MDM Lista 12

Weronika Jakimowicz

ZAD. 1.

Zakładamy, że dostajemy listę sąsiedztwa wierzchołków. Skok niżej to zejście w dół krawędziami w drzewie DFS i wrócenie jak najwyżej w górę za pomocą krawędzi niebędących w drzewie.

```
odwiedzone = [0] * n
skoki = [0] * n

nozyczki = -1 # tutaj zapiszemy wierzcholek rozspajajacy

def dfsik(j, studnia):
    # j - aktualny wierzcholek
    # studnia - glebokosc na jaka zeszlismy w drzewie dfs
    odwiedzone[j] = 1

    # skoki to bedzie tabela ktora mowi nam jak wysoko mozemy wskoczyc
    # z danego wierzcholka
    skoki[j] = studnia

    zmienna = 0

    for i in sasiedzi[j]:
        if not odwiedzone[i]:
            zmienna++
            # jezeli z dziecka mozemy skoczyc wyzej niz z rodzica,
            # to mozemy zejsc nizej i dopiero skakac, wiec chcemy max
            # ze skokow ze skoku teraz i skokow z nizszych
            skoki[j] = max(skoki[j], dfsik(i, studnia+1))

    # jezeli jestesmy w korzeniu i ma on wiecej niz jedno dziecko, to rozspaja
    # calosc, bo chcac przejsc z jednej grupy galezi do drugiej musimy przez
    # niego przejsc
    if studnia == 0 and zmienna > 1:
        nozyczki = j

    # jesli z j mozemy wyskoczyc wyzej niz korzen, to mozemy tak przekrzywic graf,
    # ze j jest korzeniem i jego usuniecie rozspaja graf
    if skoki[j] <= studnia
        nozyczki = j

    return skoki[j]


def rozspojnia():
    dfsik(0, 0) # wystarczy jeden raz, bo zakladamy spojnosc
    if nozyczki == -1:
        print("NIE ISTNIEJE WIERZCHOLEK ROZSPOJNIAJACY")
    else:
        print(nozyczki)
```

ZAD. 2.

Grafy dwudzielne mają liczbę chromatyczną 2, więc sprawdzamy, czy nasz graf może zostać pomalowany kolorami 1 i −1.

```
odwiedzone = [0] * n
kolorowanie = [0] * n

def dfsik(j, kolor):
    odwiedzone[j] = 1
    kolorowanie[j] = kolor

    for i in sasiedzi[j]:
        if not odwiedzone[i]:
            if not dfsik(i, kolor * -1):
                # jezeli kolorowanie w nastenym wierzcholku sie nie zgadza,
                # to teraz tez sie nie bedzie zgadzac
                return False

        if kolorowanie[i] == kolorowanie[j]:
            # jezeli kolorowanie w nastepnym jest takie samo jak teraz, to mamy
            # sasiednie wierzcholki tego samego koloru, czego nie chcemy
            return False

    # jak na razie nam sie zgadza, wiec moze byc dwudzielny
    return True
```

ZAD. 3.

```
odwiedzone = [0] * n
kolorowanie = [0] * n

pom = []

def dfsik(j):
    odwiedzone[j] = 1
    for i in sasiedzi[j]:
        if not visited[i]:
            # idziemy jak najglebiej w grafie
            dfs(i)

    # doklejamy na koniec do pom, wiec bedzie w odwrotnej kolejnosci
    pom.append(i)

def topo():
    for i in range(len(pom)):
        kolejnosc.append(pom[n-i])
```

ZAD. 7.

Pokaż, w jaki sposób można znaleźć najdłuższe drzewo rozpinające grafu z wagami.

Jedną możliwością jest odwrócenie sortowania kosztów i wykonanie algorytmu z zadania 6. Możemy też koszt każdej z krawędzi pomnożyć przez −1 i wykonać zadanie 6. wprost.

ZAD. 9

Oryginalnie zapisywalibyśmy tylko najmniejszą odległość między wierzchołkami w macierzy d. W tym algorytmie dodajemy jeszcze macierz p, która będzie dopisywać do listy wierzchołków które trzeba odwiedzić, żeby zminimalizować drogę.

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            if (d[j][i] < INF && d[i][k] < INF):
                nc = d[j][i] + d[i][k]
                if d[j][k] > nc:
                    d[j][k] = nc
                    p[j][k] = p[j][i] + p[i][k]
```

ZAD. 12.

Niech G będzie grafem z nieujemnymi wagami na krawędziach. Niech $MST(G)$ oznacza długość najlżejszego drzewa spinającego w G, a $TSP(G)$ oznacza długość najkrótszej drogi komiwojażera w G (komiwojażer może odwiedzać wierzchołki wielokrotnie). Wykaż, że

$$MST(G) \leq TSP(G) \leq 2 \cdot MST(G)$$

$2 \cdot MST(G) \geq TSP(G)$ - w najgorszym przypadku będziemy szli przez najmniejsze drzewo rozpinające do najodleglejszych wierzchołków i wracali do korzenia po tej samej drodze, czyli po każdej "gałęzi" przejdziemy dwukrotnie.

$TSP(G) \geq MST(G)$ - w najlepszym przypadku po prostu przejdziemy jeden raz po każdej gałęzi najmniejszego drzewa rozpinającego, bo odwiedza ono każdy wierzchołek jednokrotnie po jak najkrótszej drodze.

ZAD. 14.

Graf jest krawędziowo k-spójny gdy jest spójny i usunięcie z niego co najwyżej $(k-1)$ krawędzi nie rozspójnia go. Używając przepływów w sieciach pokaż, że G jest krawędziowo k-spójny $\iff$ między dwoma wierzchołkami istnieje k krawędziowo rozłącznych dróg.

$\Longleftarrow$

Niech G będzie grafem takim, że między dowolnymi dwoma wierzchołkami jest k krawędziowo rozłącznych dróg. Weźmy $F \subseteq E(G)$ zbiór krawędzi taki, że $G \setminus F$ jest grafem rozłącznym. Z założenia wiemy, że dla dowolnych $v, w \in G$ istnieje k rozłącznych dróg w G między tymi dwoma wierzchołkami, więc aby $G \setminus F$ było rozłączne, to musimy co najmniej jedną krawędź z każdej z tych k dróg wyjąć. Czyli $|F| \geq k$.

$\Longrightarrow$

Niech G będzie grafem krawędziowo k-spójnym. Zamieńmy ten graf na digraf rozbijając dowolną krawędź $\{v, w\} \in G$ na dwie skierowane krawędzie: (v, w) oraz (w, v). Chcemy dodać do niego źródło s oraz ujście t i każdej krawędzi przypiszemy pojemność 1. Wiemy, że maksymalny przepływ w grafie jest równy pojemności minimalnego cięcia. W przypadku grafu k-spójnego minimalne cięcie, które rozłącza wszystkie drogi między s a t jest równe k, czyli maksymalny przepływ w G jest równy k, a to znaczy, że idąc od źródła do ujścia musimy odwiedzić co najmniej k różnych krawędzi, bo każda z nich może pomieścić co najwyżej 1.