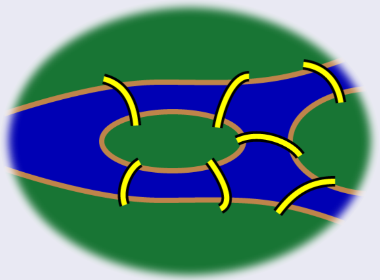
Grafy c.d. — cykl Eulera, algorytm Dijkstry i Bellmana-Forda

zajęcia 6.

Marcin Andrychowicz, Tomasz Kulczyński, Błażej Osiński

Problem mostów królewieckich

Mosty



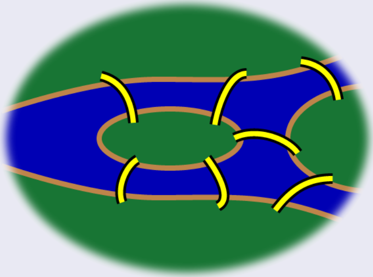
Graf

Cykl Eulera

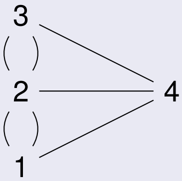
Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie grafu dokładnie raz.

Problem mostów królewieckich

Mosty



Graf

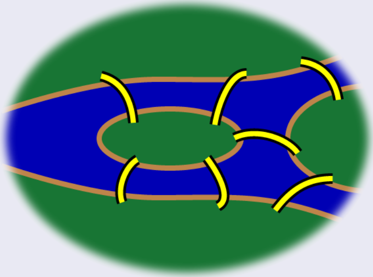


Cykl Eulera

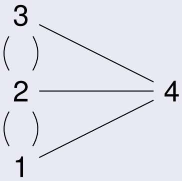
Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie grafu dokładnie raz.

Problem mostów królewieckich

Mosty



Graf

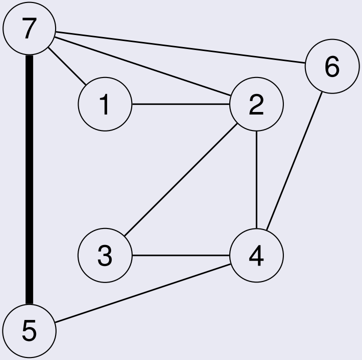


Cykl Eulera

Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie grafu dokładnie raz.

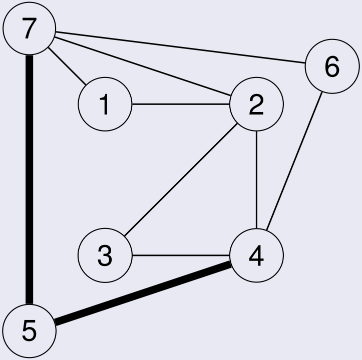
Cykl Eulera

Przykład cyklu Eulera:



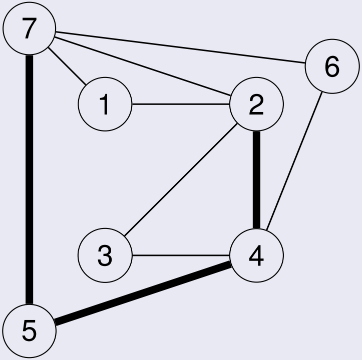
Cykl Eulera

Przykład cyklu Eulera:



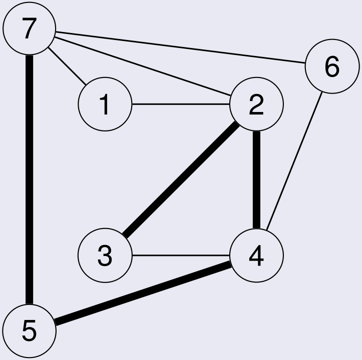
Cykl Eulera

Przykład cyklu Eulera:



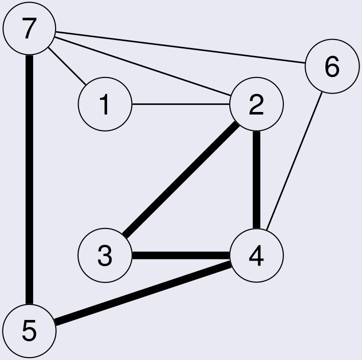
Cykl Eulera

Przykład cyklu Eulera:



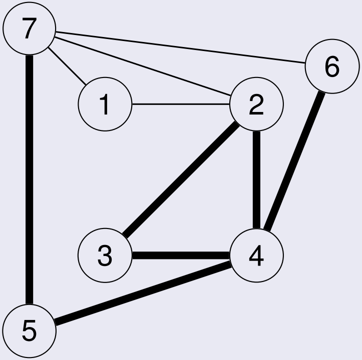
Cykl Eulera

Przykład cyklu Eulera:



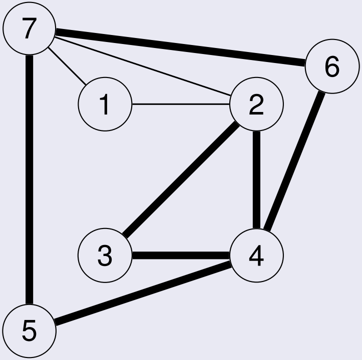
Cykl Eulera

Przykład cyklu Eulera:



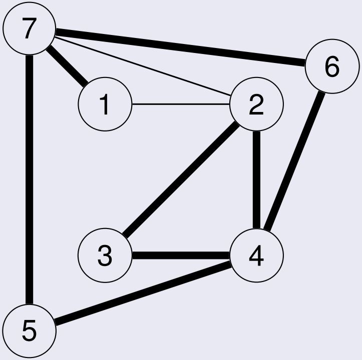
Cykl Eulera

Przykład cyklu Eulera:



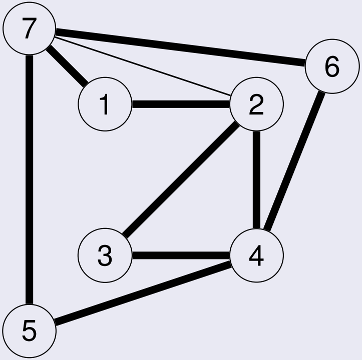
Cykl Eulera

Przykład cyklu Eulera:



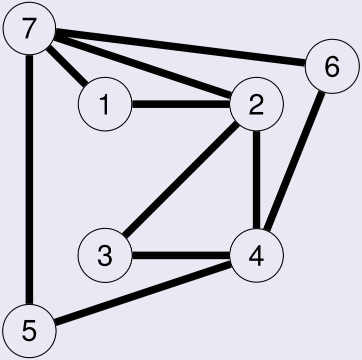
Cykl Eulera

Przykład cyklu Eulera:



Cykl Eulera

Przykład cyklu Eulera:



Cykl Eulera

Pytania

- czy graf posiada cykl Eulera? (czy jest tzw. `eulerowski`)
- jak znaleźć dowolny cykl Eulera?

Cykl Eulera

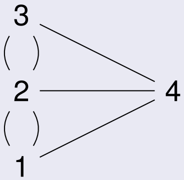
Pytania

- czy graf posiada cykl Eulera? (czy jest tzw. `eulerowski`)
- jak znaleźć dowolny cykl Eulera?

Czy graf posiada cykl Euler?

Graf

Dlaczego poniższy graf nie posiada cyklu Eulera?



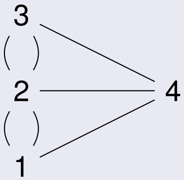
Warunki konieczne:

- parzystość stopni wierzchołków
- spójność

Czy graf posiada cykl Euler?

Graf

Dlaczego poniższy graf nie posiada cyklu Eulera?



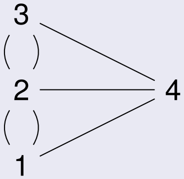
Warunki konieczne:

- parzystość stopni wierzchołków
- spójność

Czy graf posiada cykl Euler?

Graf

Dlaczego poniższy graf nie posiada cyklu Eulera?



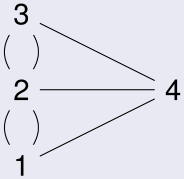
Warunki konieczne:

- parzystość stopni wierzchołków
- spójność

Czy graf posiada cykl Euler?

Graf

Dlaczego poniższy graf nie posiada cyklu Eulera?



Warunki konieczne:

- parzystość stopni wierzchołków
- spójność

Czy graf posiada cykl Eulera?

Podane warunki są dostateczne!

- dowód przez indukcję względem ilości krawędzi
- baza: dla grafu bez krawędzi zachodzi
- krok indukcyjny: zaczynamy z dowolnego wierzchołka v i poruszamy się aż trafimy znów do v ...

Czy graf posiada cykl Eulera?

Podane warunki są dostateczne!

- dowód przez indukcję względem ilości krawędzi
- baza: dla grafu bez krawędzi zachodzi
- krok indukcyjny: zaczynamy z dowolnego wierzchołka v i poruszamy się aż trafimy znów do v ...

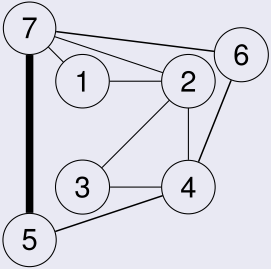
Czy graf posiada cykl Eulera?

Podane warunki są dostateczne!

- dowód przez indukcję względem ilości krawędzi
- baza: dla grafu bez krawędzi zachodzi
- krok indukcyjny: zaczynamy z dowolnego wierzchołka v i poruszamy się aż trafimy znów do v ...

Dowód c.d.

Graf

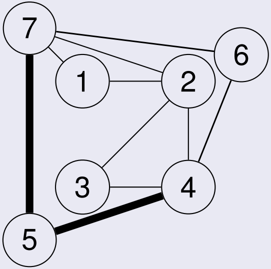


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf

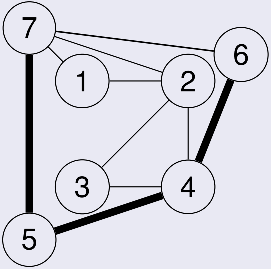


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf

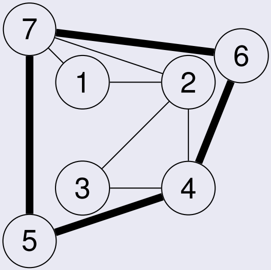


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf

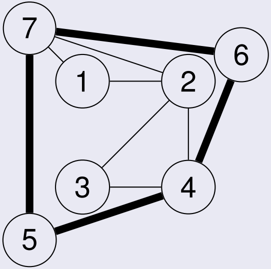


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf

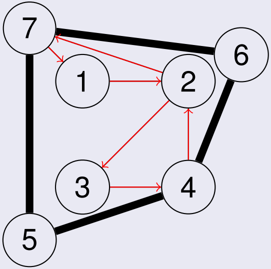


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf

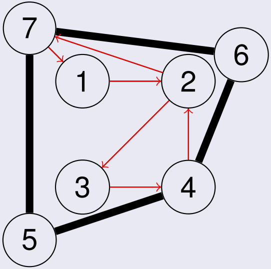


Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Dowód c.d.

Graf



Co dalej?

- dlaczego trafiliśmy do v?
- w pozostałej części grafu cykl Eulera istnieje — $(7, 1, 2, 3, 4, 2, 7)$
- łączymy oba cykle — $(7, 5, 4, 6, 7, 1, 2, 3, 4, 2, 7)$

Ścieżka Eulera

Ścieżka Eulera

- `Ścieżka` Eulera — ścieżka przechodząca przez każdą krawędź dokładnie raz nie będąca cyklem
- graf musi być spójny
- niech p i k będą początkiem i końcem ścieżki Eulera
- po dodaniu krawędzi łączącej p i k otrzymujemy graf eulerowski
- zatem w grafie muszą istnieć dokładnie dwa wierzchołki nieparzystego stopnia

Ścieżka Eulera

Ścieżka Eulera

- `Ścieżka` Eulera — ścieżka przechodząca przez każdą krawędź dokładnie raz nie będąca cyklem
- graf musi być spójny
- niech p i k będą początkiem i końcem ścieżki Eulera
- po dodaniu krawędzi łączącej p i k otrzymujemy graf eulerowski
- zatem w grafie muszą istnieć dokładnie dwa wierzchołki nieparzystego stopnia

Ścieżka Eulera

Ścieżka Eulera

- `Ścieżka` Eulera — ścieżka przechodząca przez każdą krawędź dokładnie raz nie będąca cyklem
- graf musi być spójny
- niech p i k będą początkiem i końcem ścieżki Eulera
- po dodaniu krawędzi łączącej p i k otrzymujemy graf eulerowski
- zatem w grafie muszą istnieć dokładnie dwa wierzchołki nieparzystego stopnia

Ścieżka Eulera

Ścieżka Eulera

- `Ścieżka` Eulera — ścieżka przechodząca przez każdą krawędź dokładnie raz nie będąca cyklem
- graf musi być spójny
- niech p i k będą początkiem i końcem ścieżki Eulera
- po dodaniu krawędzi łączącej p i k otrzymujemy graf eulerowski
- zatem w grafie muszą istnieć dokładnie dwa wierzchołki nieparzystego stopnia

Ścieżka Eulera

Ścieżka Eulera

- `Ścieżka` Eulera — ścieżka przechodząca przez każdą krawędź dokładnie raz nie będąca cyklem
- graf musi być spójny
- niech p i k będą początkiem i końcem ścieżki Eulera
- po dodaniu krawędzi łączącej p i k otrzymujemy graf eulerowski
- zatem w grafie muszą istnieć dokładnie dwa wierzchołki nieparzystego stopnia

Cykl Eulera w grafach skierowanych

Warunki konieczne:

- z wierzchołka nr. 1 da się dojść do każdego innego
- do każdego wierzchołka wchodzi tyle samo krawędzi ile z niego wychodzi
- powyższe warunki są dostateczne

Cykl Eulera w grafach skierowanych

Warunki konieczne:

- z wierzchołka nr. 1 da się dojść do każdego innego
- do każdego wierzchołka wchodzi tyle samo krawędzi ile z niego wychodzi
- powyższe warunki są dostateczne

Cykl Eulera w grafach skierowanych

Warunki konieczne:

- z wierzchołka nr. 1 da się dojść do każdego innego
- do każdego wierzchołka wchodzi tyle samo krawędzi ile z niego wychodzi
- powyższe warunki są dostateczne

Wyznaczanie cyklu Eulera

Wyznaczanie cyklu Eulera

- umiemy już sprawdzić czy cykl Eulera istnieje, ale jak go znaleźć?
- służy do tego algorytm `Fleury'ego`, który działa trochę na zasadzie opisanego dowodu
- działa zarówno dla grafów skierowanych jak i nieskierowanych
- jest rekurencyjny
- zaczynamy od dowolnego wierzchołka

Wyznaczanie cyklu Eulera

Wyznaczanie cyklu Eulera

- umiemy już sprawdzić czy cykl Eulera istnieje, ale jak go znaleźć?
- służy do tego algorytm `Fleury'ego`, który działa trochę na zasadzie opisanego dowodu
- działa zarówno dla grafów skierowanych jak i nieskierowanych
- jest rekurencyjny
- zaczynamy od dowolnego wierzchołka

Wyznaczanie cyklu Eulera

Wyznaczanie cyklu Eulera

- umiemy już sprawdzić czy cykl Eulera istnieje, ale jak go znaleźć?
- służy do tego algorytm `Fleury'ego`, który działa trochę na zasadzie opisanego dowodu
- działa zarówno dla grafów skierowanych jak i nieskierowanych
- jest rekurencyjny
- zaczynamy od dowolnego wierzchołka

Wyznaczanie cyklu Eulera

Wyznaczanie cyklu Eulera

- umiemy już sprawdzić czy cykl Eulera istnieje, ale jak go znaleźć?
- służy do tego algorytm `Fleury'ego`, który działa trochę na zasadzie opisanego dowodu
- działa zarówno dla grafów skierowanych jak i nieskierowanych
- jest rekurencyjny
- zaczynamy od dowolnego wierzchołka

Wyznaczanie cyklu Eulera

Wyznaczanie cyklu Eulera

- umiemy już sprawdzić czy cykl Eulera istnieje, ale jak go znaleźć?
- służy do tego algorytm `Fleury'ego`, który działa trochę na zasadzie opisanego dowodu
- działa zarówno dla grafów skierowanych jak i nieskierowanych
- jest rekurencyjny
- zaczynamy od dowolnego wierzchołka

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

- dopóki istnieje krawędź z v:
  - usuń tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

- dopóki istnieje krawędź z v:
  - **usuń** tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

- dopóki istnieje krawędź z v:
  - **usuń** tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

- dopóki istnieje krawędź z v:
  - **usuń** tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

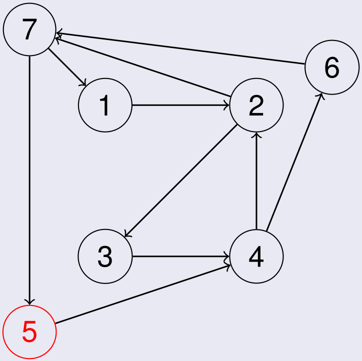
- dopóki istnieje krawędź z v:
  - **usuń** tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

Algorytm Fleury'ego

Aby przetworzyć wierzchołek v:

- dopóki istnieje krawędź z v:
  - **usuń** tą krawędź i przetwórz wierzchołek do którego prowadziła
  - wypisz v
- wypisane wierzchołki czytane w odwrotnej kolejności tworzą cykl Eulera (pierwszy wierzchołek nie jest powtórzony na końcu)
- dlaczego to działa? popatrzmy na przykład

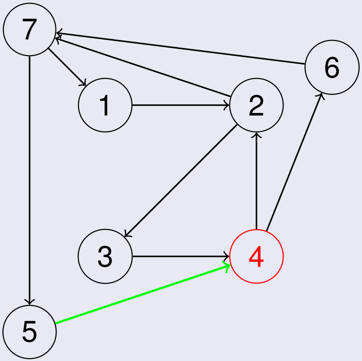
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

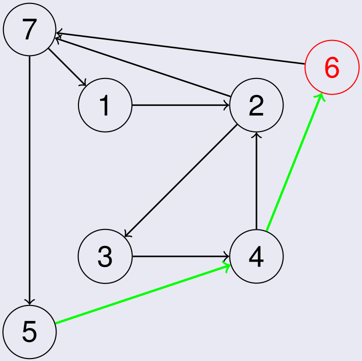
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

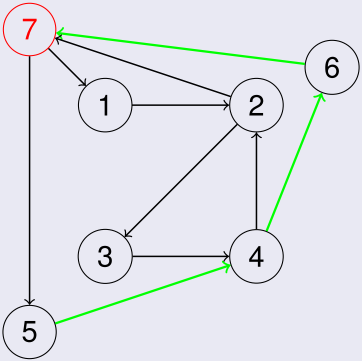
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

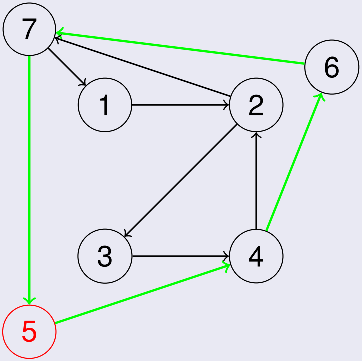
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

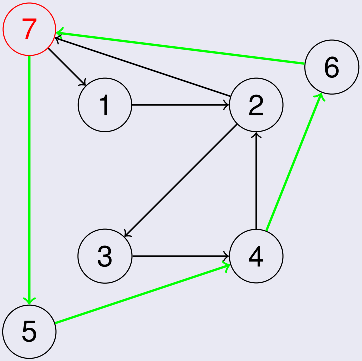
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

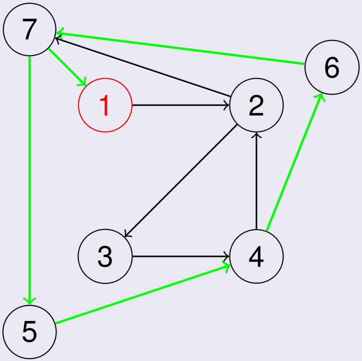
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

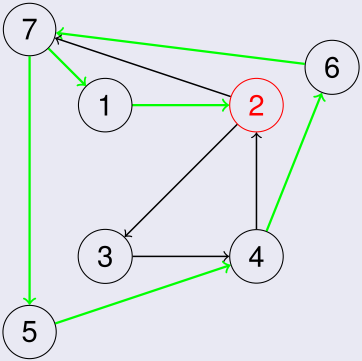
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

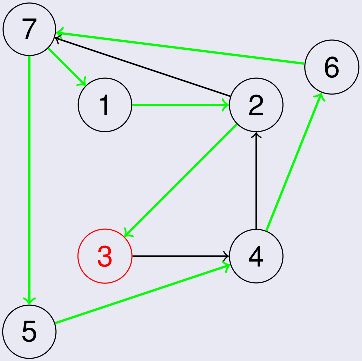
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

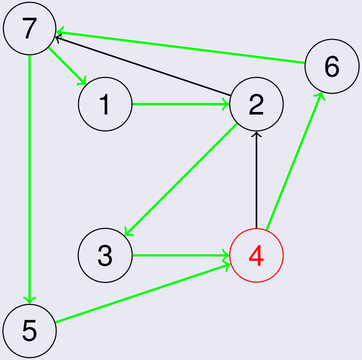
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

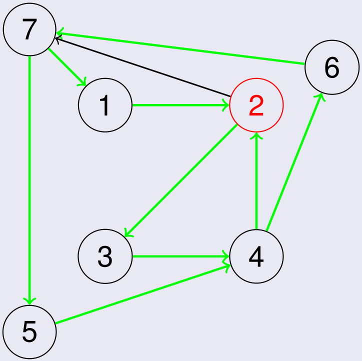
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

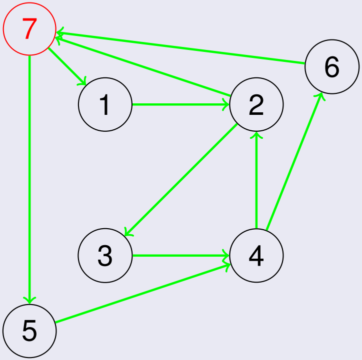
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

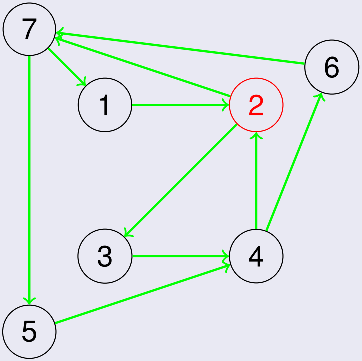
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

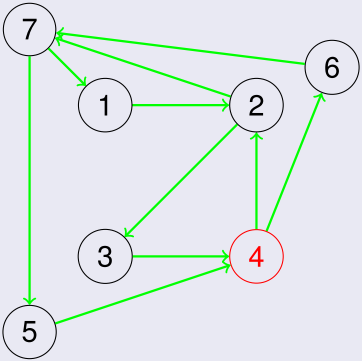
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

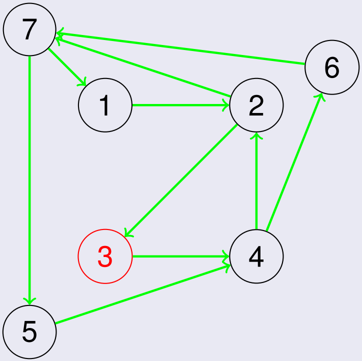
Algorytm Fleury’ego



Wyjście

7 2 4 3 2 1 7 6 4 5

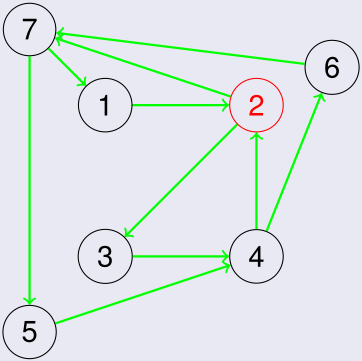
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

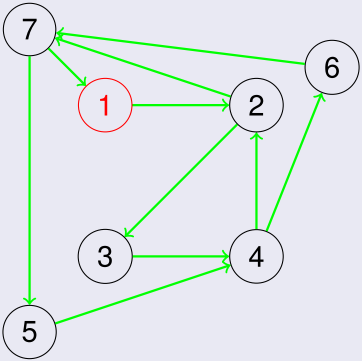
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

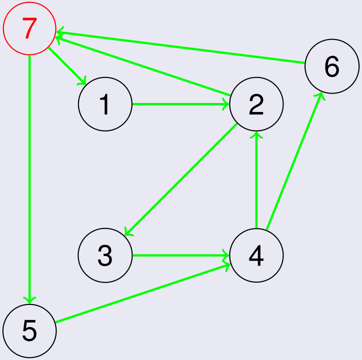
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

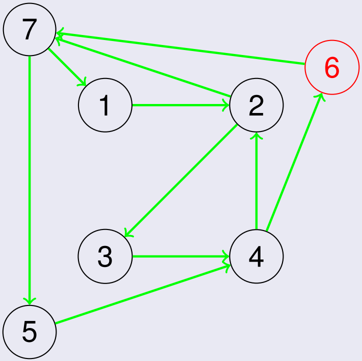
Algorytm Fleury’ego



Wyjście

7 2 4 3 2 1 7 6 4 5

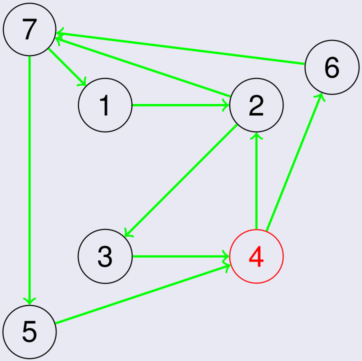
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

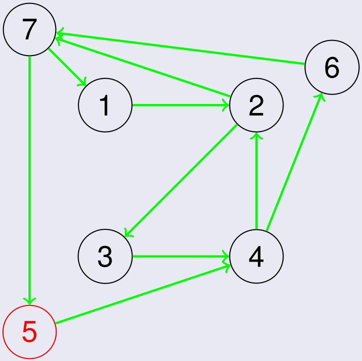
Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

Algorytm Fleury'ego



Wyjście

7 2 4 3 2 1 7 6 4 5

Algorytm Fleury'ego

Usuwanie krawędzi

- dla macierzy sąsiedztwa — łatwe
- dla list sąsiedztwa:
  - dla grafów skierowanych — metoda `pop_back()`
  - dla grafów nieskierowanych — a co z krawędzią "w drugą stronę"?
  - problem ten pominiemy do czasu aż poznamy klasę `set`

Algorytm Fleury'ego

Usuwanie krawędzi

- dla macierzy sąsiedztwa — łatwe
- dla list sąsiedztwa:
  - dla grafów skierowanych — metoda `pop_back()`
  - dla grafów nieskierowanych — a co z krawędzią "w drugą stronę"?
  - problem ten pominiemy do czasu aż poznamy klasę `set`

Algorytm Fleury'ego

Usuwanie krawędzi

- dla macierzy sąsiedztwa — łatwe
- dla list sąsiedztwa:
  - dla grafów skierowanych — metoda `pop_back()`
  - dla grafów nieskierowanych — a co z krawędzią "w drugą stronę"?
  - problem ten pominiemy do czasu aż poznamy klasę `set`

Algorytm Fleury'ego

Usuwanie krawędzi

- dla macierzy sąsiedztwa — łatwe
- dla list sąsiedztwa:
  - dla grafów skierowanych — metoda `pop_back()`
  - dla grafów nieskierowanych — a co z krawędzią "w drugą stronę"?
  - problem ten pominiemy do czasu aż poznamy klasę `set`

Algorytm Fleury'ego

Usuwanie krawędzi

- dla macierzy sąsiedztwa — łatwe
- dla list sąsiedztwa:
  - dla grafów skierowanych — metoda `pop_back()`
  - dla grafów nieskierowanych — a co z krawędzią "w drugą stronę"?
  - problem ten pominiemy do czasu aż poznamy klasę `set`

Implementacja

Implementacja

Implementacja algorytmu Fleury'ego znajduje się w notatkach.

Algorytm Fleury'ego

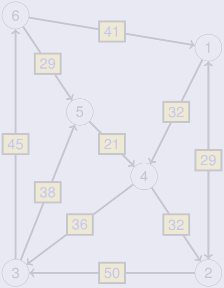
Analiza złożoności — lista sąsiedztwa:

Po każdej krawędzi przechodzimy dokładnie raz — złożoność $O(|E|)$.

Grafy ważone

Grafy ważone

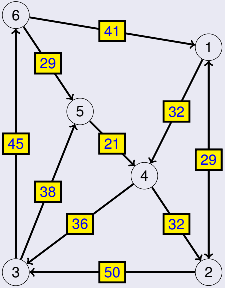
- grafem `ważonym` lub inaczej `siecią` nazywamy graf, którego krawędzie posiadają w pewnym sensie "długości"
- przykład grafu ważonego:



Grafy ważone

Grafy ważone

- grafem `ważonym` lub inaczej `siecią` nazywamy graf, którego krawędzie posiadają w pewnym sensie "długości"
- przykład grafu ważonego:



Grafy ważone

Grafy ważone

- owe liczby będziemy nazywać wagą, długością albo kosztem krawędzi
- długość ścieżki to suma długości jej krawędzi

Grafy ważone

Grafy ważone

- owe liczby będziemy nazywać wagą, długością albo kosztem krawędzi
- długość ścieżki to suma długości jej krawędzi

Najkrótsze ścieżki

Pytania

- jaka jest najkrótsza ścieżka z wierzchołka v do w?
- jakie są najkrótsze ścieżki z wierzchołka v do wszystkich pozostałych wierzchołków?
- jakie są najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków?

Najkrótsze ścieżki

Pytania

- jaka jest najkrótsza ścieżka z wierzchołka v do w?
- jakie są najkrótsze ścieżki z wierzchołka v do wszystkich pozostałych wierzchołków?
- jakie są najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków?

Najkrótsze ścieżki

Pytania

- jaka jest najkrótsza ścieżka z wierzchołka v do w?
- jakie są najkrótsze ścieżki z wierzchołka v do wszystkich pozostałych wierzchołków?
- jakie są najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków?

Najkrótsze ścieżki

Najkrótsze ścieżki

- zazwyczaj nie da się policzyć najkrótszej ścieżki z v do w bez liczenia najkrótszych ścieżek z v do wszystkich pozostałych wierzchołków
- dlatego też skupimy się na liczeniu wszystkich najkrótszych ścieżek z pojedynczego źródła (oznaczmy je s)

Najkrótsze ścieżki

Najkrótsze ścieżki

- zazwyczaj nie da się policzyć najkrótszej ścieżki z v do w bez liczenia najkrótszych ścieżek z v do wszystkich pozostałych wierzchołków
- dlatego też skupimy się na liczeniu wszystkich najkrótszych ścieżek z pojedynczego źródła (oznaczmy je s)

Najkrótsze ścieżki

Zarys metody postępowania

- w każdej chwili obliczeń będziemy trzymali pewne ścieżki z s do pozostałych wierzchołków
- konkretniej będziemy trzymali 2 tablice:
  - $dis[v]$ — długość najkrótszej znalezionej ścieżki z s do v
  - $ojc[v]$ — ostatni nie licząc v wierzchołek na najkrótszej ścieżce z s do v

Najkrótsze ścieżki

Zarys metody postępowania

- w każdej chwili obliczeń będziemy trzymali pewne ścieżki z s do pozostałych wierzchołków
- konkretniej będziemy trzymali 2 tablice:
  - $dis[v]$ — długość najkrótszej znalezionej ścieżki z s do v
  - $ojc[v]$ — ostatni nie licząc v wierzchołek na najkrótszej ścieżce z s do v

Najkrótsze ścieżki

Zarys metody postępowania

- w każdej chwili obliczeń będziemy trzymali pewne ścieżki z s do pozostałych wierzchołków
- konkretniej będziemy trzymali 2 tablice:
  - $dis[v]$ — długość najkrótszej znalezionej ścieżki z s do v
  - $ojc[v]$ — ostatni nie licząc v wierzchołek na najkrótszej ścieżce z s do v

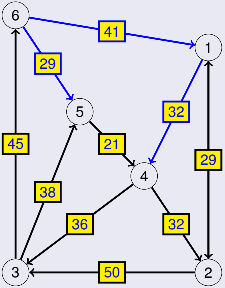
Najkrótsze ścieżki

Zarys metody postępowania

- w każdej chwili obliczeń będziemy trzymali pewne ścieżki z s do pozostałych wierzchołków
- konkretniej będziemy trzymali 2 tablice:
  - $dis[v]$ — długość najkrótszej znalezionej ścieżki z s do v
  - $ojc[v]$ — ostatni nie licząc v wierzchołek na najkrótszej ścieżce z s do v

Sytuacja w trakcie obliczeń

Graf



Tablice

v	1	2	3	4	5	6
$dis[v]$	41	∞	∞	73	29	0
$ojc[v]$	6	x	x	1	6	x

Relaksacja krawędzi

Relaksacja krawędzi

- podstawową operacją w rozważanych algorytmach będzie relaksacja krawędzi
- relaksacja krawędzi $v \rightarrow w$ polega na sprawdzeniu, czy przejście najkrótszą znaną ścieżką z s do v a następnie krawędzią z v do w nie daje krótszej ścieżki z s do w niż obecnie znana
- if dis[v]+waga(v,w) < dis[w]
  - dis[w]=dis[v]+waga(v,w)
  - ojc[w]=v

Relaksacja krawędzi

Relaksacja krawędzi

- podstawową operacją w rozważanych algorytmach będzie relaksacja krawędzi
- relaksacja krawędzi $v \rightarrow w$ polega na sprawdzeniu, czy przejście najkrótszą znaną ścieżką z s do v a następnie krawędzią z v do w nie daje krótszej ścieżki z s do w niż obecnie znana
- if dis[v]+waga(v,w) < dis[w]
  - dis[w]=dis[v]+waga(v,w)
  - ojc[w]=v

Relaksacja krawędzi

Relaksacja krawędzi

- podstawową operacją w rozważanych algorytmach będzie relaksacja krawędzi
- relaksacja krawędzi $v \to w$ polega na sprawdzeniu, czy przejście najkrótszą znaną ścieżką z s do v a następnie krawędzią z v do w nie daje krótszej ścieżki z s do w niż obecnie znana
- if dis[v]+waga(v,w) < dis[w]
  - dis[w]=dis[v]+waga(v,w)
  - ojc[w]=v

Relaksacja krawędzi

Relaksacja krawędzi

- podstawową operacją w rozważanych algorytmach będzie relaksacja krawędzi
- relaksacja krawędzi $v \rightarrow w$ polega na sprawdzeniu, czy przejście najkrótszą znaną ścieżką z s do v a następnie krawędzią z v do w nie daje krótszej ścieżki z s do w niż obecnie znana
- if dis[v]+waga(v,w) < dis[w]
  - dis[w]=dis[v]+waga(v,w)
  - ojc[w]=v

Relaksacja krawędzi

Relaksacja krawędzi

- podstawową operacją w rozważanych algorytmach będzie relaksacja krawędzi
- relaksacja krawędzi $v \rightarrow w$ polega na sprawdzeniu, czy przejście najkrótszą znaną ścieżką z s do v a następnie krawędzią z v do w nie daje krótszej ścieżki z s do w niż obecnie znana
- if dis[v]+waga(v,w) < dis[w]
  - dis[w]=dis[v]+waga(v,w)
  - ojc[w]=v

Algorytm Dijkstry

Algorytm Dijkstry

- pokażemy teraz najbardziej znany algorytm liczenia najkrótszych ścieżek z pojedynczego źródła — algorytm `Dijkstry`
- algorytm ten Działa tylko dla grafów o **nieujemnych** wagach krawędzi
- jest to założenie często spełnione, np. kiedy wagi odpowiadają odległościom

Algorytm Dijkstry

Algorytm Dijkstry

- pokażemy teraz najbardziej znany algorytm liczenia najkrótszych ścieżek z pojedynczego źródła — algorytm `Dijkstry`
- algorytm ten Działa tylko dla grafów o **nieujemnych** wagach krawędzi
- jest to założenie często spełnione, np. kiedy wagi odpowiadają odległościom

Algorytm Dijkstry

Algorytm Dijkstry

- pokażemy teraz najbardziej znany algorytm liczenia najkrótszych ścieżek z pojedynczego źródła — algorytm `Dijkstry`
- algorytm ten Działa tylko dla grafów o **nieujemnych** wagach krawędzi
- jest to założenie często spełnione, np. kiedy wagi odpowiadają odległościom

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

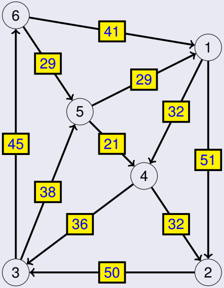
Algorytm Dijkstry

Algorytm Dijkstry

- oznacz wszystkie wierzchołki jako nieodwiedzone ($vis[v] = false$)
- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- dopóki istnieje nieodwiedzony wierzchołek o skończonej odległości:
  - niech v będzie wierzchołkiem nieodwiedzonym o najmniejszej odległości
  - oznacz v jako odwiedzony
  - zrelaksuj wszystkie krawędzie wychodzące z v

Algorytm Dijkstry

Graf

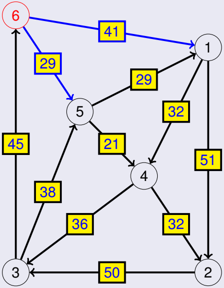


Tablice

v	1	2	3	4	5	6
$dis[v]$	∞	∞	∞	∞	∞	0
$ojc[v]$	x	x	x	x	x	x
$vis[v]$	0	0	0	0	0	0

Algorytm Dijkstry

Graf

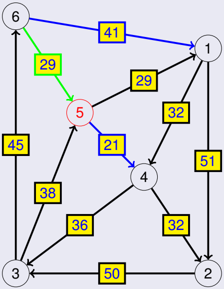


Tablice

v	1	2	3	4	5	6
$dis[v]$	41	∞	∞	∞	29	0
$ojc[v]$	6	x	x	x	6	x
$vis[v]$	0	0	0	0	1	0

Algorytm Dijkstry

Graf

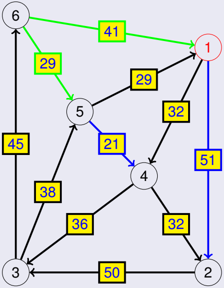


Tablice

v	1	2	3	4	5	6
$dis[v]$	41	∞	∞	50	29	0
$ojc[v]$	6	x	x	5	6	x
$vis[v]$	0	0	0	0	1	1

Algorytm Dijkstry

Graf

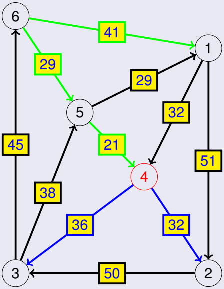


Tablice

v	1	2	3	4	5	6
$dis[v]$	41	92	∞	50	29	0
$ojc[v]$	6	1	x	5	6	x
$vis[v]$	1	0	0	0	1	1

Algorytm Dijkstry

Graf

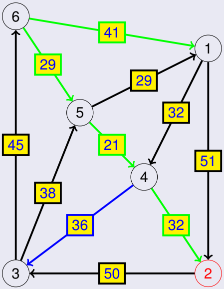


Tablice

v	1	2	3	4	5	6
$dis[v]$	41	82	86	50	29	0
$ojc[v]$	6	4	4	5	6	x
$vis[v]$	1	0	0	1	1	1

Algorytm Dijkstry

Graf

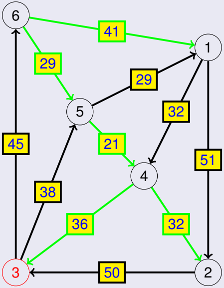


Tablice

v	1	2	3	4	5	6
$dis[v]$	41	82	86	50	29	0
$ojc[v]$	6	4	4	5	6	x
$vis[v]$	1	1	0	1	1	1

Algorytm Dijkstry

Graf



Tablice

v	1	2	3	4	5	6
$dis[v]$	41	82	86	50	29	0
$ojc[v]$	6	4	4	5	6	x
$vis[v]$	1	1	1	1	1	1

Algorytm Dijkstry

Analiza złożoności — listy sąsiedztwa:

- złożoność pamięciowa $O(|V| + |E|)$
- każdą krawędź relaksujemy co najwyżej raz — czas $O(|E|)$
- $O(|V|)$ razy wybieramy wierzchołek o najmniejszej odległości, co zajmuje jednorazowo czas $O(|V|)$, czyli łącznie $O(|V|^2)$
- złożoność czasowa $O(|V|^2 + |E|) = O(|V|^2)$
- da się tą złożoność poprawić, ale o tym będzie kiedy indziej …

Algorytm Dijkstry

Analiza złożoności — listy sąsiedztwa:

- złożoność pamięciowa $O(|V| + |E|)$
- każdą krawędź relaksujemy co najwyżej raz — czas $O(|E|)$
- $O(|V|)$ razy wybieramy wierzchołek o najmniejszej odległości, co zajmuje jednorazowo czas $O(|V|)$, czyli łącznie $O(|V|^2)$
- złożoność czasowa $O(|V|^2 + |E|) = O(|V|^2)$
- da się tą złożoność poprawić, ale o tym będzie kiedy indziej ...

Algorytm Dijkstry

Analiza złożoności — listy sąsiedztwa:

- złożoność pamięciowa $O(|V| + |E|)$
- każdą krawędź relaksujemy co najwyżej raz — czas $O(|E|)$
- $O(|V|)$ razy wybieramy wierzchołek o najmniejszej odległości, co zajmuje jednorazowo czas $O(|V|)$, czyli łącznie $O(|V|^2)$
- złożoność czasowa $O(|V|^2 + |E|) = O(|V|^2)$
- da się tą złożoność poprawić, ale o tym będzie kiedy indziej ...

Algorytm Dijkstry

Analiza złożoności — listy sąsiedztwa:

- złożoność pamięciowa $O(|V| + |E|)$
- każdą krawędź relaksujemy co najwyżej raz — czas $O(|E|)$
- $O(|V|)$ razy wybieramy wierzchołek o najmniejszej odległości, co zajmuje jednorazowo czas $O(|V|)$, czyli łącznie $O(|V|^2)$
- złożoność czasowa $O(|V|^2 + |E|) = O(|V|^2)$
- da się tą złożoność poprawić, ale o tym będzie kiedy indziej ...

Algorytm Dijkstry

Analiza złożoności — listy sąsiedztwa:

- złożoność pamięciowa $O(|V| + |E|)$
- każdą krawędź relaksujemy co najwyżej raz — czas $O(|E|)$
- $O(|V|)$ razy wybieramy wierzchołek o najmniejszej odległości, co zajmuje jednorazowo czas $O(|V|)$, czyli łącznie $O(|V|^2)$
- złożoność czasowa $O(|V|^2 + |E|) = O(|V|^2)$
- da się tą złożoność poprawić, ale o tym będzie kiedy indziej . . .

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \to v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \rightarrow v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \to v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \to v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \to v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy *dis* spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \to v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy *dis* spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \rightarrow v$

Algorytm Dijkstry

Dlaczego to działa?

- po rozpatrzeniu v wartość $dis[v]$ już się nie zmieni, bo rozpatrujemy wierzchołki coraz bardziej odległe od s
- udowodnimy, że w momencie odwiedzania v najkrótsza ścieżka z s do v ma długość $dis[v]$:
  - dowód przez indukcję "w czasie"
  - baza — w momencie wejścia do s zachodzi $dis[s] = 0$
  - krok indukcyjny — załóżmy, że odwiedzamy v:
  - najkrótsza ścieżka do v musi biec tylko przez już odwiedzone wierzchołki, bo v ma najniższą wartość w tablicy dis spośród nieodwiedzonych wierzchołków
  - jest więc ona postaci $(s, x_1, \ldots, x_n, v)$ dla odwiedzonych x_i
  - rozważyliśmy ją relaksując krawędź $x_n \rightarrow v$

Implementacja

Implementacja

Implementacja algorytmu Dijkstry znajduje się w notatkach.

Sieci z ujemnymi wagami

Sieci z ujemnymi wagami

- przypomnijmy, że algorytm Dijkstry nie dział dla sieci z ujemnymi wagami
- takie sieci czasem pojawiają się w zadaniach, np:

> *Mamy n różnych przedmiotów oraz informacje o możliwych transakcjach postaci: "przedmiot A można wymienić na B zyskując/dopłacając C zł". Masz przedmiot nr. 1 a chcesz mieć przedmiot nr. n. Ile musisz minimalnie wydać lub ile możesz maksymalnie zarobić?*

Sieci z ujemnymi wagami

- przypomnijmy, że algorytm Dijkstry nie dział dla sieci z ujemnymi wagami
- takie sieci czasem pojawiają się w zadaniach, np:

> *Mamy n różnych przedmiotów oraz informacje o możliwych transakcjach postaci: "przedmiot A można wymienić na B zyskując/dopłacając C zł". Masz przedmiot nr. 1 a chcesz mieć przedmiot nr. n. Ile musisz minimalnie wydać lub ile możesz maksymalnie zarobić?*

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ …
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Sieci z ujemnymi wagami

Ujemne cykle

- zastanówmy się co by się stało, gdybyśmy szukali najkrótszych ścieżek w grafie z ujemnym cyklem o sumie wag $-c$ ($c > 0$)
- niech v należy do takiego cyklu
- jaka jest długość najkrótszej ścieżki z v do v
- jeśli przejdziemy po cyklu to otrzymujemy ścieżkę o koszcie $-c$
- jeśli przejdziemy po cyklu dwukrotnie to otrzymujemy ścieżkę o koszcie $-2c$
- jeśli przejdziemy po cyklu trzykrotnie to otrzymujemy ścieżkę o koszcie $-3c$ ...
- widzimy, że najkrótsza ścieżka z v do v nie istnieje!

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda

- dlatego też załóżmy, że rozważamy graf z ujemnymi wagami, ale bez ujemnych cykli
- do liczenia najkrótszych ścieżek w takim grafie możemy użyć algorytmu `Bellmana-Forda`

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda

- dlatego też załóżmy, że rozważamy graf z ujemnymi wagami, ale bez ujemnych cykli
- do liczenia najkrótszych ścieżek w takim grafie możemy użyć algorytmu `Bellmana-Forda`

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda:

- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- powtórz $|V| - 1$ razy poniższy krok:
  - zrelaksuj wszystkie krawędzie w dowolnej kolejności

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda:

- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- powtórz $|V| - 1$ razy poniższy krok:
  - zrelaksuj wszystkie krawędzie w dowolnej kolejności

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda:

- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- powtórz $|V| - 1$ razy poniższy krok:
  - zrelaksuj wszystkie krawędzie w dowolnej kolejności

Algorytm Bellmana-Forda

Algorytm Bellmana-Forda:

- dla każdego $v \in V$ ustaw $dis[v] = \infty$
- ustaw $dis[s] = 0$
- powtórz $|V| - 1$ razy poniższy krok:
  - zrelaksuj wszystkie krawędzie w dowolnej kolejności

Algorytm Bellmana-Forda

Dlaczego to działa?

- po 1-szym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z 1 krawędzi
- po 2-gim wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej 2 krawędzi
- po k-tym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej k krawędzi
- najkrótsze ścieżki składają się z co najwyżej $|V| - 1$, gdyż dłuższe ścieżki zawierają cykl o nieujemnej wadze

Algorytm Bellmana-Forda

Dlaczego to działa?

- po 1-szym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z 1 krawędzi
- po 2-gim wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej 2 krawędzi
- po k-tym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej k krawędzi
- najkrótsze ścieżki składają się z co najwyżej $|V| - 1$, gdyż dłuższe ścieżki zawierają cykl o nieujemnej wadze

Dlaczego to działa?

- po 1-szym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z 1 krawędzi
- po 2-gim wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej 2 krawędzi
- po k-tym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej k krawędzi
- najkrótsze ścieżki składają się z co najwyżej $|V| - 1$, gdyż dłuższe ścieżki zawierają cykl o nieujemnej wadze

Algorytm Bellmana-Forda

Dlaczego to działa?

- po 1-szym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z 1 krawędzi
- po 2-gim wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej 2 krawędzi
- po k-tym wykonaniu pętli mamy policzone najkrótsze ścieżki do wierzchołków dla których najkrótsza ścieżka składa się z co najwyżej k krawędzi
- najkrótsze ścieżki składają się z co najwyżej $|V| - 1$, gdyż dłuższe ścieżki zawierają cykl o nieujemnej wadze

Algorytm Bellmana-Forda

Wykrywanie ujemnych cykli

- zastanówmy się co by się stało, gdybyśmy wykonali $|V|$-tą iterację pętli
- jeśli uda nam się cokolwiek zrelaksować, to wiemy, że w grafie istnieje ujemny cykl
- jeśli w grafie istnieje ujemny cykl, to uda nam się coś zrelaksować — gdybyśmy nic nie zrelaksowali to kolejne powtórzenia pętli też nic by nie relaksowały a zatem pomimo dopuszczenia ścieżek składających się z dowolnie wielu krawędzi nie otrzymali byśmy ścieżek o dowolnie małym koszcie — sprzeczność
- otrzymujemy zatem prosty test na sprawdzanie czy graf zawiera ujemny cykl

Algorytm Bellmana-Forda

Wykrywanie ujemnych cykli

- zastanówmy się co by się stało, gdybyśmy wykonali $|V|$-tą iterację pętli
- jeśli uda nam się cokolwiek zrelaksować, to wiemy, że w grafie istnieje ujemny cykl
- jeśli w grafie istnieje ujemny cykl, to uda nam się coś zrelaksować — gdybyśmy nic nie zrelaksowali to kolejne powtórzenia pętli też nic by nie relaksowały a zatem pomimo dopuszczenia ścieżek składających się z dowolnie wielu krawędzi nie otrzymali byśmy ścieżek o dowolnie małym koszcie — sprzeczność
- otrzymujemy zatem prosty test na sprawdzanie czy graf zawiera ujemny cykl

Algorytm Bellmana-Forda

Wykrywanie ujemnych cykli

- zastanówmy się co by się stało, gdybyśmy wykonali $|V|$-tą iterację pętli
- jeśli uda nam się cokolwiek zrelaksować, to wiemy, że w grafie istnieje ujemny cykl
- jeśli w grafie istnieje ujemny cykl, to uda nam się coś zrelaksować — gdybyśmy nic nie zrelaksowali to kolejne powtórzenia pętli też nic by nie relaksowały a zatem pomimo dopuszczenia ścieżek składających się z dowolnie wielu krawędzi nie otrzymali byśmy ścieżek o dowolnie małym koszcie — sprzeczność
- otrzymujemy zatem prosty test na sprawdzanie czy graf zawiera ujemny cykl

Algorytm Bellmana-Forda

Wykrywanie ujemnych cykli

- zastanówmy się co by się stało, gdybyśmy wykonali $|V|$-tą iterację pętli
- jeśli uda nam się cokolwiek zrelaksować, to wiemy, że w grafie istnieje ujemny cykl
- jeśli w grafie istnieje ujemny cykl, to uda nam się coś zrelaksować — gdybyśmy nic nie zrelaksowali to kolejne powtórzenia pętli też nic by nie relaksowały a zatem pomimo dopuszczenia ścieżek składających się z dowolnie wielu krawędzi nie otrzymali byśmy ścieżek o dowolnie małym koszcie — sprzeczność
- otrzymujemy zatem prosty test na sprawdzanie czy graf zawiera ujemny cykl

Implementacja

Implementacja

Implementacja algorytmu Bellmana-Forda znajduje się w notatkach.