**Programowanie III**

Dokumentacja projektu „Lista zakupów” z konkursu „Algorytmion” 2012

Kamil Giziński, Dominik Sigulski, Bartosz Rolnik, gr. 8 Matematyka Stosowana, Informatyka, sem. III, 2021/22

Spis treści

1. [Temat Projektu 2](#_Toc93146223)
   1. [Treść zadania „Lista zakupów” 3](#_Toc93146224)
   2. [Skrócona treść zadania „Lista zakupów” 3](#_Toc93146225)
   3. [Przykłady działania programu 3](#_Toc93146226)
2. [Opis pobieranych danych przez program - wejście 3](#_Toc93146227)
3. [Opis otrzymywanych rezultatów - wyjście 4](#_Toc93146228)
4. [Zastosowane algorytmy do rozwiązania zadania 4](#_Toc93146229)
   1. [Słowny opis wykorzystanych algorytmów 4](#_Toc93146234)
   2. [Schemat blokowy programu 5](#_Toc93146235)
   3. [Zastosowane metody i klasy w programie 6](#_Toc93146236)
5. [Testy na poprawność działania programu 6](#_Toc93146237)
6. [Wnioski 7](#_Toc93146238)

# Temat Projektu

## Treść zadania „Lista zakupów”

Masz n - złotych. Chcesz wydać dokładnie całą kwotę na zakupy. Zrobiłeś sobie listę produktów (i ich cen), które chętnie chciałbyś mieć. Napisz program, który wybierze z Twojej listy produkty, które musisz kupić w taki sposób, aby wydać wszystkie pieniądze. Takich konfiguracji produktów może być kilka, może się zdarzyć, że będzie dokładnie jedna, albo że nie ma takiej konfiguracji aby wydać dokładnie wszystkie pieniądze.

**Wejście - plik: wej.txt**

Pierwsza linia tego pliku to kwota, jaką mamy na zakupy. Kolejne linie zawierają ceny produktów, którymi jesteśmy zainteresowani oraz (po spacji do końca linii) opis tej pozycji. Produktów może być do 32. Suma cen produktów jest zawsze większa niż kwota, jaką posiadamy.

**Wyjście - plik: wyj.txt**

Każda linia tego pliku to konfiguracje produktów spełniające założenia zadania. Każda konfiguracja składa się z wybranych numerów porządkowych produktów z listy.

## Skrócona treść zadania „Lista zakupów”

W pliku wej.txt w pierwszym wierszu znajduje się budżet jaki mamy do rozdysponowania. Następne wiersze zawierają kwotę oraz - po spacji aż do końca linii - produkt. Należy do pliku wyj.txt wpisać wszystkie konfiguracje numerów porządkowych produktów, których ceny w sumie równają się wartości budżetu.

## Przykład działania programu

**wej.txt:**

100.00

59.99 płyta z muzyką

12.90 czasopismo

999.99 gitara

50.10 T-shirt

37.00 bilet do kina

89.19 film BR

45.20 książka

**wyj.txt:**

2 4 5

# Opis pobieranych danych przez program - wejście

Pierwszy wiersz pliku wejściowego zawiera budżet jaki mamy do rozdysponowania. Następne wiersze aż do końca pliku to kolejno kwota oraz (po spacji aż do końca linii) produkt, który znajduje się na naszej liście zakupów. W naszym programie poszerzyliśmy możliwość wczytywania pliku wejściowego. Użytkownik, przy pomocy pojawiającego się okienka menadżera plików może wybrać wiele plików i zatwierdzić wybór przyciskiem „Otwórz”.

# Opis otrzymywanych rezultatów - wyjście

Wyniki zapisywane są do pliku wyjściowego w postaci kombinacji indeksów poszczególnych produktów, których to wartości w sumie równe są budżetowi. Kombinacji takich produktów może być wiele, może być tylko jedna taka kombinacja oraz może się zdarzyć, iż żadna kombinacja nie będzie spełniała warunków zadania. W naszym programie wyniki wyświetlane są również w postaci graficznej w oknie aplikacji od razu po przetworzeniu danych.

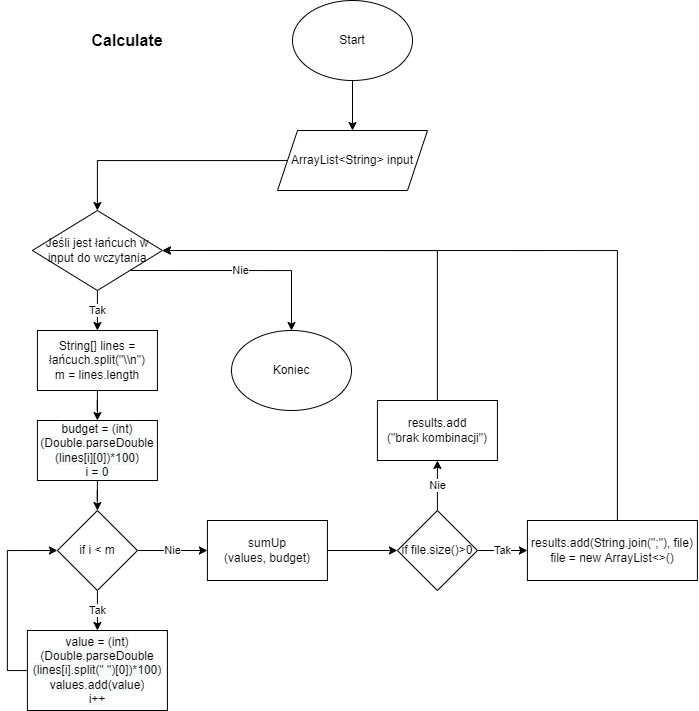
# Zastosowane algorytmy do rozwiązania zadania

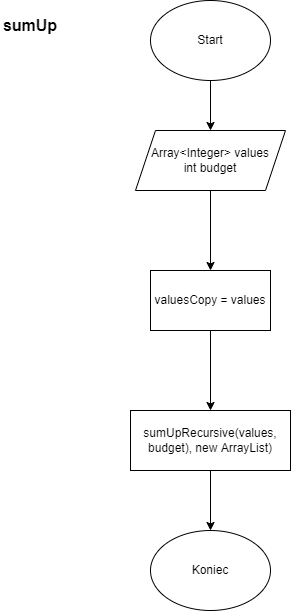


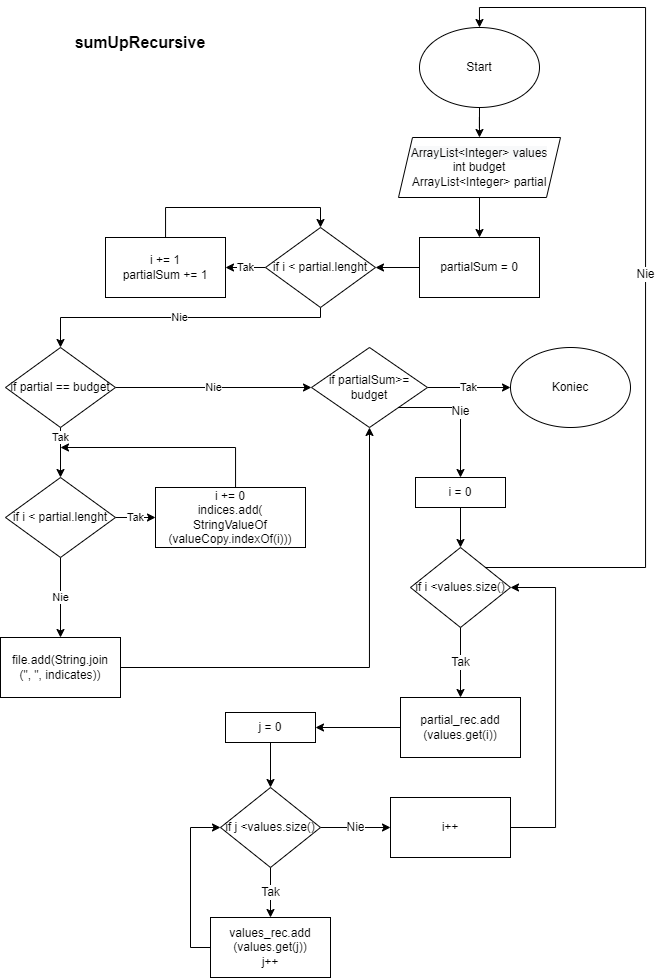
## Słowny opis wykorzystanych algorytmów

Do rozwiązania głównego problemu zadania została stworzona klasa Calculate a w niej metoda calculate(), która krok po kroku została opisana w punkcie 4.3.. Wykorzystany algorytm polega na wczytaniu pierwszej linii z pliku wejściowego jako budżetu oraz pozostałych linii, w których zawarte są ceny i produkty. Na tym etapie programu interesują nas jedynie indeksy produktów oraz ich ceny i to właśnie te wartości są zapamiętywane. Tworzone są kombinacje cen produktów, a każda z kombinacji zostaje przetwarzana i sprawdzana przy pomocy metod sumUp() oraz sumUpRecursive(). Kombinacja przede wszystkim musi w sumie równać się wartości budżetu wczytanego z pierwszej linii pliku. Jeżeli dana kombinacja nie spełnia warunku – jest odrzucana, a program przechodzi do tworzenia kolejnych kombinacji i rekursywnego sprawdzania warunków.

## Schematy blokowe programu







## Zastosowane metody i klasy w programie

Główna klasa odpowiedzialna za rozwiązanie problemu zadania to klasa **Calculate**. W tej klasie zostały stworzone metody **calculate()**, **sumUp()** oraz **sumUpRecursive().**

Metoda **calculate()**, czyli główna metoda, która działa na wczytanych z pliku wejściowego danych. Tworzy listę cen produktów oraz zmienną bugdet zawierającą budżet jaki mamy do rozdysponowania. W pętli zapisujemy ceny produktów i wywołujemy metodę pomocniczą **sumUp()** przekazując ceny i budżet. Na koniec, jeżeli plik jest pusty wypisujemy informację o braku kombinacji. Jeżeli jednak znajdują się w nim jakieś dane – do pliku wyjściowego dodajemy delimiter oznaczający koniec kombinacji dla obecnego pliku (gdyż plików wejściowych możemy mieć wiele).

Obraz zawierający tekst

Opis wygenerowany automatycznie

Metoda pomocnicza **sumUp()**, która wykonuje się tylko raz dla każdego pliku wejściowego, tworzy zmienną valuesCopy z kopią cen, która przyda się do wywołania w kolejnej linii metody **sumUpRecursive()** przekazując kolejno: ceny, budżet i listę.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Metoda **sumUpRecursive()**, w której na samym początku sumujemy wartości na liście partial. Partial będzie zwiększać się, ponieważ będziemy dodawać kolejne kombinacje spełniające warunek. Następnie, jeśli budżet został wykorzystany to do pliku wyjściowego dodawane są właściwe kombinacje produktów. Jeżeli kombinacja jest zbyt kosztowna i przekracza budżet – opuszczamy metodę. Jeśli nie przekracza – przenosimy elementy z listy cen do listy pomocniczej i rekurencyjnie wykonujemy metodę **sumUpRecurisve()** dla aktualnej listy cen.

Obraz zawierający tekst

Opis wygenerowany automatycznie

# Testy na poprawność działania programu

Program działa poprawnie dzięki odpowiednim zabezpieczeniom. W pierwszym przypadku, gdy ceny produktów tworzą kombinacje równe budżetowi, otrzymujemy indeksy produktów wypisane na ekranie oraz zapisane do pliku wyjściowego.

Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznie

W drugim przypadku, gdy żadna kombinacja nie spełnia warunków zadania, otrzymujemy informację o braku kombinacji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeżeli wybieramy kilka plików do przetworzenia, dostajemy na wyjściu jasno opisane kombinacje spełniające warunek dla poszczególnych plików.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

# Wnioski

Stworzona aplikacja poprawnie rozwiązuje problem z zadania. Interakcja człowieka z maszyną zrealizowana za pomocą graficznego interfejsu jest nieskomplikowana, estetyczna i przejrzysta. Algorytm okazał się nieskomplikowany, lecz niełatwy w implementacji, ponieważ aplikacja została wyposażona w dodatkowe zabezpieczenia i funkcjonalności, a przy tym pozostała przejrzysta.

Ze względu na dopracowaną implementację algorytmu potrzebnego do rozwiązania zadania nie jest priorytetem usprawnienie jej. Jednakże zawsze w takim przypadku pozostaje pole do rozwoju. W przyszłości aplikację można rozwinąć o współpracę ze stroną www lub dane pobierane od użytkownika, aby aplikacja mogła być pomocna w codziennym życiu.