8. Rozpięte drzewa - algorytmy

Algorytm z poprzedniego wykładu 7 musi rozstrzygać czy krawędź jest mostem. Można tego uniknąć

WE: $G=(V,E)$, $u \in V$
(1) $U := \{u\}$, $F := \emptyset$
(2) Jeśli $\exists x \in U, \exists y \in V \setminus U : \alpha = xy$,
to (i) $U := U \cup \{y\}$, (ii) $F := F \cup \{\alpha\}$, i idź do (2)
WY: $T=(U,F)$

TW1 G jest spójny $\Leftrightarrow$ T jest rozp. drzewem, tzn. $U=V$. □

W p.(2) jest wiele możliwości wyboru x i y.
Rozważmy 2 szczególne realizacje:

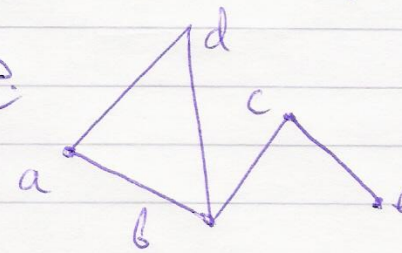
BFS (breadth-first-search)

U jest kolejką obsługiwaną wedle zasady „kto pierwszy, ten lepszy".

Każdy nowy el. w U dostaje „numerek" - $\mathrm{bf}(x)$.
Ponadto, BFS mierzy odległości od u.

Def Odległość wierzchołków $u, v \in V$ to długość (= # krawędzi) najkrótszej ścieżki o końcach u i v.

Np.



$d_G(a,b)=1$, $d_G(a,e)=3$
$d_G(d,c)=2$

WE: $G=(V,E)$, $u \in V$, u - korzeń drzewa T

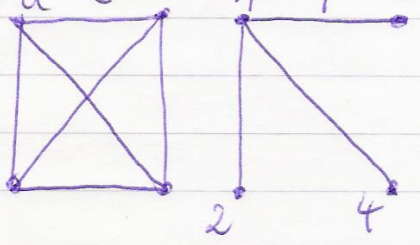
(1) $i=1$, $U=\{u\}$, $D(u)=0$, $bf(u)=1$, $F=\emptyset$, $T=(U,F)$

(2) Jeśli $d_G(u)=0$, to STOP. Znajdź $\alpha=xy \in E$, $x \in U$, $y \in V-U$, x ma najmniejszą liczbę $bf(x)$, oraz (i) $bf(y)=i+1$, (ii) $D(y)=D(x)+1$, (iii) $U=U \cup \{y\}$, (iv) $F=F \cup \{\alpha\}$, (v) $i:=i+1$, (vi) go to (2).

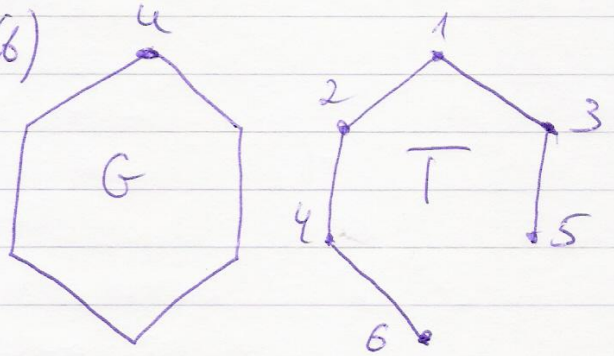
WY: $T=(U,F)$, $(D(y), y \in U)$.

TW2 G jest spójny $\Leftrightarrow U=V$ w alg. BFS
Jeśli G jest spójny, to $d_G(u,y)=d_T(u,y)=D(y)$ $\forall u \in U$ □

Prz 1 (a)



(b)

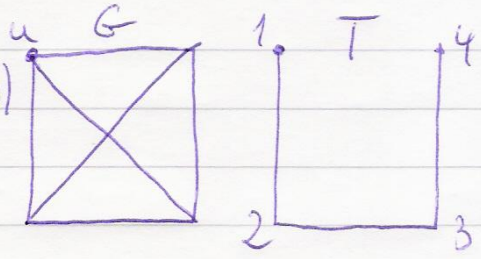


DFS (depth-first-search), (backtracking)

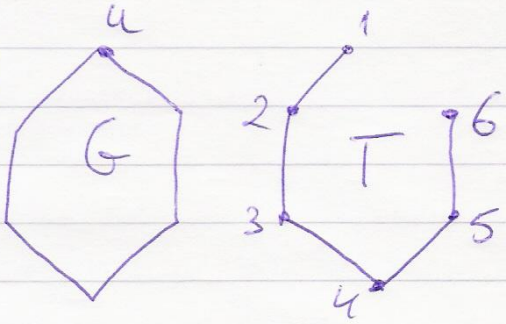
U jest stosem: „ostatni będą pierwszymi".

Opis algorytmu taki sam jak BFS, ale z 1 wyjątkiem: wybieramy x o największym „numerku" $df(x)$.

Prz2 (a)



(b)



Grafy dwudzielne - zastosowanie BFS

TW3. Graf jest dwudzielny $\Leftrightarrow$ nie posiada nieparzystych cykli.

d: $\Rightarrow$ $V = X \cup Y$, $E \subseteq \{xy : x \in X, y \in Y\}$

Każdy cykl przechodzi na przemian przez X i Y, więc musi być parzysty

$\Leftarrow$ zał., że G nie ma cykli nieparz, $u \in V$, G-spójny
$X = \{x \in V : d_G(u,x) = \text{parz}\}$, $Y = \{y \in V : d_G(u,y) = \text{nieparz}\}$.

$V = X \cup Y$; nwp., że $\exists\, ab \in E$, $a \in X$, $b \in X$

Wtedy powstanie nieparz. cykl - sprz.

(odcinki uz muszą być różne, więc odcinki za, zb mają tę samą parzystość)

Podobnie $E \cap \binom{Y}{2} = \emptyset$. Stąd G jest dwudzielny $\square$

Alg. BFS może służyć do rozstrzygania czy G jest dwudzielny. Wystarczy na końcu sprawdzić czy istnieje krawędź między x i y takimi, że $D(x) = D(y) \bmod 2$.

Załóżmy teraz, że mamy graf z wagami, tzn. $w: E \to R^+ \cup \{0\}$. Znaleźć minimalne rozpięte drzewo, tzn. $T \subseteq G : V(T) = V(G)$, T - drzewo, $w(T) := \sum_{e \in T} w(e)$ jest min.

Algorytm Kruskala (zachłanny, „greedy")

WE: $G = (V, E)$, w, G - spójny

(1) $F := \emptyset$

(2) Jeśli $\exists e \in E \setminus F$: $F \cup \{e\}$ nie zawiera cyklu, to wybierz taką e o min. $w(e)$, $F := F \cup \{e\}$, go to (2).

WY: $T = (V, F)$

TW4 Alg. Kruskala znajduje min. rozpięte drzewo T.

d: niech $T = \{e_1, e_2, \dots, e_{n-1}\}$; ponieważ $\forall i = 1, \dots, n-1$: e_i łączy 2 składowe lasu zbudowanego z $(V, \{e_1, \dots, e_{i-1}\})$, to T jest rozpiętym drzewem.

Niech T^* będzie min. rozp. drzewem o najw. $|T^* \cap T|$.
Przyp, że $T^* \neq T$. Niech $e_k \notin T^*$, ale $e_1, \dots, e_{k-1} \in T^*$.

$T^* + e_k$ zawiera cykl, a ten cykl - krawędź $f \notin T$.
$T^{**} = T^* + e_k - f$ jest drzewem, więc $c(e_k) \geq c(f)$, bo $c(T^{**}) \geq c(T^*)$
Skoro algorytm wybrał e_k, a nie f, to mamy, że $c(e_k) = c(f)$,
więc T^{**} też jest min. i $|T^{**} \cap T| = |T^* \cap T| + 1$ - sprzeczność $\square$

Alg. Kruskala nie rozszerza jednego drzewa (tak jak BFS, czy DFS), ale „sadzi" wiele drzew, które łączy ze sobą.
Wada – musi rozstrzygać, czy $F \cup \{e\}$ zawiera cykl.

Algorytm Prima (modyfikacja alg. Kruskala)

WE: $G=(V,E)$, w, G-spójny, $u \in V$

(1) $i=0$, $U_0=\{u\}$, $F_0=\emptyset$, $T_0=(U,F)$

(2) Dopóki $U_i \neq V$, wykonaj:

(i) znajdź krawędź $e_{i+1}=\{x,y\}$ o najmn. wadze:
$x \in U_i$, $y \notin U_i$

(ii) $U_{i+1}=U_i \cup \{y\}$, $F_{i+1}=F_i \cup \{e_{i+1}\}$, $T_{i+1}=(U_{i+1},F_{i+1})$

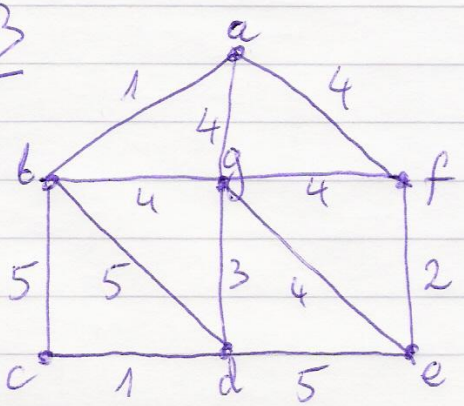
(iii) $i:=i+1$

WY: $T:=T_{n-1}$

TW5 Alg. Prima znajduje min. rozpięte drzewo.

d: podobny do dowodu TW4 – pomijamy. □

Pr3



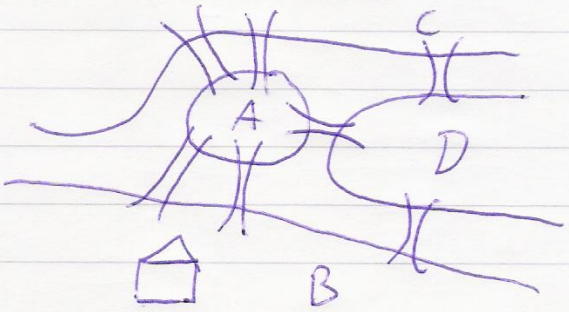
Kruskal: ab, cd, ef, dg, eg, ag

Prim: ab, af, fe, eg, gd, dc

$w(T)=15$

9. Spacerem przez graf.

Problem mostów królewieckich (Euler, XVIII w.)



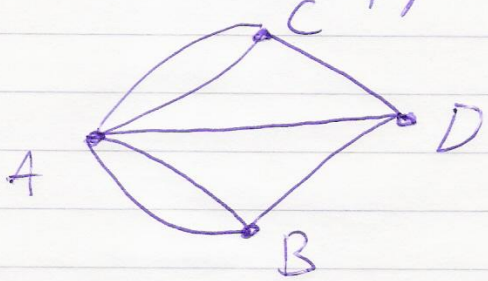
Czy można wyjść z domu, przejść każdy most raz i wrócić do domu?

NIE! Dlaczego?

Spacer w grafie to ciąg różnych krawędzi $e_1, e_2, \ldots$ taki, że $e_i \cap e_{i+1} \neq \emptyset$.

Obchód Eulera to spacer przez wszystkie krawędzie, który powraca do wierzchołka wyjścia.

Problem mostów królewieckich pyta o obchód Eulera w multigrafie



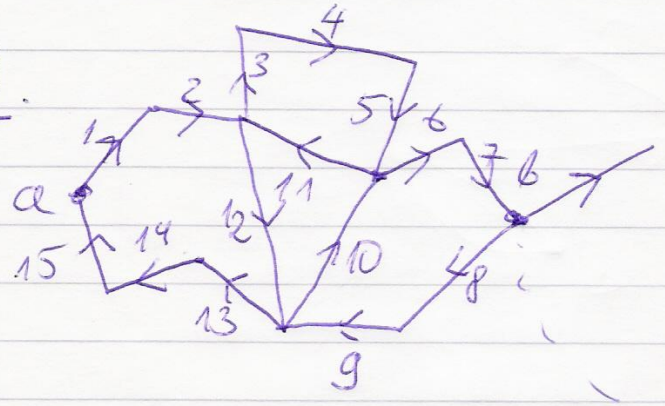
Warunki konieczne: spójność, parzystość stopni

TW 1 (Euler, ~~1736~~) Multigraf G ma obchód Eulera wtedy G jest spójny i wszystkie stopnie są parzyste.

$\Rightarrow$: $\Rightarrow$ oczywiste

$\Leftarrow$ Wyruszamy z dowolnego wierzchołka a i idziemy tak długo jak się da. Kiedy utkniemy, to będziemy znowu w wierzchołku a (bo parzystość stopni).
Jeśli dotąd nie przeszliśmy wszystkich krawędzi, to istnieje świeża krawędź incydentna z jakimś wierzchołkiem b z dotychczasowego spaceru. Wznawiamy spacer w b, aż do zatrzymania. Oba spacery można połączyć.
Jeśli nie wyczerpują wszystkich krawędzi, to rozpoczynamy kolejny spacer, itd. $\square$

Przł.



Def Otwarty obchód Eulera z u do v to spacer z u do v przez wszystkie krawędzie grafu.

Twł Spójny multigraf G ma otwarty obchód Eulera z u do v wtedy u i v są jedynymi wierzchołkami o nieparzystym stopniu.

d: dodajemy krawędź uv (nawet jeśli już istnieje – wtedy zwiększamy jej krotność o 1)
i stosujemy TW1 □

W muzeum obrazy wiszą po obu stronach korytarzy. Czy można wyznaczyć trasę zwiedzania tak, by każdy korytarz przejść dokładnie 2 razy.

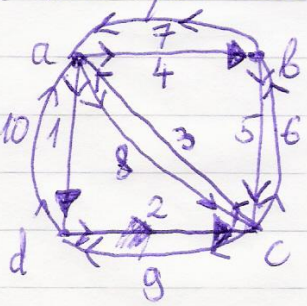
TW3 W każdym spójnym multigrafie można wyznaczyć ciąg krawędzi $e_1, e_2, \ldots$ takich, że $\forall i \; e_i \cap e_{i+1} \neq \emptyset$ oraz każda krawędź występuje 2 razy.

d: podwajamy wszystkie krawędzie i stosujemy TW1. □

Algorytm, który wyznacza ciąg $e_1, e_2, \ldots$ taki jak w TW3 o dodatkowej własności, że każda krawędź jest przemierzana raz w każdym kierunku.

1. Rozpoczynamy z dowolnego wierzchołka v_0 wzdłuż dowolnej krawędzi
2. Krawędzi, wzdłuż której wejdziemy po raz pierwszy do danego wierzchołka, oznaczamy specjalnie.
3. Wychodzimy z aktualnego wierzchołka w dowolnym, dotychczas nieprzemierzonym kierunku, ale wzdłuż krawędzi oznaczonej specjalnie idziemy tylko wtedy, gdy nie ma już innej możliwości.

Pr2



adcabcbacda