Toteutusdokumentti

Aineopintojen harjoitustyö: Tietorakenteet ja algoritmit (alkukesä)

Sami Korhonen
014021868
sami.korhonen@helsinki.fi

Tietojenkäsittelytieteen laitos
Helsingin yliopisto
23. kesäkuuta 2014

Ohjelman yleisrakenne

App ja Pakkaaja

App

Tämä luokka sisältää vain main-metodin, joka käynnistää ohjelman suorituksen.

Pakkaaja

Tämä luokka pyörittää ohjelman toiminnallisuutta ja sisältää algoritmin tärkeimmät metodit. Tämän toiminnallisuudesta on yksinkertaistettu versio määrittelydokumentissa. Ohjelman toiminnallisuus yksityiskohtaisemmalla tasolla tuskin auttaa kokonaisuuden hahmottamista.

Pakkaus: domain

Tästä luokasta löytyy ohjelmassa käytettävien olioiden luokat, joista oleennaisimpia lienevät Kontti, Laatikko, Palkki, Tilapalkki. Myös ohjelman tilaa kuvaava Tila löytyy täältä. Se sisältää Pakkaajan ohella muutamia oleennaisia metodeja, jotka liittyvät tilan ylläpitoon.

Metodit

Suuri osa tämän pakkauksen luokista on lähinnä tietueita, joten metodit ovat lähinnä konstruktoreita, gettereita ja settereitä. Monet luokista toteuttavat rajapinnan Sarmio, joka on yksinkertainen suorakulmainen särmiö

Pakkaus : structures

Tästä pakkauksesta löytyy tietorakenteet: lista, ja jono. Myös muita tietorakenteita, kuten LinkedList ja Map tulin toteuttaneeksi, mutta niistä ei ollut hyötyä tässä työssä.

Pakkaus: tools

Tästä pakkauksesta löytyy käyttäjänsyötteistä vastaava Console sekä tiedostonkäsittelyä varten toteutettu FileHandler.

Pakkaus: testing

Tämän pakkauksen tärkein luokka on Testaus, joka sisältää erilaisten laatikkolistojen generaattoreita ja muuta tarpeellista suorituskykytestausta varten. Testisetti ja SetinOminaisuudet ovat lähinnä tallentamassa muutamia parametreja elämän helpottamiseksi.

Suorituskyky ja tulokset

Tarkastellaan pseudokoodin avulla algoritmin aikavaativuutta pahimmassa tapauksessa.

```
function pakkaaKontti(kontti, laatikot)
    pakkaussuunnitelma = new PakkausSuunnitelma
    tila = new Tila
    while (tila.tilapalkkipino not empty)
         tilapalkki = tila.tilapalkkiPino.pop()
        palkki = haeParasPalkkiLaatikoista(tilapalkki, tila.vapaatLaatikot)
        if (palkki not null)
         tila.paivita(palkki, tilapalkki)
    endwhile
return pakkaussuunnitelma
```

Merkitään tässä laatikoiden lukumäärää n:llä, ja laatikkotyyppienmäärää m:llä. Metodin pakkaaKontti aikavaativuus pahimmassa tilanteessa riippuu while-silmukan iteraatioista, sekä metodin haeParasPalkkiLaatikoista aikavaativuudesta. Pahimmassa tapauksessa pakattaisiin laatikko kerrallaan, jolloin while-silmukka tulisi suorittaa luokkaa n kertaa, mahdollisesti $3n$.

```
function haeParasPalkki(tilapalkki, vapaatLaatikot)
```

```
    paras = new Palkki
    for each laatikko in vapaatLaatikot
        Palkki p = valitseOrientaatio(tilapalkki, laatikko)
        if (parempi(p, paras))
            paras = p
    end for
return paras
```

Metodin haeParasPalkki aikavaativuus määräytyy for-silmukan perusteella. Sen aikavaativuus näyttää pseudokoodissa olevan O(n), mutta toteutuksessani se on O(m), sillä tässä käydään läpi vain erityyppiset laatikot, ei jokaista yksittäistä samanlaista laatikkoa. Tällöin metodin aikavaativuus on siis O(m).

```
function valitseOrientaatio(tilapalkki, laatikko)
    paras = new Palkki
    for i = 0 to 6
        laatikko.asetaOrientaatio(i) // valitaan orientaatio
        // luodaan nx, ny ja nz
        if (nx*x*ny*y*nz*z > suurinTilavuus)
            suurinTilavuus = nx*x*ny*y*nz*z
            paras = new Palkki(sijainti, laatikko, nx, ny, nz)
    end for
return paras
```

Metodin valitseOrientaatio aikavaativuus määräytyy for-silmukan perusteella, mutta se on vakioaikainen. Pseudokoodista poiketen toteutukseni joutuu aiemmasta oikaisusta johtuen käymään kerran käymään palkkiin mahtuvat laatikot läpi. Tämä lisää aikavaativuuteen osan $nx * ny * nz$, joka on pahimmassa tapauksessa n. Nämä aikavaativuudet yhdessä tuottanevat aikavaativuuden O(mn^2).
Algoritmin todellisesta suorituksesta on esitetty mittaustuloksia ja kaavioita testausdokumentissa.

Puutteet ja parannusehdotukset

Ohjelman toiminnallisuus oli varsin työlästä toteuttaa ja saada toimimaan edes jotakuinkin tehokkaasti. Tehokkuuden osalta varmasti löytyisi vielä parannettavaa ihan kooditasollakin, mutta muutamia oikoteitä on suunnittelussa jouduttu ottamaan ajan puutteen takia. Alun perin oli tarkoitus lisätä hakuun rajoittavia parametreja ohjelman nopeuttamiseksi. Sopivan palkin etsinnässä olisi voinut käyttää aikarajoitusta, tai algoritmi voisi valita palkin, joka on riittävän tyydyttävä. Nämä olisivat lisänneet suorituskykytestauksen työurakkaa muhkeasti suuremmaksi, ja pääsääntöisesti ohjelma tuntuu riittävän tehokkaalta sellaisenaan. Erilaisia tietorakenteita olisi ollut hyvä kokeilla testaustasolla, mutta ohjelman bugien korjaus on vienyt tähän tarkoitetun ajan. Vielä esiintyy ajoittain muistin ylivuotoa rankimpia testejä ajettaessa. Tämä tosin riippuu hieman tietokoneestakin.

Algoritmi voisi olla myös hienostuneempi määrittelemään milloin mikäkin palkki on toista parempi. Nykyisessä versiossa tilavampi palkki luokitellaan autuaasti pienempää paremmaksi, vaikka tietyissä tilanteissa jäljelle jäävä tila on liian pieni täytettäväksi, mutta kuitenkin merkittävän suuri. Tämä algoritmi on tosin optimoitu kohtalaisen homogenisille laatikkosarjoille. Ohjelman olisi myös mahdollista suorittaa kaksi erityylistä ratkaisua riippuen annetuista laatikoista, joista toinen olisi optimoitu homogenisille sarjoille ja toinen heterogenisille [1].

3D-grafiikan rakentaminen jäi niinikään kesken ja se poistettiin lopputuloksesta. Tämäkin osoittautui hieman hankalemmaksi puutteellisten materiaalien vuoksi. Lopputuloksen tarkastelua varten kuva olisi yksinkertainen ja varsin hyödyllinen. Tällä hetkellä on hankala varmistaa ohjelman toiminen äärirajoillaan, sillä kolmiulotteisten kappaleiden koot ja sijainnit on kohtalaisen hankala ihmisen hahmottaa rivistä numeroita ja kirjaimia. Näitä varten olen tehnyt yksikkötestejä, mutta hankalimpien kohtien testaaminen lienee hieman vajaavaista, eikä integraatiotestejä ole tehty lainkaan.

Viitteet

[1] Tobias Fanslau, Andreas Bortfeldt, 2008
A Tree Search Algorithm for Solving the Container Loading Problem
http://www.fernuni-hagen.de/wirtschaftswissenschaft/download/beitraege/db426.pdf
viitattu 22. kesäkuuta 2014