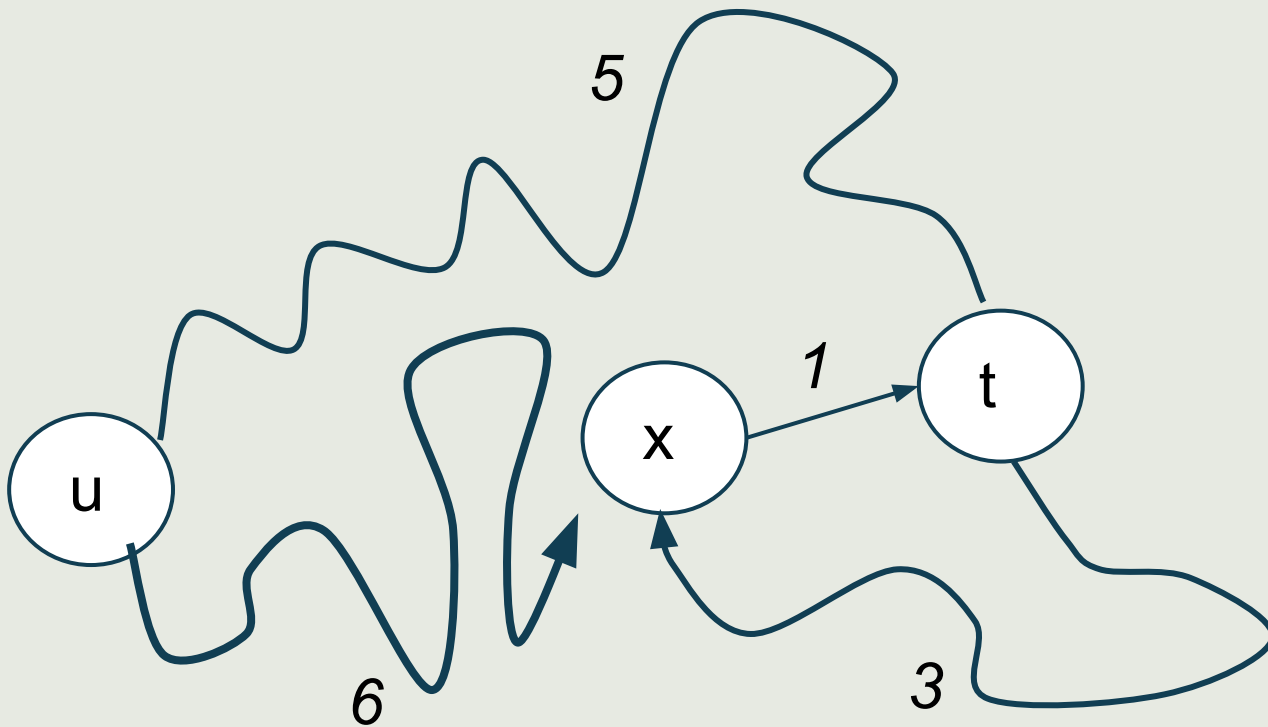
Eksempel: Lengste sti

- p'' kan godt gå gjennom t .



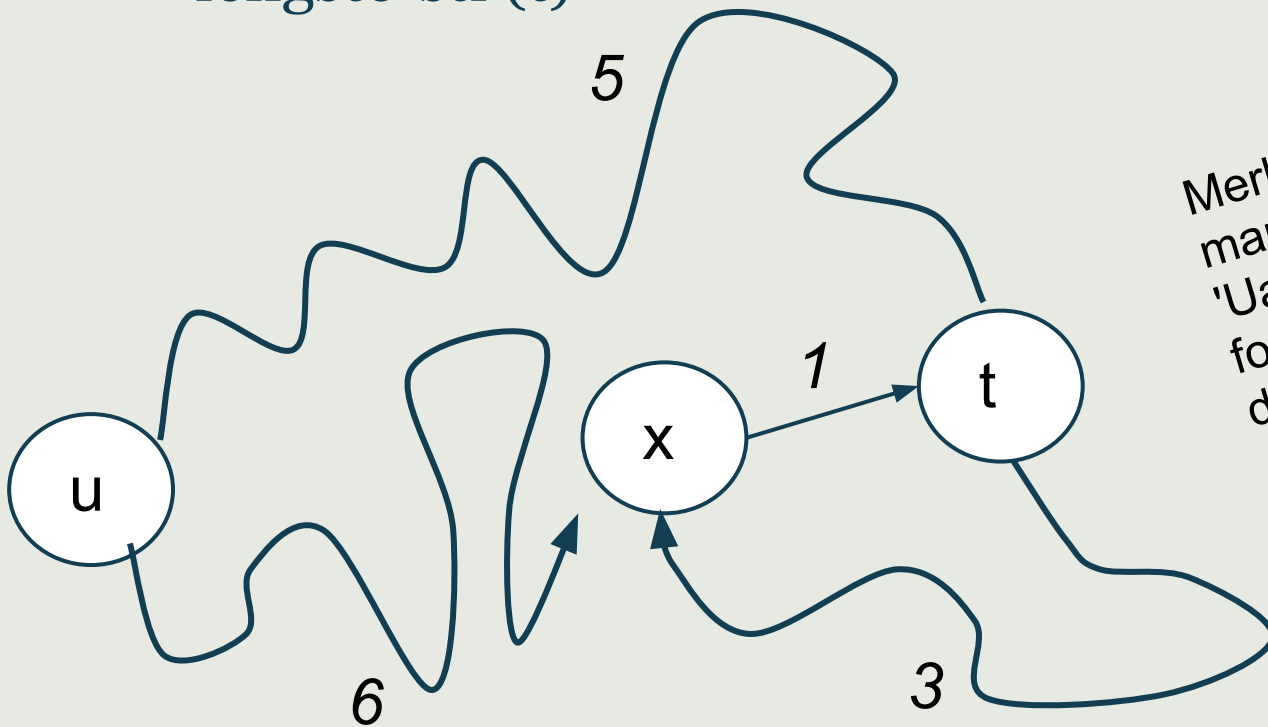
Eksempel: Længste sti

- p' er dermed ikke nødvendigvis lik p''



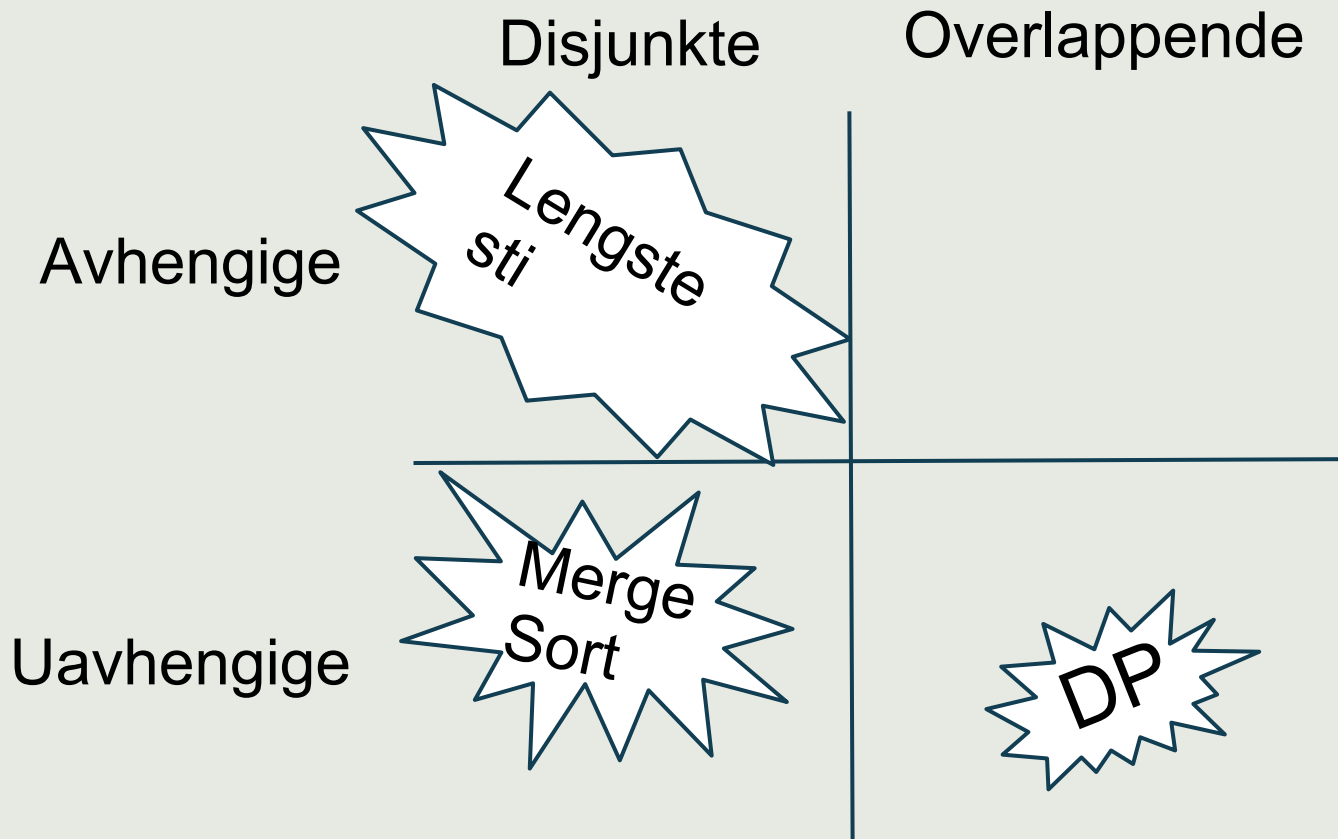
Eksempel: Lengste sti

- Vi har dermed ikke optimal substruktur
 - Delproblem lengste-sti (x) er avhengig av lengste-sti (t)



Merk: Oppdages hvis man er forsiktig. 'Uavhengighet' kjekt for å huske å være det!

Eksempel: Lengste sti



Angående bevis

- Steg 1: Finn optimal substruktur
- Må bevise at det vi beskriver stemmer
- Ofte brukt teknikk: Klipp-og-lim

Bevise optimal substruktur

1. Beskriv optimal løsning i å bestå av å ta et valg
2. Anta at valget som gir optimal løsning er gitt
3. Vis hvordan valget gir mindre delproblemer
4. Bevis at optimal løsning må bestå av optimale løsninger av delproblemene

Beskrivelse
av problemet

Her kommer
beviset inn!

Bevise optimal substruktur

- Bevis ved motsigelse
1. Anta at løsningen av delproblemene ikke er optimal
 2. "Klipp ut" denne løsningen
 3. "Lim inn" optimal løsning
 4. Vis at vi nå får en bedre løsning totalt sett enn vi hadde.

Motsigelse

Eksempel: Strengdelingssum

- Har tidligere sett dette for strengdelingssum
 1. For en streng med for stor verdi, må vi velge et sted å dele den opp.
 2. Antar at j er indeksen for delepunktet lengst til høyre som gir optimal verdi
 3. Løsningen består nå av tallverdi til strengen $s[j:]$, kalt t , pluss en oppdeling av streng $s[:j]$

Eksempel: Strengdelingssum

4.

- a. La oss anta at løsningen for $s[:j]$ gir verdi v , og at dette ikke er en optimal løsning.
- b. Optimal løsning for $s[:j]$ gir verdi v^*
- c. Løsningen for det totale problemet er
 $v + t < v^* + t$
- d. Løsningen for det totale problemet er ikke optimal
 \Rightarrow Vi må ha at løsningen for $s[:j]$ er optimal

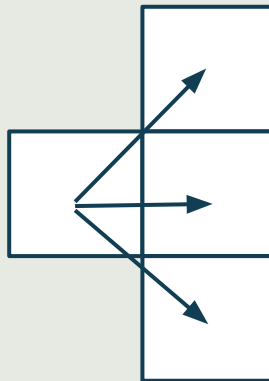
Oppsummering i bruk av DP

1. Beskriv optimal substruktur
 - a. Beskriv løsning i å bestå av å ta et valg
 - b. Anta at valget som gir optimal løsning er gitt
 - c. Vis at etter valget har vi nye delproblem å løse
 - d. Bevis at disse må løses optimalt
2. Beskriv rekursiv løsning
3. Beregn optimal løsningsverdi
4. Eventuelt: finn optimal løsning

Eksempel: Enveis Travelling Salesman

5	13	6	1	1
2	9	3	4	1
3	3	9	7	1
6	3	8	2	4

Lovlige trekk



Eksempel: Enveis Travelling Salesman

5	13	6	1	1
2	9	3	4	1
3	3	9	7	1
6	3	8	2	4

The image shows a 4x5 grid of numbers. A path of arrows is drawn starting from the top-left cell (5) and ending at the bottom-right cell (4). The path consists of the following cells: (1,1) with value 5, (2,2) with value 9, (2,3) with value 3, (1,4) with value 1, and (4,5) with value 4. The arrows indicate the direction of the path: from (1,1) to (2,2), from (2,2) to (2,3), from (2,3) to (1,4), and from (1,4) to (4,5).

Eksempel: Enveis Travelling Salesman

5	13	6	1	1
2	9	3	4	1
3	3	-5	7	1
6	3	8	2	4

Steg 1 - Beskriv optimal substruktur

1. Vi står i mål (i, j) , og ønsker å gå baklengs
Må gå opp til venstre, rett til venstre eller ned til venstre.
2. Antar at korteste vei går gjennom i', j'
3. Optimal løsning består nå av enveis vei til i', j' pluss skrittet til mål
Å kombinere disse gir ny enveis vei, siden $j' < j$
4. 'Klipp-og-lim'-bevis for at veien til i', j' må være en korteste enveis vei dit.

Steg 2 - Rekursjon

- Base case: $j = 0$
- Ellers: minimum over alternativer

$$d[i, j] = \begin{cases} c[i, j] & ; i = j = 0 \\ -\infty & ; i \neq j = 0 \\ \min (d[i+1 \bmod m, j-1] \\ d[i, j-1], \\ d[i-1 \bmod m, j-1]) & ; \text{ellers} \end{cases}$$

Steg 3 - Beregn optimal verdi

- Rekursjonen avhenger av verdiene i forrige kolonne
- Regn ut kolonne for kolonne

Om optimal løsning og verdien av en

- Av og til er vi bare interessert i verdien av en optimal løsning
 - "Korteste vei er 17"
- Av og til er vi også interessert i selve løsningen
 - "Korteste vei går fra u til a til b til x til t"
- Må da ta vare på optimale valg som tas underveis
 - Brukes til backtracking
- Gjøres kun hvis nødvendig

Eksempel: Strengdelingssum

- For hver delstreng vi så på, lagret vi punktet det lønnet seg å dele strengen i.
- Kunne så slå opp på verdien delstrengen frem til dette punktet for å finne nye oppdelinger

Eksempel: Enveis TSP

- Lagre valg i en tabell

Ønsker du trening?

- Praktisk trening
 - [Online judge](#)
 - [IdiOpen](#)
- Teoretisk trening
 - Oppgaver i læreboka
 - Gamle eksamensoppgaver

En annen vri

- For datateknikk (i hvert fall):
Komplementæremne neste høst
- Operasjonsanalyse (TIØ4120) kan gi mer informasjon og et artig perspektiv.

Ukas øving

- Teori: Alle-til-alle korteste vei
- Praksis: Mumien

Teori

- Oppgave 1
 - Alle-til-alle basert på rekursjon på antall kanter
- Oppgave 2
 - Alle-til-alle basert på rekursjon på tillatte noder
 - Floyd Warshall
- Oppgave 3
 - Transitiv tillukning
- Oppgave 4
 - Optimal algoritme

Praksis

- Skulle finne veier med høyest sannsynlighet
- Stilengde var produkt av tall < 1
- Dermed fungerer feks Dijkstra
 - Kan umulig få en bedre verdi til noden med høyeste estimat, siden all multiplikasjon med tall < 1 gir en lavere verdi
- Må ta vare på forgjenger-noder