Zad. nr 1: Macierze

Wer. 1.01 (10 pkt.)
Zadane: 1 V 2022

Macierze liczb rzeczywistych rozmiaru n*m (n, m całkowite dodatnie) można reprezentować na różne sposoby. Najoczywistszym jest dwuwymiarowa tablica liczb typu double. Jeśli jednak wiemy coś więcej o zawartości macierzy, to możemy wybrać inne rozwiązania. Szczególne wtedy, gdy nie mamy tyle pamięci, by przechowywać każdy element macierzy. Możemy na przykład nie pamiętać tych elementów, które pojawiają się najczęściej. Zwykle taką najczęściej powtarzającą się wartością jest liczba zero. Macierze, w których rzadko pojawiają się wartości różne od tej najczęstszej nazywamy rzadkimi.

Macierze rzadkie mogą mieć bardzo regularną postać. Przykładamy takich macierzy są macierz zerowa (zawiera tylko zera), macierz stała (jest wypełniona w całości tylko jedną wartością, np. macierz zerowa jest wypełniona w całości wartością zero), kwadratowa macierz jednostkowa (ma jedynki na głównej przekątnej, przyjmijmy że na tej, gdzie indeksy są sobie równe, poza tym zera), kwadratowa macierz przekątniowa (ma wartości tylko na głównej przekątnej, znów przyjmijmy, że na tej, gdzie indeksy są sobie równe), macierz kolumnowa (wszystkie wartości w każdym wierszu są sobie równe, wiersze zapewne są różne), macierz wierszowa (wszystkie wartości w każdej kolumnie są sobie równe, kolumny zapewne są różne).

Są również macierze rzadkie o nieregularnej budowie. Możemy je reprezentować na różne sposoby. Dość narzucającym się jest tablica wierszy reprezentowanych przez listy - powiązane referencjami kolejne rzadkie (tzn. różne od najczęściej się powtarzającej, np. niezerowe) wartości. Są oczywiście inne możliwości (tablica list kolumn, lista list, słownik, ...). Dla macierzy nieregularnych należy wybrać sobie jedną z realizacji przedstawionych w tym akapicie (**poza** słownikiem) i tylko tę jedną zawrzeć w rozwiązaniu.

Na macierzach można wykonywać różne operacje. Podstawową jest oczywiście podawanie pojedynczego elementu. Inne to mnożenie (oraz dodawanie i odejmowanie) przez drugą macierz lub przez skalar (liczbę typu double). Kolejna grupa operacji to liczenie norm (np. normy maksimum, czy normy Frobeniusa). Jeszcze inne operacje, to przedstawienie macierzy w postaci dwuwymiarowej tablicy liczb double, albo w postaci napisu (toString). Oczywiście macierz powinna potrafić podać swoje wymiary.

Celem tego zadania jest stworzenie zestawu klas implementujących na różne (podane powyżej) sposoby macierze. Należy zadbać o realizowanie zadanego, jednolitego interfejsu dla wszystkich macierzy oraz należy spróbować zawrzeć powtarzające się operacje w klasie/klasach abstrakcyjnych.

Sytuacje błędne (np. niepasujące do siebie rozmiary macierzy), należy wykrywać i obsługiwać za pomocą instrukcji asssert (pamiętajmy, że domyślnie sprawdzanie assert jest wyłączone, trzeba je włączyć opcją -ea). Wkrótce na zajęciach będziemy omawiać inny pasujący tu mechanizm - wyjątki, ale w tym zadaniu jeszcze go nie używamy.

Operacje na macierzach (np. liczenie normy) powinny wykorzystywać typ macierzy do ograniczenia wykonywanych operacji (np. do zredukowania liczby operacji z n^2 do n). Do mnożenia macierzy można wykorzystać standardowy algorytm, działający dla macierzy n*n w czasie n^3.

Do tego zadania dołączone jest archiwum zip, zawierające:

- interfejs IDoubleMatrix,
- szkielet klasy DoubleMatrixFactory (bez treści zawartych tam metod klasowych),
- zestaw testów korzystających z biblioteki JUnit.

Macierze z Państwa rozwiązania powinny implementować interfejs IDoubleMatrix. Należy uzupełnić treść klasy DoubleMatrixFactory i testy o te klasy macierzy, które są wymienione w treści zadania, ale nie ma ich w fabryce/testach. Państwa rozwiązanie powinno spełniać podane testy. Nie można zmieniać treści podanych klas/interfejsów, ale można dodawać do przedstawionych klas/interfejsów własne dodatkowe metody. Można oczywiście także, gdyby okazało się to potrzebne, tworzyć pomocnicze klasy/interfejsy. Można dodawać jeszcze inne (niż wymienione powyżej) własne testy (oczywiście nie modyfikując ani nie usuwając tych zadanych).

Należy zaimplementować co najmniej jedną optymalizację tworzenia macierzy wynikowych (tzn. tworzenia macierzy, która nie jest pełną macierzą, np. dodanie do siebie dwu macierzy przekątniowych da w wyniku macierz przekątniową, a nie zwykłą). To optymalizowanie powinno mieć miejsce przed (a nie po) stworzeniu macierzy (czyli nie chodzi o tworzenie pełnej macierzy, a potem sprawdzanie, czy nie ma ona szczególnej postaci). Ta optymalizacja (te optymalizacje) mogą korzystać z dodatkowej (dodatkowych) operacji dodanych do interfejsu macierzy lub z techniki podwójnego przekierowywania (ang. double dispatch) lub ze sprawdzania, czy dwa obiekty są obiektami tej samej klasy (porównanie wyników operacji getClass).

Proszę zwrócić uwagę, że w tym zadaniu macierze są niemodyfikowalne (to ma różne, pozytywne i negatywne, konsekwencje - w tym zadaniu wybraliśmy niemodyfikowalne). Proszę w związku z tym zastanowić się, czy nie pozwoli to zmniejszyć czasu wykonania niektórych operacji podanych w interfejsie macierzy i - jeśli tak - należy to zrobić.

Przy implementowaniu poszczególnych macierzy należy wykorzystywać ich strukturę, tak by nie pamiętać zbędnych wartości i - jeśli to możliwe - szybciej wyliczać wyniki.

W funkcji main należy umieścić wypisywanie każdego z rodzajów zaimplementowanych macierzy. Macierze mają mieć rozmiar 10 na 10. Ich zawartość może być dowolna (np. losowana lub ustalona w kodzie) zgodna ze specyfiką danego typu macierzy. Każda macierz ma być wypisana przy użyciu operacji toString. Macierze mają być wypisywane po jednym wierszu w jednej linii (tekst generowany przez *toString* powinien zawierać sekwencje \n). W czasie wypisywania macierze (zgodnie ze swoją specyfiką) mają zastępować serię co najmniej trzech powtórzeń w wierszu jednej wartości *w* przez zapis z trzema kropkami: *w ... w.* Uwaga: chodzi o kompresowanie (zastępowanie zapisem z kropkami) powtórzeń wynikających z rodzaju macierzy, jeśli dane przechowywane w macierzy będą zawierały dane z powtórzeniami na sąsiednich pozycjach w wierszu, nie trzeba tego wykrywać i kompresować. Należy też policzyć i wypisać wyniki (dla przykładowych danych) operacji wykonanych przez Państwa program (po jednym wypisaniu dla każdej zaimplementowanej metody).

Nie można korzystać z żadnych bibliotek poza standardową biblioteką Javy (dostarczaną z JDK).

Jako rozwiązanie należy przesłać kompletny (kompilujący się i przechodzący testy) projekt z IDE, którego Państwo używają. Jeśli Państwa rozwiązanie korzystało z jakiejś nietypowej (tzn. niestandardowej w stosowanym przez Państwa IDE) architektury projektu (do czego zdecydowanie nie zachęcamy), to należy dokładnie opisać jak uruchomić Państwa testy. Prosimy też podać w komentarzu w kodzie jakiego IDE Państwo używali. W razie jakiś wątpliwości co do formy przesyłanego rozwiązania proszę się kontaktować z osobą prowadzącą zajęcia w Państwa grupie (programy domowe są sprawdzane przez osoby prowadzące zajęcia w grupach laboratoryjnych).

Rozwiązanie należy wysłać jako spakowane archiwum zip do Moodle'a (do zadania)

Życzymy powodzenia!

Historia modyfikacji:

- 1.00 (1 V) pierwsza wersja
- 1.01 (3 V) usunięta wzmianka o DoubleVector.