Science fair:

5 - 10 min, vi er nummer 7 ut

Primtallet som en redusert delmengde

- Anta en mengde $\{a, b, c, d, e\}$. Vi kan beskrive dens delmengder på ulike måter:
  - Ved hjelp av indekser: $\{1, 3\}$, som representerer delmengden som inneholder det andre elementet fra mengden, samt det fjerde, dvs. $\{b, d\}$
  - Ved hjelp av primtall: la det første primtallet indeksere det første elementet, det andre primtallet indeksere det andre elementet, og så videre. Med andre ord, slik funksjon:

$$\{< 2, a >, \ < 3, b >, \ < 5, c >, \ < 7, d >, \ < 11, e >\}$$

- Hensikten med dette er å spare plass. Vi kan bruke bare fire bytes i stedet for enda flere, avhengig av antallet integers vi ønsker å bruke. For eksempel, $7 \cdot 13 \cdot 47 \cdot 89 = 380\,653$ kan allerede i teorien representeres med 3 bytes, mens for integer så må vi bruke $4 \cdot 4 = 16$, med mindre vi velger å bruke 1 byte per integer, da de indeksene som de fire primtallene står for, vil være under tallet 255, som vil si at vi kan i stedet bruke bare 4 bytes - som fortsatt er større enn 3.

- I teorien, kan vi også bruke en annen tolkning av bytes når det kommer til slike primtall. Vi vet, at alle partall større enn 2 vil aldri kunne være primtall, da de partallene er på formen $2 \cdot n$, der $n > 1$, dvs. ha en faktor av 2. La oss derfor bruke denne tolkningen av en byte:

  $0000\ 0000 \rightarrow 2$
  $0000\ 0001 \rightarrow 3 = 1 + 2 \cdot 1$
  $0000\ 0010 \rightarrow 5 = 1 + 2 \cdot 2$
  $0000\ 0011 \rightarrow 7 = 1 + 2 \cdot 3 \ldots$

  Slik at det siste tallet som kan representeres av en byte er
  $1 + 2 \cdot \left(2^8 - 1\right) = 511$

- Videre er spørsmålet om hvorvidt vi skal bestemme de aktuelle primtallene mens vi jobber med dataen, eller ha de regnet ut på forhånd. For det andre, kan vi ha en løsning hvor vi mapper indekser til primtall, m.a.o.:

  $[2,\ 3,\ 5,\ 7,\ 11,\ \ldots]$

Anta vi har 10 000 primtall å ta vare på (primtall nr. 10 000 er 104 729). Følgende er noen beregninger for ulike valg av teknikker:

- Dersom vi har et array med 4 bytes for hvert element, representerende et primtall:
  $10\,000 \cdot 4$ bytes $= 40\,000$ bytes $\approx 39,1$ kB

- Siden ikke alle bytes brukes, kan vi benytte oss av 3 bytes for hvert primtall ($\log_2(104\,729) \approx 16,7$ gir oss at vi må bruke 24 bits, dvs. 3 bytes). Da bruker vi så mye plass:
  $10\,000 \cdot 3$ bytes $= 30\,000$ bytes $\approx 29,3$ kB (74,9% av opprinnelig minne)

- Fra det forrige hovedpunktet om å bevare kun oddetall og tallet 2, kan vi representere 511 som det høyeste tallet ved hjelp av kun én byte, og $1 + 2 \cdot \left(2^{2 \cdot 8} - 1\right) = 131\,071$ som det høyeste tallet ved to bytes. Det tallet er også større enn primtallet vårt 104 729. Da vil vi kun bruke:
  $10\,000 \cdot 2$ bytes $= 20\,000$ bytes $\approx 19,5$ kB (49,9% av opprinnelig minne)

- Andre teknikker kan også benyttes, som kun én byte på alle primtall inntil 511, og så to bytes på alle primtall etter det. Man kunne også ha benyttet gamma-enkoding, men dette har vi ikke regnet på.

Det er enkelt å finne frem til det n'te primtallet, som har kompleksitet O(1). Å gå andre veien derimot har kompleksitet O(log(n)), da vi kan bruke binærsøk.

En annen struktur kunne også ha blitt brukt enn et array, for eksempel et HashMap.

- Så langt har vi ikke hatt noen rekkefølge på elementer innbyrdes. Foreløpig sier vi kun hvilke elementer som er med uten noen fast rekkefølge. Vi kan anta at rekkefølgen de har er slik som de er i den opprinnelige samlingen (ikke glem at i mengder så neglisjerer vi rekkefølgen og hvor mange hvert element forekommer).

  For å ta høyde for rekkefølgen, eller eventuelt hvor mange ganger et element skal forekommer i samlingen, kan vi multiplisere et primtall flere ganger, f.eks. $11 \cdot 11$ kan bety "det femte elementet plasseres på andre plass" eller "det femte elementet gjentas to ganger".

bruksområder:

description:

skal ta et eksempel og diskutere det. Eksemplet går ut på å komprimere et dokument. Dette gjøres ved å lage en mapping fra hvert ord til et korresponderende primtall. Her burde en passe på at de mest brukte ordene har et lavt primtall. jeg kommer til å bruke "ja, en blå bil ja.” som et lett eksempel.

Jeg kommer så til å si noe om egenskapene til dette bruksområdet. Her er hovedpunktene det faktum at det er lossy, hvordan en muligens kan bruke representasjonen til søking og hvordan variering i setningslengde kan påvirke plass og informasjon.

key notes:

- eksempel
    - komprimere et dokument / dokumenter i et corpus
    - sette opp ord, primtall mapping med typisk arrays eller hashmap
    - mest vanlig først
    - komprimere setning
        - ja, en blå bil ja
        - 2, 3, 5, 7 eller 2^2, 3, 5, 7
        - 210, eller 420
- egenskaper
    - kan bruke binærsøk til å søke etter korresponderende primtall om sortert, men i dette tilfellet, lineært
    - så sjekke om en setning inneholder et primtall ved å bruke modulo
    - lossy
        - justere setningslengde
        - gå tilbake til eksempel på tavle for å illustrere

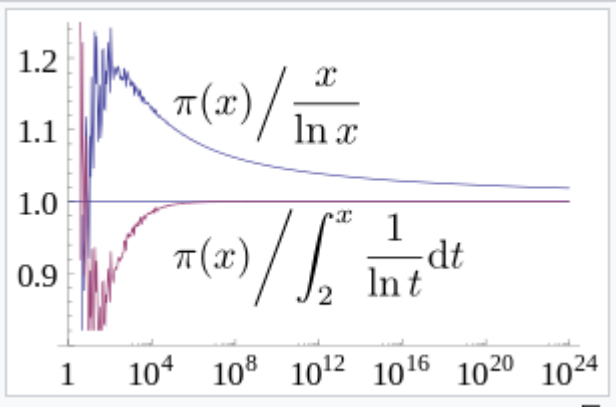
ulemper og hvorfor det kan være vanskelig å bruke:

- **Starten av kompresjonen**
- Sieve of eratosthenes(eratosthenes sil) tids kompleksitet er n log(log(n))
    - Vanskeligere å bruke for de første n(antall unike ord) primtall
        - Vi kan ikke vite hvor mange primtall er det mellom 0 og antall ord i dokumentet. Da må vi kjøre silen på en estimat av the n'te
        - kjøre sieve på en estimat av størrelsen av den n'te primtallet

A better approximation is[36]

$$\frac{p_n}{n} = \log n + \log\log n - 1 + \frac{\log\log n - 2}{\log n} - \frac{(\log\log n)^2 - 6\log\log n + 11}{2(\log n)^2} + o\left(\frac{1}{(\log n)^2}\right).$$

- primtallsfaktorisering for dekomprimering
    - tregt å gjøre
    - lett å parallellisere
- tall kan eksplodere i størrelse ved mange primtalls produkter
    - asymptotic law of prime number distribution
    - Avstanden mellom primtall øker når størrelsen på tallene øker. Hvis dokumentet som skal bli komprimert har mange tokens, da går effektiviteten av komprimeringen ned.
- primtall blir fort store
    - stor vocabulary/dictionary
        - gjenbruk på en corpus kommer til gi stor vokabular
    - Mange sjeldne tokens kommer til å gi store primtall. Så hvis vi prøver å gjenbruke primtalls en map på flere dokumenter, noen dokumenter kan bruke et sjeldent ord mange ganger. Dette kommer til å få encodingene til å bli veldig store
    - Kan prøve å gamma encode



- må sjekke hvert produkt med divisjon/modulo
- Huffman coding - n log(n)
    - Lettere å bruke og raksere
    - Løser samme problemet som algoritmen vår.
    - trenger ikke primtall eller divisjon/modulo
    - lossless