Stanisław Denkowski
305288

Teoria Współbieżności

Zadanie domowe 6

Projekt i implementacja współbieżnej metody Gaussa

Sprawozdanie

1. Definicja podstawowych niepodzielnych zadań obliczeniowych.

Zachowamy taką samą notację, jak na ćwiczeniach laboratoryjnych (M to macierz, indeksy dolne oznaczają konkretną wartość macierzy):

$A_{i,k}$ - znalezienie mnożnika dla wiersza *i*, do odejmowania go od *k*-tego wiersza

$$m_{k,i} = M_{k,i} / M_{i,i}$$

$B_{i,j,k}$ - pomnożenie *j*-tego elementu wiersza *i* przez mnożnik - do odejmowania od *k*-tego wiersza

$$n_{k,i} = M_{i,j} * m_{k,i}$$

$C_{i,j,k}$ - odjęcie *j*-tego elementu wiersza *i* od wiersza *k*,

$$M_{k,i} = M_{k,i} - n_{k,i}$$

Powyższe podstawowe i niepodzielne zadania wystarczają by wykonać całe zadanie.

2. Ciąg zadań obliczeniowych wykonywanych przez algorytm sekwencyjny.

Mamy macierz M o rozmiarze n × n, do której dodajemy kolumnę wyrazów wolnych, co w efekcie daje nam macierz n × (n+1).
Odejmujemy przeskalowany wiersz od odpowiedniego wiersza poniżej (dla każdego wiersza poniżej).
Czyli, dla każdego wiersza, patrzymy na każdy wiersz poniżej, wyznaczamy współczynnik i odejmujemy odpowiednio przeskalowany wiersz.
Słowo jest postaci:

$$for\ i \in \{1..n-1\},\ k \in \{i+1..n\}\ A_{i,k}\ (for\ j \in \{i..n+1\}\ B_{i,j,k}\ C_{i,j,k})$$

Z tą uwagą, że najpierw zwiększamy j, później k, a dopiero na koniec i.

3. Alfabet w sensie teorii śladów.

$$\Sigma = \{\forall i \in \{1..n-1\}, k \in \{i+1..n\} A_{i,k}\} \cup \{\forall i \in \{1..n-1\}, k \in \{i+1..n\}, j \in \{i..n+1\}\ B_{i,j,k}, C_{i,j,k}\}$$

4. Relacje zależności.

Zastanówmy się najpierw, od czego zależą operacje na i-tym wierszu i jakie operacje są od nich zależne.
Operacje, które miałem na myśli:

$$\forall\ k \in \{i+1..n\}\ A_{i,k},\ \forall\ k \in \{i+1..n\},\ l \in \{i..n+1\}\ B_{i,l,k} C_{i,l,k}$$

Aby wyznaczyć $A_{i,k}$ musimy skończyć wszystkie operacje odejmowania $C_{x,i,i}$ i $C_{x,i,k}$.
Aby policzyć $B_{i,l,k}$ musimy mieć obliczone $A_{i,k}$ oraz dodatkowo odpowiednie operacje $C_{x,l,i}$ muszą być skończone.

Operacje na tych samych elementach, też są od siebie zależne, by nie modyfikować tej wartości naraz.

Pamiętając o przechodniości, którą później uwzględnimy.
D_1 - w oczywisty sposób, aby móc przeskalować element, musimy mieć wyliczony współczynnik
D_2 - w oczywisty sposób, aby móc odjąć przeskalowany element, musimy go najpierw przeskalować
Następne zależności są mniej oczywiste:
D_3 - aby w dobry sposób wyznaczyć współczynnik, musimy już wykonać wszystkie poprzednie odejmowania (korzystamy z przechodniości)
D_4 - mimo, że przy odejmowaniu od elementu kolejność obliczeń nie ma znaczenia (x-a-b=x-b-a), to korzystamy z tych samych elementów i zapisujemy je w tym samym miejscu, co mogłoby generować problemy oraz pozwala nam to korzystać z przechodniości, co znacząco upraszcza pozostałe zależności
D_5 - aby móc odjąć odpowiednią wartość, przed przeskalowaniem jej, musimy mieć pewność, że jest po wykonaniu wszystkich poprzednich odejmowań od niej

$$D_1 = \{(A_{i,k}, B_{i,j,k}) \mid A_{i,k}, B_{i,j,k} \in \Sigma\}$$
$$D_2 = \{(B_{i,j,k}, C_{i,j,k}) \mid B_{i,j,k}, C_{i,j,k} \in \Sigma\}$$
$$D_3 = \{(C_{i-1,i,i}, A_{i,k}), (C_{i-1,i,k}, A_{i,k}) \mid C_{i-1,i,i}, C_{i-1,i,k}, A_{i,k} \in \Sigma\}$$
$$D_4 = \{(C_{i-1,x,y}, C_{i,x,y}) \mid C_{i-1,x,y}, C_{i,x,y} \in \Sigma\}$$
$$D_5 = \{(C_{k-1,j,k}, B_{k,j,l}) \mid C_{k-1,j,k}, B_{k,j,l} \in \Sigma\}$$
$$D = sym((D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5)^+) \cup I_\Sigma$$
$$I = \Sigma^2 - D$$

5. Algorytm eliminacji Gaussa

Chcemy wyzerować elementy macierzy pod przekątną. Zdefiniujmy operację, która wyzeruje nam konkretny element - jeden krok algorytmu, czyli odjęcie wiersza (dla $d_{i,k}$ odjęcie odpowiednio przeskalowanego i-tego wiersza od k-tego).

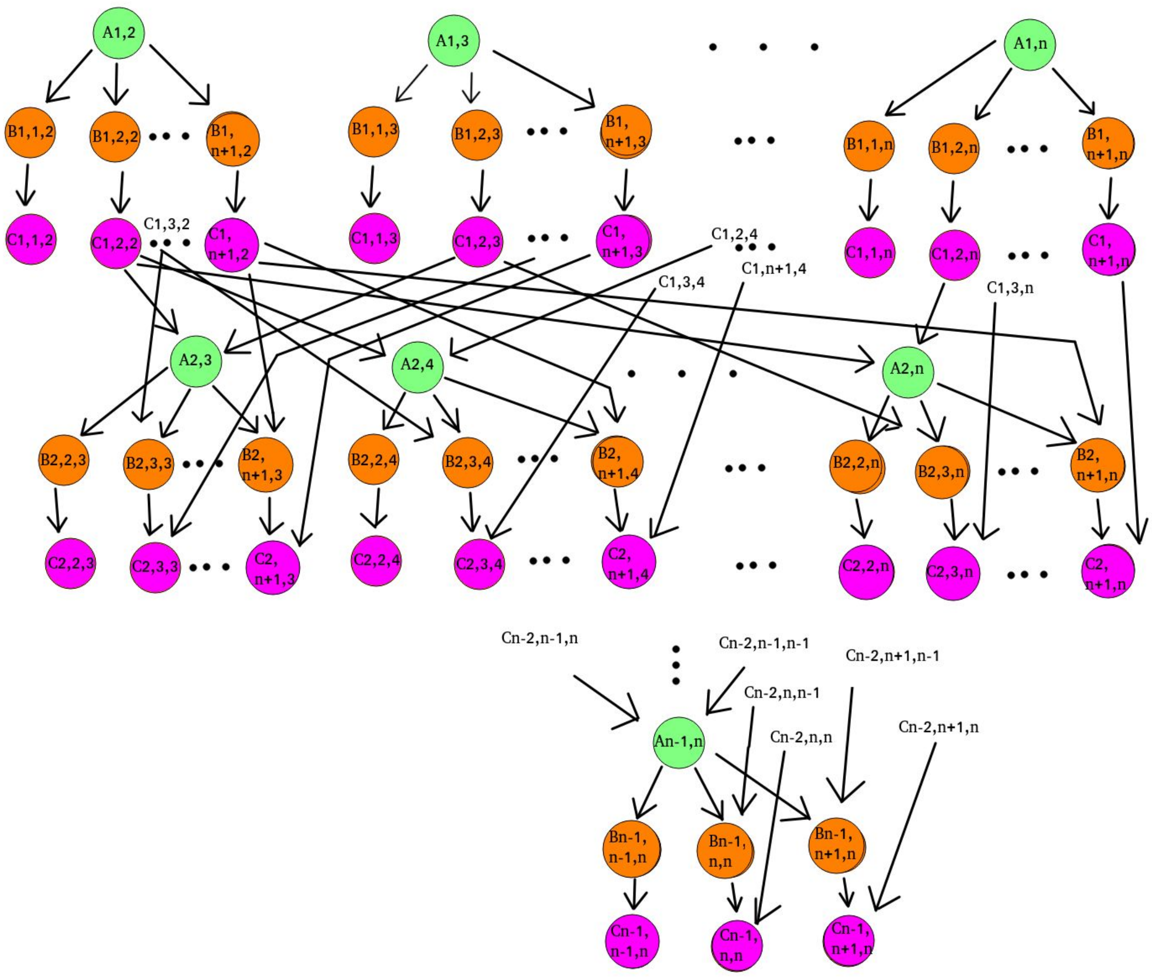
$$d_{i,k} = (A_{i,k}, B_{i,i,k}, C_{i,i,k}, B_{i,i+1,k}, C_{i,i+1,k}, ..., B_{i,n+1,k}, C_{i,n+1,k})$$

Cały algorytm to kolejne wykonywanie analogicznych kroków dla kolejnych wierszy.

$$(D_{1,2}, D_{1,3}, D_{1,4}, ..., D_{1,n}, D_{2,3}, D_{2,4}, ..., D_{n-1,n})$$

6. Graf Diekerta

Od razu kolorystycznie podzielony na klasy Foaty, graf w wersji nieskończonej.



7. Klasy Foaty

Jak widać na grafie w punkcie poprzednim, mimo dużego skomplikowania, klasy Foaty są bardzo podobne do postaci iteracyjnej algorytmu. Najpierw musimy policzyć stosunek elementów, następnie przeskalować wiersz i na koniec odjąć odpowiednie wiersze. Różnica w stosunku do algorytmu iteracyjnego jest taka, że na każdym etapie algorytmu każdy wiersz traktujemy niezależnie (przy wyliczaniu stosunku), oraz każdy element w wierszu traktujemy niezależnie (przy skalowaniu i odejmowaniu).

Możemy nazwać klasy Foaty:

$$F_{Ak} = [\forall i \in \{k+1..n\}\ A_{k,i}]_{\equiv_I^+}$$

$$F_{Bk} = [\forall i \in \{k+1..n\}\ j \in \{k..n+1\}\ B_{k,j,i}]_{\equiv_I^+}$$

$$F_{Ck} = [\forall i \in \{k+1..n\}\ j \in \{k..n+1\}\ C_{k,j,i}]_{\equiv_I^+}$$

Ostateczny wynik:

$$FNF = [F_{A1}][F_{B1}][F_{C1}]...[F_{An-1}][F_{Bn-1}][F_{Cn-1}]$$

8. Uruchomienie program

Algorytm zaimplementowałem w języku Java.
Będąc w głównym katalogu projektu należy wykonać następujące komendy:

```
$ > java  -jar target/Gauss.jar nazwaPlikuWejsciowego nazwaPlikuWyjsciowego
```

Nazwa pliku włącznie z ewentualnym rozszerzeniem i ścieżką względną (np. ../in.in)

W przypadku, gdy program nie zadziała, należy najpierw wygenerować najpierw plik jar poprzez maven:

```
$> mvn install
$> mvn package
```