public interface Enumeration

```
{  boolean hasMoreElements();
   Object nextElement();          }
```

```
public interface Graph { ...
   Enumeration vertices(); ... }

// for hver node i grafen G {gjør noe}

Enumeration En = G.vertices();
while (En.hasMoreElements()) {
   Object o = En.nextElement();
   {gjør noe}

// for hver node i grafen G {gjør noe annet}

// NB! En Enumeration kan brukes kun én gang
// Neste gang man skal traversere samme samling,
// må man hente Enumeration på nytt

Enumeration En = G.vertices();
while (En.hasMoreElements()) {
   Object o = En.nextElement();
   {gjør noe annet}
```

```
public interface Graph { ...
   PosSequence vertices();
...}

// for hver node i grafen G {gjør noe}

PosSequence Sn = G.vertices();
try { Position p = Se.first();
   while (true) {      Object o = p.element();
      {gjør noe}
      p = Se.after(p);
   }
catch(InvalidPos e) { }
```

Fargelegging

Algoritmen farger land med dets `farge()` og da fjerner denne fra mulige farger hos naboland.

```
boolean fargelegg() {
  Enumeration Ve= Kart.vertices(); boolean mulig=true;
  while (Ve.hasMoreElements() && mulig) {
    Land v= (Land)Ve.nextElement();
    if (v.mulig()) {
      int f= v.farge(); v.setFarge(f);
      Enumeration A= Kart.adjacentV(v);
      while (A.hasMoreElements() {
        ((Land)A.nextElement()).remove(f); }
    else mulig=false;
   }
return mulig; }
```

$n = \#$land (noder) og $k = \#$grenser (kanter)

1. $\mathcal{O}(n * k)$ ...? Riktig men for upresist
2. $\mathcal{O}(n + k)$
3. $\mathcal{O}(n + k)$: forutsatt at `vertices`$() = \mathcal{O}(n)$, `adjacentV(v)` $= \mathcal{O}(\#\textit{antall naboer})$, og at begge `Enumeration`-operasjonene er $\mathcal{O}(1)$.
4. $\mathcal{O}(n + k)$ (under samme antakelser som over): Hovedløkken går gjennom hver node en gang (høyst n iterasjoner). For hver node, besøker man hver av dens nabonoder (nabokanter) en gang – tilsammen, over *alle* n iterasjoner, blir det $2 * k$ (hver kant sees på to ganger). Dersom det ikke finnes isolerte noder, gir det $\mathcal{O}(n + k)$, der n kommer fra `Kart.vertices()`.
   (Dersom det finnes isolerte noder, får vi $\mathcal{O}(n' + k)$, der n' er antall isolerte noder.)

Det er ikke opplagt at denne algoritmen alltid klarer å fargelegge et kart – selv om det er mulig. For å sikre at en løsning alltid vil bli funnet, burde vi essensielt prøve alle muligheter. Dvs. når vi kommer til et punkt der det er umulig å farge et land, burde vi 'backtrack'e og prøve en annen farge for det landet som ble farget like før. Er ikke det mulig, burde vi gå enda et steg tilbake og prøve forrige land.

Verstefall kompleksiteten av en slik algoritme blir ikke mindre enn antall muligheter – 4 farger for land 1, 4 for land 2, osv., dvs.

$$\underbrace{4 * 4 * \ldots * 4}_{n} = \mathcal{O}(4^n)$$

.