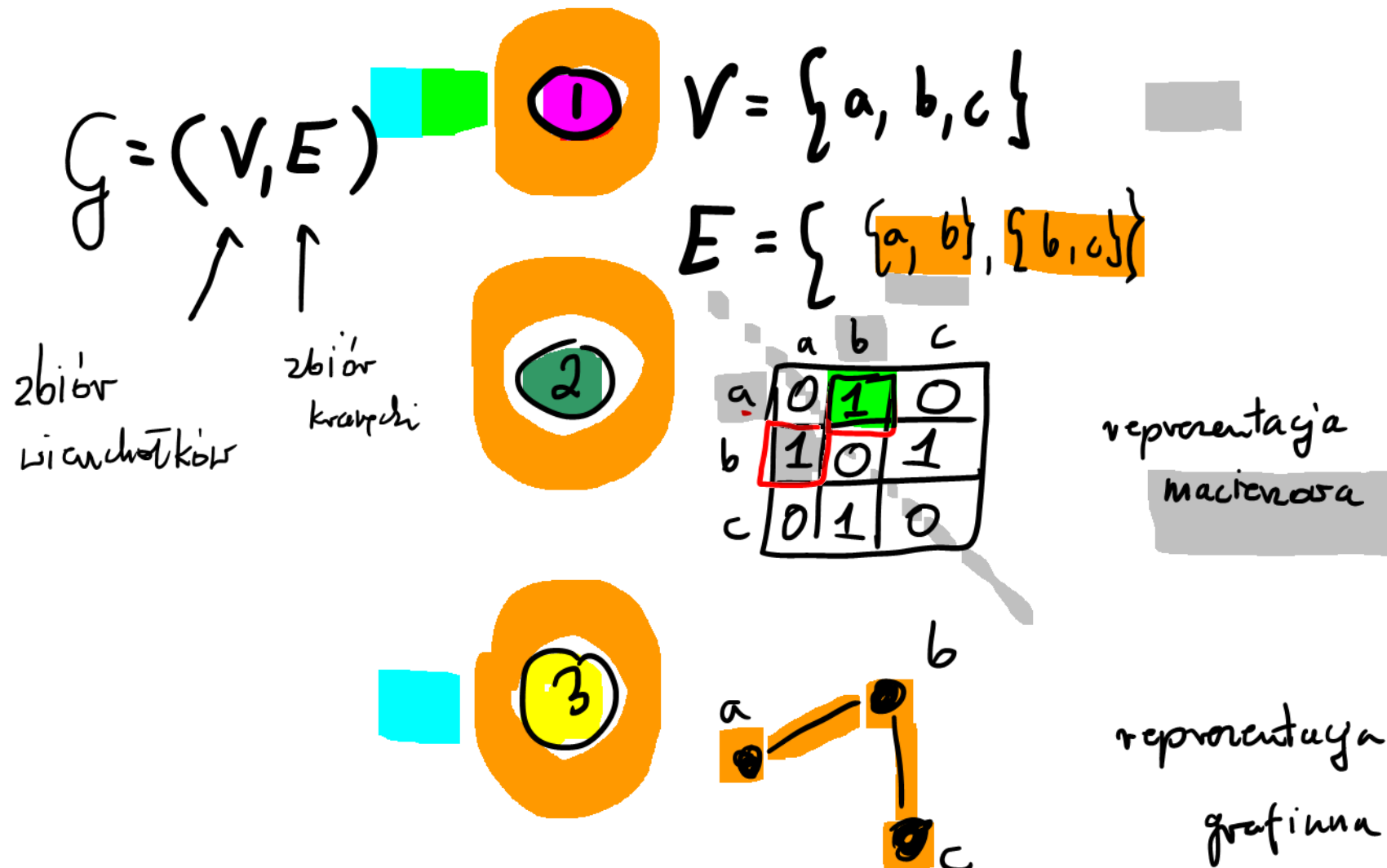
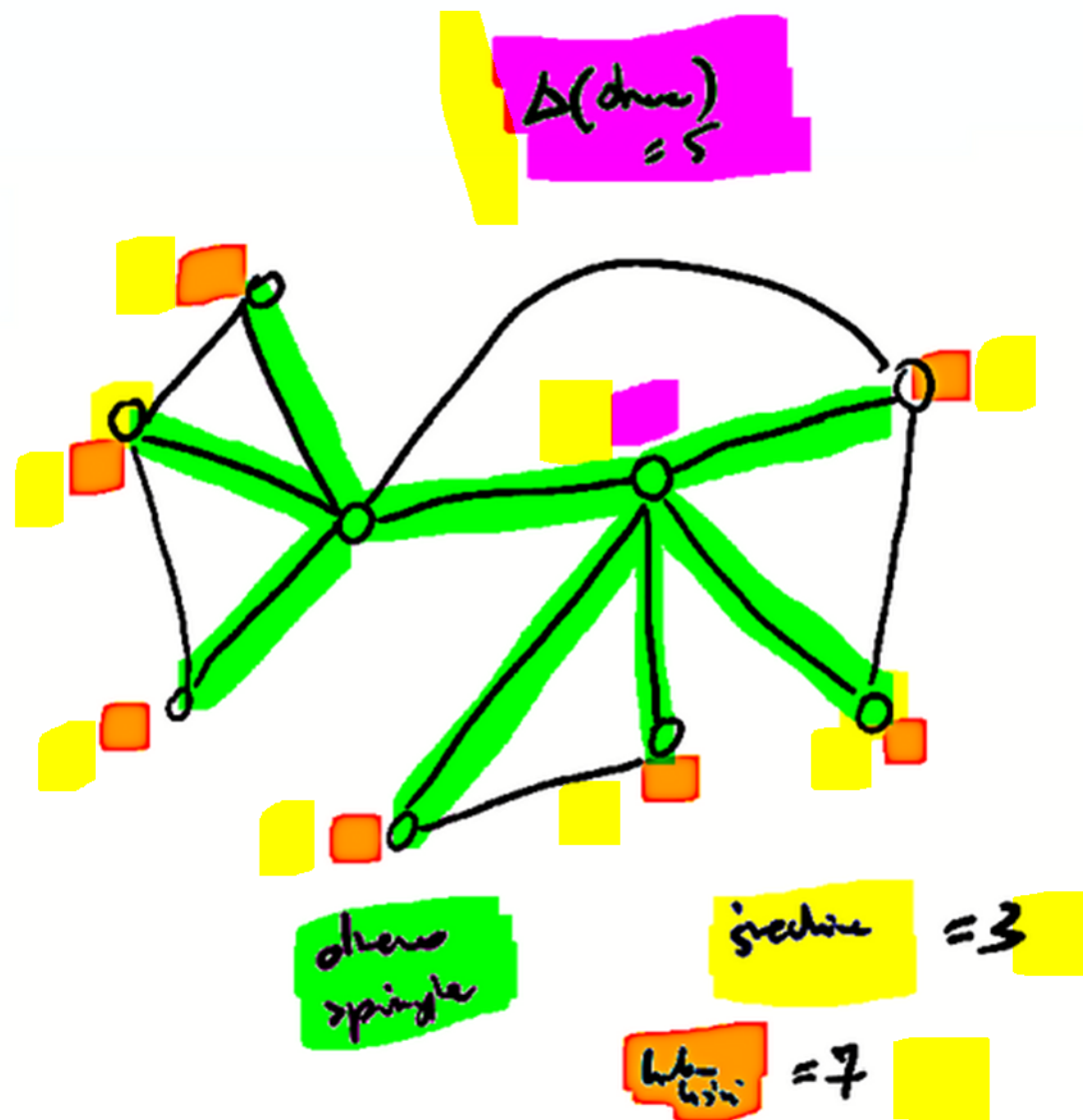
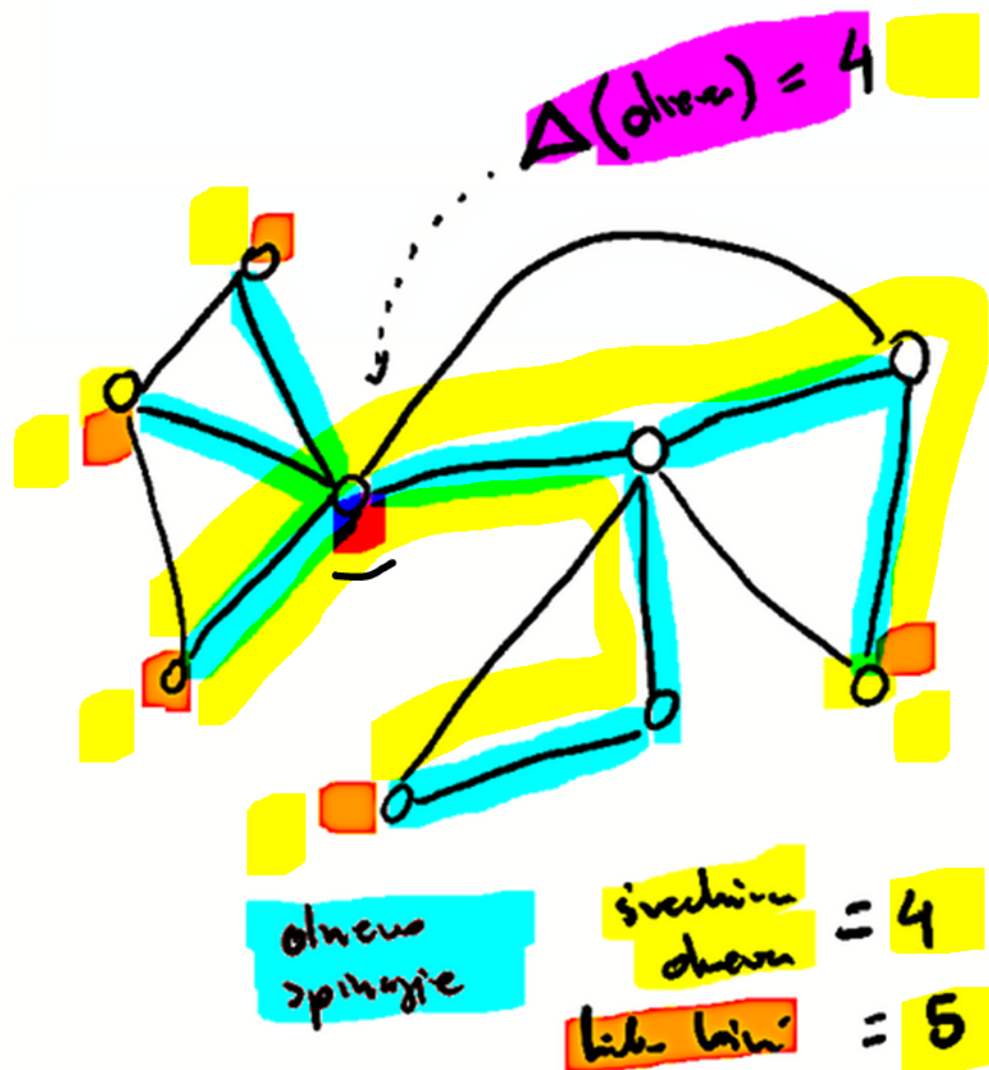


ALGORYTMICZNA TEORIA GRAFÓW



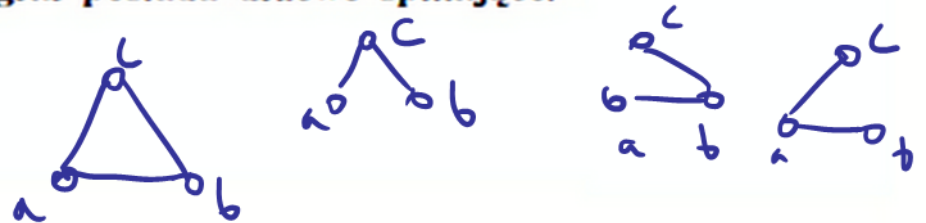
DRZEWA SPINAJĄCE

Drzewo spinające (rozpinające) multigrafu $G = (V, E)$ to dowolne drzewo $T = (V, E')$ takie, że $E' \subseteq E$. Zauważmy, że T ma taki sam zbiór wierzchołków co G , i każde drzewo spinające multigrafu G jest jego podgrafem. Można wykazać, że każdy spójny multigraf posiada drzewo spinające.



DRZEWA SPINAJĄCE

Drzewo spinające (rozpinające) multigrafu $G = (V, E)$ to dowolne drzewo $T = (V, E')$ takie, że $E' \subseteq E$. Zauważmy, że T ma taki sam zbiór wierzchołków co G , i każde drzewo spinające multigrafu G jest jego podgrafem. Można wykazać, że każdy spójny multigraf posiada drzewo spinające.



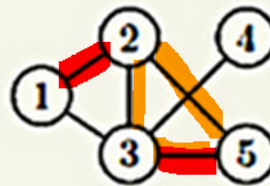
Algorytm konstrukcji drzewa spinającego.

Niech $G = (V, E)$ będzie spójnym (multi)grafem.

1. Dopóki (multi)graf nie jest drzewem, usuń dowolną krawędź dowolnego cyklu.

Zastosuj powyższy algorytm i wyznacz drzewo spinające poniższego grafu.

◀ PRZYKŁAD

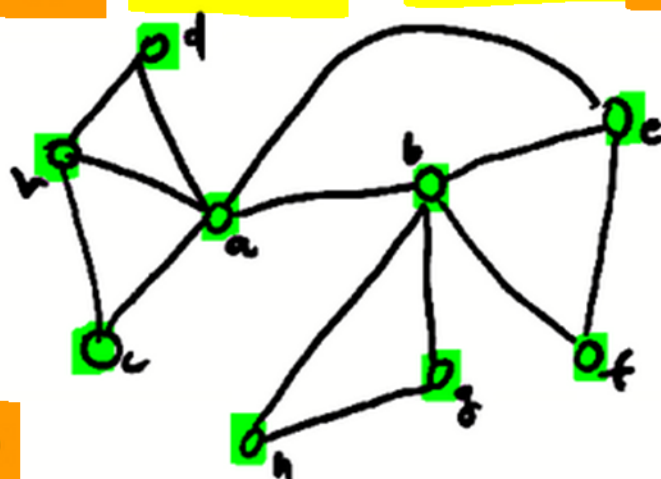


Zgodnie z algorytmem, wykonujemy:

- » Rozważamy cykl o wierzchołkach 1, 2, 5, 3 i usuwamy np. krawędź $\{1, 2\}$.
- » Rozważamy cykl o wierzchołkach 2, 3, 5 i usuwamy np. krawędź $\{3, 5\}$.
- » W otrzymanym grafie nie ma już cykli.

Otrzymujemy zatem następujące drzewo spinające $T = (V, E')$, gdzie $V = \{1, 2, 3, 4, 5\}$ oraz $E' = \{\{1, 3\}, \{2, 3\}, \{2, 5\}, \{3, 4\}\}$.

PRZESZUKIWANIE GRAFÓW W GŁĄB I WSZERZ — DRZEWA DFS I BFS



Algorytm przeszukiwania grafu w głąb

Niech $G = (V, E)$ będzie danym grafem spójnym, a $v \in V$ wierzchołkiem początkowym.

1. Odwiedzamy wierzchołek v (zaznaczamy go jako odwiedzony) i wkładamy go na STOS.

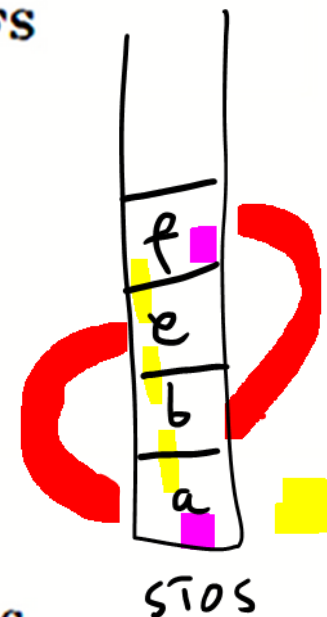
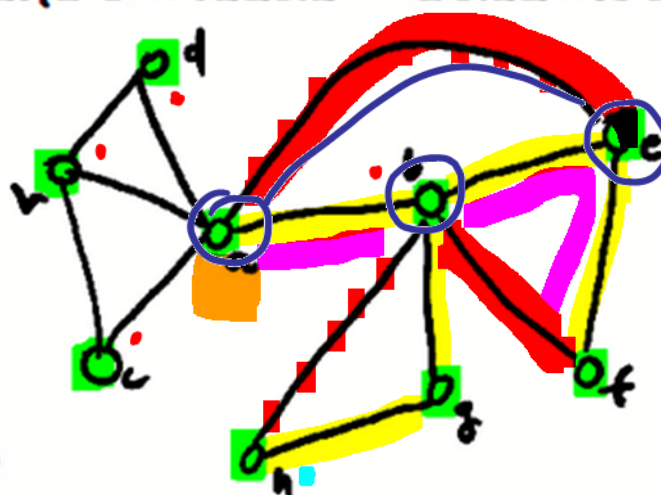
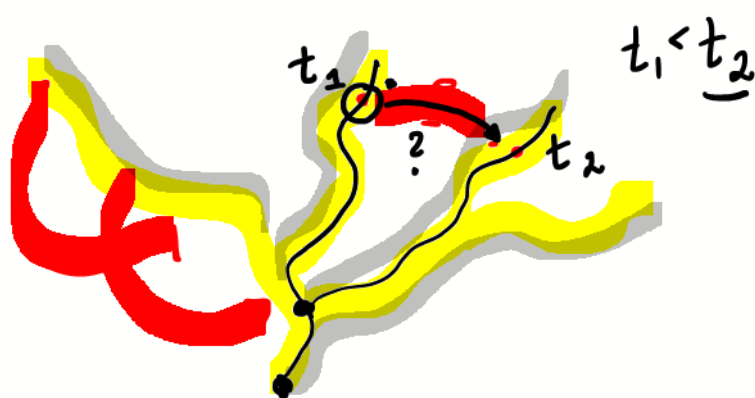
2. Dopóki STOS nie jest pusty, powtarzamy:

Jeżeli v jest wierzchołkiem na wierzchu STOSU, to sprawdzamy, czy istnieje wierzchołek sąsiedni z v , który nie był jeszcze odwiedzony.

- 2.1 Jeżeli u jest takim wierzchołkiem, to odwiedzamy u (zaznaczamy jako odwiedzony) i wkładamy go na STOS.

- 2.2 Jeżeli takiego u nie ma, to zdejmujemy v ze STOSU.

PRZESZUKIWANIE GRAFÓW W GŁĘB I WSZERZ — DRZEWA DFS I BFS



Algorytm przeszukiwania grafu w głąb

Niech $G = (V, E)$ będzie danym grafem spójnym, a $v \in V$ wierzchołkiem początkowym.

1. Odwiedzamy wierzchołek v (zaznaczamy go jako odwiedzony) i wkładamy go na STOS.

2. Dopóki STOS nie jest pusty, powtarzamy:

Jeżeli v jest wierzchołkiem na wierzchu STOSU, to sprawdzamy, czy istnieje wierzchołek sąsiedni z v , który nie był jeszcze odwiedzony.

2.1 Jeżeli u jest takim wierzchołkiem, to odwiedzamy u (zaznaczamy jako odwiedzony) i wkładamy go na STOS.

2.2 Jeżeli takiego u nie ma, to zdejmujemy v ze STOSU.

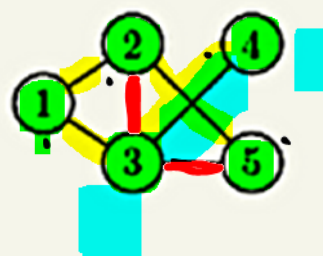
Uwaga 1. Jeśli jest kilka wierzchołków do wyboru, to wybieramy zgodnie z ustalonym porządkiem.

Uwaga 2. Wierzchołki na STOSIE w dowolnym kroku tworzą ścieżkę od korzenia do wierzchołka aktualnie odwiedzanego.

Uwaga 3. Jeśli w powyższej procedurze w kroku 2.1, w którym odwiedzamy wierzchołek u , do początkowo pustego zbioru E' krawędzi dodawać będziemy krawędź $\{v, u\}$, to otrzymamy drzewo spinające **DFS** (ang. *depth-first search*).

Przeszukaj poniższy graf $G = (V, E)$ w głąb poczynając od wierzchołka o etykiecie 3 i skonstruuj odpowiednie drzewo spinające DFS.

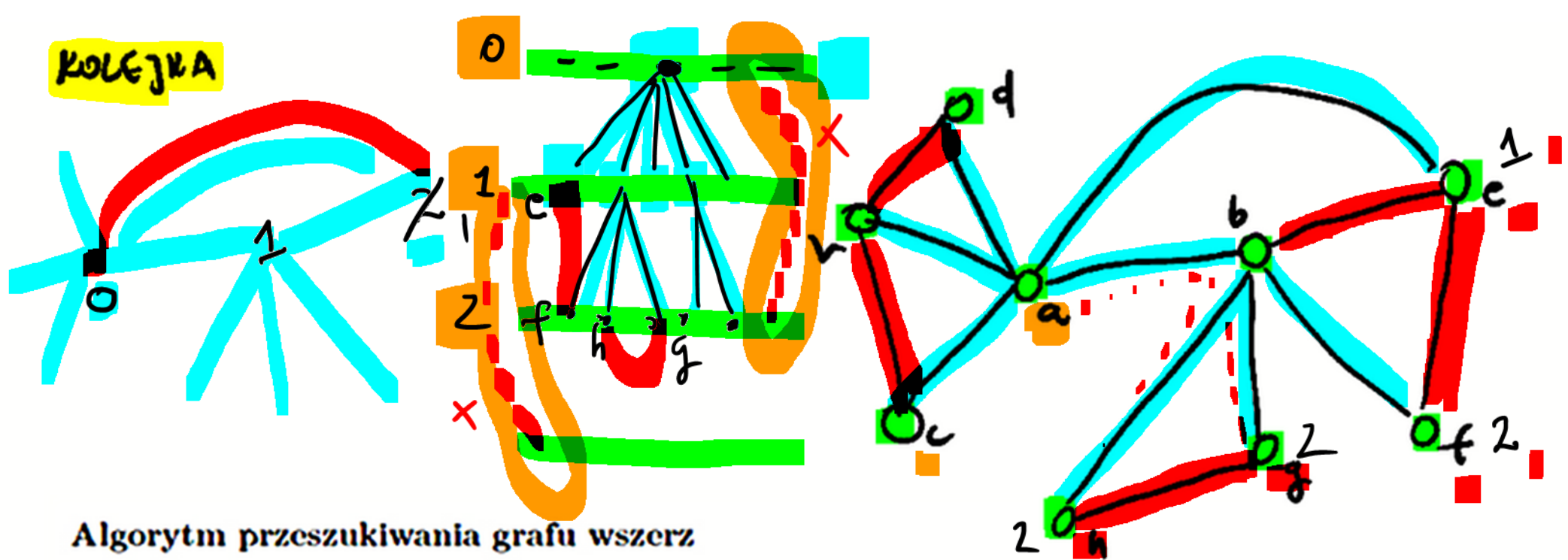
◀ PRZYKŁAD



Przebieg algorytmu jest następujący.

rozpatrywany wierzchołek	odwiedzany wierzchołek	STOS	zbiór krawędzi drzewa DFS
–	3	3	∅
3	1	3, 1	{(1, 3)}
1	2	3, 1, 2	{(1, 3), (1, 2)}
2	5	3, 1, 2, 5	{(1, 3), (1, 2), (2, 5)}
5	–	3, 1, 2	{(1, 3), (1, 2), (2, 5)}
2	–	3, 1	{(1, 3), (1, 2), (2, 5)}
1	–	3	{(1, 3), (1, 2), (2, 5)}
3	4	3, 4	{(1, 3), (1, 2), (2, 5), (3, 4)}
4	–	3	{(1, 3), (1, 2), (2, 5), (3, 4)}
3	–	∅	{(1, 3), (1, 2), (2, 5), (3, 4)}

Zatem wierzchołki były odwiedzane w kolejności 3, 1, 2, 5, 4 i otrzymaliśmy drzewo spinające DFS $T = (V, E')$, gdzie $V = \{1, 2, 3, 4, 5\}$ oraz $E' = \{(1, 3), (1, 2), (2, 5), (1, 4)\}$.



Algorytm przeszukiwania grafu wszerz

Niech $G = (V, E)$ będzie danym grafem spójnym, a $v \in V$ wierzchołkiem początkowym.

1. Odwiedzamy wierzchołek v (zaznaczamy go jako odwiedzony) i wstawiamy go do **KOLEJKI**.
2. Dopóki KOLEJKA nie jest pusta, powtarzamy:
 - 2.1 Bierzemy wierzchołek v z początku KOLEJKI.
 - 2.2 Odwiedzamy wszystkie do tej pory jeszcze nie odwiedzone wierzchołki sąsiednie z v (zaznaczamy je jako odwiedzone) i wstawiamy je na koniec KOLEJKI.

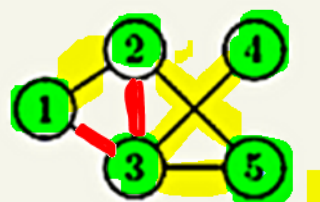
Uwaga 1. Wierzchołki wstawiamy do KOLEJKI np. w kolejności uporządkowania etykiet.

Uwaga 2. Wierzchołki przeszukiwane są w kolejności leżących najbliżej korzenia.

Uwaga 3. Jeśli w powyższej procedurze w kroku 2.2, w którym odwiedzamy wszystkie nieodwiedzone jeszcze wierzchołki sąsiednie do v , do początkowo pustego zbioru E' krawędzi dodawać będziemy odpowiednie krawędzie $\{v, u\}$, to otrzymamy drzewo spinające **BFS** (ang. *breath-first search*).

Przeszukaj poniższy graf $G = (V, E)$ **wszerz** poczynając od wierzchołka o etykiecie **5** i skonstruuj odpowiednie drzewo spinające BFS.

◀ PRZYKŁAD



Przebieg algorytmu jest następujący.

rozpatrywany wierzchołek	odwiedzane wierzchołki	KOLEJKA	zbiór krawędzi drzewa BFS
—	5	5	$\emptyset$
5	2,3	2,3	$\{\{2,5\}, \{3,5\}\}$
2	1	3,1	$\{\{2,5\}, \{3,5\}, \{1,2\}\}$
3	4	1,4	$\{\{2,5\}, \{3,5\}, \{1,2\}, \{3,4\}\}$
1	—	4	$\{\{2,5\}, \{3,5\}, \{1,2\}, \{3,4\}\}$
4	—	$\emptyset$	$\{\{2,5\}, \{3,5\}, \{1,2\}, \{3,4\}\}$

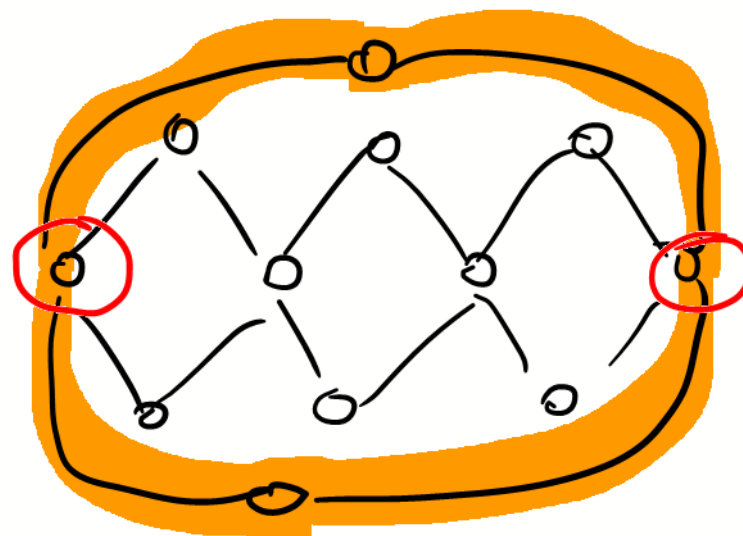
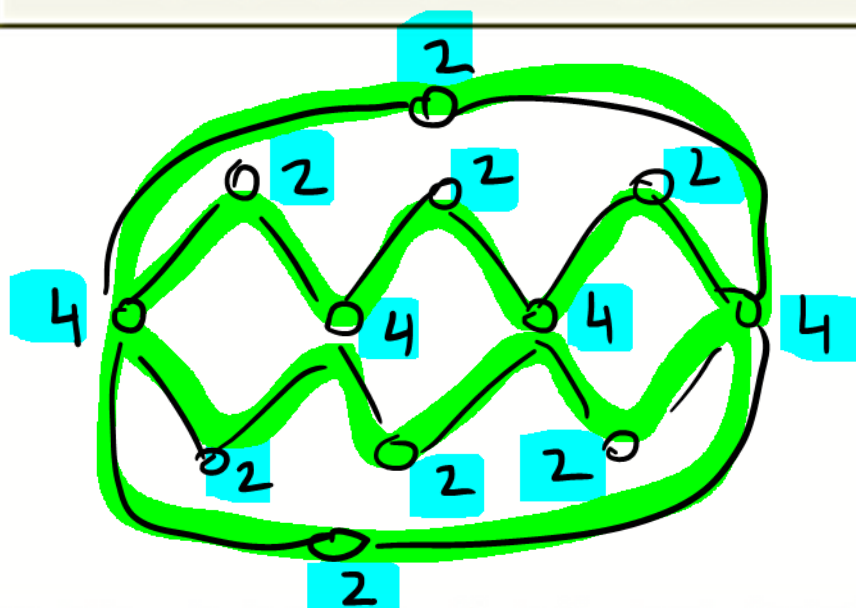
Zatem wierzchołki były odwiedzane w kolejności 5, 2, 3, 1, 4 i otrzymaliśmy drzewo spinające BFS $T = (V, E')$, gdzie $V = \{1, 2, 3, 4, 5\}$ oraz $E' = \{\{2, 5\}, \{3, 5\}, \{1, 2\}, \{3, 4\}\}$.

GRAFY EULEROWSKIE I HAMILTONOWSKIE

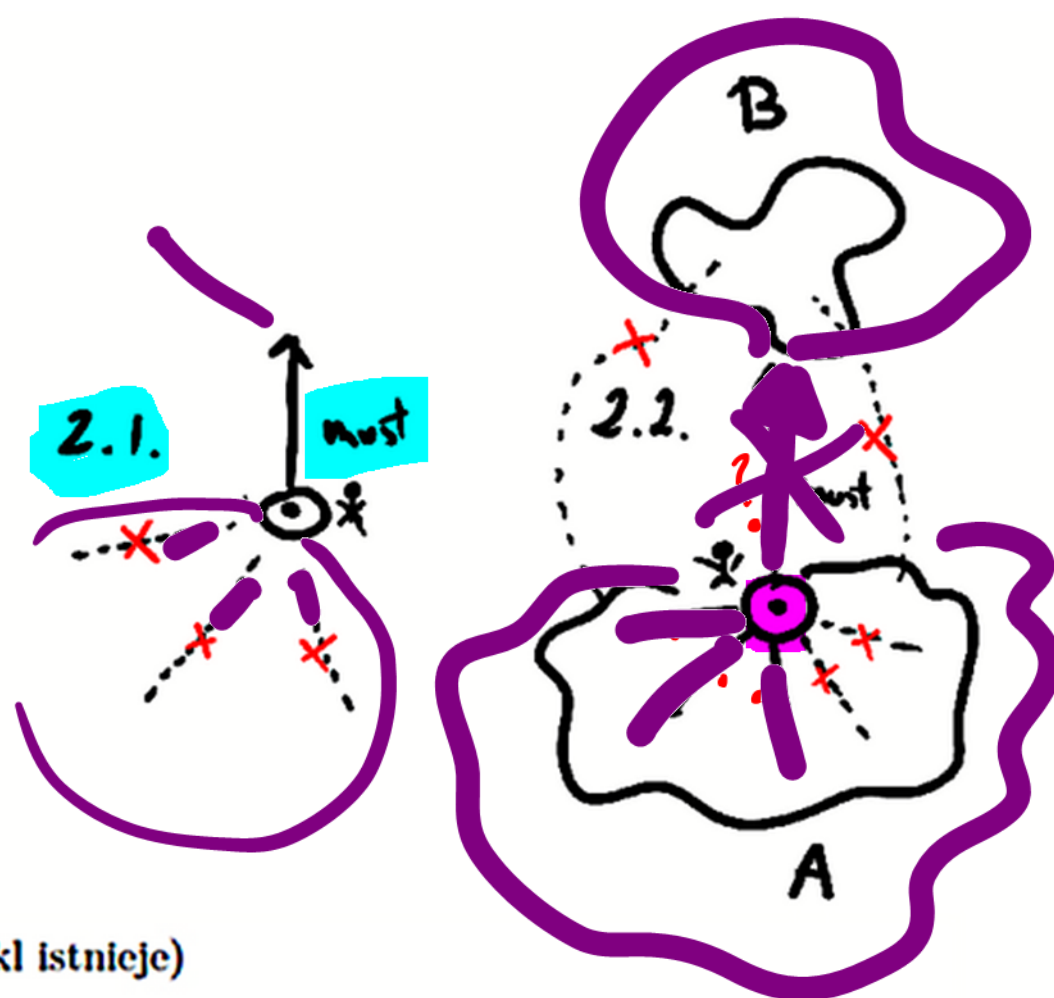
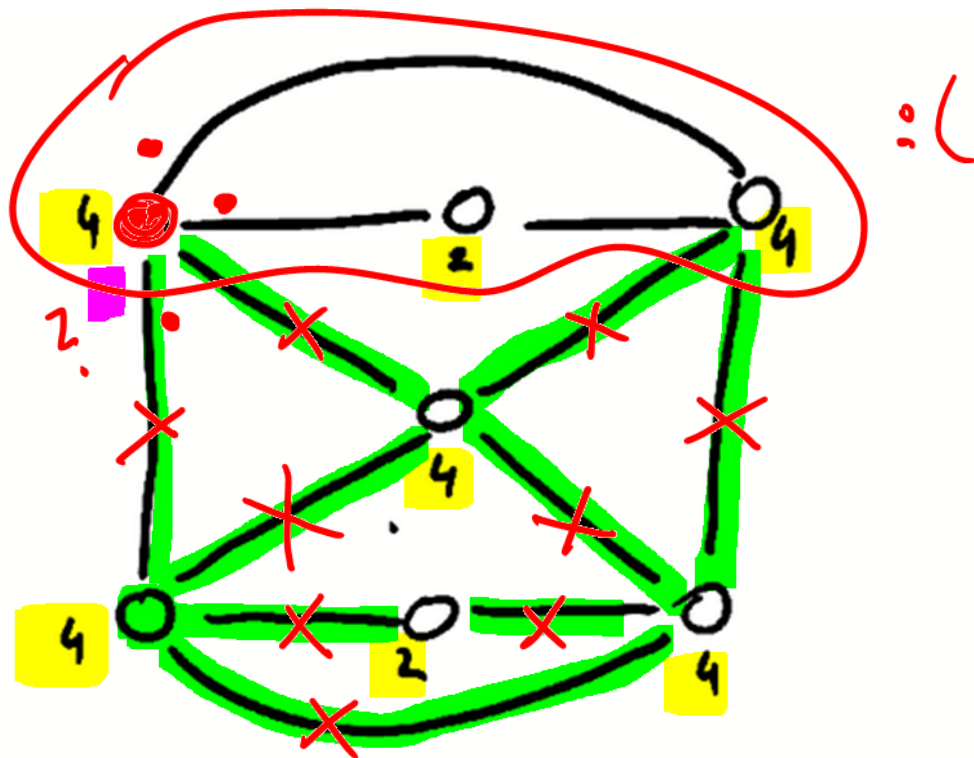
Niech dany będzie spójny multigraf $G = (V, E)$. Mówimy, że G jest *eulerowski*, jeśli istnieje łańcuch zamknięty zawierający każdą krawędź multigrafu; taki łańcuch nazywamy *cyklem Eulera*. Analogicznie, mówimy, że G jest *póleulerowski*, jeśli istnieje łańcuch zawierający każdą krawędź grafu; taki łańcuch nazywamy łańcuchem Eulera.

TWIERDZENIE 5.3

- a) Spójny multigraf $G = (V, E)$ jest *eulerowski* wtedy i tylko wtedy, gdy każdy jego wierzchołek jest *parzystego stopnia*.
- b) Spójny multigraf G jest *póleulerowski* wtedy i tylko wtedy, gdy posiada co najwyżej dwa wierzchołki nieparzystego stopnia, z czego jeden z nich jest początkiem łańcucha Eulera, a drugi jego końcem.



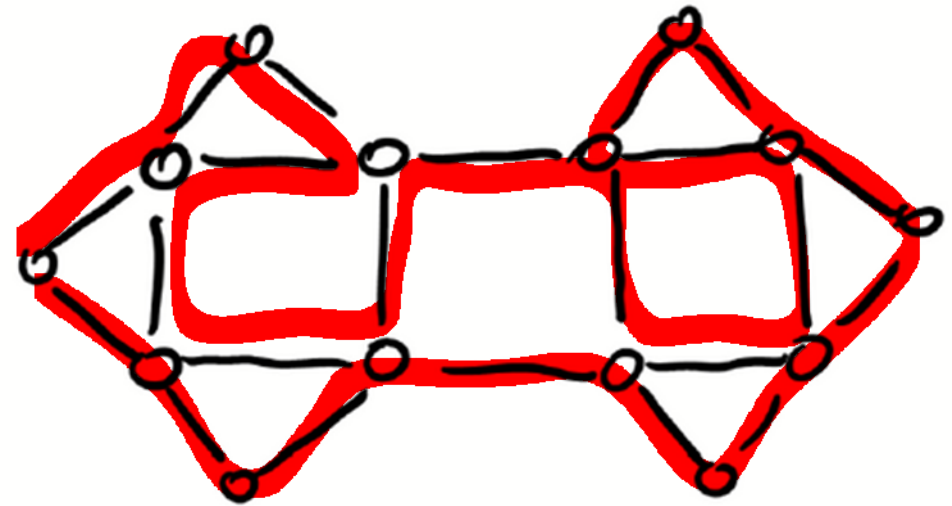
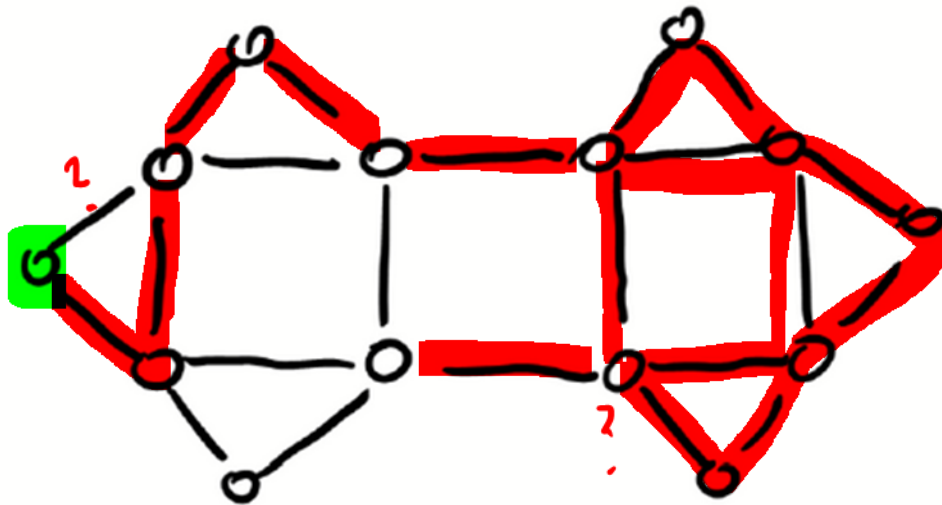
Niech dany będzie spójny (multi)graf $G = (V, E)$. Mówimy, że G jest *hamiltonowski*, jeśli istnieje cykl, który przechodzi przez każdy wierzchołek dokładnie raz; taki cykl nazywamy *cyklem Hamiltona*. Analogicznie, mówimy, że G jest *półhamiltonowski*, jeśli zawiera ścieżkę przechodzącą przez każdy wierzchołek dokładnie raz; taką ścieżkę nazywamy *ścieżką Hamiltona*.



Algorytm znajdowania **cyklu Eulera** (o ile taki cykl istnieje)

Niech $G = (V, E)$ będzie spójnym multigrafem o wszystkich wierzchołkach parzystego stopnia.

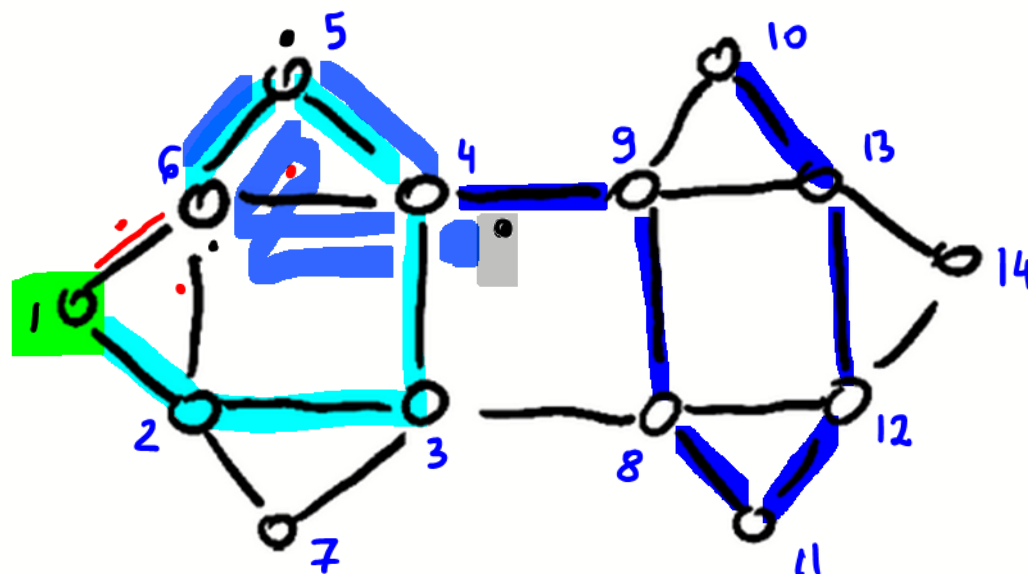
1. Zaczynamy od dowolnego wierzchołka $v \in V$.
2. Powtarzamy, aż przejdziemy wszystkie krawędzie:
 - 2.1 Jeżeli z bieżącego wierzchołka x odchodzi tylko jedna krawędź, to przechodzimy wzdłuż tej krawędzi do następnego wierzchołka i usuwamy tę krawędź wraz z wierzchołkiem x .
 - 2.2 W przeciwnym wypadku, jeżeli z x odchodzi więcej krawędzi, to wybieramy tę krawędź, której usunięcie nie rozpójnia nam grafu, i przechodzimy wzdłuż tej krawędzi do następnego wierzchołka, a następnie usuwamy tę krawędź z grafu.



Algorytm znajdowania cyklu Eulera (o ile taki cykl istnieje)

Niech $G = (V, E)$ będzie spójnym multigrafem o wszystkich wierzchołkach parzystego stopnia.

1. Zaczynamy od dowolnego wierzchołka $v \in V$.
2. Powtarzamy, aż przejdziemy wszystkie krawędzie:
 - 2.1 Jeżeli z bieżącego wierzchołka x odchodzi tylko jedna krawędź, to przechodzimy wzdłuż tej krawędzi do następnego wierzchołka i usuwamy tę krawędź wraz z wierzchołkiem x .
 - 2.2 W przeciwnym wypadku, jeżeli z x odchodzi więcej krawędzi, to wybieramy tę krawędź, której usunięcie nie rozspójnia nam grafu, i przechodzimy wzdłuż tej krawędzi do następnego wierzchołka, a następnie usuwamy tę krawędź z grafu.



~~1 2 3 4 5 6~~ "x"
 1 2 3 4 5 "x"
 1 2 3 4 6 5
 1 2 3 4 6 -5
 1 2 3 4 5 (6)
 1 2 3 4 9 >6

Algorytm z nawrotami znajdowania drogi Hamiltona (o ile taka droga istnieje)

Niech $G = (V, E)$ będzie spójnym grafem i pewnym wyróżnionym wierzchołkiem $v \in V$.

1. Wkładamy v na STOS.

2. Powtarzamy:

2.1 Jeżeli u jest wierzchołkiem na wierzchu stosu, to szukamy wierzchołka w o najniższym możliwym numerze (najwcześniejszego przy ustalonym porządku wierzchołków grafu) sąsiedniego z u i nie występującego na STOSIE, jednakże przy założeniu, że wierzchołek w jest „większy” od wierzchołka zdjętego krok wcześniej ze STOSU (o ile był taki).

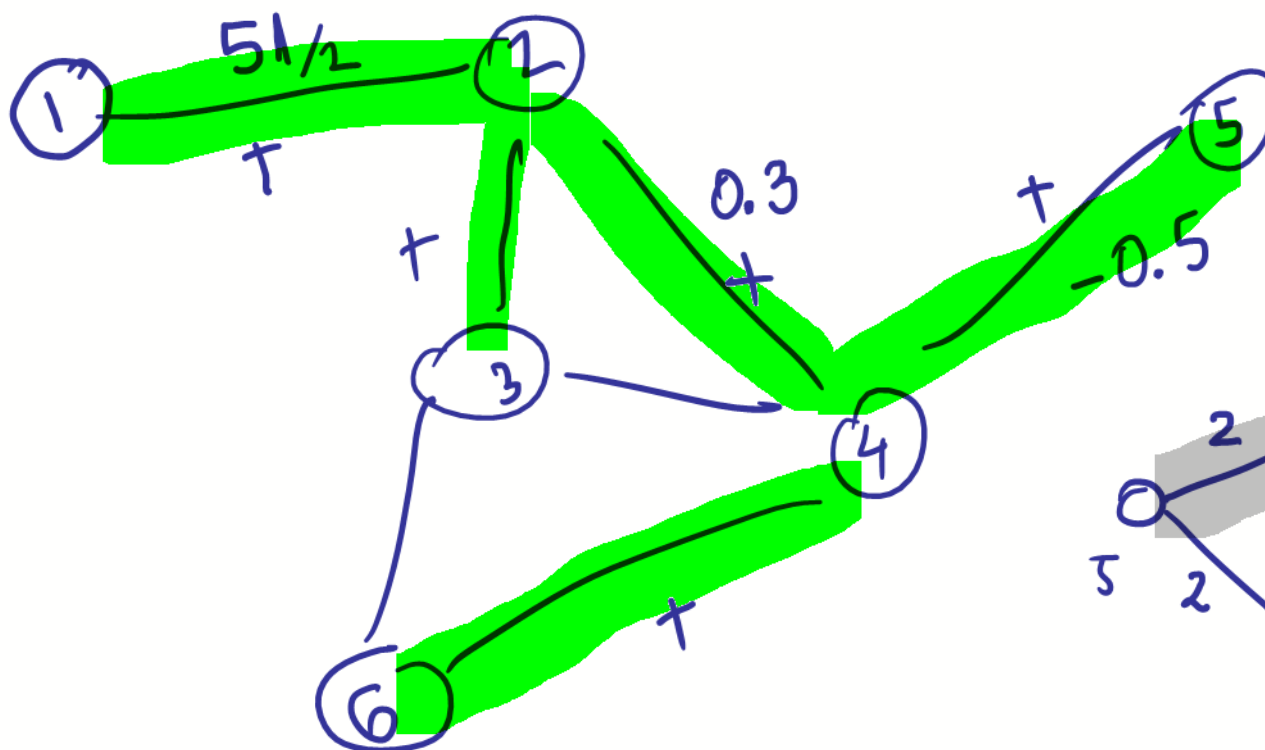
2.2 Jeśli takie w znajdziemy, to wkładamy je na stos — jeżeli dotychczasowy STOS tworzy drogę Hamiltona, to KONIEC.

2.3 Jeżeli takiego w nie znaleźliśmy, to zdejmujemy u ze stosu.

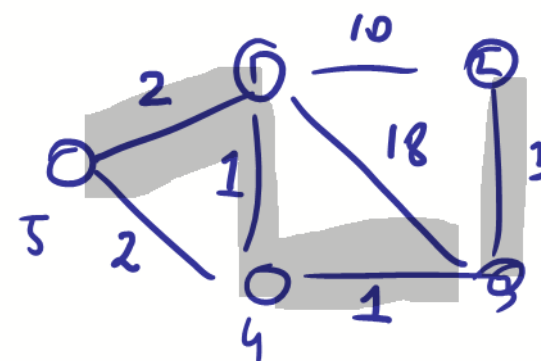
Niech $G = (V, E, w)$ będzie *grafem ważonym*, tzn. każdej krawędzi $e \in E$ przyporządkowana jest pewna waga $w(e)$. Problem *Minimalnego Drzewa Spinającego* [MDS] definiujemy jako znalezienie drzewa spinającego $T = (V, E')$ w grafie G o minimalnej sumie ważonej

$$\sum_{e \in E'} w(e).$$

Minimalne drzewo spinające znajduje zastosowanie np. przy wyznaczeniu „najtańszej” sieci dróg, torów kolejowych, itp., która łączy danych n miast.



$$k_{\text{ort}} = 7$$



Algorytm konstrukcji minimalnego drzewa spinającego (algorytm Kruskala, 1956)

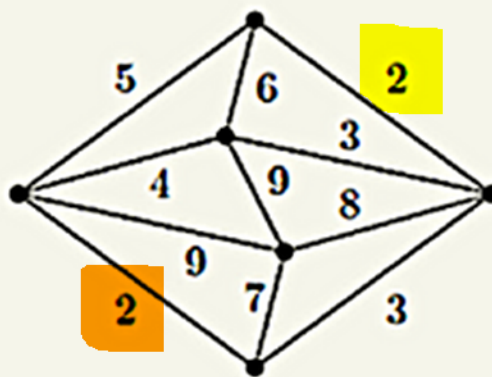
Niech $G = (V, E, w)$ będzie spójnym grafem ważonym z funkcją wagi $w: E \rightarrow R$.

1. $T := (V, E')$, gdzie $E' := \emptyset$.
2. Posortuj krawędzie grafu G w kolejności niemalejących wag.
3. Dla każdej krawędzi $e \in E$:
jeśli dodanie rozważanej krawędzi e nie utworzy cyklu w T , wówczas $E' := E' \cup \{e\}$.

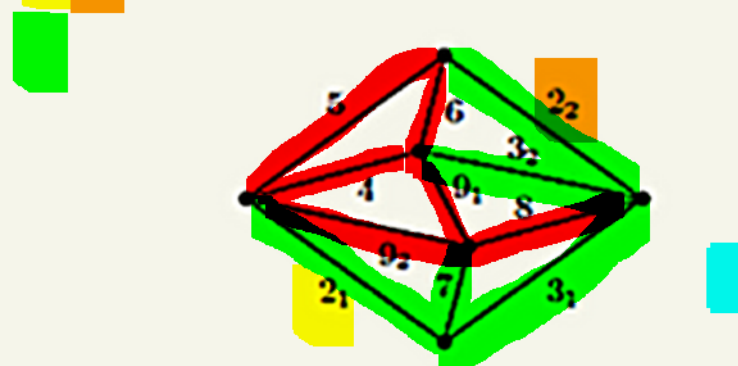
2, 2, 3, ...

Znajdź minimalne drzewo spinające dla podanego niżej grafu.

◀ PRZYKŁAD



Posortowany ciąg krawędzi wygląda następująco: 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 9. Jako że niektóre wagi krawędzi powtarzają się, należy je rozróżnić np. dodając odpowiedni indeks dolny — otrzymujemy ciąg 2₁, 2₂, 3₁, 3₂, 4, 5, 6, 7, 8, 9₁, 9₂.

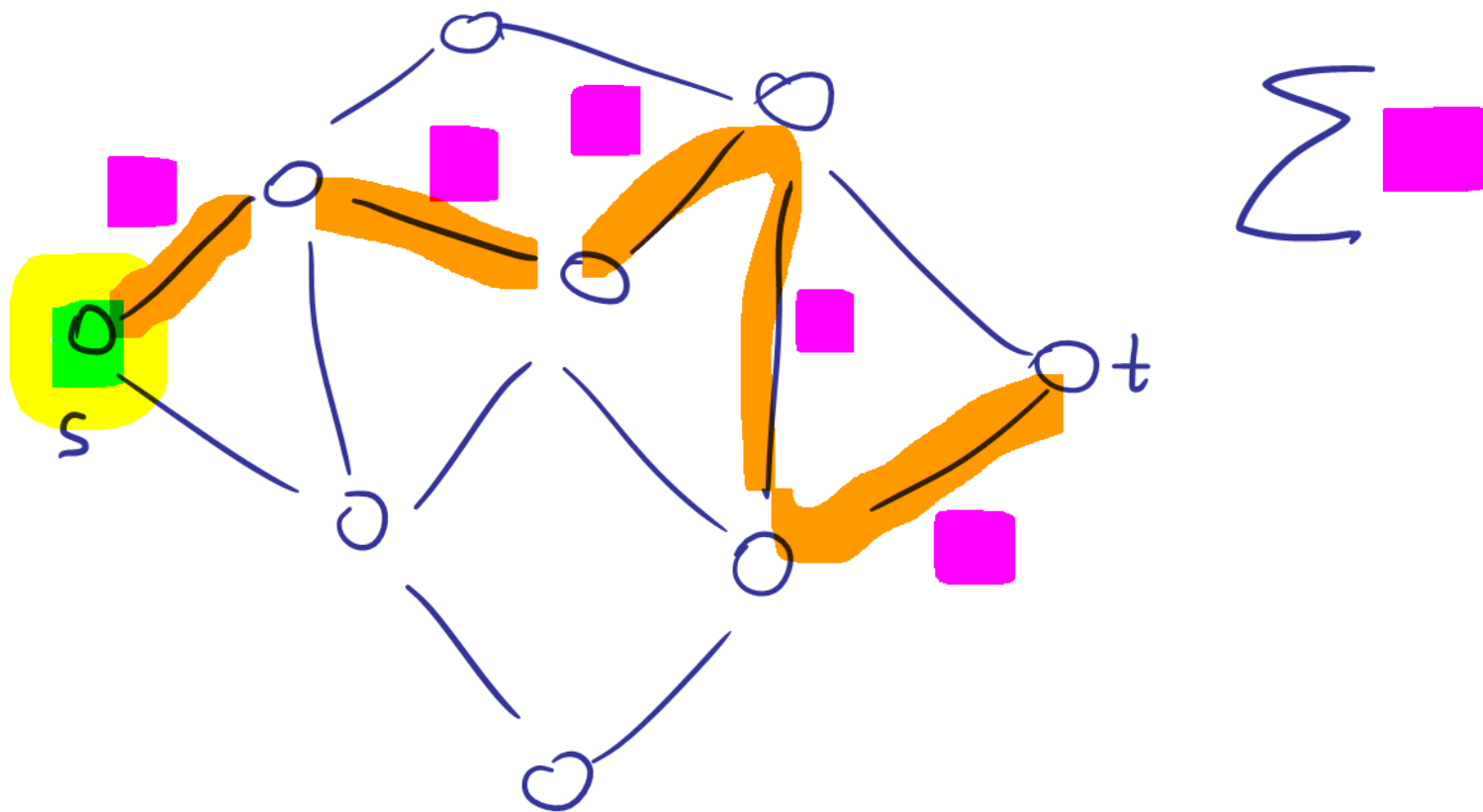


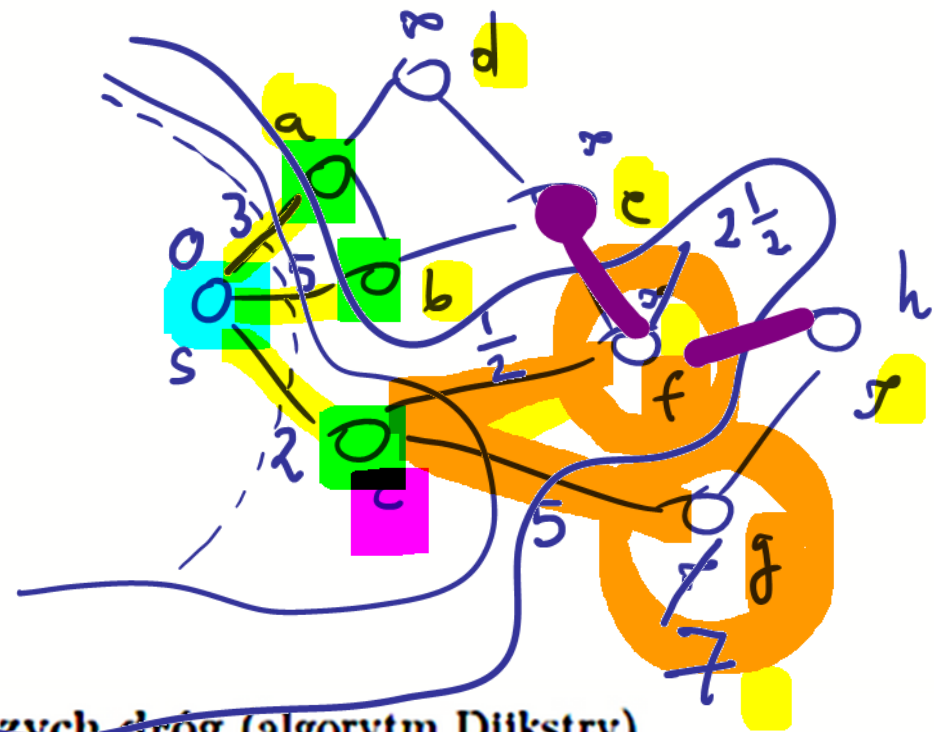
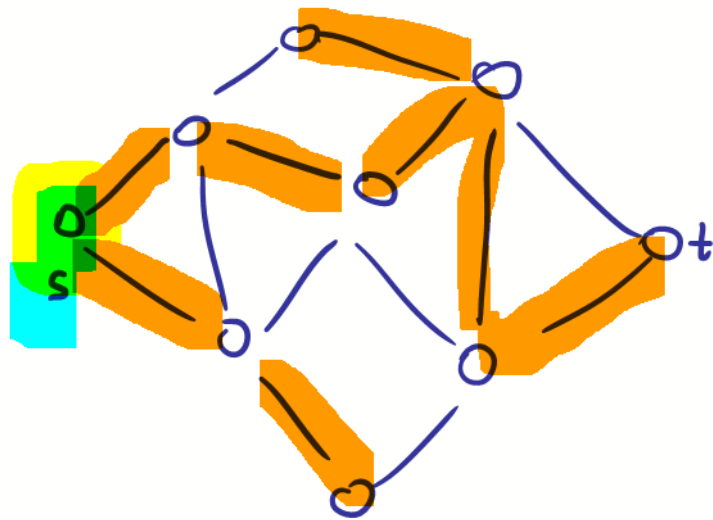
Przebieg algorytmu jest następujący (dla ułatwienia ilustracji działania algorytmu utożsamiamy wagi krawędzi z samymi krawędziami).

rozpatrywana krawędź	cykl?	krawędzie drzewa
2 ₁	–	2 ₁
2 ₂	–	2 ₁ , 2 ₂
3 ₁	–	2 ₁ , 2 ₂ , 3 ₁
3 ₂	–	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂
4	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂
5	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂
6	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂
7	–	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂ , 7
8	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂ , 7
9 ₁	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂ , 7
9 ₂	+	2 ₁ , 2 ₂ , 3 ₁ , 3 ₂ , 7

Zauważmy, że skoro graf ma 6 wierzchołków, a z definicji drzewo spinające ma 5 krawędzi, wykonywanie algorytmu można było już przerwać, gdy dodaliśmy płątkę krawędzi o wadze 7.

Rozważmy graf ważony $G = (V, E, w)$ z dodatnią funkcją kosztu, tj. $w: E \rightarrow \mathbb{R}^+$. Dla prostoty zakładamy, że jeśli $e \notin E$, to $w(e) = \infty$. Dla każdej drogi $v_0 v_1 \dots v_k$ w grafie zdefiniujemy jej *długość* jako sumę długości krawędzi, czyli $\sum_{i=1}^k (w(\{v_{i-1}, v_i\}))$. Jeżeli $k = 0$, wówczas droga składa się z pojedynczego wierzchołka i przyjmujemy wtedy, że jej długość wynosi 0.





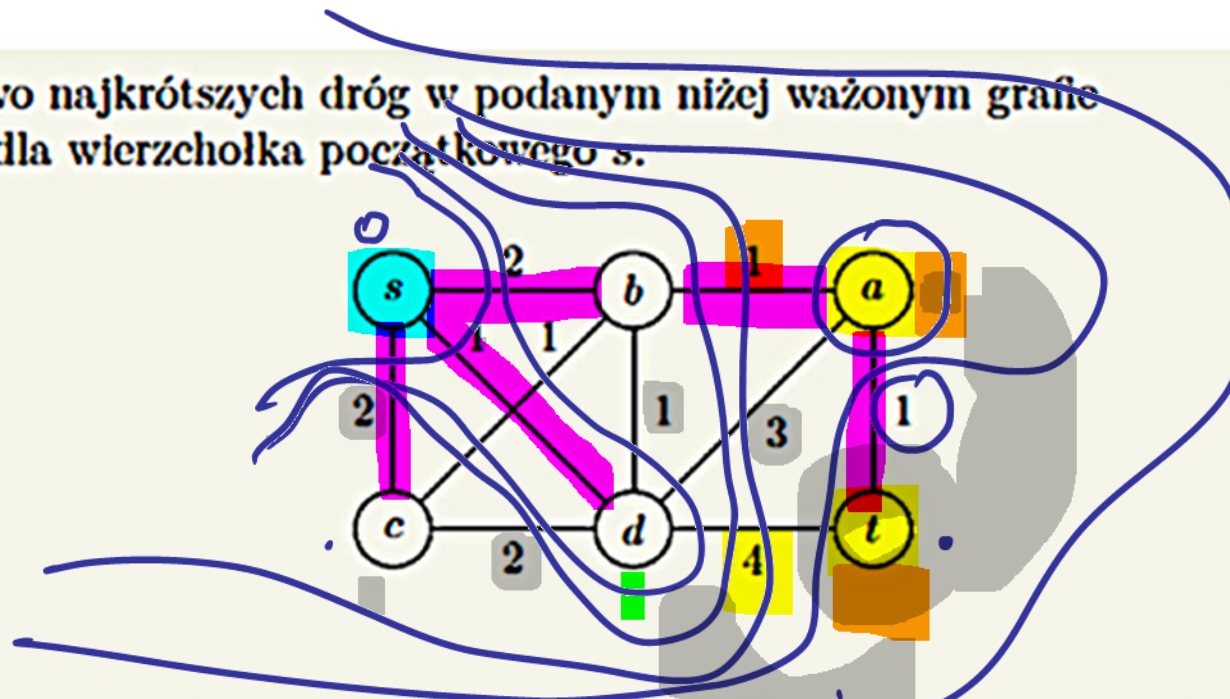
Algorytm wyznaczania długości najkrótszych dróg (algorytm Dijkstry)

Niech $s \in V$ będzie ustalonym wierzchołkiem ważonego grafu $G = (V, E, w)$ o dodatniej funkcji kosztu. Algorytm na wyjściu zwraca macierz D , gdzie dla wierzchołka $v \in V$ wartość $D[v]$ jest długością najkrótszej ścieżki z s do v .

1. $D[s] := 0$.
2. $\bar{V} := V \setminus \{s\}$.
3. Dla każdego $v \in \bar{V}$ podstaw $D[v] := w(\{s, v\})$.
4. Dopóki $\bar{V} \neq \emptyset$, wykonuj:
 - 4.1 Wybierz wierzchołek $u \in \bar{V}$ taki, że $D[u] = \min_{x \in \bar{V}} D[x]$.
 - 4.2 $\bar{V} := \bar{V} \setminus \{u\}$.
 - 4.3 Dla każdego $v \in \bar{V}$ podstaw $D[v] := \min(D[v], D[u] + w(\{u, v\}))$.

Wyznacz drzewo najkrótszych dróg w podanym niżej ważonym grafie $G = (V, E, w)$ dla wierzchołka początkowego s .

◀ PRZYKŁAD



Poniższa tabela ilustruje jak w kolejnych iteracjach zewnętrznej pętli algorytmu Dijkstry wybierany jest wierzchołek u oraz jak przedstawia się zbiór $\bar{V}$ oraz macierz D .

Iteracja	u	V	$D[s]$	$D[a]$	$D[b]$	$D[c]$	$D[d]$	$D[t]$
0		$\{a, b, c, d, t\}$	0	∞	2	2	1	∞
1	d	$\{a, b, c, t\}$		4	2	2		5
2	b	$\{a, c, t\}$		3		2		5
3	c	$\{a, t\}$		3				5
4	a	$\{t\}$						4
5	t	$\emptyset$						

Wyznacz drzewo najkrótszych dróg w podanym niżej ważonym grafie $G = (V, E, w)$ dla wierzchołka początkowego s .

◀ PRZYKŁAD

Zauważmy, że opisany wyżej algorytm Dijkstry wyznacza tylko macierz najkrótszych odległości, nie zapamiętując w czasie wykonywania żadnych dodatkowych informacji. Aby wyznaczyć najkrótszą drogę z wierzchołka s do wybranego wierzchołka v można albo zmodyfikować algorytm tak, aby za każdym razem, kiedy usuwamy wierzchołek u ze zbioru $\bar{V}$, dodawał on odpowiednią krawędź do konstruowanego drzewa najkrótszych dróg, albo też skorzystać bezpośrednio z wyznaczonej macierzy D .

Poniższa tabela ilustruje jak w kolejnych iteracjach zewnętrznej pętli algorytmu Dijkstry wybierany jest wierzchołek u oraz jak przedstawia się zbiór $\bar{V}$ oraz macierz D .

Iteracja	u	V	$D[s]$	$D[a]$	$D[b]$	$D[c]$	$D[d]$	$D[t]$
0		$\{a, b, c, d, t\}$	0	∞	2	2	<u>1</u>	∞
1	d	$\{a, b, c, t\}$	0	4	<u>2</u>	2	1	5
2	b	$\{a, c, t\}$	0	3	2	<u>2</u>	1	5
3	c	$\{a, t\}$	0	<u>3</u>	2	2	1	5
4	a	$\{t\}$	0	3	2	2	1	<u>4</u>
5	t	$\emptyset$	0	3	2	2	1	4

Najkrótszą drogę wyznaczamy od końca — najpierw szukamy przedostatniego wierzchołka tej drogi, potem trzeciego od końca i tak dalej.

- Przedostatni wierzchołek x najkrótszej drogi spełnia równość $D[t] = D[x] + w(\{x, t\})$. W naszym przykładzie (tylko) wierzchołek $x = a$ spełnia tę równość:

$$4 = D[t] = D[a] + w(\{a, t\}) = 3 + 1.$$

A zatem przedostatnim wierzchołkiem jest wierzchołek a .

- Trzeci wierzchołek y od końca najkrótszej drogi z s do t — a przedostatni wierzchołek najkrótszej drogi z s do a — spełnia równość $D[a] = D[y] + w(\{y, a\})$. W naszym przykładzie (tylko) wierzchołek $y = b$ spełnia tę równość:

$$3 = D[a] = D[b] + w(\{b, a\}) = 2 + 1.$$

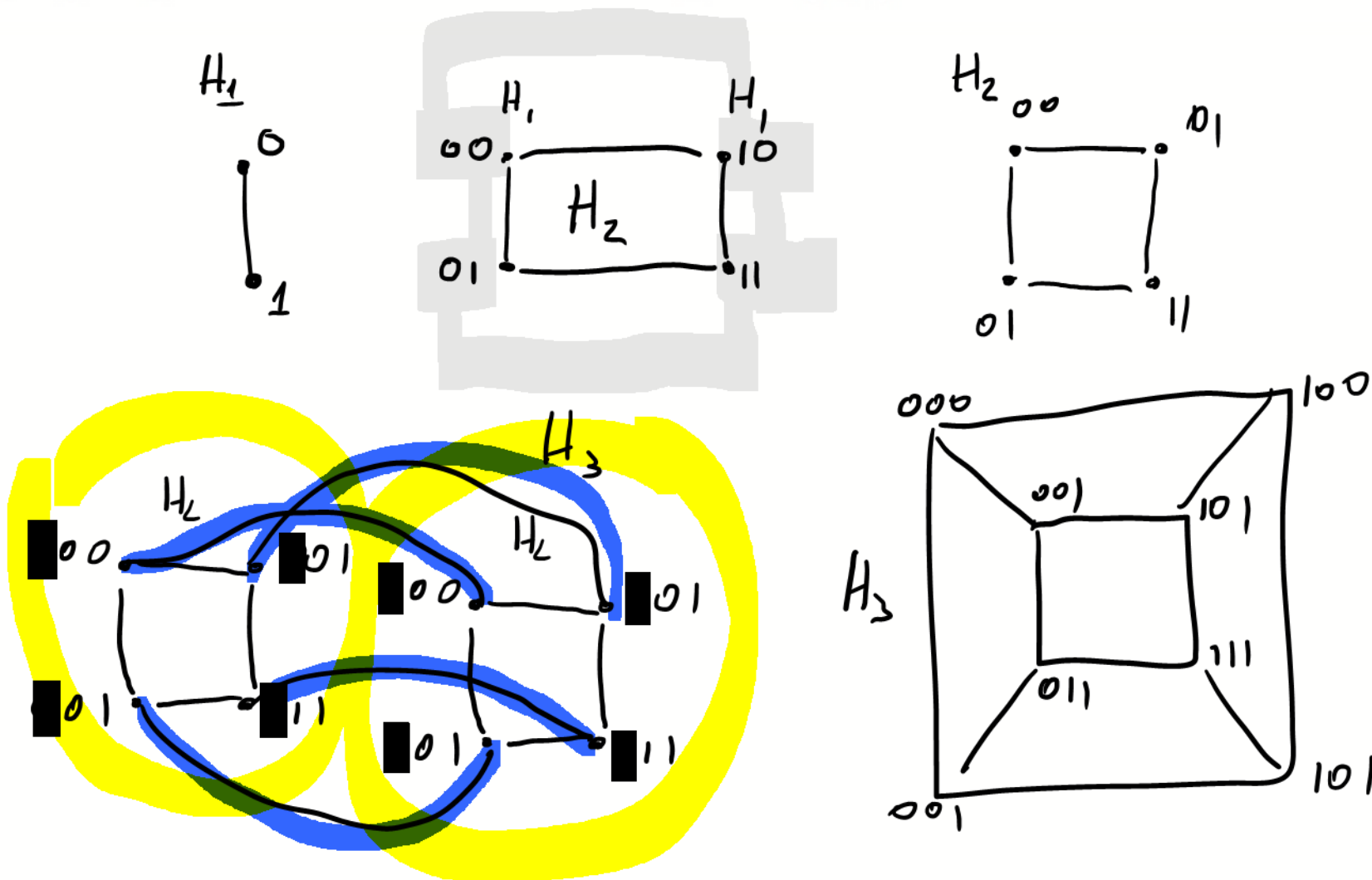
A zatem pozostaje nam znaleźć najkrótszą drogę z s do b .

- Czwarty wierzchołek z od końca najkrótszej drogi z s do t — a przedostatni wierzchołek najkrótszej drogi z s do b — spełnia równość $D[b] = D[z] + w(\{z, b\})$. W naszym przykładzie (tylko) wierzchołek $y = s$ spełnia tę równość:

$$2 = D[b] = D[s] + w(\{s, b\}) = 0 + 2.$$

W konsekwencji najkrótsza droga z s do t długości 4 wiedzie przez wierzchołki s, b, a i t .

Graf zwany *hiperkostką* H_k zdefiniowany jest rekurencyjnie. H_1 składa się z dwóch wierzchołków połączonych krawędzią. Natomiast hiperkostkę H_k wymiaru k budujemy z dwóch kostek H_{k-1} wymiaru $k - 1$. W pierwszej kostce etykietujemy wierzchołki dopisując 0 na początku nazwy każdego wierzchołka, natomiast w drugiej kostce etykietujemy wierzchołki dopisując 1 na początek. Następnie łączymy krawędziami odpowiadające sobie wierzchołki z obu kopii, czyli wierzchołek $0x$ jest połączony z wierzchołkiem $1x$ dla każdego x z $\{0, 1\}^{k-1}$.



Protokół zbierania wiadomości w hiperkostce H_k .

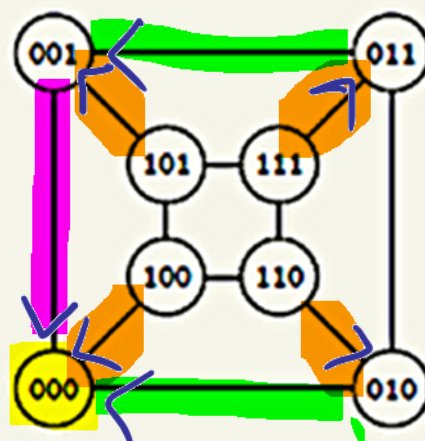
$000\underline{1}10\underline{1} \rightarrow 000\underline{0}101$

1. Dla każdego i od 1 do k , wykonuj:

1.1 Każdy wierzchołek o etykiecie $x = 0^{i-1}\underline{1}\sigma$, gdzie $\sigma \in \{0, 1\}^{k-i}$, przekazuje zebrane dane do wierzchołka o etykiecie $0^{i-1}\underline{0}\sigma$.

Prześledźmy działanie powyższego algorytmu na hiperkostce H_3 .

◀ PRZYKŁAD



$\begin{matrix} 100 \\ 101 \\ 110 \\ 111 \end{matrix} \rightarrow \begin{matrix} 000 \\ 001 \\ 010 \\ 011 \end{matrix}$
 $\begin{matrix} 010 \\ 011 \end{matrix} \rightarrow \begin{matrix} 000 \\ 001 \end{matrix}$
 $00\underline{1} \rightarrow 000$

- W pierwszej iteracji, dla $i = 1$, wierzchołek 100 przekazuje dane do 000, wierzchołek 101 do 001, wierzchołek 110 do 010, a wierzchołek 111 do 011.
- W drugiej iteracji, dla $i = 2$, wierzchołek 010 przekazuje wszystkie dane (swoje i otrzymane) do 000, a wierzchołek 011 do 001.
- W trzeciej iteracji, dla $i = 3$, wierzchołek 001 przekazuje zebrane wiadomości do 000.