Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

**Krzysztof Dragan**

**Bartłomiej Dominik**

Aplikacja komputerowa rozwiązująca problem komiwojażera

Projekt zespołowy

na studiach stacjonarnych

o kierunku Informatyka

Opiekun projektu:

Dr inż. Grzegorz Słoń

Kielce, 2018

**SPIS TREŚCI**

[1. Charakterystyka zadania 3](#_Toc291581959)

1.1 Opis problemu

1.2 Wybrane rozwiązanie

1.3 Zastosowane technologie i narzędzia

[2. Podstawa teoretyczna 3](#_Toc291581960)

[2.1. Opis matematyczny problemu 3](#_Toc291581961)

[3. Algorytm obliczeniowy 3](#_Toc291581963)

3.1 Schemat blokowy algorytmu

3.2 Główne części aplikacji

[4. Opis działania aplikacji 3](#_Toc291581964)

[5. Podsumowanie i wnioski 3](#_Toc291581965)

5.1 Wnioski

5.2 Podsumowanie

[6. Instrukcja uruchomienia i obsługi aplikacji 3](#_Toc291581966)

[7. Literatura 4](#_Toc291581967)

# Charakterystyka zadania

Problem komiwojażera (ang. travelling salesman problem, TSP) jest popularnym problemem spotykanym przy optymalizacji drogi między punktami takimi jak

np. miasta czy placówki. Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie.[1]

**1.2 Wybrane rozwiązanie**

W aplikacji zostały wykorzystane dwa algorytmy do rozwiązania problemu komiwojażera. Są to:

**- algorytm genetyczny**

**- algorytm najbliższego sąsiada**

Pierwszy algorytm:

Algorytm genetyczny to inaczej ewolucyjny algorytm służący do odnajdywania optymalnego rozwiązania naszego problemu. Algorytm taki sprawdza wiele różnych rozwiązań, po czym wybiera te, które są na danym etapie najlepsze, i na ich podstawie tworzy dużą ilość nowych rozwiązań. I tak w kółko. W ten sposób, metodą małych kroczków, dąży w kierunku coraz lepszych rozwiązań. [2]

W każdym programie wykorzystującym algorytm genetycznym trzeba zdefiniować takie obiekty jak Populacja i element Populacji. W naszym programie elementem Populacji, inaczej zwanym osobnikiem będzie Podróż (Tour). Obiekt ten to zapisane przejście przez wszystkie miasta.

Nasz algorytm genetyczny będzie tworzył nowe populacje , krzyżował je ze sobą, mutował i przedstawiał najlepsze osobniki populacji w celu osiągnięcia celu jakim jest najkrótsza podróż przez miasta.

Drugi algorytm:

Drugi algorytm jakim jest algorytm najbliższego sąsiada oblicza odległości pomiędzy wszystkimi miastami wykorzystując wszystkie kombinacje. Zaczynając od miasta startowego kieruje swoją drogę do miasta które ma do niego najmniejszą odległość . W kolejnych iteracjach ten krok jest powtarzany dla bieżącego miasta.

**1.3 Zastosowane technologie i narzędzia**

Nasza aplikacja została w całości napisana w języku Java w wersji 8. Całość programu można podzielić na dwie części :

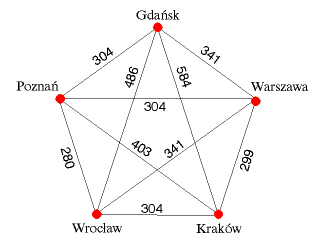
* część odpowiedzialną za logikę – *katalog src/main/java/controllers*
* część odpowiedzialną za interfejs użytkownika (GUI) *katalog src/main/resources*

Wykorzystaliśmy też dwa narzędzia programistyczne:

* Maven – narzędzie do kompilacji projektu oraz tworzenia archiwów .jar
* Git – narzędzie do kontroli wersji i osadzanie kodu źródłowego w sieci

# Podstawa teoretyczna

## Opis matematyczny problemu

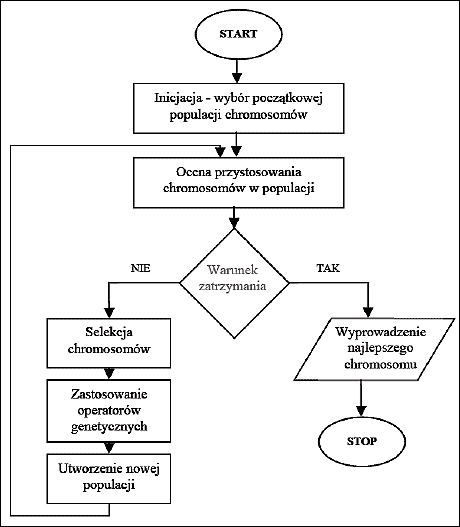
Problem komiwojażera możemy przedstawić jako graf ważony, którego wierzchołki są miastami. Każda pare miast połaczmy krawedziami. Każdej krawedzi nadajemy wage równa 'odległości' miedzy miastami odpowiadajacymi wierzchołkom, które sa końcami tej krawedzi. Otrzymujemy w ten sposób graf pełny, który ma tyle wierzchołków ile miast musi odwiedzić komiwojażer (wliczajac w to miasto, z którego wyrusza). Odwiedzenie wszystkich miast odpowiada cyklowi, ktory przechodzi przez każdy wierzchołek danego grafu dokładnie raz. Cykl taki nazywamy cyklem Hamiltona. Poszukujemy wiec w grafie pełnym cyklu Hamiltona o minimalnej sumie wag krawedzi.[3]

Tab. 2.1. Rysunek grafu Hamiltona dla problemu komiwojażera

Dowolny graf pełny posiada co najmniej jeden cykl Hamiltona. Ponieważ graf ma skończona liczbę wierzchołków, to w zbiorze cykli Hamiltona istnieje taki (niekoniecznie jedyny), który posiada minimalna sumę wag krawędzi.

# Algorytm obliczeniowy

## 3.1 Schemat blokowy algorytmu



Tab. 3.1. Schemat blokowy algorytmu genetycznego [4]

## 3.2 Główne części aplikacji

Kluczowymi elementami naszej aplikacji będzie:

* Klasa Populacji
* Klasa Podróży
* Klasa Miasta(Punktu)
* Metoda krzyżowania algorytmu genetycznego
* Metoda mutowania algorytmu genetycznego

**public class** Population {  
  
 *// Holds population of tours* **private** Tour[] **tours**;

Klasa Populacji zawierająca tablicę ze wszystkimi podróżami

**public class** Tour **implements** Comparable<Tour> {  
  
 **private** ArrayList<City> **tour** = **new** ArrayList<>();  
 **private double fitness** = 0;  
 **private int distance** = 0;

Klasa Podróży zawierająca elementy składowe : listę wszystkich miast do odwiedzenia, współczynnik optymalizacji, całkowity dystans podróży.

**public class** City **extends** Point2D **implements** Comparable<City> {  
  
 **private final** Point2D **location**;

Klasa Miasta przechowująca jego położenie na płaszczyźnie 2D.

**public static** Tour crossover(Tour parent1, Tour parent2) {  
 Tour child = **new** Tour();  
 **int** startPos = (**int**) (Math.*random*() \* parent1.tourSize());  
 **int** endPos = (**int**) (Math.*random*() \* parent1.tourSize());  
  
 **for** (**int** i = 0; i < child.tourSize(); i++) {  
 **if** (startPos < endPos && i > startPos && i < endPos) {  
 child.setCity(i, parent1.getCity(i));  
 } **else if** (startPos > endPos) {  
 **if** (!(i < startPos && i > endPos)) {  
 child.setCity(i, parent1.getCity(i));  
 }  
 }  
 }  
  
  
 **for** (**int** i = 0; i < parent2.tourSize(); i++) {  
 **if** (!child.containsCity(parent2.getCity(i))) {  
 **for** (**int** ii = 0; ii < child.tourSize(); ii++) {  
 **if** (child.getCity(ii) == **null**) {  
 child.setCity(ii, parent2.getCity(i));  
 **break**;  
 }  
 }  
 }  
 }  
 **return** child;  
}

Metoda krzyżująca materiał genetyczny dwóch rodziców. Metoda wybiera dwa losowo wybrane punkty w genotypie pierwszego rodzica i tworzy z nich odcinek genotypowy. Materiał z tego odcinka jest kopiowany do dziecka które dopiero powstanie.

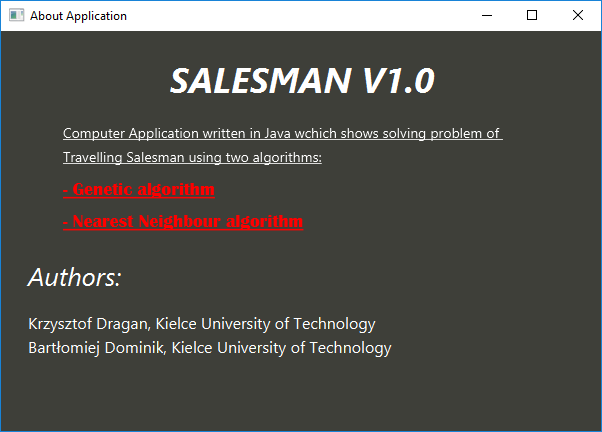
Długość genotypu dziecka jest taka sama jak rodziców , a więc po skopiowaniu materiału z pierwszego rodzica zostają puste miejsca gdzie jest dodawany materiał genetyczny z drugiego rodzica.

**private static void** mutate(Tour tour) {  
 **for** (**int** tourPos1 = 0; tourPos1 < tour.tourSize(); tourPos1++) {  
 **if** (Math.*random*() < ***mutationRate***) {  
 **int** tourPos2 = (**int**) (tour.tourSize() \* Math.*random*());  
 City city1 = tour.getCity(tourPos1);  
 City city2 = tour.getCity(tourPos2);  
  
 tour.setCity(tourPos2, city1);  
 tour.setCity(tourPos1, city2);  
 }  
 }  
}

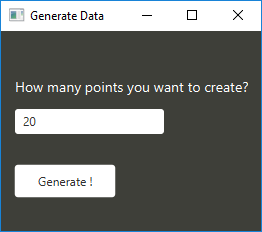
Mutacja elementu populacji. Jeśli wcześniej zdeklarowany współczynnik mutacji jest większy od wylosowanej wartości dokonywana jest mutacja polegająca na pobraniu losowo wybranych miast z Podróży i zamienieniu ich miejscami. Proces jest wtedy powtarzany dla wszystkich elementów Podróży.

# Opis działania aplikacji

Ekran startowy aplikacji, Wybieramy tutaj rodzaj algorytmu który będzie ukazywał rezultaty. Z tego menu należy przejść do sekcji File-> Generate New Input Points w celu wygenerowania punktów do pracy

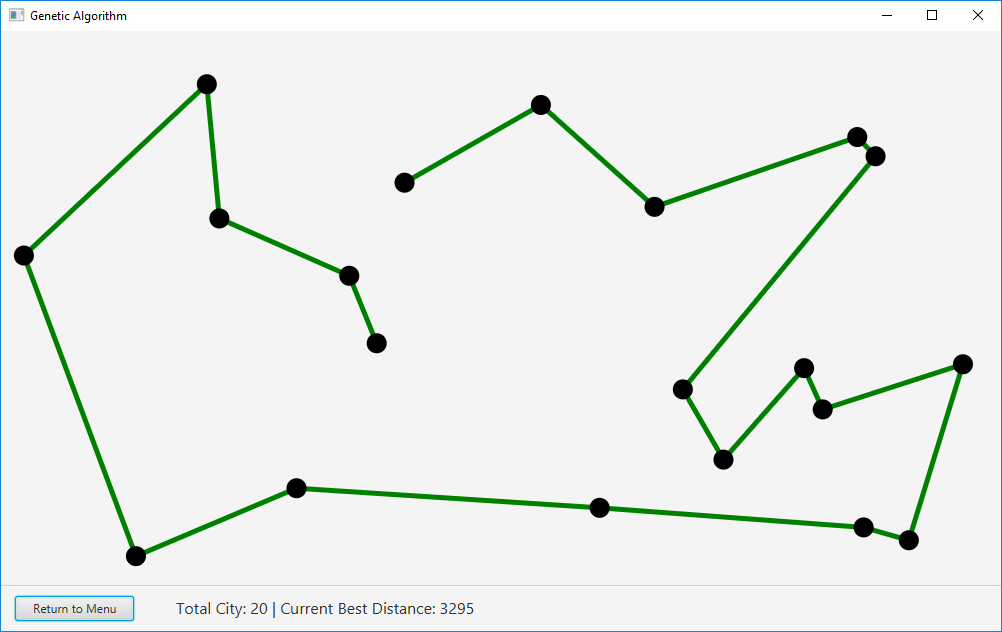
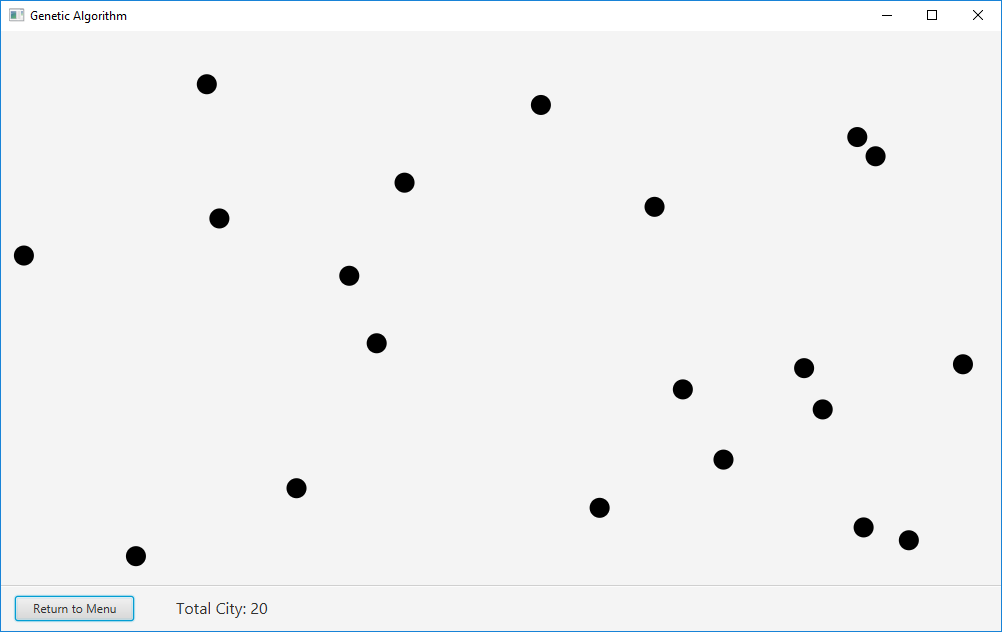


Ekran z informacją o przeznaczeniu i autorach



Generowanie punktów na którym będzie podejmowane rozwiązywanie problemu.

W tym oknie należy podać całkowitą liczbę miast które mają być stworzone.

Mapa z położeniami wygenerowanych punktów

Należy kliknąć dowolny klawisz aby uruchomić algorytm.

Mapa z zastosowaniem działania algorytmu genetycznego.

Po przetestowaniu działania program można zamknąć lub cofnąć się do menu przez zamieszczony w lewym dolnym rogu przycisk.

# Podsumowanie i wnioski

## Wnioski

Tworzenie aplikacji i rozwiązywanie napotkanych problemów zaowocowało następującymi wnioskami:

* Algorytm genetyczny jest znacznie wydajniejszy niż algorytm najbliższego sąsiada
* Obiektowe języki programowania bardzo dobrze nadają się do rozwiązywania problemów wykorzystujących algorytm genetyczny.
* Tworzenie graficznego interfejsu użytkownika przy pomocy narzędzi typu DragAndDrop znacznie ułatwia pracę.
* Problem komiwojażera znajduje szerokie zastosowanie w biznesie
* Algorytm genetyczny jest przykładem na to że Informatyka może przedstawić dowolny problem, w tym wypadku biologiczny na poziomie znacznie bliższym dla przeciętnego człowieka niż naukowe zapisy.

## Podsumowanie

Podczas poszukiwania rozwiązania musieliśmy przemyśleć co uznać za populację i element populacji. Następnym problemem było efektywna implementacja operacji krzyżowania i mutacji. Gdy zrealizowaliśmy główne części aplikacji mogliśmy je połączyć ze stworzonym już wcześniej interfejsem użytkownika. Dzięki użyciu środowiska programistycznego IntelIJ Idea znacznie zoptymalizowaliśmy procesy wytwórcze kodu źródłowego. Bardzo polecamy to środowisko każdej osobie która będzie pisać jakąkolwiek aplikację w Javie.

Aby uniknąć pułapek związanych z implementowaniem rozwiązania problemu komiwojażera bądź algorytmu genetycznego należy wcześniej przygotować sobie logiczny plan projektów i podzielić go na logiczne sekcje. Najlepiej zrobić to na kartce ponieważ w trakcie tego procesu wiele myśli może być odrzuconych i „skreślonych”.

Naszą aplikację można rozwinąć dodając nowe algorytmy rozwiązywania problemu komiwojażera. Ponadto zrealizowany algorytm genetyczny można ulepszyć zmieniając metody krzyżowania i mutacji na bardziej zaawansowane sposoby.

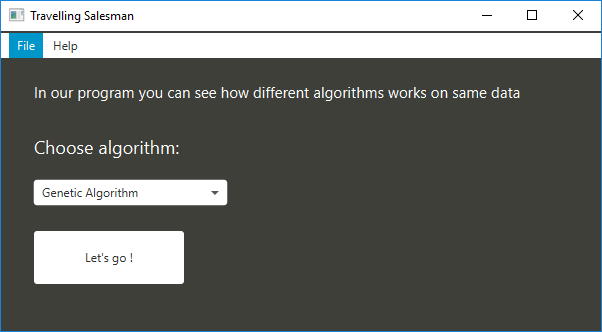
Na koniec dodamy że aplikacja może ukazać się w sieci przy użyciu technologii Java RMI.

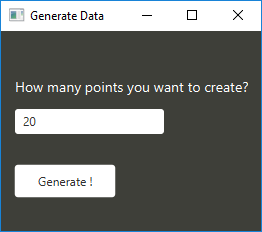
# Instrukcja obsługi aplikacji

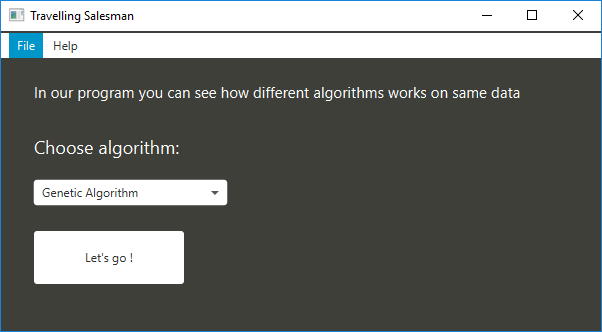
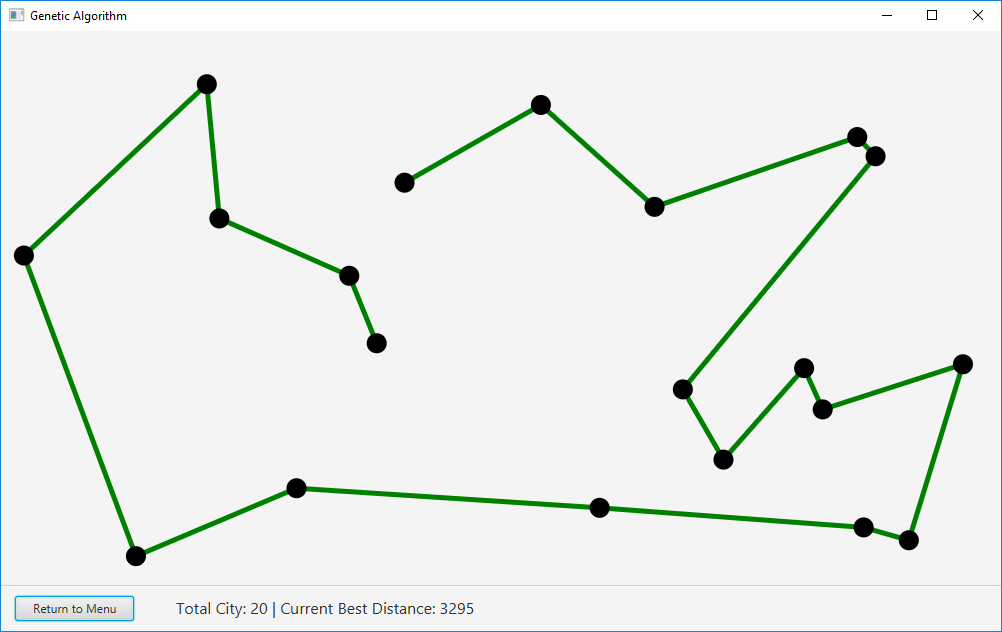
Aby uruchomić aplikację należy mieć zainstalowaną wirtualną maszyną Javy.

Jest ona instalowana z duża ilością oprogramowania więc istnieje duże prawdopodobieństwo że masz już ją zainstalowaną na swoim komputerze.

Maszyna wirtualna potrzebna jest do stworzonego przez nas i dołączonego do tej dokumentacji archuiwum wykonywalnego .jar . Należy na nie kliknąć dwa razy bądź z poziomu konsoli wykonać polecenia java –jar Salesman.jar

1. Ukaże nam się główne okno aplikacji:
2. Następnie należy przejść do sekcji File i wybrać opcje wygenerowania nowych punktów.



1. W tej części wpisujemy całkowitą liczbą i zatwierdzamy przyciskiem ‘Generate’
2. Okno wyboru ilości punktów zamyka się, przystępujemy teraz do wyboru algorytmu na którym będziemy pracować. Z dostępnej listy wybieramy algorytm i klikamy przycisk ‘Let’s go ‘
3. Ukaże nam się teraz okno narysowanymi punktami. Uruchamiamy algorytm dowolnym klawiszem klawiatury. Zostanie narysowana mapa z przebiegiem drogi. Klawisz można naciskać wielokrotnie aby zwiększyć liczbę populacji.

# Literatura

Literaturęnależy podać alfabetycznie, uporządkowaną według nazwisk autorów. Powinna ona zawierać zestaw wykorzystanych w pracy materiałów źródłowych: książek, czasopism naukowych, zeszytów naukowych, materiałów konferencyjnych, katalogów, norm, ewentualnie stron internetowych (nie więcej niż ¼ wszystkich pozycji literatury).

Przykład:

1. https://en.wikipedia.org/wiki/Travelling\_salesman\_problem
2. D.E. Goldberg: Algorytmy genetyczne i ich zastosowania. Wydawnictwa Naukowo-Techniczne, Warszawa 2003
3. MiNi, Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej, Warszawa 2000
4. Mirosław Wsciślik, Aleksander Jastriebow: Optymalizacja – teoria, algorytmy i ich realizacja w MATLAB-ie

# Ogólne wskazówki edycyjne

## Marginesy

* lewy (w przypadku druku dwustronnego wewnętrzny) - 3 cm,
* prawy (w przypadku druku dwustronnego zewnętrzny) - 2,5 cm.

## Czcionka

Praca powinna być napisana czcionką Times New Roman o wysokości 12 p.,   
z odstępem między wierszami 1,5.

## Numeracja stron

Strony powinny być liczone od strony tytułowej, ale numerowane od pierwszej strony tekstu pracy. Tytuły rozdziałów głównych(**1, 2, 3**, itd.) powinny być pisane czcionką pogrubioną i wersalikami. Rozdziały główne należy zaczynać od nowej strony.

## Numeracja wzorów, rysunków i tabel

* tytuły tabel i podpisy pod rysunkami należy pisać czcionką Times New Roma   
  o wysokości 10 p., z pojedynczym odstępem,
* numery wzorów powinny być dosunięte do prawego marginesu,
* rysunki, tabele, tytuły tabel, podpisy pod rysunkami i wzory należy wycentrować,
* pierwsza liczba - numer rozdziału głównego,
* druga liczba - numer kolejnego wzoru, rysunku, tabeli w danym rozdziale głównym, np.:

(2.1)

Tab. 2.1. Przykładowa tabela

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Rys. 2.1. Przykładowy rysunek.