Toteutusdokumentti

1. Ohjelman yleisrakenne

Olen käyttänyt ohjelmassa Model – View – Controller suunnittelumallia. Logiikkaan liittyvät luokat sijaitsevat siis controller pakkauksessa. Matriisi on toteutettu luokassa Matrix, ja kaikki matriisioperaatiot on toteutettu luokassa MatrixMath. Suorituskykytestaus on toteutettu performance_test pakkauksen luokassa Test. Ohjelman pääluokka Main sijaitsee pakkauksessa main.

2. Aika- ja tilavaativuudet

Seuraavaksi tutkin tutkittujen algortimien aika- ja tilavaativuudet.

2.1 Yhteen- ja vähennyslasku

Syöte: Kaksi m × n matriisia A ja B

Tuloste: m × n matriisi C

Yhteen-(vähennys-)laskun pseudokoodi:

```
for (i=1 to m)
        for(j=1 to n)
                C[i][j] = A[i][j] +(-) B[i][j]
return C
```

Algortimissa on kaksi sisäkkäistä for-silmukkaa. Ulompi silmukka suoritetaan m kertaa, ja sisempi n kertaa. Sisemmän silmukan runko on vakioaikainen operaatio. Se suoritetaan yhteensä m*n kertaa, joten algoritmin aikavaativuus on O(mn).

Algoritmissa luodaan matriisi, jonka tilavaativuus on O(mn), muiden apumuuttujien tilavaativuus on O(1), joten algoritmin tilavaativuus on O(mn).

2.2 Transpoosi

Syöte: m × n matriisia A

Tuloste: n × m matriisi B

Transpoosin pseudokoodi:

```
for (i=1 to n)
        for (j=1 to m)
                B[i][j] = C[j][i]
return B
```

Transpoosin aika- ja tilavaativuusanalyysiin voi soveltaa samaa logiikkaa kuten yhteen- ja vähennyslaskuihin, joten transpoosin aika- ja tilavaativuus on O(mn).

2.3 Determinantti

Syöte: n × n matriisi A

Tuloste: reaaliluku (double)

Pseudokoodi:

```
det(Matrix A){
det = 0
if (n == 1)
        return A[1][1]
for (i=1 to n)
        det += A[1][i]  * (-1)^(i+1) * det(submatrix(A,i))
return det
}
```

submatrix(A,i) on matriisi, joka saadaan poistamalla rivi 1 ja kolumni j matriisista A. Algoritmin aikavaativuus on

T(n) = O(1) kun n = 1

T(n) = n(T-1) kun n > 1

Nyt T(n) = nT(n-1) = n(n-1)T(n-2) = n(n-1)(n-2)*...*1 = n!

Aikavaativuus on siis O(n!). Tässä oletettin, että alimatriisin laskeminen on vakioaikaista. Omassa totetuksessani se on O((n-1)²), mutta se ei vaikuta algoritmin aikavaativuusluokkaan sillä O(n!) on dominoiva.

Rekursion syvyys on korkeintaan n, ja ensimmäisellä kierroksella tarvitaan tilaa (n-1)², seuraavalla (n-2)² jne. Tilavaativuus on siis $O\left(\sum_{i=1}^{n}(n-i)^2\right)$ = O(n³).

Aikavaativuusluokka O(n!) tarkoittaa, että algoritmi on käyttöalue on käytännössä hyvin pieni.

Tämän huomasi suorituskykytestauksessa: 11×11 matriisin determinantin laskeminen kestää yli kuusi sekuntia.

2.4 Kertolasku

Syöte: $n \times m$ matriisi A ja $m \times p$ matriisi B

Tuloste: $n \times p$ matriisi C

Pseudokoodi:

```
for (i=1 to n)
        for (j=1 to p)
                for (k=1 to m)
                        C[i][j] += A[i][k] + B[k[j]
return C
```

Luotavassa matriisissa on n*p alkiota ja jokaisen alkion laskeminen vaatii m vakioaikaista operaatiota, joten algoritmin aikavaativuus on O(nmp). Jos syötteenä on neliömatriisi, aikavaativuus on $O(n^3)$. Algoritmissa luodaan matriisi, jonka tilavaativuus on O(mn), muiden apumuuttujien tilavaativuus on O(1), joten algoritmin tilavaativuus on O(mn).

2.5 Strassen

Syöte: kaksi $n \times n$ matriisia A ja B

Tuloste: $n \times n$ matriisi C

Pseudokoodi karkealla tasolla (Cormen et al., 2009: 77,79):

1. Jaa matriisit A, B ja C neljään $n/2 \times n/2$ matriisiin. Aikavaativuus $O(n^2)$.

2. Luo kymmenen $n/2 \times n/2$ matriisiia, jotka ovat kahden edellisessä askeleessa luodun matriisin ero tai summa. Aikavaativuus $O(n^2)$

3. Laske seitsemän matriisituloa rekursiivisesti käyttäen askeleissa 1 ja 2 luotuja matriiseja. Jokainen matriisi on $n/2 \times n/2$.

4. Laske matriisin C neljä alimatriisia vähentämällä ja yhteenlaskemalla askeleessa 3 luotujen matriisien erilaisia yhdistelmiä. Aikavaativuus $O(n^2)$

Nyt saadaan seuraava rekursioyhtälö:

$T(n) = O(1)$ kun $n = 1$,

$T(n) = 7T(n/2) + O(n^2)$ kun $n > 1$.

Rekursion ratkaisu on O(n^l(g7)) = $O(n^{2,81})$. (Cormen et al., 2009: 79-80)

Kun syötteenä on kaksi $n \times n$ matriisia, joissa n ei ole kahden potenssi, matriisia täytyy laajentaa nollilla, jotta siitä tulee m × m matriisi, missä m on ensimmäinen n:ä suurempi kahden potenssi. Tämän takia tällaisten n × n matriisien kertomiseen menee käytännössä suunnilleen saman verran aikaa kuin laajennetujen m × m matriisien kertomiseen. Tämä havaitaan suorituskykytestauksessa. Tällä ei kuitenkaan ole vaikutusta aikavaativuusluokkaan (Cormen et al., 2009: 82). Tämän todistaminen jätetään lukijalle harjoitukseksi.

Rekursion ensimmäisellä kierroksella tarvitaan tilaa n^2 seuraavalla $n^2/4$, sitten $n^2/16$ jne.

$\sum_{k=1}^{\infty} n^2/k = 4/3\, n^2$, joten algoritmin tilavaativuus on $O(n^2)$.

Strassenin $O(n^{2,81})$ aikavaativuus verrattuna naiivin algoritmin $O(n^3)$ aikavaativuuteen kuulostaa pieneltä parannukselta, mutta suorituskykytestauksen perusteella Strassen on käytännössä selkeästi tehokkaampi. Kokeilujen perusteella on optimaalista pysäyttää rekursio ja käyttää Strassenin algoritmissa perinteistä kertolaskua, kun syötteenä on 64 × 64 tai sitä pienempi matriisi, joten tämä on käytetty rajana testauksessa.

3. Parannusehdotukset

Strassenin algoritmia voisi tehostaa niin, että kappaleessa 2.5 esitetyn pseudokoodin toisessa kohdassa loisi alimatriisit manipuloimalla indeksejä, jolloin ei tarvitsisi luoda uusia matriiseja. Tämän kohdan aikavaativuus olisi silloin vakio. Algoritmin aikavaativuusluokkaan tämä ei vaikuttaisi, mutta käytännössä algoritmi nopeutuisi.

Olisi hyvä implementoida nopeampi algoritmi determinantin laskemiseksi, sillä kuten todettua, Laplacen algortmin aikavaativuusluokka on O(n!), ja siten sen käyttöalue on hyvin rajoittunut. Matriisilaskinta voisi myös täydentää muilla matriisioperaatioilla, esimerkiksi Gauss-Jordan eliminoinnilla.

Strassenin algoritmistä on hyötyä vasta, kun syötteenä on vähintään 128 × 128 matriisi. Niin isoja matriiseja ei halua syöttää käsin, joten olisi hyödyllistä, jos voisi lukea matriiseja tiedostosta.

Lähteet

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. ja Stein, Clifford. *Introduction to algorithms.* 3. painos. Cambridge, MA : MIT Press, 2009.