Tietorakenteiden harjoitustyö, loppukesä 2012

Paula Lehtola 013032638

23. elokuuta 2012

Helsingin yliopisto
Tietojenkäsittelytieteen laitos
Ohjaaja Kristiina Paloheimo

Sisältö

1 Määrittelydokumentti

1.1 Toteutettavat algoritmit ja tietorakenteet

Toteutan N-pelin (N-puzzle) ratkaisijan IDA*-algoritmilla. N-peli on tunnettu esimerkki käytettäessä hakualgoritmeja joihin liittyy jokin heuristiikka[1]. Aikomukseni on testata ohjelmaa ainakin 3x3- ja 4x4 -pelilaudoilla.

Käyttämäni IDA*-hakualgoritmi on muunnos A*-algoritmista. IDA* käyttää iteratiivista syvyyshakua apunaan, eikä siksi kuluta muistia niin paljon kuin A*.[2]

Toteutan aputietorakenteena pinon, jota iteratiivinen syvyyshaku käyttää pitämään muistissa seuraavia pelitilanteita.

1.2 Ratkaistava ongelma ja toteutustavan perustelu

1.2.1 Iteratiivinen syvyyshaku ja IDA*-algoritmi

Alun perin ajattelin käyttää A*-algoritmia, mutta koska N-pelissä pelin tilanteiden määrä kasvaa nopeasti suureksi, vähemmän muistia kuluttava IDA* oli parempi ratkaisu. Iteratiivisen syvyyshaun ilman heuristiikkaa otin mukaan vertailun pohjaksi.

1.2.2 N-pelin ratkaisemisesta

N-Peli (15-peli, 8-peli jne) on yhden pelaajan peli, jossa on neliön muotoinen pelilauta jolla sijaitsevat sekoitettuina numerot 1-N, sekä yksi tyhjä ruutu. Tarkoituksena on saada pelilauta numerojärjestykseen käyttäen apuna tyhjää ruutua.

Yritän saada aikaan sellaisen ratkaisuohjelman, joka ratkaisee N-pelin mahdollisimman vähillä siirroilla mahdollisimman nopeasti. Aiheesta on olemassa paljon materiaalia ennestään, ja päädyin aiheen tutkimisen ja suositusten perusteella valitsemaan sellaisen algoritmin ja sellaiset heuristiikat, joilla tähän ongelmaan on olemassa tehokas ratkaisu.

Kaikki N-pelin sekoitukset eivät ole ratkaistavissa (lisää lähde). Tämä ei ole kuitenkaan ongelma työssäni, sillä toteuttamani N-peli muodostaa ainoastaan ratkaistavissa olevia alkutilanteita käyttämällä sekoitukseen pelin sallimia siirtoja.

1.2.3 Heuristiikka

Toteutan haun työssäni kolmella eri tavalla:

- Iteratiivinen syvyyshaku ilman heuristiikkaa

[1] http://en.wikipedia.org/wiki/Fifteen_puzzle
[2] http://en.wikipedia.org/wiki/IDA*

- Manhattan Distance: Heuristiikka, joka laskee etäisyyden tietyn palikan nykyisen sijainnin ja halutun sijainnin välillä koordinaattien erotuksen itseisarvona.[3]

- Linear Conflict-muunnelma: Kyseessä on Manhattan Distance -heuristiikan parannus, jossa otetaan huomioon tilanne, jossa kaksi palikkaa on oikealla lopullisella rivillään, mutta "väärinpäin"; esim. 3 on 4:n oikealla puolella. Tällainen tilanne tuottaa reittiarvioon ainakin +2 askelta.

1.3 Syötteet ja niiden käyttäminen

Ohjelman syötteenä toimii N-pelin pelitilanne, eli sekoitettu pelilauta, joka on tarkoitus saada järjestykseen käyttämällä pelin sallimia siirtoja mahdollisimman vähäinen määrä. Pelilauta syötetään algoritmille int-taulukkona, jossa tyhjää ruutua on merkitty esim. nollalla. Alussa ohjelma tarvitsee sekä sekoitetun alkutilanteen että tavoitetilanteen jossa lauta on järjestyksessä.

Ajatuksenani olisi saada ohjelma sellaiseen muotoon, että graafisen käyttöliittymän ja ratkaisualgoritmin saa kytkettyä toisiinsa. Tällöin käyttäjän olisi mahdollista syöttää ratkaistavan pelilaudan koko annettujen (järkevien) vaihtoehtojen puitteissa. N-pelin pelilogiikka ja käyttöliittymä ovat valmiina olemassa Ohjelmoinnin harjoitustyön jäljiltä.

1.4 Tavoitteena olevat aika- ja tilavaativuudet

IDA*-algoritmin aikavaativuus riippuu käytetystä heuristiikasta. Toistaiseksi en onnistunut löytämään IDA*-algoritmin aikavaativuudesta niin yksiselitteistä tietoa, että olisin sen kunnolla ymmärtänyt. Polynomisesta aikavaativuudesta ainakin puhuttiin jossain. Tavoitteet aika- ja tilavaativuuden suhteen tarkentuvat siinä vaiheessa kun alan oikeasti ymmärtää, mitä olen tekemässä.

1.5 Lähteet

- http://heuristicswiki.wikispaces.com/
- sara2000.unl.edu/Korf-slides.ppt

2 Toteutusdokumentti

2.1 Ohjelman yleisrakenne

Ohjelma on yhdistelmä tälle kurssille rakennettua IDA*-hakua tietorakenteineen, sekä Ohjelmoinnin harjoitustyönä toteuttamaani N-peliä, jolla on graafinen käyttöliittymä.

[3] http://en.wikipedia.org/wiki/Manhattan_distance

2.1.1 IdaStar-pakkaus

Varsinainen hakutoiminto sisältyy tähän pakkaukseen.

Haku on toteutettu niin, että sen voi suorittaa kolmella tavalla: iteratiivisena syvyyshakuna ilman heuristiikkaa sekä Manhattan Distance -heuristiikalla ja Linear Conflict -heuristiikalla.

2.1.2 Tietorakenteet-pakkaus

Tämä pakkaus sisältää kaksi erilaista pino-tietorakenteen toteutusta. Toinen on tehty ns. linkitettynä, ja toinen pohjautuu taulukkoon.

2.1.3 Käyttöliittymä- ja sovelluslogiikkapakkaukset

Nämä pakkaukset sisältävät Ohjelmoinnin harjoitustyönä toteuttamani N-pelin. Käsittelen näiden pakkausten sisältöä tässä dokumentissa vain niitä osin kuin se on Tietorakenteiden harjoitustyöni kannalta relevanttia. Haku käyttää sovelluslogiikka-pakkauksen Pelitapahtuma-luokkaa, jolta se saa tiedot pelin alkutilanteesta.

2.2 Saavutetut aika- ja tilavaativuudet

2.3 Suorituskyky- ja O-analyysivertailu

2.4 Puutteet ja parannusehdotukset

2.5 Lähteet

3 Käyttöohje

3.1 Ohjelman suorittaminen ja toiminnallisuuksien käyttö

3.2 Ohjelman hyväksymät syötteet

3.3 Missä kansiossa on jar ja ajamiseen tarvittavat testitiedostot

4 Testausdokumentti

4.1 Haku-luokan testaaminen

- Yksinkertainen virheiden etsintä: ohjelman ajaminen läpi ilman että se tulostaa tai palauttaa mitään, jotta näkee tuleeko virheilmoituksia. Jos tuloksena on esimerkiksi StackOverflowError, virheen pääsee jäljittämään virheilmoituksen linkistä.

- depthFirstSearch - metodin testausta tein suoraan pääohjelmassa seuraavasti: Lisäsin aputulosteita dfs-metodin sisään nähdäkseni tapahtuuko rekursiokutsuja oikea määrä. Tällöin metodi tulosti taulukkona nykyisen

pelitilanteen jokaisen kutsun jälkeen, sekä sen lapsen. Tästä lapsesta tulee uusi parametri-node. Tulosteessa lapsen taulukkoarvot oli merkitty tähdellä, jotta se erottuu parametrina saadusta taulukosta. Myöhemmin olen testannut samaa metodia myös ilman pelitilanteen tulostamista, printtaamalla pelkän tähden aina kun kyseistä metodia kutsutaan.

- Testisyötteitä: 2x2-pelilauta, jossa tarvitsi tehdä yksi tai kaksi siirtoa; esim. 1, 2, -1, 3. 3x3-pelilauta pienellä sekoitusmäärällä (10), isommalla sekoitusmäärällä (1000).
- Debugger: Olen käyttänyt NetBeansin debuggeria virheiden jäljittämisen lisäksi myös testaamiseen.
- JUnit-testit: Testiluokassa hakuTest.java on metodeita jotka testaavat hakua eri kokoisilla pelilaudoilla, eri sekoitusmäärillä ja sekä heuristiikalla että ilman. Testeille on asetettu aikarajat, jotteivät ne jäisi isoilla syötteillä pyörimään loputtomasti.

4.2 Manhattan-luokan testaaminen

Manhattan-luokkaa yksinään on testattu JUnit-testeillä. Lisäksi laskeH-metodin toiminta tulee luonnollisesti testattua samalla kun hakua heuristiikalla testataan.

4.3 LinkitettyPino- ja TaulukkoPino-luokkien testaaminen

Näiden luokkien testit ovat JUnit-testejä, jotka löytyvät testipakkauksen LinkitettyPinoTest.java- ja TaulukkoPinoTest.java-tiedostosta. Luokkien testit ovat enimmäkseen keskenään samanlaisia.