MATEMATYKA W KONKURSACH ALGORYTMICZNO-PROGRAMISTYCZNYCH

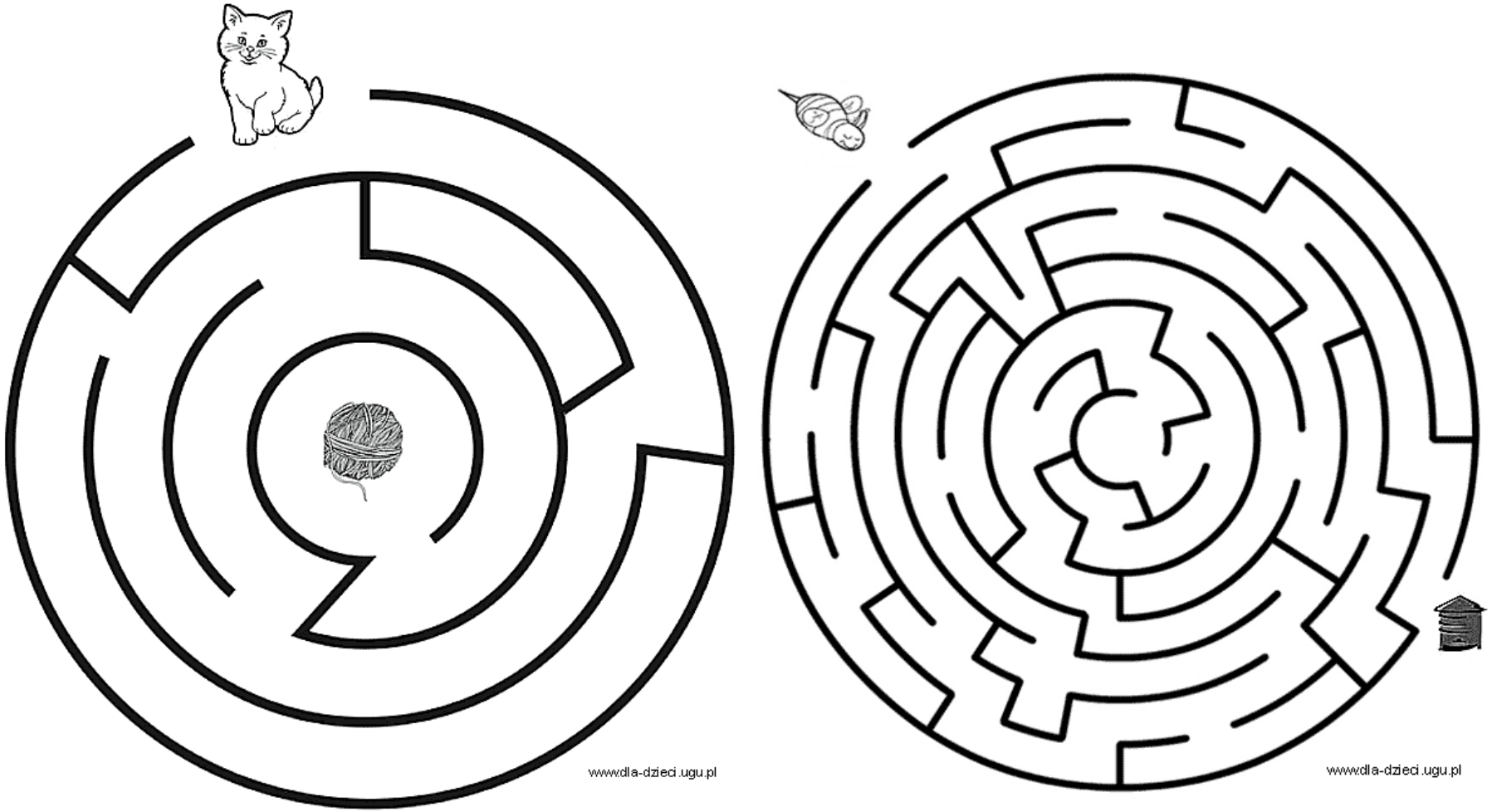
OTWARTY WEB-KURS

Webinarium przeprowadzone
w ramach projektu
"Mistrzostwa w Algorytmice
i Programowaniu - Uczniowie",
finansowanego przez:

Wstęp do algorytmicznej teorii grafów

Krzysztof Diks

Problem 1: Labirynt



Źródło: www.dla-dzieci.ugu.pl

Problem 2: Wilk, owca i kapusta

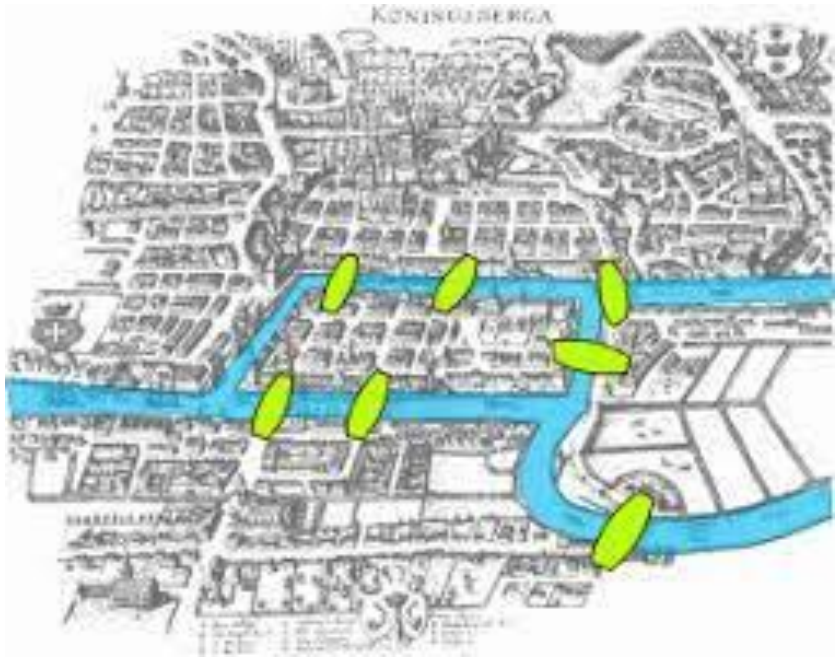
Źródło: https://play.google.com/store/apps/details?id=air.com.web4games.wolfsheepandcabbage

Problem 3: Przelewanie miodu

Dwa niedźwiadki znalazły składzik z miodem. Był w nim 8-litrowy słój pełen miodu i dwa słoje puste, 5-litrowy i 3-litrowy. W jaki sposób korzystając tylko z tych słoi podzielić równo miód?

Źródło rysunku: http://slon84.w.interia.pl/

Problem 4: Mosty w Królewcu



Leonard Euler, 1707 - 1783

Źródło: Seven Bridges of Koenigsberg – Wikipedia en.wikipedia.org

Problem 5: generowanie permutacji

Czy można ustawić permutacje zbioru {1, 2, …, n} w ciąg cykliczny w taki sposób, że każde dwie kolejne permutacje w ciągu różnią się zamianą tylko dwóch sąsiednich elementów?

Przykład dla n = 3

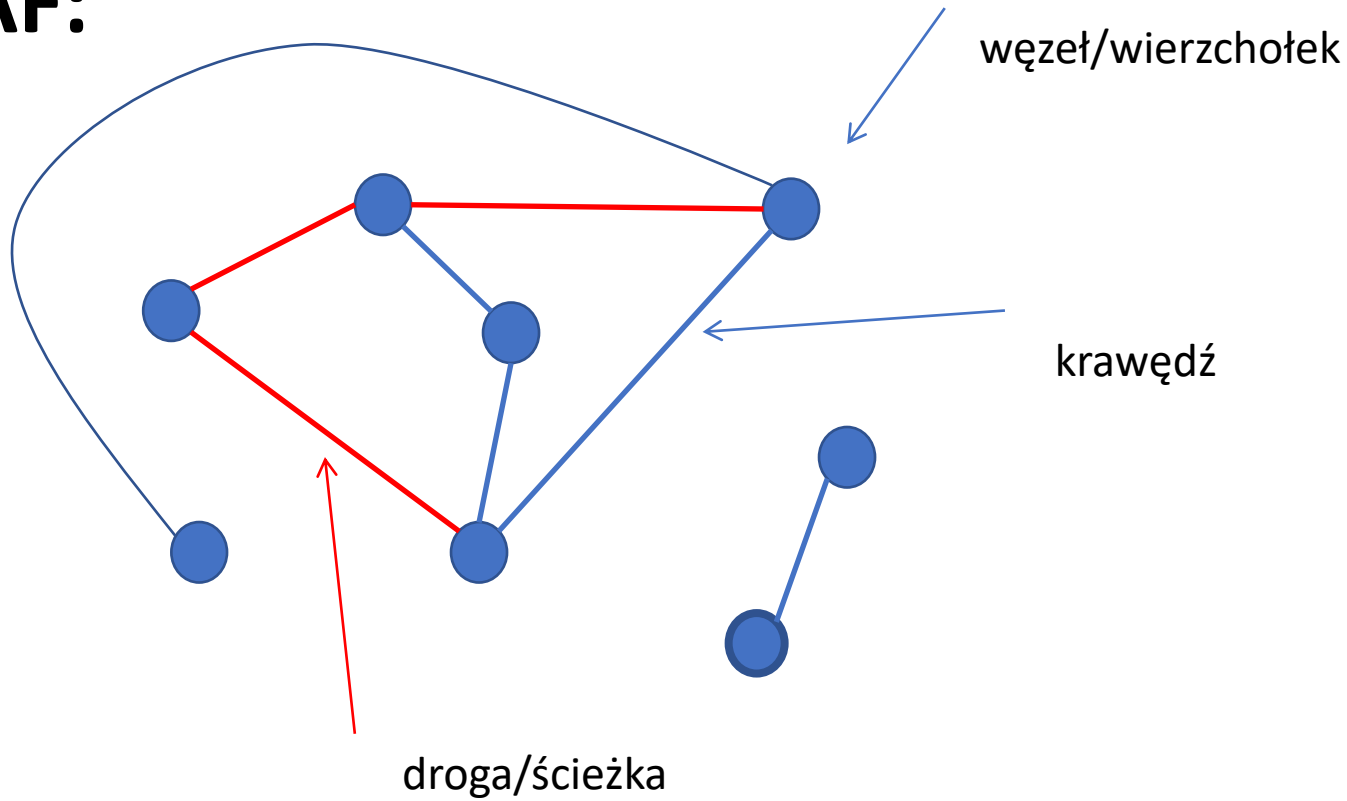
[1,2,3], [2,1,3], [2,3,1], [3,2,1], [3,1,2], [1,3,2]

Problem 6: najkrótsza trasa z Rotterdamu do Groningen

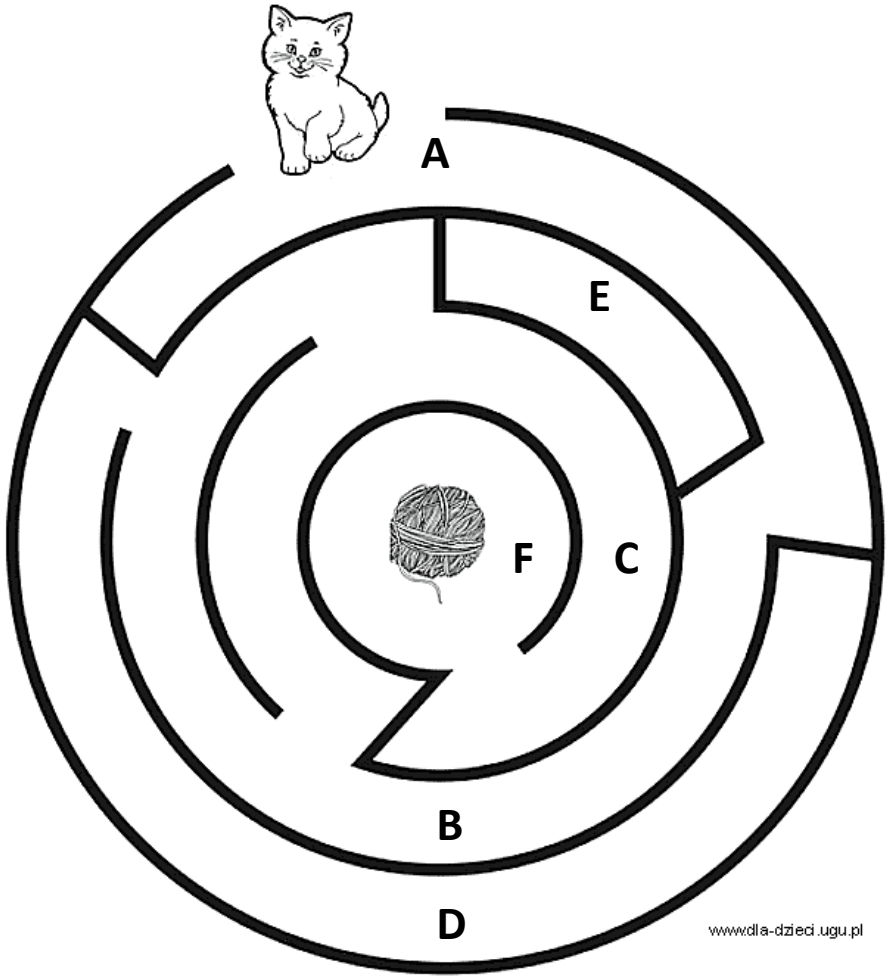
Źródło: mapy Google

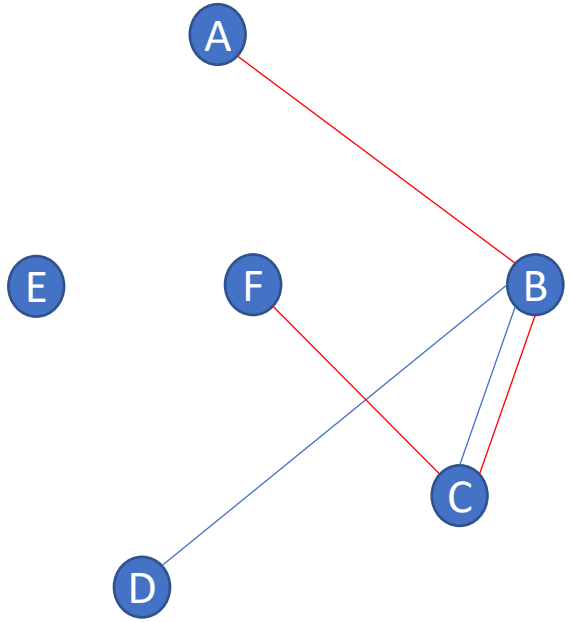
Co łączy wszystkie wspomniane problemy?

GRAF:

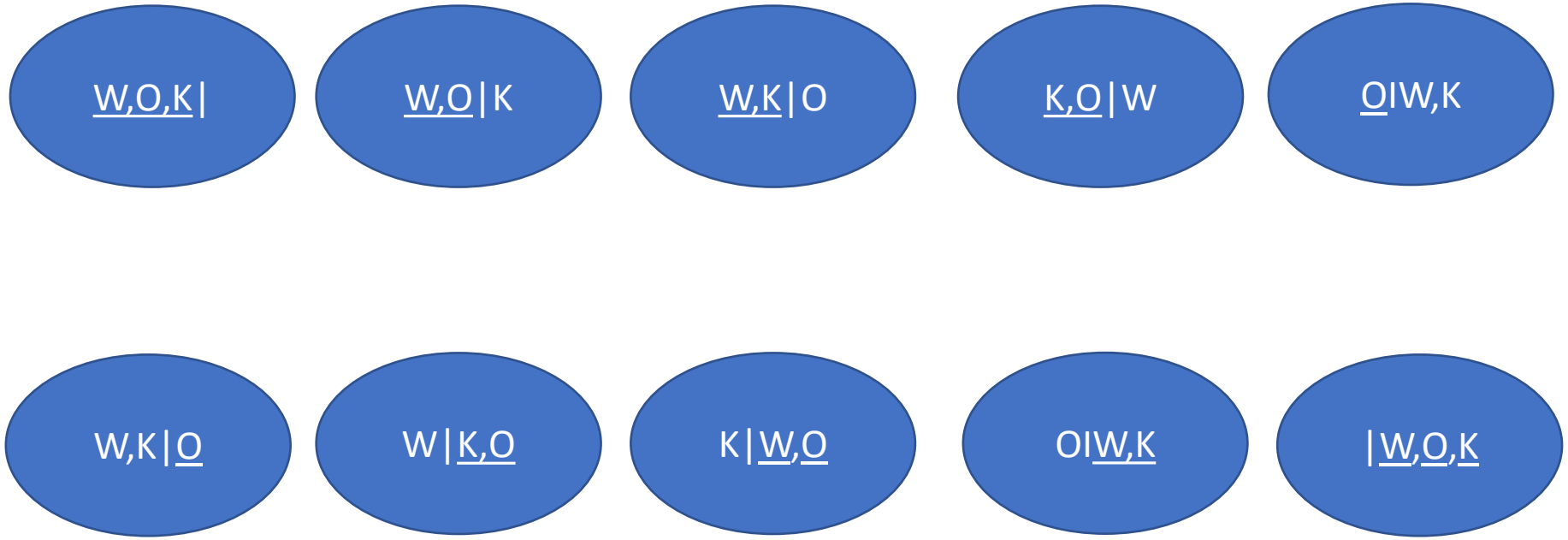


Labirynt

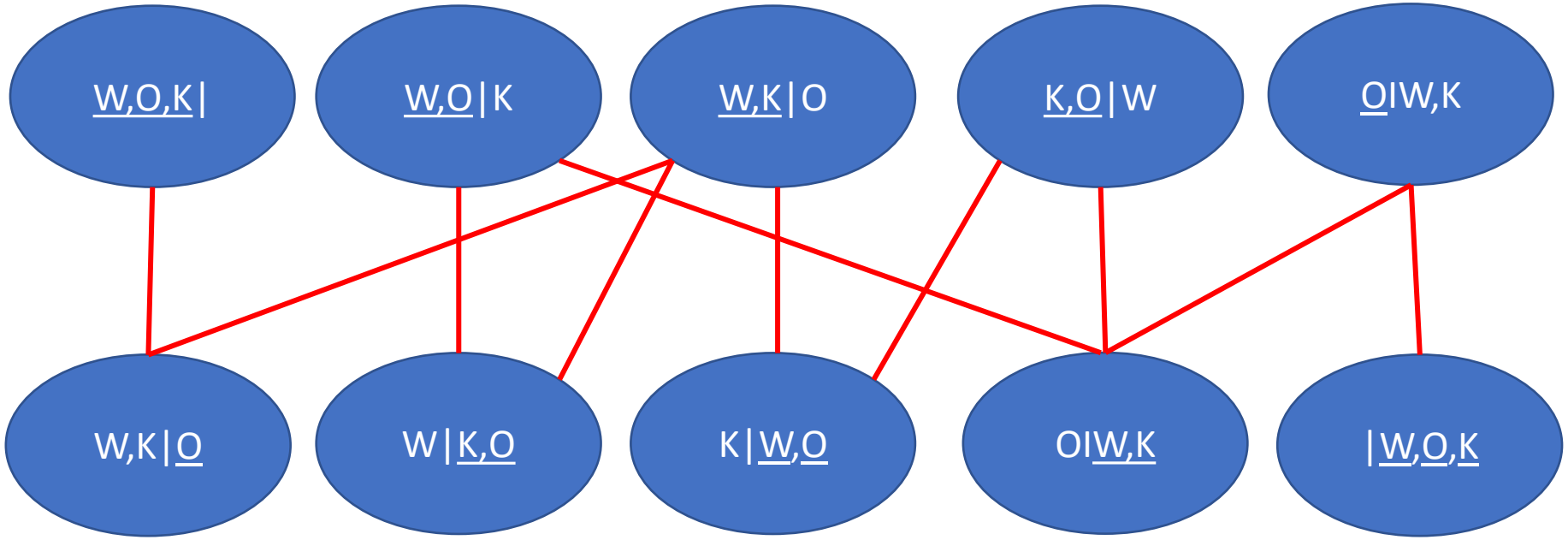




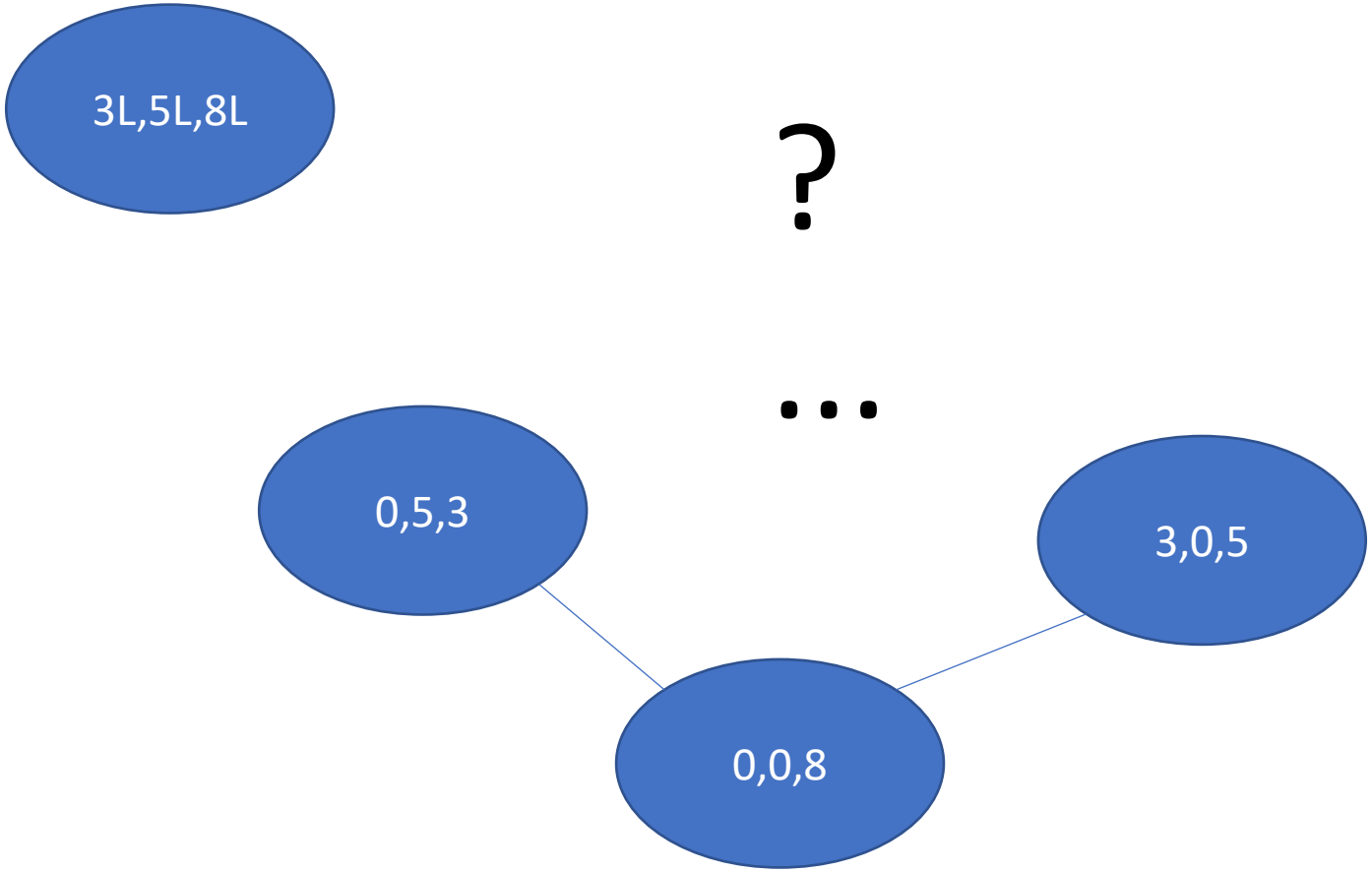
Wilk, owca i kapusta



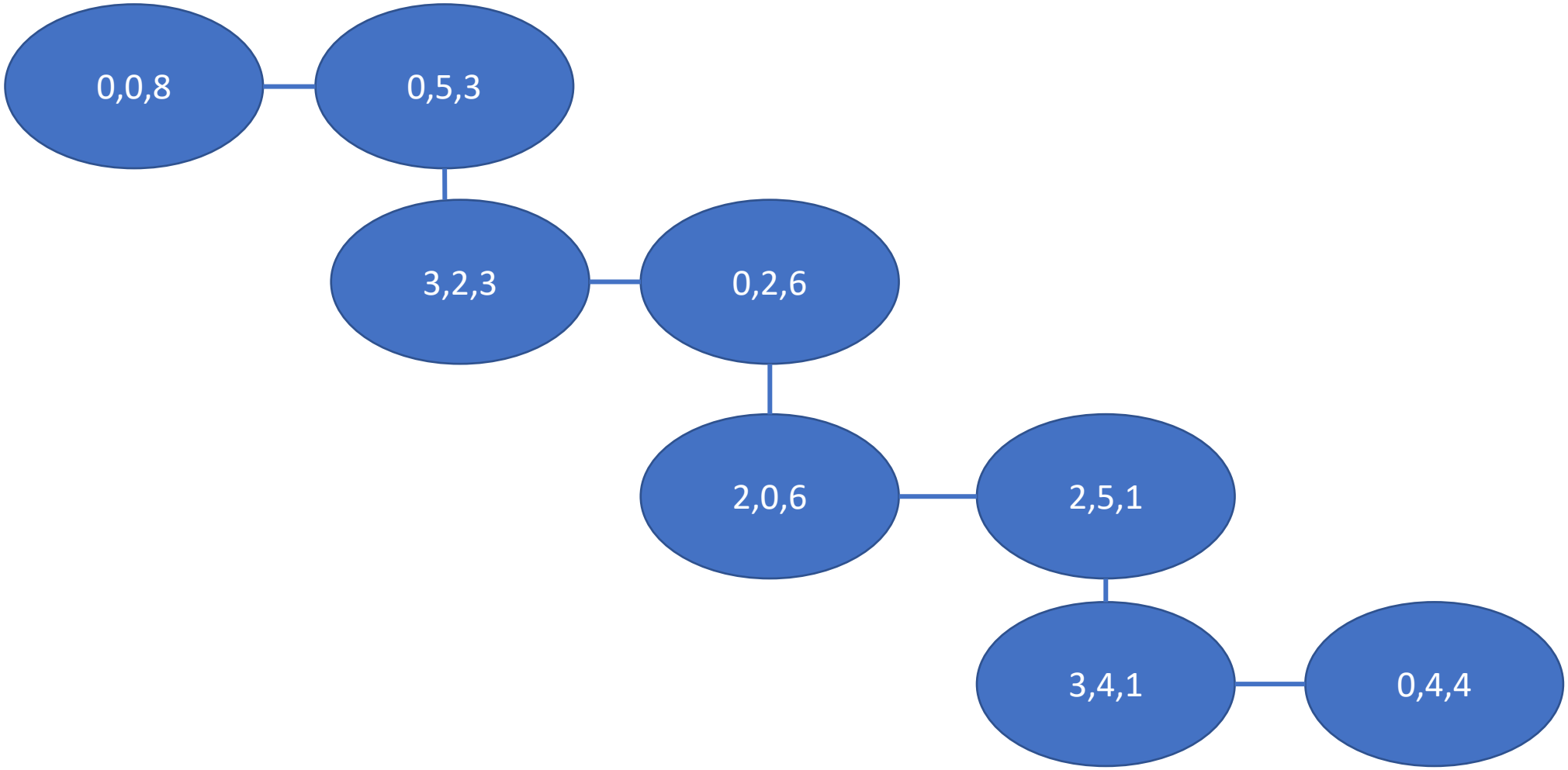
Wilk, owca i kapusta



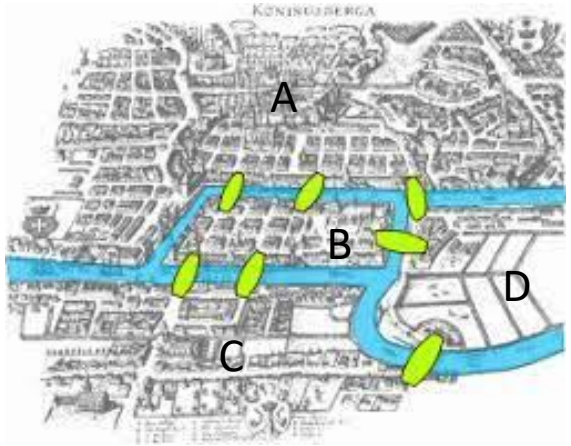
Pytanie: czy istnieje ścieżka z (0,0,8) do (0,4,4)?

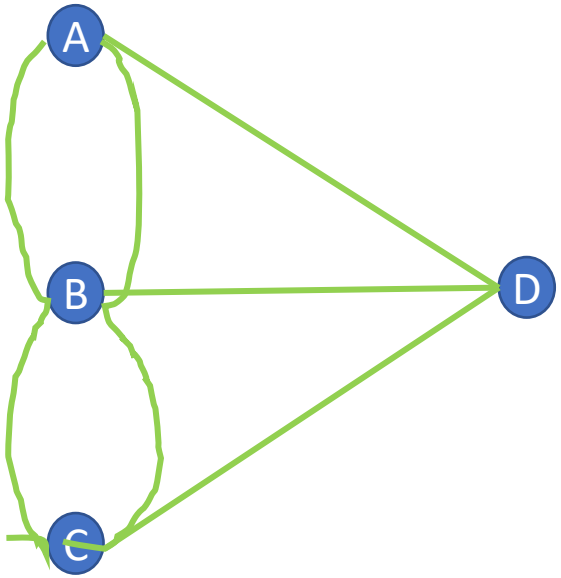


Pytanie: czy istnieje ścieżka z (0,0,8) do (0,4,4)?



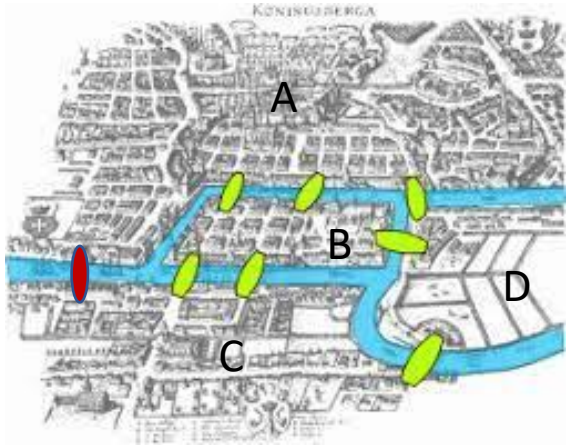
Mosty w Królewcu

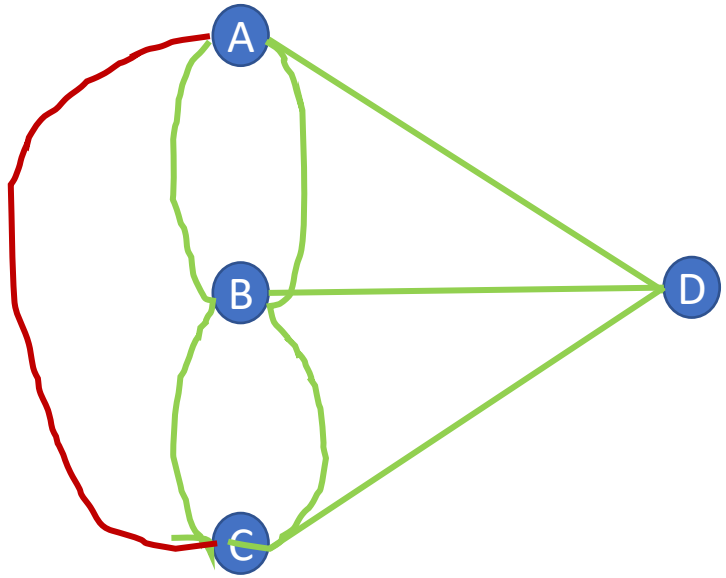




Leonard Euler, 1707 - 1783

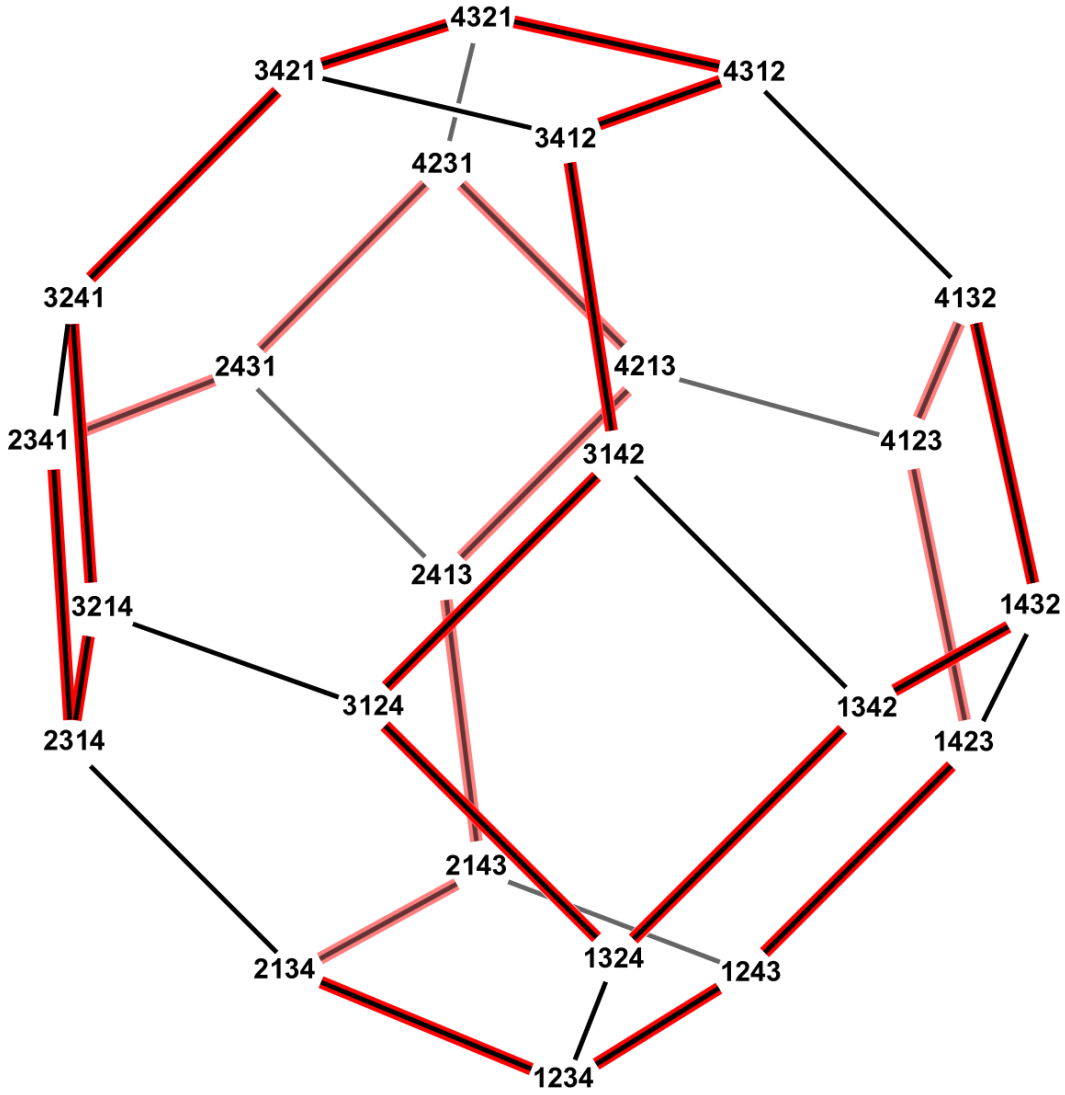
Mosty w Królewcu





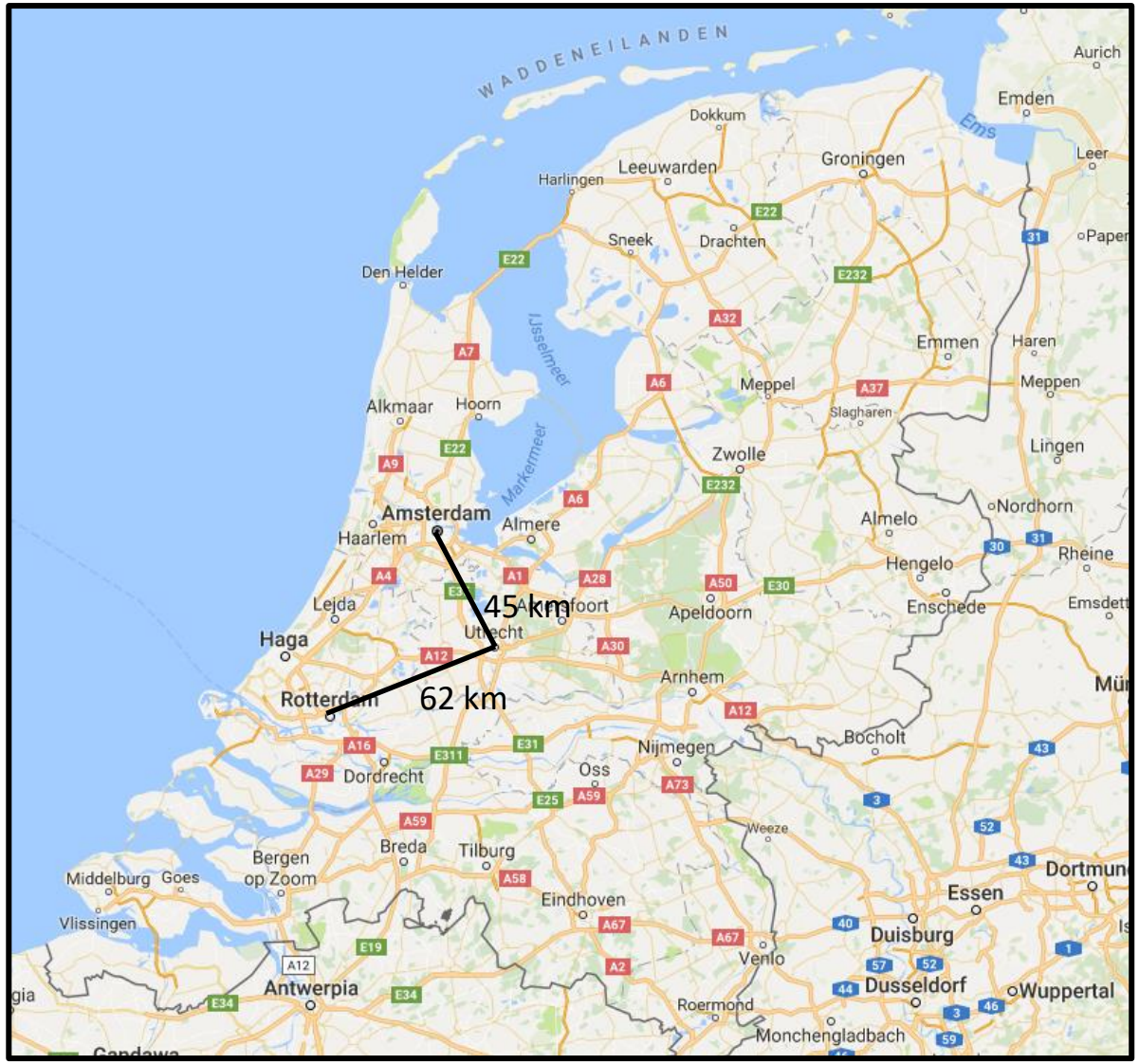
Leonard Euler, 1707 - 1783

Permutacje



Źródło: en.wikipedia.org/wiki/Steinhaus-Johnson-Trotter_algorithm

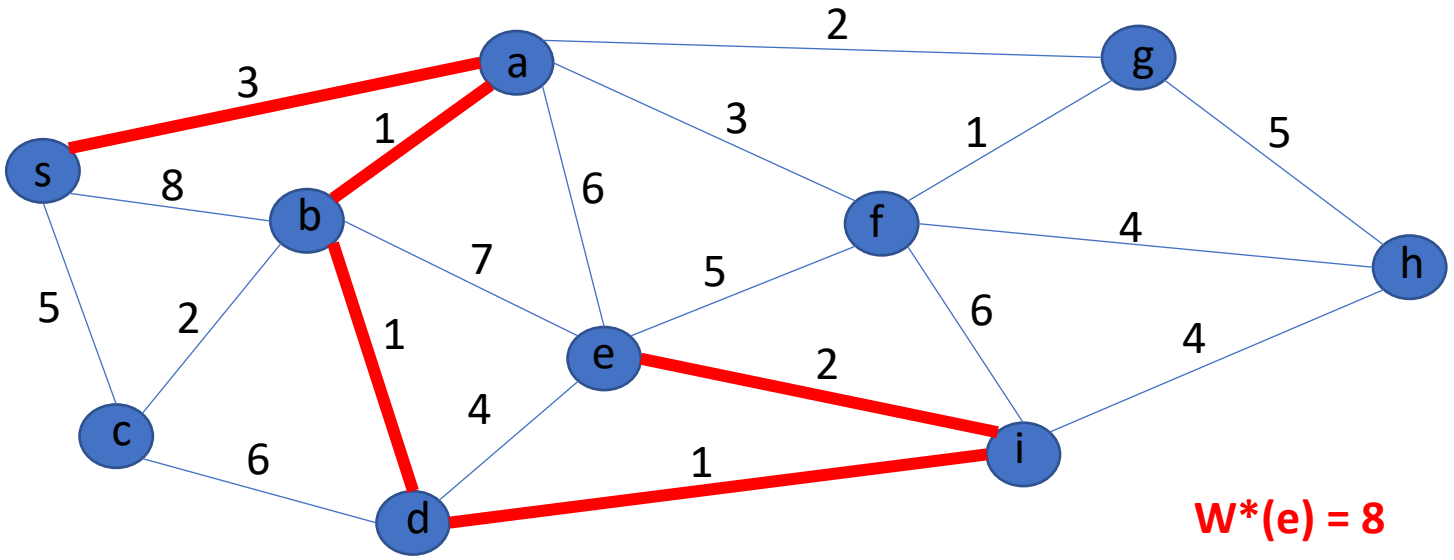
Wycieczka


45 km
62 km
Amsterdam
Haarlem
Almere
Haga
Lejda
Utrecht
Rotterdam
Dordrecht
Arnhem
Nijmegen
Apeldoorn
Zwolle
Meppel
Leeuwarden
Groningen
Emden
Dokkum
Harlingen
Sneek
Drachten
Den Helder
Alkmaar
Hoorn
Emmen
Haren
Meppen
Lingen
Nordhorn
Almelo
Hengelo
Enschede
Rheine
Bocholt
Oss
Weeze
Breda
Tilburg
Bergen op Zoom
Middelburg
Goes
Vlissingen
Eindhoven
Venlo
Roermond
Antwerpia
Essen
Duisburg
Dusseldorf
Wuppertal
Monchengladbach
Aurich
Leer
Paper
Slagharen
Ems
IJsselmeer
Markermeer
WADDENEILANDEN

Problem najlżejszych (najkrótszych) ścieżek z jednym źródłem

Dane: (skończony) spójny graf G = (V,E), wyróżniony wierzchołek s ∈ V,
funkcja wag w: E ⟶ {0, 1, 2, …}, która każdej krawędzi przyporządkowuje nieujemną wagę

Wynik: dla każdego wierzchołka v ∈ V, waga w*(v) najlżejszej ścieżki łączącej v z s,
gdzie wagę ścieżki definiujemy jako sumę wag jej krawędzi



Edsger Wybe Dijkstra
1930 - 2002

Interview

An Interview with Edsger W. Dijkstra

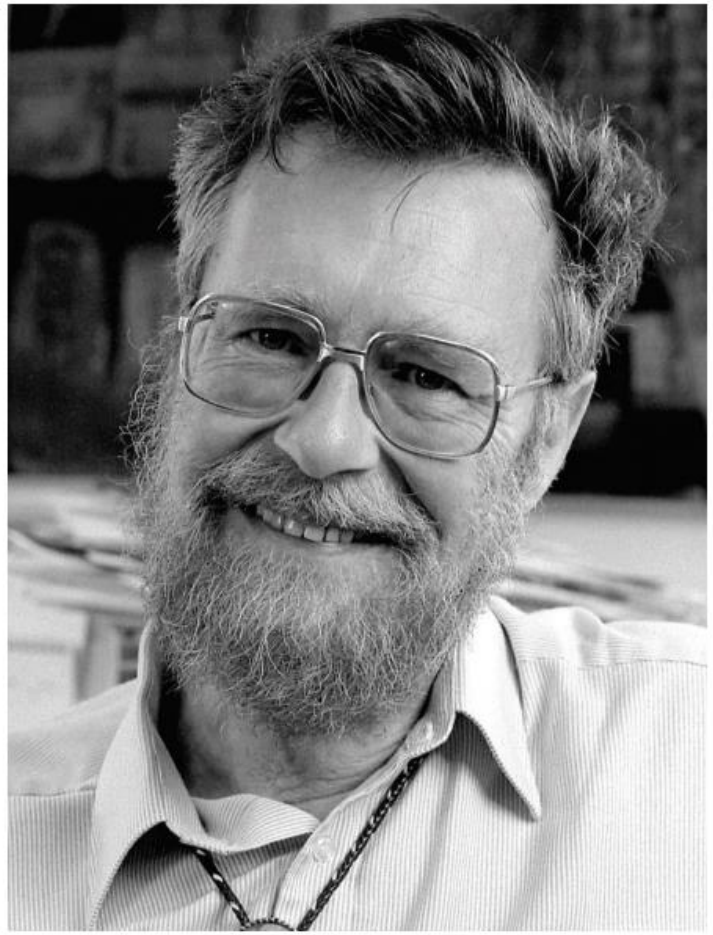
The computer science luminary, in one of his last interviews before his death in 2002, reflects on a programmer's life.

THE CHARLES BABBAGE INSTITUTE holds one of the world's largest collections of research-grade oral history interviews relating to the history of computers, software, and networking. Most of the 350 interviews have been conducted in the context of specific research projects, which facilitate the interviewer's extensive preparation and often suggest specific lines of questions. Transcripts from these oral histories are a key source in understanding the history of computing, since traditional historical sources are frequently incomplete. This interview with programming pioneer Edsger Dijkstra (1930–2002) was conducted by CBI researcher Phil Frana at Dijkstra's home in Austin, TX, in August 2001 for a NSF-KDI project on "Building a Future for Software History."

Winner of ACM's A.M. Turing Award in 1972, Dijkstra is well known for his contributions to computer science as well as his colorful assessments of the field. His contributions to this magazine continue to enrich new generations of computing scientists and practitioners.

We present this interview posthumously on the eighth anniversary of Dijkstra's death at age 72 in August 2002; this interview has been condensed from the complete transcript, available at http://www.cbi.umn.edu/oh.

—*Thomas J. Misa*




Borkum
Norden
Emden
Ems
Dokkum
Groningen
Leeuwarden
Harlingen
Drachten
Sneek
Assen
Den Helder
A7
IJsselmeer
A32
Emmen
Haren
Meppel
A37
Slagharen
Alkmaar
Hoorn
A6
A7
A9
Markermeer
Zwolle
A28
Amsterdam
Almere
Haarlem
Almelo
Nordhorn
Deventer
Hengelo
A4
A2
A1
A28
A1
Lejda
Amersfoort
A1
Apeldoorn
Enschede
Haga
Utrecht
A30
A12
Arnhem
Rotterdam
A50
A12
A27
Bocholt
Dordrecht
A15
Nijmegen
A29
A16
A73
's-Hertogenbosch
A59
Weeze
Breda
Tilburg
A73
Bergen
op Zoom
Middelburg
Goes
A58
Eindhoven
Essen
Vlissingen
A58
A67
Venlo
Duisburg
A67
A12
A2
Antwerpia
E34
E34
Roermond
Düsseldorf
Wup
Monchengladbach

There's a curious story behind your "shortest path" algorithm.

In 1956 I did two important things, I got my degree and we had the festive opening of the ARMAC.[c] We had to have a demonstration. Now the ARRA, a few years earlier, had been so unreliable that the only safe demonstration we dared to give was the generation of random numbers, but for the more reliable ARMAC I could try something more ambitious. For a demonstration for non-computing people you have to have a problem statement that non-mathematicians can understand; they even have to understand the answer. So I designed a program that would find the shortest route between two cities in the Netherlands, using a somewhat reduced road-map of the Netherlands, on which I had selected 64 cities (so that in the coding six bits would suffice to identify a city).

What's the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a 20-minute invention. In fact, it was published in 1959, three years later. The publication is still quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. Without pencil and paper you are almost forced to avoid all avoidable complexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame. I found it in the early 1960s in a German book on management science—"Das Dijkstra'sche Verfahren" ["Dijkstra's procedure"]. Suddenly, there was a method named after me. And it jumped again recently because it is extensively used in all travel planners. If, these days, you want to go from here to there and you have a car with a GPS and a screen, it can give you the shortest way.

Numerische Mathematik 1, 269—271 (1959)

A Note on Two Problems in Connexion with Graphs

By

E. W. DIJKSTRA

We consider n points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict ourselves to the case where at least one path exists between any two nodes. We now consider two problems.

Problem 1. Construct the tree of minimum total length between the n nodes. (A tree is a graph with one and only one path between every two nodes.)

In the course of the construction that we present here, the branches are subdivided into three sets:

I. the branches definitely assigned to the tree under construction (they will form a subtree);

II. the branches from which the next branch to be added to set I, will be selected;

III. the remaining branches (rejected or not yet considered).

The nodes are subdivided into two sets:

A. the nodes connected by the branches of set I,

B. the remaining nodes (one and only one branch of set II will lead to each of these nodes).

We start the construction by choosing an arbitrary node as the only member of set A, and by placing all branches that end in this node in set II. To start with, set I is empty. From then onwards we perform the following two steps repeatedly.

Step 1. The shortest branch of set II is removed from this set and added to set I. As a result one node is transferred from set B to set A.

Step 2. Consider the branches leading from the node, that has just been transferred to set A, to the nodes that are still in set B. If the branch under consideration is longer than the corresponding branch in set II, it is rejected; if it is shorter, it replaces the corresponding branch in set II, and the latter is rejected.

We then return to step 1 and repeat the process until sets II and B are empty. The branches in set I form the tree required.

The solution given here is to be preferred to the solution given by J. B. KRUSKAL [1] and those given by H. LOBERMAN and A. WEINBERGER [2]. In their solutions all the — possibly $\frac{1}{2}n(n-1)$ — branches are first of all sorted according to length. Even if the length of the branches is a computable function of the node coordinates, their methods demand that data for all branches are stored simultaneously. Our method only requires the simultaneous storing of

the data for at most n branches, viz. the branches in sets I and II and the branch under consideration in step 2.

Problem 2. Find the path of minimum total length between two given nodes P and Q.

We use the fact that, if R is a node on the minimal path from P to Q, knowledge of the latter implies the knowledge of the minimal path from P to R. In the solution presented, the minimal paths from P to the other nodes are constructed in order of increasing length until Q is reached.

In the course of the solution the nodes are subdivided into three sets:

A. the nodes for which the path of minimum length from P is known; nodes will be added to this set in order of increasing minimum path length from node P;

B. the nodes from which the next node to be added to set A will be selected; this set comprises all those nodes that are connected to at least one node of set A but do not yet belong to A themselves;

C. the remaining nodes.

The branches are also subdivided into three sets:

I. the branches occurring in the minimal paths from node P to the nodes in set A;

II. the branches from which the next branch to be placed in set I will be selected; one and only one branch of this set will lead to each node in set B;

III. the remaining branches (rejected or not yet considered).

To start with, all nodes are in set C and all branches are in set III. We now transfer node P to set A and from then onwards repeatedly perform the following steps.

Step 1. Consider all branches r connecting the node just transferred to set A with nodes R in sets B or C. If node R belongs to set B, we investigate whether the use of branch r gives rise to a shorter path from P to R than the known path that uses the corresponding branch in set II. If this is not so, branch r is rejected; if, however, use of branch r results in a shorter connexion between P and R than hitherto obtained, it replaces the corresponding branch in set II and the latter is rejected. If the node R belongs to set C, it is added to set B and branch r is added to set II.

Step 2. Every node in set B can be connected to node P in only one way if we restrict ourselves to branches from set I and one from set II. In this sense each node in set B has a distance from node P: the node with minimum distance from P is transferred from set B to set A, and the corresponding branch is transferred from set II to set I. We then return to step 1 and repeat the process until node Q is transferred to set A. Then the solution has been found.

Remark 1. The above process can also be applied in the case where the length of a branch depends on the direction in which it is traversed.

Remark 2. For each branch in sets I and II it is advisable to record its two nodes (in order of increasing distance from P), and the distance between P and that node of the branch that is furthest from P. For the branches of set I this

is the actual minimum distance, for the branches of set II it is only the minimum thus far obtained.

The solution given above is to be preferred to the solution by L. R. FORD [3] as described by C. BERGE [4], for, irrespective of the number of branches, we need not store the data for all branches simultaneously but only those for the branches in sets I and II, and this number is always less than n. Furthermore, the amount of work to be done seems to be considerably less.

References

[1] KRUSKAL jr., J. B.: On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem. Proc. Amer. Math. Soc. **7**, 48—50 (1956).

[2] LOBERMAN, H., and A. WEINBERGER: Formal Procedures for Connecting Terminals with a Minimum Total Wire Length. J. Ass. Comp. Mach. **4**, 428—437 (1957).

[3] FORD, L. R.: Network flow theory. Rand Corp. Paper, P-923, 1956.

[4] BERGE, C.: Théorie des graphes et ses applications, pp. 68—69. Paris: Dunod 1958.

Mathematisch Centrum
2e Boerhaavestraat 49
Amsterdam-O

(Received June 11, 1959)

Edsger Wybe Dijkstra

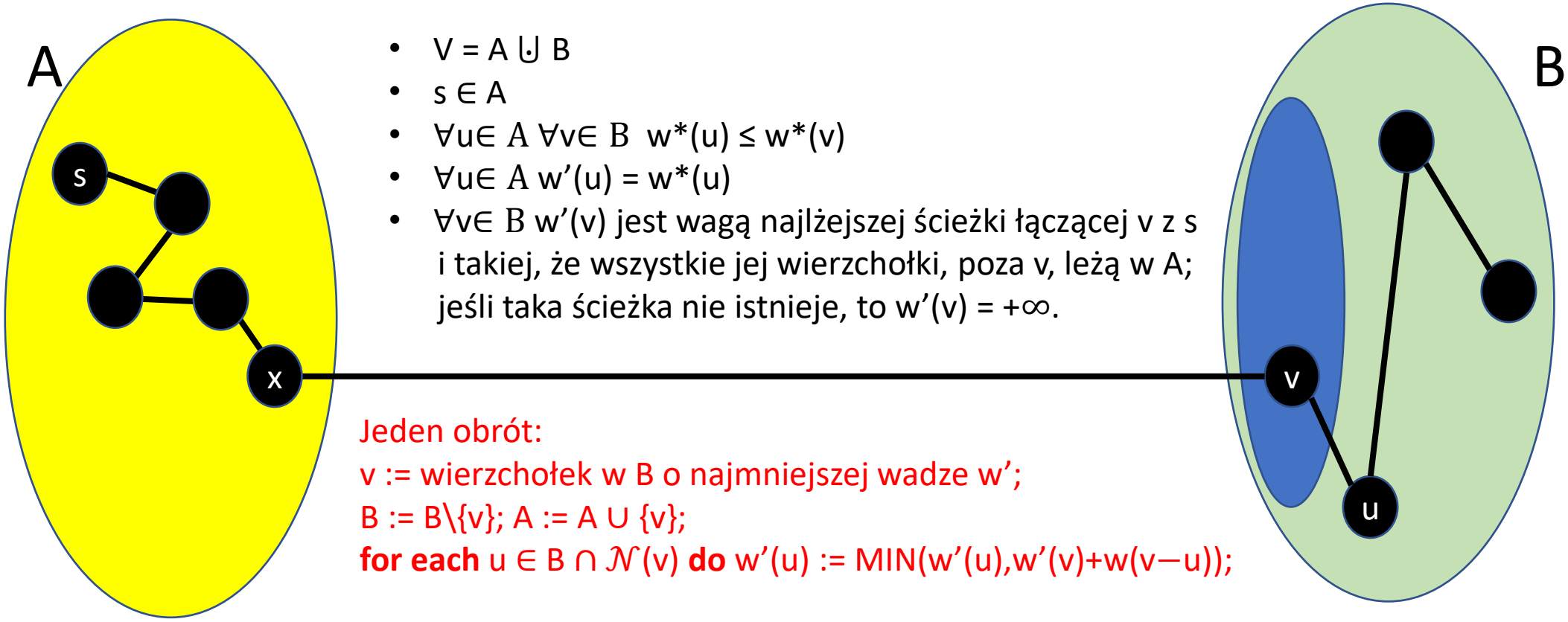
The University of Texas at Austin
No verified email

Follow

Title 1–20	Cited by	Year
A note on two problems in connexion with graphs EW Dijkstra Numerische mathematik 1 (1), 269-271	19650	1959
A discipline of programming EW Dijkstra, EW Dijkstra, EW Dijkstra, EU Informaticien, EW Dijkstra prentice-hall	7080	1976
Cooperating sequential processes EW Dijkstra The origin of concurrent programming, 65-138	2725	1968
Self-stabilizing systems in spite of distributed control EW Dijkstra Communications of the ACM 17 (11), 643-644	2368	1974
Guarded commands, nondeterminacy and formal derivation of programs EW Dijkstra Communications of the ACM 18 (8), 453-457	2276	1975
Notes on structured programming EW Dijkstra Technological University, Department of Mathematics	2093	1970
The structure of the "THE" multiprogramming system EW Dijkstra Classic operating systems, 223-236	1731	2001
Letters to the editor: go to statement considered harmful EW Dijkstra Communications of the ACM 11 (3), 147-148	1676	1968

Algorytm Dijkstry

W pętli obliczamy wagi najlżejszych ścieżek kolejnych wierzchołków w kolejności niemalejącej.
Dla każdego v mamy policzoną wagę w’(v) pewnej ścieżki łączącej v z s. Zachodzi następujący niezmiennik:

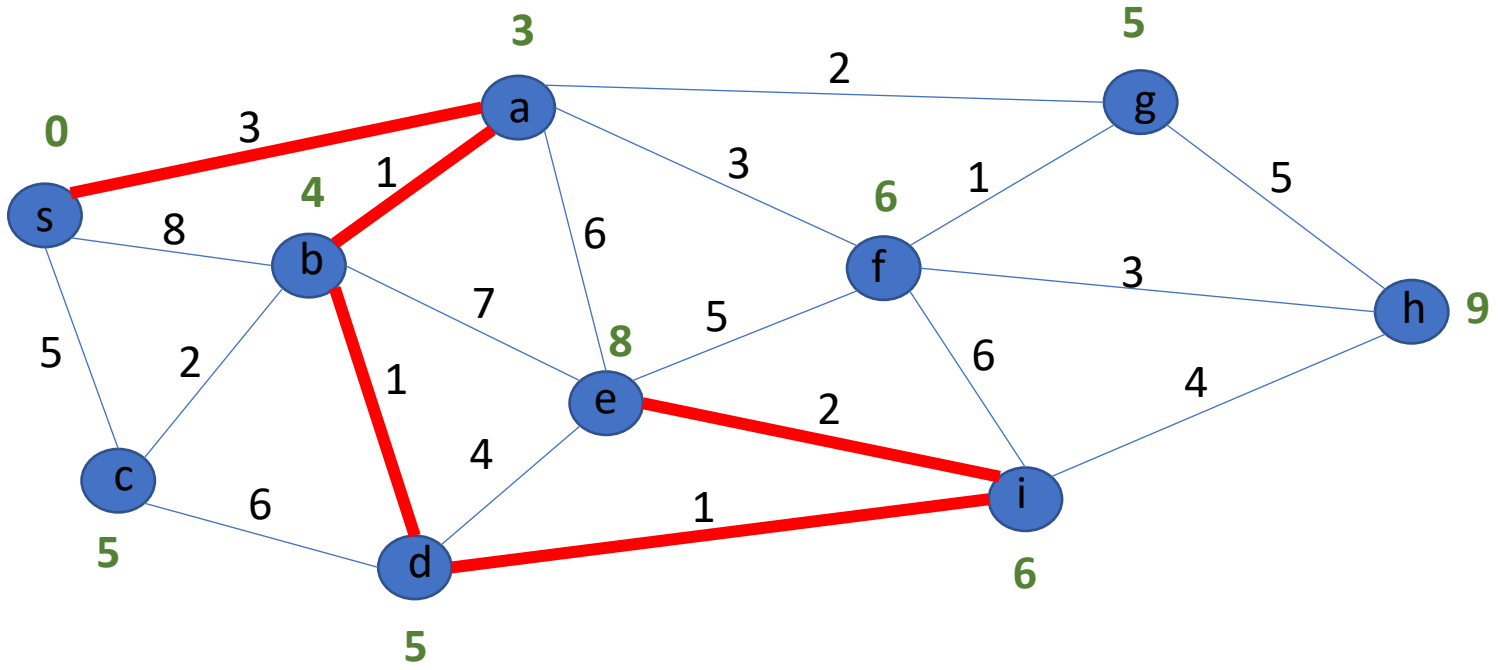


- V = A ⊍ B
- s ∈ A
- ∀u∈ A ∀v∈ B w*(u) ≤ w*(v)
- ∀u∈ A w’(u) = w*(u)
- ∀v∈ B w’(v) jest wagą najlżejszej ścieżki łączącej v z s i takiej, że wszystkie jej wierzchołki, poza v, leżą w A; jeśli taka ścieżka nie istnieje, to w’(v) = +∞.

Jeden obrót:
v := wierzchołek w B o najmniejszej wadze w’;
B := B\{v}; A := A ∪ {v};
for each u ∈ B ∩ $\mathcal{N}$(v) **do** w’(u) := MIN(w’(u),w’(v)+w(v—u));

```
DijkstraAlg::
begin
{ inicjacja}
  w’(s) := 0; A := {s}; B := V\{s};
  for each v ∈ B do w’(v) := +∞;
  for each v ∈ 𝒩(s) do w’(v) := w(s—v); {𝒩(s) – sąsiedzi s w grafie}

{pętla główna}
  while |B| > 0 do
  begin
      v := wierzchołek w B o najmniejszej wadze w’;
      B := B\{v};
      A := A ∪ {v};
      for each u ∈ B ∩ 𝒩(v) do w’(u) := MIN(w’(u),w’(v)+w(v—u))
  end
end;
```



s	a	b	c	d	e	f	g	h	I
0	3	8	5	+∞	+∞	+∞	+∞	+∞	+∞
0	3	4	5	+∞	9	6	5	+∞	+∞
0	3	4	5	5	9	6	5	+∞	+∞
0	3	4	5	5	9	6	5	10	+∞
0	3	4	5	5	9	6	5	10	+∞
0	3	4	5	5	9	6	5	10	6
0	3	4	5	5	9	6	5	9	6
0	3	4	5	5	8	6	5	9	6
0	3	4	5	6	8	6	5	9	6
0	3	4	5	6	8	6	5	9	6

Implementacje algorytmu Dijkstry

Implementacja grafu G

V = {1, 2, …, n}

Wejście:

para liczb n = |V|, m = |E| (**n-1 ≤ m ≤ n(n-1)/2**)

ciąg m trójek u_i, v_i, w_i takich, że u_i—v_i jest krawędzią z E, natomiast w_i jest wagą tej krawędzi

Reprezentacja grafu:

1) Tablicowa

   A[1..n,1..n] – (zmodyfikowana macierz sąsiedztwa grafu G)

$$A[u,v] = \begin{cases} w(u-v) & u-v \in E \\ +\infty & u-v \notin E \end{cases}$$

   rozmiar reprezentacji – Θ(n^2); czas inicjacji reprezentacji - Θ(n^2)

   nie zależy od m

2) Listowa

   L[1..n] – tablica list sąsiadów wierzchołków

   L[u] – lista sąsiadów wierzchołka u; z każdym wierzchołkiem v ∈ L[u] pamiętamy wagę w(v) - wagę krawędzi $u-v$

   rozmiar reprezentacji – Θ(n+m); czas inicjacji - Θ(n+m)

Implementacja zbioru B

```
DijkstraAlg::
begin
{ inicjacja}
  w'(s) := 0; L := {s};

  B := V\{s};
  for each v ∈ B do w'(v) := +∞;
  for each v ∈ 𝒩(s) do w'(v) := w(s—v);

{pętla główna}
  while |B| > 0 do
  begin
     v := wierzchołek w B
     o najmniejszej wadze w';
     B := B\{v};
     A := A ∪ {v};
     for each u ∈ B ∩ 𝒩(v) do
        w'(u) := MIN(w'(u),w'(v)+w(v—u))
  end
end;
```

W zbiorze B przechowujemy wierzchołki v wraz z jednym atrybutem w'(v) (kluczem), którym jest waga pewnej ścieżki z s do v.
Na zbiorze B wykonujemy następujące operacje:

Ini(B,V\{s}):: inicjacja zbioru B
wykonywania 1 raz

Min(B):: podaj element zbioru B z najmniejszym kluczem
wykonywana n-1 razy
DeleteMin:: usuń MIN(B)
wykonywana n-1 razy

DecreaseKey(B,u,w'(u))
wykonywana m razy

Algorytm Dijkstry – implementacja tablicowa

Reprezentacja grafu – macierz sąsiedztwa

Reprezentacja zbioru B – wektor charakterystyczny b[1..n] of 0..1

b[u] = 1 wtedy i tylko wtedy, gdy u ∈ B

```
begin
{ inicjacja}
  w'(s) := 0;
  b[s] := 0;
  for each v ∈ V\{s} do begin w'(v) := A[s,v]; b[v] := 1 end;

{pętla główna}
  k := n-1;
  while k > 0 do
  begin
     min_w := +∞;
     for each u ∈ V do
        if (b[u] = 1) AND (w'(u) < min_w) then begin v := u; min_w := w'(u) end;

     b[v] := 0; k := k-1;
     for each u ∈ V do
        if b[u] = 1 then w'(u) := MIN(w'(u),w'(v)+A[u,v])
  end
end;
```

złożoność czasowa – $\Theta(n^2)$ niezależnie od m

Jeśli nie zmienimy sposobu reprezentacji zbioru B, to zawsze będziemy mieli czas kwadratowy, niezależnie od sposobu reprezentacji grafu G.

Zbiór B:

- skończony zbiór elementów e z kluczami key(e), pochodzącymi z uniwersum z liniowym porządkiem
- $0 \leq |B| < n$
- operacje na B:
  - Ini(B,V\{s})

  - Min(B):: **if** $|B| > 0$ **then**
    return element z B o najmniejszym kluczu

  - DeleteMin:: **if** $|B| > 0$ **then** $B := B \setminus \{Min(B)\}$

  - DecreaseKey(B,e,new_key):: key(e) := new_key
    { $e \in B$ oraz $new\_key \leq key(e)$ }

Kolejka priorytetowa typu Min:

- skończony zbiór Q elementów e z kluczami key(e), pochodzącymi z uniwersum z liniowym porządkiem

- operacje na Q:
  - Ini(Q,{e_1, e_2, …, e_k}):: inicjacja kolejki z elementami e_1, e_2, …, e_k
  - Empty(Q):: **return** $|Q| = 0$
  - Min(Q):: **if** NOT Empty(Q) **then**
    return element z Q o najmniejszym kluczu

  - DeleteMin(Q):: **if** NOT Empty(Q) **then** $Q := Q \setminus \{Min(Q)\}$

  - DecreaseKey(Q,e,new_key):: key(e) := new_key

  - Insert(Q,e):: $Q := Q \cup \{e\}$

Algorytm Dijkstry – implementacja z kolejką priorytetową

Reprezentacja grafu – listy sąsiedztwa

Reprezentacja zbioru B – kolejka priorytetowa typu Min

```
begin
{ inicjacja}
  key(s) := 0;
  for each v ∈ V\{s} do key(v) := +∞;
  for each v ∈ L[s] do key(v) := w(s—v);
  Ini(Q,V\{s})

{pętla główna}
  while NOT Empty(Q) do
  begin
     v := Min(Q);                                   n-1 operacji MIN

     DeleteMin(Min(Q));                             n-1 operacji DeleteMin

     for each u ∈ L[v] do
        DecreaseKey(Q,v,MIN(key(u),key(v)+w(u—v))   m operacji DecreaseKey
  end
end;
```

Złożność: O((n-1)*koszt(MIN) + (n-1)*koszt(DeleteMin) + m*koszt(DecreseKey) + koszt(inicjacja))

Wykład opracowano między innymi na podstawie:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Wprowadzenie do algorytmów, PWN 2012
- Lech Banachowski, Krzysztof Diks, Wojciech Rytter, Algorytmy i struktury danych, PWN 2018

<Mistrzostwa
w Algorytmice
i Programowaniu>

Projekt „Mistrzostwa w Algorytmice i Programowaniu – Uczniowie” jest finansowany ze środków pochodzących z „Programu Rozwoju Talentów Informatycznych na lata 2019-2029”

Dofinansowanie Projektu: 4.887.850,50 zł

Całkowita wartość Projektu: 5.460.850,50 zł