ERHVERVSAKADEMI ÅRHUS

# Fletning

## 1. Fletning

I denne note skal vi betragte en klasse af problemstillinger, hvis løsning til en vis grad kan standardiseres. Problemet består i at flette to sorterede sekvenser og kan illustreres ved hjælp af et lille eksempel. Den totale fletning af sekvenserne

[3, 7, 9, 9] og [1, 2, 4, 7, 7, 8]

er sekvensen

[1, 2, 3, 4, 7, 7, 7, 8, 9, 9]

I dette eksempel er resultatet en sorteret sekvens, der indeholder alle elementerne fra de to sekvenser, der flettes. På samme vis kan man for eksempel finde de elementer de to sekvenser har fælles eller de elementer der er i den ene men ikke den anden sekvens, eller noget helt fjerde i forhold til de to sekvenser, der flettes.

I noten formuleres en generel skabelon for fletning af sorterede sekvenser, og der præsenteres et eksempel på konkretisering af skabelonen. I en konkret programmeringsopgave kan en sekvens være en ArrayList, et array, en fil mv. Ved formulering af skabelonen ønsker vi at abstrahere fra den konkrete datarepræsentation. Derfor repræsenteres sekvenser som et redskab til formulering af skabelonen.

For at få forståelse af fletningen i de tilfælde, der er nævnt overfor, er det en god idé at udføre fletning på papir, eller med et program der kan simulere processen.

## 2. Skabelon for total fletning af sekvenser

Som nævnt i indledningen, er en total fletning af to sorterede sekvenser en sorteret sekvens, som indeholder præcis de elementer, der tilsammen findes i de to sekvenser. For eksempel er den totale fletning af sekvenserne s1 og s2 sekvensen result.

s1 = [3, 7, 9, 9]
s2 = [1, 2, 4, 7, 7, 8]
result = [1, 2, 3, 4, 7, 7, 7, 8, 9, 9]

## Total fletning

Den totale fletning af to sekvenser er en sorteret sekvens, der indeholder præcis de elementer, der findes i de to sekvenser.

En algoritme, der danner den totale fletning af to sorterede sekvenser s1 og s2, kan formuleres som følger, hvor ideen er, at elementerne fra de to sekvenser behandles ét for ét ved et (flettet) sekventielt gennemløb af de to sekvenser:

```
Algoritme: totalFlet
Input: s1: sorteret sekvens, s2: sorteret sekvens
Returnerer result: sorteret sekvens

Metode:
result = den tomme sekvens
while (der er elementer i s1 && der er elementer i s2) {
  e1 = det element man er kommet til i s1
  e2 = det element man er kommet til i s2
  if (e1 <= e2) {
    tilføj e1 bagest i resultat
    gå videre i s1
  }
  else {
    tilføj e2 bagest i resultat
    gå videre i s2
  }
}

while (der er elementer i s1) {
  tilføj det element man er kommet til i s1 til resultat
  gå videre i s1
}

while (der er elementer i s2) {
  tilføj det element man er kommet til i s2 til resultat
  gå videre i s2
}

return result;
```

## 2.1 Konkretisering af total fletning med ArrayList

Et eksempel på anvendelse af skabelonen for total fletning, er fletning af to sorterede ArrayLists s1 og s2. Elementerne fra listerne behandles ét for ét ved et sekventielt gennemløb af de to lister, og elementerne overføres til en ny ArrayList, der til slut indeholder alle elementerne i sorteret rækkefølge.

```
/**
 * Returnerer en sorteret ArrayList der indeholder alle
 * elementer fra både s1 og s2
 * Krav: s1 og s2 er sorterede og indeholder Integer
 */
public static ArrayList<Integer> flet(ArrayList<Integer> s1,
ArrayList<Integer> s2) {
   ArrayList<Integer> result = new ArrayList<Integer>();
   int i1 = 0;
   int i2 = 0;

   // flet sålænge der er noget i begge lister
   while (i1 < s1.size() && i2 < s2.size()) {
      if (s1.get(i1).compareTo((s2.get(i2))) <= 0) {
         // s1's første tal er mindst
         result.add(s1.get(i1));
         i1++;
      }
      else { // s2's første tal er mindst
         result.add(s2.get(i2));
         i2++;
      }
   }

   // tøm den liste der ikke er tom
   while (i1 < s1.size()) {
      result.add(s1.get(i1));
      i1++;
   }

   while (i2 < s2.size()) {
      result.add(s2.get(i2));
      i2++;
   }

   return result;
}
```

**Forklaring til koden:**

```
// flet sålænge der er noget i begge lister
while (i1 < s1.size() && i2 < s2.size()) {
   if (s1.get(i1).compareTo((s2.get(i2))) <= 0) {
      // s1's første tal er mindst
      result.add(s1.get(i1));
      i1++;
   }
   else { // s2's første tal er mindst
      result.add(s2.get(i2));
      i2++;
   }
}
```

Så længe der er elementer i begge lister, der ikke er undersøgt, undersøges hvilken liste, der har det mindste tal forrest: s1.get(i1) eller s2.get(i2). Hvis det er s1, tilføjes s1.get(i1) til result, og man kommer videre til næste element i s1. Omvendt hvis det er s2.

Når alle elementer i en af listerne er tom, flyttes alle elementer i den ikke tomme over til result:

```
while (i1 < s1.size()) {
   result.add(s1.get(i1));
   i1++;
}

while (i2 < s2.size()) {
   result.add(s2.get(i2));
   i2++;
}
```

Bemærk, at programmet kun går ind i den ene af disse while-sætninger, i det den anden ArrayList allerede er blevet tømt.

## 2.2. Konkretisering af total fletning med int-arrays

Der kan tilsvarende laves en konkretisering med int-arrays – dette kræver kun få ændringer. Ændringerne er herunder markeret med gult:

```
/**
 * Returnerer en sorteret ArrayList der indeholder de
 * elementer der findes både i s1 og s2
 * Krav: s1 og s2 er sorterede
 */
// total fletning arrays
public ArrayList<Integer> flet(int[] s1, int[] s2) {
```

```
  ArrayList<Integer> result = new ArrayList<Integer>();

  int i1 = 0;
  int i2 = 0;
  // flet sålænge der er noget i begge lister
  while (i1 < s1.length && i2 < s2.length) {
    if (s1[i1]<s2[i2]) { // s1's første tal er mindst
      result.add(s1[i1]);
      i1++;
    }
    else { // s2's første tal er mindst
      result.add(s2[i2]);
      i2++;
    }
  }

  // tøm den liste der ikke er tom
  while (i1 < s1.length) {
    result.add(s1[i1]);
    i1++;
  }

  while (i2 < s2.length) {
    result.add(s2[i2]);
    i2++;
  }

  return result;
}
```

Da det næppe naturligt at returnere tallene i en ArrayList, når de oprindelige tal er i et array, ændrer vi derfor yderligere så metoden returnerer et array med de flettede tal:

```
/**
 * Returnerer et sorteret array der indeholder de
 * elementer, der findes både i s1 og s2
 * Krav: s1 og s2 er sorterede
 */
// total fletning arrays
public int[] flet(int[] s1, int[] s2) {
  int[] result = new int[s1.length + s2.length];

  int i1 = 0;
  int i2 = 0;
  int j = 0;
  // flet sålænge der er noget i begge lister
  while (i1 < s1.length && i2 < s2.length) {
    if (s1[i1] < s2[i2]) { // s1's første tal er mindst
      result[j] = s1[i1];
      i1++;
```

```
            j++;
        }
        else { // s2's første tal er mindst
            result[j] = s2[i2];
            i2++;
            j++;
        }
    }

    // tøm den liste der ikke er tom
    while (i1 < s1.length) {
        result[j] = s1[i1];
        i1++;
        j++;
    }

    while (i2 < s2.length) {
        result[j] = s2[i2];
        i2++;
        j++;
    }
    return result;
}
```

Og hermed har vi den endelige version af fletning af to int-arrays.

# 3 Generel fletning

Idéen fra algoritmerne i forrige afsnit er central i den generelle fletteskabelon. Men ved generel fletning er det ikke givet, at de aktuelle elementer i s1 henholdsvis s2 skal flyttes over i result – det afhænger af den konkrete anvendelse. Hvis man fx er interesseret i de elementer, der netop findes i begge sekvenser, er det altså kun, når tallene findes i begge de to sekvenser, der skal ske en flytning til result. Da er det den generelle fletteskabelon, der skal anvendes.

## 3.1. Skabelon for generel fletning af sekvenser

Skabelonen for generel fletning tager udgangspunkt i skabelonen for total fletning. Da det ikke er givet at alle elementer fra de to oprindelige sekvenser skal med i resultat sekvensen, betyder det at der er tre tilfælde der skal håndteres:

a) s1 har mindste element
b) s2 har mindste element
c) s1 og s2 har ens elementer

Dette giver anledning til følgende skabelon, hvor if-sætningen i while er udvidet med endnu et tilfælde.

```
Algoritme: generelFlet
Input: s1: sorteret sekvens, s2: soreret sekvens
Returnerer result: sorteret sekvens

Metode:
result = den tomme sekvens
while (der er elementer i s1 && der er  elementer i s2) {
  e1 = det element man er kommet til i s1
  e2 = det element man er kommet til i s2

  if (e1 < e2) {
    håndter elementet e1
    gå videre i s1
  } else if (e1 > e2) {
    håndter elementet e2
    gå videre i s2
  }
  else { // e1==e2
    håndter e1 og e2
    gå videre i s1 og s2
  }
}

while (der er elementer i s1) {
  håndter det element man er kommet til i s1
  gå videre i s1
}

while (der er elementer i s2) {
  håndter det element man er kommet til i s2
  gå videre i s2
}

return result;
```

## 3.2 Konkretisering af generel fletning med ArrayList

I denne konkretisering defineres problemet til at være det at lave en sorteret sekvens, der indeholder de elementer de to oprindelige sekvenser har fælles.

Fx, hvis s1 = {3, 7, 9, 11, 13, 17} og s2 = {5, 7, 8, 13, 20} skal result = {7, 13}.

Vi tager udgangspunkt i eksemplet med ArrayList, og viser en metode, der med generel fletning finder og returnerer en ArrayList med de elementer, der findes i begge inputlister:

```
/**
 * Returnerer en sorteret ArrayList der kun indeholder de
 * elementer, der findes både i s1 og s2
 * Krav: f1 og f2 er sorterede og indeholder Integer
 */
public ArrayList<Integer> intersection(
  ArrayList<Integer> s1, ArrayList<Integer> s2) {

    ArrayList<Integer> result = new ArrayList<Integer>();
    int i1 = 0;
    int i2 = 0;
    while (i1 < s1.size() && i2 < s2.size()) {
        if (s1.get(i1) < s2.get(i2))
            i1++;
        else if (s1.get(i1) > s2.get(i2))
            i2++;
        else {
            result.add(s1.get(i1));
            i1++;
            i2++;
        }
    }
    return result;
}
```

I denne konkretisering af generel fletning, skal der ikke ske noget efter alle elementer i den ene sekvens er behandlet. Det betyder at nedenstående del af den generelle fletning helt udgår:

```
while (der er elementer i s1) {
  håndter det element man er kommet til i s1
  gå videre i s1
}

while (der er elementer i s2) {
  håndter det element man er kommet til i s2
  gå videre i s2
}
```