Wrocław, 01.02.2022r.

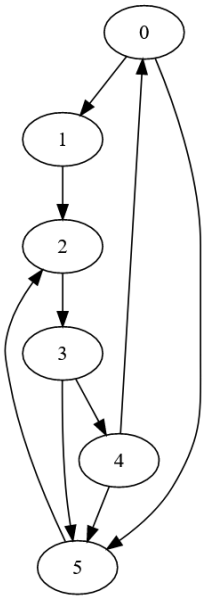
Lista zadań nr 7

Julia Mazur, 262296

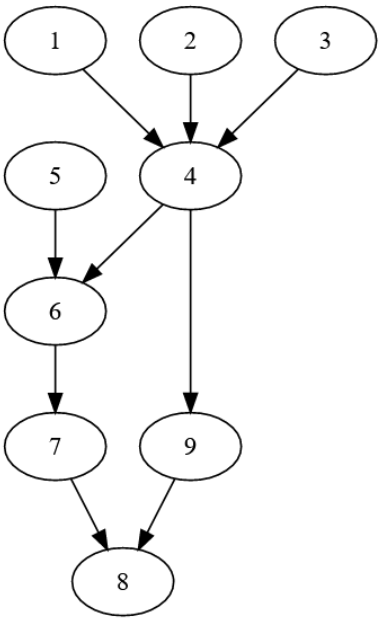
Zadanie 1, 2.

Do klasy *Graph* dodałam metodę *graph_dot* generującą reprezentację w języku dot.
Przykłady działania (te same grafy są używane w przykładach do kolejnych zadań):

```
graph 1:

digraph G{0->1;0->5;1->2;2->3;3->4;3->5;4->0;4->5;5->2;}
```




```
graph 2:

digraph G{1->4;2->4;3->4;4->6;4->9;5->6;6->7;7->8;9->8;}
```



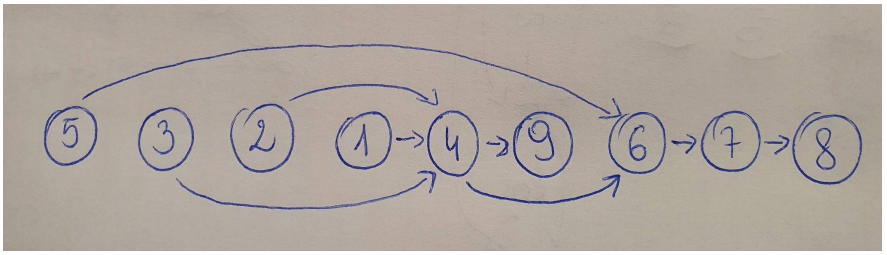
Zadanie 3, 4.

Graf został rozszerzony o metody przeszukiwania wszerz i w głąb. Dodałam metodę *sort_top,* która zwraca kolejność węzłów w grafie posortowanym topologicznie, jeśli jest to możliwe, jeśli nie rzuca to wyjątkiem ValueError.

Przykłady działania:

```
graph 1:

Traceback (most recent call last):
  File "./1.py", line 470, in <module>
    print(gr1.sort_top())
  File "./1.py", line 299, in sort_top
    raise ValueError('graph is not linear')
ValueError: graph is not linear
```

```
graph 2:

[5, 3, 2, 1, 4, 9, 6, 7, 8]
```

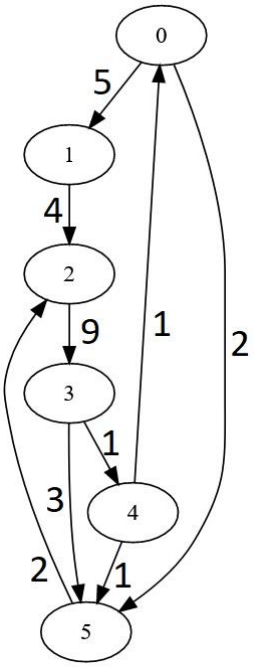


Zadanie 5.

W oparciu o przeszukiwanie wszerz napisałam funkcję zwracającą słownik z najkrótszymi ścieżkami od danego węzła do pozostałych.

Przykłady działania:

```
graph 1:
base: 2

0 :  [2, 3, 4, 0]
1 :  [2, 3, 4, 0, 1]
3 :  [2, 3]
4 :  [2, 3, 4]
5 :  [2, 3, 4, 5]

base: 5

0 :  [5, 2, 3, 4, 0]
1 :  [5, 2, 3, 4, 0, 1]
2 :  [5, 2]
3 :  [5, 2, 3]
4 :  [5, 2, 3, 4]
```



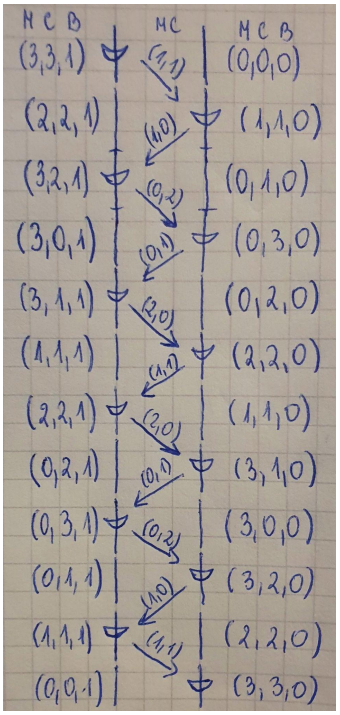
```
graph 2:
base: 3

4 :  [3, 4]
6 :  [3, 4, 6]
7 :  [3, 4, 6, 7]
8 :  [3, 4, 9, 8]
9 :  [3, 4, 9]

base: 8

Traceback (most recent call last):
  File "./1.py", line 468, in <module>
    result4 = find_fastest(gr2,8)
  File "./1.py", line 338, in find_fastest
    raise ValueError('there are no connections from this element')
ValueError: there are no connections from this element
```

Zadanie 6.

Aby rozwiązać przedstawiony problem skorzystałam z grafu i przeszukiwania wszerz aktualną sytuację opisuję krotkami :
$(ilość\ misjonarzy,\ ilość\ kanibali,\ nr\ brzegu\ przy\ którym\ jesteśmy\ (1\ -\ początkowy))$
połączenia między wszystkimi możliwymi stanami otrzymujemy dzięki założeniom problemu, czyli ilość kanibali nie może być większa od ilości misjonarzy (poza sytuacją gdzie ilość misjonarzy to 0). Po utworzeniu wszystkich połączeń wystarczy znaleźć najkrótszą ścieżkę do $(3, 3, 0)$, czyli stanu, w którym wszyscy są po drugiej stronie.
Po wywołaniu programu otrzymujemy (funkcja *mis_can*):

```
[(3, 3, 1), (1, 1, 0), (3, 2, 1), (0, 3, 0), (3, 1, 1), (2, 2, 0), (2, 2, 1),
 (3, 1, 0), (0, 3, 1), (3, 2, 0), (1, 1, 1), (3, 3, 0)]
```

Czego graficzna interpretacja wygląda następująco:



Czyli potrzebne jest 11 przepłynięć.

Zadanie 7.

Ten program działa na podobnej zasadzie do poprzedniego. Sytuację zapisuję w postaci krotki:

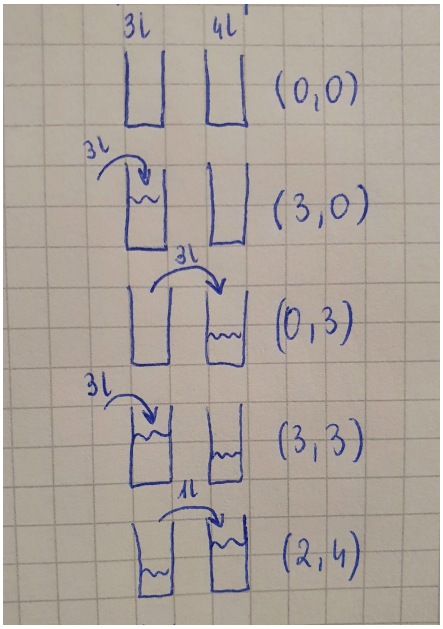
$(ilość\ litrów\ w\ pierwszym\ kanistrze, ilość\ litrów\ w\ drugim\ kanistrze)$

Połączenia tworzymy zgodnie z możliwościami przelewania, wylewania i dolewania. Znów szukamy najkrótszej ścieżki od stanu początkowego, czyli $(0, 0)$ do dowolnej krotki z dwójką na którejś pozycji.

Po wywołaniu programu otrzymujemy(funkcja *litres*):

```
[(0, 0), (3, 0), (0, 3), (3, 3), (2, 4)]
```

Czego graficzna interpretacja wygląda następująco:



Zatem należy wykonać cztery ruchy.

Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/39820f4d08e5d05a296c909813ec89a9fa84ebfb/lista7/1.py