Toteutusdokumentti

Labyrintin generoija ja –ratkoja

Tietorakenteet ja algoritmit harjoitustyö

Juri Kuronen

Kesä 2014

Sisältö

1. Ohjelman yleisrakenne

Labyrintti-luokka on ohjelman runko. Labyrintti-luokka sisältää *korkeus x leveys* -kokoisen byte-array:n, johon on tallennettu tieto labyrintin soluista lähtevistä kaarista. Luokka sisältää monia metodeja tiedon lukemiseen ja päivittämiseen tästä byte-array:sta. Lisäksi Labyrintti-luokan vastuulla on käynnistää Labyrintti-luokkaan liitetyt generointi- ja ratkaisualgoritmit oikein.

LabyrinthGenerator-luokka on labyrintin generoijien yliluokka. Jokainen aliluokka toteuttaa generateLabyrinth()-metodin. Labyrintin generoijat täyttävät (eli generoivat satunnaisesti) Labyrintti-luokan byte-array:n. Labyrintin generoijia ovat Kruskalin algoritmi, Primin algoritmi ja Rekursiivinen peruuttava haku.

LabyrinthSolver-luokka on labyrintin ratkojien yliluokka. Jokainen aliluokka toteuttaa solveLabyrinth()-metodin. Labyrintin ratkojia ovat ”Oikean käden sääntö”-algoritmi, leveys- ja syvyyssuuntainenhaku ja A*-hakualgoritmi. Yliluokan avulla voi hakea ratkaisun jälkeen tietoja ratkaisusta. Yliluokka sisältää mm. maaliin vievän reitin hakemiseen aliluokkien tekemästä työstä.

Toteutetut tietorakenteet ovat omina luokkina. Kaikille tietotyypeille on toteutettu lista, pino sekä jono. Prioriteettikeko on toteutettu A*-hakualgoritmia varten. SetElement ja TreeNode ovat spesifisempiä aputietorakenteita, joista ensimmäistä käyttää Kruskalin algoritmi, ja toista labyrintin ratkojat reitinhakuun.

Ohjelmaa käytetään graafisen käyttöliittymän avulla, joka avautuu ohjelman käynnistyessä. Graafisen käyttöliittymällä voi asettaa labyrintin koon sekä valita generoivan- että ratkovan algoritmin. Komentoriviargumenteilla voi ohittaa ohjelman tavallisen käynnistyksen, ja käynnistää sen sijaan suorituskykytestit.

Suorituskykytestaukset tapahtuvat RunTimeTesting-luokan kautta. Tulokset raportoidaan tekstitiedostoton.

2. Saavutetut aikavaativuudet

Seuraavalla sivulla on kasattu taulukkoon generointi ja –ratkoja-algoritmien keskimääräisiä suoritusaikoja. Kaikilla ko'oilla suoritettu algoritmi n = 50 kertaa ja laskettu keskiarvo. Generoivan algoritmin alla on keskimääräinen generointiaika. Ratkovien algoritmien tieto on muodossa "keskimäärinen ratkomisaika" / "keskimääräisesti vierailtujen solujen määrä".

Generointialgoritmien suoritusaikojen analysointi.

Kun jaamme kuluneen ajan labyrintin koolla (eli solujen määrällä), saamme seuraavat tulokset:

- Primin algoritmin suoritusaika on näin ko'oilla 25^2, 50^2, 100^2, 200^2 ja 300^2, n. 0,59; 0,52; 0;53; 0,57 ja 0,58 µs per solu. Näyttäisi saavuttavan O(|V|) ajan.
- Rekursiivisella peruuttavalla haulla n. 0,5; 0,39; 0,42; 0,51 ja 0,53 µs per solu. Näyttäisi myös saavuttavan O(|V|) ajan.
- Kruskalin algoritmilla huomataan taas, että kyseiset luvut ovat n. 0,66; 0,75; 0,82; 1,16 ja 1,94 µs per solu. Tässä tuntuisi siis olevan pieni logaritminen kasvu lisäksi, kuten määrittelydokumentissa on analysoitu. Näyttäisi siis saavuttavan O(|V| log |V|) ajan.

Ratkoja-algoritmien suoritusaikojen analysointi.

Kaikkien ratkoja-algoritmien on analysoitu olevan O(|V|)-ajallisia. Tehkäämme sama toimenpide, kuin yllä. (Luvut µs per solu.)

- Satunnaistettu syvyysuuntainen haku. 0,13; 0,12; 0,13; 0,14 ja 0,12. Saavuttaa O(|V|) ajan.
- Satunnaistettu leveyssuuntainen haku. 0,22; 0,22; 1, 0,25 ja 0,28. Saavuttaa O(|V|) ajan.
- A*-haku. 0,15; 0,16; 0,53; 0,13 ja 0,13. Saavuttaa O(|V|) ajan.
- "Oikean käden sääntö". 0,08; 0,09; 0,34; 0,09 ja 0,09; Saavuttaa O(|V|) ajan.

Kaikki ratkoja-algoritmit saavuttavat O(|V|) ajan.

Ratkoja-algoritmien keskimäärin tekemä työ, generointialgoritmia kohden.

Summing up	Prim	Kruskal	Backtracker
Randomized DFS	53 %	55 %	63 %
Randomized BFS	99 %	84 %	57 %
A* search	37 %	60 %	53 %
Wall follower	52 %	54 %	61 %

Isoja eroja on mm. A*-haun ja satunnaistetun leveyssuuntaisen haun välillä Primin algoritmin generoimissa labyrinteissa. Kruskalin algoritmin generoimassa labyrintissa kuitenkin A*-haku häviää (työmäärässä) yksinkertaiselle "Oikean käden sääntö" –algoritmille.

10 x 10	Prim's algorithm 0,227 ms		Kruskal's algorithm 0,132 ms		Recursive backtracker 0,060 ms	
Randomized DFS	0,085 ms	60 %	0,018 ms	59 %	0,059 ms	65 %
Randomized BFS	0,083 ms	98 %	0,022 ms	79 %	0,018 ms	66 %
A* search	0,104 ms	54 %	0,019 ms	51 %	0,062 ms	58 %
Wall follower	0,022 ms	61 %	0,011 ms	60 %	0,009 ms	65 %
Wall follower (optimized)	0,020 ms	61 %	0,009 ms	60 %	0,007 ms	65 %
25 x 25	Prim's algorithm 0,367 ms		Kruskal's algorithm 0,411 ms		Recursive backtracker 0,312 ms	
Randomized DFS	0,079 ms	52 %	0,129 ms	50 %	0,101 ms	66 %
Randomized BFS	0,137 ms	99 %	0,126 ms	84 %	0,103 ms	63 %
A* search	0,093 ms	42 %	0,111 ms	55 %	0,086 ms	57 %
Wall follower	0,062 ms	53 %	0,064 ms	56 %	0,058 ms	68 %
Wall follower (optimized)	0,053 ms	53 %	0,055 ms	56 %	0,049 ms	68 %
50 x 50	Prim's algorithm 1,304 ms		Kruskal's algorithm 1,869 ms		Recursive backtracker 0,963 ms	
Randomized DFS	0,291 ms	51 %	0,290 ms	51 %	0,354 ms	62 %
Randomized BFS	0,542 ms	99 %	0,465 ms	84 %	0,287 ms	56 %
A* search	0,408 ms	40 %	0,491 ms	58 %	0,314 ms	51 %
Wall follower	0,241 ms	51 %	0,258 ms	53 %	0,213 ms	60 %
Wall follower (optimized)	0,215 ms	51 %	0,223 ms	53 %	0,183 ms	60 %
100 x100	Prim's algorithm 5,289 ms		Kruskal's algorithm 8,195 ms		Recursive backtracker 4,179 ms	
Randomized DFS	1,292 ms	50 %	1,353 ms	56 %	1,493 ms	62 %
Randomized BFS	2,510 ms	99 %	1,987 ms	85 %	1,246 ms	55 %
A* search	1,330 ms	29 %	2,104 ms	59 %	1,354 ms	53 %
Wall follower	0,990 ms	49 %	1,002 ms	50 %	0,903 ms	60 %
Wall follower (optimized)	0,851 ms	49 %	0,853 ms	50 %	0,776 ms	60 %
200 x 200	Prim's algorithm 22,963 ms		Kruskal's algorithm 46,519 ms		Recursive backtracker 20,310 ms	
Randomized DFS	5,633 ms	57 %	5,694 ms	56 %	6,255 ms	63 %
Randomized BFS	9,929 ms	99 %	8,708 ms	85 %	4,947 ms	54 %
A* search	5,047 ms	28 %	10,616 ms	67 %	5,701 ms	53 %
Wall follower	3,890 ms	49 %	4,189 ms	54 %	3,734 ms	60 %
Wall follower (optimized)	3,408 ms	49 %	3,731 ms	54 %	3,198 ms	60 %
300 x 300	Prim's algorithm 51,984 ms		Kruskal's algorithm 174,779 ms		Recursive backtracker 47,855 ms	
Randomized DFS	11,153 ms	50 %	20,390 ms	55 %	14,321 ms	62 %
Randomized BFS	25,546 ms	99 %	25,970 ms	88 %	11,684 ms	49 %
A* search	11,806 ms	27 %	25,651 ms	70 %	11,959 ms	48 %
Wall follower	9,116 ms	51 %	9,565 ms	52 %	7,835 ms	55 %
Wall follower (optimized)	7,868 ms	51 %	8,185 ms	52 %	6,696 ms	55 %

Suorituskykytestausta, keskimääräiset tulokset kun n = 50.

3. Saavutetut tilavaativuudet

Analysoidaan tilavaativuuksia pseudokoodista.

Generointialgoritmien tilavaativuuden analysointi.

Kruskalin algoritmi

1. initialisoidaan kaari- ja joukkotaulukot.
2. while true
    i. arvo satunnainen solu, josta kaari
    ii. yhdistä kaaren erottamat joukot jos mahdollista, jolloin tee labyrinttiin kaari.
    iii. poistele arvotun solun turhia kaaria

Kohta 1 täyttää joukko- ja kaaritaulukot, labyrintin kokoisia, mutta mikään muu ei vaadi tilaa, siis O(|V|) niin kuin on analysoitukin.

Primin algoritmi

1. initialisoidaan visited-taulukko ja Primin algoritmin vaatima lista ¼ labyrintin kokoiseksi.
2. while true
    i. arvo satunnainen solu labyrintin reunojen viereltä
    ii. lisää solu satunnaista reittiä pitkin labyrinttiin
    iii. lisää arvotun solun naapurit listaan

Kohta 1 täyttää visited-taulukon ja listan, labyrintin kokoisia. Kohta 2.iii saattaa amortisoidusti kasvattaa listan kokoa, mutta se ei ikinä voi kasvaa labyrinttia isommaksi. Siis O(|V|) niin kuin on analysoitukin.

Rekursiivinen peruuttava haku

1. initialisoidaan visited-taulukko ja luodaan pino.
2. while true
    i. poppaa pinon päällimmäinen alkio käsittelyyn
    ii. hae alkion vierailemattomat naapurit
    iii. lisää satunnainen naapuri pinoon, ja lisää reitti alkion ja naapurin välille labyrinttiin

Kohta 1 täyttää visited-taulukon, labyrintin kokoisen, ja luo pinon. Kohta 2.iii saattaa tuoda pinon koon labyrintin kokoiseksi. Siis O(|V|) niin kuin on analysoitukin.

Ratkoja-algoritmien tilavaativuuden analysointi.

”Oikean käden sääntö” –algoritmi

Optimoitu versio ei käytä tilaa. O(1). Optimoimaton versio käyttää visited-taulukkoa, labyrintin kokoista, O(|V|).

Satunnaistettu leveyssuuntainen haku

Käyttää visited-taulukkoa, labyrintin kokoista, ja jonoa, joka voi kasvaa O(c|V|) kokoiseksi. Siis O(|V|) .

Satunnaistettu syvyyssuuntainen haku

Käyttää visited-taulukkoa, labyrintin kokoista, ja pinoa, joka voi kasvaa O(|V|) kokoiseksi. Siis O(|V|) .

A*-hakualgoritmi

Käyttää visited-taulukkoa, labyrintin kokoista, ja prioriteettikekoa, joka voi kasvaa O(c|V|) kokoiseksi. Siis O(|V|) .

Labyrinttiluokan reitinhakualgoritmi

Lukee labyrintin kokoista visited-taulukkoa, käyttää pinoa, joka voi kasvaa O(c|V|) kokoiseksi. Siis O(|V|) .

4. Työn mahdolliset puutteet ja parannusehdotukset

Ajan puutteen vuoksi – ja ehkä siitäkin johtuen, ettei 4 opintopisteen kurssilla voi ihan kaikkea tehdäkään – jäivät alkuperäiset oletukset labyrintista voimaan enkä tällä kertaa toteuttanut lisäsäätömahdollisuuksia generointiin ja labyrinttiin liittyen.

Sain kuitenkin mielestäni tehtyä varsin hyvän rungon labyrintin generointi ja –ratkoja-algoritmeille, johon on helppo lähteä kehittelemään lisäosia, esimerkiksi monimutkaisempia labyrintteja ja algoritmeja. Työssä on myös ihan hyvin määrin algoritmeja sekä ratkojille että generoijille.

Suorituskykytestit jäivät laittamatta gui:hin mukaan, ne täytyy ajaa komentorivin kautta, ja testitulokset kirjoitetaan tiedostoon.