Algorytm z powracaniem (ang. *backtracking algorithm*) to ogólna technika algorytmiczna, wykorzystująca rekurencję, która polega na rozwiązywaniu problemu obliczeniowego w następujący sposób:

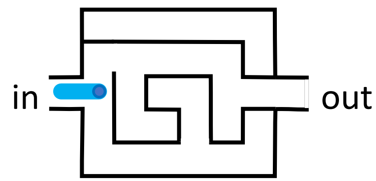
1. Algorytm buduje rozwiązanie problemu stopniowo, po każdym kroku sprawdzając czy aktualna kombinacja (kandydat na rozwiązanie) jest dopuszczalna i czy jest poszukiwanym rozwiązaniem problemu.

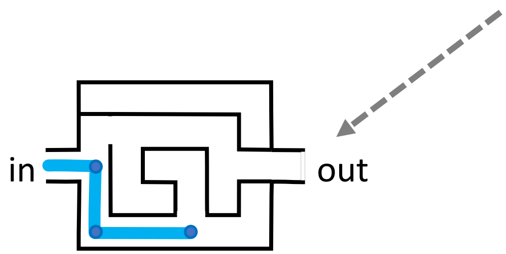
2. Jeśli dana kombinacja nie jest dopuszczalna lub nie jest szukanym rozwiązaniem algorytm powraca do stanu, w którym może wygenerować innego kandydata i stopniowo buduje dalej.

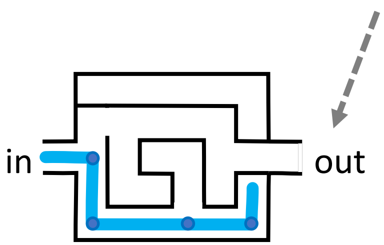
3. Jeśli dana kombinacja jest rozwiązaniem problemu algorytm kończy działanie (jeśli szukał tylko jednego rozwiązania) lub – powracając – generuje kolejne kombinacje (jeśli szukał wielu rozwiązań).

Poszukiwanie drogi w labiryncie.

• punkt decyzyjny



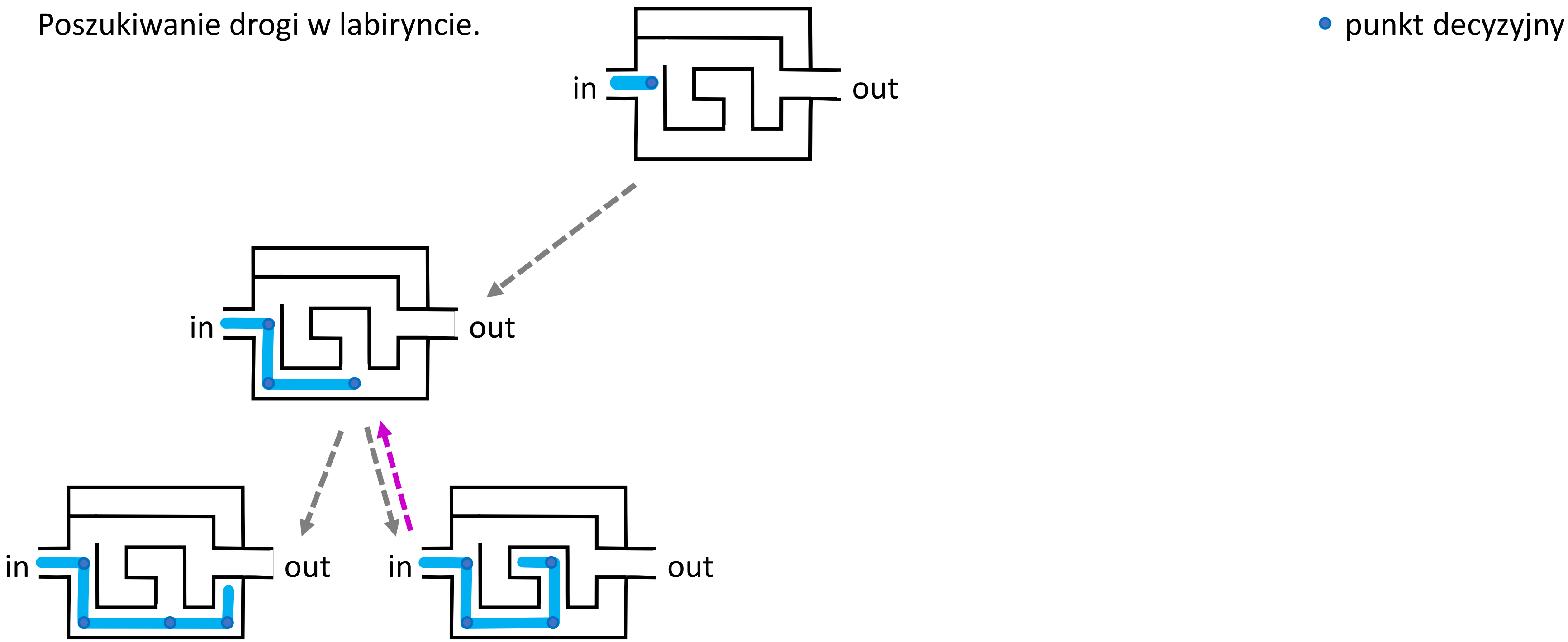




To nie jest rozwiązanie.
***Wróć** do poprzedniego stanu.*

Algorytmy z powracaniem: przykład

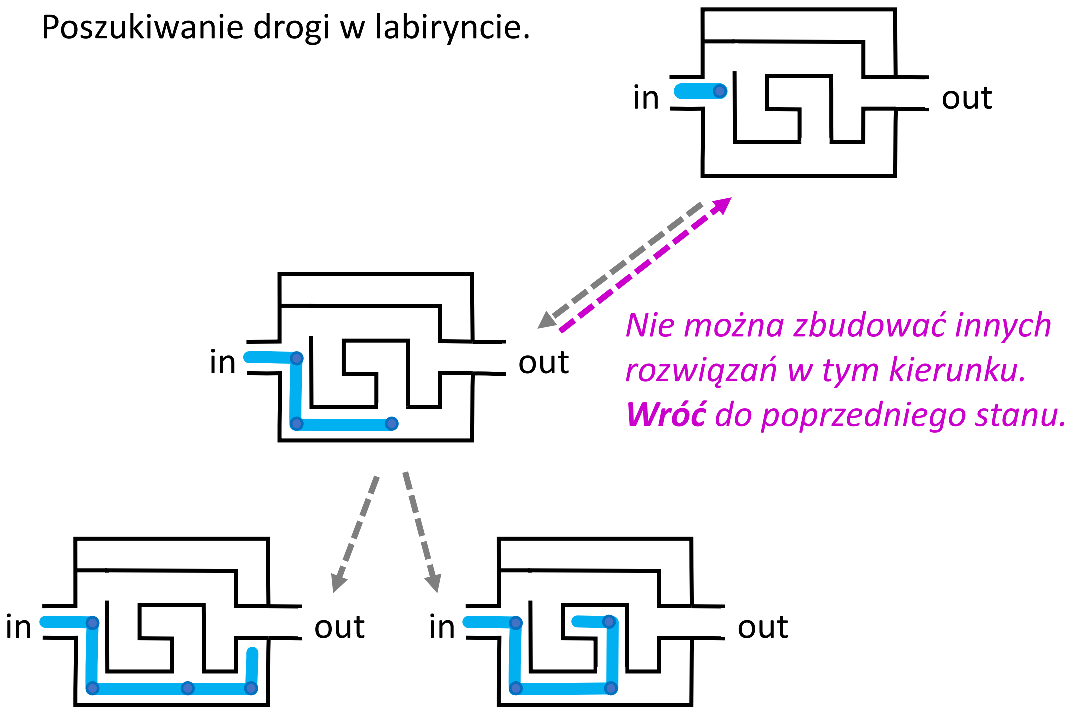
Poszukiwanie drogi w labiryncie.



To nie jest rozwiązanie.
Wróć *do poprzedniego stanu.*

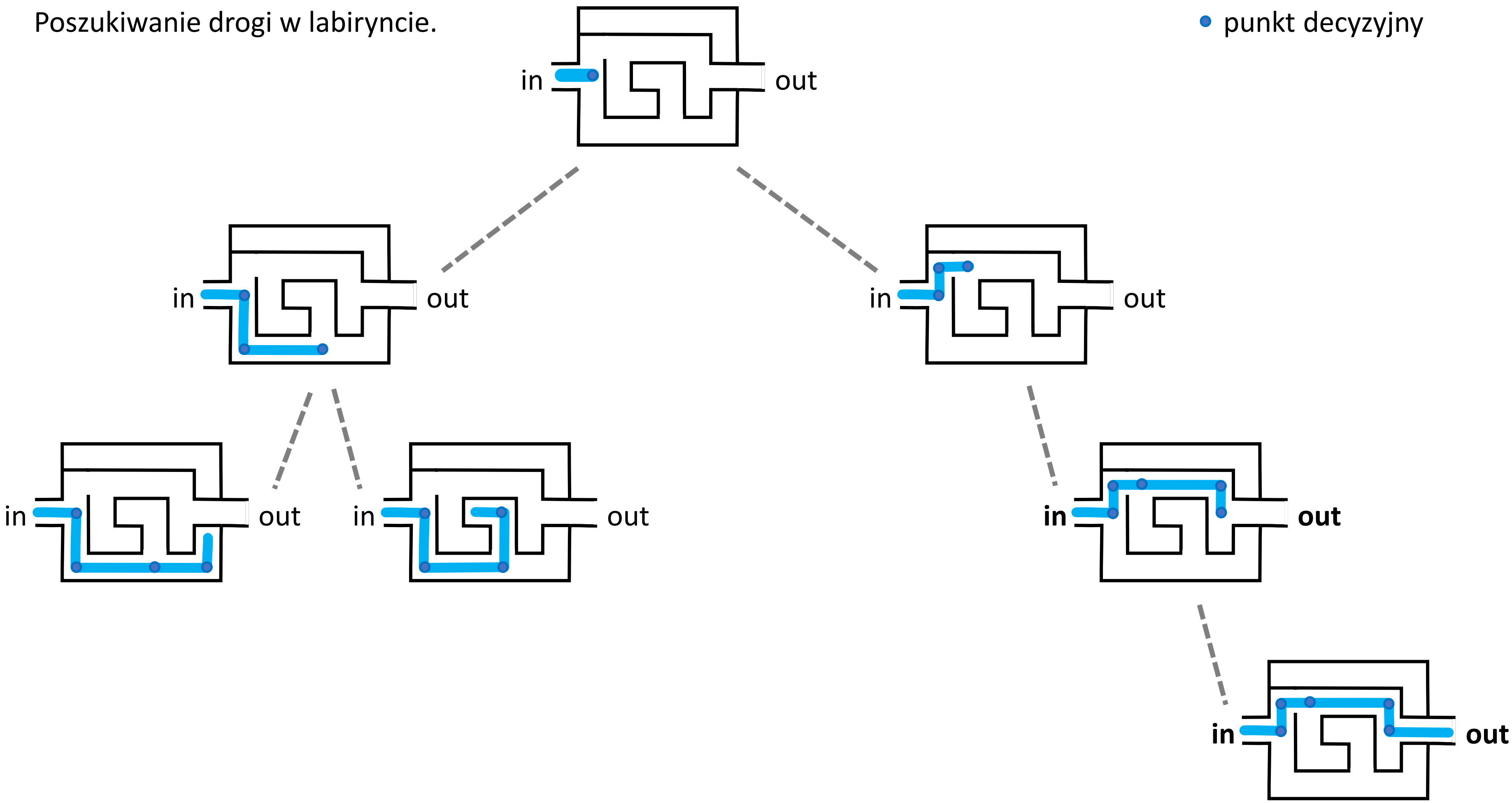
Algorytmy z powracaniem: przykład

Poszukiwanie drogi w labiryncie.



• punkt decyzyjny

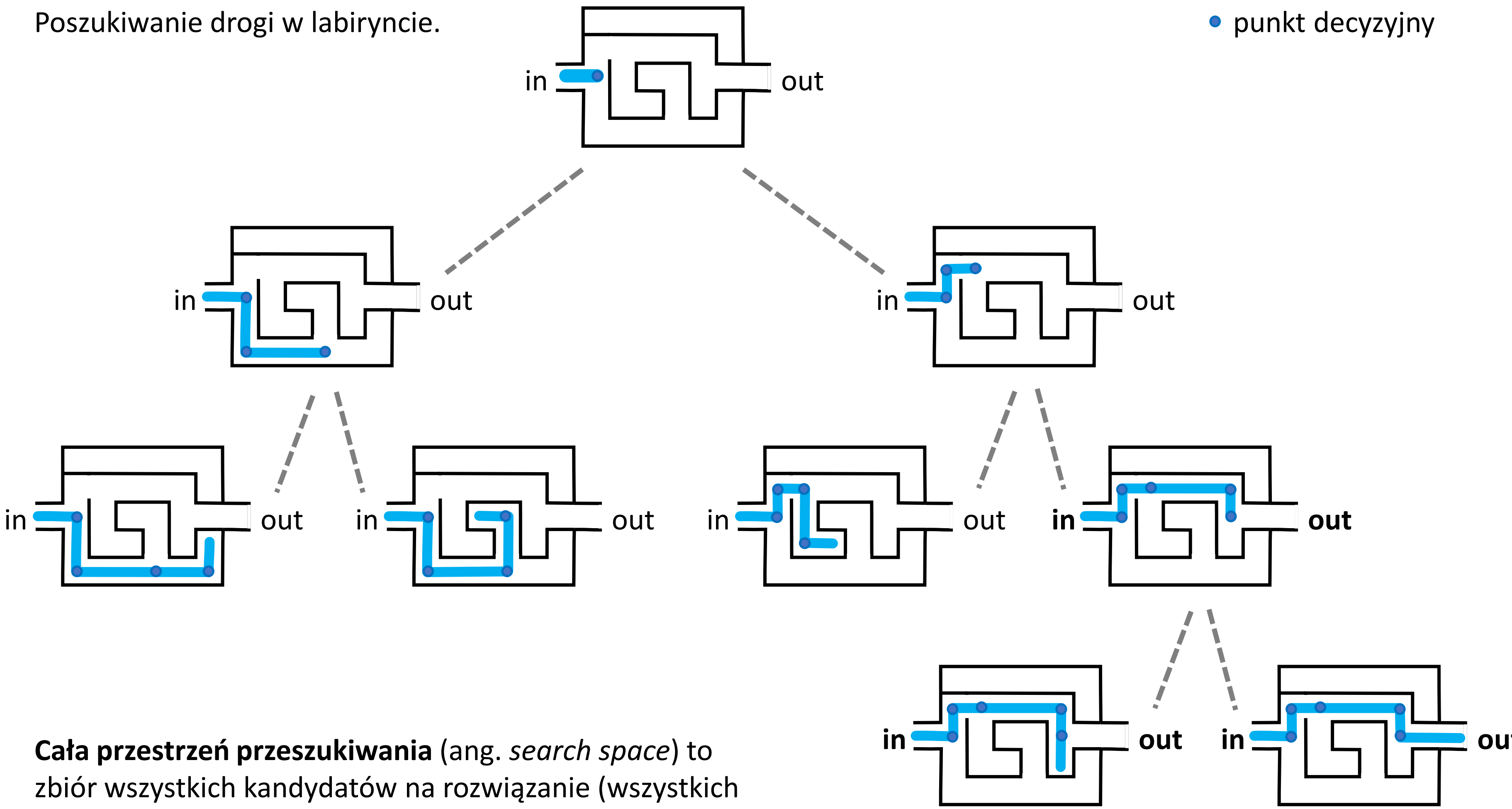
Poszukiwanie drogi w labiryncie.



To jest rozwiązanie!

Algorytmy z powracaniem: przykład

Poszukiwanie drogi w labiryncie.



Cała przestrzeń przeszukiwania (ang. *search space*) to zbiór wszystkich kandydatów na rozwiązanie (wszystkich stanów, w których może się znaleźć algorytm).

Algorytm z powracaniem

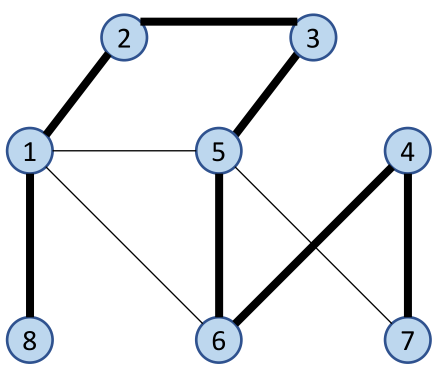
- generuje każdego kandydata co najwyżej jeden raz (tj. nie wraca do rozwiązania/stanu, w którym już był i z którego się wycofał);
- rozwiązuje **różne problemy**:
  1. Problemów przeszukiwania (ang. *search problems*) (gdzie szukamy jednego lub wielu rozwiązań dopuszczalnych)
  2. Problemów optymalizacyjnych (ang. *optimization problems*) (gdzie szukamy jednego najlepszego rozwiązania)
- jest algorytmem **dokładnym** (ang. *exact algorithm*) (tzn. znajduje najlepsze rozwiązanie problemu optymalizacyjnego lub rozwiązanie problemu przeszukiwania o ile ono istnieje);
- ma **dużą złożoność obliczeniową** (zależnie od problemu, ale najczęściej jest to złożoność wykładnicza);
- jeśli szukamy wielu rozwiązań dopuszczalnych lub rozwiązania optymalnego, to stosuje przeszukiwanie **wyczerpujące/siłowe** (ang. *exhaustive search/brute-force search*), czyli sprawdza wszystkie możliwe rozwiązania w przestrzeni przeszukiwań (jeśli szukamy tylko jednego rozwiązania dopuszczalnego to niekoniecznie sprawdzi wszystkie stany);
- zazwyczaj jest stosowany do rozwiązania problemu, dla którego nie ma lepszej metody (lepszej z punktu widzenia złożoności obliczeniowej);

Problem poszukiwania ścieżki lub cyklu Hamiltona w grafie to jeden z problemów, które rozwiązuje się algorytmem z powracaniem.

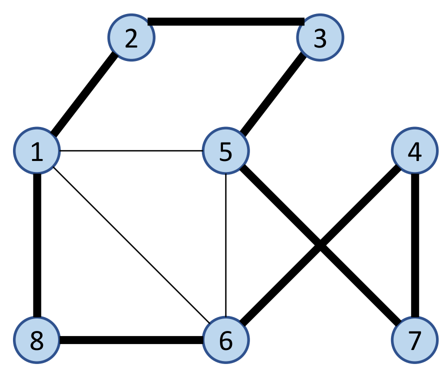
Ścieżka Hamiltona w grafie *G*=(*V*,*E*) to ścieżka prosta, która przechodzi przez wszystkie wierzchołki tego grafu (tzn. przez każdy wierzchołek grafu przechodzi dokładnie jeden raz).

Cykl Hamiltona w grafie to zamknięta ścieżka Hamiltona (zaczyna się i kończy w tym samym wierzchołku).

Graf, który zawiera ścieżkę Hamiltona to **graf półhamiltonowski**.
Graf, który zawiera cykl Hamiltona, to **graf hamiltonowski.**



Graf półhamiltonowski
Ścieżka Hamiltona: 8-1-2-3-5-6-4-7



Graf hamiltonowski
Cykl Hamiltona: 1-2-3-5-7-4-6-8-1

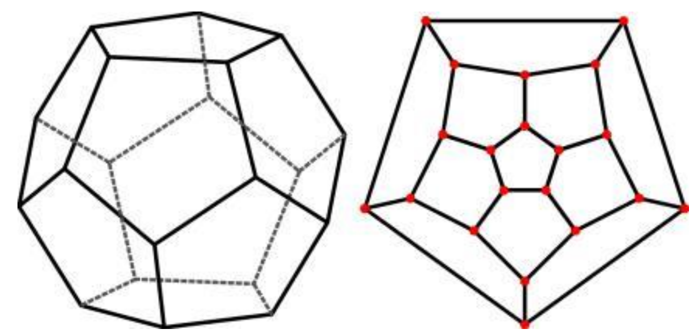
Spłaszczenie 12-ścianu

W roku 1859 irlandzki fizyk i matematyk William Rowan Hamilton spłaszczył wymyślił łamigłówkę znaną jako „Icosian game”. Planszę gry stanowił graf płaski o 20 wierzchołkach, który powstał poprzez spłaszczenie 12-ścianu foremnego.

Spłaszczenie było rzutem bryły na płaszczyznę takim, aby jej krawędzie i wierzchołki utworzyły graf płaski. Ze względów estetycznych Hamilton chciał też, aby graf był symetryczny.

Graf płaski: graf, którego krawędzie nie przecinają się.

Rzuty 12-ścianu foremnego na płaszczyznę



graf płaski

W oryginalnej łamigłówce-zabawce wierzchołki grafu były otworami w drewnianej płytce. Zabawa polegała na umieszczaniu w nich 20 ponumerowanych kołeczków tak, aby droga wzdłuż krawędzi od kołeczka 1 do 20 wiodła przez kolejne liczby, a ponadto 1 i 20 znajdowały się na końcach tej samej krawędzi. Inaczej mówiąc, należało obejść wszystkie wierzchołki, nie goszcząc dwukrotnie w żadnym – oprócz startowego, który był także metą. Całość miała kojarzyć się z podróżą, bo otwory oznaczone były pierwszymi literami nazw miast wymienionych w instrukcji. Gra dała początek wielu problemom w teorii grafów.

Problem przeszukiwania

- Poszukiwanie ścieżki Hamiltona -

Definicja problemu

Dany jest graf *G*=(*V*,*E*). **Znajdź w grafie *G* ścieżkę Hamiltona**, tj. takie uszeregowanie wierzchołków tego grafu, że każdy wierzchołek występuje w uszeregowaniu dokładnie jeden raz.

Rozwiązanie problemu

Ścieżka prosta przechodząca przez wszystkie wierzchołki grafu *G*.

Klasa złożoności problemu

Jest to problem trudny obliczeniowo. Wersja przeszukiwania należy do klasy problemów NP-trudnych, a dokładnie do problemów silnie NP-trudnych.

Problem decyzyjny

- Istnienie ścieżki Hamiltona -

Definicja problemu

Dany jest graf *G*=(*V*,*E*). **Czy graf *G* zawiera ścieżkę Hamiltona**, tj. takie uszeregowanie swoich wierzchołków, że każdy wierzchołek występuje w uszeregowaniu dokładnie jeden raz?

Rozwiązanie problemu

Odpowiedź: TAK (graf zawiera ścieżkę Hamiltona) lub NIE (graf nie zawiera takiej ścieżki).

Klasa złożoności problemu

Jest to problem trudny obliczeniowo. Wersja decyzyjna jest w klasie problemów NP-zupełnych (a dokładnie silnie NP-zupełnych).

Jak rozwiązać problem ścieżki/cyklu Hamiltona

Jak można rozwiązać problem ścieżki/cyklu Hamiltona w grafie?

Problem przeszukiwania

Poprzez zastosowanie algorytmu poszukującego ścieżkę/cykl Hamiltona w zadanym grafie $G=(V,E)$

- Algorytm Robertsa-Floresa
- Algorytm Kernighana-Lina
- Algorytm Christofidesa
- Algorytm Christofidesa-Selby'ego
- Algorytmy mrówkowe

Problem decyzyjny

Poprzez sprawdzenie czy zadany graf spełnia warunek istnienia ścieżki/cyklu Hamiltona?

Czy istnieją takie warunki?
- Znane są warunki wystarczające.
- Znany jest warunek konieczny.

Nie istnieje jednak warunek konieczny i wystarczający (jednocześnie).

Warunek dostateczny, warunek konieczny

Warunek wystarczający $\Rightarrow$ Fakt

Z warunku wystarczającego wynika fakt. Ale fakt może również zachodzić pomimo, iż warunek wystarczający nie jest spełniony!

Fakt $\Rightarrow$ Warunek konieczny

Z faktu wynika, że spełniony jest warunek konieczny. Ale jeśli jest spełniony warunek konieczny, to nie znaczy, że zachodzi fakt!

Warunki wystarczające istnienia cyklu Hamiltona

Warunki wystarczające istnienia cyklu Hamiltona w grafie definiują m.in. twierdzenia:

- twierdzenie Diraca (1952),
- twierdzenie Orego (1961),
- twierdzenie o liczbie krawędzi,
- twierdzenie Bondy'ego-Chvátala,
- twierdzenie dla grafu dwudzielnego.

Wszystkie one w różny sposób mówią o tym, że jeśli graf jest dostatecznie gęsty to zawiera cykl Hamiltona.

Warunek konieczny istnienia cyklu Hamiltona

Jeżeli graf $G=(V,E)$ jest hamiltonowski to dla każdego niepustego podzbioru $V' \subset V$ spełniony jest warunek:

$$\omega(V - V') \leq |V'|,$$

gdzie ω oznacza liczbę spójnych składowych grafu G.

Nie istnieje warunek konieczny i wystarczający istnienia ścieżki/cyklu Hamiltona w dowolnym grafie.

Istnieją grafy hamiltonowskie, które nie spełniają żadnego warunku wystarczającego.

Graf, który spełnia warunek konieczny nie musi być grafem hamiltonowskim.

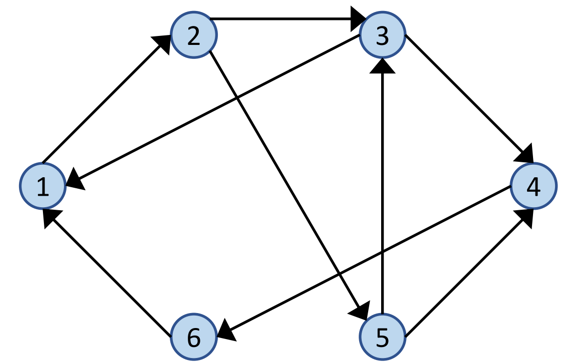
Czyli jak można rozwiązać decyzyjną wersję problemu ścieżki/cyklu Hamiltona w grafie?

Poprzez rozwiązanie wersji przeszukiwania. Jeśli znajdziemy ścieżkę/cykl to automatycznie rozwiążemy problem decyzyjny.

- Algorytm Robertsa-Floresa
- Algorytm Kernighana-Lina
- Algorytm Christofidesa
- Algorytm Christofidesa-Selby'ego
- Algorytmy mrówkowe

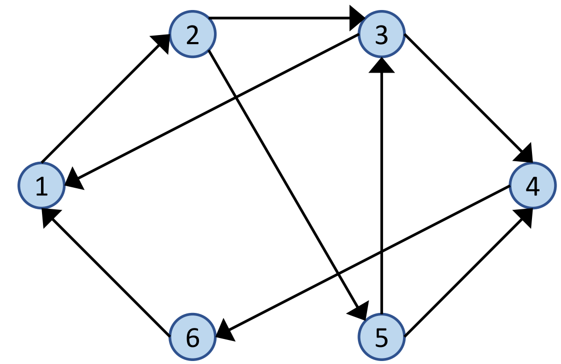
Algorytm Robertsa-Floresa

```
n - liczba wierzchołków
O[1..n] – tablica, w której zaznaczamy odwiedzone wierzchołki
Path[1..n] - tablica zawierająca konstruowany cykl Hamiltona

bool Hamiltonian(int v){
O[v] = True;
visited++;
for (i in Successors(v))
    if (i == start && visited == n)
        return True;
    if (not O[i])
        if (Hamiltonian(i))
            Path[k++] = v
            return True;
O[v] = False;
visited--;
return False; }

bool Hcycle {
for (i=1; i<=n; O[i++] = False);
Path[1] = start = 1;
visited = 0;
k = 2;
HCycle = Hamiltonian(start); }
```



S.M. Roberts, B. Flores (1966) Systematic Generation of Hamiltonian Circuits. *Communications of the ACM* 9(9):690-694.

Stos (ścieżka)	**Operacja**	**Odwiedzone O[]**					
		$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$	$\mathbf{v_4}$	$\mathbf{v_5}$	$\mathbf{v_6}$
$\{\}$	$\rightarrow v_1$	1	0	0	0	0	0
$\{\mathbf{v_1}\}$	$\rightarrow v_2$	1	1	0	0	0	0
$\{v_1, \mathbf{v_2}\}$	$\rightarrow v_3$	1	1	1	0	0	0
$\{v_1, v_2, \mathbf{v_3}\}$	$\rightarrow v_1$	1	1	1	0	0	0
$\{v_1, v_2, v_3, \mathbf{v_1}\}$	$\leftarrow$ (wróć)	1	1	1	0	0	0
$\{v_1, v_2, v_3\}$	$\rightarrow v_4$	1	1	1	1	0	0
$\{v_1, v_2, v_3, \mathbf{v_4}\}$	$\rightarrow v_6$	1	1	1	1	0	1
$\{v_1, v_2, v_3, v_4, \mathbf{v_6}\}$	$\rightarrow v_1$	1	1	1	1	0	1
$\{v_1, v_2, v_3, v_4, v_6, \mathbf{v_1}\}$	$\leftarrow$ (wróć)	1	1	1	1	0	1
$\{v_1, v_2, v_3, v_4, v_6\}$	$\leftarrow$ & $O[v_6] = 0$	1	1	1	1	0	0
$\{v_1, v_2, v_3, v_4\}$	$\leftarrow$ & $O[v_4] = 0$	1	1	1	0	0	0
$\{v_1, v_2, v_3\}$	$\leftarrow$ & $O[v_3] = 0$	1	1	0	0	0	0
$\{v_1, v_2\}$	$\rightarrow v_5$	1	1	0	0	1	0
$\{v_1, v_2, \mathbf{v_5}\}$	$\rightarrow v_3$	1	1	1	0	1	0
$\{v_1, v_2, v_5, \mathbf{v_3}\}$	$\rightarrow v_1$	1	1	1	0	1	0
$\{v_1, v_2, v_5, v_3, \mathbf{v_1}\}$	$\leftarrow$ (wróć)	1	1	1	0	1	0
$\{v_1, v_2, v_5, v_3\}$	$\rightarrow v_4$	1	1	1	1	1	0
$\{v_1, v_2, v_5, v_3, \mathbf{v_4}\}$	$\rightarrow v_6$	1	1	1	1	1	1
$\{v_1, v_2, v_5, v_3, v_4, \mathbf{v_6}\}$	$\rightarrow v_1$	1	1	1	1	1	1
$\{v_1, v_2, v_5, v_3, v_4, v_6, \mathbf{v_1}\}$	KONIEC	1	1	1	1	1	1



Lista następników

1 $\rightarrow$ 2

2 $\rightarrow$ 3 $\rightarrow$ 5

3 $\rightarrow$ 1 $\rightarrow$ 4

4 $\rightarrow$ 6

5 $\rightarrow$ 3 $\rightarrow$ 4

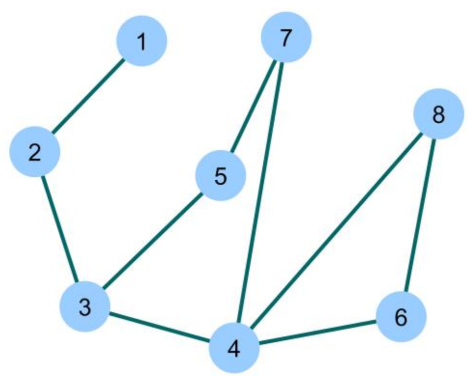
6 $\rightarrow$ 1

Problem poszukiwania ścieżki lub cyklu Eulera w grafie to kolejny problem, który rozwiązuje się algorytmem z powracaniem.

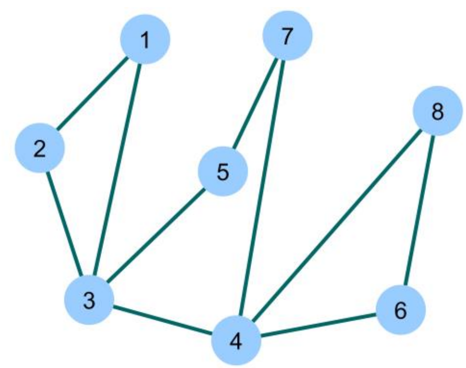
Ścieżka Eulera w grafie *G*=(*V*,*E*) to ścieżka prosta, która przechodzi przez wszystkie krawędzie tego grafu; przez każdą krawędź dokładnie jeden raz.

Cykl Eulera w grafie to zamknięta ścieżka Eulera (zaczyna się i kończy w tym samym wierzchołku).

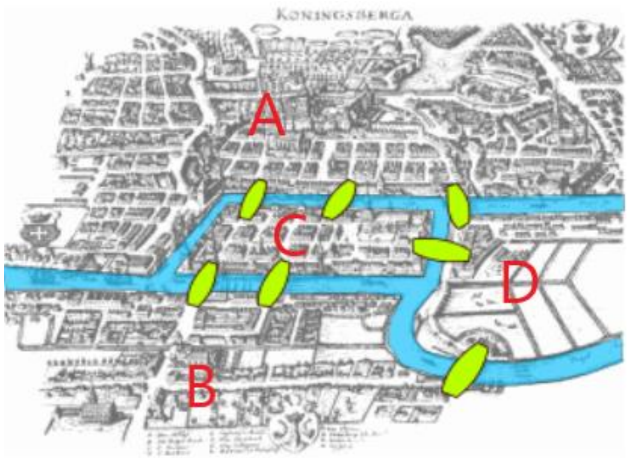
Graf, który zawiera ścieżkę Eulera to **graf półeulerowski**.
Graf, który zawiera cykl Eulera, to **graf eulerowski.**



Graf półeulerowski
Ścieżka Eulera: 1-2-3-5-7-4-8-6-4-3

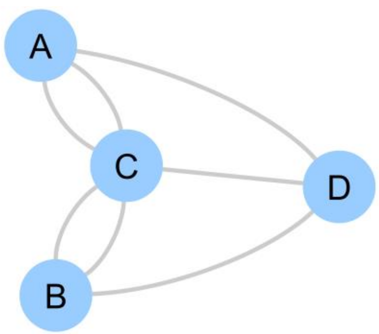


Graf eulerowski
Cykl Eulera: 1-2-3-5-7-4-8-6-4-3-1



Problem:
Czy można przejść przez wszystkie mosty, każdy dokładnie jeden raz, tak aby rozpocząć i skończyć w tym samym miejscu?

Rozwiązanie:
Leonard Euler w 1741 roku wykazał, że nie jest to możliwe ze względu na nieparzystą liczbę mostów, na każdej z wysp oraz na lądzie.



Graf, który skonstruował Leonard Euler z wierzchołków (ląd i wyspy) oraz krawędzi (mostów).

Problem ścieżki/cyklu Eulera

Problem przeszukiwania

- Poszukiwanie ścieżki Eulera -

Definicja problemu
Dany jest graf $G=(V,E)$. **Znajdź w grafie *G* ścieżkę Eulera**, tj. takie uszeregowanie krawędzi tego grafu, że każda krawędź występuje w uszeregowaniu dokładnie jeden raz.

Rozwiązanie problemu
Ścieżka prosta przechodząca przez wszystkie krawędzie grafu G.

Klasa złożoności problemu
Jest to problem łatwy obliczeniowo. Wersja przeszukiwania należy do klasy P.

Problem decyzyjny

- Istnienie ścieżki Eulera -

Definicja problemu
Dany jest graf $G=(V,E)$. **Czy graf *G* zawiera ścieżkę Eulera**, tj. takie uszeregowanie swoich krawędzi, że każda krawędź występuje w uszeregowaniu dokładnie jeden raz?

Rozwiązanie problemu
Odpowiedź: TAK (graf zawiera ścieżkę Eulera) lub NIE (graf nie zawiera takiej ścieżki).

Klasa złożoności problemu
Jest to problem łatwy obliczeniowo. Wersja decyzyjna jest w klasie problemów P.

Dobra wiadomość: dla obydwóch problemów istnieją algorytmy, którymi można rozwiązać dowolną instancję problemu **w czasie wielomianowym**!

Jak można rozwiązać problem ścieżki/cyklu Eulera w grafie?

Problem przeszukiwania

Poprzez zastosowanie algorytmu poszukującego ścieżkę/cykl Eulera w zadanym grafie $G=(V,E)$

- Algorytm Fleury'ego (1883)
  \+ wyszukiwanie mostów:
  - algorytm Tarjana (1997)
  - algorytm Thorupa (2000)
- Algorytm Hierholzera (1873)

Problem decyzyjny

Poprzez sprawdzenie czy zadany graf spełnia warunek istnienia ścieżki/cyklu Eulera.

Istnieją warunki dostateczne i konieczne istnienia ścieżki oraz cyklu Eulera w grafie nieskierowanym i skierowanym.

Warunek dostateczny i konieczny na istnienie CYKLU EULERA:

1) Graf jest spójny (poza izolowanymi wierzchołkami)

oraz

2a) jeśli graf jest *nieskierowany*: stopień każdego wierzchołka jest parzysty

2b) jeśli graf jest *skierowany*: dla każdego wierzchołka stopień wejściowy jest równy stopniowi wyjściowemu

Warunek dostateczny i konieczny na istnienie ŚCIEŻKI EULERA:

1) Graf jest spójny (poza izolowanymi wierzchołkami)

oraz

2a) jeśli graf jest *nieskierowany*: stopień każdego wierzchołka z wyjątkiem dwóch* jest parzysty

2b) jeśli graf jest *skierowany*: dla każdego wierzchołka z wyjątkiem dwóch* stopień wejściowy jest równy stopniowi wyjściowemu

* Dwa wierzchołki, które nie muszą spełniać warunku to wierzchołek startowy i wierzchołek końcowy ścieżki Eulera.

Problem przeszukiwania

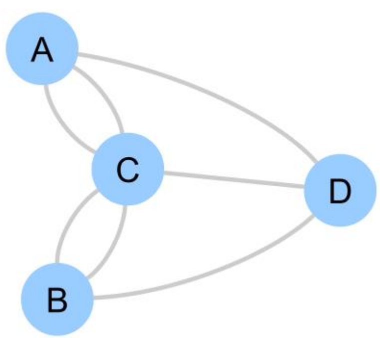
Algorytm Fleury'ego

Złożoność obliczeniowa algorytmu Fleury'ego ma dwie składowe: złożoność przejścia przez graf (O(|E|)) + złożoność algorytmu wyszukiwania mostów. Ze względu na konieczność wyszukiwania mostów jest to dość nieefektywna metoda.

- Złożoność algorytmu Fleury'ego z wyszukiwaniem mostów metodą Tarjana: $O(|E|^2)$
- Złożoność algorytmu Fleury'ego z wyszukiwaniem mostów metodą Thorupa: $O(|E|(\log|E|)^3\log\log|E|)$

Algorytm Hierholzera

Złożoność obliczeniowa: O(|E|)



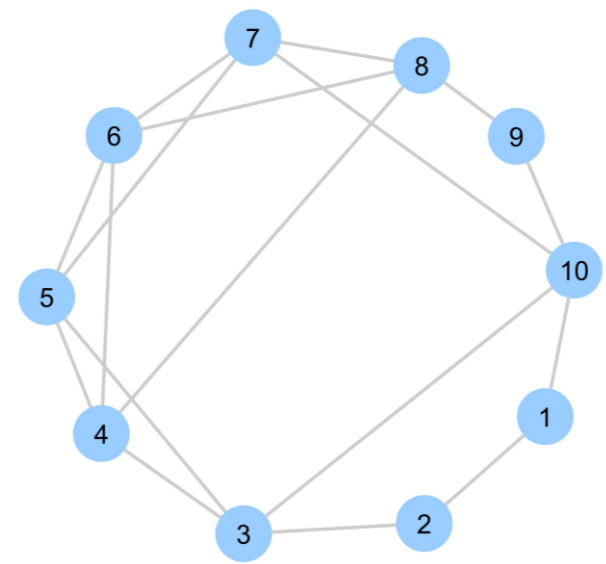
Problem decyzyjny

Jaka jest złożoność obliczeniowa stwierdzenia czy w grafie istnieje cykl Eulera?
Zastanów się jak wyglądałby taki algorytm.

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



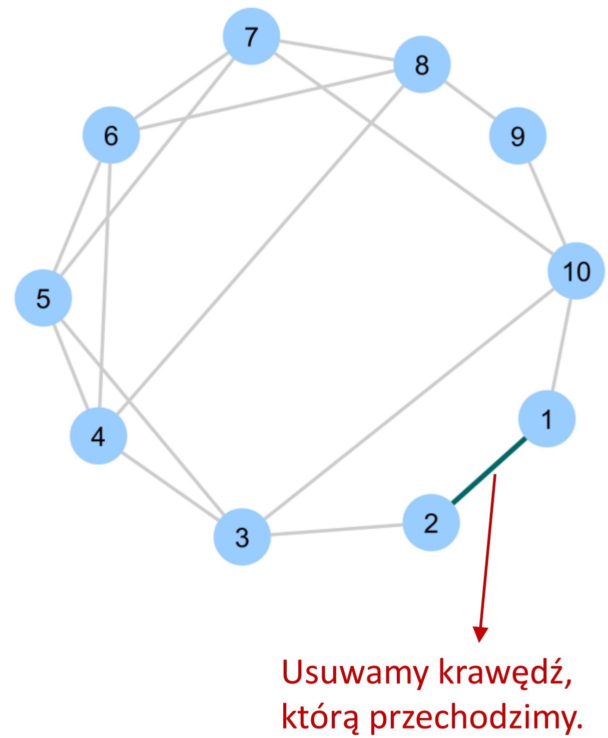
Przetwarzane wierzchołki:
1,

Stos wynikowy:

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



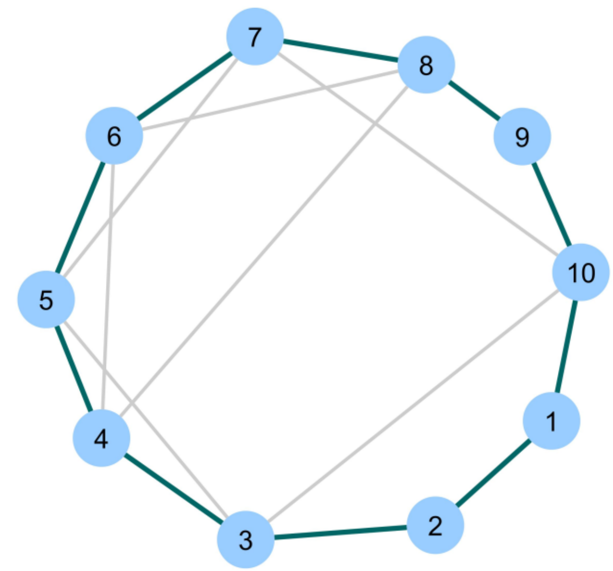
Przetwarzane wierzchołki:
1, 2

Stos wynikowy:

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1,

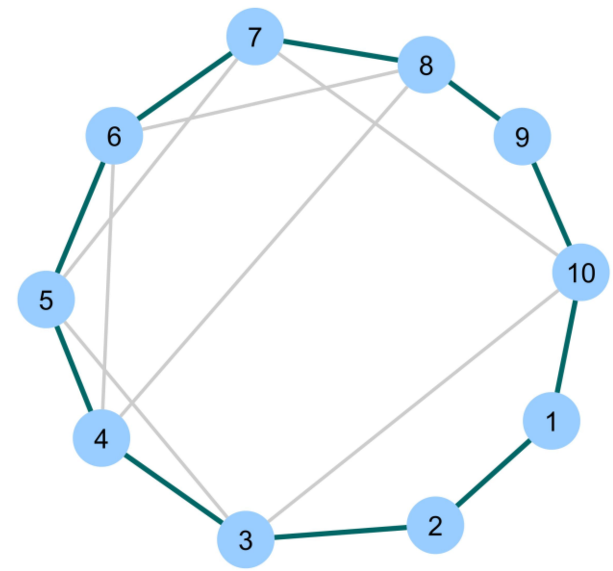
Stos wynikowy:

Brak następników. Zaczynamy odkładać wierzchołki na stos wynikowy.

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ~~1,~~

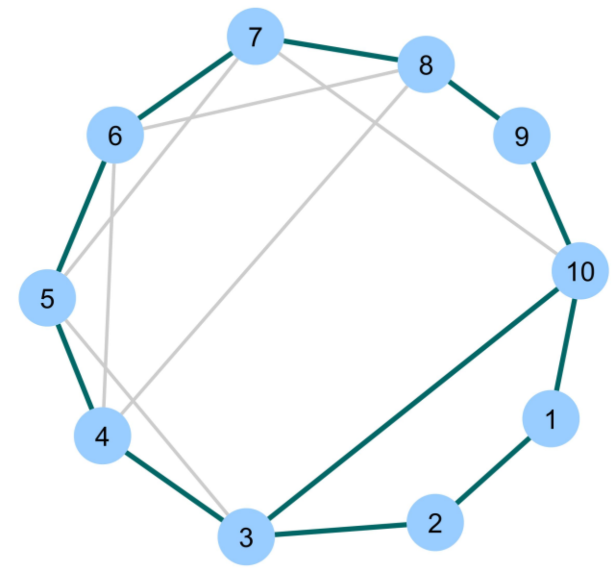
Stos wynikowy:
1,

Brak następników. Zaczynamy odkładać wierzchołki na stos wynikowy.

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



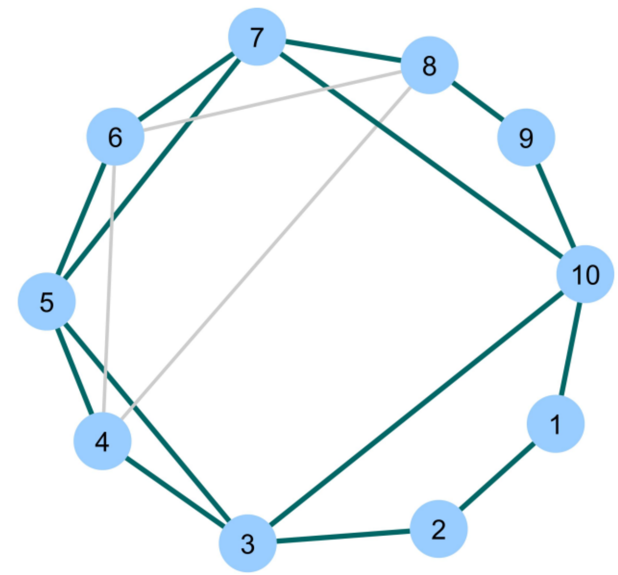
Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ~~1,~~ 3,

Stos wynikowy:
1,

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ~~1~~, 3, 5, 7, 10,

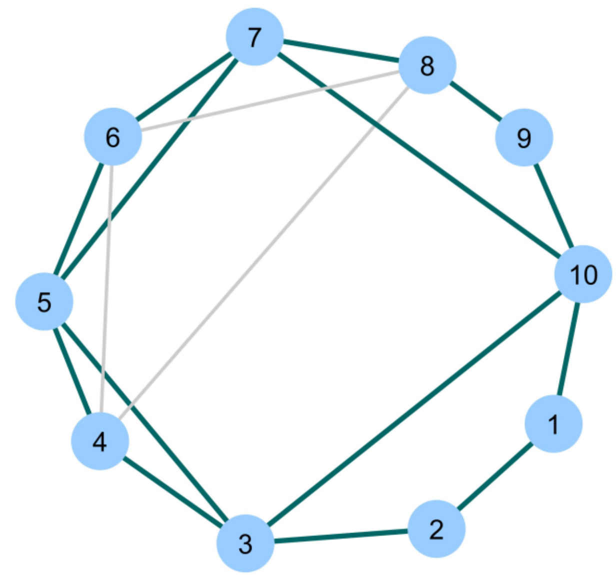
Stos wynikowy:
1,

Brak następników. Zaczynamy odkładać wierzchołki na stos wynikowy.

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, ~~9~~, ~~10~~, ~~1~~, ~~3~~, ~~5~~, ~~7~~, ~~10~~,

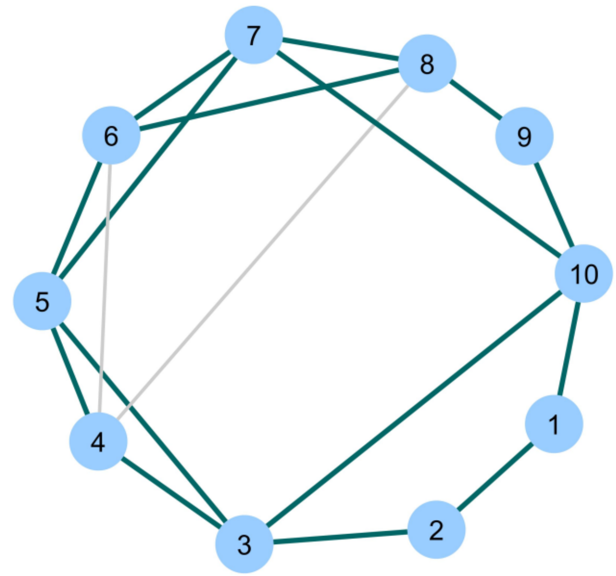
Stos wynikowy:
1, 10, 7, 5, 3, 10, 9,

Brak następników. Zaczynamy odkładać wierzchołki na stos wynikowy.

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, ~~9~~, ~~10~~, ~~1~~, ~~3~~, ~~5~~, ~~7~~, ~~10~~, 6,

Stos wynikowy:
1, 10, 7, 5, 3, 10, 9,

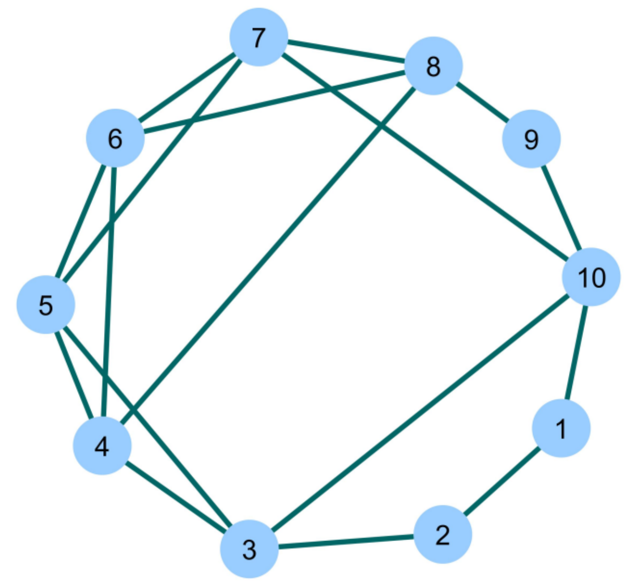
Wierzchołek 8 ma jeszcze następników,
więc przechodzimy do nich.

Poszukiwanie ścieżki/cyklu Eulera: algorytm

Problem:
Znajdź cykl Eulera w grafie

Algorytm:

```
DFS_Euler(v)
   Dla każdego sąsiada(następnika) u wierzchołka v
      usuń krawędź v-u w grafie (w obie strony)
      DFS_Euler(u)
   Odłóż v na stos wynikowy
```



Przetwarzane wierzchołki:
1, 2, 3, 4, 5, 6, 7, 8, ~~9, 10, 1, 3, 5, 7, 10~~, 6, 4, 8

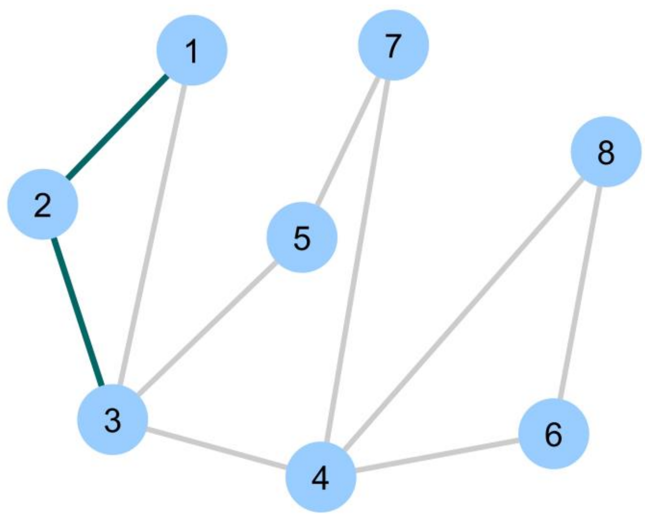
Stos wynikowy:
1, 10, 7, 5, 3, 10, 9, 8, 4, 6, 8, 7, 6, 5, 4, 3, 2, 1

Brak następników. Odkładamy wierzchołki na stos wynikowy.

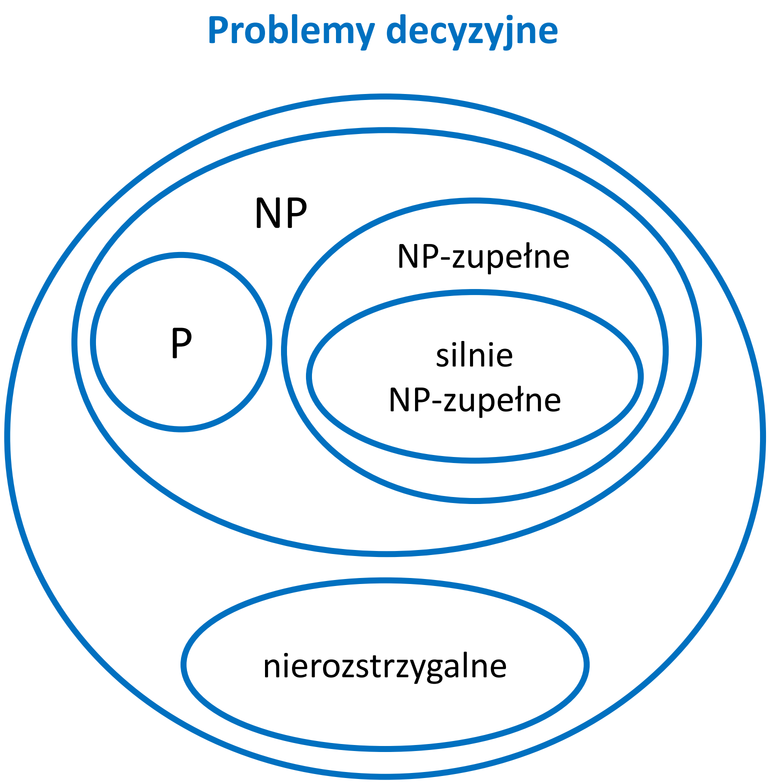
Algorytm Fleury'ego

przechodzi kolejne krawędzie grafu i na każdym kroku sprawdza, czy do kolejnego następnika prowadzi krawędź która jest mostem. Krawędź-most jest wybierana tylko jeśli nie ma innej krawędzi.

Most jest to krawędź w grafie, która po usunięciu zwiększa liczbę spójnych składowych grafu.



Po przejściu wierzchołków 1-> 2 -> 3 następnie wybranym wierzchołkiem będzie 4 lub 5. Nie możemy iść do wierzchołka 3, gdyż krawędź 1-3 jest mostem (po jej przejściu nie wrócimy już do pozostałej nieodwiedzonej części grafu).



Klasa problemów P (*deterministic polynomial*) zawiera problemy, które DTM (deterministyczna maszyna Turinga) rozwiązuje w czasie wielomianowym względem rozmiaru danych wejściowych.

Klasa problemów NP (*nondeterministic polynomial*) zawiera problemy, które NDTM (niedeterministyczna maszyna Turinga) rozwiązuje w czasie wielomianowym względem rozmiaru danych wejściowych. Poprawność rozwiązania problemu z klasy NP może być sprawdzona w czasie wielomianowym.

Klasa problemów NP-zupełnych (*NP-complete*) zawiera problemy z klasy NP, do których w czasie wielomianowym da się przetransformować dowolny inny problem z klasy NP. Problem NP-zupełny to problem, dla którego nie znaleziono algorytmu, który rozwiąże dowolną instancję tego problemu w czasie wielomianowym.
Przykłady: problem plecakowy, problem sumy podzbioru, problem podziału zbioru liczb

Klasa problemów silnie NP-zupełnych (*strongly NP-complete*) zawiera nieliczbowe problemy NP-zupełne lub takie liczbowe problemy NP-zupełne, które nawet przy ograniczeniu maksymalnej wartości występujących w nim liczb pozostają NP-zupełne.
Przykłady: problem cyklu Hamiltona, problem trójpodziału, problem komiwojażera

Skąd wiadomo, że problem decyzyjny jest NP-zupełny?

Można to pokazać przeprowadzając dowód NP-zupełności poprzez transformację wielomianową.

Co jest potrzebne do dowodu?

- Nasz nowy problem A (dla którego chcemy dowieść, że jest NP-zupełny)
- Znany problem B, o którym wiadomo, że jest NP-zupełny

Dowód składa się z dwóch kroków:

Krok 1. Pokazujemy, że problem A należy do klasy NP.

Krok 2. Wykonujemy transformację wielomianową znanego problemu B do naszego nowego problemu A. Zapisujemy to tak: B α A.

Transformacja wielomianowa jest to funkcja f: B$\rightarrow$A, która spełnia warunki:

a) dla każdej instancji problemu B odpowiedzią (rozwiązaniem) jest TAK wtedy i tylko wtedy gdy dla każdej instancji problemu A odpowiedzią jest TAK;
b) czas obliczania funkcji f przez DTM dla każdej instancji problemu B jest ograniczony od góry przez wielomian N(B).

Podstawowe **techniki transformacji** wielomianowej:

1. Ograniczanie
2. Lokalna zamiana
3. Projektowanie części składowych

Klasa złożoności problemu vs złożoność algorytmu

Problem należy do klasy złożoności.

Klasa złożoności odpowiada trudności problemu.

Klasą złożoności problemu decyzyjnego/optymalizacyjnego może być klasa NP, P, NP-zupełna/NP-trudna, silnie NP-zupełna/silnie NP-trudna,...

Algorytm ma złożoność obliczeniową.

Złożoność obliczeniowa odpowiada szacunkowej liczbie operacji podstawowych wykonywanych przez algorytm.

Złożoność algorytmu może być wielomianowa, pseudowielomianowa, logarytmiczna, liniowo-logarytmiczna, wykładnicza.

Jaki jest związek między klasą złożoności problemu a złożonością obliczeniową algorytmu, który rozwiązuje ten problem?

Jeśli problem jest w klasie … → istnieje rozwiązujący go algorytm o złożoności …

Klasa problemu	**Znalezienie rozwiązania**
P	Algorytmy wielomianowe: $O(n^x)$
silnie NP-zupełne	Algorytmy wykładnicze: $O(x^n)$
NP-zupełne	Algorytmy pseudowielomianowe lub wykładnicze: $O(n \cdot x)$, $O(x^n)$
NP	Algorytmy wielomianowe lub wykładnicze: $O(n^x)$, $O(x^n)$

Ścieżka Hamiltona vs ścieżka Eulera

Problem przeszukiwania	**Problem decyzyjny**
Problem ścieżki Hamiltona	
Klasa złożoności problemu Jest to problem trudny obliczeniowo. Wersja przeszukiwania należy do klasy problemów **silnie NP-trudnych**.	Klasa złożoności problemu Jest to problem trudny obliczeniowo. Wersja decyzyjna jest w klasie problemów **silnie NP-zupełnych**.
Rozwiązanie problemu ścieżki Hamiltona	
Złożoność algorytmu: wykładnicza	Złożoność algorytmu: wykładnicza
Problem ścieżki Eulera	
Klasa złożoności problemu Jest to problem łatwy obliczeniowo. Wersja przeszukiwania należy do **klasy P**.	Klasa złożoności problemu Jest to problem łatwy obliczeniowo. Wersja decyzyjna jest w **klasie problemów P**.
Rozwiązanie problemu ścieżki Eulera	
Złożoność algorytmu: wielomianowa	Złożoność algorytmu: wielomianowa