Lista 12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X	X	X	X	X	X	X		X					X	

Zadanie 1

Wierzchołek jest rozspajający, jeśli jego usunięcie rozspaja graf.
Do skonstruowania aklogrytmu skorzystajmy z DFS. Zauważmy, że wierzchołki w drzewie przeszukiwań będą rozspajające w dwóch przypadkach:

- jeśli wierzchołek u jest korzeniem i ma conajmniej dwójkę dzieci,
- jeśli wierzchołek u nie jest korzeniem i ma takie dziecko v, że z podrzewa ukorzenionego w v nie prowadzi krawędź (poza drzewem przeszukiwań) do któregoś przzodka u.

```
visited[] -- przechowuje informację o tym, czy dany wierzchołek został
odwiedzony
disc[] -- przechowuje czas odwiedzenia wierzchołka
low[] -- minimalny czas odwiedzenia wierzchołka, do którego można dojść
z poddrzewa v używając co najwyżej jednej krawędzi niedrzewowej.
ap[] -- przechowuje wierzchołki rozcinające
parent[v] -- przechowuje rodziców danych wierzchołków

czas = 0

AP(u):
    Niech dzieci = 0 (liczba dzieci u w drzewie DFS)

    Oznacz u jako odwiedzony
    Ustaw czas odwiedzenia wierzchołka disc[u] = low [u] = czas
    czas++

    Dla wszystkich v sąsiadów u:
        Jeśli v nieodwiedzony
            dzieci++
```

```
            Ustaw parent[v] = u
            Wywołaj AP(u)

            low[u] = min(low[u], low[v])
            (jeśli istnieje gałąź do przodka to wierzchołek
            można odwiedzić wcześniej)

            Jeżeli parent[u] == NULL oraz dzieci > 0
                Dodaj u do ap[]

            Jeżeli parent[u] != NULL oraz low[v] >= disc[u]
                Dodaj u do ap[]

        Wpp.
            low[u] = min(low[u], disc[v])
```

Zadanie 2

Graf jest dwudzielny, jeżeli wszystkie jego wierzchołki można pokolorować na dwa kolory w taki sposób, że żadne dwa sąsiednie wierzchołki nie są tego samego koloru.

```
G - graf w postaci listy list sąsiadów
Colour - tablica zawierająca 0 lub 1 -- kolor każdego wierzchołka
Visited - tablica zawierająca informację o tym czy dany wierzchołek zost
odwiedzony

czyDwudzielny(v):

Dla u będącego sąsiadem v:
    Jeśli u nie był odwiedzony:
        Oznacz u jako odwiedzony
        Ustaw Colour[u] jako !Colour[v]

        Jeśli !czyDwudzielny(u)
            Zwróć fałsz

    Jeśli Colour[u] == Colour[v]
        Zwróć fałsz

Zwróc prawdę
```

Złożoność proceudury jest taka sama jak DFS z tą różnicą, że powyższa procedura dodatkowo koloruje i sprawdza kolorowanie.

Zadanie 3

```
D - digraf acykliczny
Q - kolejka wierzchołków u, których indeg(u) = 0.
S - lista posortowanych wierzchołków

Idx = 0

Dopóki Q nie jest puste:
    Usuń v - pierwszy element z Q
    Dodaj v do S

    Dla każdego u o krawędzi e od v do u:
        Usuń e z D
        Jeżeli indeg(u) == 0
            Wstaw u do Q

Zwróc S
```

Aby uzyskać topologicznie posortowane wierzchołki iterujemy się poprzez wszystkie wierzchołki w grafie (zaczynając od najmniejszego stopnia wchodzącego wierzchołka) a następnie usuwamy kolejne krawędzie wychodzące z tych wierzchołków i jeśli któryś z sąsiadów wierzchołków z kolejki Q osiągnie stopień wchodzący równy zero to zaczynamy rozpatrywać krawędzie łączące dany wierzchołek z jego sąsiadami.

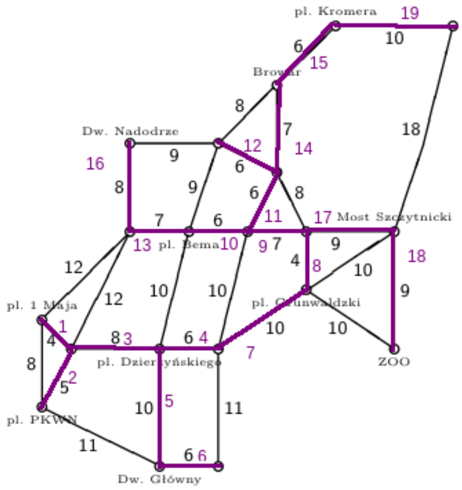
W ten sposób, jeśli graf nie posiada cyklu, po zakończeniu procedury otrzymamy pusty graf.

Ponieważ dokładnie raz rozpatrujemy każdy z wierzchołków i każdą z krawędzi to złożoność wynosi $O(n + m)$.

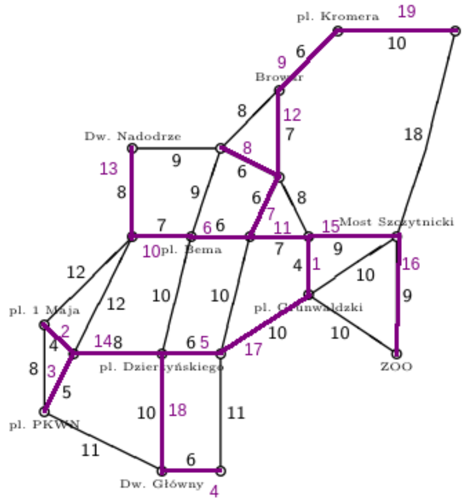
Zadanie 4

Algorytm Kruskala polega na posortowaniu krawędzi i dołączaniu zaczynając od najmniejszej tak długo, jak nie powstanie cykl.

Algorytm Prima-Dijkstry



Algorytm Kruskala



Zadanie 6

Krok 1: Niech $c(e_1) > c(e_2) > ... > c(e_m)$;
Krok 2: $T := E(G)$;
    for $e_1, e_2, ..., e_m$ **do**
    if $T \setminus e_i$ jest grafem spójnym
      then $T := T \setminus e_i$.

Zauważmy, że algorytm usuwa krawędzie tak długo jak nie rozspójnia grafu. Zatem T na wyjściu będzie grafem spójnym bez cykli (jakby miał cykl to istniałaby krawędź, którą można usunąć i nie rozspójniałaby grafu). Zatem T to drzewo rozpinające grafu G.

Algorytm usuwa krawędzie w kolejności malejącej. Za każdym razem, gdy usuwana jest krawędź, graf jest spójny i usuwana jest najdroższa krawędź w danym cyklu. Skoro usuwane są najdroższe krawędzie każdego cyklu to wszystkie te usunięte krawędzie nie należą do najkrótszego drzewa rozpinającego.

Zatem T po zakończeniu algorytmu będzie najkrótszym drzewem rozpinającym.

Zadanie 7

Aby znaleźć najdłuższe drzewo rozpinające możemy wyokrzystać algorytm Kruskala i uporządkować krawędzie w odwrotny sposób.

```
c(e) - waga krawędzi e.

Uporządkuj krawędzie w kolejności c(e1) > c(e2) > ... > c(en).
Niech T -- pusty zbiór

Dla i = 1,2,...,n
    Jeżeli T z krawędzią {ei} nie zawiera cyklu
        Dodaj krawędź {ei} do T.

Zwróc T
```

Zadanie 8

```
n = 32

Dla j= 1,2,...,n:
    Dla i = 1,2,...,n:
        Dla j = 1,2,...,n:
            Jeżeli (M[i] && (1 << (n-k))) & (M[k] && (1 << (n-j)))
                M[i] = M[i] | (1 << ( n-j))
```

Zadanie 9

```
d - długości najkrótszych dróg (na końcu algorytmu)
Nxt - tablica wskazująca na następny element
      Zainicjalizowana w ten sposób:
      Nxt[i,j] = j jeśli istnieje krawędź (i,j)
      Wpp. Nxt[i,j] = -1

Warshall-modyfied:
    Dla k = 1,2,...,n
        Dla i = 1,2,...,n
            Dla j = 1,2,...,n
                Jeśli d[i,j] > d[i,k] + d[k,j]
                    d[i,j] = d[i,k] + d[k,j]
```

```
Nxt[i,j] = Nxt[i,k]
```

Aby otrzymać drogę wystarczy przejść po Nxt dopóki nie dojdziemy do wierzchołka docelowego:

```
D - droga z wierzchołka u do v

Droga(u,v):
    Dodaj u do D
    Dopóki u != v
        u = Nxt[u,v]
        Dodaj u do D
```

Zadanie 14

14. (Twierdzenie Mengera) 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 wtedy i tylko wtedy gdy między każdymi dwoma wierzchołkami istnieje k krawędziowo rozłącznych dróg.

G jest krawędziowo k spójny $\iff$ między każdymi dwoma wierzchołkami istnieje k krawędziowo rozłącznych dróg.

Możemy równoważnie pokazać, że minimalna liczba krawędzi potrzebna do odzielenia dwóch różnych wierzchołków jest równa maksymalnej liczbie krawędziowo rozłącznych dróg między tymi dwoma wierzchołkami.

Weźmy dwa różne wierzchołki s, t. Niech odpowiednio s to będzie źródło, a t – ujście.
Następnie, każdą z nieskierowanych krawędzi $\{x, y\}$ zamieńmy na krawędzie (x, y) i (y, x).

Niech c będzie funkcją na grafie G, która każdej krawędzi przyporządkowuje pojemność 1.

Niech f będzie maksymalnym przepływem w grafie G, a S, T odpowiadającym przekroju o minimalnej pojemności (z Tw. Forda-Fulkersona).

Maksymalny przepływ wprost wyznacza nam ilość krawędziowo rozłącznych dróg z s do t, gdyż każda z krawędzi ma przypisaną pojemność 1, stąd wszystkie krawędzie na danej drodze mają pojemność 1, a przepływ drogi również jest równy 1.

Z drugiej strony S, T jest przekrojem o minimalnej pojemności. Skoro każda krawędź ma pojemność 1, a pojemność przekroju to suma pojemności wszystkich krawędzi, to wartość $c(S, T)$ jest minimalną liczbą krawędzi potrzebną do odseparowania dwóch wierzchołków.

Z Tw. Forda-Fulkersona $c(S, T)$, co należało pokazać.

Zadanie 10

d[z] = 0

Q = V(G)

Dopóki Q nie jest puste:

  u = zdejmij min (Q)

  Dla każdego wierzchołka v będącego sąsiadem u:

    nd = d[v] + waga (u, v)

    Jeśli nd < d[v]:

      d[v] = nd

      prev[v] = {u}

    W p.p. jeśli nd = d[v]:

      prev[v] = prev[v] ∪ {u}

Zwróć tablicę prev

Zadanie 11

11 Skorzystamy z algorytmu Kruskala: do znalezionego MST dodamy pierwszą krawędź odrzuconą przez algorytm (a zatem najmniejszą spośród odrzuconych)

D-d poprawności:

Załóżmy nie wprost, że znalezione 1-drzewo nie jest najkrótsze. Istnieje wtedy 1-drzewo o sumie wag mniejszej niż zwrócone przez przedstawiony algorytm. Składa się ono z jakiegoś MST i dodanej krawędzi. (dłuższej niż zwracana wcześniej) Usuńmy tę dodaną krawędź. Jeżeli waga tej krawędzi wynosiła c, a suma ~~wag MST~~ tego 1-drzewa t, to MST ma długość t-c. Zauważmy jednak, że zaproponowany algorytm zwrócił 1-drzewo o wadze s z dodaną krawędzią o wadze d. MST zawarte w tym drzewie ma długość s-d. Skoro $s>t$ oraz ~~d~~ $d<c$ to $s-d > t-c$, co stoi w sprzeczności ~~z~~ z tym, że obie te wartości oznaczają równe długości MST.

Zadanie 12

a) $MST(G) \leq TSP(G)$

Załóżmy nie wprost, że $TSP(G) < MST(G)$.

Jednakże oznacza to, że istnieje ~~graf spójny~~ MST(G) lżejsze drzewo spinające w G niż ~~MST(G)~~ to, które ~~opisuje~~

(Drzewo to otrzymujemy z najkrótszej drogi komiwojażera).

Zatem otrzymujemy sprzeczność z założeniem, że $MST(G)$ to długość najlżejszego drzewa spinającego w G.

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

Mając najlżejsze drzewo spinające w G, zawsze będziemy w stanie odwiedzić w nim każdy wierzchołek przechodząc każdą krawędź co najwyżej dwa razy (wystarczy przechodzenie takiego drzewa wgłąb).

tags: `mdm`