Subanagram

Vi ønsker å lage en nettside for å hjelpe sleipe Scrabble-spillere. En Scrabble-spiller har noen bokstaver som de skal prøve å forme ord fra, og disse ordene må stå i en ordbok. I denne oppgaven ignorerer alle andre Scrabble-regler, og fokuserer på det vi skal kalle *subanagrammer*.

La oss anta at bokstavene man har tilgjengelig er gitt som en streng S, og at et ord W også er gitt som en streng. Vi sier at W er et *subanagram* av S, dersom W kan skrives ved å kun bruke bokstaver fra S.

Merk at vi ikke tillater gjenbruk av bokstaver fra S. For eksempel er `"hole"` et subanagram av `"ehlo"`. Derimot er `"hello"` *ikke* et subanagram av `"ehlo"`, fordi det ikke er to `"l"`-er i `"ehlo"`. Både `"hole"` og `"hello"` er subanagrammer av `"hheelloo"`.

Her er noen eksempler på subanagramer av `"aghilmort"`:

- `"algorithm"`
- `"logarithm"`
- `"alright"`
- `"right"`
- `"math"`
- `"git"`

I oppgavene som følger ser vi på ulike måter å finne subanagrammer fra en gitt ordbok.

(a)

ALGORITHM: ER W ET SUBANAGRAM AV S?

Input: En streng W og en streng S
Output: Returner **true** hvis W et subanagram av S, **false** ellers

```
1 Procedure IsSubanagramOf1(W, S)
2     r ← copy of W
3     for c in S do
4         if c is in r then
5             remove an occurrence of c from r
6     return r.length = 0
```

Algoritmen ovenfor skal sjekke om ordet W kan skrives med bokstavene fra S.

- Anta at en streng fungerer akkurat som et array med bokstaver.
- Å kopiere et array gjøres i lineær tid.
- Å sjekke om et element er i et array gjøres i lineær tid.
- Å fjerne et element fra et array gjøres i lineær tid.
- Å finne lengden av et array gjøres i konstant tid.
- Vi lar w angi størrelsen på W og s angi størrelsen på S.

Hva er kjøretidskompleksiteten til algoritmen?

(b)

Du skal nå implementere et alternativ til `IsSubanagramOf1`. Du skal ta utgangspunkt i en *frekvenstabell*. Anta at du har en prosedyre `FreqTable` tilgjengelig, som bygger et map fra bokstaver til antall forekomster i lineær tid.

Hvis F er en frekvenstabell for strengen `"abbcccdddd"` vil for eksempel F.`get("a")` returnere 1 og F.`get("d")` returnere 4. Du kan anta at F.`get`(x) returnerer 0 for alle bokstaver som ikke er med i ordet frekvenstabellen bygges fra. Du kan oppdatere verdien i frekvenstabellen ved å bruke F.`put`, som et vanlig map. Både `get` og `put` er i $O(1)$.

Fullfør implementasjonen av `IsSubanagramOf2`, slik at den har kjøretidskompleksitet $O(w + s)$, der w er størrelsen av W og s er størrelsen av S.

ALGORITHM: ER W ET SUBANAGRAM AV S?

Input: En streng W og en streng S
Output: Returner **true** hvis W et subanagram av S, **false** ellers

```
1 Procedure IsSubanagramOf2(W, S)
2     F ← FreqTable(W)
      // Fyll ut
```

(c)

Strategi 1

```
1  Procedure SubanagramsOf1(D, S)
2      r ← empty list
3      for i ← 0 to d − 1 do
4          W ← D[i]
5          if IsSubanagramOf2(W, S)
            then
6              add W to r
7      return r
```

I pseudokoden ovenfor er vi gitt en ordbok `D` av lengde d, som er et array av strenger. Den bruker `IsSubanagramOf2` fra forrige deloppgave til å finne alle ord i ordboka som er et subanagram av S. Å finne subanagrammer med denne teknikken kjører i $O(d \cdot (w + s))$, der d er størrelsen på ordboka, w er størrelsen til det største ordet i ordboka, og s er størrelsen til S.

Strategi 2

```
1  Procedure Build(D)
2      M ← empty map
3      for i ← 0 to d − 1 do
4          sortedword ← Sort(D[i])
5          if M.get(sortedword) is empty then
6              M.put(sortedword, empty list)
7          add D[i] to M.get(sortedword)
8      return M
9  Procedure SubanagramsOf2(M, S)
10     r ← empty list
11     for x in sorted substrings of S do
12         append M.get(x) to r
13     return r
```

I pseudokoden ovenfor lager vi et hashmap, der hver nøkkel k er en streng med bokstaver sortert i alfabetisk rekkefølge, og verdien er en liste med anagrammer av k som finnes i ordboka. For å finne alle subanagrammer i ordboken, må vi slå opp på alle sorterte substrenger av S i hashmapet. Å finne subanagrammer med denne teknikken kjører i $O(2^s)$, altså eksponensiell tid med hensyn til størrelsen av S.

Drøft fordeler og ulemper mellom strategi 1 og strategi 2 med tanke på kjøretid.

(d)

Nå som nettsiden vår har funksjonalitet for å finne subanagrammer av en streng S, gjenstår det bare å presentere resultatene for brukeren. Anta at subanagrammene som blir returnert av `Subanagrams0f2` er gitt som et array, sortert *alfabetisk*. Vi ønsker at subanagrammene skal vises sortert etter *lengde*, og at ord av samme lengde sorteres alfabetisk, slik som i følgende eksempel:

Input	**Output**
`"algorithm"`	`"algorithm"`
`"alright"`	`"logarithm"`
`"git"`	`"alright"`
`"logarithm"`	`"right"`
`"math"`	`"math"`
`"right"`	`"git"`

Fyll ut pseudokode nedenfor med så lav kjøretidskompleksitet som mulig.

ALGORITHM: SORTERING AV SUBANAGRAMMER

Input: Et array A av størrelse n som inneholder strenger
Output: Et array med de samme n strengene sortert etter lengde der ord av samme lengde er sortert alfabetisk

```
1 Procedure SortSubanagrams(A)
    // Fyll ut
```