WdGKiGO: toolbox

Celem poniższych zadań jest (między innymi) przygotowanie narzędzi do realizacji przyszłych projektów. Rozwiązanie zadań oznaczonych (!) będzie przydatne w przyszłych projektach.

Uwaga! Język programowania nie jest narzucony, jednak proszę ograniczyć się do Python, C++, C, Java (w tej kolejności). Na nasze potrzeby Python jest dobrym kompromisem.

Reprezentacja danych

Celem zadania jest przygotowanie struktur danych. Mogą to być własne projekty lub gotowe rozwiązania, np.: wbudowane lub biblioteczne. W tym drugim przypadku zalecam opakowanie istniejącej struktury we własny kontener z jednolitym interfejsem.

Przyjmujemy konwencje:

- **kolor** to wartość całkowitoliczbowa wyróżniająca rodzinę obiektów np.: odcinki niebieskie i odcinki czerwone.
  - W niektórych projektach może zajść potrzeba dodania do obiektu pola kolor a w niektórych będzie ono zbędne. Ustalmy, że wartość 0 oznacza brak koloru, 1 kolor niebieski, 2 kolor czerwony, 3 kolor zielony, kolejne numery oznaczają kolory użytkownika
- **punkty** są reprezentowane przez parę liczb zmiennoprzecinkowych i kolor: `(px,py,c)`; dodatkowo dodajemy specjalne punkty
  - `(nan,nan,0)`
  - `(-inf,inf,0)`
  - wartości nan i inf są zdefiniowane dla arytmetyki zmiennoprzecinkowej przez standard IEEE, w Pythonie są dostepne w pakiecie `numpy`
- **odcinki** reprezentowane są przez parę punktów i kolor: `((px,py),(qx,qy),c)`
- **odcinki skierowane** reprezentowane są przez parę punktów, pierwszy punkt oznacza początek a drugi koniec
- **wielokąt** reprezentowny jest przez listę swoich wierzchołków i kolor
- **strona odcinka skierowanego** lewa=`-1`, na odcinku = `0`, prawa=`1`

Uwaga: co prawda wydajność operacji na strukturze danych ma duże znaczenie, ale dla uproszczenia proszę potraktować to jako sprawę drugorzędną. Dla tego przedmiotu wystarczy, gdy będzie to zwykła tablica dynamiczna.

Uwaga: Może się okazać, że zajdzie potrzeba modyfikacji utworzonych struktur np.: dodania jakiejś operacji.

Uwaga: słowo lista w tym zadaniu, nie musi oznaczać listy w sensie struktury danych.

Zadanie 1.

1. Opracować struktury danych, które będą służyć do przechowywania:
   - (!) uporządkowanej listy punktów
   - (!) uporządkowanej listy odcinków skierowanych
   - (!) uporządkowanej listy wierzchołków wielokąta
   - (!) listy trójek złożonej z dwóch odcinków i punktu: `(seg1, seg2, p)`,
     - punkt p oznacza punkt przecięcia odcinków,
     - w przypadku, gdy się nie przecinają jego wartość jest równa `(nan,nan)`
     - w przypadku, gdy się przecinają w wiecej niż jednym punkcie jego wartość jest równa `(-inf,inf)`
   - (!) listy trójek złożonej z odcinka, punktu i strony: `(seg, p, s)`
   - uporządkowanej listy wielokątów
   - uporządkowanej listy okręgów
2. Każda ze struktur danych powinna udostepniać operacje:
   - `Add*` : dodawania na początek, na koniec i po określonym elemencie
   - `Del` : usuwania określnego elementu
   - `Find` : stwierdza czy element jest przechowywany w strukturze, zwraca pozycję tego elementu lub informacje, że nie ma
   - `IsEmpty` : sprawdza czy struktura zawiera elementy

- `Size` : podaje ilość elementów przechowywanych w strukturze

3. Każda struktura powinna umożliwiać dynamiczą zmianę rozmiaru

Operacje I/O

Celem tego zadania jest opracowanie operacji I/O. W projektach będzie wymagane, aby dane do algorytmu były odczytywane z pliku o dowolnej nazwie, podobnie wynik działania algorytmu. Aby ujednolicić interfejs ustalamy następujacy format :

Format pliku (v. 1.0)

- pliki z danymi są plikami tekstowymi, dla prostoty z rozszerzeniem `txt`
- dane są pogrupowane w sekcje, każda sekcja zaczyna się linią zawierającą typ sekcji w nawiasach `[ ]`
- dozwolone nazwy sekcji:
  - `[points c]` lista punktów koloru `c`
    - każdy wiersz zawiera parę liczb
  - `[segments c]` lista odcinków skierowanych koloru `c`
    - każdy wiersz zawiera cztery liczby, kolejno współrzędne początku i końca
  - `[polygon c]` lista wierzchołków wielokąta koloru `c`
    - każdy wiersz zawiera współrzędne jednego wierzchołka
  - `[intersections]` lista par odcinków z kolerem i punktem przecięcia
    - każdy wiersz zawiera wsp. odcinka1 kolor1 wsp. odcinka2 kolor2 punkt przeciecia
    - jeżeli odcinki się nie przecinają to punkt przeciecia ma wartość `nan nan`
    - jeżeli odcinki przecinaja się w więcej niż jednym punkcie, to punkt przecięcia ma wartość `-inf inf`, a nastepna linia zawiera wpis `seg seg nan nan`, gdzie `seg` oznacza część wspólną odcinków
  - [info] dowolne dane tekstowe zawierające dodatkowe informacje
  - sekcje zdefiniowane przez użytkownika
- pierwsza linia każdej sekcji zaczyna sie znakiem komentarza # i zawiera opis zawartości sekcji
- po nazwie sekcji nastepuje lista obiektów, jeden obiekt w jednej lini
- wartości pól każdego obiektu zapisywane są jako kolejne wartości bez nawiasów
- sekcja kończy się, gdy
  - rozpocznie się kolejna sekcja
  - zostanie napotkany koniec pliku
  - zostanie napotkana pusta linia

Zaleca się:

- rozważne dodawanie nowych sekcji i ich dobre dokumentowanie, domyślnie powinny być odczytywane tylko sekcje wymienione wyżej, zaś niestandardowe ignorowane
- wartości zmiennoprzecinkowe powinny być zapisywane w formacie +xx.yyy tzn. pole na znak, dwie cyfry, kropka, trzy cyfry, z pominięciem zer na końcu i początku) i były odzielone jedną spacją. Pogorszy to dokładność, ułatwi ,,naoczną'' analizę
- odzielanie sekcji pustą linią
- wyrozumiałość

Przykładowy plik

```
[info]
# Przykładowy plik z danymi

[points 0]
# Zestaw nr 3
 -0.001   0.5
-10.5    10.5
  1.03  -20.0
 13.5    13.45
  0.4    -2.75
```

```
[points 2]
# Zestaw czerwony
-11.8    -5.0
 -6.0   -10.65
  1.2   -19.4
-11.8    -7.0


[segments 1]
# Zestaw testowy 1a
  1.0     4.0     0.0   -12.0
 10.0    12.0    -0.667   0.667
 -0.667 -20.667 -11.0     0.0
-14.0     2.0    -2.0     2.0
 -4.0     2.0    -3.0     0.0
[intersections]
# przykładowe punkty przecięcia
 -1.0     0.0     1.0     0.0   1   0.0    -1.0     0.0     1.0   2   0.0     0.0
  2.0     0.0     3.0     0.0   1   0.0    -1.0     0.0     1.0   2     nan     nan
 -1.0     0.0     1.0     0.0   1   0.0     0.0     2.0     0.0   2    -inf     inf
  0.0     0.0     1.0     1.0   0   0.0     0.0     1.0     1.0   0     nan     nan
[info]
# format sekcji intersection
odcinek 1                        odcinek 2                        punkt przecięcia
sxx.yyy sxx.yyy sxx.yyy sxx.yyy c sxx.yyy sxx.yyy sxx.yyy sxx.yyy c sxx.yyy sxx.yyy

- pierwsza linia zwykłe przecięcie
- druga linia brak przecięcia
- trzecia i czwarta linia nakładanie się odcinków

[polygon 1]
# otoczka wypukła dla zestawu TEST 1
  0.0   0.0
  9.0  11.0
  3.0   0.0
 -5.0   2.0
 -4.0   9.0
 -7.0   5.0
[info]
# statystyki dla zestawu TEST 1
Algorytm naiwny
t = 0.943ms,
Liczba punktów wejsciowych:13
Liczba punktów wyjsciowych: 6

[whichside]
# sekcja zdefiniowana przez użytkownika, format: segment point side
 -1.0     0.0     1.0     0.0     0.0     1.0   -1
 -1.0     0.0     1.0     0.0     0.0    -1.0    1
 -1.0     0.0     1.0     0.0     0.0     0.0    0
```

Zadanie 1.

Zaimplementować następujące funkcje:

1. `DataInfo` wczytuje nazwy sekcji z pliku, wraz z komentarzem. W przypadku sekcji `[info]` całą zawartość sekcji bez końcowych pustych lini
2. (!) `DataRead` wczytuje zawartosc pliku do odpowiednich struktur danych
3. (!) `DataWrite` zapisuje wybrane dane do pliku
4. funkcje pomocnicze np.: wypisujące na konsolę odczytane informacje

Dla uproszczenia, można założyć, że pliki mają prawidłową strukturę, jednak zgodnie z dobrymi praktykami należałoby wyposażyć każdą funkcję w jakiś mechanizm kontroli błędów.

Uwaga: w rzeczywistym świecie zwykle nie tworzy się własnego formatu pliku, tutaj świetnie pasował by XML albo JSON, proszę traktować to zadanie szkoleniowo, ale wiecej tak nie robić

Zadanie 2.

Napisać program który wczytuje z pliku dane i rysuje odpowiadajace im obiekty na ekranie.

Elementarne funkcje

Zadanie 1. (!)

1. Napisać funkcję `SortLex(v,ord)` która sortuje listę punktów leksykograficznie zgonie z podanym porządkiem `ord`.
2. parametr 'ord' jest łańcuchem znakowym i koduje kolejność sortowania wg schematu:
   - LD - najpierw od lewej potem od dołu
   - LU - najpierw od lewej potem od góry
   - RD - najpierw od prawej potem od dołu
   - itd.
3. Dostępne powinno być przynajmniej sortowanie *LU: od lewej od góry* oraz *UL: od góry od lewej*
4. Napisać program który wczytuje z pliku listę punktów, sortuje je leksykograficznie i zapisuje do (innego) pliku wynik sortowania
5. (*) Wyświetlić na ekranie posortowane punkty wraz z etykietami oznaczającymi kolejność

Zadanie 2. (!)

1. Napisać funkcję `CCW(p,q,r)` która sprawdza czy trzy różne punkty `p,q,r` (w tej kolejności) tworzą zakręt w lewo.
2. Napisać program, który:
   - wczytuje z pliku listę punktów
   - dla każdej trójki kolejnych odcinków znajduje punkt przeciecia
   - otrzymane wyniki zapisuje do pliku.
3. (*) Wyświetlić na ekranie łamane o trzech wierzchołkach i pokolorować je w zależności od kierunku zakrętu

Zadanie 3. (!)

1. Napisać funkcję `WhichSide(seg,p)` sprawdzającą po której stronie odcinka leży punkt. Rozumiemy to tak, że sprawdzamy po której prostej zawierającej odcinek `seg` leży punkt `p`. Kierunek prostej jest zgodny z kierunkiem odcinka.
2. Napisać program, który:
   - wczytuje z pliku listę punktów i odcinek
   - dla każdego punktu sprawdza po której stronie odcinka leży ten punkt
   - otrzymane wyniki zapisuje do pliku.
3. Napisać program, który:
   - wczytuje z pliku listę odcinków i punkt
   - dla każdego odcinka sprawdza po której stronie odcinka leży ten punkt
   - otrzymane wyniki zapisuje do pliku.
4. (*) Wyświetlić na ekranie proste i punkty, pokolorować punkty w zależności od strony po której leżą

Zadanie 4. (!)

Przy założeniu, że odcinki przecinają się w co najwyżej jednym punkcie (przypadek niezdegenerowany) zaimplementować funkcje:

1. `IsIntersection` sprawdza czy dwa odcinki się przecinają
2. `Intersection` znajduje punkt przecięcia dwóch odcinków
3. (*) Rozszerzyć funkcjonalność na przypadek zdegenerowany
4. Napisać program, który:
   - wczytuje z pliku listę odcinków
   - dla każdej pary kolejnych odcinków znajduje punkt przeciecia
   - otrzymane wyniki zapisuje do pliku.
5. (*) Wyświetlić na ekranie odcinki i znalezione punkty przecięcia

Zadanie 5.

Napisać funkcję, która przetwarza nieposortowaną listę krawędzi wielokąta na listę posortowaną w kierunku CW lub CCW. Napisać program, który:

1. wczytuje z pliku nieposortowaną listę krawędzi wielokąta
2. wyznacza wierzchołki tego wielokąta
3. zapisuje do pliku posortowaną w kierunku CW listę wierzchołków i krawędzi
4. (*) rysuje otrzymany wielokąt etykietyjąc kolejne krawędzie