Algorytmy Grafowe

dr hab. Bożena Woźna-Szcześniak, prof. UJD

Uniwersytet Jana Długosza w Częstochowie

b.wozna@ujd.edu.pl

Wykład 13 i 14

Spis treści

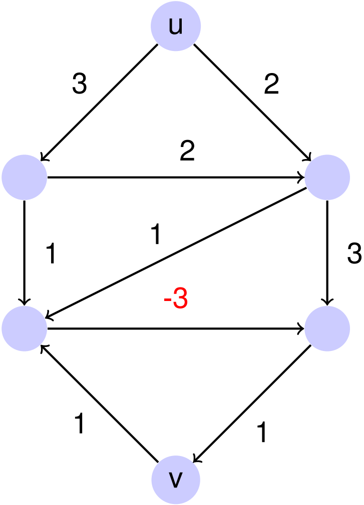
Graf skierowany z wagami

Definicja

Grafem skierowanym z wagami nazywamy strukturę $G = (V, E, weight : E \rightarrow Z)$ gdzie

- V to zbiór wierzchołków,
- $E \subseteq \{(u, v) : u, v \in V\}$ to zbiór uporządkowanych par wierzchołków ze zbioru V, zwanych krawędziami.
- $weight : E \rightarrow Z$ jest funkcją wagi (odległości).

Graf skierowany z wagami - przykład



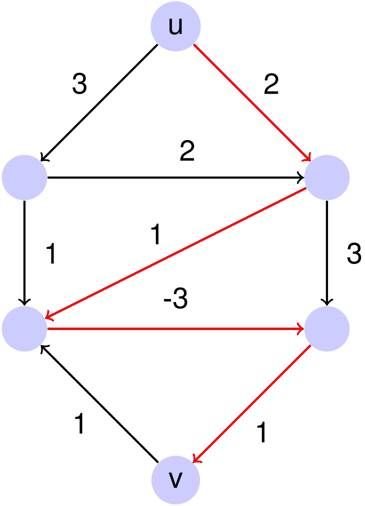
Drzewo najkrótszych ścieżek

Sformułowanie problemu

- **Wejście:**
  - Graf skierowany z wagami $G = (V, E, weight : E \to Z)$
  - Wierzchołek $r \in V$, zwany **korzeniem**.
- **Wyjście:**
  - Drzewo T o korzeniu r takie, że ścieżka z r do każdego wierzchołka u w T jest najkrótszą scieżką z r do u w grafie G.
- **Założenie:** Rozważane grafy mają wierzchołki osiągalne z wybranego wierzchołka (korzenia) r [a].

[a] Dlaczego? Wierzchołki nieosiągalne mogą być usunięte w czasie liniowym

Graf skierowany z wagami - przykład



Pytanie:

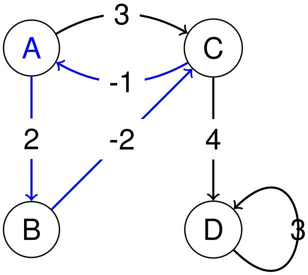
- Czy najkrótsza ścieżka pomiędzy wierzchołkami *u* i *v* może zawierać cykl?

Odpowiedź:

- Jeśli w grafie istnieje najkrótsza ścieżka z *u* do *v*, to w grafie tym również istnieje najkrótsza ścieżka z *u* do *v*, która nie zawiera cykli.

Ujemne cykle

- Niech $G = (V, E, weight : E \rightarrow Z)$ będzie grafem skierowanym z wagami. **Cykl ujemny** w grafie G, to cykl C, którego waga $weight(C)$ jest ujemna.



- C=(A,B,C,A), w(C) = -1.

Richard Ernest Bellman - (26.08.1920-19.03.1984)

- Richard Ernest Bellman znany jako twórca:
  - programowania dynamicznego.
  - algorytmu Bellmana–Forda.
- W 1979 roku otrzymał IEEE Medal of Honor za ”wkład w teorie sterowania i procesów decyzyjnych, szczególnie za opracowanie teorii programowania dynamicznego”.

Źródło: http://logistyka.math.uni.lodz.pl/Bellman.jpeg

- Więcej na http://en.wikipedia.org/wiki/Richard_E._Bellman

Lester Randolph Ford, junior (23.09.1927-26.02.2017)

- Amerykański matematyk specjalizujący się w algorytmach przepływu w sieci.
- Syn matematyka Lester R. Forda, seniora.
- Autor algorytmu Bellmana-Forda, służcego do znajdowania najkrótszej ścieżki w grafach z wagami.

Źródło:`http://wazniak.mimuw.edu.pl/index.php?title=Grafika:Ford-portret.jpg`

Algorytm Bellmana-Forda

- Algorytm obliczający najkrótsze ścieżki z danego wierzchołka źródłowego do wszystkich pozostałych wierzchołków w skierowanym grafie z wagami.
- Wolniejszy od **algorytmu Dijkstry** dla tego samego problemu, ale bardziej uniwersalny, ponieważ jest w stanie obsługiwać grafy, które zawierają krawędze o ujemnej wadze, ale nie zawierają ujemnych cykli.
- Algorytm po raz pierwszy został zaproponowany przez **Alfonso Shimbela** w 1955 roku, ale został nazwany imieniem **Richarda Bellmana** i **Lestera Forda, Jr.**, gdyż to oni opublikowali ten algorytm odpowiednio w 1958 i 1956 roku.
- **Edward F. Moore** opublikował również ten sam algorytm w 1957 r., Dlatego też algorytm ten nazywany jest również algorytmem Bellmana-Forda-Moore'a [1].

[1] Bang-Jensen, Jorgen; Gutin, Gregory (2000). Digraphs: Theory, Algorithms and Applications

Algorytm Bellmana-Forda - pseudokod

Input: Graf $G(V, E, weight)$, wierzchołek początkowy src.

```
for all v in V do
    v.distance = ∞;
    v.prev = Null;
end for
src.distance = 0
for i=1 to |V| - 1 do
    for all (u, v) in E do
        relax(u, v);
    end for
end for
```

Algorytm Bellmana-Forda - pseudokod

Input: Graf $G(V, E, weight)$, wierzchołek początkowy *src*.

```
1:  for all v in V do
2:      v.distance = ∞;
3:      v.prev = Null;
4:  end for
5:  src.distance = 0
6:  for i=1 to |V| - 1 do
7:      for all (u, v) in E do
8:          relax(u, v);
9:      end for
10: end for
```

- L.1-4: Pętla **for** ustawia dla każdego wierzchołka grafu parametr `distance(odległość)` na nieskończoność oraz wskaźnik **prev** na zero. **prev** jest wskaźnikiem do poprzednika danego wierzchołka – pozwala na odtworzenie najkrótszej ścieżki.

Algorytm Bellmana-Forda - pseudokod

Input: Graf $G(V, E, weight)$, wierzchołek początkowy *src*.

```
1:  for all v in V do
2:      v.distance = ∞;
3:      v.prev = Null;
4:  end for
5:  src.distance = 0
6:  for i=1 to |V| - 1 do
7:      for all (u, v) in E do
8:          relax(u, v);
9:      end for
10: end for
```

- L.1-4: Pętla **for** ustawia dla każdego wierzchołka grafu parametr `distance(odległość)` na nieskończoność oraz wskaźnik **prev** na zero. **prev** jest wskaźnikiem do poprzednika danego wierzchołka – pozwala na odtworzenie najkrótszej ścieżki.
- L.5: Odległość dla wierzchołka początkowego ustawiana jest na zero.

Algorytm Bellmana-Forda - pseudokod

Input: Graf $G(V, E, weight)$, wierzchołek początkowy *src*.

```
1:  for all v in V do
2:      v.distance = ∞;
3:      v.prev = Null;
4:  end for
5:  src.distance = 0
6:  for i=1 to |V| - 1 do
7:      for all (u, v) in E do
8:          relax(u, v);
9:      end for
10: end for
```

- L.1-4: Pętla **for** ustawia dla każdego wierzchołka grafu parametr `distance(odległość)` na nieskończoność oraz wskaźnik **prev** na zero. **prev** jest wskaźnikiem do poprzednika danego wierzchołka – pozwala na odtworzenie najkrótszej ścieżki.
- L.5: Odległość dla wierzchołka początkowego ustawiana jest na zero.
- L.7-9: Pętla **for** przechodzi przez każdą krawędź grafu (u, v) i dokonuje jej tzw. **relaksacji**. Proces ten wykonywany jest $|V| - 1$ razy

Relaksacja krawędzi

- **Relaksacja krawędzi** jest najważniejszym krokiem w algorytmie Bellmana-Forda.
- **Relaksacja krawędzi** to sprawdzenie, czy przy przejściu daną krawędzią grafu (u, v) (tj. z '*u*' do '*v*') , nie otrzymamy krótszej ścieżki niż dotychczasowa ze źródła '*src*' do '*v*'. Jeżeli tak, to zmniejszamy oszacowanie wagi $v.distance$ najkrótszej ścieżki.

```
relax(u, v):
if v.distance > u.distance + weight(u, v) then
    v.distance = u.distance + weight(u, v)
    v.prev = u
end if
```

Wykrywanie ujemnych cykli - pseudokod

Input: Graf $G(V, E, weight)$, wierzchołek początkowy *src*.

```
for all v in V do
    v.distance = ∞;
    v.prev = Null;
end for
src.distance = 0
for i=1 to |V| - 1 do
    for all (u, v) in E do
        relax(u, v);
    end for
end for
for all (u, v) in E do
    if v.distance > u.distance + weight(u, v) then
        "Istnieje ujemny cykl!"
    end if
end for
```

Krótki i prosty dodatek do algorytmu Bellmana-Forda pozwala mu wykrywać **ujemne cykle.**

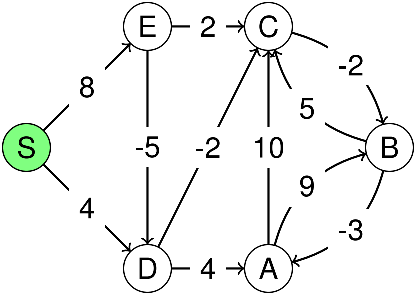
Złożoność algorytmu Bellmana-Forda

- Algorytm Bellman-Forda wykonuje $|E|$ relaksacji dla każdej iteracji.
- Iteracji jest $|V| - 1$.
- Dlatego w najgorszym przypadku, algorytm Bellmana-Forda działa w czasie $O(|V| \cdot |E|)$.

Ustawienie początkowe parametru `distance`

Input: *G*(*V*, *E*, *weight*), *src*.

```
 1: for all v in V do
 2:     v.distance = ∞;
 3:     v.prev = Null;
 4: end for
 5: src.distance = 0
 6: for i=1 to |V| - 1 do
 7:     for all (u, v) in E do
 8:         relax(u, v);
 9:     end for
10: end for
11: for all (u, v) in E do
12:     if v.distance > u.distance + weight(u, v) then
13:         "Istnieje ujemny cykl!"
14:     end if
15: end for
```



	s	a	b	c	d	e
l.1-4:	∞	∞	∞	∞	∞	∞
l.5:	0	∞	∞	∞	∞	∞

Proces relaksacji, iteracja 1, slajd I

Require: $G(V, E, weight)$, src.

```
1: ...
2: for i=1 to |V| - 1 do
3:     for all (u, v) in E do
4:         relax(u, v);
5:     end for
6: end for
7: ...
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $\infty > \infty + 9$ – false
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $\infty > \infty + 10$ – false

Proces relaksacji, iteracja 1, slajd II

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $\infty > \infty - 3$ – false
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $\infty > \infty + 5$ – false
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $\infty > \infty - 2$ – false
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $\infty > \infty + 4$ – false

Proces relaksacji, iteracja 1, slajd III

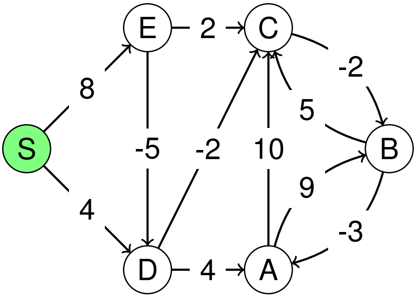
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $\infty > \infty - 2$ – false
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $\infty > \infty + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $\infty > \infty - 5$ – false
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $\infty > 0 + 4$ – true; $d.distance = 4$
- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $\infty > 0 + 8$ – true; $e.distance = 8$

Proces relaksacji, iteracja 1, slajd IV

Input: $G(V, E, weight)$, src.

```
1:  ...
2:  for i=1 to |V| - 1 do
3:      for all (u, v) in E do
4:          relax(u, v);
5:      end for
6:  end for
7:  for all (u, v) in E do
8:      if v.distance > u.distance +
        weight(u, v) then
9:          "Istnieje ujemny cykl!"
10:     end if
11: end for
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.



	s	a	b	c	d	e
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8

Proces relaksacji, iteracja 2, slajd I

Require: $G(V, E, weight)$, src.

```
1: ...
2: for i=1 to |V| - 1 do
3:     for all (u, v) in E do
4:         relax(u, v);
5:     end for
6: end for
7: ...
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $\infty > \infty + 9$ – false
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $\infty > \infty + 10$ – false

Proces relaksacji, iteracja 2, slajd II

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $\infty > \infty - 3$ – false
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $\infty > \infty + 5$ – false
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $\infty > \infty - 2$ – false
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $\infty > 4 + 4$ – true; $a.distance = 8$

Proces relaksacji, iteracja 2, slajd III

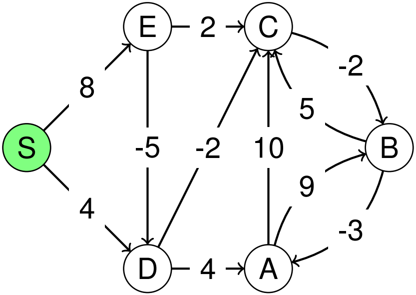
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $\infty > 4 - 2$ – true; $c.distance = 2$
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $2 > 8 + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $4 > 8 - 5$ – true; $d.distance = 3$
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $3 > 0 + 4$ – false;
- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $8 > 0 + 8$ – false;

Proces relaksacji, iteracja 2, slajd IV

Input: *G*(*V*, *E*, *weight*), *src*.

```
 1: ...
 2: for i=1 to |V| - 1 do
 3:     for all (u, v) in E do
 4:         relax(u, v);
 5:     end for
 6: end for
 7: for all (u, v) in E do
 8:     if v.distance > u.distance +
        weight(u, v) then
 9:         "Istnieje ujemny cykl!"
10:     end if
11: end for
```

W pętli l.2-6 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.



	s	a	b	c	d	e
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8
i=2	0	8	∞	2	3	8

Proces relaksacji, iteracja 3, slajd I

Require: $G(V, E, weight)$, src.

```
1: ...
2: for i=1 to |V| - 1 do
3:     for all (u, v) in E do
4:         relax(u, v);
5:     end for
6: end for
7: ...
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $\infty > 8 + 9$ – true; $b.distance = 17$
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $2 > 8 + 10$ – false

Proces relaksacji, iteracja 3, slajd II

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $8 > 17 - 3$ – false
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $2 > 17 + 5$ – false
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $17 > 2 - 2$ – true; $b.distance = 0$
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $8 > 3 + 4$ – true; $a.distance = 7$

Proces relaksacji, iteracja 3, slajd III

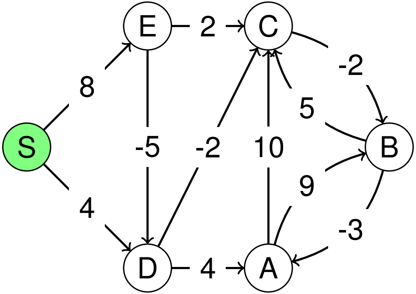
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $2 > 3 - 2$ – true; $c.distance = 1$
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $1 > 8 + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $3 > 8 - 5$ – false;
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $3 > 0 + 4$ – false;
- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $8 > 0 + 8$ – false;

Proces relaksacji, iteracja 3, slajd IV

Input: *G*(*V*, *E*, *weight*), *src*.

```
1:  ...
2:  for i=1 to |V| - 1 do
3:      for all (u, v) in E do
4:          relax(u, v);
5:      end for
6:  end for
7:  for all (u, v) in E do
8:      if v.distance > u.distance +
        weight(u, v) then
9:          "Istnieje ujemny cykl!"
10:     end if
11: end for
```

W pętli l.2-6 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.



	s	a	b	c	d	e
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8
i=2	0	8	∞	2	3	8
i=3	0	7	0	1	3	8

Proces relaksacji, iteracja 4, slajd I

Require: *G*(*V*, *E*, *weight*), *src*.

```
1: ...
2: for i=1 to |V| - 1 do
3:     for all (u, v) in E do
4:         relax(u, v);
5:     end for
6: end for
7: ...
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $0 > 7 + 9$ – false;
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $1 > 7 + 10$ – false

Proces relaksacji, iteracja 4, slajd II

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $7 > 0 - 3$ – true; $a.distance = -3$
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $1 > 0 + 5$ – false
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $0 > 1 - 2$ – true; $b.distance = -1$
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $-3 > 3 + 4$ – false;

Proces relaksacji, iteracja 4, slajd III

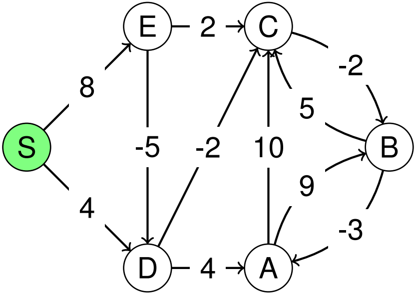
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $1 > 3 - 2$ – false;
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $1 > 8 + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $3 > 8 - 5$ – false;
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $3 > 0 + 4$ – false;
- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $8 > 0 + 8$ – false;

Proces relaksacji, iteracja 4, slajd IV

Input: $G(V, E, weight)$, src.

```
 1: ...
 2: for i=1 to |V| - 1 do
 3:     for all (u, v) in E do
 4:         relax(u, v);
 5:     end for
 6: end for
 7: for all (u, v) in E do
 8:     if v.distance > u.distance +
        weight(u, v) then
 9:         "Istnieje ujemny cykl!"
10:     end if
11: end for
```

W pętli l.2-6 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.



	s	a	b	c	d	e
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8
i=2	0	8	∞	2	3	8
i=3	0	7	0	1	3	8
i=4	0	-3	-1	1	3	8

Proces relaksacji, iteracja 5, slajd I

Require: $G(V, E, weight)$, src.

```
1: ...
2: for i=1 to |V| - 1 do
3:    for all (u, v) in E do
4:       relax(u, v);
5:    end for
6: end for
7: ...
```

W pętli l.3-5 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $-1 > -3 + 9$ – false;
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $1 > -3 + 10$ – false;

Proces relaksacji, iteracja 5, slajd II

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $-3 > -1 - 3$ – true; $a.distance = -4$
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $1 > -1 + 5$ – false;
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $-1 > 1 - 2$ – false;
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $-3 > 3 + 4$ – false;

Proces relaksacji, iteracja 5, slajd III

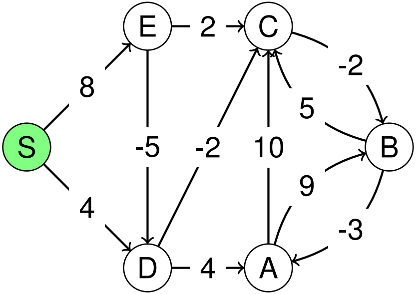
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $1 > 3 - 2$ – false;
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $1 > 8 + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $3 > 8 - 5$ – false;
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $3 > 0 + 4$ – false;
- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $8 > 0 + 8$ – false;

Proces relaksacji, iteracja 5, slajd IV

Input: *G*(*V*, *E*, *weight*), *src*.

1: ...
2: **for** i=1 **to** |V| - 1 **do**
3: **for all** (u, v) in E **do**
4: relax(u, v);
5: **end for**
6: **end for**
7: **for all** (u, v) in E **do**
8: **if** v.distance > u.distance + weight(u, v) **then**
9: "Istnieje ujemny cykl!"
10: **end if**
11: **end for**

W pętli l.2-6 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

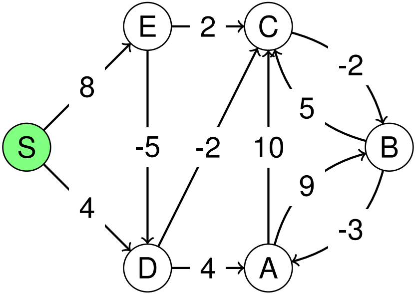


	s	a	b	c	d	e
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8
i=2	0	8	∞	2	3	8
i=3	0	7	0	1	3	8
i=4	0	-3	-1	1	3	8
i=5	0	-4	-1	1	3	8

Badanie istnienia ujemnych cykli - slajd I

Input: *G*(*V*, *E*, *weight*), *src*.

```
 1: ...
 2: for i=1 to |V| - 1 do
 3:     for all (u, v) in E do
 4:         relax(u, v);
 5:     end for
 6: end for
 7: for all (u, v) in E do
 8:     if v.distance > u.distance +
        weight(u, v) then
 9:         "Istnieje ujemny cykl!"
10:     end if
11: end for
```



s	a	b	c	d	e
0	-4	-1	1	3	8

W pętli l.7-11 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

Badanie istnienia ujemnych cykli - slajd II

Require: $G(V, E, weight)$, src.

```
1: ...
2: for all (u, v) in E do
3:     if v.distance > u.distance + weight(u, v) then
4:         "Istnieje ujemny cykl!"
5:     end if
6: end for
```

W pętli l.3-6 krawędzie rozważane są w porządku leksykograficznym: *ab*, *ac*, *ba*, *bc*, *cb*, *da*, *dc*, *ec*, *ed*, *sd*, *se*.

- Krawędź *ab*:
  $b.distance > a.distance + weight(a, b)$
  $-1 > -3 + 9$ – false;
- Krawędź *ac*:
  $c.distance > a.distance + weight(a, c)$
  $1 > -4 + 10$ – false;

Badanie istnienia ujemnych cykli - slajd III

- Krawędź *ba*:
  $a.distance > b.distance + weight(b, a)$
  $-4 > -1 - 3$ – false;
- Krawędź *bc*:
  $c.distance > b.distance + weight(b, c)$
  $1 > -1 + 5$ – false;
- Krawędź *cb*:
  $b.distance > c.distance + weight(c, b)$
  $-1 > 1 - 2$ – false;
- Krawędź *da*:
  $a.distance > d.distance + weight(d, a)$
  $-4 > 3 + 4$ – false;

Badanie istnienia ujemnych cykli - slajd IV

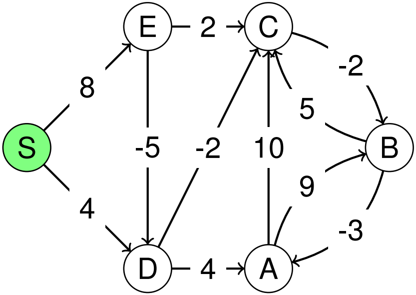
- Krawędź *dc*:
  $c.distance > d.distance + weight(d, c)$
  $1 > 3 - 2$ – false;
- Krawędź *ec*:
  $c.distance > e.distance + weight(e, c)$
  $1 > 8 + 2$ – false
- Krawędź *ed*:
  $d.distance > e.distance + weight(e, d)$
  $3 > 8 - 5$ – false;
- Krawędź *sd*:
  $d.distance > s.distance + weight(s, d)$
  $3 > 0 + 4$ – false;

Badanie istnienia ujemnych cykli - slajd V

- Krawędź *se*:
  $e.distance > s.distance + weight(s, e)$
  $8 > 0 + 8$ – false;

Wniosek: Brak ujemnych cykli !

Przykład - podsumowanie



	s	a	b	c	d	e
	∞	∞	∞	∞	∞	∞
	0	∞	∞	∞	∞	∞
i=1	0	∞	∞	∞	4	8
i=2	0	8	∞	2	3	8
i=3	0	7	0	1	3	8
i=4	0	-3	-1	1	3	8
i=5	0	-4	-1	1	3	8

Algorytm Dijkstry I

- **Algorytm Dijkstry** został opublikowany w 1959 roku i nazwany na cześć swojego twórcy holenderskiego informatyka Edsgera Dijkstry.
- **Algorytm Dijkstry** można zastosować do grafu z wagami $G(V, E, weight)$. Graf ten może być skierowany lub nieskierowany.
- **Algorytm Dijkstry** służy do wyznaczania najmniejszej odległości od ustalonego wierzchołka *scr* do wszystkich pozostałych w grafie.
- W odróżnieniu jednak od **Algorytmu Bellmana-Forda**, graf wejściowy nie może zawierać krawędzi o ujemnych wagach.

Algorytm Dijkstry II

- W algorytmie pamiętany jest zbiór wierzchołków Q, dla których nie obliczono jeszcze najkrótszych ścieżek, oraz wektor *distance*, odległości od wierzchołka *src* do pozostałych wierzchołków.
- Początkowo zbiór Q zawiera wszystkie wierzchołki, a wektor *distance* jest ustawiony na wartość "nieznana".
- Dopóki zbiór Q nie jest pusty wykonuj:
  - Pobierz ze zbioru Q wierzchołek v o najmniejszej wartości $v.distance$ i usuń go ze zbioru.
  - Dla każdego następnika u wierzchołka v sprawdź, czy $u.distance > v.distance + w((v, u))$, tzn. czy aktualne oszacowanie odległości do wierzchołka u jest większe od oszacowania odległości do wierzchołka v plus waga krawędzi (v, u).

Algorytm Dijkstry - pseudokod I

Require: Graf $G(V, E, weight)$ taki, że $weight(e) \geq 0$ dla każdego $e \in E$ oraz wierzchołek źródłowy $src \in V$.

Ensure: Parametr $v.distance$ dla każdego $v \in V$ taki że $v.distance$ jest równe minimalnej odległości od src do v.

$src.distance$ = 0 ;
$Q = \{src\}$;
for all v in $V \setminus \{src\}$ **do**
    $v.distance = \infty$;
    $Q = Q \cup \{v\}$;{Dodaj v do Q.}
end for
while $Q \neq \emptyset$ **do**
    v = wierzchołek z Q o najmniejszej wartości $v.distance$;
    $Q = Q \setminus \{v\}$; {Usuń v z Q.}
    for all u in $adj[v]$ **do**

Algorytm Dijkstry - pseudokod II

```
        if u.distance > v.distance + weight(v, u) then
            u.distance = v.distance + weight(v, u);
        end if
    end for
end while
return wartości distance dla każdego wierzchołka v ∈ V;
```

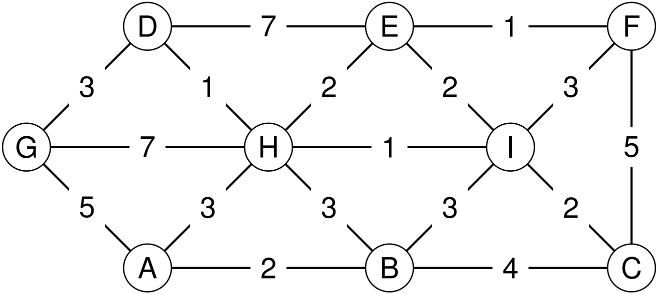
Złożoność[a]

[a] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. J. Assoc. Computer Machinery, 34(3):596–615,1987

Algorytm Dijkstry można zaimplementować (przy użyciu tak zwanych kopców Fibonacciego) w czasie $O(|V| \cdot log(|V|) + |E|)$.

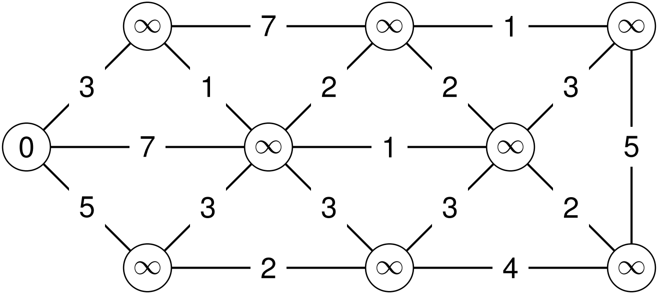
Slajd I

Przykład zaczerpnięty z `https://brilliant.org/wiki/dijkstras-short-path-finder/`.



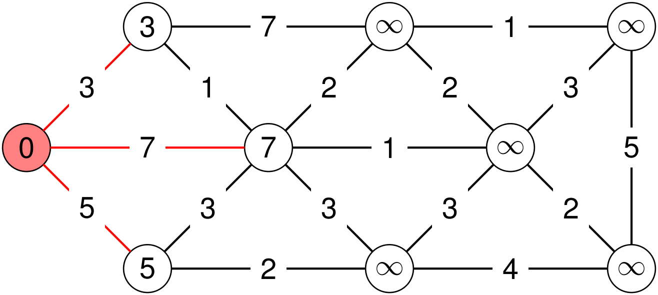
Slajd II

- Krok 1: Odległości zainicjalizowane zgodnie z algorytmem.
- $scr = G$
- $Q = \{A, B, C, D, E, F, G, H, I\}$.



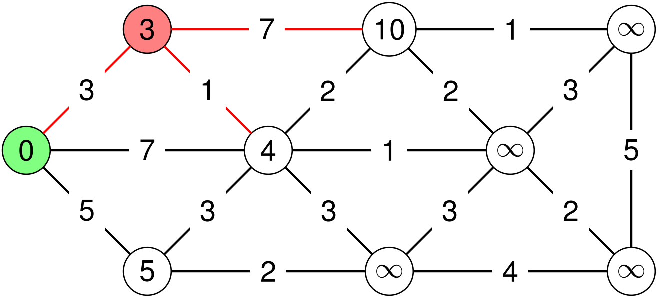
Slajd III

- Krok 2: Wybierz pierwszy wierzchołek i obliczyć odległości do jego sąsiadów.
- $v = G; Q = Q \setminus \{G\} = \{A, B, C, D, E, F, H, I\}$.
- relaksacja krawędzi incydentnych z G.



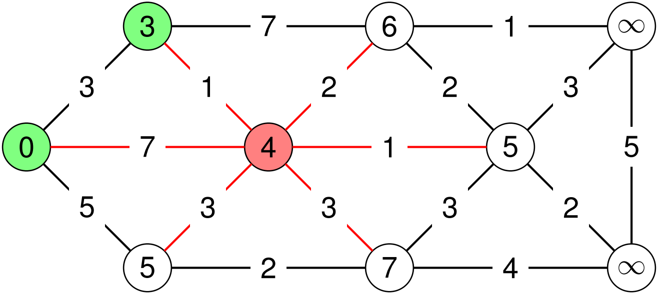
Slajd IV

- Krok 3: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = D; Q = Q \setminus \{D\} = \{A, B, C, E, F, H, I\}$.
- relaksacja krawędzi incydentnych z D.



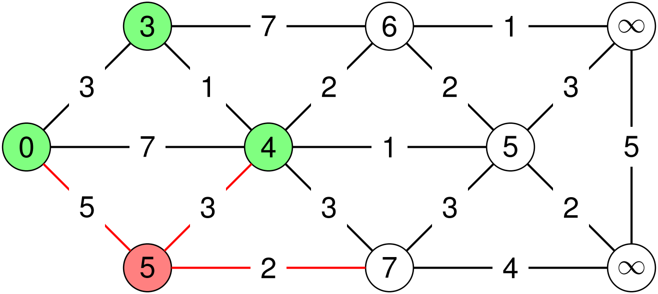
Slajd V

- Krok 4: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = H; Q = Q \setminus \{H\} = \{A, B, C, E, F, I\}$.
- relaksacja krawędzi incydentnych z H.



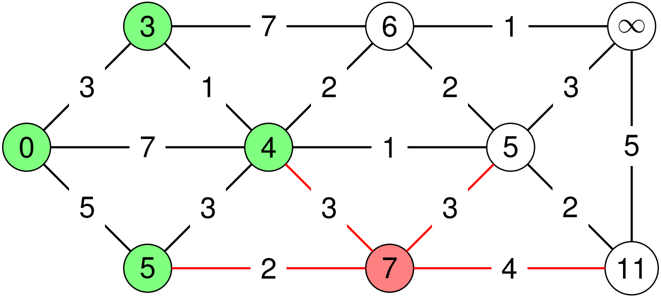
Slajd VI

- Krok 5: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = A$; $Q = Q \setminus \{A\} = \{B, C, E, F, I\}$.
- relaksacja krawędzi incydentnych z A.



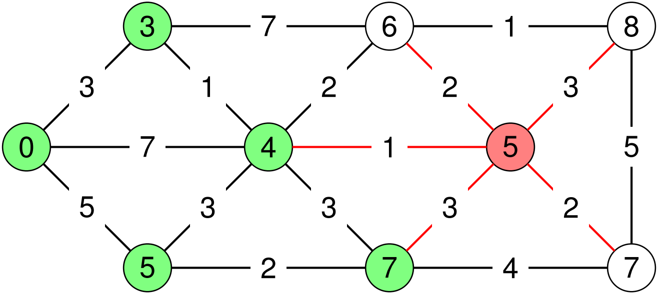
Slajd VII

- Krok 6: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = B$; $Q = Q \setminus \{B\} = \{C, E, F, I\}$.
- relaksacja krawędzi incydentnych z B.



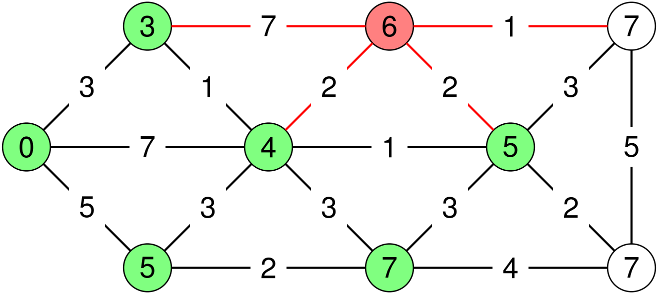
Slajd VIII

- Krok 7: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = I$; $Q = Q \setminus \{I\} = \{C, E, F\}$.
- relaksacja krawędzi incydentnych z I.



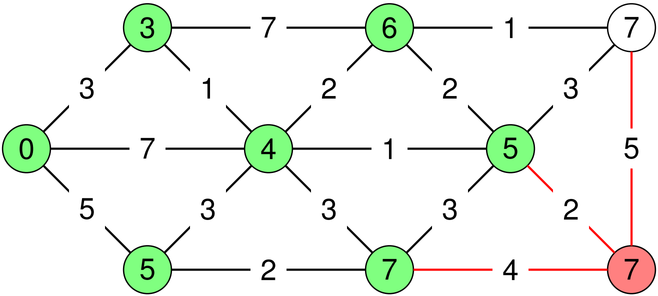
Slajd IX

- Krok 8: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = E$; $Q = Q \setminus \{E\} = \{C, F\}$.
- relaksacja krawędzi incydentnych z E.



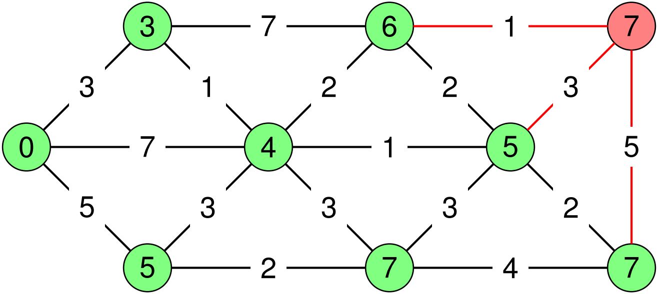
Slajd X

- Krok 9: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = C$; $Q = Q \setminus \{C\} = \{F\}$.
- relaksacja krawędzi incydentnych z C.



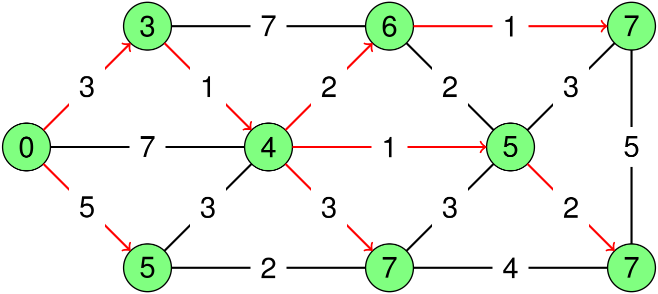
Slajd XI

- Krok 10: Wybieramy kolejny węzeł o minimalnej odległości; następnie powtarzamy obliczenia odległości dla sąsiadujących wierzchołków.
- $v = F$; $Q = Q \setminus \{F\} = \emptyset$.
- relaksacja krawędzi incydentnych z F.



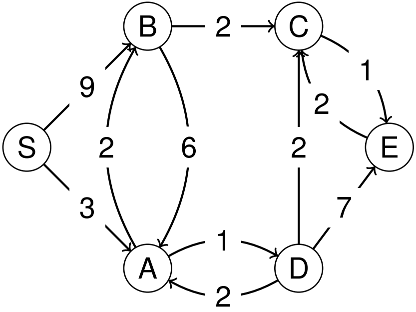
Slajd XII

Wynikowe drzewo najkrótszych ścieżek:

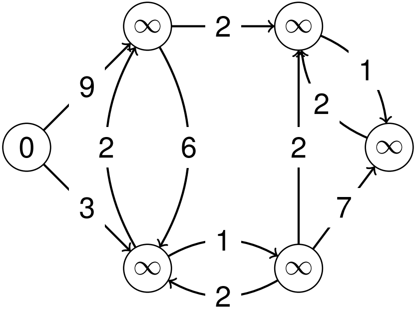


Slajd I

Dany jest graf:

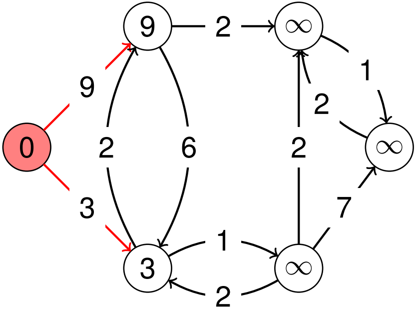


- Krok 1: Odległości zainicjalizowane zgodnie z algorytmem.
- *src* $= S$;
  $Q = \{S, A, B, C, D, E\}$

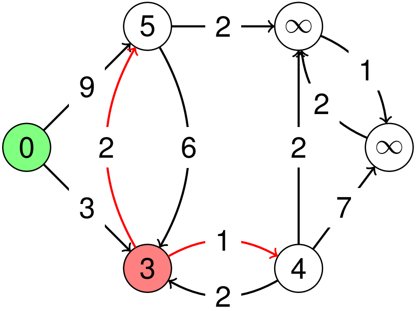


Slajd II

- Krok 2:

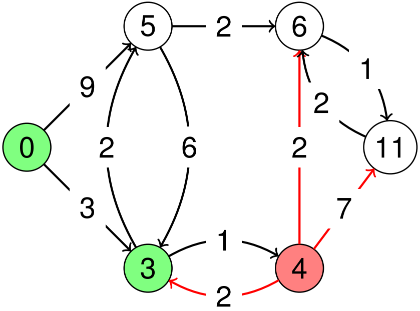


- Krok 3:

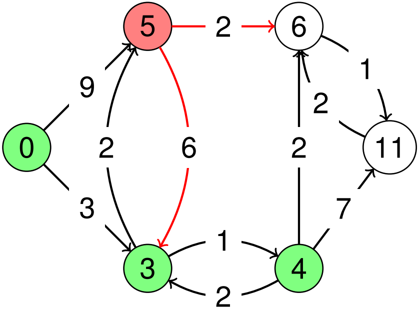


Slajd III

- Krok 4:

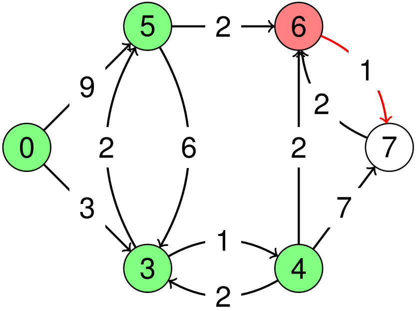


- Krok 5:

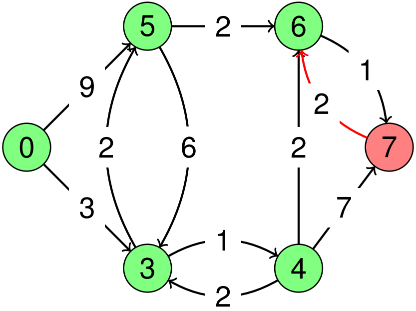


Slajd IV

- Krok 6:

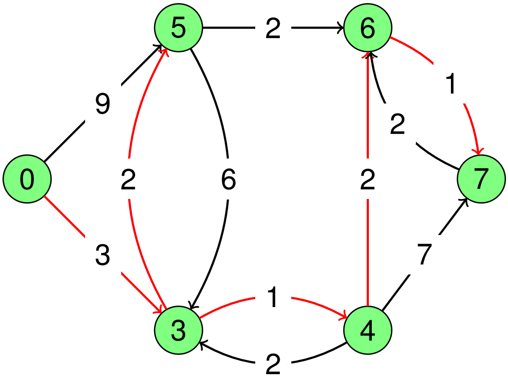


- Krok 7:



Slajd V

Wynikowe drzewo najkrótszych ścieżek:



Robert W. Floyd

- Robert W (Bob) Floyd, ur. 9.06.1936, zm. 26/09.2001 – amerykański informatyk.
- Znany z algorytmu Floyda-Warshalla, który efektywnie znajduje wszystkie najkrótsze ścieżki w grafie.
- Laureat nagrody Turinga nadawanej przez organizację ACM – 1978.
- Laureat nagrody Pioniera Informatyki, nadawanej przez ogranizację IEEE Computer Society – 1991
- Więcej na: `https://en.wikipedia.org/wiki/Robert_W._Floyd`

Stephen Warshall

- Stephen Warshall, ur. 15.11.1935, zm. 11.12.2006 – amerykański informatyk.
- Znany z algorytmu Floyda-Warshalla, który efektywnie znajduje wszystkie najkrótsze ścieżki w grafie.
- Podczas swojej kariery naukowej S. Warshall prowadził badania w zakresie systemów operacyjnych, projektowania kompilatorów, projektowania języków oraz analizy operacyjnej.
- Więcej na: `https://en.wikipedia.org/wiki/Stephen_Warshall`

Algorytm Floyda-Warshalla

- **Algorytm Floyda-Warshalla** został zaproponowany niezależnie przez **Roberta W. Floyda** i **Stephena Warshalla** w pracach:
  - R. W. Floyd. *Algorithm 97: shortest path*. Communications of the ACM, 5:345, 1962.
  - S. Warshall. A theorem on boolean matrices. J. ACM, 9:11–12, 1962
- **Algorytm Floyda-Warshalla**, podobnie jak algorytm Bellmana-Forda lub algorytm Dijkstry, oblicza najkrótszą ścieżkę w grafie.
- Algorytmy Bellmana-Forda oraz Dijkstry obliczają najkrótszą ścieżkę tylko z jednego źródła. Algorytm Floyda-Warshalla oblicza najkrótsze odległości pomiędzy każdą parą wierzchołków w grafie.

Slajd I

Input: Graf $G(V, E, weight)$.

Output: Miacierz D taka, że $D[i][j]$ jest najkrótszą odległością od wierzchołka i do j

```
for all D[i][j] ∈ D do
    if i == j then
        D[i][j] = 0;
    end if
    if (i, j) jest krawędzią z E then
        D[i][j] = weight(i, j);
    else
        D[i][j] = ∞;
    end if
end for
for k=1 to |V| do
```

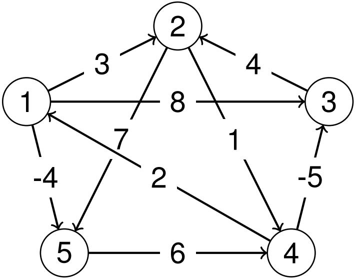
Slajd II

```
    for all i=1 to |V| do
        for all j=1 to |V| do
            if D[i][j] > D[i][k] + D[k][j] then
                D[i][j] = D[i][k] + D[k][j];
            end if
        end for
    end for
end for
```

Złożoność obliczeniowa I

- Algorytm Floyda-Warshalla można zaimplementować w czasie $O(|V|^3)$.
- Algorytm Floyda-Warshall jest zależny tylko od liczby wierzchołków w grafie. Czyni go to szczególnie użytecznym dla grafów gęstych, gdyż w ogóle nie zależy od liczby krawędzi.

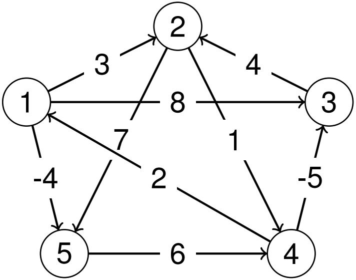
Slajd I



	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

Macierz D po wykonaniu linii 1–10 algorytmu Floyda-Warshalla.

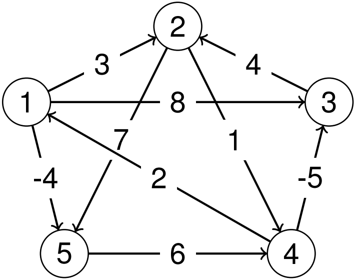
Slajd II



	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

Macierz D po wykonaniu linii 11–19 algorytmu Floyda-Warshalla, dla k=1.

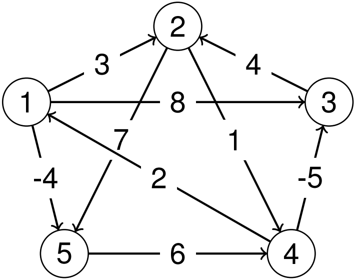
Slajd III



	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

Macierz D po wykonaniu linii 11–19 algorytmu Floyda-Warshalla, dla k=2.

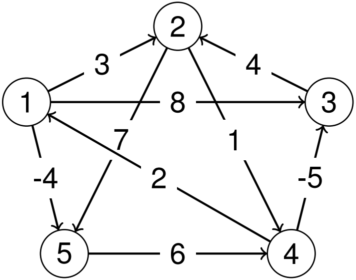
Slajd IV



	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

Macierz *D* po wykonaniu linii 11–19 algorytmu Floyda-Warshalla, dla k=3.

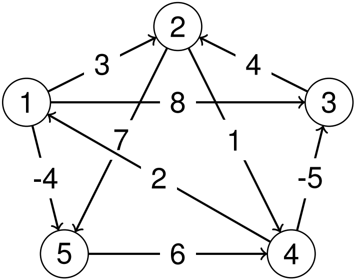
Slajd V



	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Macierz *D* po wykonaniu linii 11–19 algorytmu Floyda-Warshalla, dla k=4.

Slajd VI



	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Macierz *D* po wykonaniu linii 11–19 algorytmu Floyda-Warshalla, dla k=5.

Wprowadzenie, slajd I

- **Algorytm Johnsona** podobnie jak algorytm Floyda-Warshalla zajmuje się znalezieniem najkrótszych ścieżek dla wszystkich par wierzchołków.
- **Algorytm Floyda-Warshalla** jest najbardziej efektywny w przypadku grafów gęstych (wiele krawędzi), natomiast gdy **algorytm Johnsona** jest najbardziej skuteczny w przypadku grafów rzadkich (niewiele krawędzi).
- Powodem, dla którego **algorytm Johnsona** jest lepszy dla grafów rzadkich, jest to, że jego złożoność czasowa zależy również od liczby krawędzi w grafie, podczas gdy w algorytmie Floyda-Warshalla nie.

Wprowadzenie, slajd II

- Algorytm Johnsona działa w czasie $O(|V|^2 \cdot log(|V|) + |V| \cdot |E|)$. Zatem, jeśli liczba krawędzi jest mała (tzn. graf jest rzadki), będzie on działał szybciej niż algorytm Floyd-Warshall, który działa w czasie $O(|V|^3)$.

Algorytm Johnsona składa się z trzech głównych kroków.

1. Dodajemy nowy wierzchołek do grafu i łączymy go krawędziami o zerowej wadze z wszystkimi pozostałymi wierzchołkami grafu.
2. Wykonujemy dla wszystkich krawędzi proces ponownego ważenia, który eliminuje ujemne wagi, przy pomocy algorytmu Bellmana-Forda.
3. Usuwamy wierzchołek dodany w kroku 1 i uruchamiamy algorytm Dijkstry dla każdego węzła w grafie.

Slajd I

Require: Graf $G(V, E, weight)$.

Ensure: Miacierz D taka, że $D[i][j]$ jest najkrótszą odległością od wierzchołka i do j

Utwórz G' taki, że $G'.V = G.V + \{s\}$, $G'.E = G.E + \{(s, u)|$ dla każdego $u \in G.V\}$ oraz $weight(s, u) = 0$ dla każdego $u \in G.V$

if $Bellman - Ford(G', s) == False$ **then**

    return Graf ma ujemny cykl.

else

    for all $v \in G'.V$ **do**

        $h(v) = distance(s, v)$ obliczony przez algorytm Bellmana-Forda

    end for

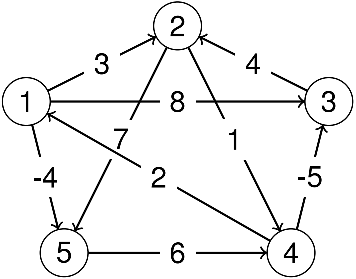
    for all $(u, v) \in G'.E$ **do**

        $weight'(u, v) = weight(u, v) + h(u) - h(v)$;

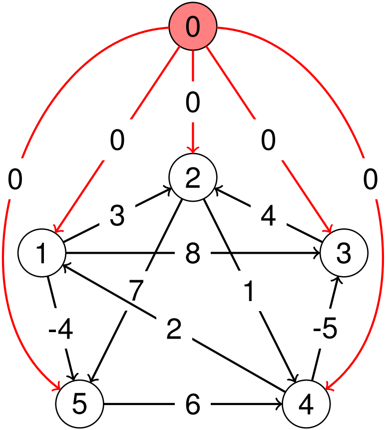
Slajd II

end for
for all $D[i][j] \in D$ **do**
  $D[i][j] = \infty$;
end for
for all $u \in G.V$ **do**
  wykonaj $Dijkstra(G, weight', u)$ aby obliczyć $distance'(u, v)$ dla każdego $v \in G.V$.
  for all $v \in G.V$ **do**
    $D[u][v] = distance'(u, v) + h(v) - h(u)$;
  end for
end for
end if

Linia 1 algorytmu.



Graf wejściowy G.



Graf G' z dodanym wierzchołkiem i krawędziami. L. 1 algorytmu.

Linie 2-7 algorytmu. Slajd I

- Wiemy, że graf nie ma ujemnych cykli.
- Poniżej wykonanie L. 5-7 algorytmu.
- Wierzchołek źródłowy: **0**.
- Lista krawędzi: $(0,1)$, $(0,2)$, $(0,3)$, $(0,4)$, $(0,5)$, $(1,2)$, $(1,3)$, $(1,5)$. $(2,4)$, $(2,5)$, $(3,2)$, $(4,1)$, $(4,3)$, $(5,4)$.
- Relaksacja, wykonana $|V'| - 1 = 5$ razy:

1. i=1:

0	1	2	3	4	5
0	∞	∞	∞	∞	∞

   - $(0,1)$: $1.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0,1)$. $\infty > 0 + 0$. $1.\mathit{distance} = 0$.

Linie 2-7 algorytmu. Slajd II

- $(0, 2)$: $2.distance > 0.distance + weight(0, 2)$.
  $\infty > 0 + 0$. $2.distance = 0$.
- $(0, 3)$: $3.distance > 0.distance + weight(0, 3)$.
  $\infty > 0 + 0$. $3.distance = 0$.
- $(0, 4)$: $4.distance > 0.distance + weight(0, 4)$.
  $\infty > 0 + 0$. $4.distance = 0$.
- $(0, 5)$: $5.distance > 0.distance + weight(0, 5)$.
  $\infty > 0 + 0$. $5.distance = 0$.
- $(1, 2)$: $2.distance > 1.distance + weight(1, 2)$.
  $0 > 0 + 3$. brak korekty.
- $(1, 3)$: $3.distance > 1.distance + weight(1, 3)$.
  $0 > 0 + 8$. brak korekty.
- $(1, 5)$: $5.distance > 1.distance + weight(1, 5)$.
  $0 > 0 + (-4)$. $5.distance = -4$.
- $(2, 4)$: $4.distance > 2.distance + weight(2, 4)$.
  $0 > 0 + 1$. brak korekty.

Linie 2-7 algorytmu. Slajd III

- $(2,5)$: $5.\mathit{distance} > 2.\mathit{distance} + \mathit{weight}(2,5)$. $-4 > 0 + 7$. brak korekty.
- $(3,2)$: $2.\mathit{distance} > 3.\mathit{distance} + \mathit{weight}(3,2)$. $0 > 0 + 4$. brak korekty.
- $(4,1)$: $1.\mathit{distance} > 4.\mathit{distance} + \mathit{weight}(4,1)$. $0 > 0 + 2$. brak korekty.
- $(4,3)$: $3.\mathit{distance} > 4.\mathit{distance} + \mathit{weight}(4,3)$. $0 > 0 + (-5)$. $3.\mathit{distance} = -5$.
- $(5,4)$: $4.\mathit{distance} > 5.\mathit{distance} + \mathit{weight}(5,4)$. $0 > -4 + 6 = 2$. brak korekty.

2. i=2:

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

- $(0,1)$: $1.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0,1)$. $0 > 0 + 0 = 0$. brak korekty.

Linie 2-7 algorytmu. Slajd IV

- $(0,2)$: $2.distance > 0.distance + weight(0,2)$.
  $0 > 0 + 0 = 0$. brak korekty.
- $(0,3)$: $3.distance > 0.distance + weight(0,3)$.
  $-5 > 0 + 0 = 0$. brak korekty.
- $(0,4)$: $4.distance > 0.distance + weight(0,4)$.
  $0 > 0 + 0 = 0$. brak korekty.
- $(0,5)$: $5.distance > 0.distance + weight(0,5)$.
  $-4 > 0 + 0 = 0$. brak korekty.
- $(1,2)$: $2.distance > 1.distance + weight(1,2)$.
  $0 > 0 + 3 = 3$. brak korekty.
- $(1,3)$: $3.distance > 1.distance + weight(1,3)$.
  $-5 > 0 + 8 = 8$. brak korekty.
- $(1,5)$: $5.distance > 1.distance + weight(1,5)$.
  $-4 > 0 - 4 = -4$. brak korekty.
- $(2,4)$: $4.distance > 2.distance + weight(2,4)$.
  $0 > 0 + 1 = 1$. brak korekty.

Linie 2-7 algorytmu. Slajd V

- $(2,5)$: $5.\textit{distance} > 2.\textit{distance} + \textit{weight}(2,5)$.
  $-4 > 0 + 7 = 7$. brak korekty.
- $(3,2)$: $2.\textit{distance} > 3.\textit{distance} + \textit{weight}(3,2)$.
  $0 > -5 + 4 = -1$. $2.\textit{distance} = -1$.
- $(4,1)$: $1.\textit{distance} > 4.\textit{distance} + \textit{weight}(4,1)$.
  $0 > 0 + 2 = 2$. brak korekty.
- $(4,3)$: $3.\textit{distance} > 4.\textit{distance} + \textit{weight}(4,3)$.
  $-5 > 0 - 5 = -5$. brak korekty.
- $(5,4)$: $4.\textit{distance} > 5.\textit{distance} + \textit{weight}(5,4)$.
  $0 > -4 + 6 = 2$. brak korekty.

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

3. i=3:
   - $(0,1)$: $1.\textit{distance} > 0.\textit{distance} + \textit{weight}(0,1)$.
     $0 > 0 + 0 = 0$. brak korekty.

Linie 2-7 algorytmu. Slajd VI

- $(0, 2)$: $2.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0, 2)$.
  $-1 > 0 + 0 = 0$. brak korekty.
- $(0, 3)$: $3.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0, 3)$.
  $-5 > 0 + 0 = 0$. brak korekty.
- $(0, 4)$: $4.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0, 4)$.
  $0 > 0 + 0 = 0$. brak korekty.
- $(0, 5)$: $5.\mathit{distance} > 0.\mathit{distance} + \mathit{weight}(0, 5)$.
  $-4 > 0 + 0 = 0$. brak korekty.
- $(1, 2)$: $2.\mathit{distance} > 1.\mathit{distance} + \mathit{weight}(1, 2)$.
  $-1 > 0 + 3 = 3$. brak korekty.
- $(1, 3)$: $3.\mathit{distance} > 1.\mathit{distance} + \mathit{weight}(1, 3)$.
  $-5 > 0 + 8 = 8$. brak korekty.
- $(1, 5)$: $5.\mathit{distance} > 1.\mathit{distance} + \mathit{weight}(1, 5)$.
  $-4 > 0 - 4 = -4$. brak korekty.
- $(2, 4)$: $4.\mathit{distance} > 2.\mathit{distance} + \mathit{weight}(2, 4)$.
  $0 > -1 + 1 = 0$. brak korekty.

Linie 2-7 algorytmu. Slajd VII

- $(2,5)$: $5.\mathit{distance} > 2.\mathit{distance} + \mathit{weight}(2,5)$. $-4 > -1 + 7 = 6$. brak korekty.
- $(3,2)$: $2.\mathit{distance} > 3.\mathit{distance} + \mathit{weight}(3,2)$. $-1 > -5 + 4 = -1$. brak korekty.
- $(4,1)$: $1.\mathit{distance} > 4.\mathit{distance} + \mathit{weight}(4,1)$. $0 > 0 + 2 = 2$. brak korekty.
- $(4,3)$: $3.\mathit{distance} > 4.\mathit{distance} + \mathit{weight}(4,3)$. $-5 > 0 - 5 = -5$. brak korekty.
- $(5,4)$: $4.\mathit{distance} > 5.\mathit{distance} + \mathit{weight}(5,4)$. $0 > -4 + 6 = 2$. brak korekty.

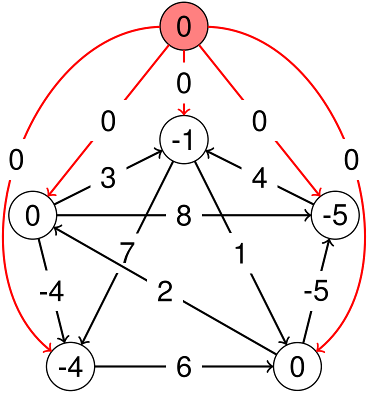
0	1	2	3	4	5
0	0	-1	-5	0	-4

4. i=4, i=5: nic nie zmieni.

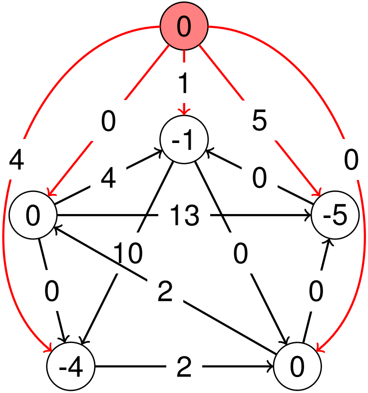
Linie 8-10 algorytmu. Slajd I

- $weight'(0,1) = weight(0,1) + h(0) - h(1) = 0 + 0 - 0 = 0,$
- $weight'(0,2) = weight(0,2) + h(0) - h(2) = 0 + 0 - (-1) = 1,$
- $weight'(0,3) = weight(0,3) + h(0) - h(3) = 0 + 0 - (-5) = 5,$
- $weight'(0,4) = weight(0,4) + h(0) - h(4) = 0 + 0 - 0 = 0,$
- $weight'(0,5) = weight(0,5) + h(0) - h(5) = 0 + 0 - (-4) = 4,$
- $weight'(1,2) = weight(1,2) + h(1) - h(2) = 3 + 0 - (-1) = 4,$
- $weight'(1,3) = weight(1,3) + h(1) - h(3) = 8 + 0 - (-5) = 13,$
- $weight'(1,5) = weight(1,5) + h(1) - h(5) = -4 + 0 - (-4) = 0,$
- $weight'(2,4) = weight(2,4) + h(2) - h(4) = 1 - 1 - 0 = 0,$
- $weight'(2,5) = weight(2,5) + h(2) - h(5) = 7 - 1 - (-4) = 10,$
- $weight'(3,2) = weight(3,2) + h(3) - h(2) = 4 - 5 - (-1) = 0,$
- $weight'(4,1) = weight(4,1) + h(4) - h(1) = 2 + 0 - 0 = 2,$
- $weight'(4,3) = weight(4,3) + h(4) - h(3) = -5 + 0 - (-5) = 0,$
- $weight'(5,4) = weight(5,4) + h(5) - h(4) = 6 - 4 - 0 = 2.$

Linie 8-10 algorytmu. Slajd II



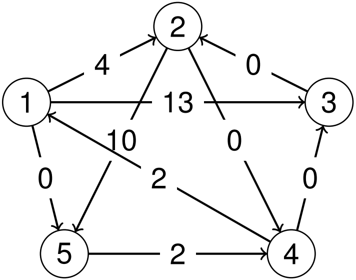
Graf G’ z aktualizacją wagi w wierzchołkach po uruchomieniu algorytmu Bellmana-Forda dla wierzchołka **0**.



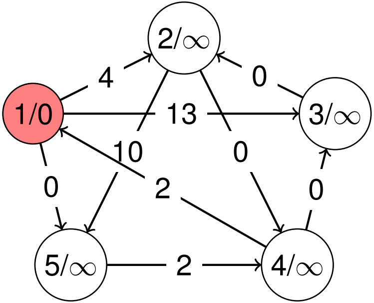
Korekta wag na krawędziach po uruchomieniu algorytmu Bellmana-Forda dla każdego wierzchołka w G’.

Linie 11-19 algorytmu. Slajd I

Teraz dla każdego wierzchołka należy wykonać algorytm Dijkstry. Niech wierzchołkiem źródłowym będzie: **1**

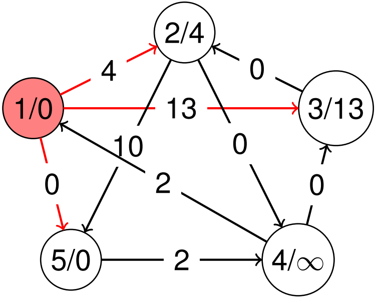


Graf wejściowy

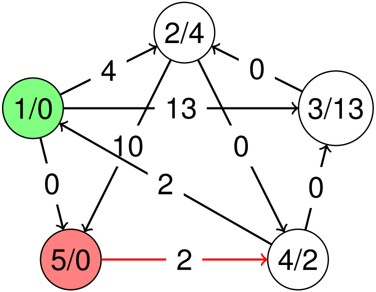


Krok 1. Etykieta x/y oznacza: x – nr wierzchołka, y – aktualną odległość od wierzchołka źródłowego.

Linie 11-19 algorytmu. Slajd II

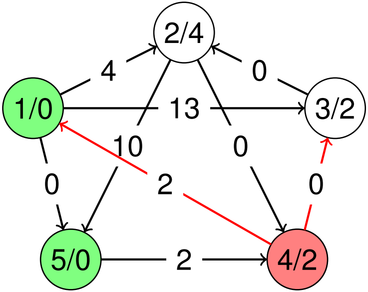


Krok 2:

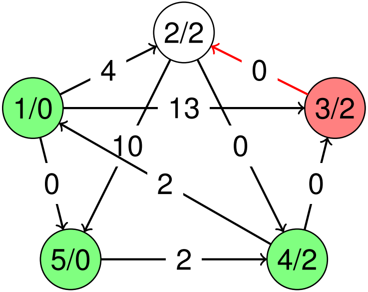


Krok 3:

Linie 11-19 algorytmu. Slajd III

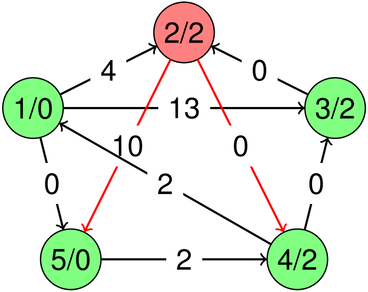


Krok 4:

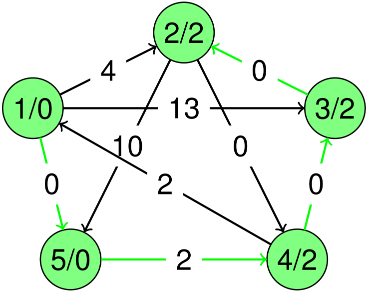


Krok 5:

Linie 11-19 algorytmu. Slajd IV



Krok 6:



Krok 7:

Linie 11-19 algorytmu. Slajd V

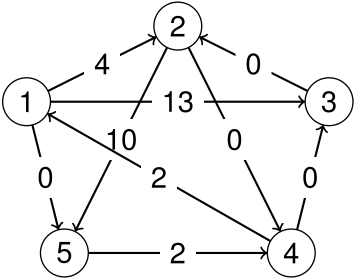
Obliczamy 1 wiersz macierz $D_{5\times5}$:

- $D[1][1] = distance'(1,1) = 0$
- $D[1][2] = distance'(1,2) + h(2) - h(1) = 2 + (-1) - 0 = 1$
- $D[1][3] = distance'(1,3) + h(3) - h(1) = 2 + (-5) - 0 = -3$
- $D[1][4] = distance'(1,4) + h(4) - h(1) = 2 + 0 - 0 = 2$
- $D[1][5] = distance'(1,5) + h(5) - h(1) = 0 + (-4) - 0 = -4$

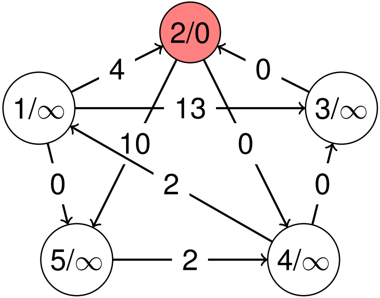
	1	2	3	4	5
1	0	1	-3	2	-4

Linie 11-19 algorytmu. Slajd VI

Teraz dla każdego wierzchołka należy wykonać algorytm Dijkstry.
Niech wierzchołkiem źródłowym będzie: **2**

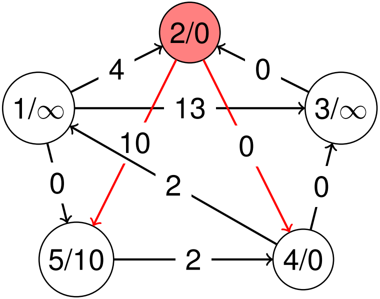


Graf wejściowy

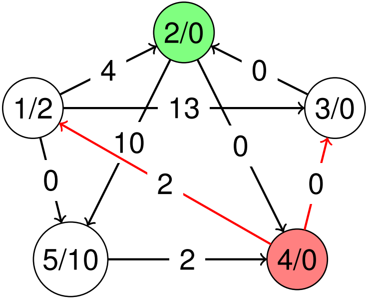


Krok 1. Etykieta *x*/*y* oznacza: *x* – nr wierzchołka, *y* – aktualną odległość od wierzchołka źródłowego.

Linie 11-19 algorytmu. Slajd VII

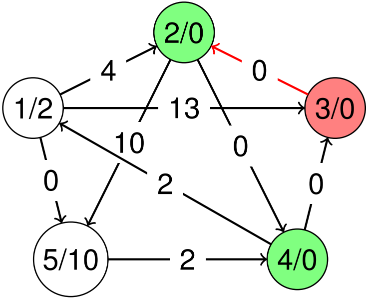


Krok 2.

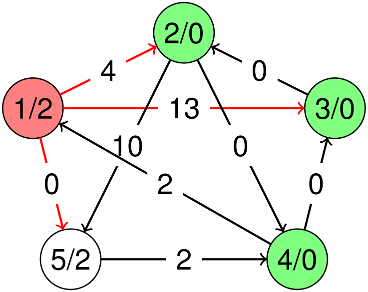


Krok 3.

Linie 11-19 algorytmu. Slajd VIII

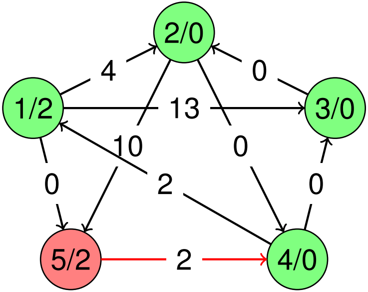


Krok 4.

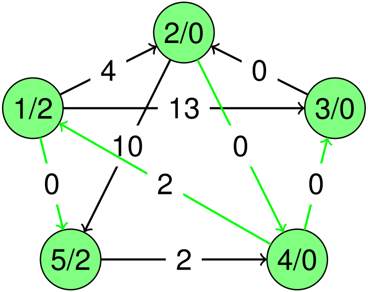


Krok 5.

Linie 11-19 algorytmu. Slajd IX



Krok 6.



Krok 7.

Linie 11-19 algorytmu. Slajd X

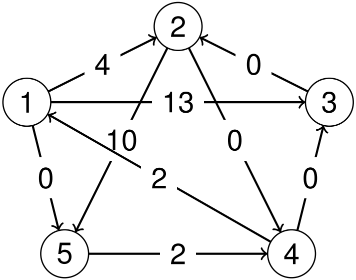
Obliczamy 2 wiersz macierz $D_{5\times 5}$:

- $D[2][1] = distance'(2,1) + h(1) - h(2) = 2 + 0 - (-1) = 3$
- $D[2][2] = distance'(2,2) + h(2) - h(2) = 0$
- $D[2][3] = distance'(2,3) + h(3) - h(2) = 0 + (-5) - (-1) = -4$
- $D[2][4] = distance'(2,4) + h(4) - h(2) = 0 + 0 - (-1) = 1$
- $D[2][5] = distance'(2,5) + h(5) - h(2) = 2 + (-4) - (-1) = -1$

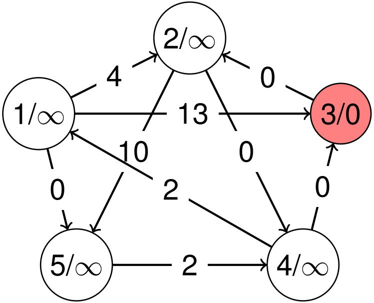
	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1

Linie 11-19 algorytmu. Slajd XI

Teraz dla każdego wierzchołka należy wykonać algorytm Dijkstry.
Niech wierzchołkiem źródłowym będzie: **3**

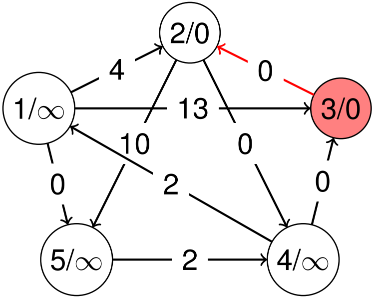


Graf wejściowy

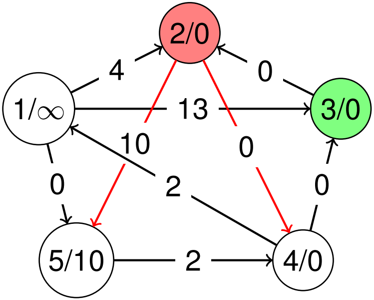


Krok 1. Etykieta x/y oznacza: x – nr wierzchołka, y – aktualną odległość od wierzchołka źródłowego.

Linie 11-19 algorytmu. Slajd XII

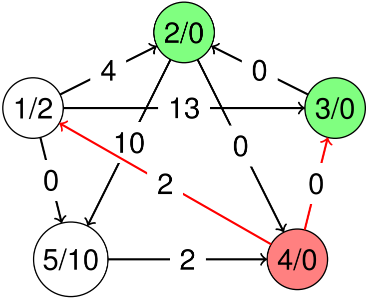


Krok 2.

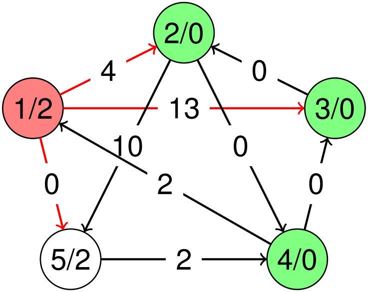


Krok 3.

Linie 11-19 algorytmu. Slajd XIII

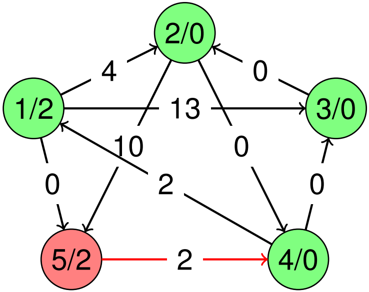


Krok 4.

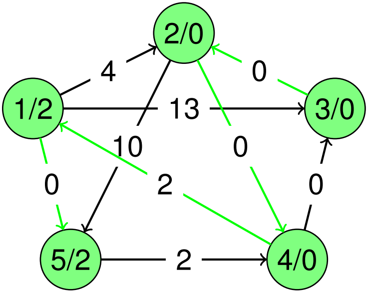


Krok 5.

Linie 11-19 algorytmu. Slajd XIV



Krok 6.



Krok 7.

Linie 11-19 algorytmu. Slajd XV

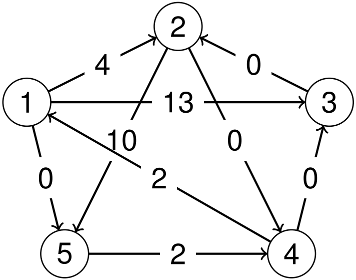
Obliczamy 3 wiersz macierz $D_{5\times 5}$:

- $D[3][1] = distance'(3,1) + h(1) - h(3) = 2 + 0 - (-5) = 7$
- $D[3][2] = distance'(3,2) + h(2) - h(3) = 0 + (-1) - (-5) = 4$
- $D[3][3] = distance'(3,3) + h(3) - h(3) = 0$
- $D[3][4] = distance'(3,4) + h(4) - h(3) = 0 + 0 - (-5) = 5$
- $D[3][5] = distance'(3,5) + h(5) - h(3) = 2 + (-4) - (-5) = 3$

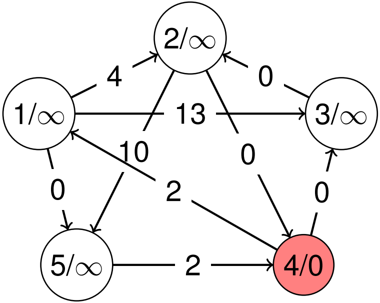
	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3

Linie 11-19 algorytmu. Slajd XVI

Teraz dla każdego wierzchołka należy wykonać algorytm Dijkstry.
Niech wierzchołkiem źródłowym będzie: **4**

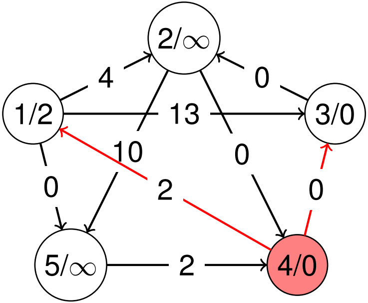


Graf wejściowy

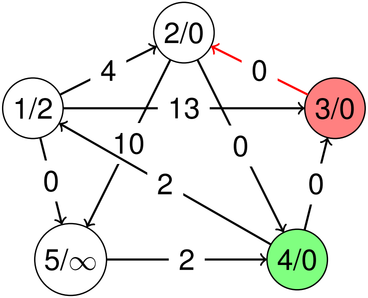


Krok 1. Etykieta x/y oznacza: x – nr wierzchołka, y – aktualną odległość od wierzchołka źródłowego.

Linie 11-19 algorytmu. Slajd XVII

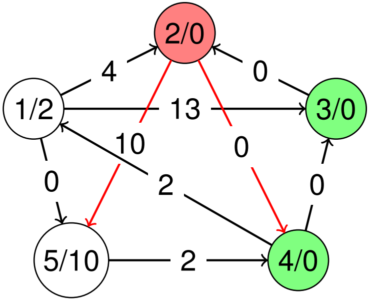


Krok 2.

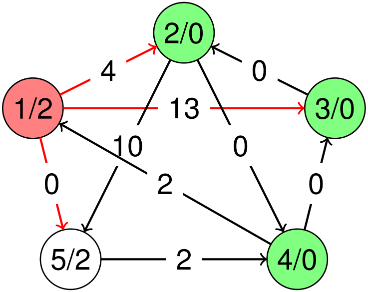


Krok 3.

Linie 11-19 algorytmu. Slajd XVIII

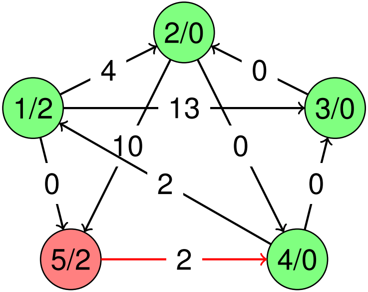


Krok 4.

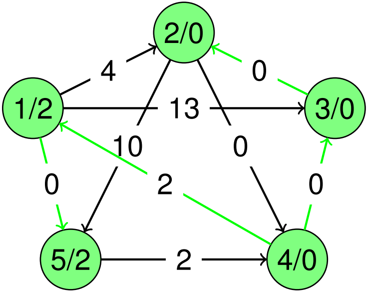


Krok 5.

Linie 11-19 algorytmu. Slajd XIX



Krok 6.



Krok 7.

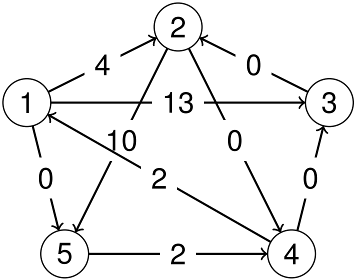
Linie 11-19 algorytmu. Slajd XX

Obliczamy 4 wiersz macierz $D_{5\times 5}$:

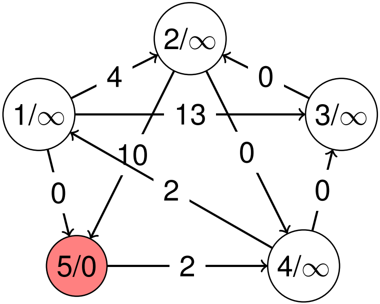
- $D[4][1] = distance'(4,1) + h(1) - h(4) = 2 + 0 - 0 = 2$
- $D[4][2] = distance'(4,2) + h(2) - h(4) = 0 + (-1) - 0 = -1$
- $D[4][3] = distance'(4,3) + h(3) - h(4) = 0 + (-5) - 0 = -5$
- $D[4][4] = distance'(4,4) + h(4) - h(4) = 0 + 0 - 0 = 0$
- $D[4][5] = distance'(4,5) + h(5) - h(4) = 2 + (-4) - 0 = -2$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2

Linie 11-19 algorytmu. Slajd XXI

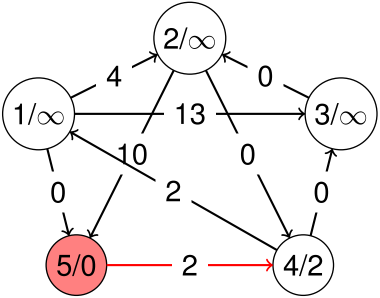


Graf wejściowy

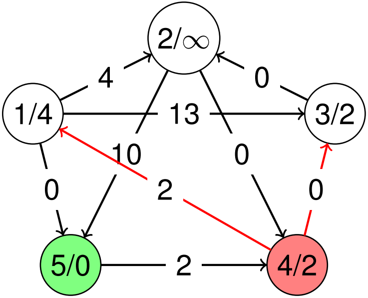


Krok 1. Etykieta x/y oznacza: x – nr wierzchołka, y – aktualną odległość od wierzchołka źródłowego.

Linie 11-19 algorytmu. Slajd XXII

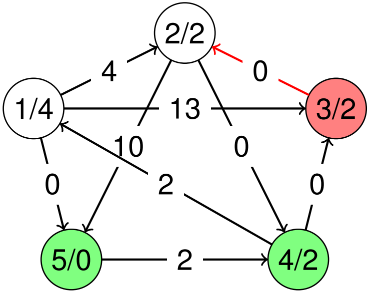


Krok 2.

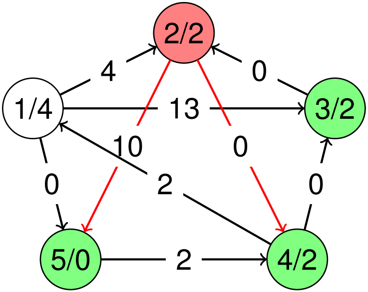


Krok 3.

Linie 11-19 algorytmu. Slajd XXIII

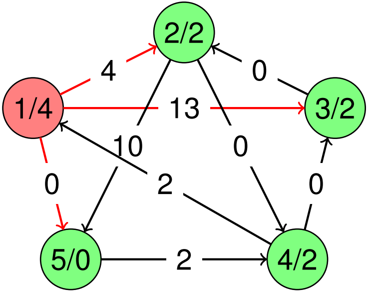


Krok 4.

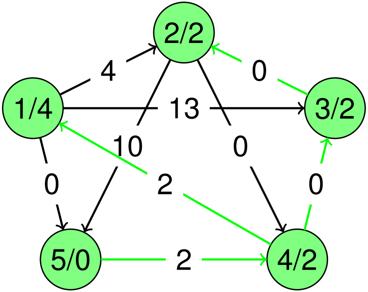


Krok 5.

Linie 11-19 algorytmu. Slajd XXIV



Krok 6.



Krok 7.

Linie 11-19 algorytmu. Slajd XXV

Obliczamy 5 wiersz macierz $D_{5\times5}$:

- $D[5][1] = distance'(5,1) + h(1) - h(5) = 4 + 0 - (-4) = 8$
- $D[5][2] = distance'(5,2) + h(2) - h(5) = 2 + (-1) - (-4) = 5$
- $D[5][3] = distance'(5,3) + h(3) - h(5) = 2 + (-5) - (-4) = 1$
- $D[5][4] = distance'(5,4) + h(4) - h(5) = 2 + 0 - (-4) = 6$
- $D[5][5] = distance'(5,5) + h(5) - h(5) = 0$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0