Toteutusdokumentti

Ohjelman yleisrakenne

Ohjelma koostuu java-koodista sekä sanasto-tiedostosta, joka annetaan xml-muotoisena. Ohjelma suoritetaan ajamalla crossword.crosswordmaker-pakkauksessa oleva Main-metodi. Käyttöliittymän avulla käyttäjä voi valita generoitavan krypton koon sekä aloitussanan. Ohjelman koodi on jaettu kolmeen pakkaukseen:

- crossword.crosswordmaker
    - Pakkauksessa on Main-metodin lisäksi käyttöliittymä.
- crossword.datastructures
    - Itse toteutetut tietorakenteet CustomArrayList ja SearchTree
- crossword.lexicon
    - Xml-muotoisen sanasto-tiedoston käsittely
- crossword.logic
    - Kaikki krypton generointiin tarvittava koodi

Yksikkötestit ovat vastaavissa pakkauksissa.

Pakkauksessa crossword.logic on seuraavat luokat:

- Alignment
    - enum-luokka: kryptolaudalle asetettavien sanojen suuntaus (horisontaalinen tai vertikaalinen)
- BoardOfWords
    - huolehtii kryptolaudan piirtämisestä sekä sanojen piirtämisestä laudalle
- CrossWordMaker
    - jos aloitussanaa ei anneta, hakee satunnaisen sanan aloitussanaksi
    - kutsuu kaikkia niitä metodeita, joita tarvitaan krypton generointiin
- QuickSort
    - itse toteutettu pikajärjestämisalgoritmi WordPosition-olioiden järjestämiseen
- WordFinder
    - Sanojen etsiminen kryptolaudalle eli ohjelman varsinainen ydin on tässä luokassa
- WordPosition
    - malliluokka, tallentaa sanojen sijainnit kryptolaudalla
- WordPositionFinder
    - Etsii valmiiksi piirretyltä kryptolaudalta ne paikat, joihin tulee löytää sana

Saavutetut aika- ja tilavaativuudet

Ohjelman ydin on metodi, joka etsii jokaiseen krypton sanapaikkaan sanan. Tämä on toteutettu rekursion avulla niin, että jokaiseen sanapaikkaan kokeillaan jokaista siihen sopivaa sanaa (varatut kirjaimet huomioiden) kunnes päädytään lopputulokseen, jossa krypton jokainen sanapaikka on täytetty. Pseudokoodina:

positions on globaali muuttuja

```
layWords(index, boardOfWords)
  if index == positions.size
    return boardOfWords
  position = positions[index]
  fittingWords = findWords(position)
  if fittingWords.isEmpty return null
  for each word in fittingWords
    copyOfBoard = copy(boardOfWords)
    copyOfBoard.drawWord(word)
    solution = layWords(index + 1, copyOfBoard)
    if (solution ≠ null) return solution
  return null

findWords(position)
  mask = makeMask(position)
  fittingWords = new ArrayList
  return searchWordTree(mask, root of searchTree, 0, fittingWords)

searchWordTree(mask, node, level, fittingWords)
   if mask.charAt(level) == node.getKey
    if level == maskin pituus
      if nodeen on talletettu sana
        add sana to fittingwords
      return fittingWords
    childNode = node.getChild
    while childnode NOT null
      searchWordTree(mask, childNode, level + 1, fittingwords)
      childNode = childNode.getNext
  return fittingWords
```

Koska kryptolauta kopioidaan jokaisessa rekursiokutsussa, muistissa voi kerrallaan olla puun syvyyden verran kryptolautoja eli niin monta kuin laudalla on sanapaikkoja. Jos merkitään, että s = sanapaikkojen määrä, l = kryptolaudan leveys ja k = laudan korkeus, niin tilavaativuus on $O(s \cdot l \cdot k).$ Kryptolaudan lisäksi rekursiokutsuissa kuljetetaan mukana listaa laudalle jo piirretyistä sanoista, minkä tilavaativuus on $O(s!)$.

Aikavaativuus on hyvin suuri. Aikaa kuluu pääasiassa sopivien sanojen sovittamiseen laudalle.

Sopivien sanojen hakeminen hakupuusta on periaatteessa aikaavievää, mutta käytännössä hyvin nopeaa, koska sanojen määrä on niin pieni verrattuna mahdollisten sanojen määrään (siis jos kaikki mahdolliset kirjainyhdistelmät muodostaisivat sanoja). Hakupuun solmulla voi olla korkeintaan suomalaisten aakkosten verran eli 29 solmua. Jos maskissa ei ole kirjaimia

rajoittamassa haun laajuutta, hakupuussa mennään syvyyssuunnassa korkeintaan maskin pituuden verran solmuja, eli pahimmassa tapauksessa käydään läpi 29^p solmua, missä p = sanan pituus. Käytännössä pahin tapaus ei koskaan toteudu lähimainkaan, koska kaikki kirjaimet eivät esiinny peräkkäin. Haun nopeuttamiseksi eripituisille sanoille on omat hakupuut.

Sanojen sovittaminen laudalle on ohjelman aikaavievin operaatio. Jos jokaiseen sanapaikkaan pystyisi sijoittamaan n määrän sanoja ja kryptoon sijoitettavien sanojen määrää merkittäisiin s:llä, aikavaativuus olisi n^s, mutta käytännössä n pienenee laudalle jo asetettujen sanojen rajoittaessa seuraavaksi lisättävien mahdollisten sanojen määrää. Lisäksi jokaisessa rekursiokutsussa käytettyjen sanojen lista (usedWords) käydään läpi ja siinä olevat sanat etsitään listalta fittingWords. Alussa fittingWords sisältää paljon sanoja, mutta usedWords vain vähän. Pidemmälle mentäessä fittingWords-lista lyhenee ja usedWords-lista kasvaa.

Ohjelman pahimman tapauksen aikavaativuus on siis eksponentiaalinen. Käytännössä ratkaisu löytyy usein huomattavasti pahinta tapausta nopeammin. Käytännössä ohjelman suoritukseen kuluvaan aikaan vaikuttaa paljon sanojen asettelu kryptolaudalle, eli kuinka pitkiä sanat ovat ja kuinka suuri osa sanan kirjaimista risteää toisen sanan kanssa. Sellaisia sanoja on helpoin löytää, joita on paljon suhteessa mahdollisten samanpituisten sanojen määrään. Esimerkiksi viisikirjaimisia sanoja on käyttämässäni sanalistassa (Kotuksen sanalista, johon on lisätty substantiivien monikkomuotoja) 3914 kpl, mikä on 0,019 % mahdollisten viisikirjaimisten sanojen määrästä. Seitsenkirjaimisia sanoja on 7620 kpl, mutta se on vain $4{,}4 \times 10^{-5}$ % mahdollisten seitsenkirjaimisten sanojen määrästä. Paljon kahden kirjaimen mittaisia sanoja sisältävälle laudalle on hyvin vaikea löytää ratkaisua, koska kaksikirjaimisia sanoja on Kotuksen sanalistassa vain 35.

Puutteet ja parannusehdotukset

Kryptolaudan generointi

Lähteet

- Kivinen, Jyrki. Tietorakenteet ja algoritmit -kurssin luentomateriaali. Syksy 2017.
- http://www.java2novice.com/java-sorting-algorithms/quick-sort/ (viitattu 27.2.2018)
- http://www.docjar.com/html/api/java/util/ArrayList.java.html (viitattu 27.2.2018)
- https://coderanch.com/t/607340/java/Implementing-ArrayList-class-manually (viitattu 27.2.2018)
- https://www.emogic.com/notes/crossword_generator_script_algorithm_and_programming (viitattu 27.2.2018)