Algorytmy Grafowe

dr hab. Bożena Woźna-Szcześniak, prof. UJD

Uniwersytet Jana Długosza w Częstochowie

b.wozna@ujd.edu.pl

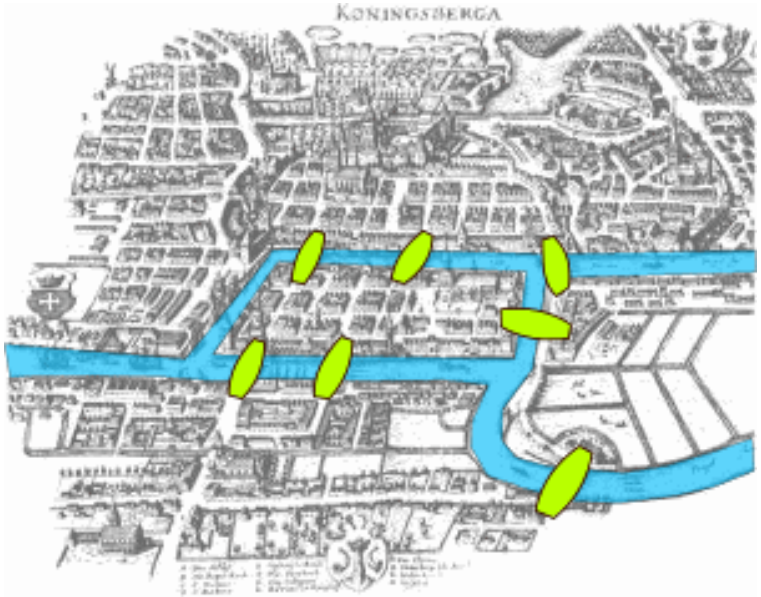
Wykład 9 i 10

Spis treści

Spis treści

Leonhard Euler i Mosty królewieckie I

Przez Królewiec przepływała rzeka Pregole, w której rozwidleniach znajdowały się dwie wyspy. Ponad rzeką przerzucono siedem mostów, z których jeden łączył obie wyspy, a pozostałe mosty łączyły wyspy z brzegami rzeki.



Źródło: `https://commons.wikimedia.org/w/index.php?curid=112920`

Leonhard Euler i Mosty królewieckie II

Pytanie: czy można przejść przez każdy z siedmiu mostów dokładnie jeden raz tak, aby powrócić do punktu wyjścia ?

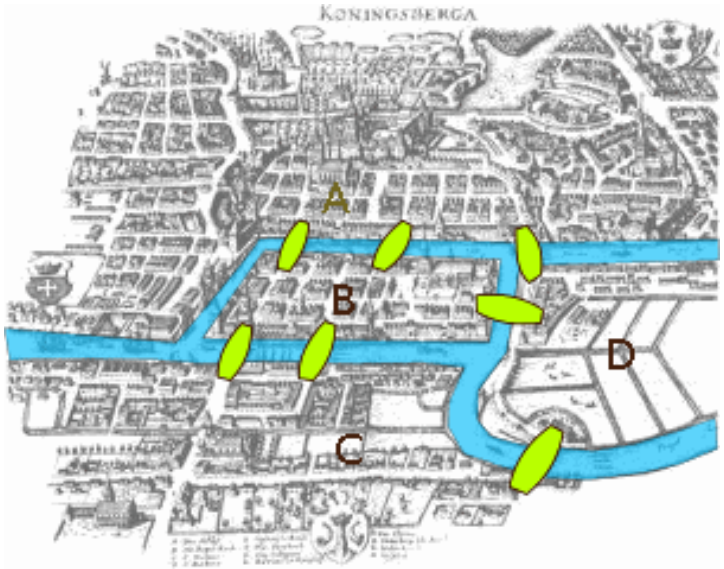
Wybitny szwajcarski matematyk Leonhard Euler (1707-1783) zainteresował się problemem mostów Królewieckich około 1735 roku i opublikował rozwiązanie („Solutio problematis ad geometriam situs pertinentis”) w 1741 roku.

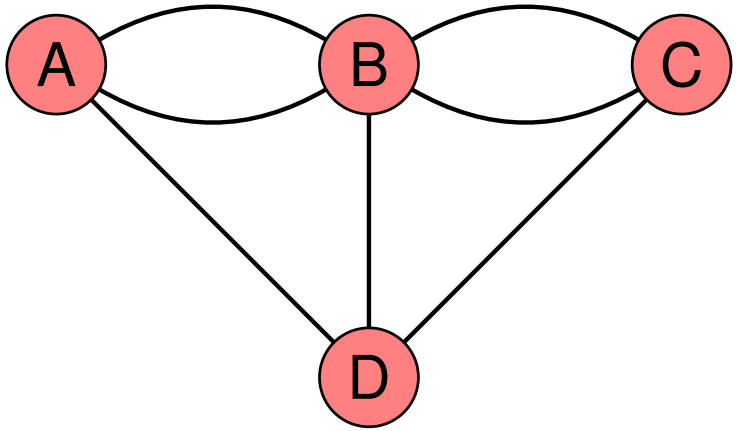
Leonhard Euler (1707 – 1783)

Leonhard Euler i Mosty królewieckie III

Intuicja Eulera: fizyczna mapa nie ma znaczenia. Liczy się tylko lista regionów połączonych mostami.

Odpowiedź: problem mostów królewieckich równoważny jest pytaniu, czy graf pokazany na rysunku po prawej stronie jest grafem eulerowskim.





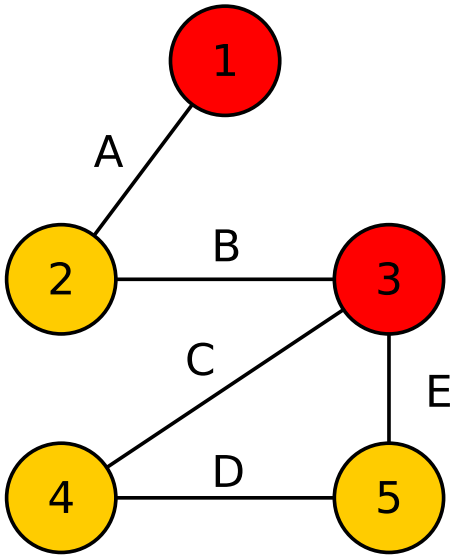
Spis treści

Ścieżka Eulera

- Ścieżka Eulera w grafie (skierowanym), to ścieżka (droga) prosta, która zawiera każdą krawędź grafu dokładnie jeden raz.
- Warunkiem istnienia ścieżki są:
  1. spójność grafu.
  2. dla grafu skierowanego należy sprawdzić, czy dla każdego wierzchołka, za wyjątkiem dwóch, stopień wyjściowy jest równy stopniu wejściowemu.
  3. dla grafu nieskierowanego z każdego wierzchołka, za wyjątkiem dwóch, musi wychodzić parzysta liczba krawędzi.
- Graf, który posiada ścieżkę Eulera nazywamy grafem półeulerowskim

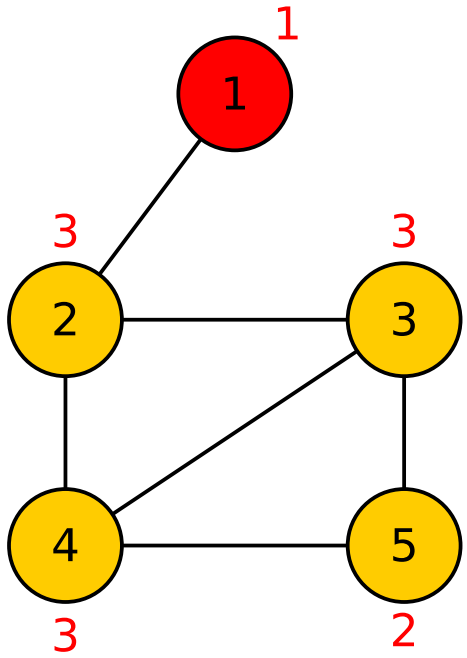
Ścieżka Eulera - przykład

Graf ze ścieżką Eulera:



Ścieżka: 1-> ABCDE -> 3

Graf bez ścieżki Eulera:



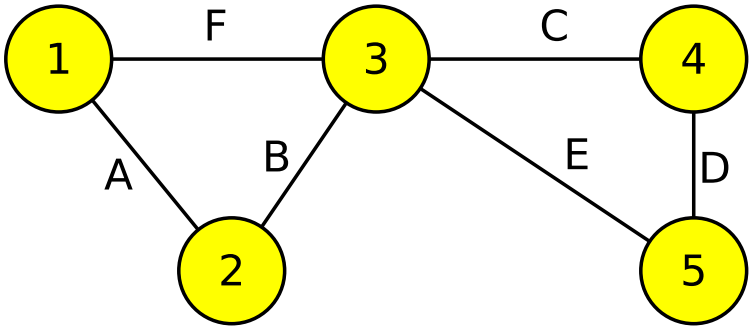
Nie spełniony warunek 3 istnienia ścieżki Eulera

Cykl Eulera

- Cykl Eulera to taki cykl w grafie, który zawiera każdą krawędź grafu dokładnie jeden raz.
- Warunkiem istnienia cyklu są:
  - spójność grafu.
  - dla grafu skierowanego należy sprawdzić, czy dla każdego wierzchołka stopień wyjściowy jest równy stopniu wejściowemu.
  - dla grafu nieskierowanego z każdego wierzchołka musi wychodzić parzysta liczba krawędzi.
- Graf, który posiada cykl Eulera nazywamy grafem eulerowskim

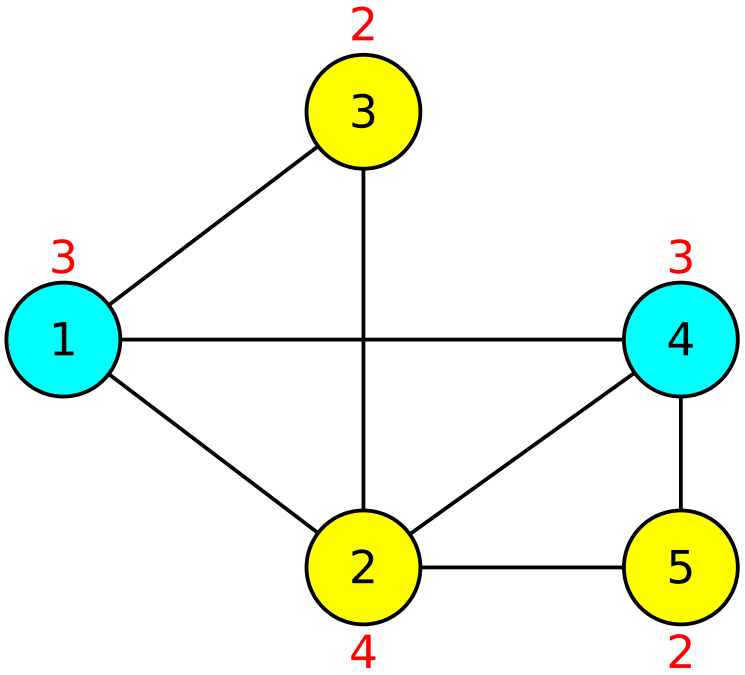
Cykl Eulera - przykład

Graf z cyklem Eulera:



Cykl 1: 1-> ABCDEF -> 1
Cykl 2: 1-> ABEDCF -> 1
Cykl 3: 1-> FCDEBA -> 1
Cykl 4: 1-> FEDCBA -> 1

Graf bez cyklu Eulera:



Nie spełniony warunek 3 istnienia cyklu Eulera

Twierdzenie Eulera, 1736

"This question is so banal, but seemed to me worthy of attention in that neither geometry, nor algebra, nor even the art of counting was sufficient to solve it."

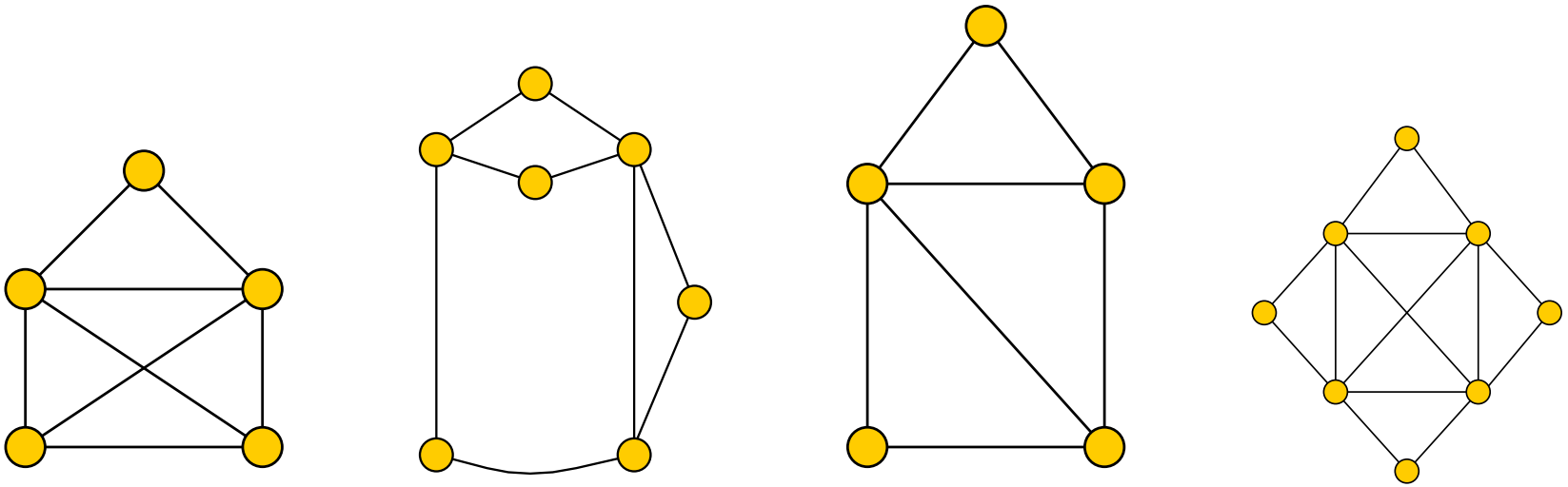
(Leonhard Euler, Marzec 1736)

Twierdzenie

Spójny graf G (nieskierowany) ma cykl Eulera wtedy i tylko wtedy, gdy stopień każdego wierzchołka w G jest parzysty.

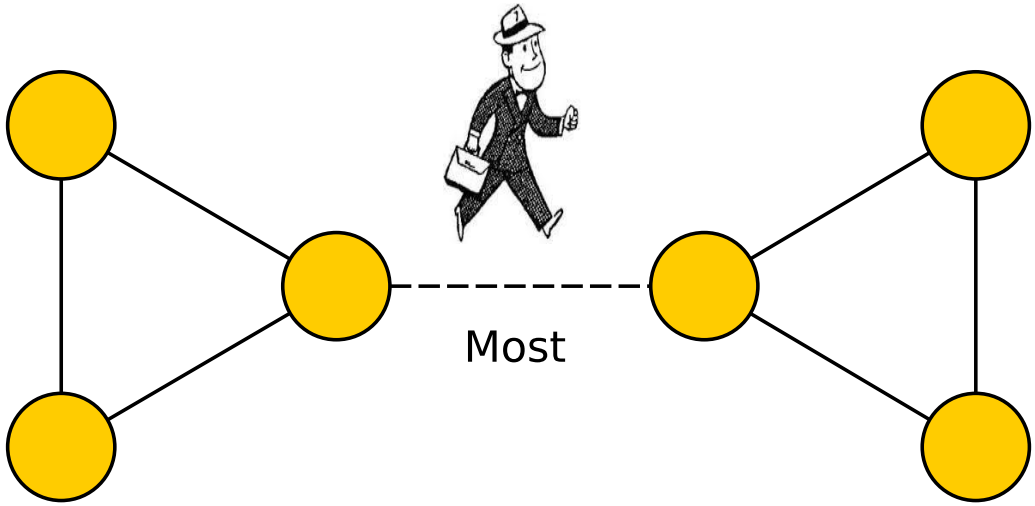
Zagadki

- Sprawdź, w którym z poniższych grafów istnieje cykl lub droga Eulera. Jeśli w grafie istnieje cykl lub droga Eulera, to można ją narysować nie odrywając ołówka od papieru.



Most

Mostem nazywamy taką krawędź grafu, której usunięcie zwiększa liczbę spójnych składowych tego grafu.



Wyznaczanie cyklu Eulera

- Do wyznaczania cyklu Eulera służy algorytm **Fleury'ego**:
  - działa zarówno dla grafów skierowanych jak i nieskierowanych.
  - jest rekurencyjny.
  - zakłada, że graf jest Eulerowski.
- Algorytm Fleury'ego opiera się na prostej zasadzie: aby znaleźć cykl Eulera lub ścieżkę Eulera, mosty są ostatnimi krawędziami, które należy przejść.

Algorytm Fleury'ego

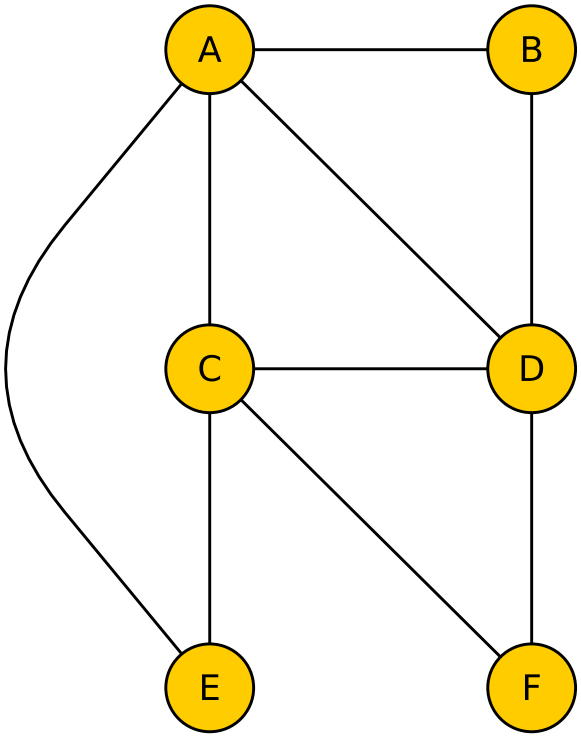
- **Warunek wstępny:**
  - Wejściowy graf jest grafem eulerowskim, czyli:
    - jest spójny
    - w przypadku poszukiwania ścież Eulera - posiada co najwyżej dwa wierzchołki o nieparzystym stopniu
    - w przypadku poszukiwania cyklu Eulera - wszytkie wierzchołki muszą mieć parzysty stopień
- **Warunek startowy:**
  - W przypadku cyklu: wybierz dowolny wierzchołek
  - W przypadku ścieżki: wybierz jeden z wierzchołków nieparzystych

Algorytm Fleury'ego

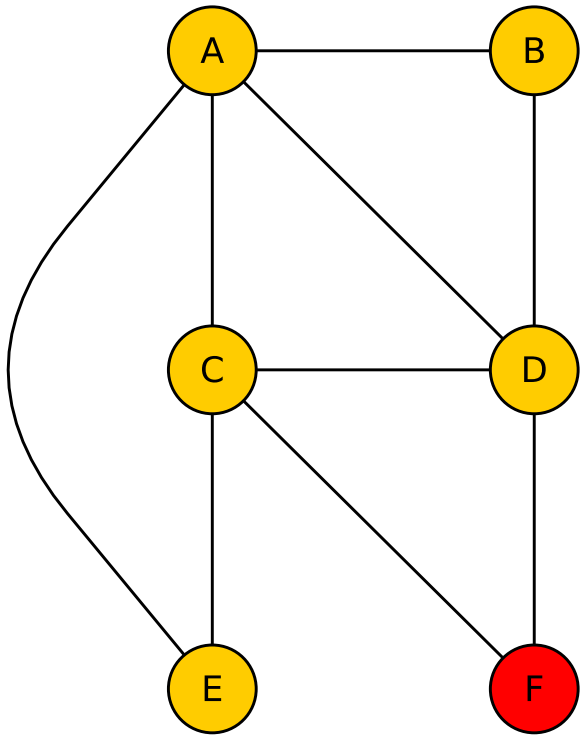
- **Kroki pośrednie**: Na każdym kroku, jeśli jest wybór, nie wybieraj **mostu** występujacego w części grafu, która nie została jeszcze odwiedzona. Jednakże, jeśli jest tylko jeden wybór, to weź go.
- **Warunek Końcowy**: Kiedy nie można przechodzić już dalej, cykl (ścieżka) jest kompletna. [W przypadku cyklu, wracamy do wierzchołka wyjściowego; w przypadku ścieżki dochodzimy do drugiego wierzchołka o nieparzystym stopniu.]

Algorytm Fleury'ego

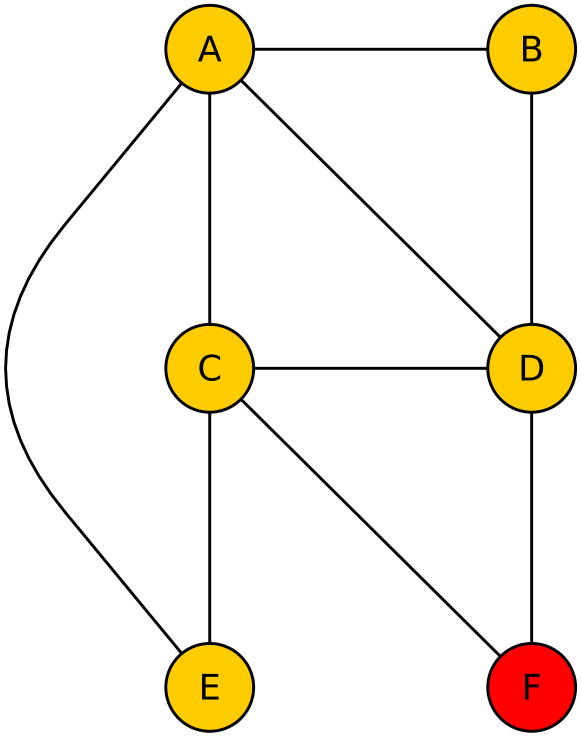
Graf eulerowski:



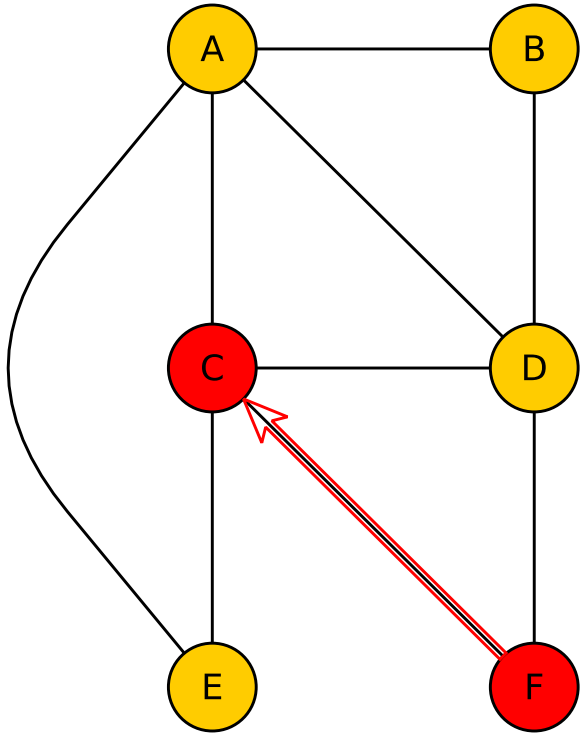
Wybieramy wierzchołek F jak startowy



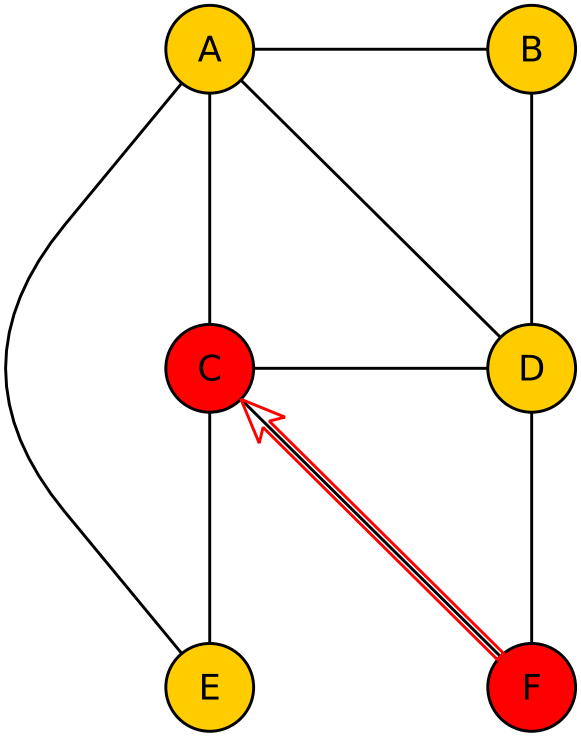
Algorytm Fleury'ego



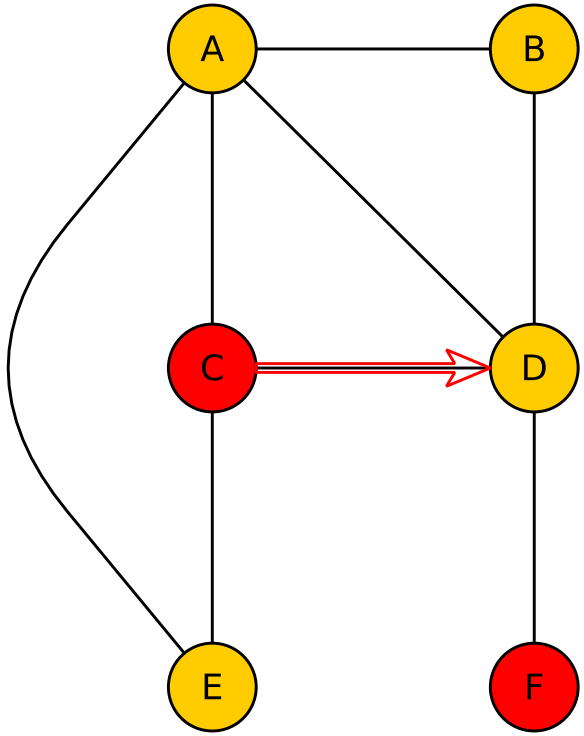
Idziemy z F do C



Algorytm Fleury'ego

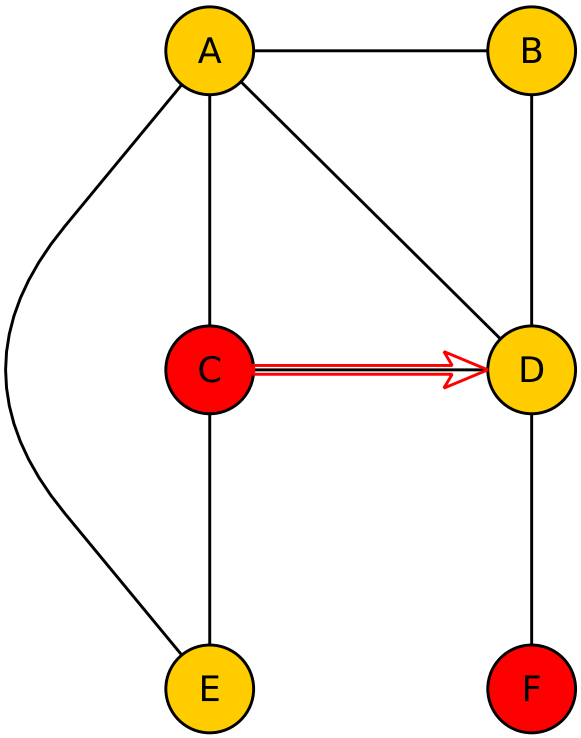


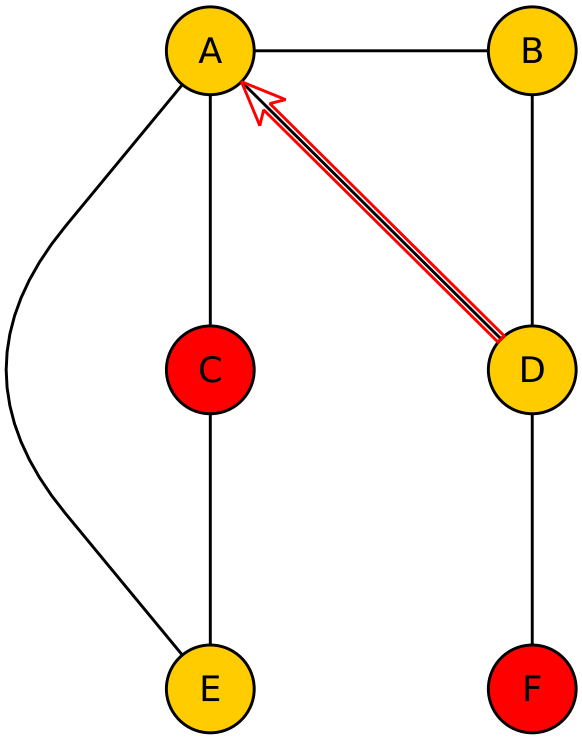
Idziemy z C do D



Algorytm Fleury'ego

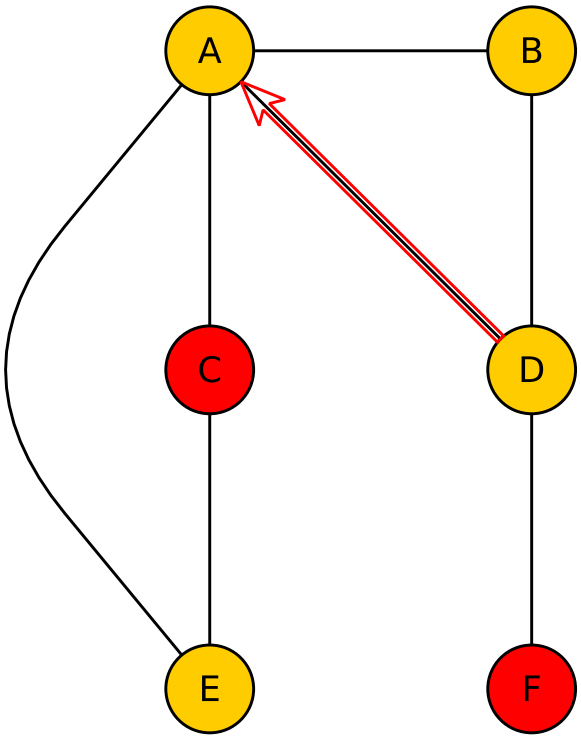
Idziemy z D do A. (Można również iść z D do B, ale nie można z D do F, bo DF jest mostem)

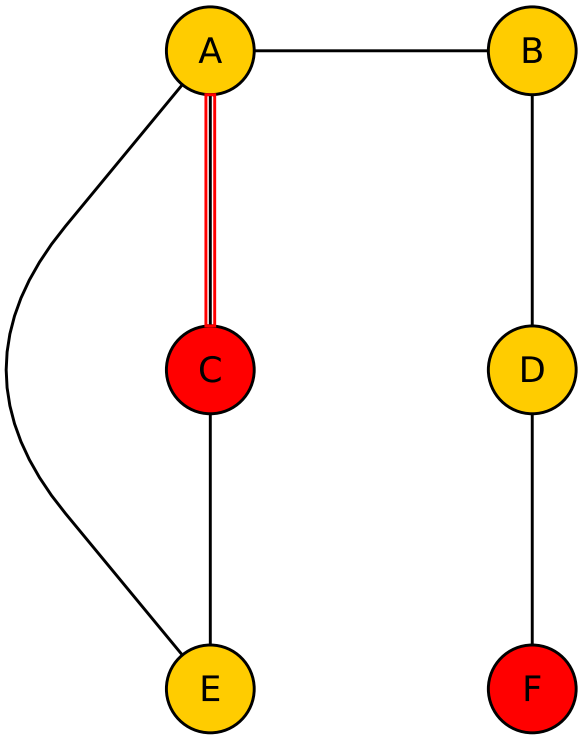




Algorytm Fleury'ego

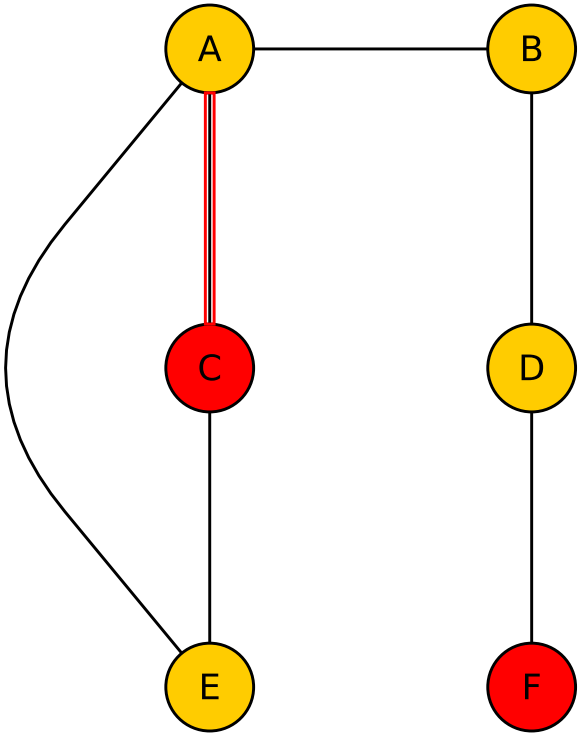
Idziemy z A do C. (Można również iść z A do E, ale nie można z A do B, bo AB jest mostem)

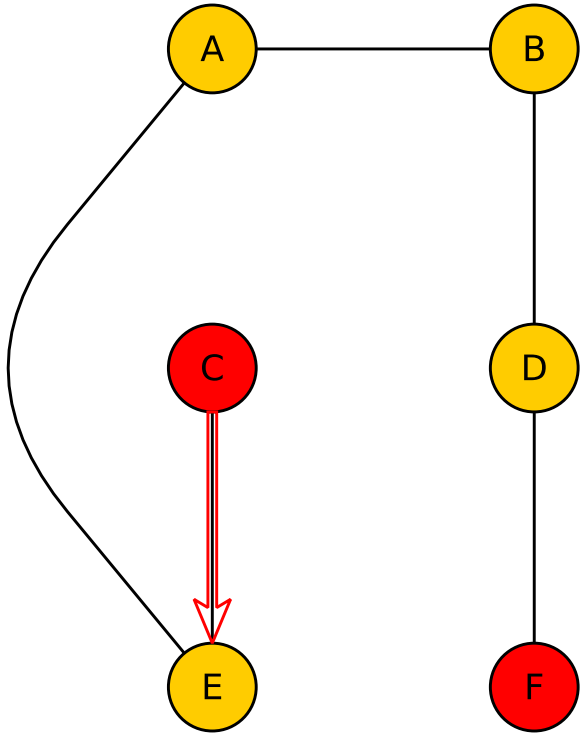




Algorytm Fleury'ego

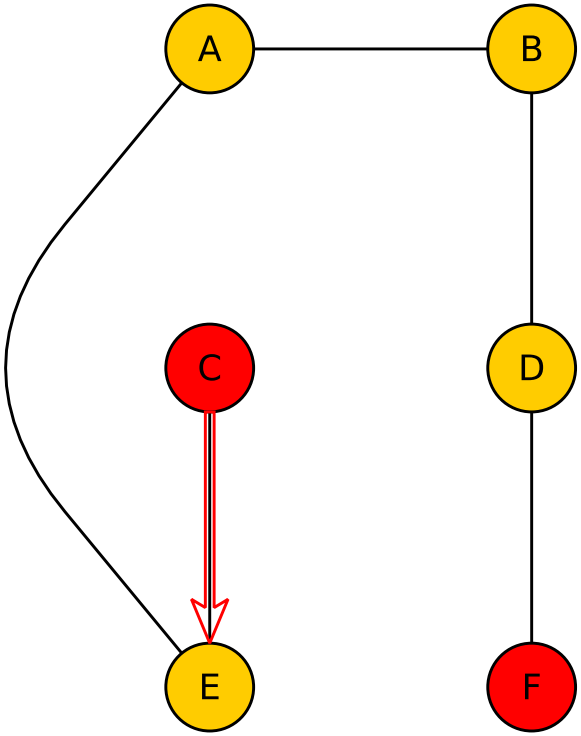
Idziemy z C do E. Nie ma wyboru.

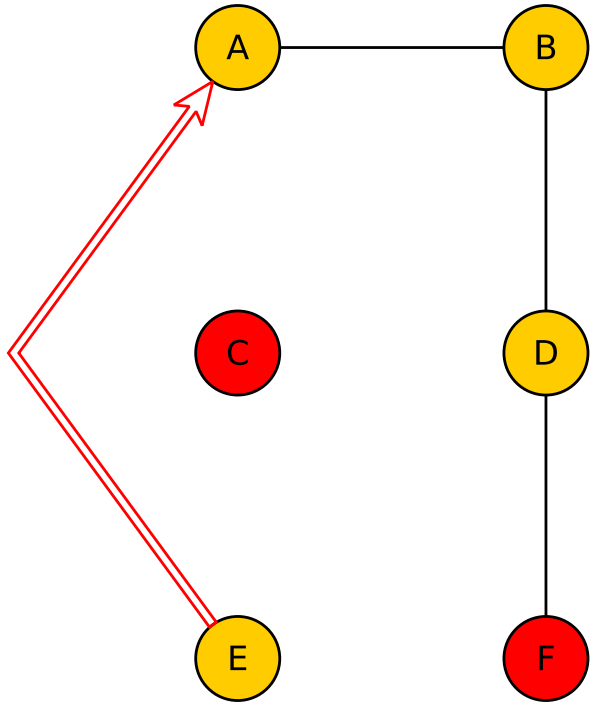




Algorytm Fleury'ego

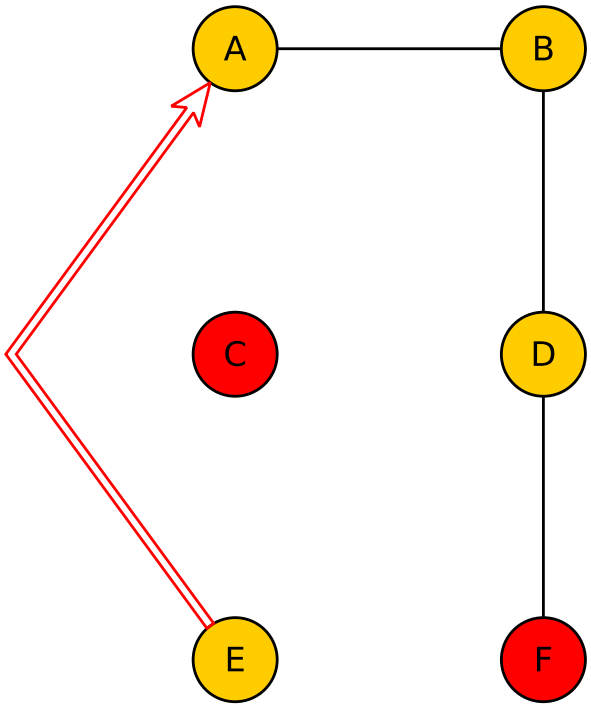
Idziemy z E do A. Nie ma wyboru.

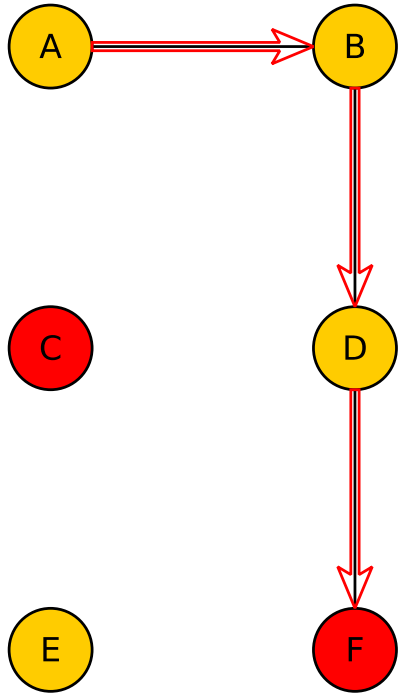




Algorytm Fleury'ego

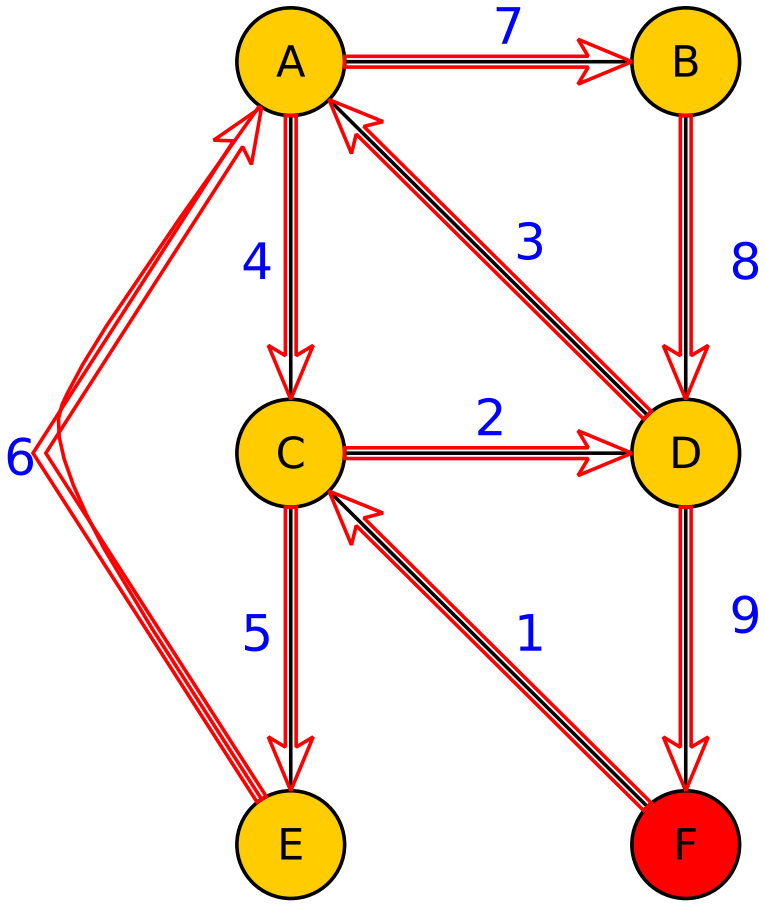
Idziemy z A do B, potem z B do D i na koniec z D do F. Jednoznaczna ściezka.





Algorytm Fleury'ego

Kolejne kroki algorytmu - podsumowanie.



Algorytm Fleury'ego - Java I

```
/* Zakładamy, że:
   (1) Graf jest reprezentowany przez
       macierz sąsiedztwa
   (2) w grafie sa 2 lub 0 wierzcholki
       o nieparzystym stopniu   */
public void printEulerTour() {
   //Znajdź wierzchołek z nieparzystym stopniem
   //Dymyślnie startujemy od wierzchołka z indeksem 0
   int u = 0;
   for (int row = 0; row < V; ++row) {
        int odd = 0;
        for (int col = 0; col < V; ++col) {
            if (adj[row][col] == true) odd++;
        }
```

Algorytm Fleury'ego - Java II

```
        if (odd % 2 == 1) {
            u = row; break;
        }
    }
    // funkcja pomocnicza drukująca ścieżkę/cykl.
    // poczynajac od wierzchołka u
    this.printEulerUtil(u);
}

void printEulerUtil(int u) {
    for (int v = 0; v < V; v++) {
        //jeśli krawedx u-v nie jest usunieta
        //i jest "poprawna"
        if (adj[u][v] == true &&
            this.isValidNextEdge(u,v) == true) {
```

Algorytm Fleury'ego - Java III

```
                System.out.println(u + "-" + v) ;
                this.removeEdge(u,v);
                this.printEulerUtil(v);
            }
        }
}

boolean isValidNextEdge(int u, int v)
{
    // liczba sasiadow wierzcholka u
    int count = 0;
    for (int i = 0; i < V; ++i) {
         if (adj[u][i] == true) count++;
    }
    //krawedz jest poprawna jesli zachodzi:
```

Algorytm Fleury'ego - Java IV

```
    //(1) v jest jedynym sasiadem wierzcholka u
    if (adj[u][v] == true && count == 1)
       return true;
    //Jesli u ma wielu sasiadow, to należy
    //sprawdzić, czy u-v jest mostem.
    //liczba wierzcholkow osiagalnych z u
    int count1 = this.DFSCount(u);
    this.removeEdge(u,v);
    int count2 = this.DFSCount(u);
    this.addEdge(u,v);
    if (count1 > count2)
       return false;  //u-v jest mosetem
    else
       return true;
}
```

Algorytm Fleury'ego - Java V

```
// Funkcja bazujaca na algorytmie DFS
// zliczajaca wierzcholki osiagalne z wierzchołka a
int DFSCount(int a) {
  boolean[] visited = new boolean[V];
  for (int k = 0; k < V; ++k) visited[k] = false;
  Stack<Integer> s = new Stack<Integer>();
  visited[a] = true;
  s.push(a);
  int count = 1;
  while (!s.empty()) {
     int b = getUnVisitedVertex(s.peek(), visited);
     if (b == -1) {
        s.pop();
     } else {
```

Algorytm Fleury'ego - Java VI

```
            visited[b] = true;
            count++;
            s.push(b);
        }
    }
    return count;
}
```

Agorytm Fleury'ego - wykonanie I

```
bwozna@vostro:~/Graf-Fleury$ java Graph Koperta.in
Wierzchołków: 5
Krawędzi: 8
Krawedzie grafu:
0-1 0-2
1-0 1-2 1-3 1-4
2-0 2-1 2-3 2-4
3-1 3-2 3-4
4-1 4-2 4-3

Macierz sąsiedztwa:
false true true false false
true false true true true
true true false true true
```

Agorytm Fleury'ego - wykonanie II

```
false true true false true
false true true true false

Ścieżka startuje z:3
3-1 1-0 0-2 2-1 1-4
4-2 2-3 3-4

bwozna@vostro:~/Graf-Fleury$ java Graph Euler0.in
Wierzchołków: 6
Krawędzi: 9
Krawedzie grafu:
0-1 0-4
1-0 1-2 1-3 1-4
2-1 2-3
3-1 3-2 3-4 3-5
```

Agorytm Fleury'ego - wykonanie III

```
4-0 4-1 4-3 4-5
5-3 5-4

Macierz sąsiedztwa:
false true false false true false
true false true true true false
false true false true false false
false true true false true true
true true false true false true
false false false true true false

Ścieżka startuje z:0
0-1 1-2 2-3 3-1
1-4 4-3 3-5 5-4 4-0
```

Pseudokod algorytmu Fleury'ego I

Pseudokod na podstawie opracowania:
https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/

```
/*
 Funkcja drukuje ścieżkę/cykl Eulera.
 Najpierw znajdowany jest nieparzysty wierzchołek
 (jeśli istnieje), a następnie wywoływana jest
 pomocnicza funkcja printEulerUtil(), aby wydrukować
 ścieżkę. Zakłada się, że graf G=(V,E)
 reprezentowany jest przez listy sąsiedztwa,
 rozmiar V jest n, oraz że są 2 lub 0 wierzchołków
 o stopniu nieparzystym.
 adj[i] - lista sąsiedztwa dla wierzchołka i.
*/
```

Pseudokod algorytmu Fleury'ego II

```
printEulerTour(Graph G)
{
  // Znajdź wierzchołek z nieparzystym stopniem
  // Dymyślnie startujemy od wierzchołka
  // z indeksem 0.
  vertex u = 0;
  for (each v in V)
      if ( length of adj[v] is odd) then
          u = v;
          break;
      endif
  endfor
  printEulerUtil(G,u);
}
```

Pseudokod algorytmu Fleury'ego III

```
//Drukuje ścieżkę/cykl Eulera poczynając
//od wierzchołka u
printEulerUtil(Graph G, vertex u)
{
  for (each v in  adj[u]) do
  //jeśli krawędź u-v nie jest usunięta i jest
  //"poprawną" następna krawędzią.
      if (v != -1 and isValidNextEdge(G,u,v)) then
          print(u,"-",v) ;
          removeEdge(G,u,v);
          printEulerUtil(G,v);
      endif
  endfor
}
```

Pseudokod algorytmu Fleury'ego IV

```
// Funkcja usuwa krawędź u-v z grafu.
// Usuwa krawędź, zastępując wartość
// sąsiedniego wierzchołka wartością -1.
removeEdge(Graph G, vertex u, vertex v)
{
//Znajdź v na liście sąsiedztwa u i zastąp go -1
  for (each i in  adj[u]) do
     if (i==v) then i = -1;
  endfor
//Znajdź u na liście sąsiedztwa v i zastąp go -1
  for (each i in  adj[v]) do
     if (i==u) then i = -1;
  endfor
}
```

Pseudokod algorytmu Fleury'ego V

```
//Funkcja sprawdzająca, czy krawędź u-v może
//być uważana za następną krawędź
//w cyklu/ścieżce Eulera
bool isValidNextEdge(Graph G, vertex u, vertex v)
{
  //liczba sąsiadów wierzchołka u
  int count = 0;
  for (each i in adj[u]) do
    if (i != -1) then
       count=count+1;
  endif
  endfor
  //Krawędź u-v jest poprawna, jeśli zachodzi
  //jeden z następujących przypadków:
  // 1) v jest jedynym sąsiadem wierzchołka u
```

Pseudokod algorytmu Fleury'ego VI

```
if (count == 1) then
  return true;
endif
// 2) Jeśli u ma wielu przyległych sąsiadów, to
// wykonaj następujące kroki, aby sprawdzić,
//czy u-v jest mostem
// 2.a) liczba wierzchołków osiągalnych z u
boolvector visited[n];
for(each v in V) do
    visited[v]=false;
endfor;
count1 = DFSCount(G,u,visited);
// 2.b) Usuń krawędź (u,v), a po jej usunięciu,
// policz liczbę wierzchołków osiągalnych z u
removeEdge(G,u,v);
```

Pseudokod algorytmu Fleury'ego VII

```
  for(each v in V) do
     visited[v]=false;
  endfor;
  count2 = DFSCount(G, u, visited);
  // 2.c) Dodaj krawędź z powrotem do grafu.
  addEdge(u, v);
  // 2.d) Jeśli count1 > count2, to
  // krawędź (u, v) jest mostem
  if  (count1 > count2) then
      return false;
  else
      retunr true;
  endif
}
```

Pseudokod algorytmu Fleury'ego VIII

```
// Funkcja bazująca na algorytmie DFS
// zliczająca wierzchołki osiągalne z wierzchołka v
int DFSCount(Graph G, verex v, boolvector visited)
{
  visited[v] = true;
  int count = 1;
  for (each i in adj[v])
      if (i != -1 and visited[i]==false) then
          count = count+ DFSCount(G,i, visited);
      endif
  endfor
  return count;
}
```

Złożoność czasowa algorytmu Fleury'ego

- Złożoność czasowa powyższej implementacji wynosi $O(|V|^4)$.
  Dlaczego ? Funkcja `printEulerUtil()` działa jak DFS i wywołuje metodę `isValidNextEdge()`, która również wykonuje DFS dwa razy. Złożoność czasowa DFS dla reprezentacji macierzowej wynosi $O(|V|^2)$. Zatem, ogólna złożoność czasowa wynosi $O(|V|^4)$.
- Jeżli zmodyfikujemy powyższą implementację do pracy z grafami reprezentowanymi przez listy sąsiedztwa, to jej złożoność czasowa wyniesie $O((|V| + |E|)^2)$.
  Dlaczego ? Funkcja `printEulerUtil ()` działa jak DFS i wywołuje metodę `isValidNextEdge ()`, która również wykonuje DFS dwa razy. Złożoność czasowa DFS dla reprezentacji listy sąsiedztwa wynosi $O(|V| + |E|)$. Zatem ogólna złożoność czasowa wynosi $O((|V| + |E|)^2)$.

Cykl Eulera - Zastosowania

- Rysowanie/wycinanie figur przy pomocy plotera
- Problem chińskiego listonosza – W roku 1962 chiński matematyk Mei-Ko Kwan sformułował następujący problem:
  Listonosz roznosząc listy musi przejść przez wszystkie ulice w swojej dzielnicy co najmniej jeden raz i wrócić na pocztę. Ponieważ jest człowiekiem leniwym, chciałby mieć jak najkrótszą do przejścia trasę. Znalezienie takiej trasy jest problemem, który nazwano problemem chińskiego listonosza (ang. Chinese postman problem - CPP).