Kurs Pythona v0.2

Lista zadaniowa VI

Wojciech Adamiec

10 stycznia 2023

Spis treści

1 Recenzenci

Chcemy napisać skrypt, który wygeneruje nam przyporządkowanie recenzentów do recenzowanych w ramach danej listy zadań. Takie przyporządkowanie musi spełniać 3 reguły (przy założeniu, że każda osoba jest jednocześnie recenzentem i recenzowanym):

- Nie można być swoim własnym recenzentem.
- Nie można recenzować tej samej osoby dwie listy z rzędu.
- Nie można być recenzentem swojego recenzenta.

Jedną z najłatwiejszych w implementacji metod do rozwiązywania takich problemów (chociaż definitywnie nie najbardziej optymalną) jest metoda *szczęśliwego trafu*. Polega ona na generowaniu losowych przyporządkowań tak długo, aż natrafimy na takie, które będzie spełniać wszystkie warunki zadania.

Twoim zadaniem jest napisanie procedury `generate_assignments(previous_assignments, coders)`, która na wejściu przyjmuje przyporządkowania z poprzedniej listy `previous_assignments` w postaci słownika recenzowany - recenzent oraz listy osób z aktualnej listy zadań - `coders`, a na wyjściu zwraca przyporządkowania dla aktualnej listy zadań w tej samej postaci co `previous_assignments`.

Uwaga! Nie wolno Ci modyfikować oryginalnego słownika oraz listy, które są przekazywane do funkcji w postaci argumentów. Weź pod uwagę, że poprzednie przyporządkowanie mogło zawierać osoby, których nie ma już na kursie.

Wskazówka: Wygenerowanie losowego przyporządkowania może być bardzo proste. Możesz w tym celu wykorzystać funkcję `shuffle` z modułu `random`.

2 Metoda D'Hondta

W Polsce obowiązuje ordynacja wyborcza D'Hondta - powiedział dziennikarz znanej stacji telewizyjnej po czym w kilku zdaniach udowodnił, że nie ma pojęcia o czym mówi.

W jaki sposób wybieramy w Polsce swoich reprezentantów do Sejmu? Szczegóły możemy znaleźć w ustawie z 12 kwietnia 2001 roku zatytułowanej *Ordynacja wyborcza do Sejmu Rzeczypospolitej Polskiej i do Senatu Rzeczypospolitej Polskiej*, której treść można znaleźć tutaj: ISAP: Ustawa.

Tłumacząc owy dokument z prawniczego na informatyczno-sarkastyczny dostajemy:

- Polska podzielona jest na 41 niezależnych okręgów wyborczych różnych rozmiarów (pod względem ilości mandatów).
- Zainteresowani (partie polityczne lub ruchy obywatelskie) tworzą komitety wyborcze i wystawiają listy kandydatów w okręgach wyborczych (zgodnie z ideą powszechności wyborów samodzielny start jest niemożliwy).
- Wyborcy głosują na konkretnego kandydata (oraz symultanicznie na komitet, którego jest reprezentantem).
- Wyrzucamy do śmieci wszystkie głosy oddane na komitety, które w skali kraju nie uzyskały 5% poparcia - dla partii politycznych lub 8% poparcia - dla komitetów koalicyjnych. Z progu wyborczego zwolnione są komitety mniejszości narodowych. (Ważne, aby wyrzucić głosy wyborców zgodnie z demokracją, w 2015 roku takich głosów było ponad 2.5 miliona).
- Pozostałe głosy (liczone jako suma głosów oddanych na dany komitet wyborczy) są niezależnie w każdym z 41 okręgów przeliczane wg metody D'hondta na odpowiednią dla danego okręgu liczbę mandatów - mówimy tutaj o podziale mandatów z danego okręgu na komitety wyborcze.
- Mandaty przyznane konkretnemu komitetowi wyborczemu przypadają osobom, które zdobyły najwięcej głosów na danej liście.

Jak nietrudno zauważyć sformułowanie *Mamy w Polsce system D'Hondta* zdecydowanie nie wyczerpuje tematu, gdyż nasza ordynacja wyborcza korzysta z owego systemu jedynie przy przeliczaniu głosów oddanych na dany komitet w danym okręgu na mandaty zdobyte przez ten komitet w tym okręgu. Na czym dokładnie polega taki podział?

Najlepiej zobaczyć to na animacji lub przeczytać na spokojnie w artykule.

W tym zadaniu będziemy chcieli zweryfikować czy PKW (Państwowa Komisja Wyborcza) poprawnie przeliczyła zdobyte przez komitety głosy na zdobyte mandaty. Dla uproszczenie będziemy zajmowali się jedynie pojedynczymi okręgami wyborczymi, a w danych zawierających oddane głosy nie będzie już głosów uznanych za nieważne lub oddanych na komitety, które nie przekroczyły progu wyborczego.

Twoim zadaniem jest napisanie funkcji `get_seats(seats_number, votes_data)`, która przyjmuje liczbę mandatów do wyznaczenia oraz wyniki głosowania w postaci słownika komitet - liczba głosów oraz zwraca podział mandatów w postaci słownika komitet - liczba mandatów. Przykładowe wywołanie tej funkcji może wygladać np. tak (dane z okręgu 30):

```
DATA = {
    "PiS": 161160,
    "PO": 92493,
    "SLD": 32300,
    "Konfederacja": 23939,
    "PSL": 18816,
}

    get_seats(9, DATA) ->

  {
      "PiS": 5,
      "PO": 3,
      "SLD": 1,
      "Konfederacja": 0,
      "PSL": 0,
  }
```

Po napisaniu poprawnie działającej funkcji do wyznaczania mandatów zerknij na stronę wyników wyborów z 2019 roku, znajdź swój okręg wyborczy do sejmu i upewnij się, że PKW Cię nie oszukała.

3 Gra w życie

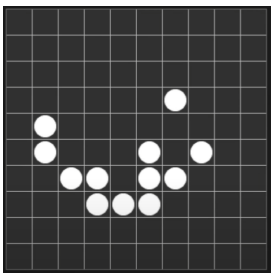

Gra w życie jest popularnym automatem komórkowym - czyli grą bezosobową (po ustaleniu warunków startowych gra toczy się już sama), która dzieje się w systemie komórkowym (podobnym do arkusza kalkulacyjnego), w którym komórki sąsiadują ze sobą zgodnie z pewnym ustalonym wzorcem. Stan komórki zmieniany jest synchronicznie (w każdej turze/epoce) zgodnie z regułami mówiącymi, w jaki sposób nowy stan komórki zależy od jej obecnego stanu i stanu jej sąsiadów.

W grze w życie mówimy o zaledwie dwóch stanach komórek - komórce martwej (pustej) i komórce żywej (domku). Sąsiadem każdej komórki jest komórka stykająca się z nią bokiem lub wierzchołkiem (pomijając komórki na skraju mapy, każda komórka ma 8 sąsiadów).

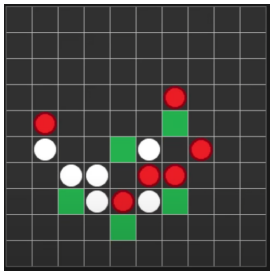
W każdej epoce synchronicznie dzieją się dwie rzeczy:

- Martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa (Kolonia się rozwija).
- Żywa komórka przy innej liczbie żywych sąsiadów niż 2 lub 3 umiera (Odpowiednio z osamotnienia lub przeludnienia).

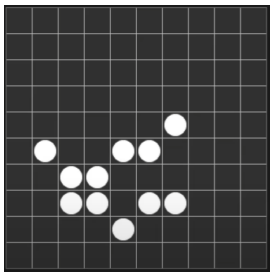
Przykładowo dla startowego ułożenia (białe kółka - żywe komórki, czarne puste pole - martwe komórki):



Nastąpią takie zmiany (zielone - miejsca narodzenia nowych żywych komórek, czerwone - miejsca śmierci starych komórek):



Co ostatecznie daje taki stan w następnej epoce:



Będziemy modelowali grę w życie za pomocą macierzy (listy list) rozmiaru `n` na `m`, której elementami są znaki `"X"` (żywa komórka) lub `"."` (martwa komórka).

Napisz dwie procedury, które będą grać w życie. Pierwsza z nich `play(starting_board, turns)` powinna zwracać stan planszy po rozegraniu `turns` tur gry w życie (bez używania funkcji `print`).

Druga z kolei `play_forever(starting_board, interval)` powinna grać w życie w nieskończoność, a stan planszy po każdej kolejnej epoce za pomocą funkcji `info` rysować na terminalu. Między każdymi dwoma epokami powinna nastąpić pauza długości `interval` pozwalająca na przyjrzenie się zmianom na planszy.

Wskazówka: W celu zrobienia pauzy wykorzystaj funkcję `sleep` z modułu `time`.