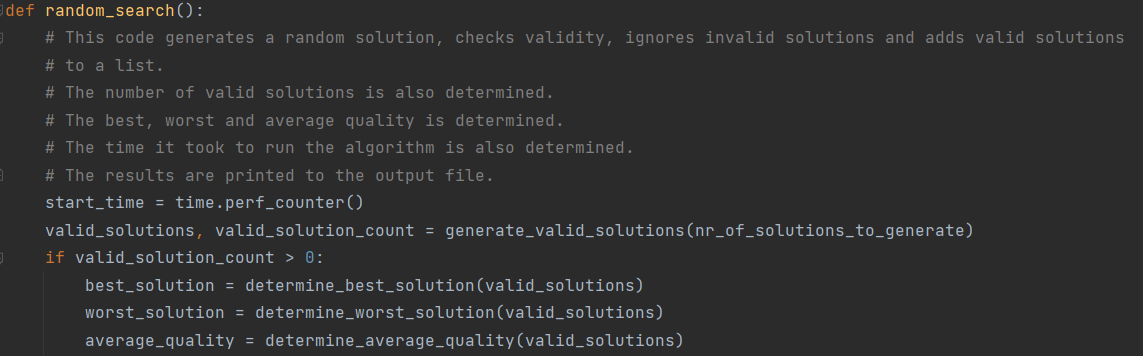
AI – Laborator 1

Pentru acest laborator a fost implementat o metoda de cautare aleatoare, si Random Hill-Climbing pentru problema rucsacului.

1. **Functii principale:**

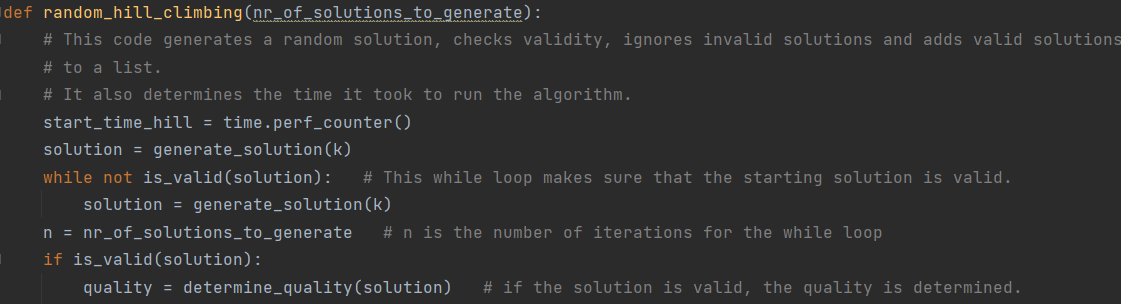
In functia **random\_search()**:



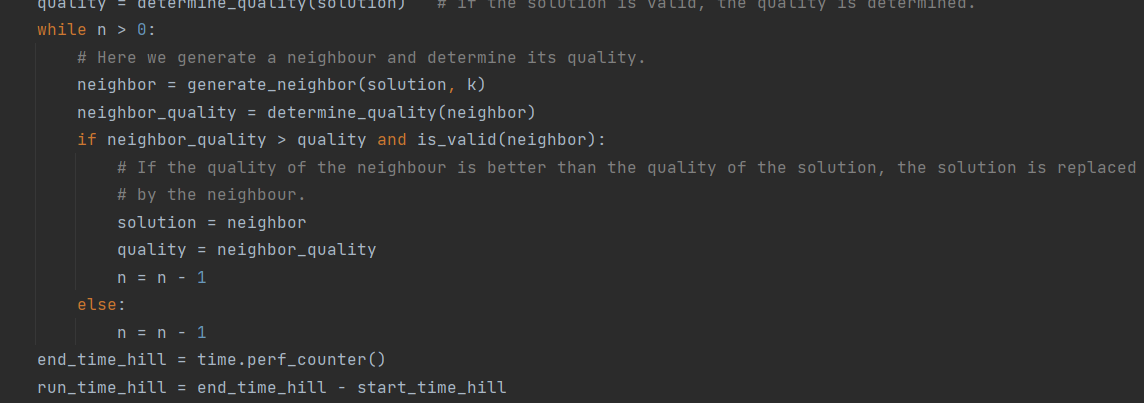


* A fost generata o lista de ‘n’ solutii valide
* Daca s-au gasit solutii valide, atunci se determina cea mai buna si cea mai rea solutie, precum si calitatea medie a solutiilor gasite.
* De asemenea, se va determina si timpul de rulare a acestui algoritm.

In functia **random\_hill\_climbing()**:



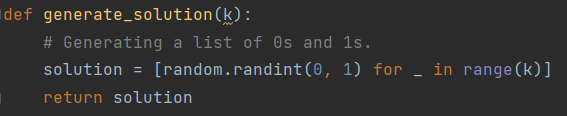
* Incepem prin a genera o solutie de lungime ‘k’ *(‘k’ de 0 si 1)*
* Verificam ca solutia de start sa fie valida, in caz contrar, generam alta solutie pana la gasirea unei solutii valide
* Daca solutia este valida, mergem mai departe si ii determinam calitatea



* Cat timp ‘n’(in acest caz n este numarul de vecini pe care vrem sa ii determinam) este mai mare ca 0, generam un vecin pentru solutia curenta, si in functie de calitatea(calitatea vecinului sa fie mai mare ca si calitatea solutiei curente) si validitatea lui, inlocuim solutia curenta cu vecinul determinat, sau ignoram vecinul
* De asemenea se determina si timpul de rulare al acestui algoritm

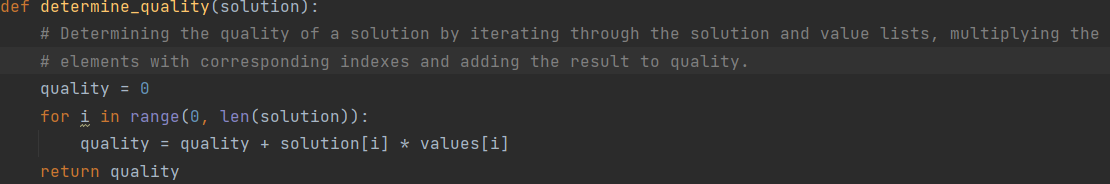
1. **Functii ajutatoare:**

In functia **generate\_solution(k)**:



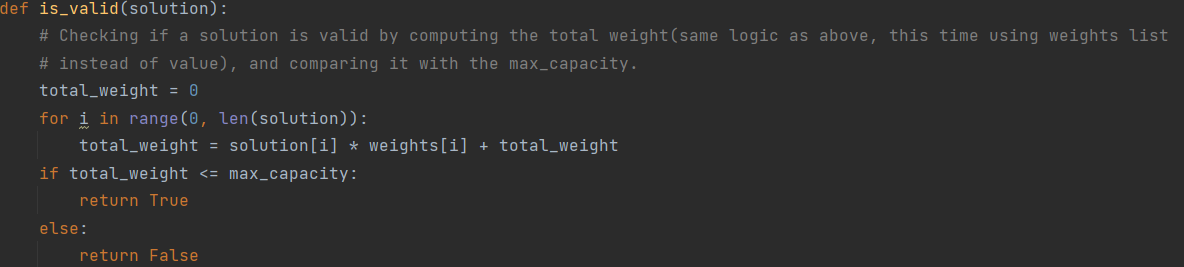
* Se genereaza lista de lungime ‘k’, formata doar din 0 si 1
* Se returneaza solutia

In functia **determine\_quality(solution)**:



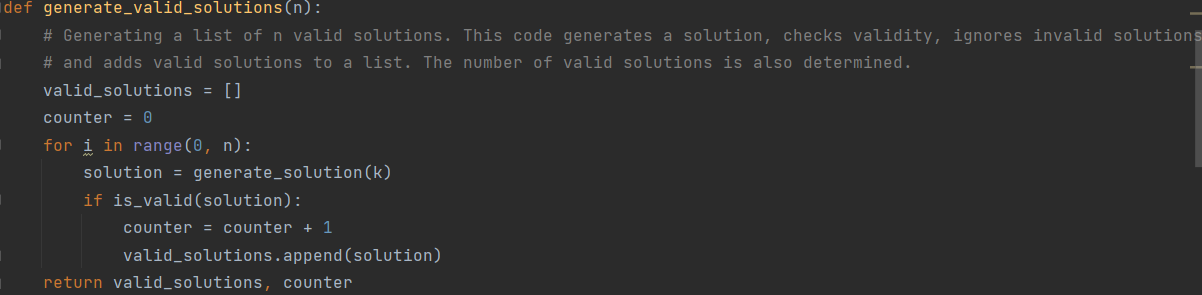
* Se determina calitatea unei solutii, prin inmultirea elementelor de pe indecsi corespunzatori a listelor ‘solution’ si ‘values’
* Se returneaza calitatea gasita

In functia **is\_valid(solution)**:



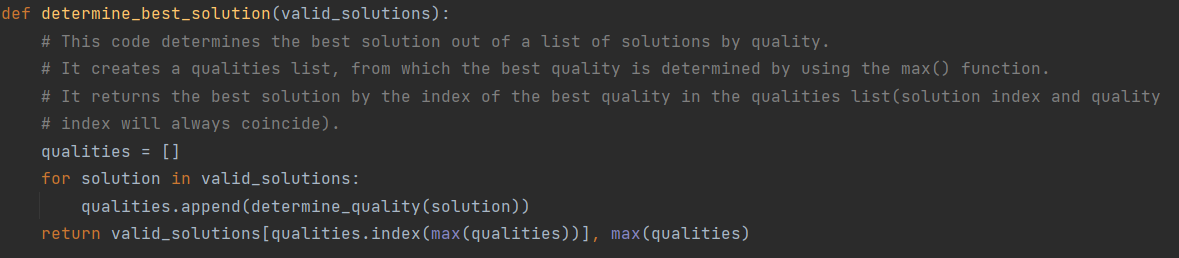
* Se determina validitatea unei solutii in functie de greutatea solutiei, prin inmultirea elementelor ca si mai sus, din listele ‘solution’ si ‘weights’.
* Daca greutatea totala a solutiei este mai mica sau egala decat capacitatea maxima a rucsacului, se returneaza True. In caz contrar se returneaza False

In functia **generate\_valid\_solutions(n)**:



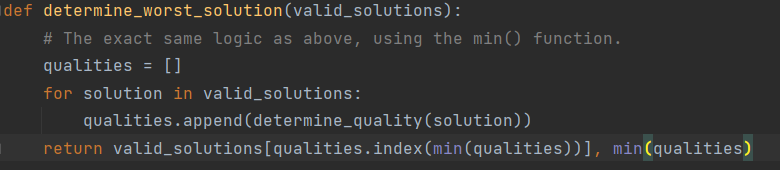
* Se genereaza o lista de solutii valide, si se numara totalul de solutii generate(si valide, si invalide)
* Se returneaza lista si numarul

In functia **determine\_best\_solution(valid\_solutions)**:



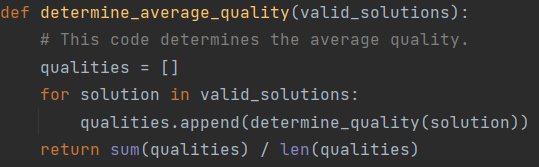
* Se afla cea mai buna solutie dintr-o lista de solutii valide, in funtie de calitatea solutiei. Se foloseste de functia max()
* Se returneaza solutia gasita si calitatea ei

In functia **determine\_worst\_solution(valid\_solutions)**:



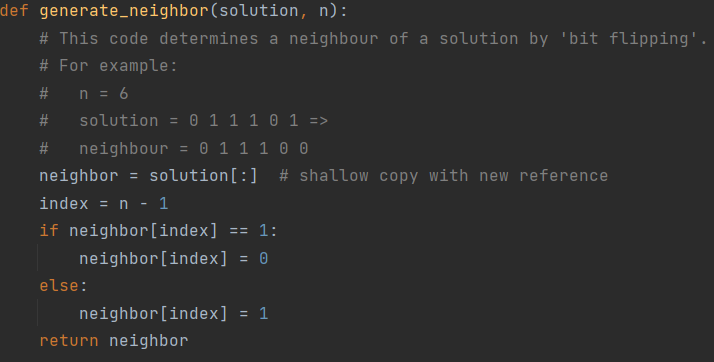
* Acelasi lucru ca si mai sus. Se foloseste functia min()

In functia **determine\_average\_quality(valid\_solutions)**:



* Se determina calitatea mediei dintr-o lista de solutii valide
* Se face media aritmetica a calitatilor si se returneaza

In functia **generate\_neighbour(solution, n)**:



* Se genereaza un vecin, in functie de solutia data si n-lungimea solutiei
* Se foloseste metoda de ‘bit flipping’, adica 0->1 sau 1->0

1. **Concluzii si analiza rezultatelor:**

Setul 1 de valori:

max\_capacity = 1000

weights = [56, 121, 200, 5, 343, 65, 23, 434, 150, 90]

values = [333, 34, 1231, 6, 44, 1222, 543, 522, 999, 10000]

k = 10

**RANDOM SEARCH (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Medie best quality | 13243.7 | 14176.6 | 14475.5 |
| Medie worst quality | 1598.8 | 170.2 | 5.8 |
| Medie average quality | 7421.25 | 7173.4 | 7240.4 |
| Medie timp de rulare | 0.0000775600026  s | 0.00064231797  s | 0.0045715479  s |

**RHC (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Medie solutii | 13041.9 | 12326.6 | 12165.6 |
| Cea mai buna solutie | 14517 | 14328 | 14073 |
| Cea mai rea solutie | 11648 | 10889 | 10000 |
| Medie timp de rulare |  |  |  |

Setul 2 de valori:

k = 20

max\_capacity = 524

weights = [91, 60, 61, 9, 79, 46, 19, 57, 8, 84, 20, 72, 32, 31, 28, 81, 55, 43, 100, 27]

values = [29, 65, 71, 60, 45, 71, 22, 97, 6, 91, 57, 60, 49, 89, 2, 30, 90, 25, 82, 19]

**RANDOM SEARCH (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Medie best quality | 530.7 | 647.7 | 709.5 |
| Medie worst quality | 311 | 178 | 114 |
| Medie average quality | 420.85 | 412.85 | 411.75 |
| Medie timp de rulare |  |  |  |

**RHC (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Medie solutii | 430.1 | 466.6 | 445.7 |
| Cea mai buna solutie | 556 | 609 | 628 |
| Cea mai rea solutie | 341 | 281 | 191 |
|  |  |  |  |

Setul 3 de valori:

k = 200

max\_capacity = 112648

weights = [835, 1670, 3340, 1087, 1087, 517, 1034, 2068, 1034, 630, 1260, 1260, 1071, 165, 330, 495, 176, 663, 1326, 1326, 984, 1968, 2952, 829, 1658, 3316, 829, 663, 1326, 1989, 1086, 1086, 639, 1278, 2556, 1917, 895, 1790, 3580, 888, 1776, 3552, 2664, 232, 464, 928, 464, 472, 944, 691, 1382, 2764, 1052, 1052, 1057, 2114, 2114, 456, 912, 651, 1302, 1953, 711, 1422, 1042, 232, 464, 696, 1099, 1099, 1029, 2058, 4116, 139, 297, 602, 602, 461, 922, 461, 231, 462, 924, 895, 1790, 1790, 888, 1776, 2664, 575, 1150, 2300, 524, 1048, 1572, 1063, 2126, 1063, 912, 1824, 3648, 2736, 1095, 2190, 3285, 1042, 2084, 2084, 770, 1540, 3080, 973, 1946, 1017, 697, 1394, 2788, 697, 586, 1172, 1758, 734, 1468, 2936, 734, 437, 874, 437, 365, 365, 794, 1588, 2382, 987, 1974, 1974, 258, 258, 907, 1814, 3628, 2721, 304, 608, 1216, 912, 575, 1150, 2300, 1725, 300, 600, 979, 1958, 220, 440, 660, 958, 1916, 3832, 303, 606, 606, 789, 789, 832, 1664, 982, 851, 1702, 3404, 686, 1372, 543, 543, 303, 606, 909, 864, 1728, 3456, 864, 522, 1044, 2088, 1044, 703, 1406, 2812, 2109, 1021, 2042, 4084, 2042, 753, 1506, 3012, 2259, 801, 1602]

values = [735, 1470, 2940, 987, 987, 417, 834, 1668, 834, 530, 1060, 1060, 971, 65, 130, 195, 76, 563, 1126, 1126, 884, 1768, 2652, 729, 1458, 2916, 729, 563, 1126, 1689, 986, 986, 539, 1078, 2156, 1617, 795, 1590, 3180, 788, 1576, 3152, 2364, 132, 264, 528, 264, 372, 744, 591, 1182, 2364, 952, 952, 957, 1914, 1914, 356, 712, 551, 1102, 1653, 611, 1222, 942, 132, 264, 396, 999, 999, 929, 1858, 3716, 39, 197, 502, 502, 361, 722, 361, 131, 262, 524, 795, 1590, 1590, 788, 1576, 2364, 475, 950, 1900, 424, 848, 1272, 963, 1926, 963, 812, 1624, 3248, 2436, 995, 1990, 2985, 942, 1884, 1884, 670, 1340, 2680, 873, 1746, 917, 597, 1194, 2388, 597, 486, 972, 1458, 634, 1268, 2536, 634, 337, 674, 337, 265, 265, 694, 1388, 2082, 887, 1774, 1774, 158, 158, 807, 1614, 3228, 2421, 204, 408, 816, 612, 475, 950, 1900, 1425, 200, 400, 879, 1758, 120, 240, 360, 858, 1716, 3432, 203, 406, 406, 689, 689, 732, 1464, 882, 751, 1502, 3004, 586, 1172, 443, 443, 203, 406, 609, 764, 1528, 3056, 764, 422, 844, 1688, 844, 603, 1206, 2412, 1809, 921, 1842, 3684, 1842, 653, 1306, 2612, 1959, 701, 1402]

**RANDOM SEARCH (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Medie best quality | 93472 | 93763.8 | 96491 |
| Medie worst quality | 93414 | 87542 | 82046.4 |
| Medie average quality | 93443 | 90652.9 | 89268.7 |
| Medie timp de rulare |  |  |  |

**RHC (10 iteratii):**

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. Solutii generate | 10 | 100 | 1000 |
| Calitatea medie | 93509.9 | 93607 | 91932.1 |
| Cea mai buna calitate | 96054 | 96565 | 96644 |
| Cea mai rea calitate | 89765 | 87582 | 74767 |
|  |  |  |  |

Din rezultate deducem concluziile:

* Pentru 1000 solutii generate, algoritmul random search da rezultate egale sau mai bune decat RHC
* Pentru 10 solutii generate, algoritmul RHC da rezultate mai bune decat random search
* Calitatea solutiilor rele devine tot mai mica cu cat avem mai multe solutii generate
* De la 100 la 1000 solutii generate, rezultatele nu difera mult, dar timpul de rulare da. Ca urmare, pentru un caz optim, vom alege un numar mai apropiat de 100 cand alegem numarul de solutii de generat.
* RHC are un timp de rulare mult mai scurt decat random search.

Ca rezultate final, algoritmul optim ar fi RHC, cu numarul de solutii generate > apropiat de 100.