Variantide läbivaatamine ja kombinatoorika

Järgnevad ülesanded tuleks lahendada rekursiivsete funktsioonide abil taandades ülesande teatavaks arvuks esialgse ülesandega sarnasteks alamülesanneteks. Proovige ülesanne jaotada osadeks (alamülesanneteks) nii, et iga osa jaoks saaks rakendada rekursiivselt sama funktsiooni.

Kombinatsioone, permutatsioone ja muid valikuid teostavaid standardfunktsioone kasutada ei tohi.

Ülesanne 1

1. Kirjutada rekursiivne funktsioon, mis väljastab ekraanile kõik bittvektorid, mille pikkus on võrdne etteantud arvuga.
2. Modifitseerida seda funktsiooni nii, et ekraanile väljastamise asemel funktsioon tagastaks kõikide sobivate bittvektorite hulka. Modifitseeritud funktsioon peaks implementeerima liidest *BitVectorGenerator*.

Ülesanne 2

Implementeerida liidest *SubSetGenerator*.

1. Kirjutada rekursiivne funktsioon *allSubsets*, mis tagastab määratud hulga kõik alamhulgad.
2. Kirjutada rekursiivne funktsioon *allSubsetsOfSizeK*, mis tagastab määratud hulga kõik alamhulgad, kus on täpselt k elementi.

Ülesanne 3

1. Kirjutada rekursiivne funktsioon, mis tagastab listi kõike etteantud listi permutatsioonidega. Funktsioon peaks implementeerima liidest *PermutationGenerator*.
2. Seda funktsiooni katsetades teha kindlaks maksimaalne sõnepikkus, mille korral see funktsioon töötab veel alla 1 sekundi.

Ülesanne 4

Järgmistest ülesannetest valida üks ja lahendada see. Vajalikud andmed genereerida juhuslikult sobivast arvuvahemikust. Valitud ülesande lahendusprogramm peab sisaldama mõistliku arvu teste.

a) *Bobimeeskonna koostamine.* n-liikmelisest treeningrühmast ($n = 10 \pm 2$) tuleb eelolevaks võistluseks välja valida neljase bobi meeskond. Treeningrühma iga liikme kohta on antud nimi, kaal ja naturaalarvuline reiting (mis näitab treenituse taset). Kirjutada programm, mis leiab sellise bobimeeskonna, mille kogukaal ei ületa etteantud väärtust ja mille reitingute summa on võimalikult suur. Realiseerida liides *BobsledTeam*.

b) *Sobivaima töörühma väljavalimine.* Antud on töötajate nimekiri ja sobivusmaatriks (s_{ij}), mille element s_{ij} näitab töötajate i ja j vastastikuse sobivuse taset reaalarvuna lõigul $[0; 1]$. Kirjutada programm, mis leiab võimalikult suure rühma töötajaid, mille korral vastastikuste sobivuste keskmine on suurem kui 0,5. Realiseerida liides *HarmonicWorkgroup*.

c) *Praktikumijuhendajaga kokkuleppel* valida ise üks ülesanne, mille lahendusalgoritm vajab kõikide variantide läbivaatamist (kas siis eksponentsiaal- või faktoriaalkeerukusega algoritmi abil). Nõutav on, et see ülesanne teeks midagi „kasulikku“.

Lihtsamad puud

Puu on hierarhiline andmestruktuur, mis koosneb **tippudest** (ehk sõlmedest; inglise keeles *nodes* või *vertices*), ja nende vahelistest **kaartest** *(edges)*. Kaared ühendavad tippu tema **vanemaga** *(parent)* ning **lastega** *(children)*. Igas mittetriviaalses[1] puus eksisteerib üks teistest erinev tipp, mida nimetatakse **juureks** *(root)*. Igal tipul on täpselt üks vanem, v.a. juurtipp, millel ei ole vanemat. Tippe, millel ei ole lapsi, nimetatakse **lehtedeks** (*leafs*). Täpsemalt puudest ja nende tüüpidest räägitakse hilisemates loengutes.

Selles praktikumis me vaatleme ühte lihtsamat puutüübi — **kahendpuud**. Kahendpuu tippudel võib olla maksimaalselt kaks last, mille järjekord on tavaliselt tähtis (on *vasak* ja *parem* laps). Niisuguse puu tippu realisatsioon võib olla järgmine (täielik realisatsioon on toodud materjalide lõpus):

```
public class BinaryTreeVertex<Data>{
   private BinaryTreeVertex<Data> parent = null;
   private BinaryTreeVertex<Data> leftVertex = null;
   private BinaryTreeVertex<Data> rightVertex = null;

   private Data data = null;
}
```

Järgmised ülesanded eeldavad seda, et etteantud puu on esitatud kasutades seda realisatsiooni.

Ülesanne 5
Implementeerida liidest *BinaryTreeMeasurer*.

1. Kirjutada funktsioon *measureTreeHeight*, mis tagastab etteantud puu kõrgust. Kui etteantud juurtipul ei ole ühtegi last, see funktsioon peaks tagastama 0.
2. Kirjutada funktsioon *countNodes*, mis tagastab etteantud puu tippude arvu (k.a. juur).
3. Kirjutada funktsioon *countLeaves*, mis tagastab etteantud puu lehtede arvu.

Ülesanne 6
Implementeerida liidest *BinaryTreeDataCounter*. Selleks kirjutada funktsioon *countData*, mis tagastab mitu tippu etteantud puust sisaldab funktsiooni argumendina antud väärtust.

[1] Triviaalne puu on niisugune puu, mis ei sisalda ühtegi tippu ega kaart, ehk tühi puu.