Pamiętanie (reprezentacja) grafów

$$G = (V, E)\,,\ V = \{1, 2, \cdots, n\}\,,\ |V| = n,\ |E| = m$$

1. Macierz sąsiedztwa,

macierz $A_{n\times n}$, gdzie

$$a_{i,j} = \begin{cases} 1, \text{if } i, j \in E \\ 0, \text{w pp.} \end{cases}$$

A - macierz symetryczna.

2. Macierz incydencji,

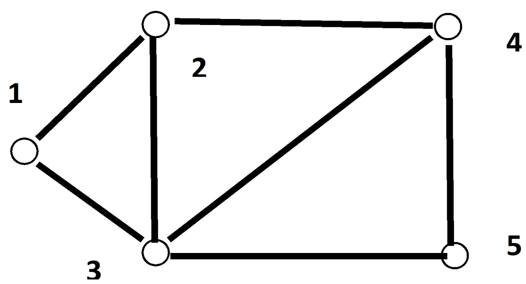
macierz $B_{n\times m}$ gdzie

$$b_{i,e} = \begin{cases} 1, \text{if } i \in e \\ 0, \text{w pp.} \end{cases} , i \in V, e \in E$$

3. Listy sąsiadów,

lista wierzchołków, każdy wierzchołek zawiera listę swoich sąsiadów.

PRZYKŁAD

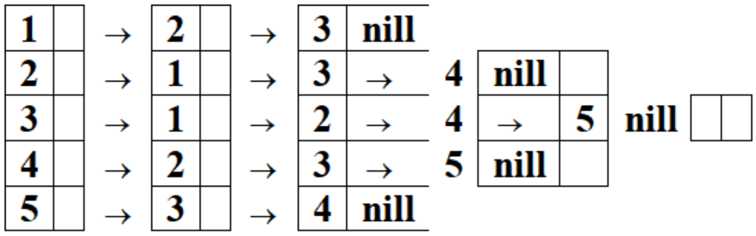


	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	1	0
3	1	1	0	1	1
4	0	1	1	0	1
5	0	0	1	1	0

macierz sąsiedztwa

	a	b	c	d	e	f	g
1	1	0	1	0	0	0	0
2	1	1	0	0	0	0	1
3	0	0	1	1	1	0	1
4	0	1	0	1	0	1	0
5	0	0	0	0	1	1	0

macierz incydencji



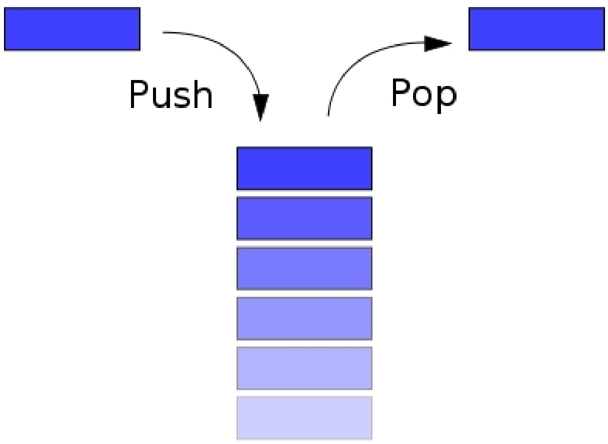
listy sąsiadów

Sprawdzenie – czy istnieje krawędź w grafie? operacje: wstawiania, usuwania – wierzchołka, krawędzi.

Struktury danych

1. Lista Uporządkowana struktura danych.

2. Stos Jest to lista, w której wszystkie operacje dodawania oraz usuwania elementu dotyczą wyłącznie ostatniego elementu, czyli tzw. *wierzchołka stosu.* Dzięki temu element ostatnio wstawiony zostanie usunięty jako pierwszy (stąd nazwa struktury – LIFO, last in, first out). Operacje obsługi stosu to włożenie na stos, zdjęcie ze stosu.



Są pewne operacje, jakie można wykonywać na stosie. Oto ich formalny zapis:

- push(obiekt) – czyli odłożenie obiektu na stos;
- pop() – ściągnięcie obiektu ze stosu i zwrócenie jego wartości;
- isEmpty() - sprawdzenie czy na stosie znajdują się już jakieś obiekty.

Implementacja:

Strukturami danych służącymi do reprezentacji stosu mogą być *tablice* (gdy znamy maksymalny rozmiar stosu), *tablice dynamiczne* lub *listy.* Złożoność obliczeniowa operacji na stosie zależy od konkretnej implementacji, ale w większości przypadków jest to czas stały O(1).

3. Kolejka

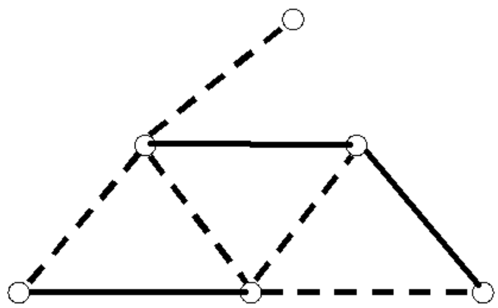
Kolejka to lista, w której wstawianie nowych elementów odbywa się na końcu listy, zaś usuwanie na początku listy (stąd nazwa struktury – FIFO, *first in, first out*).

Kolejkę można implementować jako zwarty blok komórek pamięci z dwiema komórkami specjalnymi przechowującymi wskaźniki początku i końca kolejki.

Definicja

$G(V, E)$ graf spójny. Spójny, acykliczny podgaraf G, zawierający wszystkie wierzchołki grafu G nazywamy *drzewem rozpinającym* (spinającym).

PRZYKŁAD



Przeszukiwanie grafu

$$G = (V, E)\,,\ V = \{1, 2, \cdots, n\}\,,\ |V| = n,\ |E| = m$$

graf pamiętany w postaci list sąsiadów wierzchołków, tj.

$\forall v \in V$ dana jest lista $ZA(v)$ – wierzchołków sąsiednich z v.

Rozwiązanie problemu grafowego, które zależy od poznania struktury połączeń w całym grafie, wymaga przeszukania jego wierzchołków i krawędzi. Takie przeszukanie polega na *odwiedzeniu* każdego wierzchołka i zbadaniu krawędzi incydentnych z odwiedzanym wierzchołkiem.

Przeszukiwanie w głąb (*Depth-first search*, w skrócie *DFS*)

Idea:

Startujemy z ustalonego wierzchołka i odwiedzamy jego sąsiadów (jeszcze nie odwiedzonych). Następnie, postępujemy analogicznie z ostatnim z odwiedzonych sąsiadów (realizujemy to poprzez umieszczanie odwiedzanych wierzchołków na stosie).

Za(*v*) – lista sąsiadów $v \in V$, **for** *i*:=1 **to** *n* **do** *Nowy*[*i*]:=**true**;

```
Procedure DFS(v);
    Stos:=v;  Nowy[v]:=false;
    while Stos≠∅ do
        begin
            t:=Stos;            {pobranie ze stosu}
            Odwiedz(t);         {*}
            for u∈Za[t] do
                if Nowy[u] then
                    begin
                        Nowy[u]:=false;     { Poprz[u]:=t }
                        Stos:=u
                    end {if}
        end {while}
end
```

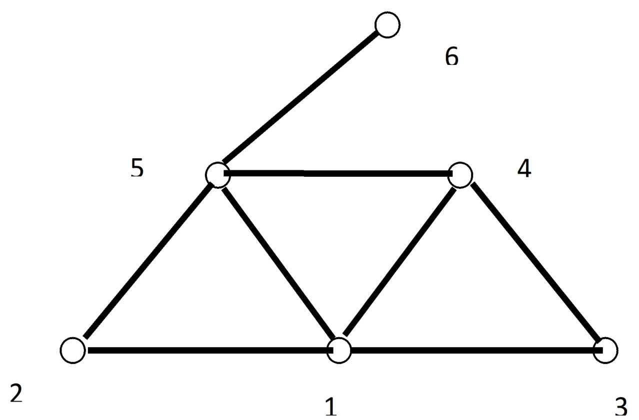
Uwagi:

1. Każdy wierzchołek jest odwiedzany dokładnie raz (zmienna *Nowy[u]*).
2. Efektem jest drzewo spinające *DFS* (wstawiając w miejsce * instrukcję: *Poprz[u]:=t*).

Złożoność czasowa algorytmu wynosi $O(n + m)$.

PRZYKŁAD.

Wyznaczyć kolejność (numery) odwiedzanych wierzchołków (startując z 1).



i	*Za*[*i*]	*Nowy*[*i*]
1	2,3,4,5	T
2	1,5	T
3	1,4	T
4	1,3,5	T
5	1,2,4,6	T
6	5	T

Zastosowania algorytmu

- Sprawdzania, czy istnieje ścieżka między dwoma wierzchołkami w grafie.
- Wyznaczania spójnych składowych (dodatkowa tablica z numerem wierzchołka startowego – identyfikuje składową spójności).
- Wyznaczanie mostów, tj. krawędzi, których usunięcie rozspaja graf (usunąć krawędź i sprawdzić liczbę składowych).

Złożoność $O(m(n+m))$.

Przeszukiwanie w wszerz (*Breadth-first search*, w skrócie *BFS*)

Idea:

Startujemy z ustalonego wierzchołka i odwiedzamy jego sąsiadów (jeszcze nie odwiedzonych). Następnie, postępujemy analogicznie zgodnie z kolejnością ich odwiedzania (realizujemy to poprzez umieszczanie odwiedzanych wierzchołków w kolejce).

```
    Za(v) – lista sąsiadów v ∈ V,  for i:=1 to n do  Nowy[i]:=true;
Procedure BFS(v);
    Kolejka:=v;  Nowy[v]:=false;
    while Kolejka ≠ ∅ do
        begin
            p:=Kolejka;        {pobranie z kolejki}
            for  u ∈ Za[p]  do
                if  Nowy[u]  then
                    begin
                        Nowy[u]:=false;
                        Odwiedz(u);            {*}      {Poprz[u]:=p}
                        Kolejka:=u
                    end {if}
        end {while}
end
```

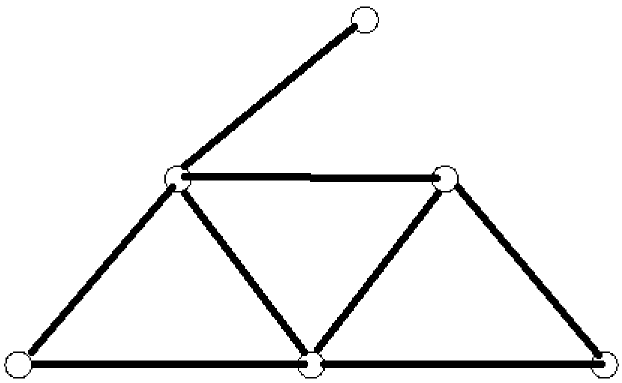
Złożoność czasowa algorytmu wynosi $O(n+m)$.

Uwagi:

1. Każdy wierzchołek jest odwiedzany dokładnie raz (zmienna *Nowy[u]*).
2. Efektem jest drzewo spinające *BFS* (wstawiając w miejsce * in-strukcję: *Poprz[u]:=p*).

PRZYKŁAD.

Wyznaczyć kolejność (numery) odwiedzanych wierzchołków oraz najkrótsze drogi z wierzchołka 1.



i	*Za[i]*	*Nowy[i]*
1	**2,3,4,5**	**T**
2	**1,5**	**T**
3	**1,4**	**T**
4	**1,3,5**	**T**
5	**1,2,4,6**	**T**
6	**5**	**T**

Niech T będzie *drzewem spinającym* wyznaczonym przez powyższy algorytm. Zawiera ono najkrótsze drogi z v do pozostałych wierzchołków w G.

Definicja

Wysokość $h(T)$ drzewa zakorzenionego w v jest równa długości najdłuższej drogi w T, której początkiem jest v.

Własność

Drzewo *BFS(v)* jest drzewem spinającym o najmniejszej wysokości spośród drzew rozpinających zakorzenionych w v.

Drzewo binarne:

- jeden wierzchołek stopnia 2 – korzeń,
- pozostałe wierzchołki stopnia 1 lub 3.

T – drzewo binarne o n wierzchołkach. Niech p będzie liczbą wierzchołków wiszących w T. Suma stopni wierzchołków

$$p + 3(n - p - 1) + 2 = 2(n - 1), \quad \text{stąd } p = \frac{n+1}{2}.$$

Definicja

Wierzchołek v jest na poziomie l drzewa, jeśli jego odległość od korzenia wynosi l.

na pooziomie 0	jest	1 wierzchołek
1	są co najwyżej	2 wierzchołki
2		4 wierzchołki
3		8 wierzchołków
…		

Zatem liczba wierzchołków w drzewie o k poziomach wynosi co najwyżej

$$2^0 + 2^1 + 2^2 + \cdots + 2^k \geqslant n,$$

$$\frac{2^{k+1}}{2-1} \geqslant n \quad \Rightarrow \quad 2^{k+1} \geqslant n+1 \quad \Rightarrow \quad k+1 \geqslant \log_2(n+1) \quad \Rightarrow \quad k \geqslant \log_2(n+1)+1.$$

Zatem minimalna liczba poziomów n wierzchołkowego drzewa binarnego

$$k_{\min} = \lceil \log_2(n+1)+1 \rceil$$

a maksymalna

$$k_{\max} = \frac{n-1}{2},$$

bo na każdym poziomie (z wyjątkiem zerowego) muszą być 2 wierzchołki.

Minimalne drzewa spinające

$G_c = (V; E; c)$, gdzie $G = (V, E)$ graf prosty, $(|V| = n, |E| = m)$, $c : E \to R^+$ *wagi krawędzi*

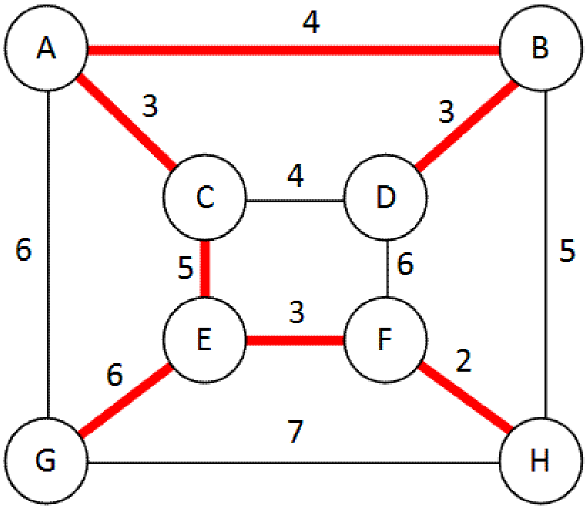
PROBLEM.
Wyznaczyć w G_c minimalne drzewo spinające, tj. o minimalnej sumie wag krawędzi.

Algorytm Kruskala (chciwości, zachłanny, skąpca)

Krok 0:
posortować tak krawędzie, aby
$c(e_1) \le c(e_2) \le \ldots \le c(e_m)$
Krok 1:
$T^* \leftarrow \varnothing$;
<u>for</u> i**:=1** **<u>to</u>** m **<u>do</u>**
<u>if</u> $T^* \cup \{e_i\}$ **[*nie zawiera cyklu*]** **<u>then</u>**
$T^* \leftarrow T^* \cup \{e_i\}$.

Uwagi:

1. Złożoność obliczeniowa
2. Budowany graf (drzewo) może nie być spójny.



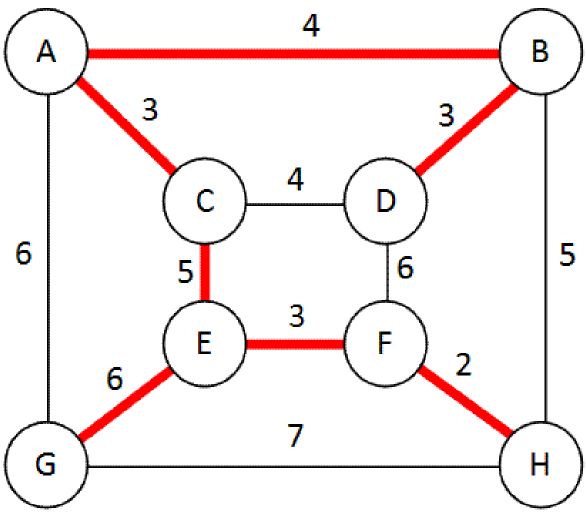
Dualny algorytm chciwości

Algorytm Prima

$T^* \leftarrow \emptyset$;
wybrać dowolny wierzchołek $x \in V$;
$W \leftarrow \{x\}$;
while $W \neq V$ **do**
begin
    wybrać w E **krawędź** $\{w,v\}$ **najmniejszej wadze taką, że**
    $[w \in W \wedge v \in V \setminus W]$ **and** [*po jej dodaniu nie powstanie cykl*];
        $T^* \leftarrow T^* \cup \{w,v\} \wedge W \leftarrow W \cup \{v\}$
end.

(Złożoność $O(n^2)$ przy pamiętaniu grafu – macierz sąsiedztwa)

PRZYKŁAD.



$G_c = (V, E, c)$ - sieć. Wyznaczyć minimalne drzewo spinające.

Algorytm Prima-Dijkstry

Z każdym wierzchołkiem $i \in V$ są związane dwie cechy:

p_i - odległość (w sensie c) od bieżącego rozwiązania częściowego,

q_i - nr. wierzchołka z rozw. częściowego realizującego tę odległość.

$s \in V$ - wierzchołek startowy.

Krok 1:

$p_s := 0;\ q_s := 0;\ r := s;\ T^* \leftarrow \varnothing;$

$p_v := \infty;\ q_v := 0\ \forall v \in W = V \setminus \{s\};$

Krok 2:

Wykonać *n*-1 razy następujące polecenia:

(i) *{uaktualnić cechy}*

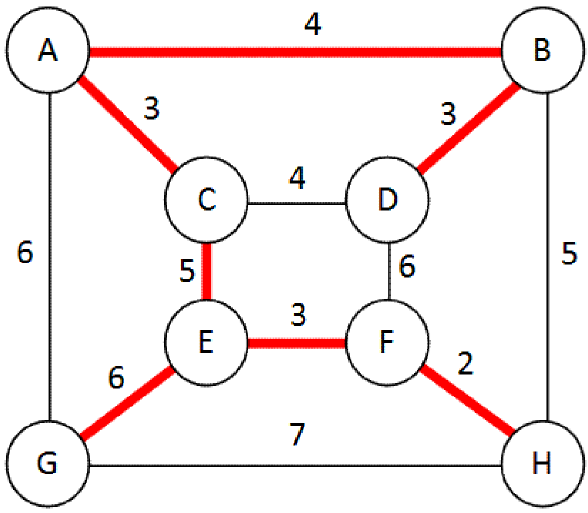
dla $v \in W$**, jeśli** $p_v > c_{r,v}$**, to**

$p_v := c_{r,v} \wedge q_v := r;$

(ii) *{wyznaczenie kolejnej dołączanej do drzewa krawędzi}*

Wyznaczyć $p_w = \min\{p_v : v \in W\};$

$T^* \leftarrow T^* \cup \{q_w, w\} \wedge W \leftarrow W \setminus \{w\} \wedge r := w;$



$G = (V, A)$ - digraf (graf skierowany)

indeg(v), outdeg(v) – stopień wejściowy i wyjściowy wierzchołka.

Droga $d(u, v)$ – ciąg łuków; *cykl* – droga zamknięta.

$\forall u, v \in V$

a) istnieje droga z $d(u, v)$ lub $d(v, u)$ – digraf *spójny*,

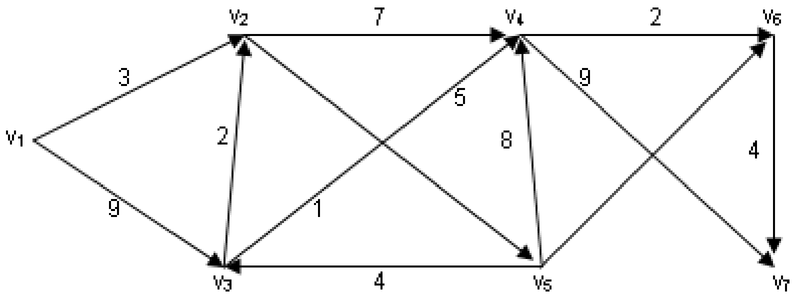
b) istnieje droga z $d(u, v)$ i $d(v, u)$ – digraf *silnie spójny*,

c) po zamianie łuków na krawędzie graf jest spójny – digraf *słabo spójny*.

Topologiczne porządkowanie

Wierzchołki acyklicznego digrafu można uporządkować (tzw. porządek topologiczny). Efekt – łuki prowadza od wierzchołka o mniejszym, do wierzchołka o większym numerze.

```
Q←Kolejka wierzchołkami o stopniu wchodzącym 0
while Q ≠ ∅ do
begin
    pobierz wierzchołek v z Q i nadaj mu kolejny numer;
    for all wierzchołka u o łuku e=(v,u) do
    begin
            usuń łuk e z grafu;
            if u nie jest końcem żadnego łuku then
            wstaw u do Q
    end
end.
```



Własności acyklicznego digrafy:

1. Istnieje wierzchołek v taki, że $\text{indeg}(v) = 0$.

2. Niech $\text{indeg}(v) = 0$. Jeżeli z digrafu usuniemy wierzchołek v z „wychodzącymi” łukami, to powstały digraf ma własność 1.

Kolorowanie

Definicja

Graf jest k-kolorowalny, gdy

$$(\exists f : V \to \{1, 2, \cdots, k\})\,(\forall i)\left(P_2(f^{-1}(i)) \cap E = \varnothing\right)$$

tj. zbiór wierzchołków można podzielić na k podzbiorów w ten sposób, aby żadne dwa wierzchołki należące do jednego podzbioru nie były połączone krawędzią.

Definicja

Najmniejsze k takie, że G jest k-kolorowalny nazywamy liczbą chromatyczną grafu G i oznaczamy $\chi(G)$.

Definicja

Graf jest k-kolorowalny krawędziowo jeżeli jego krawędzie można pokolorować k kolorami tak, żeby żadna para krawędzi sąsiednich nie miała tego samego koloru.

Definicja

Najmniejsze k takie, że G jest k-kolorowalny krawędziowo nazywamy indeksem chromatycznym grafu G i oznaczamy $\chi'(G)$.

Twierdzenia

Twierdzenie
Grafy dwudzielne są dwukolorowalne.

Twierdzenie
W grafie G o p wierzchołkach zachodzi nierówność:

$$\frac{p}{\beta_0} \leqslant \chi(G) \leqslant p - \beta_0 + 1.$$

Twierdzenie Szekerés, Wilf (1968)
Jeżeli H są indukowanymi podgrafami G, to zachodzi nierówność:

$$\chi(G) \leqslant \max \delta(H) + 1.$$

Twierdzenie Brooks (1941)
Jeżeli G jest spójnym grafem prostym, nie będącym grafem pełnym, i jeśli największy stopień wierzchołka grafu G wynosi Δ (gdzie $\Delta \geqslant 3$), to graf G jest Δ - kolorowalny.

Twierdzenie Appel, Haken (1976)
Każdy graf planarny jest 4-kolorowalny.

Twierdzenie (1968)

$$(\forall G)\, \Delta(G) \leqslant \chi'(G) \leqslant \Delta(G) + 1.$$

Grafy planarne

Definicja
G jest grafem planarnym $\equiv$ istnieje taka reprezentacja graficzna grafu G na płaszczyźnie, że łuki reprezentujące krawędzie nie mają punktów wspólnych poza wierzchołkami w których się łączą.

Twierdzenie Kuratowski (1930)
G jest grafem planarnym iff, gdy G nie zawiera podgrafu homeomorficznego z K_5 , ani $K_{3,3}$.

Twierdzenie Euler (1750)
Niech G będzie rysunkiem płaskim spójnego grafu płaskiego i niech n, m i f oznaczają odpowiednio liczbę wierzchołków, krawędzi i ścian grafu G. Wtedy:

$$n - m + f = 2.$$

Wniosek

Jeśli G jest spójnym planarnym grafem prostym, mającym n wierzchołków (gdzie $n \geqslant 3$) i m krawędzi, to $m \leqslant 3n - 6$. Jeśli ponadto graf G nie zawiera K_3, to $m \leqslant 2n - 4$.

Lemat

G – planarny $\equiv \delta(G) \leqslant 5$

Twierdzenie Tutte (1956)

G – planarny i $\kappa(G) \geqslant 4 \Rightarrow G$ ma cykl Hamitona.

Algorytm Fleury'go (wyznaczanie drogi Eulera)

Oznaczenia: ES – ciąg krawędzi drogi lub cyklu Eulera, VS – ciąg wierzchołków tej drogi lub cyklu, $V(G)$ – zbiór wierzchołków, $E(G)$ – zbiór krawędzi grafu G.

1. Wybierz dowolny wierzchołek v nieparzystego stopnia, jeśli taki istnieje, wybierz dowolny wierzchołek za v.

   Niech $VS = v$ oraz $ES = \varnothing$.

2. Jeśli z wierzchołka v nie wychodzi już żadna krawędź, to STOP.

   Jeśli jest dokładnie jedna krawędź incydentna z v, powiedzmy krawędź e z wierzchołka v do w, to usuń e z $E(G)$ oraz v z $V(G)$ i przejdź do kroku 4.

3. Jeśli została więcej niż jedna krawędź incydentna z v, to wybierz krawędź, powiedzmy e z v do w, po usunięciu której graf pozostanie spójny, następnie usuń e z $E(G)$.

4. Dołącz w na końcu ciągu VS, dołącz e na końcu ciągu ES, zastąp v wierzchołkiem w i przejdź do kroku 2.