Velkommen til IN2010 gruppe 6

Dagens plan

- Praktisk info
- Kjøretid fra forrige uke
- De nye algoritmene
- Eksamen oppgave i plenum
- Lab

Forrige ukes algoritmer

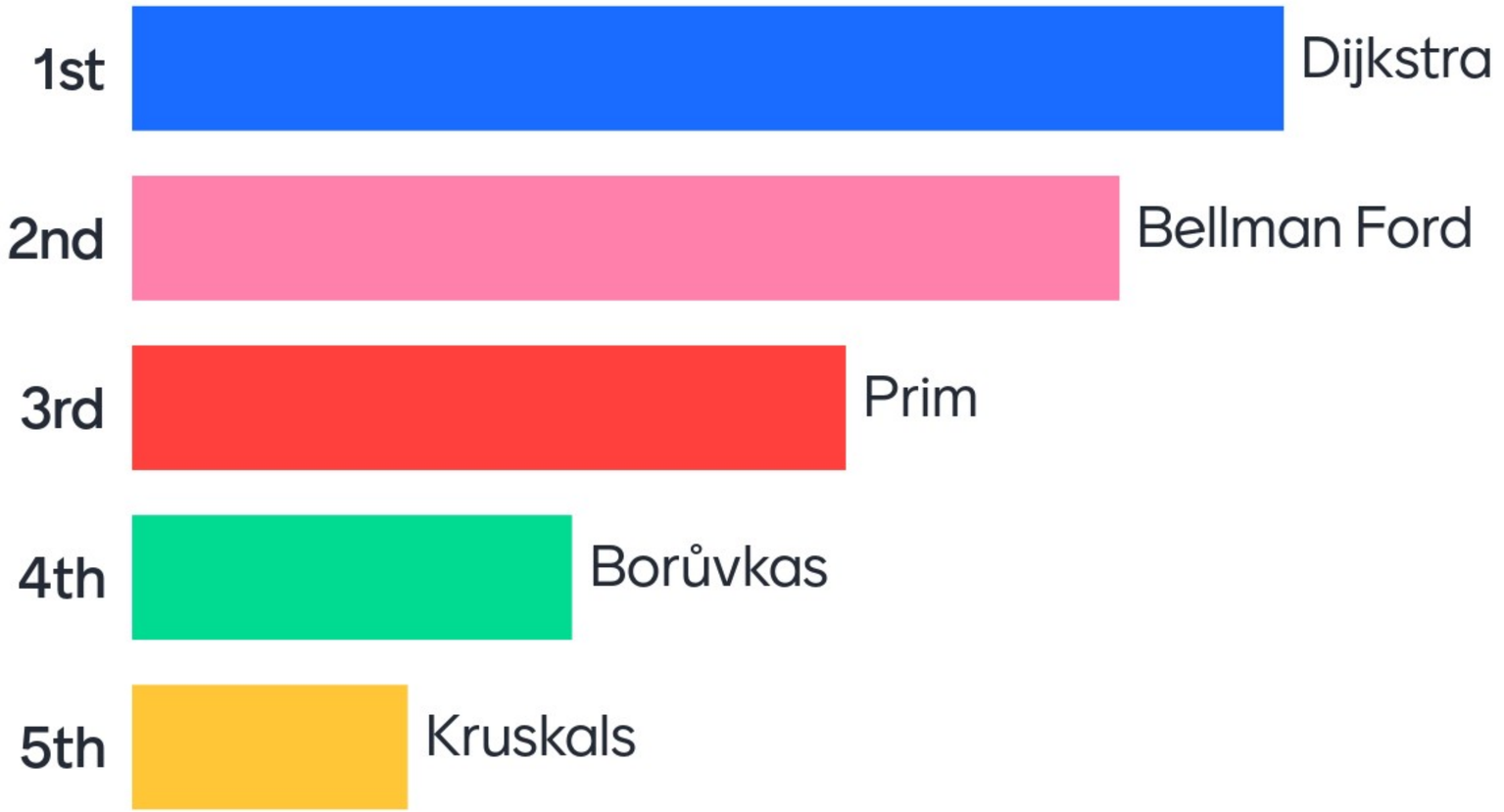
- Bredde først søk
  - Bruker en kø
- Dybde først søk
  - Bruker en stack
- Topologisk sortering
  - Starter på alle noder uten inngang
  - Bruker en stack basert på forrige punkt

Praktisk info

- Kanskje ikke så praktisk, men Will Code for Drinks er imorgen
  - Fint å få testet ut deres algoritme evner
- Innlevering 4 er nå ute
- Bruk alle ressursene dere kan finne på emnesiden
  - Notasjon om Grafer, Kjøretid, Hashing (kommer senere)

Trenger ikke å kunne implementere Kruskals og Borůvkas algoritmene (tror jeg)

Hva var vanskelig med ukas pensum?



Mentimeter

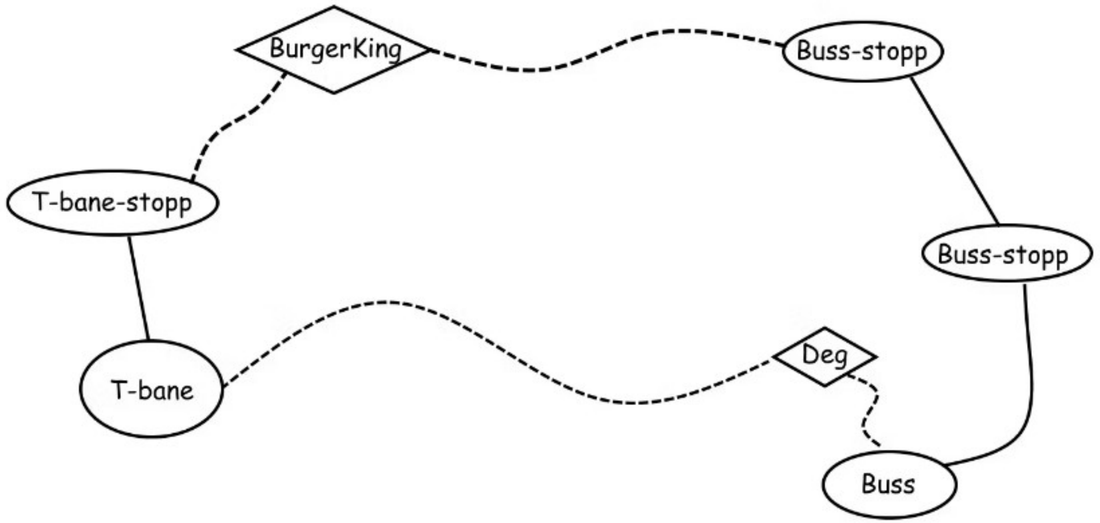
Nå over til selve ukas pensum

Mentimeter

Korteste stier

- Ofte har vi lyst til å finne den korteste stien mellom to noder i en graf
  - Finne nærmeste BurgerKing

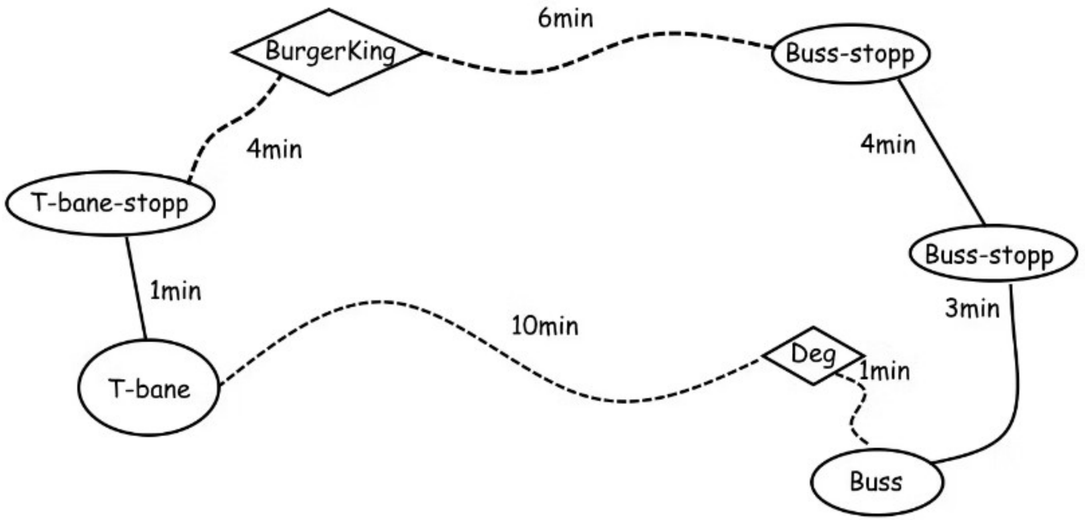
- Så langt har vi jobbet hovedsakelig med uvektede grafer
  - Mangel på representasjon av data
  - Det kan fort skje at det tar lengere tid å komme seg til den nærmeste BurgerKingen



Vektede grafer

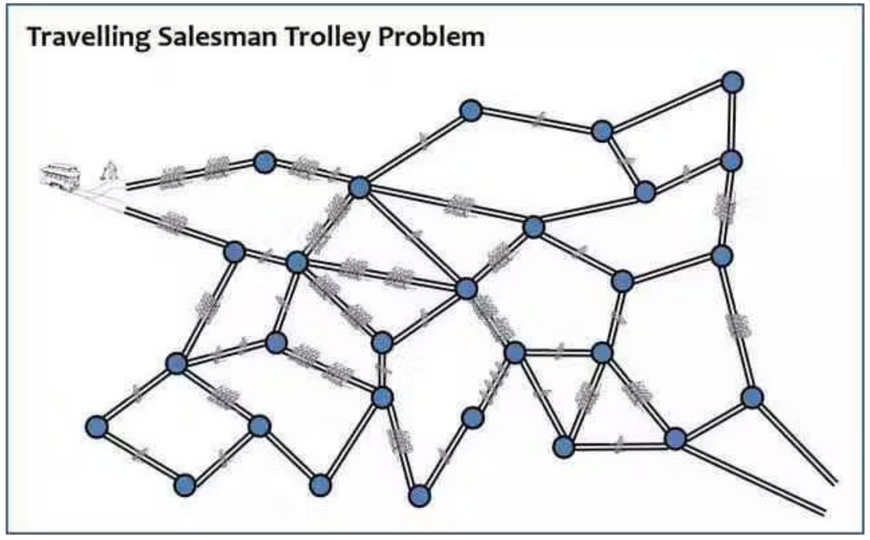
- Vektede grafer er som vanlige grafer bare at man legger til vekter...
- Det vil si at dersom man går over en kant har det en viss kostnad

- Man kan se for seg google maps som en vekta graf
  - Bruker kollektiv transport til en BurgerKing

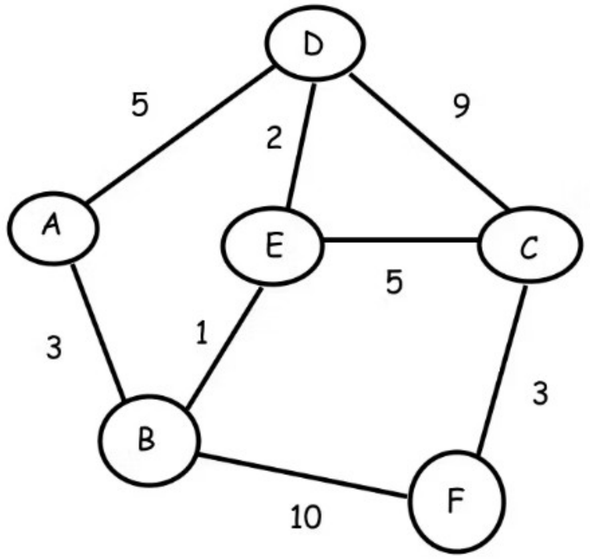


Dijkstra

- Er en veldig fin algoritme for å finne den korteste veien i en vekta graf
- Baserer seg på en prioritetskø som holder på noder med vekten det koster for å besøke den
  - Noden med minst vekt ligger først i prioritetskøen
  - Er en heap blant annet så operasjonene for å hente minste og legge inn er O(log n)

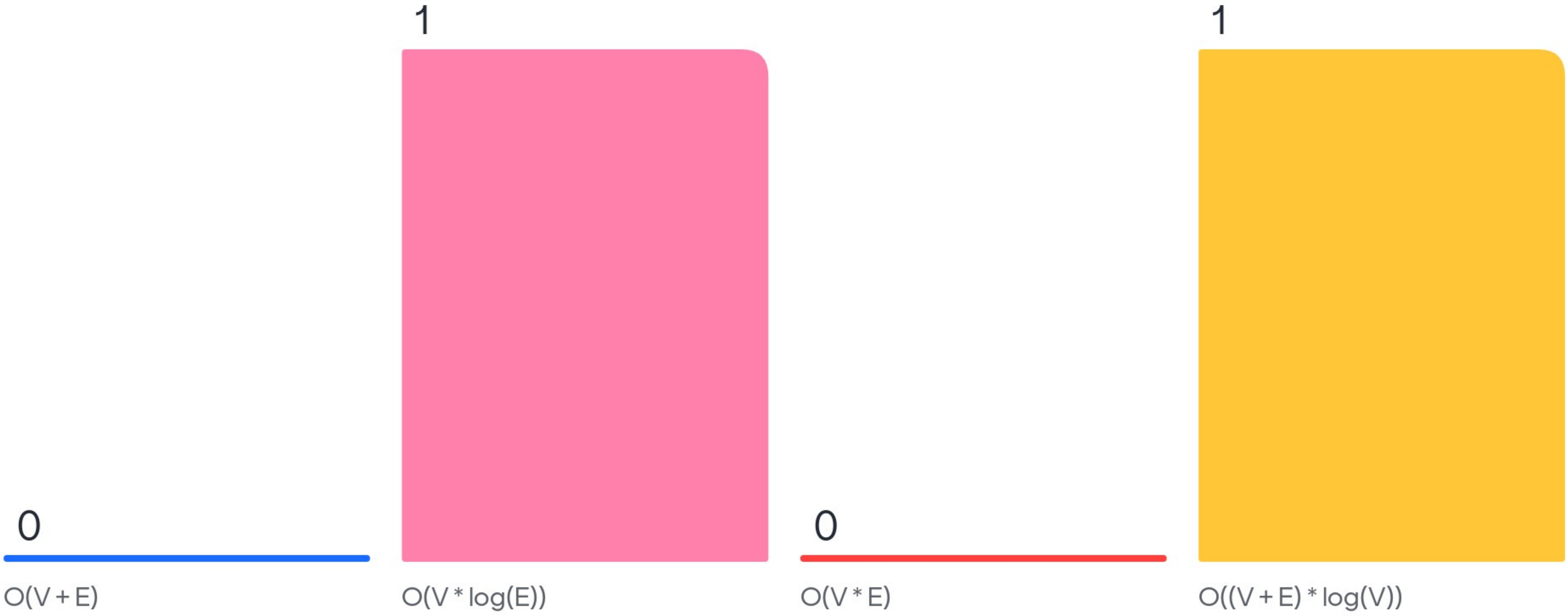


```python
def dijkstras(G, s):
    _, E, w = G
    dist = defaultdict(lambda: float('inf'))
    dist[s] = 0
    queue = [(0, s)]
    parents = {s: None}

    while queue:
        cost, v = heappop(queue)
        if cost != dist[v]:
            continue
        for u in E[v]:
            c = cost + w[(v,u)]
            if dist[u] > c:
                dist[u] = c
                heappush(queue, (c, u))
                parents[u] = v
    return parents
```



Mentimeter

Hva er kjøretiden til dijkstras? Diskuter!

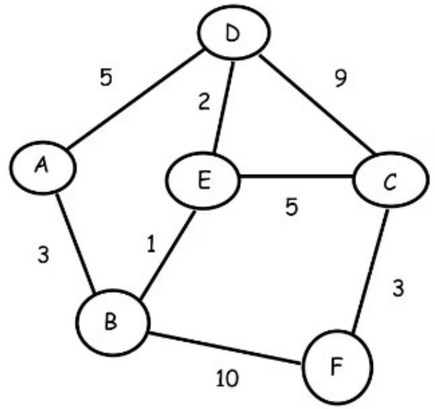


2

Negative grafer er onde

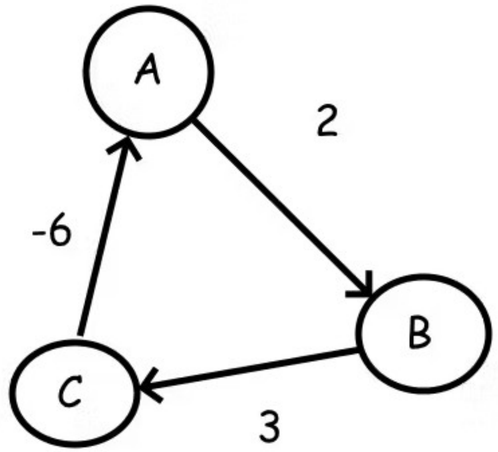
- Kan ikke bruke Dijkstra på negative grafer
  - Eksempel til høyre
  - Fører til sykler der hvor det koster "mindre" hver gang

```python
def dijkstras(G, s):
    _, E, w = G
    dist = defaultdict(lambda: float('inf'))
    dist[s] = 0
    queue = [(0, s)]
    parents = {s: None}

    while queue:
        cost, v = heappop(queue)
        if cost != dist[v]:
            continue
        for u in E[v]:
            c = cost + w[(v,u)]
            if dist[u] > c:
                dist[u] = c
                heappush(queue, (c, u))
                parents[u] = v
    return parents
```



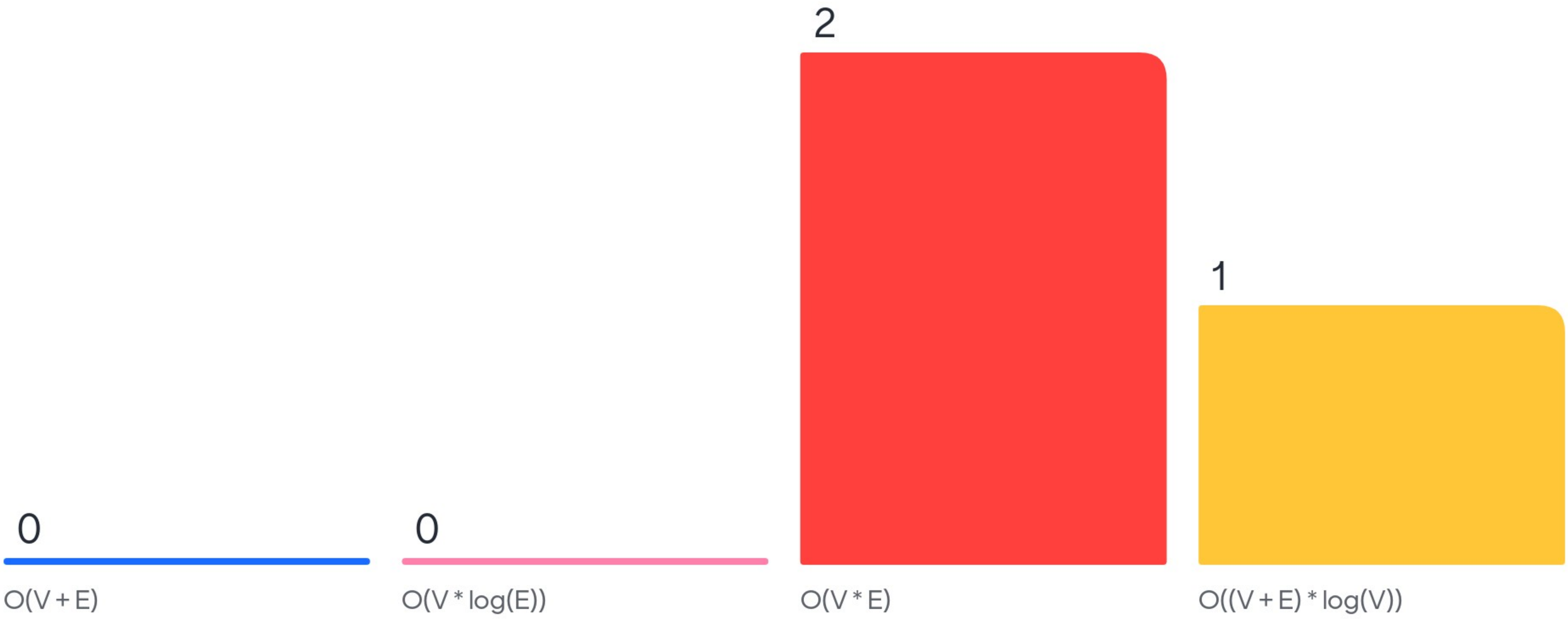
Bellman-Ford

- Er en algoritme for å finne distansen fra en start node og sjekker om det er en negativ sykel
- Baserer seg på at en sti består av |V|-1 kanter
- Setter av startnoden til å distanse 0
- Sjekker alle kanter |V|-1 ganger og oppdaterer distansene dersom det finnes en mindre en
- Etter |V|-1 iteasjoner sjekker vi en siste gang
- Dersom det er en endring er det en negativ sykel



Mentimeter

Hva er kjøretiden til bellman? Diskuter!

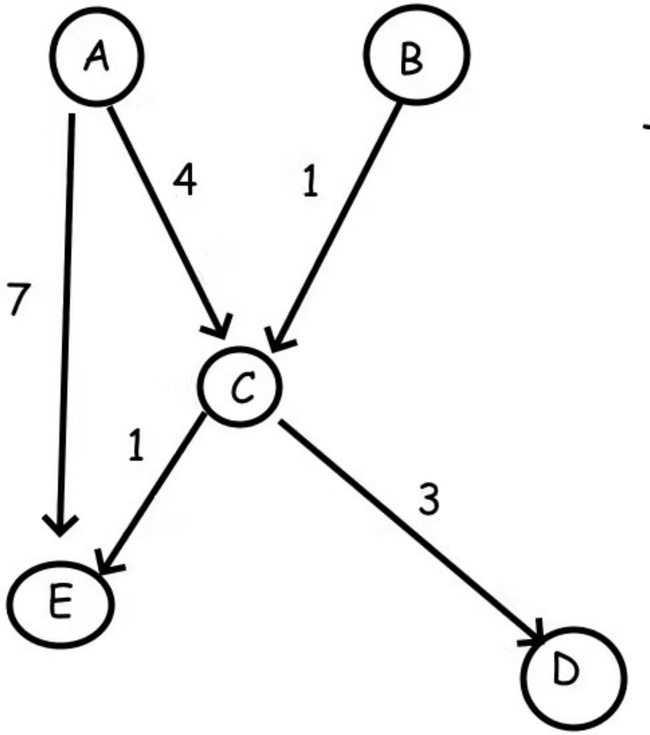


3

Mentimeter

Korteste stier i en DAG

- Sist gikk vi igjennom hva en DAG er
  - Directed-Acyclic-Graph
- Bruker TOPSort til å sortere hvilke noder vi besøker først



TOPSort:

A
B
C
E
D

Dist:

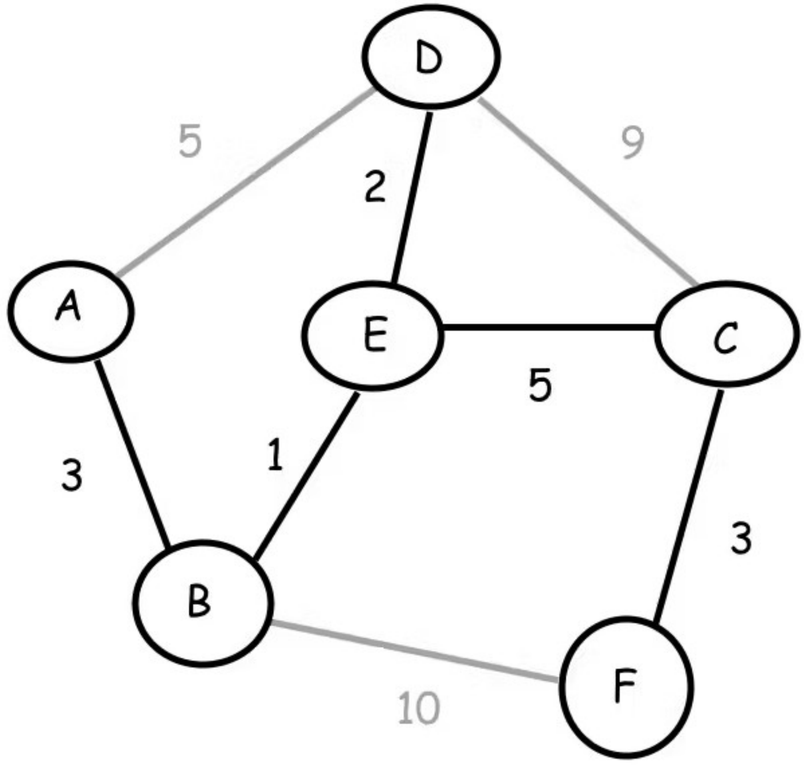
A -> 0
C -> 4
E -> 7 -> 5
D -> 3
B -> inf

Spenntrær

- Trær er sammenhengende, asyklisk, urettetede grafer, SAU
- Består av |V| - 1 kanter. Dersom man legger til |V| kanter i et tre så blir den syklisk
- Kan lage vanlige spenntrær allerede med BFS

Minimale spenntrær

- Er som et vanlig spenntre bare at man velger ut kantene basert på hvor mye vekt de har
- Målet er å "plotte" ut de minste kantene slik at man får den korteste stien fra en start node til alle andre noder



Prim

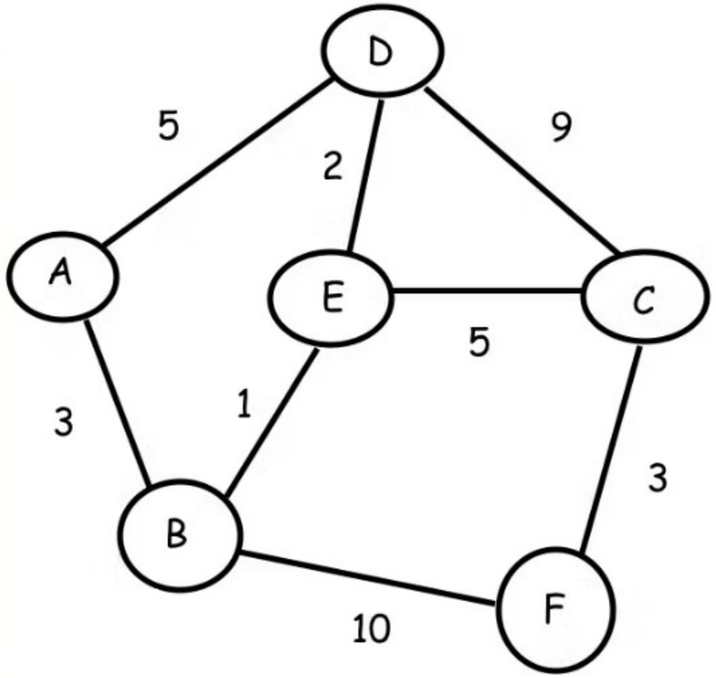
- Bruker en prioritetskø som dijkstra
- Forskjellen er at man besøker noden med **minst vekt** og ikke **minst akkumalert vekt**
- Hvis en node er besøkt fra før av så ignorerer vi den
- Hvis ikke den er besøkt hopper vi inn i den og legger til naboene med deres vekt og ikke akkumalert vekt

```python
def prim(G):
    V, E, w = G
    s = next(iter(V))
    queue = [Primtrip(0, s, None)]
    parents = dict()

    while queue:
        _, v, p = queue.pop(0)
        if v not in parents:
            parents[v] = p
            for u in E[v]:
                queue.append(Primtrip(w[(v,u)], u, v))
                queue.sort()
    return parents
```

Mentimeter

Prim eksempel

```python
def prim(G):
    V, E, w = G
    s = next(iter(V))
    queue = [Primtrip(0, s, None)]
    parents = dict()

    while queue:
        _, v, p = queue.pop(0)
        if v not in parents:
            parents[v] = p
            for u in E[v]:
                queue.append(Primtrip(w[(v,u)], u, v))
                queue.sort()
    return parents
```

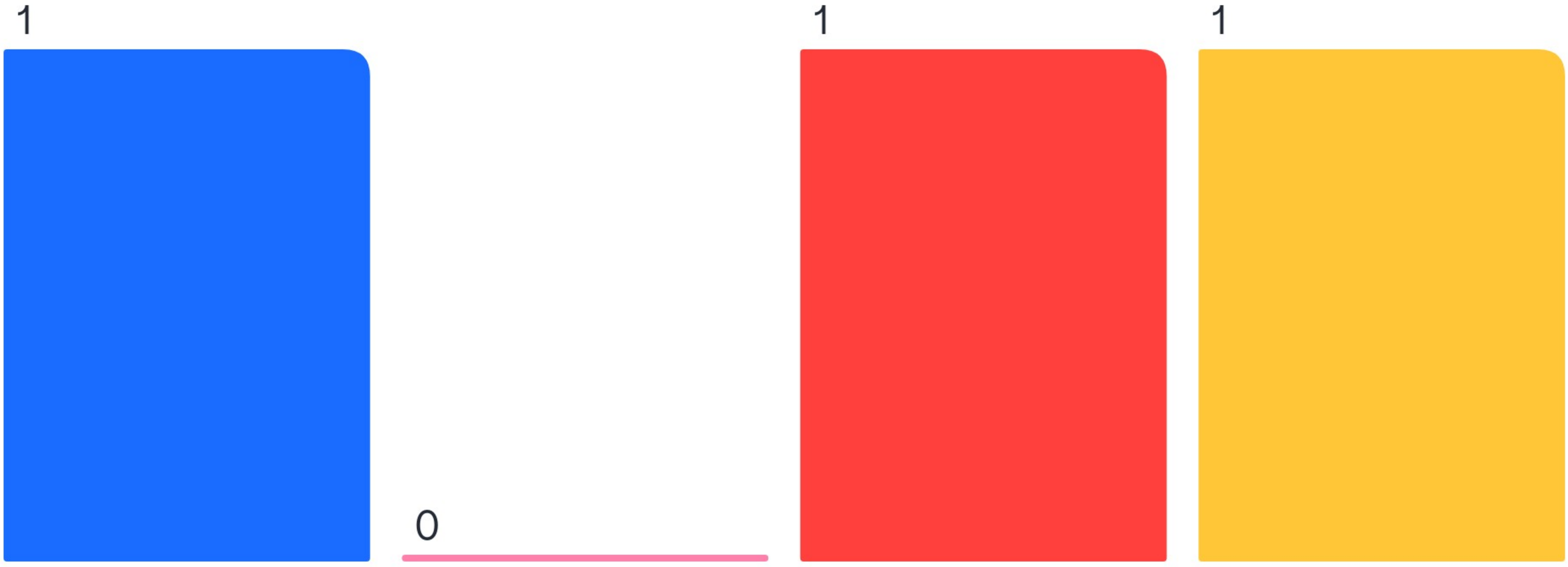


KØ: Parent:

Se for deg en heap og ikke en hjemme-mekka python sortering i eksempelet

Mentimeter

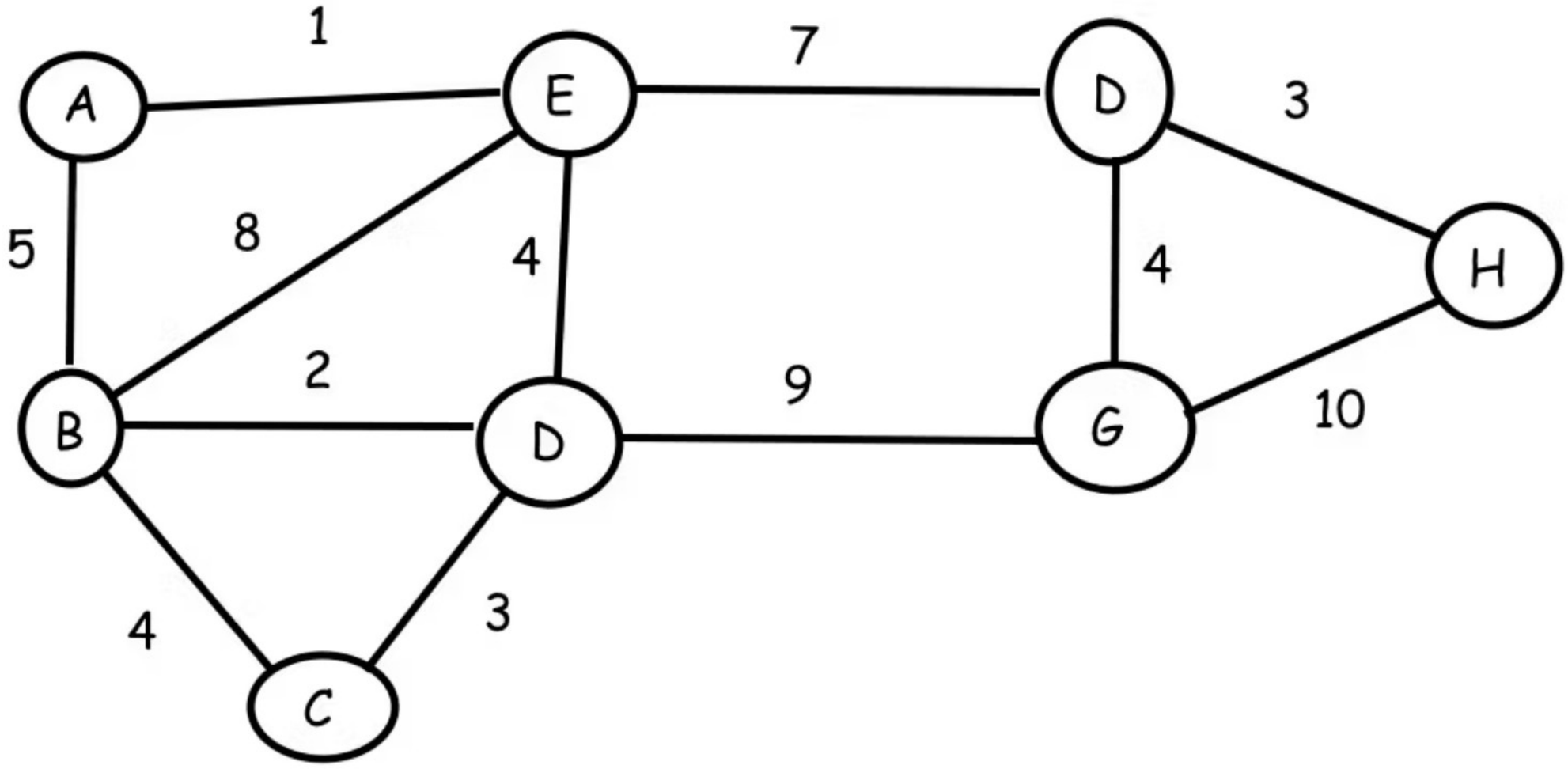
Hva er kjøretiden til prim? Diskuter!



3

Kruskals

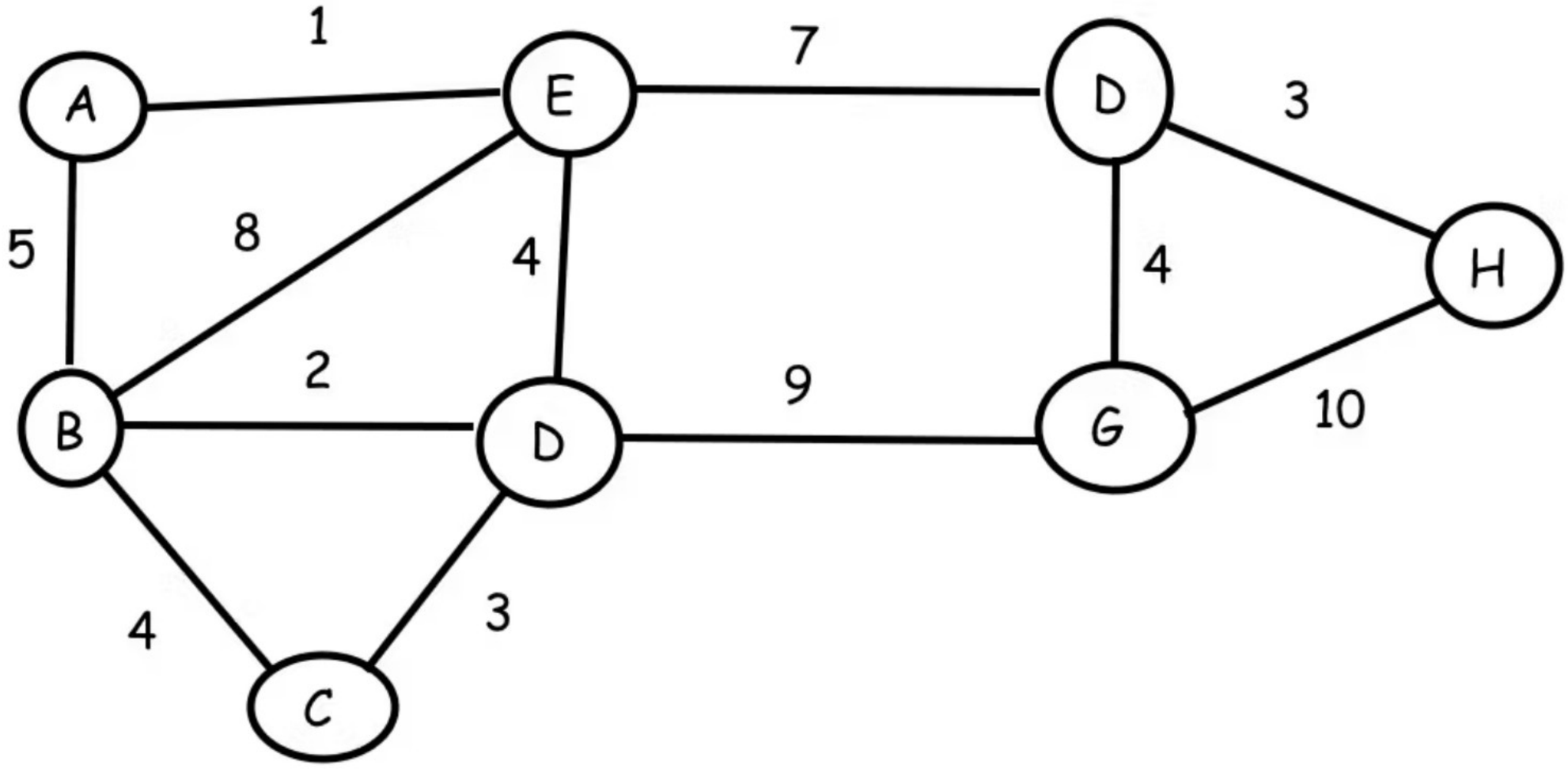
- Lager komponenter basert på vakten til kantene
- Plukker ut den billigste kanten og lager en komponent av nodene
  - Repeteres helt til...
- Sjekker hver gang om det skaper en sykel, dersom det gjøres ignoreres kanten



Borůvkas

- Ligner på Kurskals alogritme, men man kobler opp komponentene med den billigste kanten imellom dem

- Lager komponenter av de minste kantene
- Kobler opp komponentene med den billigste kanten
  - Fortsetter til vi har en sammenhegnende graf



Mentimeter

Eksamen eksempel

Blindern-problemet 10 poeng

Blindern-problemet er en utfordring som er kjent for mange studenter ved Universitet i Oslo, som oppstår når man må gå fra en forelesning til en annen på motsatt side av campus på et knapt kvarter. Du er blitt bedt om å konsultere i UiO sitt nye prosjekt hvor det skal utvikles et tunnelsystem som forbinder alle bygningene som hører til campus.

Det er allerede kartlagt hvor mye det vil koste å grave tunnel mellom hvert par av bygninger (og anta at ingen av de kartlagte tunnelene kolliderer). Din oppgave er å finne den mest kostnadseffektive måten å grave et tunnelsystem på, ut ifra de kartlagte tunnelene, slik at man kan gå mellom alle bygningene kun ved å bruke tunnelsystemet.

(a) Forklar kort hvordan dette problemet kan uttrykkes som et grafproblem. Svaret ditt bør inkludere:

- Hva slags graf (rettet/urettet og vektet/uvektet) egner seg her?
- Hva representerer nodene?
- Hva representerer kantene?

(b) Oppgi en egnet algoritme, kjent fra pensum, som kan brukes til å finne den mest kostnadseffektive måten å grave ut et tunnelsystem på, slik at man kan komme seg mellom alle bygningene kun ved å bruke tunnelsystemet. For algoritmen må du oppgi:

- de mest sentrale datastrukturene den bruker,
- en *kort* forklaring på hvordan algoritmen fungerer, og
- kjøretidskompleksiteten på algoritmen.

(c) Prosjektet blir ferdig i rekordfart og er klar til å brukes. Beskriv, med naturlig språk, en effektiv algoritme for å finne korteste sti fra en bygning til en annet kun ved å bruke tunnelsystemet. Du kan bruke algoritmer kjent fra pensum *uten* å gjøre rede for dem. Oppgi kjøretidskompleksiteten på algoritmen.

Fra eksamen 2023

Mentimeter

Thats it folks
Hvis dere lurer på noe er det bare å sende mail:
mafredri@ifi.uio.no