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

1.1. 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.

1.2. 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.

1.3. 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
```

2. 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".

3. Opis otrzymywanych rezultatów - wyjście

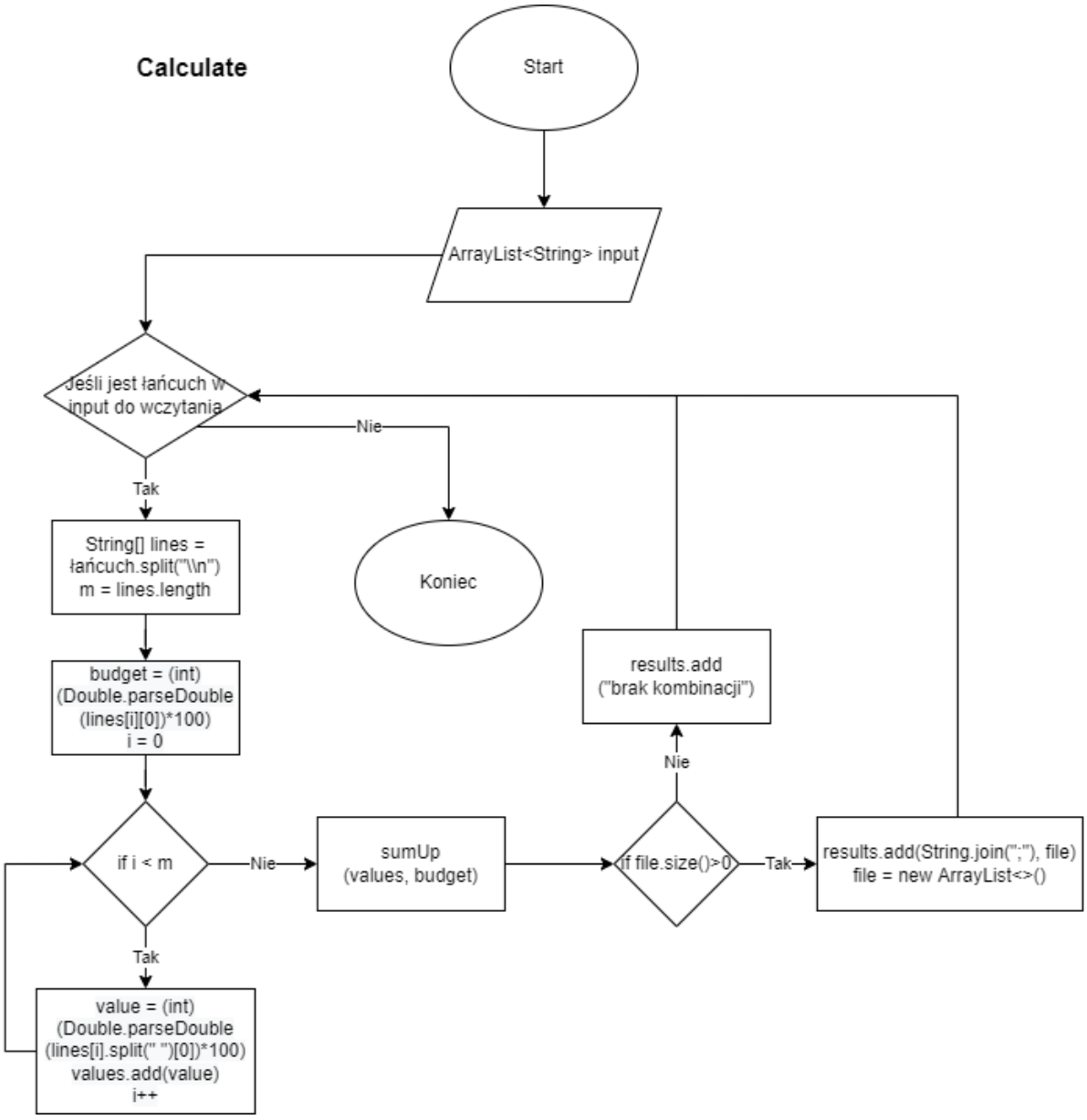
Wyniki zapisywane są do pliku wyjściowego w postaci kombinacji indeksów poszczególnych produktów, których to wartości w sumie równe są budżetowi. Kombinacji takich produktów może być wiele, może być tylko jedna taka kombinacja oraz może się zdarzyć, iż żadna kombinacja nie będzie spełniała warunków zadania.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

4. Zastosowane algorytmy do rozwiązania zadania

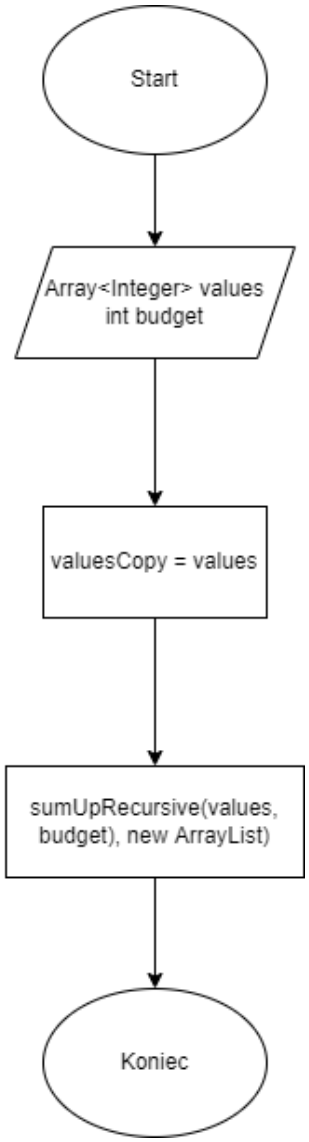
4.1. Słowny opis wykorzystanych algorytmów

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.

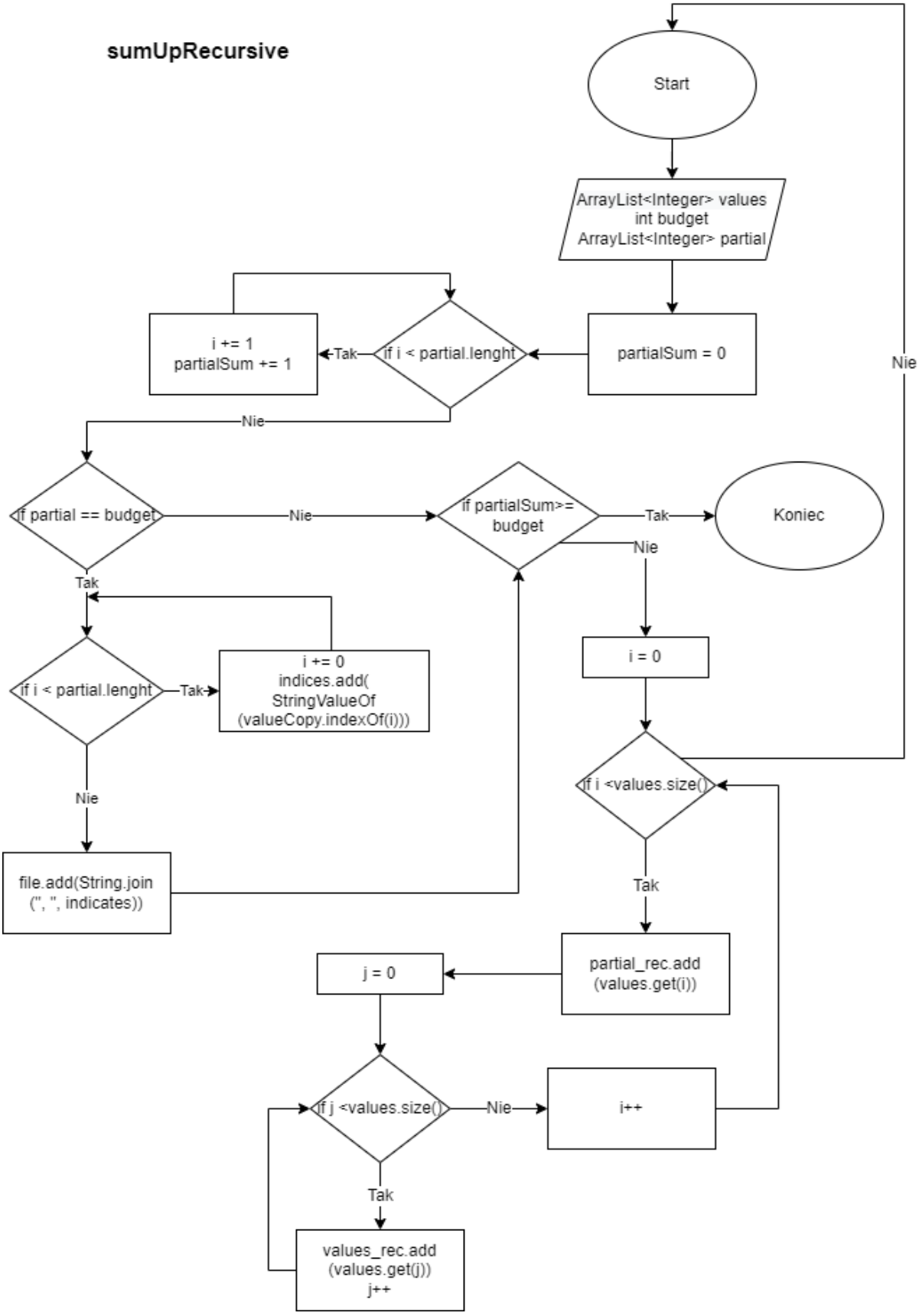
4.2. Schematy blokowe programu



sumUp


Start
Array<Integer> values
int budget
valuesCopy = values
sumUpRecursive(values, budget), new ArrayList)
Koniec

sumUpRecursive



4.3. 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).

```
public void calculate(ArrayList<String> input) {
    int m;
    ArrayList<Integer> values;
    int budget;

    for (String s : input) {
        String[] lines = s.split( regex: "\\n");
        m = lines.length;
        values = new ArrayList<>();
        budget = (int) (Double.parseDouble(lines[0]) * 100);
        for(int i = 1; i < m; i++) {
            int value = (int) (Double.parseDouble(lines[i].split( regex: " ")[0]) * 100);
            values.add(value);
        }
        sumUp(values, budget);
        if(file.size() > 0) {
            result.add(String.join( delimiter: ";", file));
            file = new ArrayList<>();
        }
        else
            result.add("Brak kombinacji.");
    }
}
```

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.

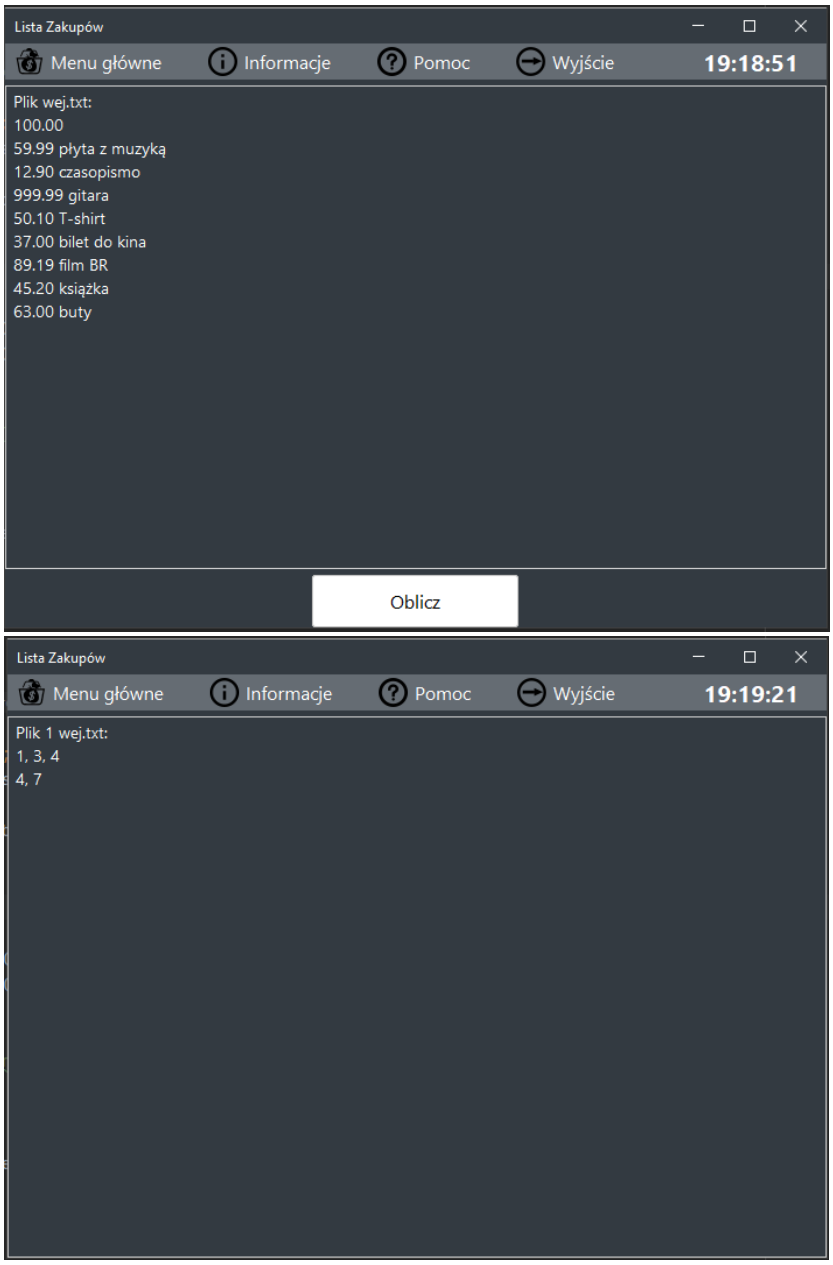
```
void sumUp(ArrayList<Integer> values, int budget) {
    valuesCopy = values;
    sumUpRecursive(values, budget, new ArrayList<>());
}
```

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.

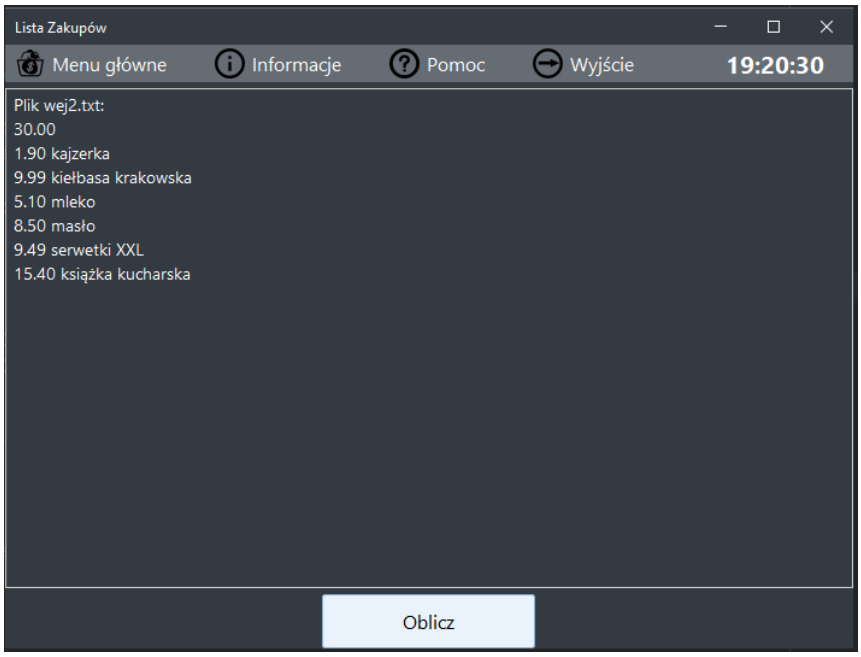
```
void sumUpRecursive(ArrayList<Integer> values, int budget, ArrayList<Integer> partial) {
    // Sumujemy wartości w partial:
    int partialSum = 0;
    for(int i : partial)
        partialSum += i;

    // Jeśli budżet został wykorzystany:
    if(partialSum == budget) {
        ArrayList<String> indices = new ArrayList<>();
        for(int i : partial) {
            indices.add(String.valueOf(valuesCopy.indexOf(i)));
        }
        file.add(String.join( delimiter: ", ", indices));
    }
    // Jeśli przekroczyliśmy budżet, metoda kończy działanie:
    if(partialSum >= budget)
        return;
    // Przenosimy elementy do remaining oraz partial_rec
    for(int i = 0; i < values.size(); i++) {
        // Nowa lista partial_rec równa partial
        ArrayList<Integer> partial_rec = new ArrayList<>(partial);
        // Nowa pusta lista values_rec
        ArrayList<Integer> values_rec = new ArrayList<>();
        // Dodajemy i-ty element values do partial_rec
        partial_rec.add(values.get(i));
        // Pozostałe elementy z values dodajemy do values_rec
        for(int j = i + 1; j < values.size(); j++)
            values_rec.add(values.get(j));
        // Wykonujemy sumUpRecursive rekurencyjnie dla values_rec oraz partial_rec
        sumUpRecursive(values_rec, budget, partial_rec);
    }
}
```

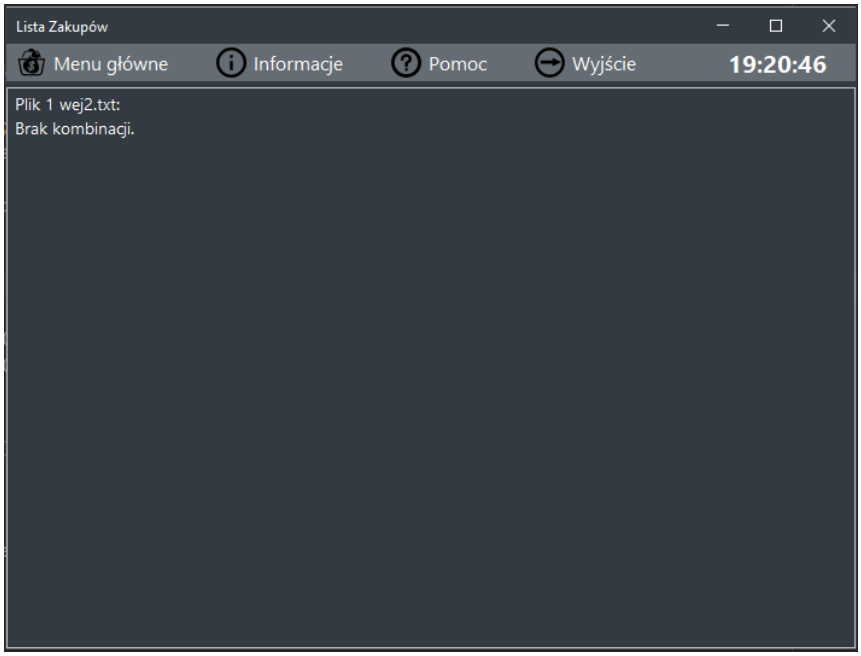
5. Testy na poprawność działania programu

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.

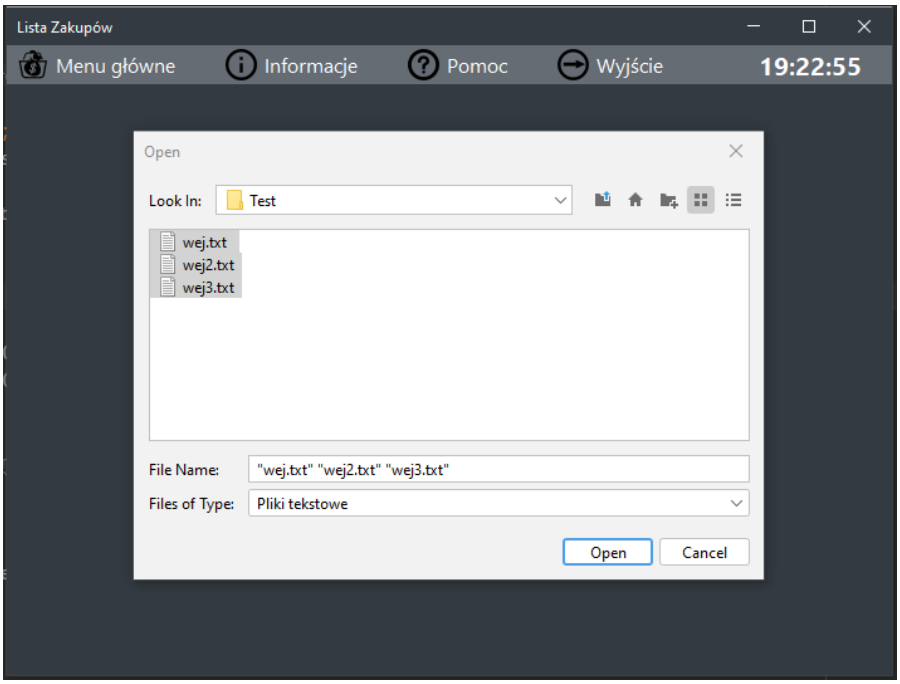


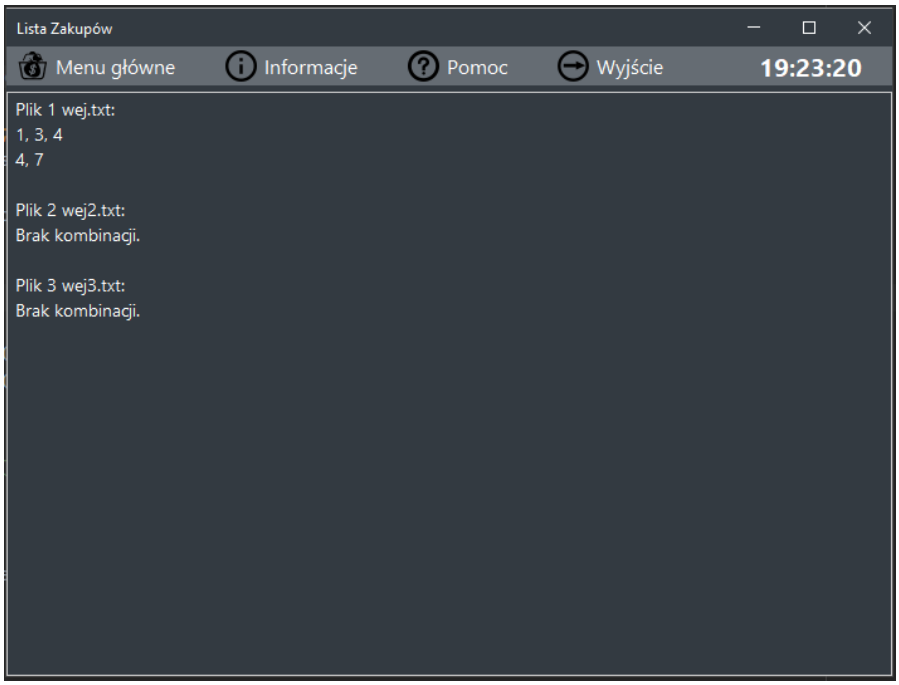
W drugim przypadku, gdy żadna kombinacja nie spełnia warunków zadania, otrzymujemy informację o braku kombinacji.





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





6. Wnioski

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

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.