Lista 6 - wysukiwanie i funkcje haszujące

Zadanie 0. - przygotowanie strutury

• Utwórz strukturę (może być klasa) do przechowywania opisów floty robotów mobilnych wykorzysty- wanych w zadaniach eksploracji i inspekcji. Pola składowe struktury to: identyfikator, typ, masa, zasięg, rozdzielczość (kamery).\ • Zaimplementuj funkcję generującą opis pojedynczego robota (jeden obiekt) o parametrach losowanych jak następuje: identyfikator – ciąg znaków alfanumerycznych o długości N, typ – „AUV", "AFV", „AGV", masa – od 50 do 2000 [dag], zasięg – od 1 do 1000 [km], rozdzielczość – od 1 do 30 [MP]. Wartości liczbowe są całkowite.\ • Zaimplementuj funkcję generującą opisy M robotów o losowych parametrach. Opisy, w postaci struktur są zapisywane w M-elementowym wektorze.\ • Zaimplementuj funkcję wyświetlającą wygenerowaną strukturę w postaci tabelki (jeden wiersz – jeden robot; równe odstępy między kolejnymi polami).\ • Zaimplementuj funkcję zapisującą/odczytującą opisy do/z pliku.\ Na tym wektorze robotów realizowane są kolejne zadania. \ Uwaga: nie zakładaj, że wartości parametrów są unikalne.

In [2]:
```
import random
import pandas as pd
import numpy as np
```

In [18]:
```
class Robot:
    def __init__(self, id, typ, masa, zas, roz):
        self.id = id
        self.typ = typ
        self.masa = masa
        self.zas = zas
        self.roz = roz

    def save_robot(self):
        """
        :return dict robot: zwraca słownik z oarametrami robota
        """
        robot = {
            'id': self.id,
            'typ': self.typ,
            'masa': self.masa,
            'roz': self.roz
        }
        return robot


def create_robot():
    """
    Funkcja tworząca robota
    :return bot: objekt klasy robot
    """
    bot = Robot(
        id=''.join(random.choices(['0', '1'], k=4)),
        typ=''.join(random.choices(['A', 'U', 'V', 'G', 'C', 'W', 'N'], k=3)),
        masa=np.random.randint(50, 2001),
        zas=np.random.randint(1, 1000),
        roz=np.random.randint(1, 30)
    )
    return bot


def create_m_bots(m):
    """
    Funkcja wywolująca m razy funkcję tworzącom robota, dodatkowo zapisuje obiekty w liście
    :param int m: liczba robotów
    :return list bots: lista robotów
    """
    bots = list()
    for _ in range(m):
        bots.append(create_robot())
    return bots


# stworzenie m robotów
botslist = create_m_bots(10)
# stworzenie ramnki danych (tabelka)
df = pd.DataFrame()
# zapisanie robotów do tabelki
for b in botslist:
    df = df.append(b.save_robot(), ignore_index=True)



def save_to_file(data):
    """
    Funkcja zapisująca listę robotów do pliku
    :param data: ramka Pandas
    """
    data.to_csv('pliki/roboty.csv')


def read_from_file(path):
    """
    Funkcja odczytująca plik csv i zapisująca zawartość do ramki pandas
    :param string path: ścieżka do pliku
    """
    data = pd.read_csv(path)
    return data
```

In [19]:
```
# wyświetlenie tabeli
print(df)

# zapisanie do plik
save_to_file(df)

# odczytanie z pliku
print(read_from_file('pliki/roboty.csv'))
```

```
     id  typ    masa   roz
0  1001  UAV  1074.0  19.0
1  1101  UGU  1992.0  23.0
2  0100  AWG   503.0  25.0
3  0100  GGV   310.0  26.0
4  0000  WNA   731.0  21.0
5  1100  WWW  1777.0  17.0
6  0100  GNG  1145.0   1.0
7  1010  GAN   569.0  12.0
8  0110  VUG  1155.0   1.0
9  1101  NWG   495.0  21.0
   Unnamed: 0    id  typ    masa   roz
0           0  1001  UAV  1074.0  19.0
1           1  1101  UGU  1992.0  23.0
2           2   100  AWG   503.0  25.0
3           3   100  GGV   310.0  26.0
4           4     0  WNA   731.0  21.0
5           5  1100  WWW  1777.0  17.0
6           6   100  GNG  1145.0   1.0
7           7  1010  GAN   569.0  12.0
8           8   110  VUG  1155.0   1.0
9           9  1101  NWG   495.0  21.0
```

Zadanie 1. - wyszukiwanie liniowe

• Zaimplementuj funkcję wyszukującą jednego robota z wektora robotów. Poszukiwanie odbywa się względem jednego parametru (parametr wybierany przez użytkownika; szukana wartość wybierana przez użytkownika). Funkcja realizuje algorytm wyszukiwania liniowego i zwraca strukturę z pierwszym znalezionym robotem (lub None, jeśli robota nie znajdzie).\ • Rozwiń napisaną funkcję tak, aby wyszukiwanie odbywało się po zbiorze parametrów. Zbiór i wartości są zadawane przez użytkownika w postaci struktury analogicznej do tej reprezentującej pojedynczego robota; parametry, po których wyszukiwanie się nie odbywa pozostają puste (None), np. (None, „AUV", 50, 10, None).\ • Rozwiń napisaną funkcję tak, aby akceptowała też wektory wartości parametrów, np. (None, „AUV", [90, 51, 52, 53], [10, 11, 15], None). Zagnieźdź algorytmy wyszukiwania liniowego tak, aby parametr robota był wyszukiwany w liście dopuszczalnych wartości.

In [20]:
```
from time import sleep
```

In [21]:
```
bots = create_m_bots(1000)
if type(bots) != dict:
    robots = []
    for bot in bots:
        robots.append(bot.save_robot())
```

In [22]:
```
# zamiast losować można też użyć poniższy wektor robotów
robots = [
    {'id': 'X11298816', 'typ': 'CWN', 'masa': 660, 'roz': 20},
    {'id': '3Y059465Z', 'typ': 'WAC', 'masa': 83, 'roz': 25},
    {'id': '3241363YX', 'typ': 'CAG', 'masa': 1848, 'roz': 28},
    {'id': 'X81X30X7Y', 'typ': 'ACU', 'masa': 1964, 'roz': 11},
    {'id': '294Y0760X', 'typ': 'WCV', 'masa': 342, 'roz': 1},
    {'id': 'Z49161Z71', 'typ': 'GUC', 'masa': 810, 'roz': 8},
    {'id': 'Z27535428', 'typ': 'UAV', 'masa': 452, 'roz': 4},
    {'id': '30XY877X0', 'typ': 'WUU', 'masa': 1071, 'roz': 14},
    {'id': '4Y89757Y1', 'typ': 'VNA', 'masa': 1979, 'roz': 2},
    {'id': '6831Z4147', 'typ': 'WCW', 'masa': 574, 'roz': 7}]
```

In [23]:
```
def find_robot_1(p_name, p_value, robots_list=robots, many=0):
    """
    Funkcja szukająca pierwszego robota z zadanym parametrem
    :param p_name: nazwa parametru
    :param p_value: wartość parametru
    :param robots_list: lista robotów do pszeszukania, domyślnie lista wcześniej wygenerowanych robotów
    :param bool many: informacja czy zwrócić wszytkie roboty spełniające kryteria czy tylko jednego
    :return b or None: zwraca robota (słownik) lub None jeśli dany robot nie istnij
    """
    fulfilling = []
    for b in robots_list:
        if b[p_name] == p_value:
            if not many:
                return b
            fulfilling.append(b)

    return


def find_robot_many(features, display=False):
    """
    Funkcja szuka robota z wieloma parametrami
    :param dict features: słownik z parametrami dla szukanego robota
    :param display: domyślnie False, można ustawić na True wtedy będzie wyświetlany krok wisualizacji co 1s
    :return dict b or None: zwracamy robota o określonych parametrach lub None jeśli taki robot nie istnieje
    """
    # słownik cech które nie są None
    params = dict()
    for f in features:
        if features[f]:
            params[f] = features[f]

    for b in robots:
        if display:
            print(b)
            print(features)
            sleep(1)
        correct_params = 0
        for p in params:
            if type(params[p]) == list:
                if b[p] in params[p]:
                    correct_params += 1
                else:
                    break
            else:
                if b[p] == params[p]:
                    correct_params += 1
                else:
                    break

        if correct_params == len(params):
            return b
    return
```

In [24]:
```
print('Wektor robotów:')
print(robots[:10])

feature = 'ACU'
print('\nWyszukiwaie z jedną cechą:', feature)
print('Znaleziony robot:')
print(find_robot_1('typ', feature, robots))

print("\nPoszukiwanie z wieloma cechami i z wektorem cech:")
desired_fetures = {'id': None, 'typ': ['AUC', 'AWU', 'AVC'], 'masa': None, 'roz': [11, 12, 13, 14, 15, 16, 17,
print('pożądane cechy:', desired_fetures, '\n')
print('Znaleziony robot:')
print(find_robot_many(desired_fetures))
```

```
Wektor robotów:
[{'id': 'X11298816', 'typ': 'CWN', 'masa': 660, 'roz': 20}, {'id': '3Y059465Z', 'typ': 'WAC', 'masa': 83, 'ro
z': 25}, {'id': '3241363YX', 'typ': 'CAG', 'masa': 1848, 'roz': 28}, {'id': 'X81X30X7Y', 'typ': 'ACU', 'masa':
1964, 'roz': 11}, {'id': '294Y0760X', 'typ': 'WCV', 'masa': 342, 'roz': 1}, {'id': 'Z49161Z71', 'typ': 'GUC',
'masa': 810, 'roz': 8}, {'id': 'Z27535428', 'typ': 'UAV', 'masa': 452, 'roz': 4}, {'id': '30XY877X0', 'typ': 'W
UU', 'masa': 1071, 'roz': 14}, {'id': '4Y89757Y1', 'typ': 'VNA', 'masa': 1979, 'roz': 2}, {'id': '6831Z4147',
'typ': 'WCW', 'masa': 574, 'roz': 7}]

Wyszukiwaie z jedną cechą: ACU
Znaleziony robot:
{'id': 'X81X30X7Y', 'typ': 'ACU', 'masa': 1964, 'roz': 11}

Poszukiwanie z wieloma cechami i z wektorem cech:
pożądane cechy: {'id': None, 'typ': ['AUC', 'AWU', 'AVC'], 'masa': None, 'roz': [11, 12, 13, 14, 15, 16, 17, 1
8, 19]}

Znaleziony robot:
None
```

Zadanie 2. - wyszukiwanie binarne.

• Zaimplementuj funkcję sortującą wektor robotów względem wskazanego parametru. Wykorzystaj wbudowaną funkcję sort.\ • Zaimplementuj funkcję wyszukującą jednego robota o zadanym parametrze, przy założeniu, że wektor jest odpowiednio posortowany. Funkcja realizuje algorytm wyszukiwania binarnego i zwraca strukturę z pierwszym znalezionym robotem (lub None, jeśli robota nie znajdzie).\ • Rozwiń napisaną funkcję tak, aby akceptowała wektor dopuszczalnych wartości parametrów (posorto- wany). Wyszukiwanie binarne wykorzystaj wielokrotnie.

Funkcje wyszukiwania uzupełnij o tryb wizualizacji krok-po-kroku.

In [25]:
```
# stworzenie wektora robotów
vector = create_m_bots(20)
```

In [26]:
```
# przekształcenie wektora z obiektu klasy na listę
vector = [b.save_robot() for b in vector]
vector
```

Out[26]:
```
[{'id': '1001', 'typ': 'WGC', 'masa': 850, 'roz': 11},
 {'id': '1101', 'typ': 'UAG', 'masa': 1268, 'roz': 17},
 {'id': '1001', 'typ': 'AWN', 'masa': 1227, 'roz': 26},
 {'id': '1110', 'typ': 'AWU', 'masa': 884, 'roz': 5},
 {'id': '1000', 'typ': 'NAU', 'masa': 377, 'roz': 16},
 {'id': '0111', 'typ': 'WWU', 'masa': 609, 'roz': 1},
 {'id': '1010', 'typ': 'UUW', 'masa': 1120, 'roz': 12},
 {'id': '1100', 'typ': 'WGA', 'masa': 537, 'roz': 1},
 {'id': '0000', 'typ': 'AUW', 'masa': 723, 'roz': 24},
 {'id': '1100', 'typ': 'CAA', 'masa': 309, 'roz': 14},
 {'id': '0100', 'typ': 'CWG', 'masa': 856, 'roz': 23},
 {'id': '1100', 'typ': 'AGV', 'masa': 211, 'roz': 14},
 {'id': '1111', 'typ': 'WNV', 'masa': 1907, 'roz': 11},
 {'id': '0001', 'typ': 'WNC', 'masa': 1866, 'roz': 21},
 {'id': '1111', 'typ': 'WNV', 'masa': 468, 'roz': 25},
 {'id': '0000', 'typ': 'GNU', 'masa': 1238, 'roz': 3},
 {'id': '1001', 'typ': 'WVG', 'masa': 1244, 'roz': 4},
 {'id': '0110', 'typ': 'VWU', 'masa': 1521, 'roz': 17},
 {'id': '1110', 'typ': 'CVA', 'masa': 1668, 'roz': 3},
 {'id': '1001', 'typ': 'UNV', 'masa': 586, 'roz': 22}]
```

In [27]:
```
# sortowanie wektora wedłu wartosci ID
def sort_vector(vector, key_feature):
     return sorted(vector, key=lambda x: x[key_feature])
```

In [28]:
```
def binary_search(w, value, feature):
    """
    :param list w:
    :param string value:
    :return optional dict: słownik - robot z odpowiadająca cechą
    """
    w = sort_vector(w, feature)
    min_index = 0
    max_index = len(w)
    index = None

    # jeśli podanych jest wiele wartości szukamy robota dla każdej z nich
    if type(value) == list:
        results = list()
        for v in value:
            results.append(binary_search(w, v, feature))
        return results

    while index != (max_index + min_index) // 2:
        index = (max_index + min_index) // 2
        if w[index]['id'] == value:
            return w[index]
        elif w[index]['id'] <= value:
            min_index = index+1
        elif w[index]['id'] >= value:
            max_index = index-1


print('id=1010', binary_search(vector, '1010', 'id'))
print('id=1111', binary_search(vector, '1111', 'id'))
print("id=['1010', '1011', '1111']\n", binary_search(vector, ['1010', '1011', '1111'], 'id'))
```

```
id=1010 {'id': '1010', 'typ': 'UUW', 'masa': 1120, 'roz': 12}
id=1111 {'id': '1111', 'typ': 'WNV', 'masa': 1907, 'roz': 11}
id=['1010', '1011', '1111']
 [{'id': '1010', 'typ': 'UUW', 'masa': 1120, 'roz': 12}, None, {'id': '1111', 'typ': 'WNV', 'masa': 1907, 'ro
z': 11}]
```

Zadanie 3. - wyszukiwanie z wykorzystaniem funkcji haszującej

• Zaproponuj i zaimplementuj funkcję haszującą działającą na pojedynczej strukturze robota. Zbiór wartości funkcji to liczba całkowita z zakresu od 0 do H − 1. Funkcja powinna uwzględniać wszystkie parametry robota. Do ograniczenia wartości wykorzystaj operację modulo.

• Zaimplementuj funkcję generującą wektor wartości funkcji haszującej dla zadanego wektora robotów. Wektor wartości jest generowany jak następuje:

Algorithm 1 GWW – Generacja wektora wartości

- wejscie: wektor robotów, H >> |wektor robotów|

1. niech wektor wartości będzie H −1 elementowym wektorem wypełnionym pustymi elementami (None)
2. dla n := 0 do |wektor robotów| − 1
3. oblicz wartość h funkcji haszującej na ntym elemencie wektora robotów
4. na pierwszej pustej pozycji wektora wartości licząc od h tej pozycji (włącznie) wstaw n; jeśli dokońca wektora wartości brakuje wolnych pozycji, to kontynuuj od jego początku

- wyjście: wektor wartości

• Zaimplementuj funkcję wyszukiwania robota o zadanych wszystkich parametrach, która korzysta z wektora wartości i funkcji haszującej.

• Zaproponuj i zaimplementuj metodę wyszukiwania robota po 2 parametrach: masa i zasięg, dopuszczając sytuację, gdy dowolny z parametrów jest nieznany, np. (None, 40). Wykorzystaj (być może zmodyfikowaną) funkcję haszującą (generacja wektora wartości może odbywać się inaczej niż w algorytmie GWW). Zwracaj listę wszystkich robotów spełniających postawione wymagania.

Funkcje wyszukiwania i generacji wektora wartości uzupełnij o tryb wizualizacji krok-po-kroku.

In [29]:
```
def get_hash(x, H):
    """
    Funkcja wyliczająca hash wartości dla tabeli o podanej długości
    :parma x:  wartość
    :param H: długość struktury danych w której się znajduje
    """
    h = 0
    for v in x.values():
        h += hash(v)
    return h % (H-1)
```

In [30]:
```
def gww(vector):
    """
    Funkcja wyznacza indeksy dla kolejnych wartosci wektora przy użyciu funkcji haszującej.

    :param vector: wektor do haszowania (w tym przypadku wektor robotów)
    :return list hashed: posortowana lista według hashy
    """
    H = len(vector)
    hashed = [None] * H

    for n in vector:
        h = get_hash(n, H)

        while hashed[h]:
            h += 1
            if h == H:
                h -= H

        hashed[h] = n

    return hashed
```

In [284…
```
robots = create_m_bots(10)
robots = [r.save_robot() for r in robots]

print(robots)
print(gww(robots))
```

```
[{'id': '0010', 'typ': 'GUG', 'masa': 728, 'roz': 3}, {'id': '0011', 'typ': 'AWC', 'masa': 1995, 'roz': 7}, {'i
d': '1111', 'typ': 'GAG', 'masa': 1468, 'roz': 6}, {'id': '0111', 'typ': 'NCN', 'masa': 1046, 'roz': 19}, {'i
d': '0110', 'typ': 'WAA', 'masa': 1628, 'roz': 18}, {'id': '0011', 'typ': 'AUN', 'masa': 1844, 'roz': 28}, {'i
d': '0111', 'typ': 'WAU', 'masa': 592, 'roz': 15}, {'id': '0010', 'typ': 'CGA', 'masa': 1564, 'roz': 20}, {'i
d': '0110', 'typ': 'UNW', 'masa': 401, 'roz': 18}, {'id': '1000', 'typ': 'UWU', 'masa': 1719, 'roz': 3}]
[{'id': '0110', 'typ': 'WAA', 'masa': 1628, 'roz': 18}, {'id': '0111', 'typ': 'WAU', 'masa': 592, 'roz': 15},
{'id': '0110', 'typ': 'UNW', 'masa': 401, 'roz': 18}, {'id': '0010', 'typ': 'CGA', 'masa': 1564, 'roz': 20},
{'id': '0011', 'typ': 'AUN', 'masa': 1844, 'roz': 28}, {'id': '1000', 'typ': 'UWU', 'masa': 1719, 'roz': 3},
{'id': '1111', 'typ': 'GAG', 'masa': 1468, 'roz': 6}, {'id': '0111', 'typ': 'NCN', 'masa': 1046, 'roz': 19},
{'id': '0010', 'typ': 'GUG', 'masa': 728, 'roz': 3}, {'id': '0011', 'typ': 'AWC', 'masa': 1995, 'roz': 7}]
```

In [ ]:
```
def hash_bin_search(features, val_vector):
    h = get_hash(features, len(val_vector))
```

Zadanie 4. - binarne wyszukiwanie zbioru wartości

• Zaimplementuj funkcję zwracającą zbiór indeksów wszystkich robotów, których wybrany parametr ma zadaną wartość. Wykorzystaj procedurę poszukiwania binarnego w celu znalezienia pierwszego robota spełniającego wymaganie. Zauważ, że w wyniku działania procedury znaleziony został też przedział, w którym znajdują się wszystkie poszukiwane roboty. Dokładniej, w ostatniej iteracji algorytmu, wykonano operację $mid = round(\frac{max+min}{2})$, gdzie mid jest zwróconym indeksem robota, zaś min i max to krańce przedziału poszukiwania w ostatniej iteracji. Znając min i max, uruchom wyszukiwanie binarne na przedziałach [min, mid] i [mid, max] w celu znalezienia minimalnej i maksymalnej indeksów robotów spełniających wymaganie.

• Zaimplementuj funkcję zwracającą zbiór indeksów wszystkich robotów o parametrze liczbowym z za- danego przedziału. Ponownie, wykorzystaj wyszukiwanie binarne.

In [188…
```
# stworzenie wektora robotów
vector = create_m_bots(10)

# przekształcenie wektora z obiektu klasy na listę
vector = [b.save_robot() for b in vector]
```

In [203…
```
def bin_ser_rec(v, imin, imax, value, feature):
    """
    Funkcja wyszukuje binarnie wszystkich indeksów robotów o zadanej warotości danego parametru
    :param list v: wektor robotów
    :param int imin: dolny indeks przeszukiwania binarnego
    :param int imax: górny indeks przeszukiwnaia binarnego
    :param value: wartość wyszukiwanego elementu
    :param feature: cecha według której sortujemy a następnie przeszukujemy listę
    :return list: zwraca listę jeśli nie ma robota o zadanej cesze lista będzie pusta
    """
    v = sorted(v, key=lambda x: x[feature])

    if imax >= imin:
        i = (imin + imax) // 2
        if v[i][feature] == value:
            if imin == imax:
                return [i]
            else:
                return bin_ser_rec(v, imin, i, value, feature) + bin_ser_rec(v, i+1, imax, value, feature)
        elif v[i][feature] > value:
            return bin_ser_rec(v, imin, i-1, value, feature)
        else:
            return bin_ser_rec(v, i+1, imax, value, feature)

    return []
```

In [192…
```
# wyszukaanie indeksu robota o zadanej cesze
val = '0001'
feat = 'id'
print('Binarne wyszukiwanie zbioru wartości dla parametr:', feat, 'o wartości:', val)
print('Indeksy:')
print(bin_ser_rec(v=vector, imin=0, imax=len(vector), value=val, feature=feat))
sorted(vector, key=lambda x: x[feat])
```

```
Binarne wyszukiwanie zbioru wartości dla parametr: id o wartości: 0001
Indeksy:
[1]
```

Out[192]:
```
[{'id': '0000', 'typ': 'CGA', 'masa': 678, 'roz': 19},
 {'id': '0001', 'typ': 'UAV', 'masa': 970, 'roz': 9},
 {'id': '0010', 'typ': 'AUU', 'masa': 359, 'roz': 23},
 {'id': '0011', 'typ': 'CNC', 'masa': 1489, 'roz': 25},
 {'id': '0011', 'typ': 'WVV', 'masa': 562, 'roz': 21},
 {'id': '0100', 'typ': 'NCG', 'masa': 1150, 'roz': 18},
 {'id': '0101', 'typ': 'AVN', 'masa': 679, 'roz': 29},
 {'id': '0110', 'typ': 'GWU', 'masa': 339, 'roz': 29},
 {'id': '0110', 'typ': 'AGG', 'masa': 294, 'roz': 20},
 {'id': '1001', 'typ': 'GGC', 'masa': 1963, 'roz': 3}]
```

In [198…
```
def bin_ser_range(vec, imin, imax, val_min, val_max, feature):
    """
    Funkcja wyszukuje binarnie wszystkich indeksów robotów o zadanej warotości danego parametru
    :param list v: wektor robotów
    :param int imin: dolny indeks przeszukiwania binarnego
    :param int imax: górny indeks przeszukiwnaia binarnego
    :param val_min: dolna wartość przedziału
    :param val_max: górna wartość przedziału
    :param feature: cecha według której sortujemy a następnie przeszukujemy listę
    :return list: zwraca listę jeśli nie ma robota o zadanej cesze lista będzie pusta
    """
    v = sorted(vec, key=lambda x: x[feature])

    if imax >= imin:
        i = (imin + imax) // 2
        if val_min < v[i][feature] < val_max:
            if imin == imax:
                return [v.index(vec[i])]
            else:
                return bin_ser_range(v, imin, i, val_min, val_max, feature) + \
                       bin_ser_range(v, i+1, imax, val_min, val_max, feature)
        elif v[i][feature] > val_max:
            return bin_ser_range(v, imin, i-1, val_min, val_max, feature)
        elif v[i][feature] < val_min:
            return bin_ser_range(v, i+1, imax, val_min, val_max, feature)

    return []
```

In [199…
```
# wyszukanie indeksów robotów o cechach z zadanego przedziału
feat = 'masa'
vmin = 100
vmax = 1000
print('Binarne wyszukiwanie zbioru wartości dla parametr:', feat, 'z przedziału:', [vmin, vmax])
print('Indeksy:')
print(bin_ser_range(vec=vector, imin=0, imax=len(vector), val_min=vmin, val_max=vmax, feature=feat))
```

```
Binarne wyszukiwanie zbioru wartości dla parametr: masa z przedziału: [100, 1000]
Indeksy:
[0, 1, 2, 3, 4, 5, 6]
```

In [201…
```
sorted(vector, key=lambda x: x[feat])
```

Out[201]:
```
[{'id': '0110', 'typ': 'AGG', 'masa': 294, 'roz': 20},
 {'id': '0110', 'typ': 'GWU', 'masa': 339, 'roz': 29},
 {'id': '0010', 'typ': 'AUU', 'masa': 359, 'roz': 23},
 {'id': '0011', 'typ': 'WVV', 'masa': 562, 'roz': 21},
 {'id': '0000', 'typ': 'CGA', 'masa': 678, 'roz': 19},
 {'id': '0101', 'typ': 'AVN', 'masa': 679, 'roz': 29},
 {'id': '0001', 'typ': 'UAV', 'masa': 970, 'roz': 9},
 {'id': '0100', 'typ': 'NCG', 'masa': 1150, 'roz': 18},
 {'id': '0011', 'typ': 'CNC', 'masa': 1489, 'roz': 25},
 {'id': '1001', 'typ': 'GGC', 'masa': 1963, 'roz': 3}]
```

Zadanie 5. - wieloatrybutowe, wielowartościowe wyszukiwanie binarne

• Zaimplementuj funkcję tworzącą, dla każdego parametru robota wektor pomocniczy. Każdy wektor pomocniczy, o rozmiarze takim samym jak wektor robotów, zawiera indeksy z wektora robotów posortowane względem danego parametru.

• Zaimplementuj funkcję wyszukiwanie binarnego realizowaną po zbiorze parametrów (każdy z dopuszczalnymi wieloma wartościami lub None, tak jak na końcu zadania nr 1). Wyszukiwanie wykonaj po każdym parametrze z osobna, korzystając z procedury opracowanej w poprzednim zadaniu. Następnie weź część wspólną wszystkich zbiorów indeksów (wykorzystaj funkcję intersection). Nie zakładaj, że wektor robotów jest jakkolwiek posortowany, a korzystaj z wektorów pomocniczych. Funkcje wyszukiwania uzupełnij o tryb wizualizacji krok-po-kroku.

In [275…
```
def bin_ser_rec(v, vsorted, imin, imax, value, feature):
    """
    Funkcja wyszukuje binarnie wszystkich indeksów robotów o zadanej warotości danego parametru

    :param list v: wektor robotów
    :param int imin: dolny indeks przeszukiwania binarnego
    :param int imax: górny indeks przeszukiwnaia binarnego
    :param value: wartość wyszukiwanego elementu
    :param feature: cecha według której sortujemy a następnie przeszukujemy listę
    :return list: zwraca listę jeśli nie ma robota o zadanej cesze lista będzie pusta
    """

    if imax >= imin:
        i = (imin + imax) // 2
        if v[vsorted[i]][feature] == value:
            if imin == imax:
                return [vsorted[i]]
            else:
                return bin_ser_rec(v, vsorted, imin, i, value, feature) +\
                        bin_ser_rec(v, vsorted, i+1, imax, value, feature)
        elif v[vsorted[i]][feature] > value:
            return bin_ser_rec(v, vsorted, imin, i-1, value, feature)
        else:
            return bin_ser_rec(v, vsorted, i+1, imax, value, feature)

    return []
```

In [276…
```
def sort_vector(v):
    """
    Funkcja zwraca N posortowanych parametrów wektorów robotów każde według kolejnego parametru
    :param v: wektor robotów
    :return list vectors: wektor posortowanych indeksów według kolejnych parametrów
    """
    vectors = list()
    keys = list(v[0].keys())
    for k in keys:
        vectors.append([v.index(i) for i in sorted(v, key=lambda x: x[k])])
    return vectors
```

In [277…
```
def intersection(lists):
    s = set(lists[0])
    for i in range(1, len(lists)):
        s = s & set(lists[i])
        print(s)
    return s
```

In [278…
```
def search(vector, features):
    """
    Binarne wyszukiwanie
    :param vector: wektor robotów
    :param features: dopuszczalne paramtry
    """
    indexes = list()
    sv = sort_vector(vector)

    for n in range(len(features)):
        f = list(features.values())[n]
        if f:
            temp_i = list()
            for i in f:
                # dodanie indexów
                temp_i += bin_ser_rec(
                    v=vector,
                    vsorted=sv[n],
                    imin=0,
                    imax=len(sv[n]),
                    value=i,
                    feature=list(features.keys())[n]
                                        )
            indexes.append(temp_i)

    indexes = intersection(indexes)
    return indexes
```

In [288…
```
robots = create_m_bots(30)
robots = [r.save_robot() for r in robots]
```

In [289…
```
# pożadane cechy
desired_features = {
    'id': ['0011','0101'],
    'typ': None,
    'masa': None,
    'roz': [10, 11, 12, 13, 14, 15, 16, 17, 18],
}

result_i = search(robots, desired_features)
```

```
{1, 21, 5}
```

In [290…
```
print('Indexy robotów spełniajacych kryteria')
print(result_i)
print('Roboty spełniające wymagania')
for i in result_i:
    print(robots[i])
```

```
Indexy robotów spełniajacych kryteria
{1, 21, 5}
Roboty spełniające wymagania
{'id': '0011', 'typ': 'WCW', 'masa': 1974, 'roz': 11}
{'id': '0101', 'typ': 'UNU', 'masa': 1415, 'roz': 16}
{'id': '0101', 'typ': 'WUU', 'masa': 750, 'roz': 12}
```