Programowanie dynamiczne–liniowe zagadnienie załadunku

Zadanie 1

Implementacja funkcji rozwiązującej całkowitoliczbowy problem liniowy. Do funkcji użytkownik podaje: a – listę wag jednej jednostki danego zasobu/przedmiotu, d – listę opisującą liczbę dostępnych sztuk danego przedmiotu, q – macierz opisującą karę za niewybranie określonej liczby sztuk danego przedmiotu, y – maksymalna ładowność.

Uwaga: funkcja oblicza wartości oceny dla 1 etapu dla wszystkich możliwych y, co poniekąd nie ma sensu, gdyż na początku mamy dostępną całą ładowność, jednak w żaden sposób nie psuje to rozwiązania. Natomiast pozbycie się tego wprowadza niepotrzebne trudności podczas programowania.

```
def dynamic(a, d, q, y):
    # a - waga jednej sztuki
    # d - liczba dostępnych sztuk
    # q - macierz kosztów zabrania odpowiedzniej liczby towarów
    # koszty w formie kary, tzn dla makzymalnej liczby danego towaru koszt wynosi 0
    # y - maksymalna ładowność

    # inicjalizacja x jako macierz None'ów
    # i f jako macierz wartości maksymalnych
    # x_matrix = np.array([[[None] for i in range(y+1)] for j in range(len(d))])
    x_matrix = np.zeros((y+1, len(d)))
    f = np.full((y+1, len(d)), np.inf)

    for j in range(f.shape[1]):
        for i in range(f.shape[0]):
            # indeksowanie dla macierzy wejściowych
            # czyli dal j = 0, input_index = etap końcowy
            input_index = f.shape[1] - j - 1

            # wyliczenie maksymalnej liczby przedmiotów
            x = int(i / a[input_index])
            if x > d[input_index]:
                x = d[input_index]

            # jeśli jest to etap ostatni
            if j == 0:
                x_matrix[i, j] = x
                f[i, j] = q[x, input_index]
            # jeśli nie
            else:
                # sprawdzenie wszyskich możliwych ilości x
                for possible_x in range(x+1):
                    # obliczenie kosztu
                    cost = q[possible_x, input_index] + f[i -
a[input_index]*possible_x, j - 1]

                    # sprawdzenie czy koszt jest mniejszy lub równy
                    if cost < f[i, j]:
                        # uaktualnienie
                        x_matrix[i, j] = possible_x
                        f[i, j] = cost

    return x_matrix, f
```

Poza główną funkcją zaimplementowałem także nieco kodu pomocniczego:

- funkcję pozwalającą na wyświetlenie optymalnej strategii w formie: |liczba_sztuk: numer przedmiotu (f=watość_zmacierzy_f)|,

```
def get_results(a, y, x_matrix, f):
    free_space = y
    strategy = list()

    for j in range(f.shape[1]-1, -1, -1):

        input_index = f.shape[1] - j - 1
        strategy.append(f"| {int(x_matrix[free_space, j])}: x{input_index} |")
        free_space = int(free_space - a[input_index] * x_matrix[free_space, j])

    return strategy
```

- oraz kawałek kodu do wygenerowania macierzy q dla wymaganego problemu o przynajmniej 10 zmiennych

```
y = 33
a = np.array([1, 2, 3, 4, 3, 2, 10, 2, 3, 3, 13])
d = np.array([6, 3, 2, 1, 8, 2, 1, 2, 7, 3, 5])
q = np.zeros((np.max(d)+1, len(d)))

for j in range(q.shape[1]):
    for i in range(q.shape[0]):
        if i == 0:
            q[i, j] = random.randint(15, 47)
        else:
            q[i, j] = q[i-1, j] - random.randint(0, 15)
            q[i, j] = max((q[i, j], 0))
```

Zadanie 2

Zadanie rozwiązuję najpierw dla problemu z wykładu, w celu łatwiejszej kontroli poprawności rozwiązania – otrzymuję następujący wynik, który jest zgodny z wynikiem uzyskanym na wykładzie.

```
Macierz x =
[[0. 0. 0.]
 [0. 0. 1.]
 [0. 1. 2.]
 [1. 0. 3.]
 [1. 2. 4.]
 [1. 1. 5.]
 [2. 3. 6.]
 [2. 2. 5.]]

Macierz f =
[[ 6. 15. 35.]
 [ 6. 15. 33.]
 [ 6. 12. 29.]
 [ 2. 11. 26.]
 [ 2.  9. 22.]
 [ 2.  8. 17.]
 [ 0.  6. 15.]
 [ 0.  5. 14.]]


 ['| 5: x0 (f=14.0)|', '| 1: x1 (f=12.0)|', '| 0: x2 (f=6.0)|']
```

Następnie używam wcześniej opisanego kodu do wygenerowania macierzy *q* dla problemu o przynajmniej 10 zmiennych – konkretnie rozpatruje problem o 11 zmiennych. Otrzymuję następujące warunki początkowe:

```
Maksymalna ładowność: y =  33
Wektor a =
 [ 1  2  3  4  3  2 10  2  3  3 13]
Wektor d =
 [6 3 2 1 8 2 1 2 7 3 5]
Macierz q =
 [[36. 47. 35. 43. 45. 28. 25. 40. 17. 38. 35.]
 [25. 33. 32. 32. 41. 25. 19. 38. 13. 33. 22.]
 [10. 28. 23. 21. 29. 18.  5. 23. 13. 32. 10.]
 [ 9. 27. 10. 12. 25. 16.  2. 10.  6. 25.  6.]
 [ 5. 21.  9.  9. 21.  8.  0.  1.  0. 14.  0.]
 [ 0. 19.  0.  2. 18.  6.  0.  0.  0.  2.  0.]
 [ 0. 14.  0.  0. 15.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  3.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Dla których odpowiedź wynosi:

```
Macierz x =
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 2.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 3.]
 [0. 1. 0. 2. 0. 0. 0. 0. 0. 2. 2.]
 [0. 1. 0. 2. 0. 0. 0. 0. 0. 1. 3.]
 [0. 2. 1. 2. 0. 1. 0. 0. 0. 1. 2.]
 [0. 2. 1. 2. 0. 0. 0. 0. 0. 1. 5.]
 [0. 2. 1. 2. 0. 2. 0. 1. 0. 2. 2.]
 [0. 3. 0. 2. 0. 2. 0. 1. 0. 1. 3.]
 [0. 3. 0. 2. 0. 2. 2. 0. 0. 1. 2.]
 [0. 3. 0. 2. 0. 2. 2. 0. 0. 1. 5.]
 [0. 3. 1. 2. 0. 2. 2. 1. 0. 1. 2.]
 [1. 0. 1. 2. 0. 2. 2. 0. 0. 1. 5.]
 [1. 0. 1. 2. 0. 2. 2. 1. 0. 1. 2.]
 [1. 0. 4. 2. 0. 2. 2. 1. 0. 1. 5.]
 [1. 1. 4. 2. 0. 2. 2. 1. 0. 1. 2.]
 [1. 1. 4. 2. 0. 2. 2. 1. 0. 1. 5.]
 [1. 1. 3. 2. 0. 2. 2. 1. 0. 2. 2.]
 [1. 2. 3. 2. 0. 2. 2. 1. 0. 1. 5.]
 [1. 2. 3. 2. 0. 2. 2. 1. 2. 1. 2.]
 [1. 2. 4. 2. 0. 2. 2. 1. 0. 1. 5.]
 [1. 3. 4. 2. 0. 2. 2. 1. 0. 2. 2.]
 [1. 3. 4. 2. 0. 2. 2. 1. 2. 1. 5.]
 [1. 3. 4. 2. 0. 2. 2. 1. 2. 2. 2.]
 [1. 3. 1. 2. 0. 2. 7. 1. 2. 2. 5.]
 [2. 3. 1. 2. 0. 2. 2. 1. 2. 1. 4.]
 [2. 3. 1. 2. 0. 2. 7. 1. 2. 1. 5.]
 [2. 3. 4. 2. 0. 2. 7. 1. 2. 2. 5.]
 [2. 1. 4. 2. 0. 2. 7. 1. 2. 1. 5.]
 [2. 1. 4. 2. 0. 2. 7. 1. 2. 3. 5.]
 [2. 1. 3. 2. 0. 2. 7. 1. 2. 2. 5.]
 [2. 2. 3. 2. 0. 2. 7. 1. 0. 2. 4.]
 [2. 2. 3. 2. 0. 2. 7. 1. 0. 3. 5.]]
```

```
Macierz f =
[[ 35.  73.  90. 130. 155. 183. 228. 271. 306. 353. 389.]
 [ 35.  73.  90. 130. 155. 183. 228. 271. 306. 353. 378.]
 [ 35.  73.  90. 128. 153. 180. 225. 268. 303. 339. 363.]
 [ 35.  68.  85. 125. 150. 178. 223. 266. 301. 339. 362.]
 [ 35.  68.  85. 113. 138. 166. 211. 254. 289. 334. 349.]
 [ 35.  68.  85. 113. 138. 166. 211. 254. 289. 334. 348.]
 [ 35.  67.  81. 113. 138. 163. 208. 251. 286. 322. 344.]
 [ 35.  67.  81. 108. 133. 161. 206. 249. 284. 322. 339.]
 [ 35.  67.  81. 108. 133. 156. 201. 243. 278. 317. 332.]
 [ 35.  60.  77. 108. 133. 156. 201. 243. 278. 317. 331.]
 [ 35.  60.  77. 104. 129. 156. 195. 238. 273. 311. 327.]
 [ 35.  60.  77. 104. 129. 151. 195. 238. 273. 311. 322.]
 [ 35.  60.  73. 104. 129. 151. 192. 233. 268. 306. 321.]
 [ 22.  60.  73. 100. 125. 151. 190. 233. 268. 306. 317.]
 [ 22.  60.  73. 100. 125. 147. 185. 227. 262. 301. 316.]
 [ 22.  60.  68. 100. 125. 147. 185. 227. 262. 301. 311.]
 [ 22.  55.  68.  96. 121. 147. 185. 224. 259. 295. 311.]
 [ 22.  55.  68.  96. 121. 143. 180. 222. 257. 295. 306.]
 [ 22.  55.  66.  96. 121. 143. 180. 217. 252. 290. 305.]
 [ 22.  54.  66.  91. 116. 143. 180. 217. 252. 290. 301.]
 [ 22.  54.  66.  91. 116. 139. 176. 217. 250. 285. 300.]
 [ 22.  54.  60.  91. 116. 139. 176. 212. 247. 285. 295.]
 [ 22.  47.  60.  89. 114. 139. 176. 212. 247. 280. 295.]
 [ 22.  47.  60.  89. 114. 134. 172. 212. 245. 280. 290.]
 [ 22.  47.  60.  89. 114. 134. 172. 208. 240. 278. 290.]
 [ 22.  47.  60.  83. 108. 134. 169. 208. 240. 275. 285.]
 [ 10.  47.  60.  83. 108. 132. 168. 208. 240. 273. 285.]
 [ 10.  47.  60.  83. 108. 132. 166. 204. 235. 273. 280.]
 [ 10.  47.  55.  83. 108. 132. 164. 204. 235. 268. 280.]
 [ 10.  43.  55.  83. 108. 126. 159. 201. 235. 268. 278.]
 [ 10.  43.  55.  83. 108. 126. 159. 200. 231. 267. 275.]
 [ 10.  43.  53.  83. 108. 126. 159. 198. 231. 263. 273.]
 [ 10.  42.  53.  78. 103. 126. 154. 196. 231. 263. 273.]
 [ 10.  42.  53.  78. 103. 126. 154. 191. 226. 262. 268.]]
```

Optymalna strategia prezentuje się następująco (wyświetlana w formie omówionej w „Zadaniu 1”).

```
['| 5: x0 (f=268.0)|', '| 2: x1 (f=268.0)|', '| 2: x2 (f=240.0)|', '| 1: x3 (f=217.0)|', '| 2: x4 (f=185.0)|',
'| 2: x5 (f=156.0)|', '| 0: x6 (f=138.0)|', '| 2: x7 (f=113.0)|', '| 0: x8 (f=90.0)|', '| 0: x9 (f=73.0)|', '| 0: x10 (f=35.0)|']
```

Uzyskana wartość funkcji celu to wartość f, dla x0 – w tym przypadku wynosi ona 268.

Zadanie 3

- Jakie założenia muszą być spełnione dla wag i zysków?

  Wektory wag i zysków powinny być tej samej długości, ponadto powinny spełniać warunki zależne od rozważanego problemu rzeczywistego – zazwyczaj jednym z takich warunków będzie np. nieujemność wag i zysków.

- Co się stanie, jeśli tych założeń nie spełnimy?

  W przypadku niespełnienia warunku dotyczącego równego rozmiaru wektorów zysków i wag rozwiązanie problemu stanie się niemożliwe ze względu na błędy działania samego kodu, co jest lepszą sytuacją ponieważ szybko taki błąd jesteśmy w stanie wychwycić i poprawić. Natomiast w przypadku niedopasowania wartości w tych wektorach do problemu rzeczywistego sytuacja jest nieco gorsza, ponieważ algorytm się wykona jednak wynik będzie nieprawdziwy, co może być trudne do wykrycia.

- Jaka jest złożoność obliczeniowa algorytmu?

  Algorytm przechodzi po całej macierzy o rozmiarze (y+1, len(d)) zakładając *n = y+1, m = len(d),* wskazuje to na złożoność *O(n*m)*, jednak rzeczywista złożoność będzie nieco większa ze względu na rozważanie w każdym kroku wszystkich możliwych *x* od 0 do y/a