PDAJ zadatak za ocenjivanje

Problem koji se rešava:

- Generisati koordinate dvodimenzione table na osnovu prosleđenih vrednosti za veličinu table(n i m).
- Navesti niz koordinata polja koje se nalaze na tabli i koja predstavljaju specijalna polja. Broj polja u ovom nizu treba da bude znatno manji od ukupnog broja polja.
- Za svako od polja na tabli pronaći udaljenost do svakog od specijalnih polja.
- Pronaći najbliže specijalno polje svakom polju na tabli.
- Kao rezultat, za svako od polja vratiti indeks njegovog najbližeg specijalnog polja iz niza specijanih polja.
- Ako je neko polje jednako udaljeno od dva specijalna polja, vratiti indeks bilo kojeg od njih.

Zadatak rešiti na sledeće načine:

- sekvencijalno
- upotrebom list comprehension-a
- upotrebom generatora
- upotrebom multiprocessing biblioteke

Rešenje prvo kreirati zasebno za svaki od načina rešavanja u programskom jeziku Python, a zatim u obliku Django rest API-ja.

Za svaki od načina rešavanja kreirati odvojenu putanju (npr: calculation/sequential, /calculation/multiprocessing, ...). Svaka putanja prima ulazne podatke u istom obliku i na osnovu njih vraća izlazne podatke u istom obliku kao sve ostale putanje.

Ako API zahtev u JSON obliku i izgleda ovako:

```
{
	"n":"10",
	"m":"10",
	"points":["1,3", "3,2", "6,8", "9,6", "5,5"]
}
```

Očekivani rezultat bi bio:

```
{
	"result": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1,
	1, 1, 0, 0, 0, 0,
4, 2, 2, 1, 1, 1, 1, 1, 4, 4, 4, 2, 2, 1, 1, 1, 1, 4, 4, 4, 2, 2, 2, 1, 1, 1,
4, 4, 4, 4, 2, 2, 2, 1,
1, 1, 4, 4, 4, 4, 2, 2, 2, 1, 1, 4, 4, 4, 4, 3, 2, 2, 2, 1, 4, 3, 3, 3, 3, 3, 3,
2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
	"time_in_s": 0.0213773250579834,
	"max_memory_in_MB": 0.035714
```

}

Izabrati veličinu table tako da sekvincalno vreme izvršavanja bude barem 5 sekundi. Svakoj putanji proslediti iste koordinate specijalnih polja i veličinu table. Na osnovu dobijenih rezultata popuniti sledeće tabele:

	n	m	points
vrednost	1000	1000	["31,110","19,420","11,3", "3,21", "6,8", "9,16", "55","15,15","76,54","89,430","331,423","211,211","350,350","426,18","0,500","7,420","766,894","999,3","876,542","667,718","319,54","1,800","803,20"]

Način rešavanja	Vreme u s	Memorija u MB	Broj jezgara	Zaključak
sekvencijalno	121.3422	93.7272	1	Imajući u vidu da se program sekvencijalno izvršava I da se koristi samo jedno jezgro računara, performanse za manje dimenzije I manji broj specijalnih polja su odlične. U rangu su performansi generatora I paralelnog izvršavanja. Jedini problem je konzumacija RAM-a koja je mnogo veća od generatora I paralelnog izvršavanja. Kako se dimenzije povećaju, tako performanse postepeno opadaju.
comperhension	160.4321	93.7322	1	Ubedljivo najlošije performanse. I pored toga što je koda manje I što je sve čitljivije, performanse su loše. Koristi čak I malo više RAM-a od sekvencijalnog koda a vreme izvršavanja mnogo veće od sekvencijalnog. Vreme čak duplo duže od vremena generatora,a 3 puta duže od paralelnog izvršavanja. Ukoliko nam je bitna brzina Izračunavanja, bolje koristiti sekvencijalni kod bez list I dict comprehension-a.
generator	89.6762	8.4545	1	Vrlo efikasan način izračunavanja. U nekim situacijama pokazivao slične ili čak I malo bolje performanse u odnosu na paralelni kod. Nije previše brži od sekvencijalnog koda ali je utrošak RAM-a oko 11 puta manji od sekvenijalnog koda. Koristi samo jedno jezgro računara. Količina rama se povećava na 35MB kada su dimenzije 2000x2000 što svakako nije puno.
multiprocessing	60.6594	8.1342	4	Vremenski ubedljivo najbolji način izračunavanja. Dosta brži od generatora (29s), a koristi sličnu količinu RAM-a kada su dimenzije 1000x1000. Koristi sva 4 jezgra računara. Performanse se menjaju u odnosu na izabranu metodu (map,imap,imap_unordered). Probao sam sve metode, ali najbolje rezultate daje imap. Map čeka sve rezultate pa ih onda vraća, a imap_unordered vraća rezultate odmah koji ne moraju biti sortirani (što mi ne odgovara). Veliki značaj na performanse ima chunksize. Kritično je odabrati dobar chunksize koji nije prevelik a ni premali. Ja sam izabrao da on bude broj redova matrice I tako mi radi najbrže.

- Za računanje vremena izvršavanja proračuna koristiti Python biblioteku time.

- Za računanje maksimalnog zauzeća memorije pri izvršavanju proračuna koristiti Python biblioteku tracemalloc.
- Broj procesorskih jezgara zaključiti na osnovu načina rešavanja zadatka. Izvršavanje optimizovati za izvršavanje na svom računaru.
- Na osnovu dobijenih rezultata, doneti zaključak o prednostima i manama svakog načina rešavanja.
- Za multiprocessing obrazložiti i izbor funkcije koja je korišćena (map, imap, imap_unordered).

Za testiranje primera manjih dimenzija table koristiti `display_results.py`. Prikaz primera biće kreiran u 2 oblika:

- konzolni oblik:

```
   Time is  0.002613067626953125s
   Max memory is  0.036935MB
1  0  0  0  0  0  0  0  0  0
2  0  0  *  0  0  0  0  0  2
   1  1  1  0  0  0  0  4  2  2
   1  1  *  1  1  4  4  4  2  2
   1  1  1  1  4  4  4  2  2  2
   1  1  1  4  4  *  4  2  2  2
   1  1  1  4  4  4  4  2  *  2
   1  1  4  4  4  4  3  2  2  2
   1  4  3  3  3  3  3  3  2  2
   3  3  3  3  3  3  *  3  3  3
```

- grafički prikaz: