Wstęp do programowania
Pracownia 9

Data publikacji 2.12.2020

Uwaga: Premia za tę listę wynosi 0.5, przyznawana jest osobom, które zdobyły co najmniej 2p za zadania z tej listy. Data publikacji specjalnie jest nieco późniejsza, żeby dodać trochę 'oddechu'

Zadanie 1.(1pkt)⋆ Prezent od Mikołaja: rozwiąż wybrane stare zadanie, którego nie robiłeś. Możesz też wskazać jakieś zadanie, za które dostałeś 0.5p ze względu na oddanie w późniejszym terminie i poprosić o poprawienie wyniku o 0.5.

Zadanie 2.(1pkt) Napisz program, który wykorzystuje rekurencję do utworzenia ładnego rysunku. To może być kwadrat lub roślinka, która znajdowała się w pierwszym zestawie rysunków na SKOSie, trójkąt lub dywan Sierpińskiego, drzewo Pitagorasa, lub też dowolny fraktal, który znajdziesz w Internecie. Oczywiście powinien to być rysunek, którego nie robiłeś.

Zadanie 3.(1pkt) Wróćmy do zadania z przestawianiem literek z imienia i nazwiska. Zmodyfikujemy je bardzo nieznacznie: z zadanych literek należy ułożyć nie dwa, lecz trzy słowa. Oznacza to w szczególności, że rozwiązanie, w którym znajdujemy słowa układalne i sprawdzamy wszystkie pary takich słów przestaje być akceptowalne (bo sprawdzenie wszystkich trójek trwa zbyt długo). Wskazówka: czy znając dwa słowa musisz przeglądać **wszystkie** układalne, żeby znaleźć trzecie? Wskazówka 2 (rot13): Cbzlfy b fybjavxh, j xgbelz xyhpmnzv fn cbfbegbjnar yvgrel qnartb fybjn, n jnegbfpvn wrfg yvfgn fybj (wnxn?)

Zadanie 4.(1pkt) Program `wykresy_czasow.py` z Wykładu 8 rysuje wykresy z dużym „szumem". Twoim zadaniem jest zmniejszyć ten szum na dwa sposoby:

1) Wyznaczać wartość czasu działania jako minimum kilku pomiarów, ale wykonanych w znacznym odstępie czasu (żeby chwilowe zwolnienie komputera nie miało wpływu na wynik). Oczywiście czas pomiędzy pomiarami należy wykorzystać – najlepiej na robienie innych pomiarów.

2) Implementując funkcję wygładź(L), która zwraca listę N o tej samej długości, w której pierwsza i ostatnia wartość są kopiami odpowiednich wartości z L, natomiat dla pozostałych i mamy: `N[i] == 0.1*L[i-1] + 0.8 * L[i] + 0.1 * L[i+1]`. W celu lepszego wygładzenia, zastosuj tę funkcję kilkukrotnie.

Zaprezentuj na jednym rysunku 3 wykresy: zależność czasu działania rev1 i rev2 od wielkości danych (wygładzona zgodnie z punktem 1.) oraz efekt wygładzenia czasu rev1 metodą drugą. Sprawdź, co robi:

```
plt.plot(xs3, ys3, color='lightblue', linewidth=7, alpha=0.5)
```

Zadanie 5.(1pkt) Korzystając w dowolny sposób z programu zycie.py z wykładu 8 zaimplementuj inny automat komórkowy, mianowicie *Papier, nożyce, kamień*. Zasady są następujące:

1. Automat działa na prostokątnej planszy, wypełnionej kwadratowymi komórkami.
2. Komórki sąsiadują ze sobą jeżeli stykają się bokami (4-sąsiedztwo, inaczej niż w 'życiu'). Nie ma zawijania planszy, komórki przy bokach po prostu mają mniej sąsiadów.
3. Mamy cztery rodzaje pól: pola puste oraz pola zawierające komórkę typu papier, nożyce lub kamień.
4. Pola niepuste dodatkowo mają zapisaną siłę (liczbę od 1 do 5).
5. Przy przejściu do nowego stanu dla każdego zajętego pola wykonujemy następującą operację: wybieramy losowo sąsiada i następnie:
   a) Jeżeli sąsiadem jest pole puste, a nasza siła jest co najmniej 1, to „zasiedlamy" je z siłą o 1 mniejszą (czyli na przykład papier z siłą 4 spowoduje wpisanie na puste miejsce papieru z siłą 3).

b) Jeżeli sąsiadem jest pole naszego koloru to nic się nie dzieje.

c) Jeżeli sąsiadem jest pole innego rodzaju, to następuje pojedynek w wyniku którego przegrany traci jeden punkt siły, a zwycięzca zyskuje 1 (chyba że już ma maksymalną siłę). Pojedynek rozstrzygany jest zgodnie z zasadami oryginalnej gry: papier pokonuje kamień, kamień nożyce, a te papier (siła nie ma znaczenia)

d) Jeżeli w wyniku pojedynku siła pola zmaleje do zera, pole staje się puste.

6. Odczyt pola wykonywany jest na „starej wersji planszy", modyfikacje – na nowej, którą utworzyliśmy kopiując starą[1]

Napisz program, który implementuje te zasady. Podobnie jak w programie zycie.py stan początkowy powinien być definiowany przez wielolinijkowy napis.

Rady do testowania Przygotowując plansze testowe uwzględnij następujące kwestie:

1. Jeżeli na planszy początkowej jest kilka pól jednego rodzaju, wówczas powinny one zasiedlić swoją okolicę, po czym sytuacja się ustabilizuje.

2. Jeżeli są dwa rodzaje, wówczas albo się nie spotkają, albo „ten silniejszy" zje w całości słabszego (na przykład „kamień" zje „nożyce")

3. Rozważ wykonywanie testów za pomocą intepretera `pypy` (czyli zamiast pisać `python3 prog.py` pisz `pypy3 prog.py` lub `pypy prog.py`). Powinno to kilkukrotnie przyśpieszyć działanie.

4. Niektóre plansze generują dramatyczne rozgrywki. Przykładowo poniższa:

```
...kkkkkkkkkkk.......
....................
....................
....................
..nnnn..............
..nnnn..............
..n.................
....................
.........kkkkkkkkkkk..
....................
....................
....................
....................
.........ppppp........
....................
```

Jeżeli nożycom nie uda się wyrwać z okrążenia, stosunkowo szybko wygrywa papier. Jeżeli się uda, to gra jest dość długa i może się skończyć zwycięstwem każdej strony (lub być może trwać w nieskończoność).

[1] Rozważ użycie funkcji deepcopy z modułu copy