1. Variantide läbivaatamine

I Generaatori mõiste (Java)

Generaator (ehk generaator-klass) on klass, milles leidub (vähemalt) isendimeetod *next*(). Konstruktorile antakse andmed, mis iseloomustavad mingit abstraktset, veel mitte eksisteerivat andmekogumit.

Olgu loodud ehk konstrueeritud generaator-klassi isend, nt nimega *gen*, andmekogumi *D* jaoks. Siis igal järjekordsel pöördumisel *gen.next*() tagastab (annab välja) järjekordse elemendi andmekogumist *D*, juhul kui viimane ei ole veel ammendatud.

Ammendatuse kontrollimiseks kirjeldatakse generaatori klassis tavaliselt ka loogilist tüüpi isendimeetod *hasNext*(), mis tagastab väärtuse `true` parajasti siis, kui vaadeldav andmekogum ei ole veel ammendatud.

Generaator võimaldab mingi andmekogumi elemendikaupa läbivaatamist, seda ilma vajaduseta kõiki elemente eelnevalt valmistada ja mällu salvestada. Näiteks,

```
while(gen.hasNext()){
   // valmistada jrk elemendi väärtus, omistada muutjale e:
   T e =  gen.next();
   // töödelda D element e:
   ...
}
```

Üks üldisematest Java generaatori klassi kujudest on esitatud järgmisel leheküljel.

Selle kasutamine:

```
Generaator<T> gen = new Generaator<T>(<D parameetrid>);

for(T e : gen){ // andmekogumi D iga elemendi e korral
   // töödelda element e:
   ...
}
```

Java generaatori klassi lihtsustatud skeem on toodud ülejärgmisel leheküljel. Ainukeseks avalikuks isendimeetodiks selles on meetod *next*(), mis annab välja (tagastab) järjekordse elemendi andmekogumist (juhul kui viimane ei ole veel ammendatud), või tühiviida `null` ammendatuse tunnusena.

Sellise generaatori (nt nimega `Gen_`)kasutamine:

```
Gen_<T> gen = new Gen_<T>(<andmekogumi parameetrid>);
T e;
while((e = gen.next()) != null){
   // e on andmekogumi järjekordne element, töödelda see:
   ...
}
```

```
import java.util.Iterator;
import java.util.NoSuchElementException;
public class Generaator<T>implements Iterator<T>,Iterable<T>{
   // isendimuutujad:
   protected ... ;
      ...
   Generaator<T>(<andmed hulga D kohta>){ // konstruktor
      //isendimuutujate sätestamine:
         ...
   }//konstruktor

   /////// liideses Iterator ettenähtud meetodid:
   @Override
   public boolean hasNext(){
   // tagastatakse true või false (isendimuutujate põhjal):
      ...
   }//hasNext
   @Override
   public T next(){
      if(hasNext()){
         // tagastada jrk element(isendimuutujate põhjal):
            ...
      }
      throw new NoSuchElementException("Juba ammendatud!");
   }//next
   @Override
   public void remove(){
      throw new UnsupportedOperationException
         ("Eemaldamise tehet ei toetata.");
   }//remove

   /////// liideses Iterable ettenähtud meetod:
   @Override
   public Iterator<int[]> iterator(){
      return this;
   }//iterator

   /////// abimeetodeid (isendimuutujate käitlemiseks):

   protected ...

      ...

}//class
```

```
public class Gen_<T>{ // lihtsustatud skeemiga generaator

   // isendimuutujad:
   protected ...
      ...
   protected boolean ammendatud;    // ammendatuse tunnus

   public Gen_<T>(<andmekogumi parameetrid>){ // konstruktor
      ...
      ammendatud = false;
   }//konstruktor

   public T next(){
      if(ammendatud)
         return null;
      <leida andmekogumi järjekordne element e>
      if(<seda enam ei leidunud>){
         ammendatud = true;
         return null;
      }
      return e;
   }//next

   /////// abimeetodeid (isendimuutujate käitlemiseks):

   protected ...

      ...

}//class
```

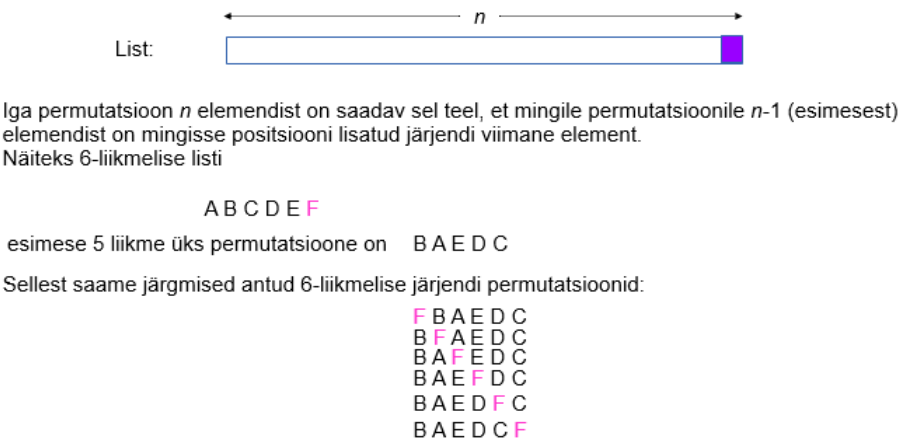
Ülalesitatud lihtsustatud skeemi kohaselt on programmeeritud kõik käesolevas materjalis välja pakutud generaatorid.

Programm *Algarvukaksikud.java* esitab näitegeneraatori antud lõigul asuvate algarvukaksikute (*twin primes*) leidmiseks.

2. Permutatsioonid

Antud listi kõikide permutatsioonide generaator: *Gen_Permut.java*.

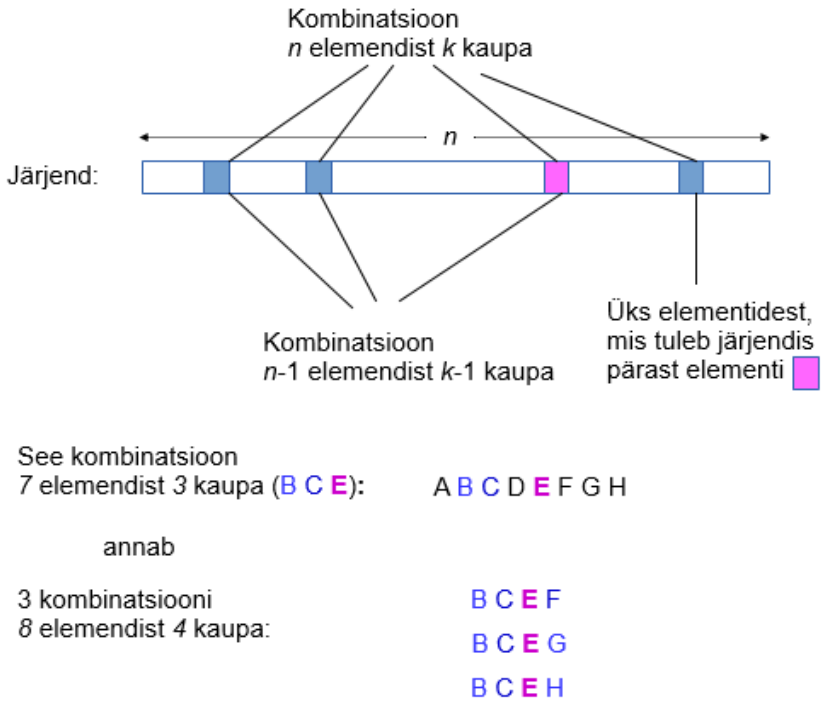
Tegemist on rekursiivse generaatoriga: järjekordne n-elemendilise listi a permutatsioon saadakse sel teel, et listi a viimase elemendita ($n-1$ elemendiga) listi mingisse permutatsiooni mingile kohale lisatakse listi a viimane element, vt joonis 1.



Joonis 1: Permutatsioonide rekursiivselt leidmise idee.

3. Kombinatsioonid

Joonis 2 selgitab võimalust kombinatsioonide leidmiseks; võib olla aluseks kombinatsioonide koostamisel nii rekursiivsel kui ka mitterekursiivsel moel.



Joonis 2: Kombinatsioonide leidmise idee.

I Indeksite kombinatsioonid

Paljudel juhtudel sobib tugineda indeksite kombinatsioonide genereerimisele. Kui on olemas generaator-klass indeksite hulgast $\{0,1,2,\ldots,n-1\}$ antud arvu k kaupa kombinatsioonide genereerimiseks, siis mingi n-elemendilise listi (L) elementide järjekordne kombinatsioon (KL) k kaupa on määratud järjekordse k kaupa kombinatsiooniga (KI) indeksite hulgast $\{0,1,2,\ldots,n-1\}$: KL saadakse valikuna listist L indeksite KI järgi.

Fikseeritud k korral saab vastava indeksite kombinatsiooni genereerida k-kordse tsükliga, näiteks juhul $k=3$ tsükliga

```
Iga i1 korral, i1 = 0 .. n-k:
   Iga i2 korral, i2 = i1+1 .. n-k+1:
     Iga i3 korral, i3 = i2+1 .. n-k+2:
                anda välja [i1, i2, i3]
```

Sellest lähtudes on võimalik välja töötada nii mitterekusiivne kui ka rekursiivne üldine indeksite kombinatsioonide valmistamise meetod.

Rekursiivne meetod:

```
ArrayList<ArrayList<Integer>> tehaIndKomb(int n, int k){
// Antud: n ja k, 0 <= k <= n
// Tulemus: tagastatakse list, mille elementideks on listid -
//          kombinatsioonid k kaupa arvudest 0,1, ..., n-1

   ArrayList<ArrayList<Integer>>tulem =
                          new ArrayList<ArrayList<Integer>>();

   if (k == 0) // baasjuht
      return tulem;

   ArrayList<ArrayList<Integer>> tulem0 =
                                        tehaIndKomb(n-1, k-1);

   if (tulem0.isEmpty())
      tulem0.add(new ArrayList<Integer>());
   for(ArrayList<Integer> x : tulem0){
      int algus = (x.size() >  0)? x.get(x.size()-1) + 1: 0;
      for( int i = algus; i <= n-1; i++ ){
         ArrayList<Integer> xUus = new ArrayList<Integer>(x);
         xUus.add(i);
         tulem.add(xUus);
      }
   }
   return tulem;
}
```

Mitterekursiivse lähenemise näiteks on generaator-klass *Gen_IndKombin.java* indeksite hulgast $\{0,1,2,\ldots,n-1\}$ kombinatsioonide genereerimiseks.

II Kombinatsioonid listi elementidest

Generaator-klass listist antud arvu k kaupa kombinatsioonide leidmiseks: *Gen_Kombin.java*.

Abiks on generaator-klass *Gen_IndKombin.java*: n-elemendilise listi a elementide järjekordse kombinatsiooni k kaupa määrab abi-generaatorist *Gen_Kombin* saadav järjekordne indeksite $0,1,\ldots,n-1$ kombinatsioon k kaupa: listist a valitakse just nende indeksitega elemendid.

4. Alamhulgad

Tähistame: $P(A)$ - hulga A kõigi alamhulkade hulk.

I Alamhulgad rekursiivselt

Alamhulkade genereerimise rekursiivse skeemi aluseks on võrdus

$P(B \cup \{x\}) = P(B) \cup X$,

kus X on hulk, mille iga element on saadud elemendi x lisamisel ühele hulga $P(B)$ elemendile (vt joonis 3).

$$
\begin{aligned}
&A = \{2,-1,3\}\\
&B = \{2,-1\}\\
&A = B \cup \{3\}\\
&\mathrm{P}(B) = \{\{\},\{2\},\{-1\},\{2,-1\}\}\\
&\mathrm{P}(A) = \mathrm{P}(B \cup \{3\}) = \mathrm{P}(B) \cup X =\\
&= \{\{\},\{2\},\{-1\},\{2,-1\}\} \cup \{\{3\},\{2,3\},\{-1,3\},\{2,-1,3\}\} =\\
&= \{\{\},\{2\},\{-1\},\{2,-1\},\{3\},\{2,3\},\{-1,3\},\{2,-1,3\}\}
\end{aligned}
$$

Joonis 3: Näitehulga A alamhulkade hulk rekusiivselt.

Selline skeem on realiseeritud antud hulga kõigi alamhulkade generaatoris *Gen_Ah.java*.

II Alamhulgad mitterekursiivselt

Hulga (A) alamhulkade hulk on selle kõikvõimalike kombinatsioonide ühend,

$$P(A) = \bigcup_{k=0}^{|A|} Komb(A,k)$$

kus $Komb(A,k)$ on hulga A kõigi k kaupa kombinatsioonide hulk (vt joonis 4). Sellest lähtuvalt võib näiteks leida ja töödelda hulga a alamhulgad suuruse (ele-

$$
\begin{aligned}
&\mathrm{P}(\{2,-1,3\}) = \{\{\}\} \cup\\
&\cup \{\{2\},\{-1\},\{3\}\} \cup\\
&\cup \{\{2,-1\},\{2,3\},\{-1,3\}\} \cup\\
&\cup \{\{2,-1,3\}\} =\\
&= \{\{\},\{2\},\{-1\},\{3\},\{2,-1\},\{2,3\},\{-1,3\},\{2,-1,3\}\}
\end{aligned}
$$

Joonis 4: Näitehulga alamhulkade hulk kombinatsioonide ühendina.

mentide arvu) kahanemise järjekorras tsükli täitmise käigus:

```
ArrayList<T> komb;
for(int k = a.size(); k >= 0; k--){
   Gen_Kombin<T> gen = new Gen_Kombin<T>(a, k); // generaator
   while((komb = gen.next()) != null){
      // komb on jrk kombinatsioon a elementidest k kaupa
      <töödelda komb>
   }
}
```

Veel üks mitterekursiivne moodus oleks alamhulgad $P(A)$ valida hulgast A maskide abil. Maskideks on bitijärjendid – arvude $0, 1, \ldots, 2^{|A|-1}$ esitused kahendsüsteemis. (Siin üksikasjalikumalt ei käsitle.)