Algorytmy i Struktury Danych
Kolokwium I (29. III 2021)

Format rozwiązań

Rozwiązanie każdego zadania musi się składać z opisu algorytmu (wraz z uzasadnieniem poprawności i oszacowaniem złożoności obliczeniowej) oraz jego implementacji. Zarówno opis algorytmu jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
2. modyfikowanie testów dostarczonych wraz z szablonem,
3. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania),
4. korzystanie z wbudowanych algorytmów sortowania i zaawansowanych struktur danych (np. słowników czy zbiorów).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania. Na potrzebny analizy złożoności algorytmów można zakładać, że funkcja `Partition` z algorytmu `QuickSort` dzieli tablicę na dwie równe części (nawet jeśli dostarczona implementacja nie ma tej własności).

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 pkt. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Proszę pamiętać, że rozwiązania trochę wolniejsze niż oczekiwane, ale poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybkie ale błędnie otrzymają 0 punktów. Proszę mierzyć siły na zamiary!

Testowanie rozwiązań

Żeby przetestować rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`

Dana jest dwuwymiarowa tablica T o rozmiarach $N \times N$ wypełniona liczbami naturalnymi (liczby są parami różne). Proszę zaimplementować funkcję `Median(T)`, która przekształca tablicę T, tak aby elementy leżące pod główną przekątną nie były większe od elementów na głównej przekątnej, a elementy leżące nad główną przekątną nie były mniejsze od elementów na głównej przekątnej. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci ponad tę, która potrzebna jest na przechowywanie danych wejściowych (choć algorytm nie musi działać w miejscu). Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.

Przykład. Dla tablicy:

```
T = [ [ 2, 3, 5],
      [ 7,11,13],
      [17,19,23] ]
```

wynikiem jest, między innymi tablica:

```
T = [ [13,19,23],
      [ 3, 7,17],
      [ 5, 2,11] ]
```

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Węzły jednokierunkowej listy odsyłaczowej reprezentowane są w postaci:

```
class Node:
    def __init__(self):
        self.val  = None  # przechowywana liczba rzeczywista
        self.next = None  # odsyłacz do nastepnego elementu
```

Niech p będzie wskaźnikiem na niepustą listę odsyłaczową zawierającą parami różne liczby rzeczywiste $a_1, a_2, \ldots, a_n$ (lista nie ma wartownika). Mówimy, że lista jest k-chaotyczna jeśli dla każdego elementu zachodzi, że po posortowaniu listy znalazłby się na pozycji różniącej się od bieżącej o najwyżej k. Tak więc 0-chaotyczna lista jest posortowana, przykładem 1-chaotycznej listy jest $1, 0, 3, 2, 4, 6, 5$, a $(n-1)$-chaotyczna lista długości n może zawierać liczby w dowolnej kolejności. Proszę zaimplementować funkcję `SortH(p,k)`, która sortuje k-chaotyczną listę wskazywaną przez p. Funkcja powinna zwrócić wskazanie na posortowaną listę. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci (w sensie asymptotycznym, mierzonym względem długości n listy oraz parametru k). Proszę oszacować jego złożoność czasową dla $k = \Theta(1)$, $k = \Theta(\log n)$ oraz $k = \Theta(n)$.

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Mamy daną N elementową tablicę T liczb rzeczywistych, w której liczby zostały wygenerowane z pewnego rozkładu losowego. Ten rozkład mamy zadany jako k przedziałów $[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ takich, że i-ty przedział jest wybierany z prawdopodobieństwem c_i, a liczba z przedziału jest wybierana zgodnie z rozkładem jednostajnym. Przedziały mogą na siebie nachodzić, liczby a_i, b_i są liczbami naturalnymi ze zbioru $\{1, \ldots, N\}$. Proszę zaimplementować funkcję `SortTab(T,P)` sortująca podaną tablicę. Pierwszy argument to tablica do posortowania a drugi to opis przedziałów w postaci:

`P = [`(a_1, b_1, c_1)`, `(a_2, b_2, c_2)`, ..., `(a_k, b_k, c_k)`]`.

Na przykład dla wejścia:

```
P = [(1,5, 0.75) , (4,8, 0.25)]
T = [6.1, 1.2, 1.5, 3.5, 4.5, 2.5, 3.9, 7.8]
```

po wywołaniu `SortTab(T,P)` tablica `T` powinna być postaci:

```
T = [1.2, 1.5, 2.5, 3.5, 3.9, 4.5, 6.1, 7.8]
```

Algorytm powinien być możliwie jak najszybszy. Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.

Algorytmy i Struktury Danych
Kolokwium II (17. V 2021)

Format rozwiązań

Rozwiązanie każdego zadania musi się składać z **krótkiego** opisu algorytmu (wraz z uzasadnieniem poprawności i oszacowaniem złożoności obliczeniowej) oraz jego implementacji. Zarówno opis algorytmu jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
2. modyfikowanie testów dostarczonych wraz z szablonem,
3. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania),
4. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue`),
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).
3. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania.

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 pkt. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Proszę pamiętać, że rozwiązania trochę wolniejsze niż oczekiwane, ale poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybkie ale błędnie otrzymają 0 punktów.

Testowanie rozwiązań

Żeby przetestować rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`
Inwestor planuje wybudować nowe osiedle akademików. Architekci przedstawili projekty budynków, z których inwestor musi wybrać podzbiór spełniając jego oczekiwania. Każdy budynek reprezentowany jest jako prostokąt o pewnej wysokości h, podstawie od punktu a do punktu b, oraz cenie budowy w (gdzie h, a, b i w to liczby naturalne, przy czym $a < b$). W takim budynku może mieszkać $h \cdot (b - a)$ studentów.

Proszę zaimplementować funkcję:

```
def select_buildings(T, p):
    ...
```

która przyjmuje:

- Tablicę `T` zawierająca opisy `n` budynków. Każdy opis to krotka postaci `(h, a, b, w)`, zgodnie z oznaczeniami wprowadzonymi powyżej.
- Liczbę naturalną `p` określającą limit łącznej ceny wybudowania budynków.

Funkcja powinna zwrócić tablicę z numerami budynków (zgodnie z kolejnością w `T`, numerowanych od 0), które nie zachodzą na siebie, kosztują łącznie mniej niż `p` i mieszczą maksymalną liczbę studentów. Jeśli więcej niż jeden zbiór budynków spełnia warunki zadania, funkcja powinna zwrócić zbiór o najmniejszym łącznym koszcie budowy. Dwa budynki nie zachodzą na siebie, jeśli nie mają punktu wspólnego.

Można założyć, że zawsze istnieje rozwiązanie zawierające co najmniej jeden budynek. Funkcja powinna być możliwie jak najszybsza i zużywać jak najmniej pomięci. Należy bardzo skrótowo uzasadnić jej poprawność i oszacować złożoność obliczeniową.

Przykład. Dla argumentów:

```
T = [ (2, 1, 5, 3),
      (3, 7, 9, 2),
      (2, 8, 11, 1) ]
p = 5
```

wynikiem jest tablica: `[ 0, 2 ]`

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Dany jest graf nieskierowany $G = (V, E)$ oraz dwa wierzchołki $s, t \in V$. Proszę zaproponować i zaimplementować algorytm, który sprawdza, czy istnieje taka krawędź $\{p, q\} \in E$, której usunięcie z E spowoduje wydłużenie najkrótszej ścieżki między s a t (usuwamy tylko jedną krawędź). Algorytm powinien być jak najszybszy i używać jak najmniej pamięci. Proszę skrótowo uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def enlarge(G, s, t):
   ...
```

która przyjmuje graf `G` oraz numery wierzchołków `s`, `t` i zwraca dowolną krawędź spełniającą warunki zadania, lub `None` jeśli takiej krawędzi w `G` nie ma. Graf przekazywany jest jako lista list sąsiadów, t.j. `G[i]` to lista sąsiadów wierzchołka o numerze `i`. Wierzchołki numerowane są od 0. Funkcja powinna zwrócić krotkę zawierającą numery dwóch wierzchołków pomiędzy którymi jest krawędź spełniająca warunki zadania, lub `None` jeśli takiej krawędzi nie ma. Jeśli w grafie oryginalnie nie było ścieżki z `s` do `t` to funkcja powinna zwrócić `None`.

Przykład. Dla argumentów:

```
G = [ [1, 2],
      [0, 2],
      [0, 1] ]
s = 0
t = 2
```

wynikiem jest np. krotka: `(0, 2)`

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

W roku 2050 Maksymilian odbywa podróż przez pustynię z miasta A do miasta B. Droga pomiędzy miastami jest linią prostą na której w pewnych miejscach znajdują się plamy ropy. Maksymilian porusza się 24 kołową cysterną, która spala 1 litr ropy na 1 kilometr trasy. Cysterna wyposażona jest w pompę pozwalającą zbierać ropę z plam. Aby dojechać z miasta A do miasta B Maksymilian będzie musiał zebrać ropę z niektórych plam (by nie zabrakło paliwa), co każdorazowo wymaga zatrzymania cysterny. Niestety, droga jest niebezpieczna. Maksymilian musi więc tak zaplanować trasę, by zatrzymać się jak najmniej razy. Na szczęście cysterna Maksymiliana jest ogromna - po zatrzymaniu zawsze może zebrać całą ropę z plamy (w cysternie zmieściłaby się cała ropa na trasie).

Zaproponuj i zaimplementuj algorytm wskazujący, w których miejscach trasy Maksymilian powinien się zatrzymać i zebrać ropę. Algorytm powinien być możliwe jak najszybszy i zużywać jak najmniej pamięci. Uzasadnij jego poprawność i oszacuj złożoność obliczeniową.

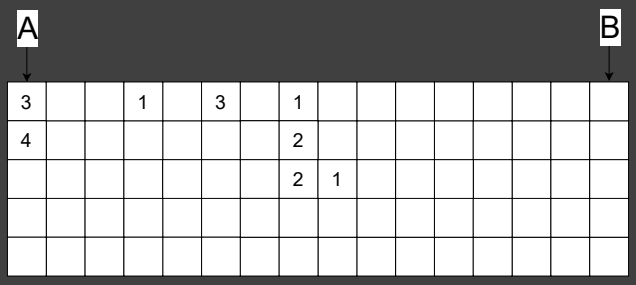
Dane wejściowe reprezentowane są jako dwuwymiarowa tablica liczb naturalnych `T`, w której wartość `T[u][v]` to objętość ropy na polu o współrzędnych (u, v) (objętość 0 oznacza brak ropy). Współrzędne u należą do zbioru $\{0, 1, \ldots, n-1\}$ a współrzędne v to zbioru $\{0, 1, \ldots, m-1\}$. Miasto A znajduje się na polu $(0, 0)$, zaś miasto B na polu $(0, m-1)$. Maksymilian porusza się jedynie po polach $(0, 0)$, $(0, 1), \ldots (0, m-1)$. Bok każdego pola ma długość 1 kilometra. Plamą ropy jest dowolny spójny obszar pól zawierających ropę. Dwa pola należą do spójnego obszaru jeśli mają wspólny bok lub są połączone sekwencją pól (zawierających ropę) o wspólnych bokach. Zakładamy, że początkowo cysterna jest pusta, ale pole $(0, 0)$ jest częścią plamy ropy, którą można zebrać przed wyruszeniem w drogę. Zakładamy również, że zadanie posiada rozwiązanie, t.j. da się dojechać z miasta A do miasta B.

Algorytm należy zaimplementować w funkcji:

```
def plan(T):
   ...
```

która przyjmuje tablicę z opisem zadania i zwraca listę współrzędnych `v` pól na których należy zatrzymać cysternę w celu zebrania ropy (cysterna porusza się po tylko polach $(0, v)$, więc wystarczy zwrócić współrzędną `v`). Lista powinna być posortowana w kolejności postojów. Postój na polu $(0, 0)$ również jest częścią rozwiązania.

Przykład. Dla mapy (w pustych polach są wartości 0, a więc brak ropy):



wynikiem jest np. lista: `[0, 5, 7]`

Algorytmy i Struktury Danych
Kolokwium III (14.VI 2021)

Format rozwiązań

Rozwiązanie każdego zadania musi się składać z **krótkiego** opisu algorytmu (wraz z uzasadnieniem poprawności i oszacowaniem złożoności obliczeniowej) oraz jego implementacji. Zarówno opis algorytmu jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
2. modyfikowanie testów dostarczonych wraz z szablonem,
3. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania),
4. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue`),
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).
3. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania.

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 punktów. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Proszę pamiętać, że rozwiązania trochę wolniejsze niż oczekiwane, ale poprawne, mają szanse na otrzymanie 1 punktu. Rozwiązania szybkie ale błędne otrzymają 0 punktów.

Testowanie rozwiązań

Żeby przetestować rozwiązania zadań należy wykonać:

```
python3 zad1.py

python3 zad2.py

python3 zad3.py
```

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`
Carol musi przewieźć pewne niebezpieczne substancje z laboratorium x do laboratorium y, podczas gdy Max musi zrobić to samo, ale w przeciwną stronę. Problem polega na tym, że jeśli substancje te znajdą się zbyt blisko siebie, to nastąpi reakcja w wyniku której absolutnie nic się nie stanie (ale szefowie Carol i Max nie chcą do tego dopuścić, by nie okazało się, że ich praca nie jest nikomu potrzebna). Zaproponuj, uzasadnij i zaimplementuj algorytm planujący jednocześnie trasy Carol i Maxa tak, by odległość między nimi zawsze wynosiła co najmniej d. Mapa połączeń dana jest jako graf nieskierowany, w którym każda krawędź ma dodatnią wagę (x i y to wierzchołki w tym grafie). W jednostce czasu Carol i Max pokonują dokładnie jedną krawędź. Jeśli trzeba, dowolne z nich może się w danym kroku zatrzymać (wówczas pozostaje w tym samym wierzchołku). Carol i Max nie mogą równocześnie poruszać się tą samą krawędzią (w przeciwnych kierunkach).

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def keep_distance(M, x, y, d):
   ...
```

która przyjmuje numery wierzchołków `x` oraz `y`, minimalną odległość `d` i graf reprezentowany przez kwadratową, symetryczną macierz sąsiedztwa `M`. Wartość `M[i][j] == M[j][i]` to długość krawędzi między wierzchołkami `i` oraz `j`, przy czym `M[i][j] == 0` oznacza brak krawędzi między wierzchołkami. W macierzy nie ma wartości ujemnych. Funkcja powinna zwrócić listę krotek postaci:

```
[(x, y), (u1, v1), (u2, v2), ..., (uk, vk), (y, x)]
```

reprezentującą ścieżki Carol i Max. W powyższej liście element `(ui, vi)` oznacza, że Carol znajduje się w wierzchołku `ui`, zaś Max w wierzchołku `vi`. Można założyć, że rozwiązanie istnieje.

Przykład. Dla argumentów:

```
M = [
      [0, 1, 1, 0],
      [1, 0, 0, 1],
      [1, 0, 0, 1],
      [0, 1, 1, 0],
    ]
x = 0
y = 3
d = 2
```

wynikiem jest na przykład lista: `[(0, 3), (1, 2), (3, 0) ]`

Podpowiedź. Proszę rozważyć nowy graf, być może z dużo większą liczbą wierzchołków niż graf wejściowy.

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Dane jest drzewo BST zbudowane z węzłów

```
class BNode:
  def __init__( self, value ):
    self.left = None
    self.right = None
    self.parent = None
    self.value = value
```

Klucze w tym drzewie znajdują się w polach `value` i są liczbami całkowitymi. Mogą zatem mieć wartości zarówno dodatnie, jak i ujemne. Proszę napisać funkcję, która zwraca wartość będącą minimalną możliwą sumą kluczy zbioru wierzchołków oddzielających wszystkie liście od korzenia w taki sposób, że na każdej ścieżce od korzenia do liścia znajduje się dokładnie jeden wierzchołek z tego zbioru. Zakładamy że korzeń danego drzewa nie jest bezpośrednio połączony z żadnym liściem (ścieżka od korzenia do każdego liścia prowadzi przez co najmniej jeden dodatkowy węzeł). Jako liść jest rozumiany węzeł `W` typu `BNode` such that `W.left = W.right = None`.

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def cutthetree(T):
   ...
```

która przyjmuje korzeń danego drzewa BST i zwraca wartość rozwiązania. Nie wolno zmieniać definicji `class BNode`.

Przykład. Dla drzewa BST, utworzonego przez dodawanie do pustego drzewa kolejno elementów z kluczami 10, 3, 15, 11, 17, -1, -5, 0 wynikiem jest 14 (usuwamy węzły o wartościach -1 oraz 15).

[2pkt.] Zadanie 3.

Szablon rozwiązania: `zad3.py`

Dany jest ważony graf nieskierowany reprezentowany przez macierz T o rozmiarach $n \times n$ (dla każdych i, j zachodzi $T[i][j] = T[j][i]$; wartość $T[i][j] > 0$ oznacza, że istnieje krawędź między wierzchołkiem i a wierzchołkiem j z wagą $T[i][j]$). Dana jest także liczba rzeczywista d. Każdy wierzchołek w G ma jeden z kolorów: zielony lub niebieski. Zaproponuj algorytm, który wyznacza największą liczbę naturalną ℓ, taką że w grafie istnieje ℓ par wierzchołków $(p, q) \in V \times V$ spełniających warunki:

1. q jest zielony, zaś p jest niebieski,
2. odległość między p i q (liczona jako suma wag krawędzi najkrótszej ścieżki) jest nie mniejsza niż d,
3. każdy wierzchołek występuje w co najwyżej jednej parze.

Rozwiązanie należy zaimplementować w postaci funkcji:

```
def BlueAndGreen(T, K, D):
    ...
```

która przyjmuje:

T: graf reprezentowany przez kwadratową macierz sąsiedztwa, gdzie wartość 0 oznacza brak krawędzi, a liczba większa od 0 przedstawia odległość pomiędzy wierzchołkami,

K: listę przedstawiającą kolory wierzchołków,

D: odległość o której mowa w warunku 2 opisu zadania.

Funkcja powinna zwrócić liczbę ℓ omawianą w treści zadania.

Przykład. Dla argumentów:

```
T = [
      [0, 1, 1, 0, 1],
      [1, 0, 0, 1, 0],
      [1, 0, 0, 0, 1],
      [0, 1, 0, 0, 1],
      [1, 0, 1, 1, 0],
    ]
K = ['B', 'B', 'G', 'G', 'B']
D = 2
```

wynikiem jest wartość 2.

Maksymalny przepływ. Do treści zadania dostarczony jest plik zad3EK.py, w którym zaimplementowany jest algorytm Edmondsa-Karpa obliczający maksymalny przepływ w grafie, w następującej postaci:

```
edmonds_karp(graph, source, sink)
```

który przyjmuje:

graph: graf reprezentowany przez kwadratową macierz sąsiedztwa, gdzie wartość oznacza pojemność (ang. capacity) danej krawędzi,

source: numer wierzchołka-źródła,

sink: numer wierzchołka-ujścia.