Programowanie wizualne

opracował: Wojciech Frohmberg

Lab 6

Zagadnienia do opanowania:

- NUnit
- Test-driven development

Zarys problemu:
Do projektu z poprzednich zajęć chcielibyśmy dodać funkcjonalność równomiernego pod względem czasochłonności podziału zadań pomiędzy użytkowników. Problem ten z natury rzeczy jest trudny do rozwiązania w postaci ogólnej (jest jednym z problemów należących do klasy problemów silnie NP-trudnych). Skupimy się zatem na jego podproblemie tj. na problemie podziału zadań między dwóch użytkowników, który również jest NP-trudny, ale posiada pseudowielomianowy algorytm rozwiązujący. Skorzystamy tutaj z metodyki test-driven development tj. spróbujemy opracować testy jednostkowe, które wspomogą nas w implementacji algorytmu. Celem zajęć jest pokazanie, iż pomimo dość skomplikowanego do zaimplementowania algorytmu – jego testy mogą być naprawdę proste do napisania!

1. Do utworzonego na poprzednich zajęciach projektu dodaj folder Algorithms.
2. W ramach folderu utwórz interfejs IPartitionAlgorithm sparametryzowany typem T. W zamierzeniach za pomocą tego typu przekażemy informację jaki jest typ elementu, który przechowujemy w zbiorze do podziału. Chcemy zatem napisać algorytm na tyle ogólnie, żeby można było go zastosować nie tylko w przypadku podziału zadań pomiędzy użytkowników, ale w przypadku dowolnego zbioru elementów, który chcielibyśmy podzielić na podzbiory o równiej sumie pewnych wag.
3. Do interfejsu dodaj właściwość Predicate (z publicznymi elementami get i set), w ramach której przechowasz funktor przyporządkowujący elementowi naszego zbioru – całkowitoliczbową wartość jego wagi (w naszym przypadku czasochłonności zadania). Możesz przyjąć, że typ zwracany przez funktor to int. Jakiego typu dostępnego ze standardu C# możesz użyć do przechowania funktora? Jeśli jeszcze go nie użyłeś – spróbuj zmienić użyty typ na Func.
4. Do interfejsu dodaj właściwość SubsetsCount (z publicznymi elementami get i set), w ramach której przechowasz liczbę podzbiorów, na które ma zostać podzielony zbiór wejściowy przy użyciu algorytmu.
5. Do folderu dodaj klasę wyjątku, która będzie obrazowała nieobsługiwaną przez algorytm liczbę podzbiorów, na które miałby zostać podzielony zbiór wejściowy.
6. Do interfejsu dodaj metodę Run, która będzie przyjmowała wejściowy zbiór elementów do podziału w formie List<T>. Metoda powinna zwracać List<List<T>> tj. listę podzbiorów, na którą algorytm podzielił nasz zbiór wejściowy.
7. Do folderu dodaj klasę wyjątku, która będzie obrazowała sytuację, gdy nie da się dokonać podziału naszego zbioru wejściowego na zadaną liczbę podzbiorów o tych samych sumarycznych wagach.
8. Do folderu dodaj klasę wyjątku, która będzie obrazowała sytuację przekazania w ramach zbioru elementu o niedodatniej wadze.
9. Do rozwiązania dodaj projekt testowy NUnit i nazwij go TaskShareAlgorithmsTests.
10. Do projektu dodaj referencję projektu MVC.
11. W ramach projektu testów jednostkowych utwórz klasę testową TaskBinaryPartitionAlgorithmsTests sparametryzowaną typem algorytmu do przetestowania.

12. Na parametr klasy nałóż ograniczenie spełniania interfejsu IPartitionAlgorithm<Task> oraz możliwości utworzenia instancji klasy poprzez domyślny bezparametrowy konstruktor (ograniczenie new()).
13. Utwórz metodę inicjalizacyjną testów. W jej ramach zawrzyj tworzenie instancji algorytmu i wstawianie jej do prywatnego pola klasy (typu IPartitionAlgorithm<Task>), pamiętaj również o zainicjalizowaniu właściwości Predicate. Zwróć uwagę, że dzięki wspomnianemu podejściu na tym etapie możesz zupełnie abstrahować od faktycznej klasy podlegającej testom, wystarczy że klasa trzyma się wskazanego, zaproponowanego a priori interfejsu.
14. Utwórz zbiór testów, które dla zakresu n większego od 2 i mniejsze lub równego 11 (z krokiem 1) zweryfikuje czy próba ustawienia pod właściwość SubsetsCount wartości n skutkuje wyrzuceniem wyjątku. Wyrzucenie wyjątku jest tutaj pożądane z racji, że testujemy funkcjonalność binarnego podziału zbioru.
15. Utwórz zbiór testów, w ramach których testowane będzie wyrzucanie wyjątku w przypadku przekazania do zbioru wartości niedodatniej (być może jako pojedynczy element spośród wielu dodatnich). Posłuż się do tego celu atrybutem TestCaseSource, który powinien odpowiadać za wygenerowanie (w pętli) 100 zbiorów z losowymi elementami (niekoniecznie niedodatnimi). Każdy spośród zbiorów ponienien posiadać 3 elementy. W ramach testu skorzystaj z klasy Assume, żeby zweryfikować czy test dla podanego zbioru wejściowego powinien być uruchomiony tj. czy posiada przynajmniej jedną wartość niedodatnią.
16. Utwórz zbiór 5 testów ze zbiorem, elementów którego jesteś pewna/pewny, że nie da się podzielić. Np. Zbiór jednoelementowy. Zadbaj by każdy spośród podanych zbiorów posiadał inne właściwości, które chcesz przetestować. Do przeprowadzenia testów skorzystaj z atrybutu TestCase umieszczonego na metodzie testującej wielokrotnie. Zbadaj czy w ramach metody Run zostanie wyrzucony wyjątek braku możliwości podziału zbioru na podzbiory.
17. W ramach klasy testującej utwórz metodę generującą losowy zbiór, który na pewno da się podzielić na dwa podzbiory o równej sumie. Niech metoda przyjmuje w parametrze pożądaną sumę podzbioru oraz wartość maksymalną pojedynczego elementu. Metoda powinna w dwukrotnej iteracji dodawać do wynikowego zbioru losowe elementy typu Task o czasochłonności nieprzekraczającej maksymalnego elementu przekazanego w parametrze, jednak w taki sposób żeby suma tego elementu i poprzednich nie przekraczała pożądanej sumy podzbioru. W tym przypadku, gdy przekroczy – element przed dodaniem elementu należy zmniejszyć jego czasochłonność tak by dopełnił sumę.
18. Skorzystaj z metody do wygenerowania 50 testów weryfikujących poprawność działania algorytmu, zadbaj by przed uruchomieniem testu liczby zostały „przetasowane”.
19. Zaimplementuj algorytm partycjonowania BinaryPartitionAlgorithm sparametryzowany typem T pojedynczego elementu zbioru do podziału. Niech algorytm implementuje interfejs IPartitionAlgorithm<T>. Do celu implementacji algorytmu możesz skorzystać z zewnętrznych źródeł (np. https://www.geeksforgeeks.org/partition-a-set-into-two-subsets-such-that-the-difference-of-subset-sums-is-minimum/). W międzyczasie implementacji weryfikuj co jakiś czas, które spośród testów jednostkowych dla klasy algorytmu zaczynają przechodzić pozytywnie. Do tego celu skorzystaj z odpowiedniego atrybutu (TestFixture, za pomocą którego przekażesz BinaryPartitionAlgorithm do generycznego parametru testu).