**Algorytmy i struktury danych, Teleinformatyka, I rok**

**Raport z laboratorium nr: 9**

**Imię i nazwisko studenta: Patryk Kurek**

**nr indeksu: 414887**

*1. W pole poniżej wklej najważniejszy (według Ciebie) fragment kodu źródłowego z zajęć (maksymalnie 15 linii).*

def add\_item(self,item):

item\_width,item\_height = item.height,item.width

for w in range(self.width- item\_width+1):

if w+item\_width> self.width + 1 :

break

for h in range(self.height - item\_height+1):

if h + item\_height > self.height +1 :

break

if np.all(self.field[w: w + item\_width, h: h + item\_height] == 0):

self.field[w: w + item\_width, h: h + item\_height] = item.\_id

self.value += item.value

return True

#rotation

item\_width,item\_height = item\_height,item\_width

for w in range(self.width- item\_width +1):

if w+item\_width> self.width :

break

for h in range(self.height - item\_height+1 ):

if h + item\_height > self.height +1 :

break

if np.all(self.field[w: w + item\_width, h: h + item\_height] == 0):

self.field[w: w + item\_width, h: h + item\_height] = item.\_id

self.value += item.value

return True

return False

*Uzasadnij swój wybór.*

Nie zmieściłem się w limicie 15 linii, ale uważam że jest to najważniejszy kawałek kodu w tym ćwiczeniu. Pokazuje algorytm wkładania rzeczy do plecaka. Jest to bardzo ważny algorytm, ponieważ najmniejszy błąd może spowodować całkowicie błędne wyniki w algorytmach i wartościach zwracanych dla plecaka.

*2. Podsumuj wyniki uzyskane podczas wykonywania ćwiczenia. Co ciekawego zauważyłeś? Czego się nauczyłeś? Jeśli instrukcja zawierała pytania, odpowiedz na nie. Do sprawozdania możesz dodać wykresy jeśli jest taka potrzeba.*

W swoim rozwiązaniu użyłem algorytmu genetycznego. Naśladuje on procesy ewolucyjne w naturze. Wykorztysuje pojęcia takie jak selekcja naturalna, krzyżowanie i mutacja, aby ewoluwać populację rozwiązań w kierunku lepszych wyników.

Parametry:

\* p: Rozmiar populacji-określa liczbę osobników w każdym pokoleniu populacji.

\* pc: Procent najlepszych osobników, którzy są wybierani do krzyżowania. W tym przypadku jest to 20%

\* pm: Prawdopodbieństwo mutacji- określa prawdopodbieństwo wystąpienia mutacji w każdym osobniku. W tym przypadku jest to 1%.

\* ps: Intensywność mutacji – określa zakres mutacji, czyli jak bardzo geny są modyfikowane w procesie mutacji. W tym przypadku jest to 5%.

\* iterations: liczba iteracji – określa liczbę pokoleń, przez które algorytm będzie ewoluował populację.

Opis:

Tworzona jest początkowa populacja losowych rozwiązań, gdzie każde rozwiązanie to zestaw losowo wybranych przedmiotów.

Ocena jakości(Density) każdego rozwiązania jest dokonywana poprzez symulację umieszczenia przedmiotów w plecaku i obliczenie wartości.

Populacja jest sortowana malejąco według oceny jakości.

Następuje selekcja najlepszych osobników na podstawie oceny jakości (elitarność).

Przeprowadzane jest krzyżowanie, w którym losowo wybrane pary osobników z populacji są łączone, aby utworzyć nowe osobniki.

Później następuje mutacja, w której losowe geny w wybranych osobnikach są zamieniane, aby wprowadzić nowe kombinacje.

Powtarzane są kolejne iteracje algorytmu, aż do osiągnięcia zadanej liczby iteracji.

Po zakończeniu algorytmu, zwracane jest najlepsze znalezione rozwiązanie, czyli zestaw przedmiotów o najwyższej wartości, który zmieścił się w plecaku.

Porównujemy wyniki: (Algorytm populacji jest ustawiony na 50 iteracji oraz plecak jest wymiarów 10x10)

Greedy Algorithm time to execute: 0.005207061767578125

Greedy Algorithm Value: 58

Greedy Algorithm Backpack Content:

[[ 1. 1. 24. 24. 23. 5. 5. 15. 15. 15.]

[22. 22. 22. 0. 23. 5. 5. 15. 15. 15.]

[22. 22. 22. 0. 23. 5. 5. 15. 15. 15.]

[22. 22. 22. 0. 23. 5. 5. 15. 15. 15.]

[ 0. 0. 0. 0. 23. 0. 0. 15. 15. 15.]

[11. 11. 11. 11. 11. 11. 11. 0. 0. 0.]

[19. 19. 19. 19. 19. 19. 0. 0. 0. 0.]

[19. 19. 19. 19. 19. 19. 0. 0. 0. 0.]

[19. 19. 19. 19. 19. 19. 0. 0. 0. 0.]

[18. 18. 18. 18. 18. 0. 0. 0. 0. 0.]]

Population Algorithm time to execute: 8.8248929977417

Population Algorithm Value: 69

Population Algorithm Backpack Content:

[[15. 15. 15. 8. 8. 8. 8. 8. 8. 8.]

[15. 15. 15. 8. 8. 8. 8. 8. 8. 8.]

[15. 15. 15. 8. 8. 8. 8. 8. 8. 8.]

[15. 15. 15. 5. 5. 5. 5. 24. 24. 23.]

[15. 15. 15. 5. 5. 5. 5. 1. 1. 23.]

[21. 21. 21. 21. 21. 21. 21. 21. 21. 23.]

[21. 21. 21. 21. 21. 21. 21. 21. 21. 23.]

[19. 19. 19. 19. 19. 19. 22. 22. 22. 23.]

[19. 19. 19. 19. 19. 19. 22. 22. 22. 0.]

[19. 19. 19. 19. 19. 19. 22. 22. 22. 0.]]

Greedy Algorithm for 100 Items Time to execute: 0.1470930576324463

Greedy Algorithm for 100 Items Value: 232

[[ 7. 267. 58. 261. 261. 119. 133. 133. 172. 172.]

[278. 442. 223. 315. 315. 134. 445. 445. 445. 419.]

[115. 442. 223. 435. 435. 134. 235. 235. 216. 419.]

[475. 475. 373. 373. 373. 134. 0. 46. 216. 419.]

[358. 358. 358. 358. 221. 221. 221. 46. 216. 274.]

[269. 269. 292. 292. 8. 8. 8. 46. 216. 274.]

[269. 269. 292. 292. 8. 8. 8. 46. 0. 274.]

[103. 103. 103. 103. 103. 103. 0. 46. 117. 117.]

[107. 107. 107. 409. 409. 409. 409. 0. 117. 117.]

[107. 107. 107. 427. 427. 427. 427. 427. 427. 0.]]

Population Algorithm for 100 Items Time to execute: 202.53289794921875

Population Algorithm for 100 Items Value: 170

[[115. 427. 384. 425. 310. 373. 373. 373. 305. 416.]

[ 46. 427. 384. 425. 310. 336. 336. 85. 305. 416.]

[ 46. 427. 384. 425. 310. 336. 336. 85. 305. 416.]

[ 46. 427. 384. 425. 310. 336. 336. 85. 305. 416.]

[ 46. 427. 384. 425. 310. 336. 336. 419. 305. 416.]

[ 46. 427. 384. 425. 310. 336. 336. 419. 305. 416.]

[363. 58. 384. 442. 310. 223. 7. 419. 305. 416.]

[363. 445. 384. 442. 310. 223. 172. 172. 305. 416.]

[363. 445. 384. 267. 310. 91. 91. 91. 91. 133.]

[363. 445. 384. 315. 315. 91. 91. 91. 91. 133.]]

Analizując obydwa wyniki możemy wyciągnąć następujące wnioski:

\* Czasy wykonania dla obydwu algorytmów są dłuższe dla większej liczby elementów

\* Algorytm Naiwny jest znacznie szybszy niż algorytm populacji.

\* Dla odpowiedniej liczby iteracji zazwyczaj otrzymujemy dokładniejsze i najbardziej optymalne wyniki w przypadku algorytmu populacji. (Dla 100 elementów ta teza się nie sprawdza, ale przypuszczam, że gdybyśmy ustawili większą liczbę iteracji algorytm po odpowiednim czasie dałby nam wynik lepszy niż algorytm naiwny)

\* Algorytm naiwny sprawdza się, gdy potrzebujemy relatywnie szybko dostać suboptymalne wyniki. Jest, również łatwy w implementacji.

\* Algorytm Populacji, jest trudny w implementacji, ale za to daje znacznie dokładniejsze wyniki, niestety kosztem zazwyczaj bardzo długiego wykonania i dostosowywania odpowiednich parametrów.

Wyniki dla 500 elementów i 1000 elementów zwyczajnie trwają zbyt długo w przypadku algorytmu populacji.