Akademia Górniczo-Hutnicza
Wydział Elektrotechniki, Automatyki,
Informatyki i Inżynierii Biomedycznej

Adrian Horzyk

WSTĘP DO INFORMATYKI

Grafy
i struktury grafowe

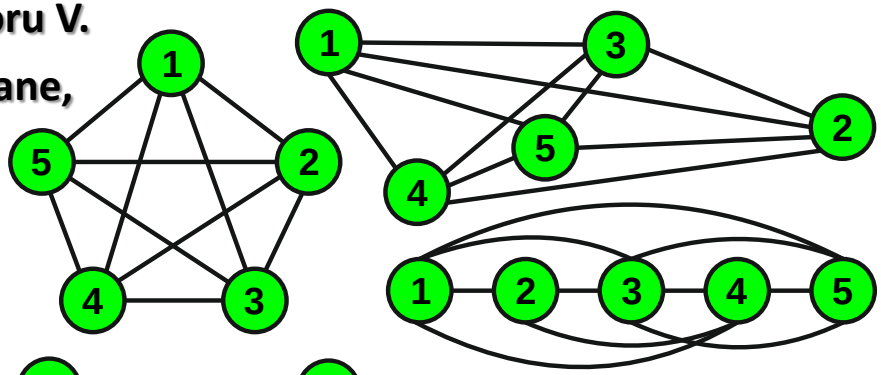
DEFINICJA GRAFU

Graf to struktura danych składająca się z **wierzchołków** i **krawędzi**, łączących dwa wierzchołki.

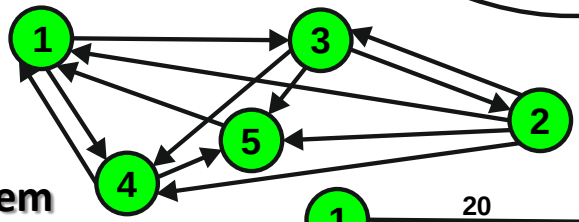
Grafem nieskierowanym nazywamy parę G = (V, E), gdzie V jest niepustym i skończonym zbiorem wierzchołków (węzłów), a E jest dowolnym skończonym zbiorem krawędzi $E \subseteq P_2(V)$ rozpoczynających się w jednym a kończących się w drugim wierzchołku, reprezentowanych w postaci nieuporządkowanych par elementów zbioru V.

Wierzchołki grafu mogą być etykietowane, kolorowane, krawędzie ważone, niosąc dodatkową informację.

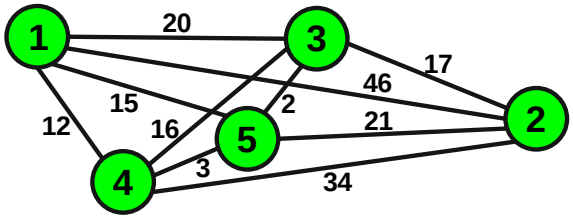
Grafem pełnym nazywamy graf, w którym każde dwa wierzchołki są ze sobą połączone krawędzią. Graf pełny zawierający k wierzchołków nazywamy **k-kliką**.



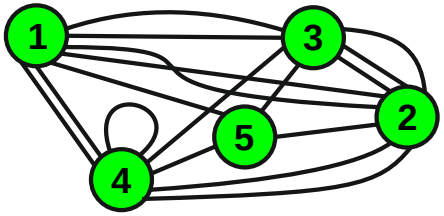
Grafem skierowanym nazywamy parę G = (V, D), gdzie V jest niepustym i skończonym zbiorem wierzchołków, a D jest dowolnym skończonym zbiorem **krawędzi skierowanych** $D \subseteq P(V, V)$, reprezentowanych w postaci uporządkowanych par elementów zbioru V.



Grafem ważonym nazywamy trójkę G = (V, E, w), gdzie G = (V, E) jest grafem, a w: E → R jest funkcją określającą **wagę** krawędzi.



Multigrafem nazywamy trójkę G = (V, E, φ), gdzie G = (V, E) jest grafem, a φ: E → N jest funkcją określającą krotność krawędzi.
W multigrafie dopuszcza się istnienie **pętli** i krawędzi wielokrotnie łączących te same wierzchołki.



WŁAŚCIWOŚCI GRAFU

Graf jest spójny, gdy istnieje droga w grafie pomiędzy każdą parą wierzchołków.

Lasem nazywamy graf, którego żaden podgraf nie zawiera cyklu.

Wierzchołki nazywamy **sąsiednie**, jeśli istnieje krawędź (skierowana lub nieskierowana) pomiędzy nimi.

Wierzchołek nazywamy **izolowanym**, jeśli nie jest połączony żadną krawędzią z innym wierzchołkiem.

Wierzchołki incydentne to wierzchołki, do których prowadzi krawędź z danego wierzchołka.

Stopień wierzchołka w grafie G określony jest poprzez jego ilość wierzchołków incydentnych, tzn. krawędzi wychodzących z niego.

Ścieżką (drogą) w grafie nazywamy ciąg krawędzi, po których należy przejść od wierzchołka startowego do wierzchołka końcowego poprzez ew. wierzchołki pośrednie.
W grafie może istnieć wiele różnych ścieżek pomiędzy tymi samymi wierzchołkami.

Długość ścieżki (drogi) w grafie określona jest przez ilość krawędzi na tej ścieżce (drodze).

Ścieżkę nazywamy **prostą**, jeśli przez każdą krawędź na ścieżce przechodzimy tylko raz.

Najkrótsza ścieżka w grafie pomiędzy dwoma wierzchołkami to ścieżka o minimalnej ilości krawędzi spośród dostępnych ścieżek prowadzących pomiędzy tymi wierzchołkami.

Najkrótsza ścieżka w grafie ważonym określana jest przez minimalną sumę wag krawędzi na ścieżkach prowadzących pomiędzy dwoma wierzchołkami.

WŁAŚCIWOŚCI GRAFU

Ścieżką Hamiltona nazywamy ścieżkę prostą przechodzącą przez wszystkie wierzchołki grafu dokładnie raz.

Ścieżką Eulera nazywamy ścieżkę prostą przechodzącą przez wszystkie krawędzie grafu dokładnie raz.

Cyklem w grafie nazywamy ścieżkę (drogę) zamkniętą, czyli taką, która rozpoczyna się i kończy w tym samym wierzchołku grafu.

Cyklem Hamiltona nazywamy cykl prosty przechodzący przez wszystkie wierzchołki grafu dokładnie raz (rozpoczynający się i kończący w tym samym wierzchołku).

Cyklem Eulera nazywamy cykl prosty przechodzący przez wszystkie krawędzie grafu dokładnie raz (rozpoczynający się i kończący w tym samym wierzchołku).

Graf nazywamy **acyklicznym**, jeśli nie posiada żadnych cykli.

Graf nazywamy **planarnym**, jeśli da się go narysować na płaszczyźnie tak, aby żadne jego krawędzie nie przecinały się.

Pętlą w grafie nazywamy krawędź łączącą wierzchołek sam ze sobą.

REPREZENTACJA GRAFU

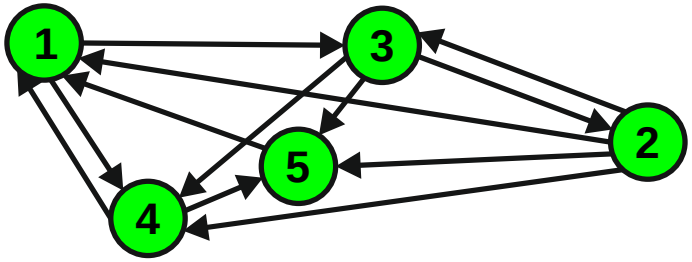
Grafy nie są strukturą hierarchiczną ani liniową, ich wierzchołki mogą mieć różną ilość krawędzi wychodzących (różną ilość incydentnych wierzchołków), więc struktura grafu nie odzwierciedla w naturalny sposób budowy pamięci komputerowych RAM. Implementując grafy wykorzystujemy tablice lub listy do przechowywania informacji na temat sąsiednich (incydentnych) wierzchołków lub łączących je krawędzi.

Grafy reprezentujemy i implementujemy zwykle w postaci:

- Listy krawędzi (*list of edges*)
- Macierzy sąsiedztwa (*adjacency matrix*)
- Macierzy incydencji (*incidence matrix*)
- Listy sąsiedztwa (*adjacency list*)

Przy założeniu, iż żaden wierzchołek w grafie nie jest izolowany,
lista krawędzi składa się z dwójek lub trójek, zawierających:

- etykietę wierzchołka, z którego krawędź wychodzi,
- etykietę wierzchołka, do którego krawędź dochodzi,
- wagę krawędzi (opcjonalnie).



PRZYKŁAD: [(1,3) , (1,4) , (2,1) , (2,3) , (2,4) , (2,5) , (3,2) , (3,4) , (3,5) , (4,1) , (4,5) , (5,1)]

MACIERZ SĄSIEDZTWA

Macierz sąsiedztwa jest macierzą kwadratową o stopniu równym ilości wierzchołków grafu.

Każdy wierzchołek grafu jest indeksowany jedną z kolejnych liczb od 0 do N-1, gdzie N określa ilość wierzchołków takiego grafu.

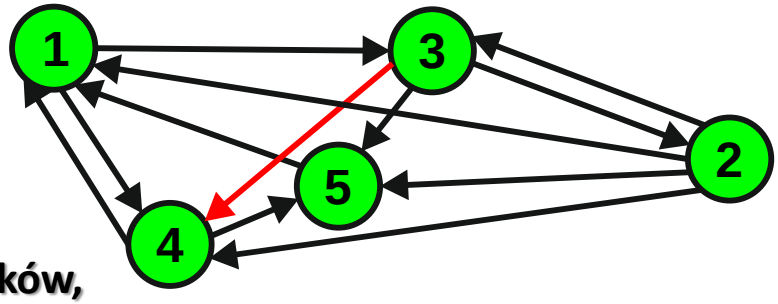
Indeksy wierszy macierzy sąsiedztwa odpowiadają indeksom wierzchołków startowych v_i.

Indeksy kolumn macierzy sąsiedztwa odpowiadają indeksom wierzchołków końcowych v_j.

Każda komórka takiej macierzy reprezentuje jedno możliwe połączenie pomiędzy wierzchołkami v_i oraz v_j:

0 – oznacza brak krawędzi

1 – oznacza istniejącą krawędź
albo w - oznacza wagę krawędzi



i-ty wiersz takiej macierzy jest listą indeksów wierzchołków, do których prowadzą krawędzie z wierzchołka o indeksie i, gdy wartości w komórce są niezerowe.

3	0	1	0	1	1

j-ty wiersz takiej macierzy jest listą indeksów wierzchołków, z których prowadzą krawędzie do wierzchołka o indeksie j, gdy wartości w komórce są niezerowe.

3
1
1
0
0
0

Macierz sąsiedztwa dla grafu nieskierowanego jest symetryczna względem głównej przekątnej, ponieważ jeśli istnieje krawędź (v_i, v_j), to istnieje również krawędź (v_j, v_i).

Vi - WIERZCHOŁEK STARTOWY \ Vj - WIERZCHOŁEK KOŃCOWY	1	2	3	4	5
1	0	0	1	1	0
2	1	0	1	1	1
3	0	1	0	1	1
4	1	0	0	0	1
5	1	0	0	0	0

MACIERZ SĄSIEDZTWA

Stopień i wierzchołka grafu nieskierowanego wyznaczymy zliczając liczbę niezerowych komórek w i-tym wierszu macierzy sąsiedztwa.

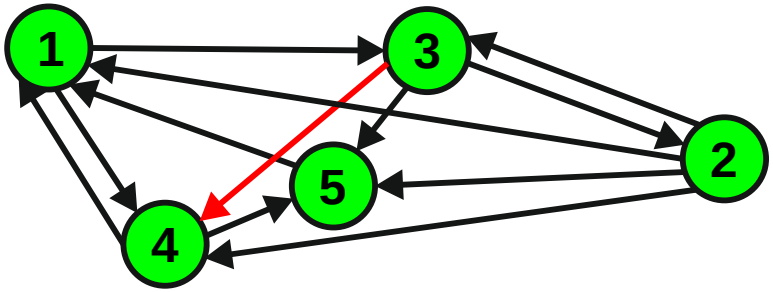
Stopień i wierzchołka grafu skierowanego wyznaczymy zliczając liczbę niezerowych komórek w i-tym wierszu oraz i-tej kolumnie macierzy sąsiedztwa. Możemy też osobno policzyć ilość krawędzi wychodzących (w wierszach) oraz ilość krawędzi wchodzących (w kolumnach).

Indeksy **sąsiadów** wierzchołka i-tego znajdziemy w i-tym wierszu, jeśli wartość pola jest niezerowa.

i-ty wierzchołek jest **izolowany**, jeśli zarówno i-ty wiersz jak również i-ta kolumna zawiera same zera.

Wierzchołki zawierające **pętle** posiadają niezerowe wartości na przekątnej macierzy sąsiedztwa.

Sprawdzenie istnienia krawędzi pomiędzy wierzchołkami w macierzy sąsiedztwa jest operacją o złożoności O(1).



Wadą takiej reprezentacji jest **kwadratowa złożoność pamięciowa**.

Wiele pól takich macierzy zawiera zera, poza przypadkiem grafów pełnych (klik).

Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY				
	1	2	3	4	5
1	0	0	1	1	0
2	1	0	1	1	1
3	0	1	0	1	1
4	1	0	0	0	1
5	1	0	0	0	0

MACIERZ INCYDENCJI

Macierz incydencji jest macierzą o wymiarze N x K, gdzie N – oznacza ilość wierzchołków grafu, a K liczbę jego krawędzi. Każdy wiersz odwzorowuje jeden wierzchołek grafu, a każda kolumna jedną jego krawędź. Zawartość komórki takiej macierzy określa powiązanie i-tego wierzchołka z k-tą krawędzią w następujący sposób dla grafu skierowanego:

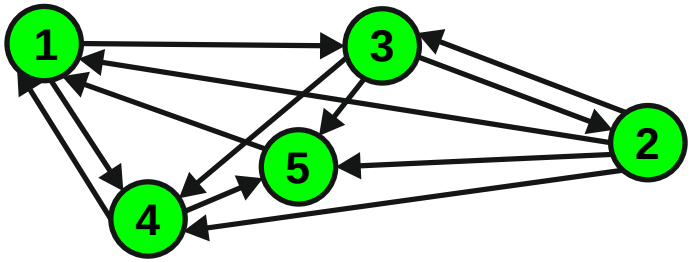
- 0 – oznacza, że i-ty wierzchołek nie jest ani początkiem ani końcem k-tej krawędzi,
- 1 albo w – oznacza, że i-ty wierzchołek jest początkiem k-tej krawędzi (o wadze w - opcjonalnie),
- -1 – oznacza, że i-ty wierzchołek jest końcem k-tej krawędzi.

Jeśli graf jest nieskierowany, wtedy początek i koniec oznaczony jest wartością 1.

Jeśli graf zwiera pętle, wtedy początek i koniec wypada w tym samym wierzchołku, co zaznaczamy w takim grafie wartością 2. Krawędzie wielokrotne możemy reprezentować więc bez trudu.

Ze względu na to, iż każda krawędź ma dokładnie jeden początek i jeden koniec, w każdej kolumnie takiej macierzy występuje dokładnie jedna 1 i jedna -1, a pozostałe wartości są 0.

Każdy wiersz takiej macierzy może natomiast zawierać wiele 1 i -1, gdyż może być początkiem lub końcem wielu krawędzi.



i-ty wiersz dla wierzchołka izolowanego zawiera same zera.

Złożoność pamięciowa: O(N x K)

Vi - WIERZCHOŁEK STARTOWY	Vk - KRAWĘDŹ											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**11**
1	1	1	-1	0	0	0	0	0	0	-1	0	-1
2	0	0	1	1	1	1	-1	0	0	0	0	0
3	-1	0	0	-1	0	0	1	1	1	0	0	0
4	0	-1	0	0	-1	0	0	-1	0	1	1	0
5	0	0	0	0	0	-1	0	0	-1	0	-1	1

LISTA SĄSIEDZTWA

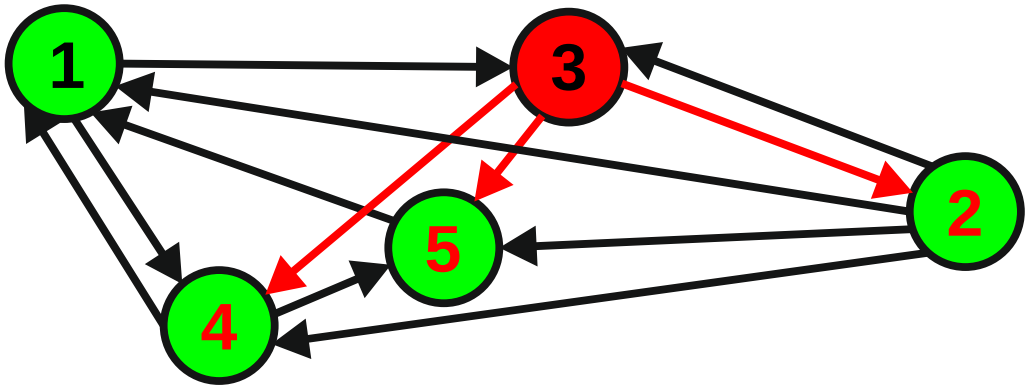
Lista sąsiedztwa implementowana jest jako tablica list (w Pythonie jako lista list), tzn. każdy element listy wierzchołków startowych zawiera listę wierzchołków końcowych, czyli listę sąsiadów, z którymi jest połączony krawędzią.

Nieco trudniej w grafie skierowanym znajdujemy wierzchołki, od których prowadzi krawędź do danego wierzchołka, gdyż trzeba przeszukać wszystkie listy zawierające indeks wierzchołka końcowego O(K), lecz częściej interesuje nas lista sąsiadów, czyli do których wierzchołków prowadzi krawędź z danego wierzchołka. Problem ten nie występuje w przypadku grafów nieskierowanych.

To zdecydowanie najefektywniejszy (pod względem pamięciowym) i najczęściej stosowany sposób implementacji grafu, gdyż nie zawiera niewykorzystanych komórek pamięci (za wyjątkiem wierzchołków izolowanych).

Chcąc zapisać wagę krawędzi, należy wykorzystać listę list obiektów (klas), których kluczem jest indeks (etykieta) wierzchołka docelowego, a wewnątrz klasy zapisana jest waga.

Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY				
1	3	4			
2	1	3	4	5	
3	2	4	5		
4	1	5			
5	1				



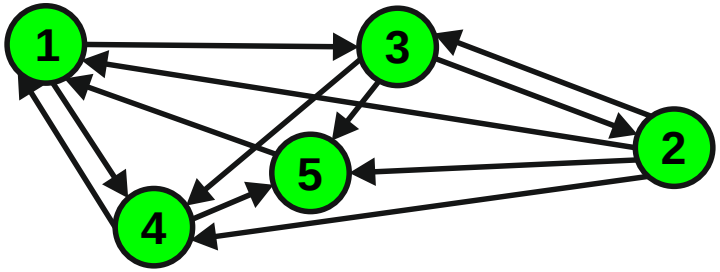
PRZECHODZENIE PO WIERZCHOŁKACH GRAFU

Po wierzchołkach grafu nie sposób przechodzić w sposób liniowy, więc najczęściej stosowane są drzewa rozpięte na grafie oraz algorytmy wykorzystywane do przeszukiwania drzew.

W grafach jednak mogą występować cykle i pętle, co mogłoby spowodować zapętlenie się takiego algorytmu i kręcenie się po tych samych wierzchołkach wielokrotnie.

Dla uniknięcia tego stosujemy dodatkowy parametr typu boolowskiego, w którym zapisujemy, czy wierzchołek został już odwiedzony/przeszukany czy też nie.

Jeśli więc wzbogacimy algorytmy przeszukiwania drzew o taki parametr, będziemy mogli wykorzystać algorytmy:



DFS – *depth first search* – przeszukiwanie wgłąb,

BFS – *breadth first search* – przeszukiwanie wszerz

również do przeszukiwania grafów.

Przejście grafu (*graph traversal*) algorytmem DFS wykonujemy następująco:

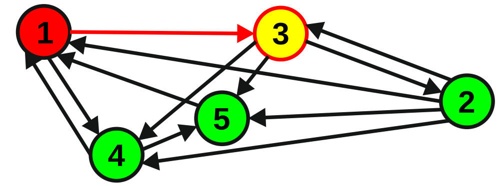
1. Przeszukujemy bieżący wierzchołek.
2. Zaznaczamy bieżący wierzchołek jako przeszukany.
3. Przechodzimy do pierwszego nie przeszukanego jeszcze wierzchołka incydentnego bieżącego wierzchołka.
4. Jeśli wszystkie incydentne wierzchołki zostały już przeszukane, wtedy wracamy do wierzchołka, z którego przeszliśmy do tego wierzchołka i wykonujemy w nim punkt 3.

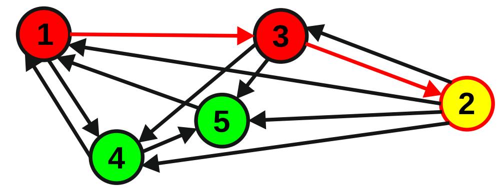
Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

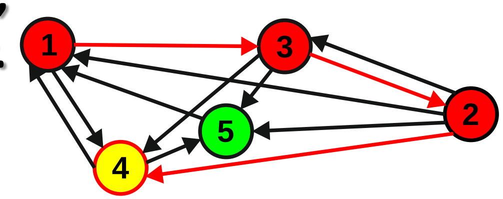
5. Kończymy przeszukiwać graf, gdy wierzchołek nie posiada poprzednika, z którego weszliśmy.

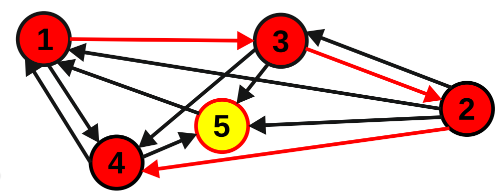
PRZECHODZENIE PO GRAFIE ALGORYTMEM DFS

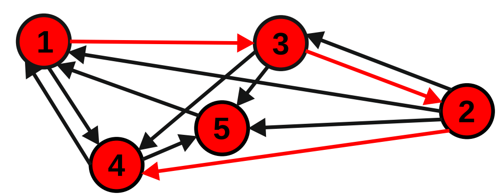
1. Rozpoczynamy przeszukiwanie grafu od wierzchołka nr 1.
2. Przechodzimy do pierwszego jego incydentnego i jeszcze nie odwiedzonego wierzchołka, czyli do wierzchołka nr 3.
3. Z wierzchołka nr 3. przechodzimy do jego pierwszego incydentnego, nie odwiedzonego wierzchołka nr 2.
4. Przeszukując listę incydentnych wierzchołków wierzchołka nr 2, pomijamy wierzchołek nr 1 i nr 3, gdyż one były już odwiedzone, znajdując wierzchołek nr 4 jako pierwszy jeszcze nieodwiedzony.
5. W wierzchołku nr 4 na liście incydencji pomijamy odwiedzony wcześniej wierzchołek nr 1 i dochodzimy do nieodwiedzonego jeszcze wierzchołka nr 5 i przechodzimy do niego.
6. Na liście incydencji w wierzchołku nr 5 nie znajduje się już żaden jeszcze nieodwiedzony wierzchołek, więc wycofujemy się.
7. Podobnie postępujemy w wierzchołkach nr 4, 2, 3, 1, gdzie też nie znajdujemy już żadnego nieodwiedzonego wierzchołka.
8. Kończymy przeszukiwanie grafu.











Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

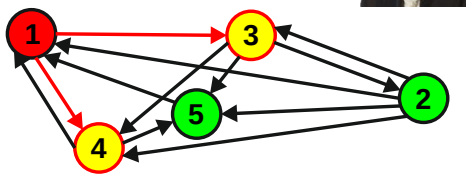
Vi - WIERZCHOŁEK STARTOWY	Vj - WIERZCHOŁEK KOŃCOWY			
1	3	4		
2	1	3	4	5
3	2	4	5	
4	1	5		
5	1			

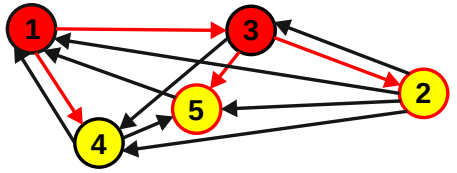
PRZECHODZENIE PO GRAFIE ALGORYTMEM BFS

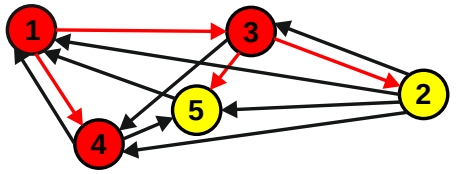
Przejście grafu algorytmem BFS wykonujemy następująco:

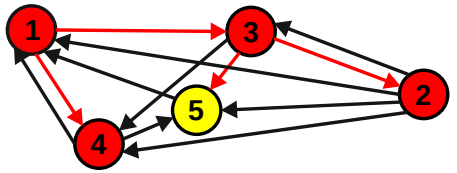
1. Tworzymy pustą kolejkę przeszukiwanych wierzchołków.
2. Dodajemy do kolejki wierzchołek, od którego rozpoczynamy przeszukiwanie grafu, i zaznaczamy go jako uwzględniony w przeszukiwaniu (odwiedzony).
3. Ściągamy z kolejki pierwszy wierzchołek i go przeszukujemy.
4. Do kolejki przeszukiwanych wierzchołków wpisujemy wszystkie jego incydentne wierzchołki, które nie zostały dotąd jeszcze przeszukane i zaznaczamy je jako uwzględnione w przeszukiwaniu.
5. Przechodzimy do punktu 3.
6. Kończymy przeszukiwać graf, gdy kolejka zostanie opróżniona.

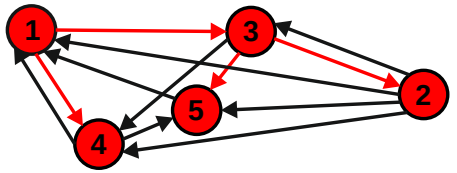
Ilość potencjalnie dodawanych elementów do kolejki może jest O(N), a więc algorytm BFS jest obarczony liniową złożonością pamięciową. Zwykle dodawanych jest mniej elementów do kolejki, lecz w przypadku grafu pełnego, zostanie dodanych N-1 elementów (sąsiadów).

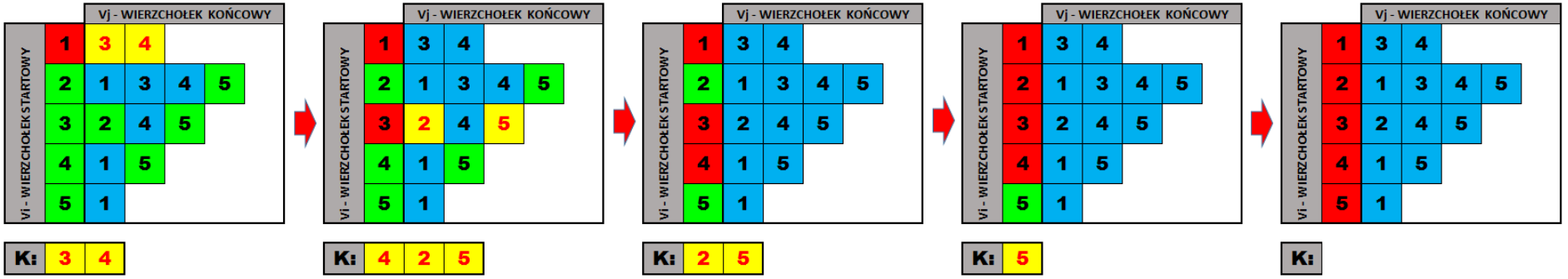








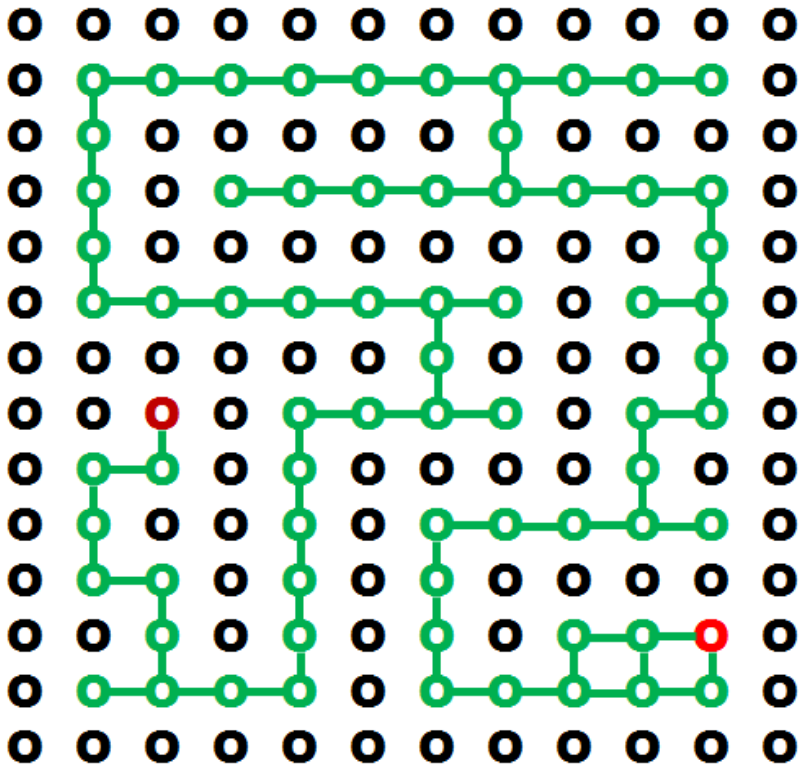




ZNAJDYWANIE DROGI W LABIRYNCIE

Załóżmy, że mamy określony dowolny labirynt jako macierz, w której każde pole jest wierzchołkiem grafu. Wierzchołki są ze sobą połączone, jeśli istnieje bezpośrednie przejście pomiędzy nimi. Oznacza to, iż wierzchołki reprezentujące ściany będą wierzchołkami izolowanymi. Najkrótszą drogę w grafie możemy znaleźć algorytmem BFS, który będzie przechodził od startu (korzenia drzewa BFS) do mety szukając przejścia na najwyższym poziomie w drzewie przeszukiwań, co zapewni znalezienie najkrótszej ścieżki w grafie. Do zaznaczenia odwiedzonych wierzchołków oraz drogi w wierzchołkach zapisuje się informację, skąd algorytm dotarł do danego wierzchołka (L – z lewa, P – z prawa, G – z góry, D – z dołu). Przykład demonstruje typowe przekształcenie zadania na problem grafowy i szukanie rozwiązania z wykorzystaniem teorii grafów.

#	#	#	#	#	#	#	#	#	#	#	#
#											#
#		#	#	#	#	#		#	#	#	#
#		#									#
#		#	#	#	#	#	#	#	#		#
#								#			#
#	#	#	#	#	#		#	#	#		#
#	#	M	#					#			#
#			#		#	#	#	#		#	#
#		#	#		#						#
#			#		#		#	#	#	#	#
#	#		#		#		#			S	#
#					#						#
#	#	#	#	#	#	#	#	#	#	#	#



ZAGADNIENIA GRAFOWE

Można badać różne ciekawe i potrzebne właściwości grafów:

- **Spójność (umożliwiająca znalezienie drogi w grafie pomiędzy wierzchołkami)**
- **Znajdywanie spójnych podgrafów**
- **Znajdywanie silnie spójnych podgrafów**
- **Znajdywanie drzew rozpinających**
- **Znajdywanie mostów (krawędzi) i punktów artykulacji (wierzchołków), których usunięcie spowoduje zwiększenie liczby spójnych podgrafów w grafie.**

Graf/Podgraf nazywamy spójnym (*connected*), gdy dla każdych dwóch wierzchołków istnieje ścieżka, która je ze sobą łączy (nie koniecznie dwustronnie – w przypadku grafu/podgrafu skierowanego). Graf/Podgraf jest niespójny w przeciwnym wypadku.

Graf/Podgraf nazywamy silnie spójnym (*strongly connected*), gdy istnieją ścieżki pomiędzy każdymi dwoma wierzchołkami tego grafu/podgrafu, czyli można z każdego wierzchołka przejść do każdego innego.

Drzewo rozpinające (*spanning tree*) to drzewo zawierające wszystkie wierzchołki grafu i część jego krawędzi. Węzły w takim drzewie połączone są istniejącymi krawędziami grafu. Może istnieć wiele różnych drzew rozpinających dla danego grafu. Drzewa te możemy znaleźć np. wcześniej omówionymi algorytmami DFS lub BFS.

Mostem (*bridge*) w grafie nazywamy każdą krawędź, której usunięcia spowoduje zwiększenie liczby spójnych podgrafów.

Punktem artykulacji (*articulation point / cut vertex*) jest każdy wierzchołek w grafie, którego usunięcie spowoduje zwiększenie liczby spójnych podgrafów.

GRAFY DWUDZIELNE

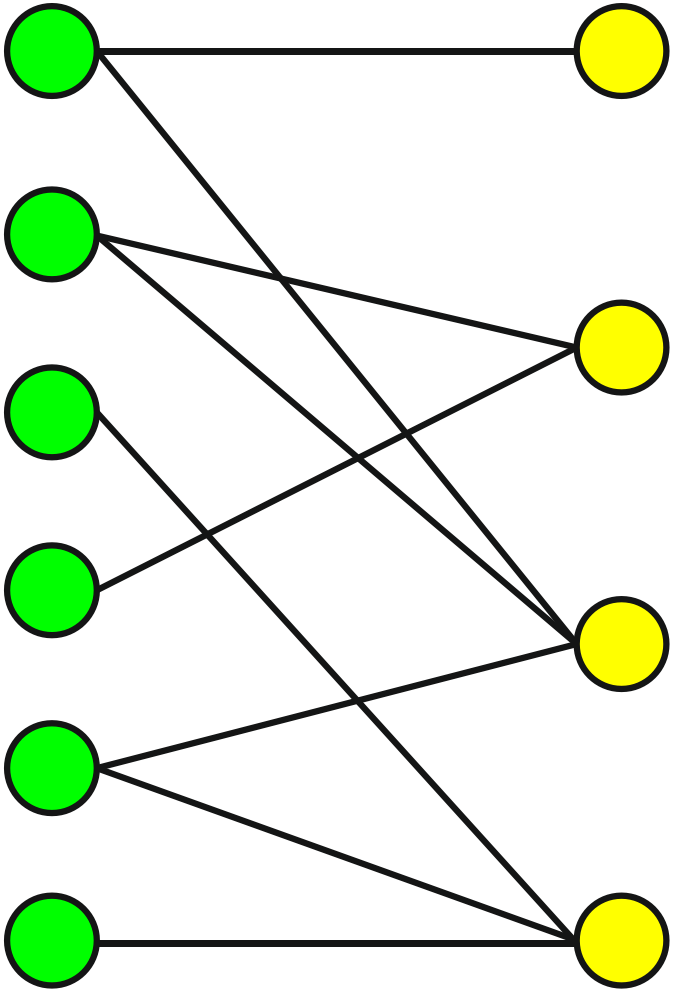
Dowolny nieskierowany, spójny graf nazywamy **dwudzielnym (*bigraph, bipartiteness*)**, jeśli jego wierzchołki możemy podzielić na dwa rozłączne zbiory w taki sposób, iż wierzchołki są połączone tylko pomiędzy wierzchołkami tych zbiorów, natomiast nie istnieją połączenia pomiędzy wierzchołkami wewnątrz tych zbiorów.

Jeśli krawędź grafu dwudzielnego łączy ze sobą wierzchołki z dwóch różnych zbiorów, wtedy graf będzie dwudzielny, jeśli uda nam się pokolorować (poetykietować) wszystkie jego wierzchołki dwoma kolorami tak, aby żaden z jego sąsiadów nie był tego samego koloru, co wierzchołek, którego sąsiedztwo rozważamy.

Szukanie grafu dwudzielnego polega na przechodzeniu po grafie algorytmem BFS lub DFS i naprzemiennym kolorowaniu kolejnych wierzchołków. Jeśli na drodze napotkamy na pokolorowany już węzeł tym samym kolorem, tzn. że graf nie jest dwudzielny.
Jeśli natomiast uda nam się w taki sposób pokolorować cały graf, wtedy graf jest dwudzielny.

Przykład i algorytm można znaleźć:

http://eduinf.waw.pl/inf/alg/001_search/0131.php



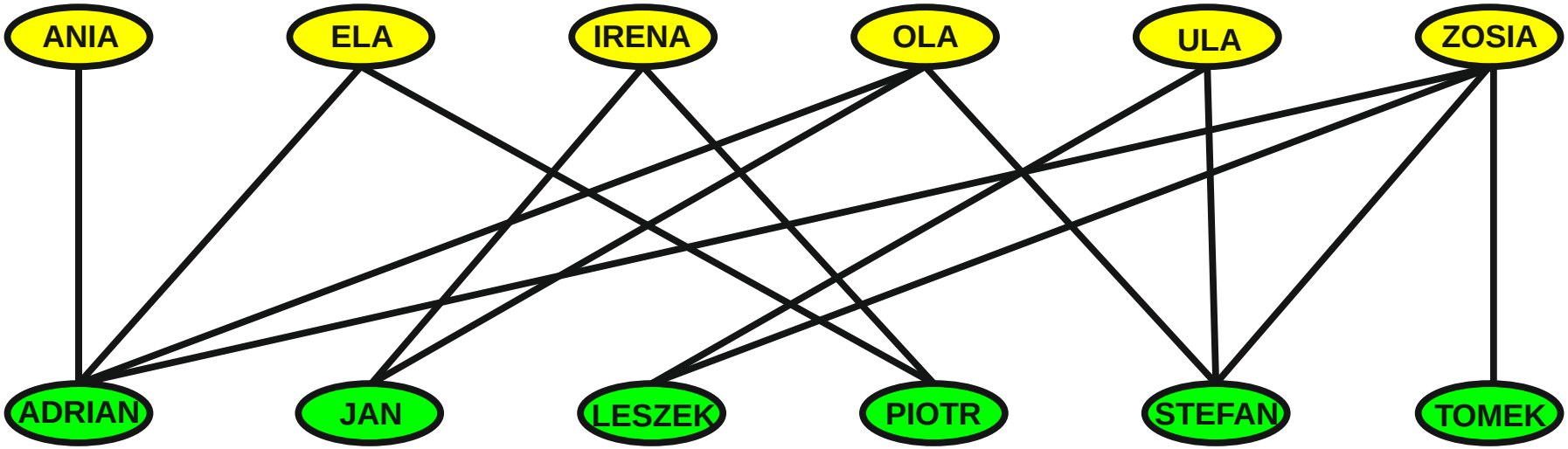
PROBLEM KOJARZENIA MAŁŻEŃSTW

Zagadnienie kojarzenia małżeństw polega na dobraniu w pary n panien i n kawalerów, przy czym każda panien jest skłonna zaakceptować tylko jednego z k ≤ n kawalerów.

Jest to typowy przykład **zagadnienia grafowego**, które możemy rozwiązać przy pomocy **grafu dwudzielnego**, gdyż wierzchołki jednego zbioru reprezentują kawalerów, drugiego zbioru panny, a krawędzie określają preferencję panien.

Graf jest dwudzielny przy założeniu związków heteroseksualnych,
czyli żadne dwie panny ani dwóch kawalerów nie łączy się ze sobą.

Jeśli każdy podzbiór k panien akceptuje co najmniej k kawalerów, wtedy problem jest zawsze rozwiązywalny (twierdzenie Philipa Halla). Rozważamy **ścieżki alternatywne**.



Przykład i algorytm można znaleźć:

http://eduinf.waw.pl/inf/alg/001_search/0131a.php

CYKLICZNOŚĆ GRAFÓW

Cykl jest **ścieżką zamkniętą** prowadzącą przez wierzchołki grafu w taki sposób, iż rozpoczyna się i kończy w tym samym wierzchołku.
Cykle nie muszą przechodzić po wszystkich wierzchołkach grafu, aczkolwiek często takie cykle rozważamy (np. cykl Hamiltona, cykl Eulera).

W celu sprawdzenia, czy graf zawiera jakiś cykl, wystarczy przejść po nim algorytmem DFS i jeśli natrafimy na wierzchołek wcześniej odwiedzony, tzn. że graf jest **cykliczny**.
Jeśli nie znajdziemy wcześniej odwiedzonego wierzchołka, wtedy graf jest **acykliczny**.

Przykładami **grafów acyklicznych** są **drzewa** i **lasy**.

Chcąc wyznaczyć cykl w grafie, kolejno przechodzone wierzchołki należy zapisywać na stosie. Jeśli znajdziemy odwiedzony już wierzchołek, wtedy rozbierając stos wyznaczymy cykl rozpoczynający się i kończący w tym wierzchołku.

Cykl Eulera jest ścieżką zamkniętą (cyklem), która przechodzi dokładnie jeden raz przez każdą krawędź grafu, tzn. kończy się w wierzchołku, w którym się rozpoczęła.

Cykl Hamiltona jest ścieżką zamkniętą (cyklem), która przechodzi dokładnie jeden raz przez każdy wierzchołek grafu, tzn. kończy się w wierzchołku, w którym się rozpoczęła.

Ścieżka Eulera jest ścieżką, która przechodzi dokładnie jeden raz przez każdą krawędź grafu.

Ścieżka Hamiltona jest ścieżką, która przechodzi dokładnie jeden raz przez każdy wierzchołek grafu.

Przykłady i algorytmy można znaleźć:

Cykle: http://eduinf.waw.pl/inf/alg/001_search/0132.php, http://eduinf.waw.pl/inf/alg/001_search/0133.php

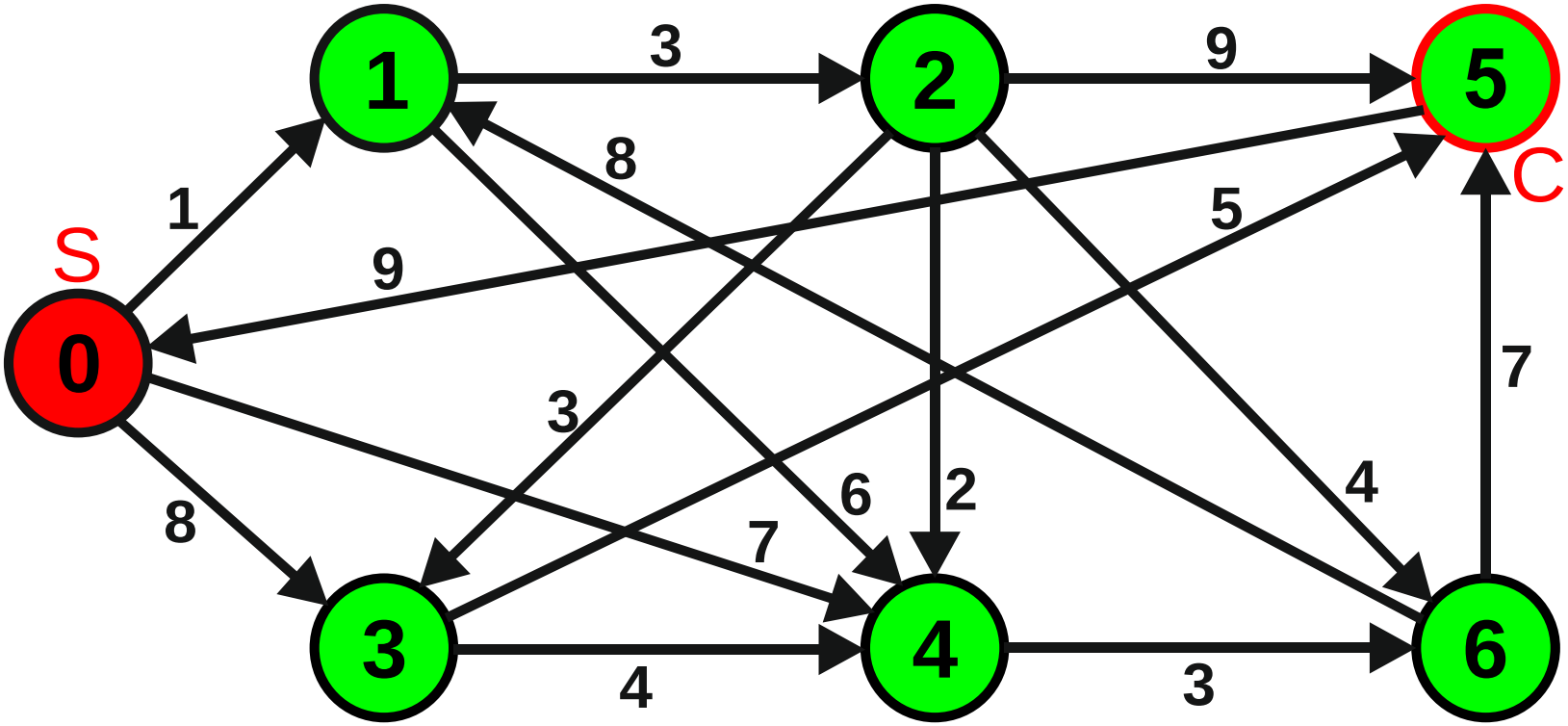
Cykl Eulera: http://eduinf.waw.pl/inf/alg/001_search/0135.php

Cykl Hamiltona: http://eduinf.waw.pl/inf/alg/001_search/0136.php

ALGORYTMY NAJKRÓTSZEJ ŚCIEŻKI W GRAFIE WAŻONYM DIJKSTRY, BELLMANA-FORDA, FLOYDA-WARSHALLA, A*

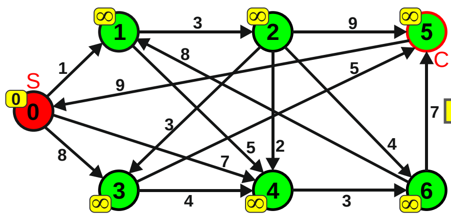
Szukanie najkrótszej ścieżki w grafie ważonym prowadzącej od wybranego wierzchołka do wskazanego lub wszystkich pozostałych jest jednym z podstawowych zagadnień w teorii grafów, mających wiele praktycznych zastosowań, np. w nawigacji satelitarnej, przesyłania wiadomości przez sieć routerów, w sieciach telekomunikacyjnych, wyznaczanie połączeń samolotowych o najniższym koszcie lub czasie przelotu itd.

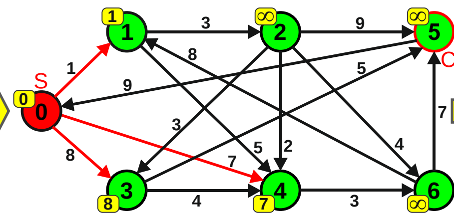
Jak dotrzeć z wierzchołka S do wierzchołka C przy najmniejszym koszcie wyznaczonym przez wagi połączeń pomiędzy wierzchołkami grafu?

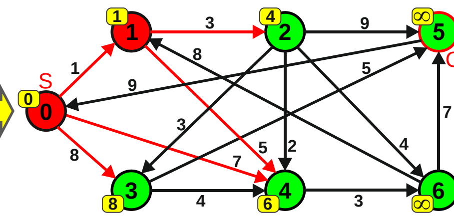


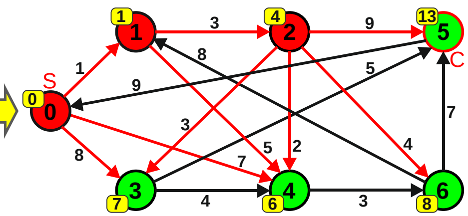
ALGORYTM DIJKSTRY

W algorytmie Dijkstry każdy wierzchołek posiada cechę odległości stałą lub tymczasową, określającą najkrótszą (w sensie sumy dodatnich wag) ścieżkę prowadzącą od wierzchołka początkowego S do wierzchołka końcowego (celu) C. Na początku cechy wszystkich wierzchołków poza startowym są tymczasowe o wartości +∞, natomiast wierzchołek startowy ma cechę stałą o wartości 0. W każdym kroku algorytmu wybieramy wierzchołek u o najmniejszej wartości tymczasowej, przechodzimy po wychodzących z niego krawędziach do pozostałych wierzchołków tymczasowych v i jeśli droga dojścia do tego wierzchołka dist(u) plus wartość krawędzi dist(u,v) jest mniejsza od wartości tymczasowej tego wierzchołka dist(v), wtedy aktualizujemy tą wartość oraz poprzednika Succ(v)=u tego wierzchołka.







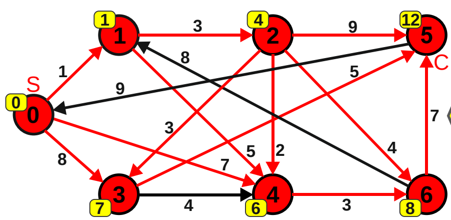


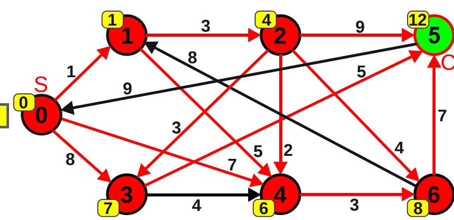
Node u:	0	1	2	3	4	5	6
Dist(u):	0	∞	∞	∞	∞	∞	∞
Succ(u):	-1	-1	-1	-1	-1	-1	-1

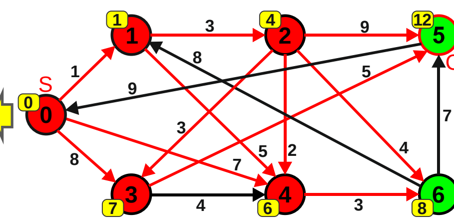
Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	∞	8	7	∞	∞
Succ(u):	-1	0	-1	0	0	-1	-1

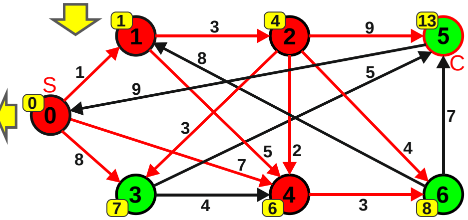
Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	8	6	∞	∞
Succ(u):	-1	0	1	0	1	-1	-1

Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	13	8
Succ(u):	-1	0	1	2	1	2	2









Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	12	8
Succ(u):	-1	0	1	2	1	3	2

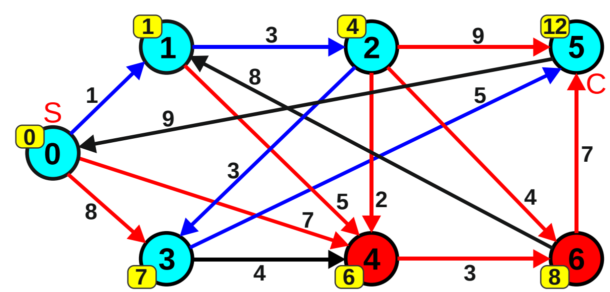
Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	12	8
Succ(u):	-1	0	1	2	1	3	2

Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	12	8
Succ(u):	-1	0	1	2	1	3	2

Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	13	8
Succ(u):	-1	0	1	2	1	2	2

Gdy wierzchołek docelowy C przyjmie wartość stałą, wtedy możemy przerwać algorytm i odczytać wstecz drogę dojścia do tego wierzchołka od wierzchołka startowego S:

Node u:	0	1	2	3	4	5	6
Dist(u):	0	1	4	7	6	12	8
Succ(u):	-1	0	1	2	1	3	2



Przykłady i algorytmy można znaleźć na:

http://eduinf.waw.pl/inf/alg/001_search/0138.php

ALGORYTM FLEURY'EGO

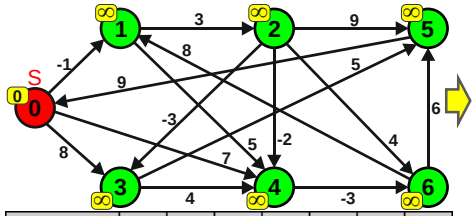
Algorytm 3 ***Fleury'ego*** *szukania ścieżki Eulera w grafie spójnym* $G = (V, E)$.

1. $P := \emptyset$, $H := G$
2. $l :=$ liczba wierzchołków stopnia nieparzystego w grafie
3. jeśli $l = 2$ to $v :=$ dowolny wierzchołek grafu stopnia nieparzystego
   w przeciwnym przypadku jeśli $l = 0$ to $v :=$ dowolny wierzchołek grafu
   w przeciwnym przypadku KONIEC (nie ma ścieżki)
4. wybierz krawędź $e \in H$ incydentną z v, nie będącą (o ile to możliwe) mostem w H
5. dodaj do ścieżki P wierzchołek v i krawędź e
6. $v :=$ drugi koniec krawędzi e
7. usuń e z grafu H
8. jeśli $H \neq \emptyset$ idź do 4
9. dodaj do ścieżki P wierzchołek v

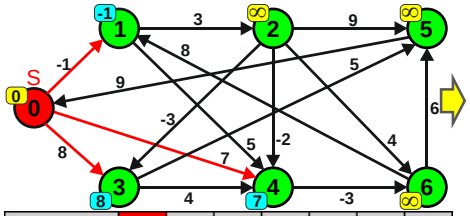
Aby sprawdzić, czy dana krawędź jest mostem można posłużyć się algorytmem Floyda-Warshalla szukania wszystkich dróg w grafie.

ALGORYTM BELLMANA-FORDA

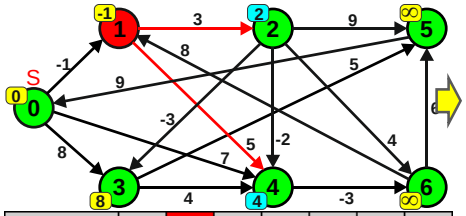
Algorytm Bellmana-Forda ma zastosowanie wtedy, gdy niektóre krawędzie w grafie posiadają krawędzie o wagach ujemnych, jednak nie występuje ujemny cykl (*negative cycle*), który by spowodował możliwość dowolnego skrócenia każdej ścieżki w grafie, która może przejść przez ten cykl. Algorytm ten dokonuje n-1 obiegów wszystkich wierzchołków **u** o drodze dist(u)<∞ celem tzw. **relaksacji krawędzi** powodując wyznaczenie dla każdego incydentnego wierzchołka **v** i łączącej ich krawędzi **u-v** drogi krótszej, jeśli **dist(v) > dist(u) + dist(u-v)**. Wtedy również dla wierzchołka **v** zaznaczamy, iż jego poprzednikiem jest wierzchołek **u**. Jeśli w którymś z cykli nie doszło do żadnej aktualizacji drogi, kończymy algorytm. Jeśli aktualizacji trwały przez n-1 cykli, wtedy należy wykonać jeszcze n-ty obieg, w którym sprawdzamy, czy nie dojdzie do kolejnej aktualizacji najkrótszej drogi któregoś z wierzchołków, co oznacza, iż graf zawiera **cykl ujemny**.



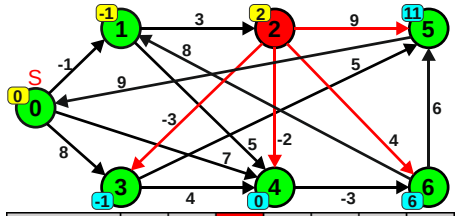
Node u:	0	1	2	3	4	5	6
Dist(u):	0	∞	∞	∞	∞	∞	∞
Succ(u):	-1	-1	-1	-1	-1	-1	-1



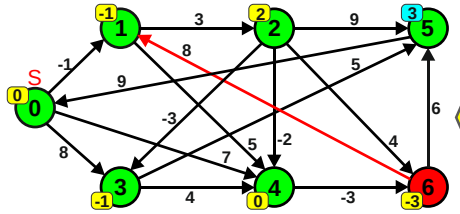
Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	∞	8	7	∞	∞
Succ(u):	-1	0	-1	0	0	-1	-1



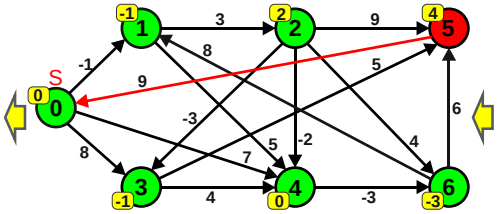
Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	8	4	∞	∞
Succ(u):	-1	0	1	0	1	-1	-1



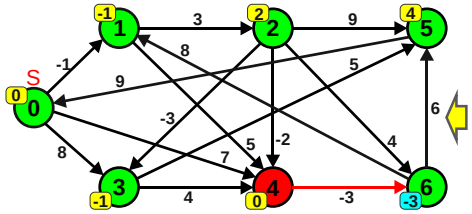
Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	11	6
Succ(u):	-1	0	1	2	2	2	2



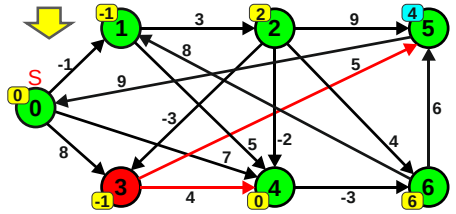
Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	3	-3
Succ(u):	-1	0	1	2	2	6	4



Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	4	-3
Succ(u):	-1	0	1	2	2	3	4



Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	4	-3
Succ(u):	-1	0	1	2	2	3	4



Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	4	6
Succ(u):	-1	0	1	2	2	3	2

W powyższym przykładzie kolejny obieg nie wniesie już żadnych zmian, więc zakończy pozytywnie działanie algorytmu.

Można więc teraz wyznaczyć najkrótszą drogę od wierzchołka 0 do każdego innego, np. wierzchołka nr 5:

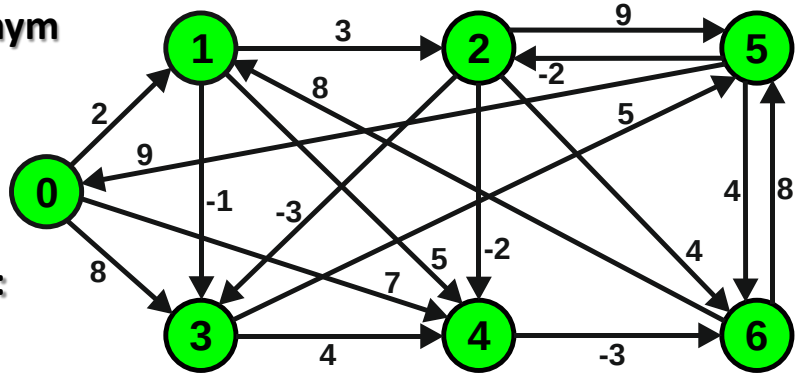
Node u:	0	1	2	3	4	5	6
Dist(u):	0	-1	2	-1	0	3	-3
Succ(u):	-1	0	1	2	2	6	4

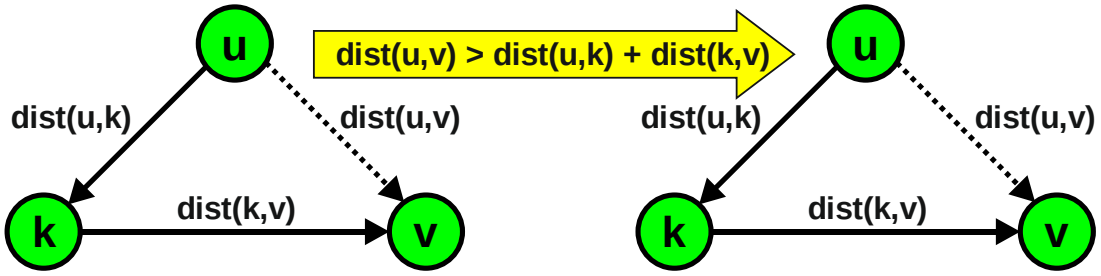
Przykłady i algorytmy można znaleźć na:

http://eduinf.waw.pl/inf/alg/001_search/0138a.php

ALGORYTM FLOYDA-WARSHALLA

Algorytm Floyda-Warshalla umożliwia określenie najkrótszej ścieżki pomiędzy wszystkimi parami wierzchołków w grafie ważonym mogącym zawierać ujemne wagi, lecz nie ujemne cykle.
Algorytm zakłada, iż jeśli droga pośrednia przez jakiś wierzchołek jest krótsza, wtedy należy nią zastąpić tą wcześniej wyznaczoną lub ew. jej brak (waga = ∞), jeśli nie istnieje krawędź bezpośrednio łącząca wierzchołki:





dist	0	1	2	3	4	5	6
0	0	2	∞	8	7	∞	∞
1	∞	0	3	-1	5	∞	∞
2	∞	∞	0	-3	-2	9	4
3	∞	∞	∞	0	4	5	∞
4	∞	∞	∞	∞	0	∞	-3
5	9	∞	-2	∞	∞	0	4
6	∞	8	∞	∞	∞	8	0

Algorytm tworzy macierz N x N, gdzie N to ilość wierzchołków grafu, inicjuje ją wartościami wag krawędzi albo plus nieskończonością, jeśli krawędź nie istnieje, zaś na głównej przekątnej umieszcza zera, wskazujące zerowy koszt przejścia wierzchołka do samego siebie.

Algorytm Floyda-Warshalla następnie wykonuje kolejne iteracje po wierzchołkach pośrednich k oraz po parach wierzchołków w macierzy dokonując aktualizacji długości ścieżek w następujący sposób:

Jeśli dist(u,v) > dist(u,k) + dist(k,v) dochodzi do podstawienia wartości: dist(u,v) = dist(u,k) + dist(k,v).

Algorytm wymaga zastosowania trzech pętli obliczeniowych, gdyż musi wykonać przejść po wszystkich parach wierzchołków (NxN) oraz wszystkich możliwych wierzchołkach pośrednich.

Przykłady i algorytmy można znaleźć na: http://eduinf.waw.pl/inf/alg/001_search/0138b.php

ALGORYTM FLOYDA-WARSHALLA

Algorytm 1 ***Floyda-Warshalla****: szukania najkrótszej drogi pomiędzy wszystkimi parami wierzchołków w grafie ważonym* $G = (V, E, w)$.

dla wszystkich wierzchołków $u, v \in V$ **wykonuj**
  jeśli $e = \{u, v\}$ **jest krawędzią w** G **to**
    $d_u^v := w(e)$
    $p_u^v := v$
  w przeciwnym przypadku
    $d_u^v := \infty$
    $p_u^v :=$ "$niezdefiniowane$"

dla każdego wierzchołka $x \in V$ **wykonuj**
  dla każdego wierzchołka $u \in V$ **wykonuj**
    dla każdego wierzchołka $v \in V$ **wykonuj**
      jeżeli $d_u^v > d_u^x + d_x^v$ **to**
        $d_u^v := d_u^x + d_x^v$
        $p_u^v := p_u^x$

Po zakończeniu działania algorytmu wartość p_u^v *określa kolejny wierzchołek na minimalnej ścieżce z* u *do* v. *Istnieje nieznaczne uproszczenie tego algorytmu, (tzw. algorytm* ***Bellmana-Forda****), które znajduje najkrótszą drogę pomiędzy tylko jedną parą wierzchołków, ale złożoność obu algorytmów jest tego samego rzędu* $O(|V|^3)$.

ALGORYTMY A*

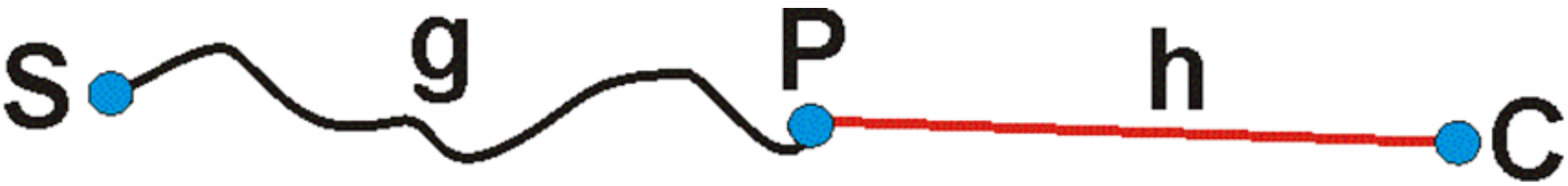
Nawigacje satelitarne nie używają zwykle algorytmu Dijkstry, gdyż trwałoby to za długo!

Algorytm A* SEARCH stosowny w GPSach oraz maps.google.pl do wyznaczania najkrótszej drogi w grafie pomiędzy dwoma jego wierzchołkami S i C.

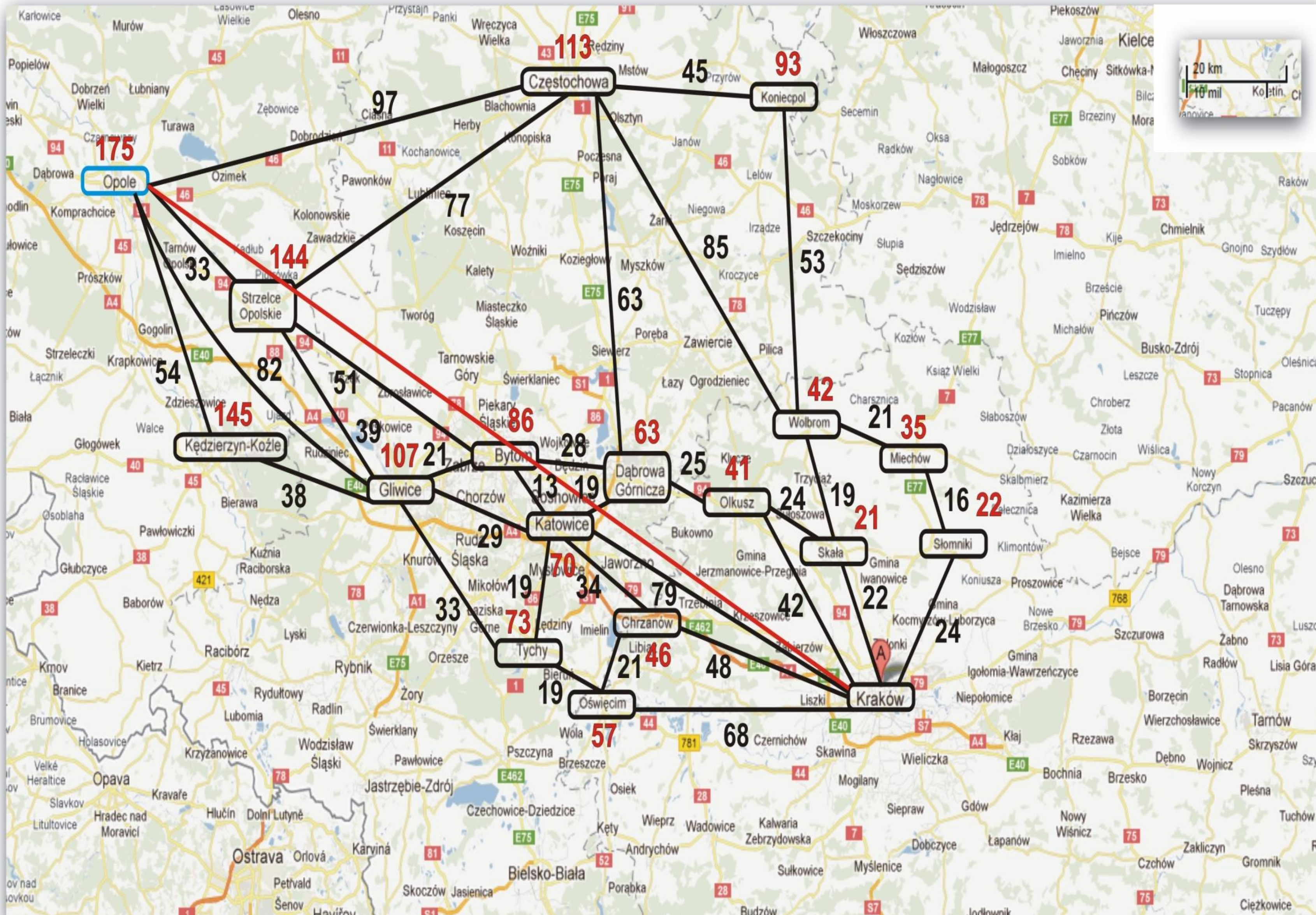
Algorytm A* minimalizuje funkcję kosztu: f = g + h

g – to faktycznie przebyta odległość od wierzchołka początkowego P do aktualnego w grafie

h – to szacowana optymistyczna, nie większa niż rzeczywista odległość od aktualnego wierzchołka w grafie do wierzchołka celu C

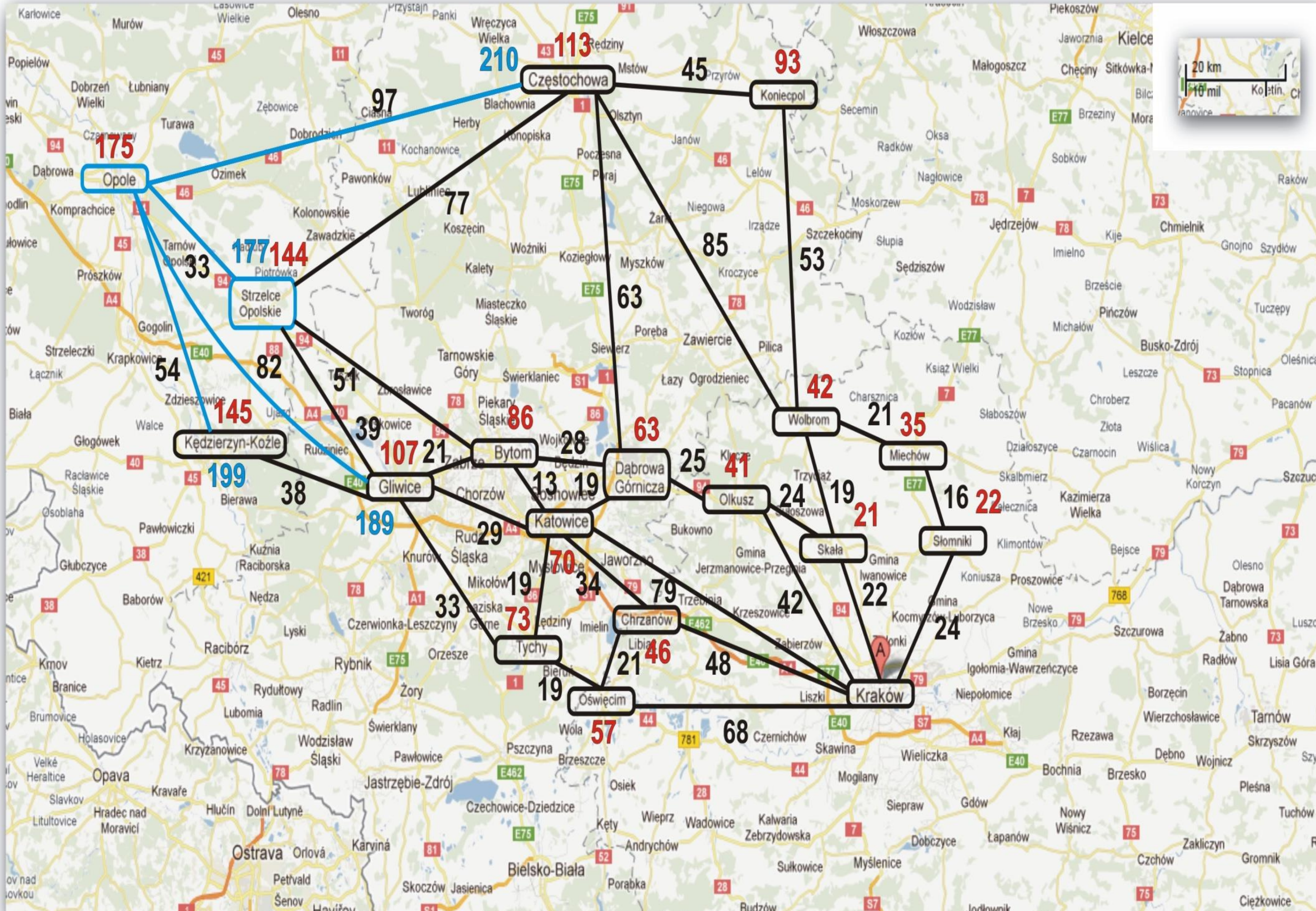


ALGORYTM A*



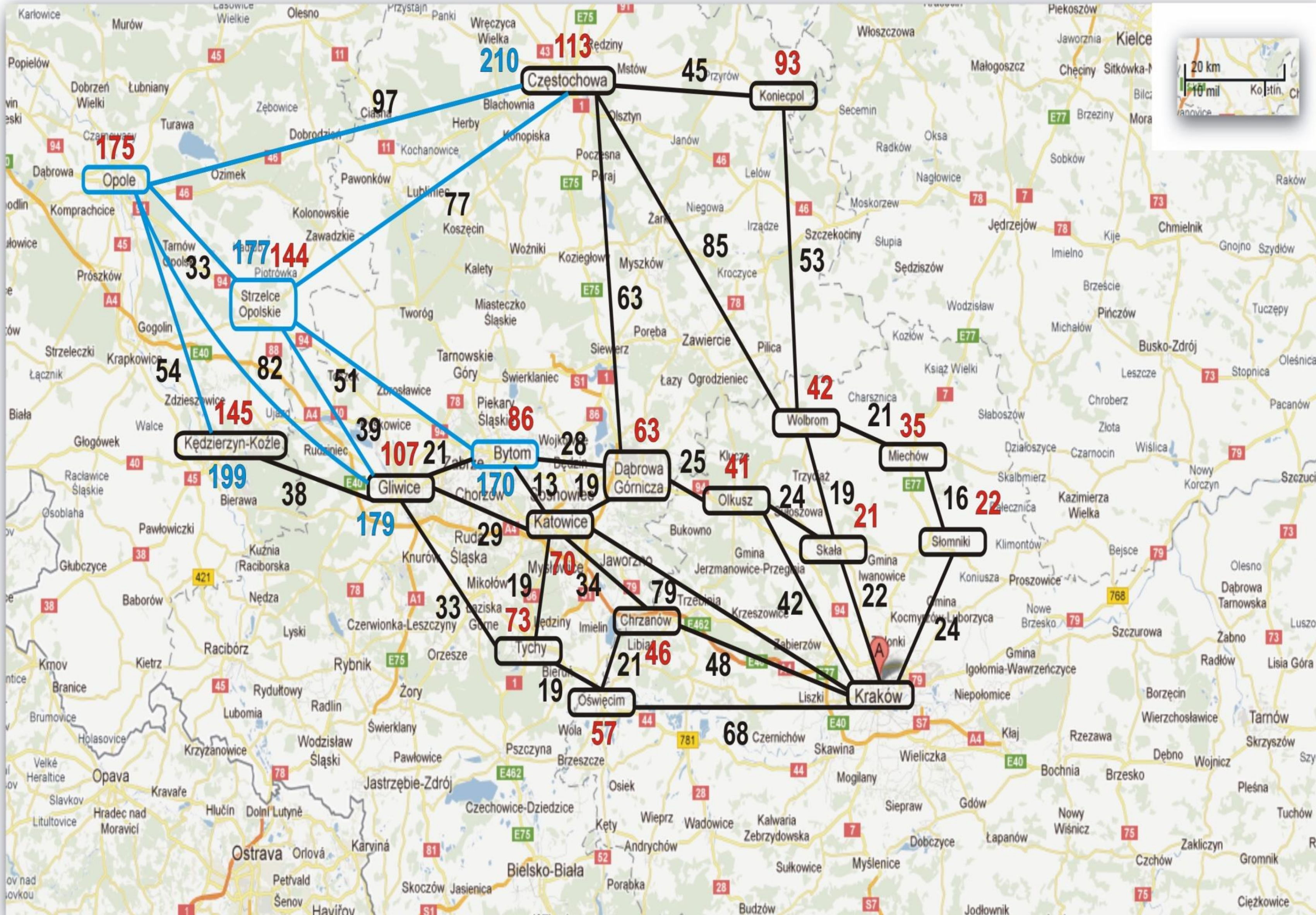
Opole
175
Częstochowa
113
Koniecpol
93
Strzelce Opolskie
144
Kędzierzyn-Koźle
145
Gliwice
107
Bytom
86
Dąbrowa Górnicza
63
Olkusz
41
Wolbrom
42
Miechów
35
Skała
21
Słomniki
22
Katowice
70
Tychy
73
Chrzanów
46
Oświęcim
57
Kraków
97
45
77
85
53
63
33
54
82
51
39
21
28
13
19
25
24
19
21
16
29
19
34
79
42
22
24
33
21
48
19
68
38
20 km
10 mil

ALGORYTM A*



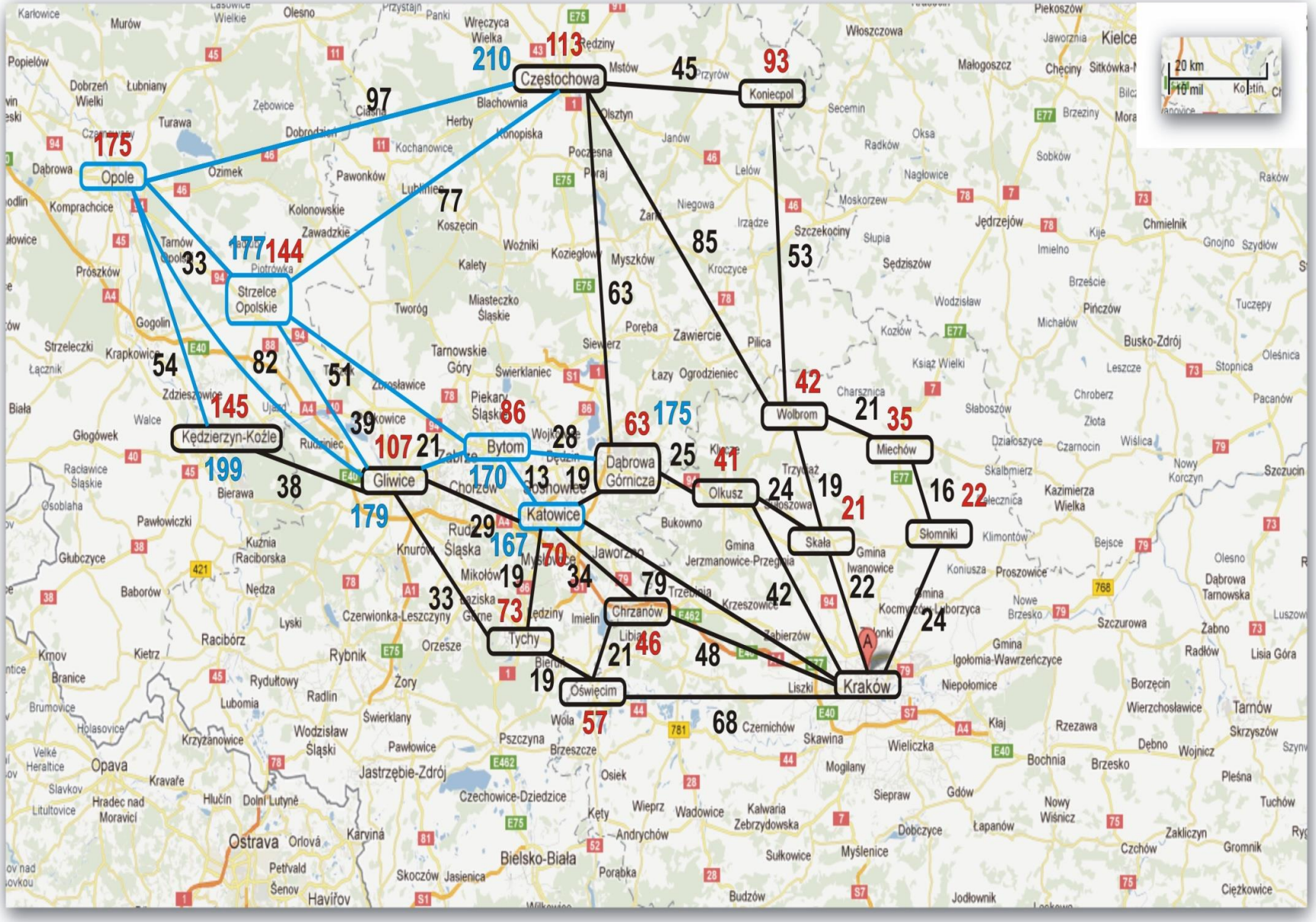
Częstochowa
210
113
Koniecpol
93
Opole
175
Strzelce Opolskie
177
144
Kędzierzyn-Koźle
145
199
Gliwice
189
107
Bytom
86
Dąbrowa Górnicza
63
Olkusz
41
Wolbrom
42
Miechów
35
Słomniki
22
Skała
21
Katowice
70
Tychy
73
Chrzanów
46
Oświęcim
57
Kraków
97
45
77
85
53
63
33
54
82
51
39
21
28
38
13
19
25
24
19
21
16
29
19
34
79
42
22
24
33
21
48
19
68
20 km
10 mil

ALGORYTM A*

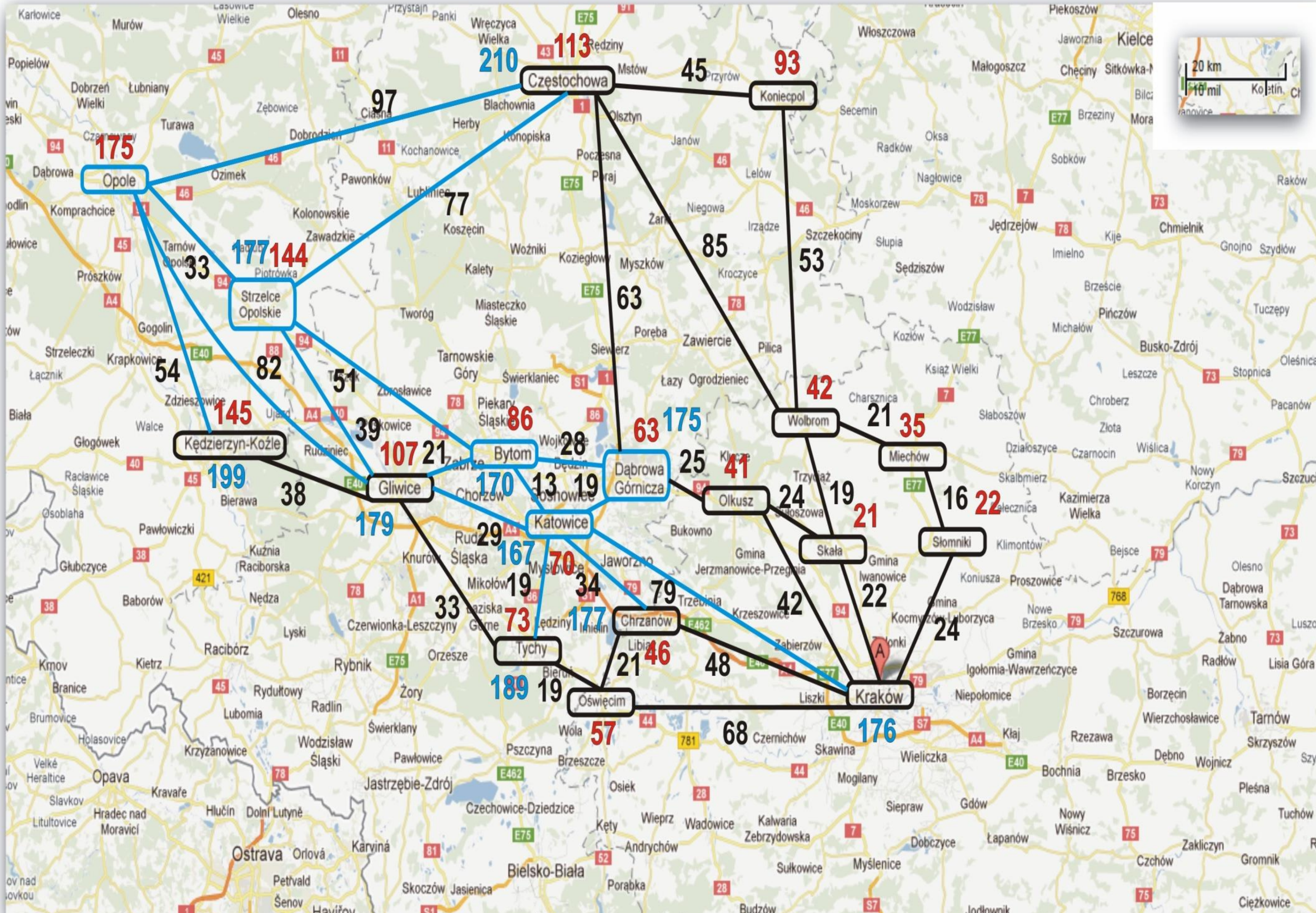


Częstochowa
210
113
Koniecpol
93
45
Opole
175
97
77
Strzelce Opolskie
177
144
33
85
53
63
54
82
51
Wolbrom
42
21
Miechów
35
Kędzierzyn-Koźle
145
199
39
107
21
Bytom
86
170
28
Dąbrowa Górnicza
63
25
Olkusz
41
24
19
Skała
21
16
Słomniki
22
Gliwice
179
38
13
19
Katowice
29
70
34
79
42
22
24
19
73
33
Chrzanów
Tychy
21
46
48
19
Oświęcim
57
68
Kraków

ALGORYTM A*

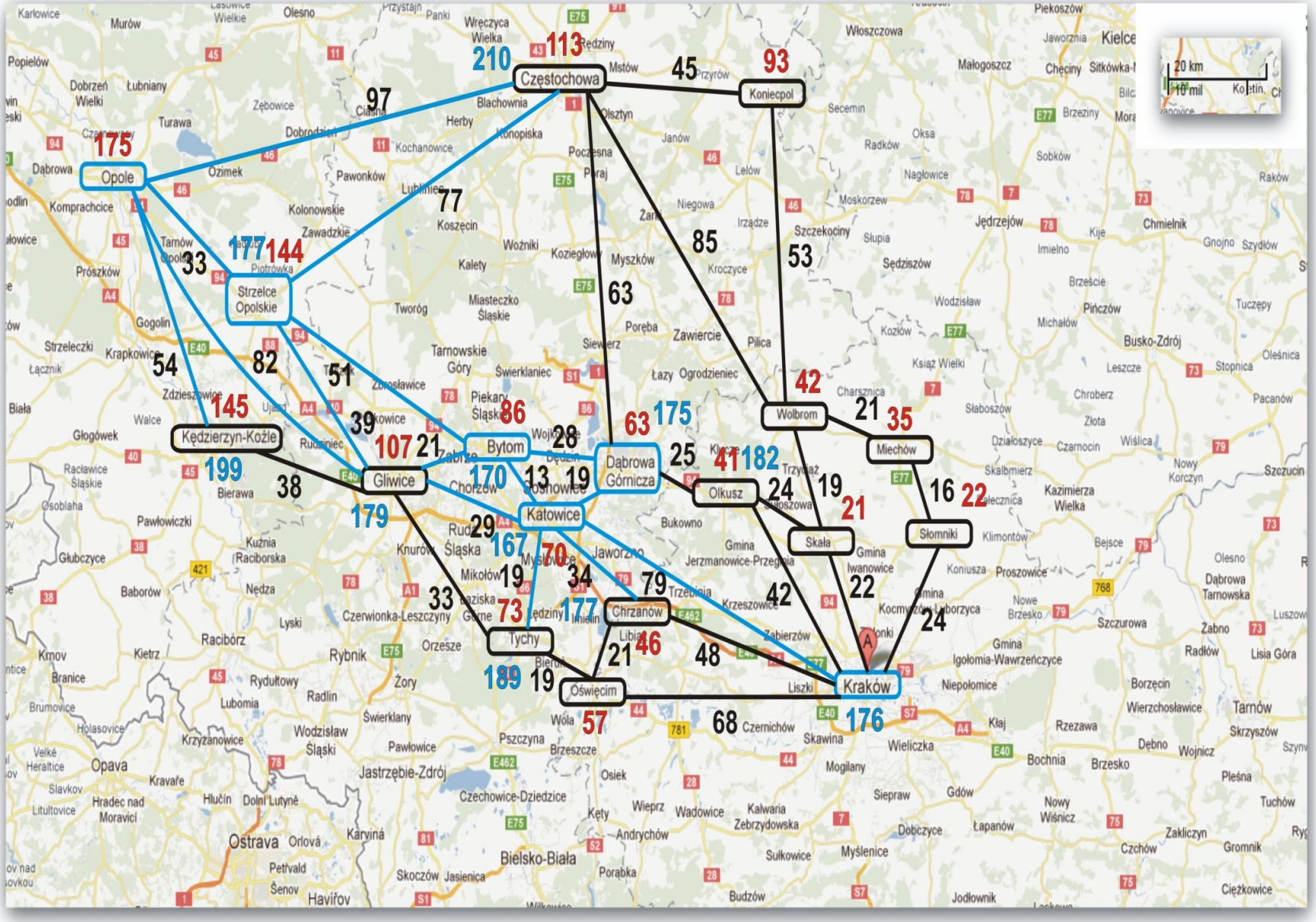

Częstochowa
Koniecpol
Wolbrom
Miechów
Słomniki
Kraków
Skała
Olkusz
Dąbrowa Górnicza
Katowice
Bytom
Gliwice
Chrzanów
Oświęcim
Tychy
Kędzierzyn-Koźle
Strzelce Opolskie
Opole
20 km
10 mil

ALGORYTM A*

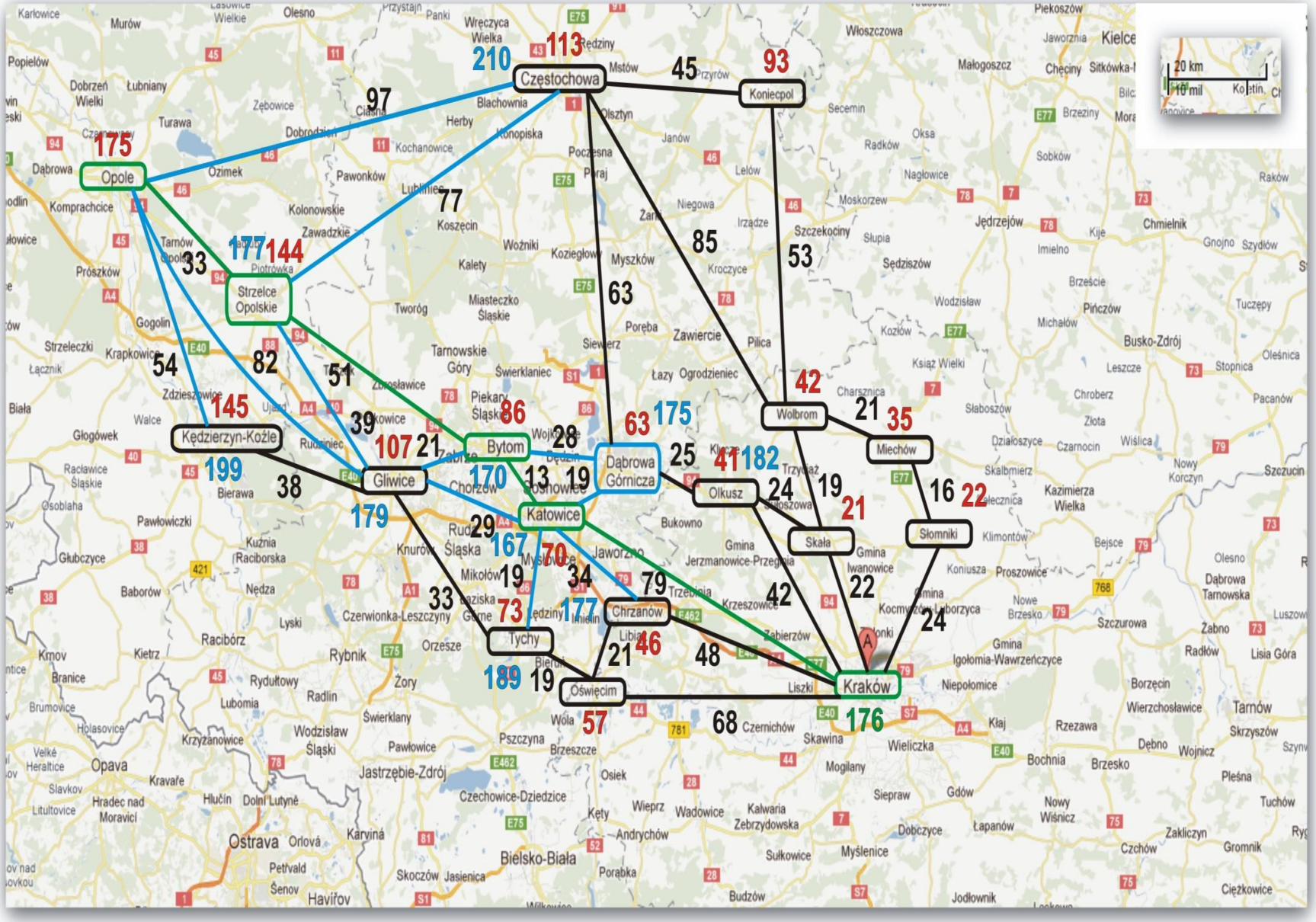


Częstochowa
Koniecpol
Opole
Strzelce Opolskie
Kędzierzyn-Koźle
Gliwice
Bytom
Dąbrowa Górnicza
Katowice
Olkusz
Wolbrom
Miechów
Skała
Słomniki
Chrzanów
Tychy
Oświęcim
Kraków
210
113
45
93
175
97
77
177
144
33
85
53
63
82
54
51
145
39
107
21
86
28
63
175
42
21
35
199
38
170
13
19
25
41
24
19
21
16
22
179
29
167
70
19
34
79
42
22
24
33
73
177
46
21
48
189
19
57
68
176
20 km
10 mil

ALGORYTM A*


Częstochowa
Koniecpol
Wolbrom
Miechów
Słomniki
Kraków
Skała
Olkusz
Dąbrowa Górnicza
Katowice
Bytom
Chrzanów
Oświęcim
Tychy
Gliwice
Kędzierzyn-Koźle
Strzelce Opolskie
Opole
113
210
45
93
97
175
85
53
63
77
177
144
33
54
82
51
145
39
107
21
86
28
63
175
42
21
35
199
38
179
170
13
19
25
41
182
24
19
21
16
22
29
167
70
34
79
22
19
73
177
42
24
21
46
48
189
19
57
68
176
Murów
Olesno
Wręczyca Wielka
Popielów
Dobrzeń Wielki
Łubniany
Turawa
Dąbrowa
Komprachcice
Ozimek
Pawonków
Kolonowskie
Zawadzkie
Prószków
Gogolin
Strzeleczki
Krapkowice
Zdzieszowice
Biała
Walce
Głogówek
Racławice Śląskie
Osobłaha
Pawłowiczki
Głubczyce
Baborów
Kietrz
Krnov
Branice
Brumovice
Holasovice
Velké Heraltice
Opava
Slavkov
Litultovice
Hradec nad Moravicí
Hlučín
Dolní Lutyně
Ostrava
Orlová
Karviná
Petřvald
Šenov
Havířov
Skoczów
Jasienica
Bielsko-Biała
Porąbka
Budzów
Jodłownik
Ciężkowice
Myślenice
Sułkowice
Dobczyce
Łapanów
Siepraw
Gdów
Nowy Wiśnicz
Zakliczyn
Czchów
Gromnik
Tuchów
Pleśna
Brzesko
Bochnia
Mogilany
Skawina
Wieliczka
Czernichów
Kłaj
Rzezawa
Dębno
Wojnicz
Skrzyszów
Tarnów
Wierzchosławice
Borzęcin
Niepołomice
Igołomia-Wawrzeńczyce
Liszki
Zabierzów
Krzeszowice
Trzebinia
Jaworzno
Mysłowice
Chorzów
Bytom
Ruda Śląska
Knurów
Gliwice
Rudziniec
Sośnicowice
Wojkowice
Dąbrowa Górnicza
Olkusz
Trzyniec
Klucze
Bukowno
Jerzmanowice-Przegibek
Gmina Iwanowice
Proszowice
Koniusza
Kocmyrzów-Luborzyca
Nowe Brzesko
Szczurowa
Żabno
Radłów
Lisia Góra
Dąbrowa Tarnowska
Olesno
Klimontów
Bejsce
Sandomierz
Kazimierza Wielka
Działoszyce
Czarnocin
Wiślica
Nowy Korczyn
Szczucin
Skalbmierz
Słaboszów
Złota
Pacanów
Chroberz
Charsznica
Książ Wielki
Busko-Zdrój
Stopnica
Leszcze
Oleśnica
Michałów
Pińczów
Tuczępy
Kozłów
Wodzisław
Brzeście
Sędziszów
Szczekociny
Słupia
Jędrzejów
Chmielnik
Gnojno
Szydłów
Imielno
Nagłowice
Moskorzew
Radków
Oksa
Secemin
Włoszczowa
Małogoszcz
Jaworznia
Kielce
Checiny
Sitkówka-Nowiny
Pokoszów
Raków
Sobków
Brzeziny
Mora
Bilc
Kostrzyn
Janów
Lelów
Olsztyn
Mstów
Przyrów
Poczesna
Blachownia
Herby
Konopiska
Kochanowice
Zębowice
Ciasna
Dobrodzień
Lubliniec
Koszęcin
Kalety
Woźniki
Koziegłowy
Myszków
Żarki
Niegowa
Irządze
Kroczyce
Poraj
Tworóg
Miasteczko Śląskie
Tarnowskie Góry
Świerklaniec
Siewierz
Poręba
Zawiercie
Pilica
Łazy
Ogrodzieniec
Piekary Śląskie
Zbrosławice
Ujazd
Łącznik
Lasowice Wielkie
Przystajń
Panki
Rędziny
Wola
Pszczyna
Brzeszcze
Osiek
Kęty
Wieprz
Wadowice
Kalwaria Zebrzydowska
Andrychów
Czechowice-Dziedzice
Jastrzębie-Zdrój
Wodzisław Śląski
Pawłowice
Świerklany
Lubomia
Radlin
Rydułtowy
Żory
Rybnik
Orzesze
Racibórz
Lyski
Czerwionka-Leszczyny
Nędza
Kuźnia Raciborska
Mikołów
Łaziska Górne
Chorzów
Siemianowice
Piekary Śląskie
Zabrze
Gliwice
Rudziniec
Wojkowice
Sosnowiec
Dąbrowa Górnicza
Trzyniec
Klucze
Olkusz
Bukowno
Jerzmanowice-Przegibek
Gmina Iwanowice
Proszowice
Koniusza
Kocmyrzów-Luborzyca
Nowe Brzesko
Szczurowa
Żabno
Radłów
Lisia Góra
Dąbrowa Tarnowska
Olesno
Klimontów
Bejsce
Sandomierz
Kazimierza Wielka
Działoszyce
Czarnocin
Wiślica
Nowy Korczyn
Szczucin
Skalbmierz
Słaboszów
Złota
Pacanów
Chroberz
Charsznica
Książ Wielki
Busko-Zdrój
Stopnica
Leszcze
Oleśnica
Michałów
Pińczów
Tuczępy
Kozłów
Wodzisław
Brzeście
Sędziszów
Szczekociny
Słupia
Jędrzejów
Chmielnik
Gnojno
Szydłów
Imielno
Nagłowice
Moskorzew
Radków
Oksa
Secemin
Włoszczowa
Małogoszcz
Jaworznia
Kielce
Checiny
Sitkówka-Nowiny
Pokoszów
Raków
Sobków
Brzeziny
Mora
Bilc
Kostrzyn
Janów
Lelów
Olsztyn
Mstów
Przyrów
Poczesna
Blachownia
Herby
Konopiska
Kochanowice
Zębowice
Ciasna
Dobrodzień
Lubliniec
Koszęcin
Kalety
Woźniki
Koziegłowy
Myszków
Żarki
Niegowa
Irządze
Kroczyce
Poraj
Tworóg
Miasteczko Śląskie
Tarnowskie Góry
Świerklaniec
Siewierz
Poręba
Zawiercie
Pilica
Łazy
Ogrodzieniec
Piekary Śląskie
Zbrosławice
Ujazd
Łącznik
Lasowice Wielkie
Przystajń
Panki
Rędziny
Wola
Pszczyna
Brzeszcze
Osiek
Kęty
Wieprz
Wadowice
Kalwaria Zebrzydowska
Andrychów
Czechowice-Dziedzice
Jastrzębie-Zdrój
Wodzisław Śląski
Pawłowice
Świerklany
Lubomia
Radlin
Rydułtowy
Żory
Rybnik
Orzesze
Racibórz
Lyski
Czerwionka-Leszczyny
Nędza
Kuźnia Raciborska
Mikołów
Łaziska Górne
Chorzów
Siemianowice
20 km
10 mil

ALGORYTM A*


Częstochowa
Koniecpol
Opole
Strzelce Opolskie
Kędzierzyn-Koźle
Gliwice
Bytom
Dąbrowa Górnicza
Olkusz
Wolbrom
Miechów
Słomniki
Skała
Katowice
Chrzanów
Tychy
Oświęcim
Kraków
210
113
45
93
175
97
77
85
53
177
144
33
63
54
82
51
42
145
86
63
175
21
35
39
107
21
28
25
41
182
199
170
13
19
24
19
16
22
38
179
29
167
70
34
79
42
22
21
33
73
177
46
21
48
24
189
19
68
57
176

TABELE i TABLICE kontra GRAFY!

FAKT: Najczęściej dane przechowujemy w tablicach i tabelach (bazodanowych).

PYTANIE: Dlaczego?

MOŻLIWE ODPOWIEDZI I MITY:

- Ponieważ to proste i intuicyjne.
- Tak nas nauczono w szkole / na studiach.
- Dzięki bezpośredniemu adresowaniu poszczególnych komórek pamięci mamy błyskawiczny (w czasie stałym) dostęp do poszczególnych danych w przechowywanych w tabelach.

	ATTRIBUTES				
SAMPLE OBJECTS	SEPAL LENGTH	SEPAL WIDTH	PETAL LENGTH	PETAL WIDTH	CLASS LABEL
O1	5.4	3.0	4.5	1.5	Versicolor
O2	6.3	3.3	4.7	1.6	Versicolor
O3	6.0	2.7	5.1	1.6	Versicolor
O4	6.7	3.0	5.0	1.7	Versicolor
O5	6.0	2.2	5.0	1.5	Virginica
O6	5.9	3.2	4.8	1.8	Versicolor
O7	6.0	3.0	4.8	1.8	Virginica
O8	5.7	2.5	5.0	2.0	Virginica
O9	6.5	3.2	5.1	2.0	Virginica

- W tabelach wszystko widzimy lub możemy łatwo wyszukać.
- Tabelę możemy przechować przy pomocy tablic w pamięci RAM, która ma strukturę tablicy bajtów, oraz na dysku, który ma strukturę powiązanych sekwencyjnie bloków, które też są tablicami, więc najlepiej się nadają do ich przechowywania!
- (Prawie) wszyscy tak robią (zasada owczego pędu).

ZDERZENIE Z FAKTAMI Z BIOLOGII: Dlaczego w naszym mózgu nie ma tabel?

Dlaczego biologiczne systemu używają grafowych neuronowych struktur do przechowywania danych, informacji i wiedzy a nie tabel czy tablic?

PRAWDZIWE OBLICZE TABEL I TABLIC

TABELE I TABLICE:

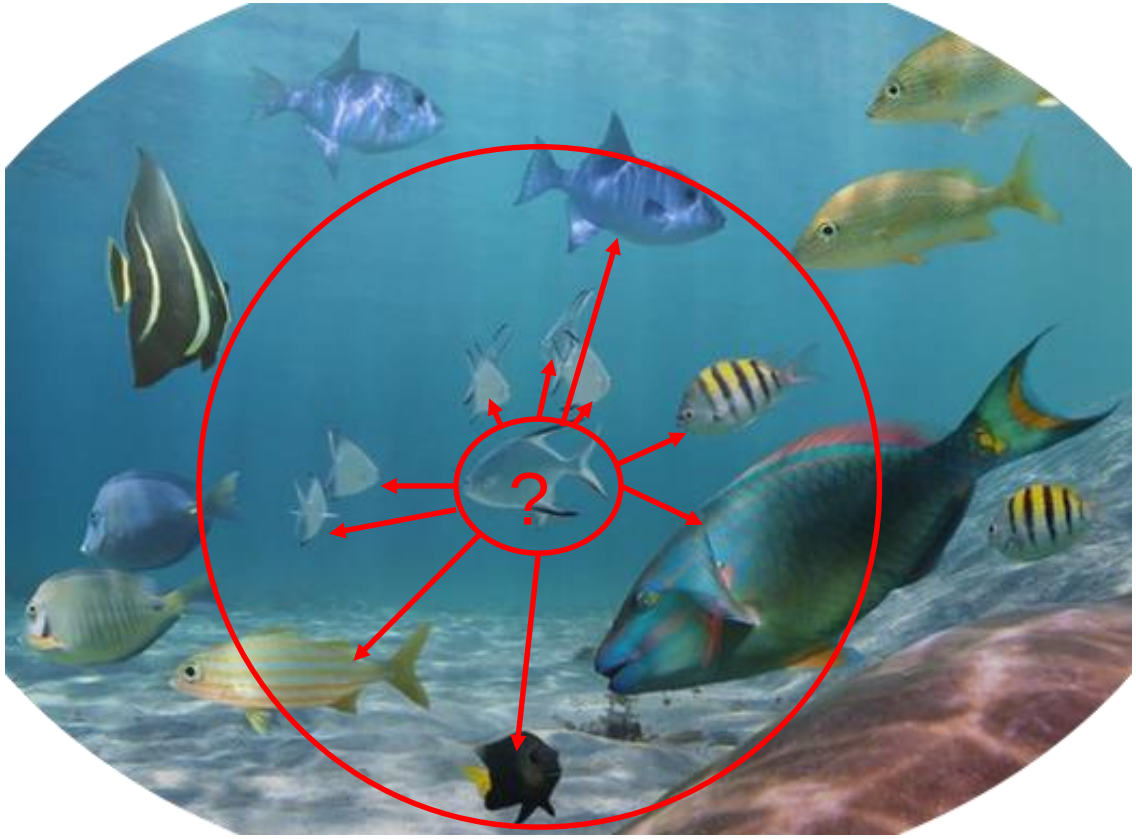
- Tak, są bardzo proste dlatego uniemożliwiają efektywną reprezentację bogactwa relacji pomiędzy danymi, np. relacji wertykalnych, tj. minima, maksima, sąsiedztwa, podobieństwa, duplikacji, średnich, median, sum…!
- Nie można równocześnie posortować wartości wszystkich atrybutów tabel i tablic.
- Intuicyjne?! Kto z Państwa chce się nauczyć na pamięć kilku tabelek danych na egzamin ze Wstępu do informatyki?
- Tak, dostęp jest szybki, gdyż pamięci komputerowe projektowano w oparciu o tablice, ALE już bardzo powszechne operacje wstawiania czy usuwania z posortowanych tablic/tabel danych wymagają sporo (liniowego) czasu!

	ATTRIBUTES				
SAMPLE OBJECTS	SEPAL LENGTH	SEPAL WIDTH	PETAL LENGTH	PETAL WIDTH	CLASS LABEL
O1	5.4	3.0	4.5	1.5	Versicolor
O2	6.3	3.3	4.7	1.6	Versicolor
O3	6.0	2.7	5.1	1.6	Versicolor
O4	6.7	3.0	5.0	1.7	Versicolor
O5	6.0	2.2	5.0	1.5	Virginica
O6	5.9	3.2	4.8	1.8	Versicolor
O7	6.0	3.0	4.8	1.8	Virginica
O8	5.7	2.5	5.0	2.0	Virginica
O9	6.5	3.2	5.1	2.0	Virginica

- W tabelach praktycznie nic nie widzimy, co najwyżej definicje poszczególnych obiektów/rekordów/encji zdefiniowanych za pośrednictwem wartości zbioru atrybutów, NATOMIAST całą resztę często pożądanych relacji MUSIMY SZUKAĆ!
- Szukanie w tablicach to wykonywanie różnych (czasami zagnieżdżonych) pętli i ewaluacja różnych warunków, a to kosztuje cenny czas obliczeń i niepotrzebnie zwiększa złożoność obliczeniową algorytmów operujących na tablicach i tabelach.
- Polak jest przekorny. Zróbmy inaczej, mądrzej… wzorujmy się na naszym mózgu!

PORÓWNANIE SZYBKOŚCI I EFEKTYWNOŚCI

Porównajmy szybkość działania algorytmu K Najbliższych Sąsiadów, stosowanego do klasyfikacji obiektów, operującego na tabelach oraz grafach AGDS:

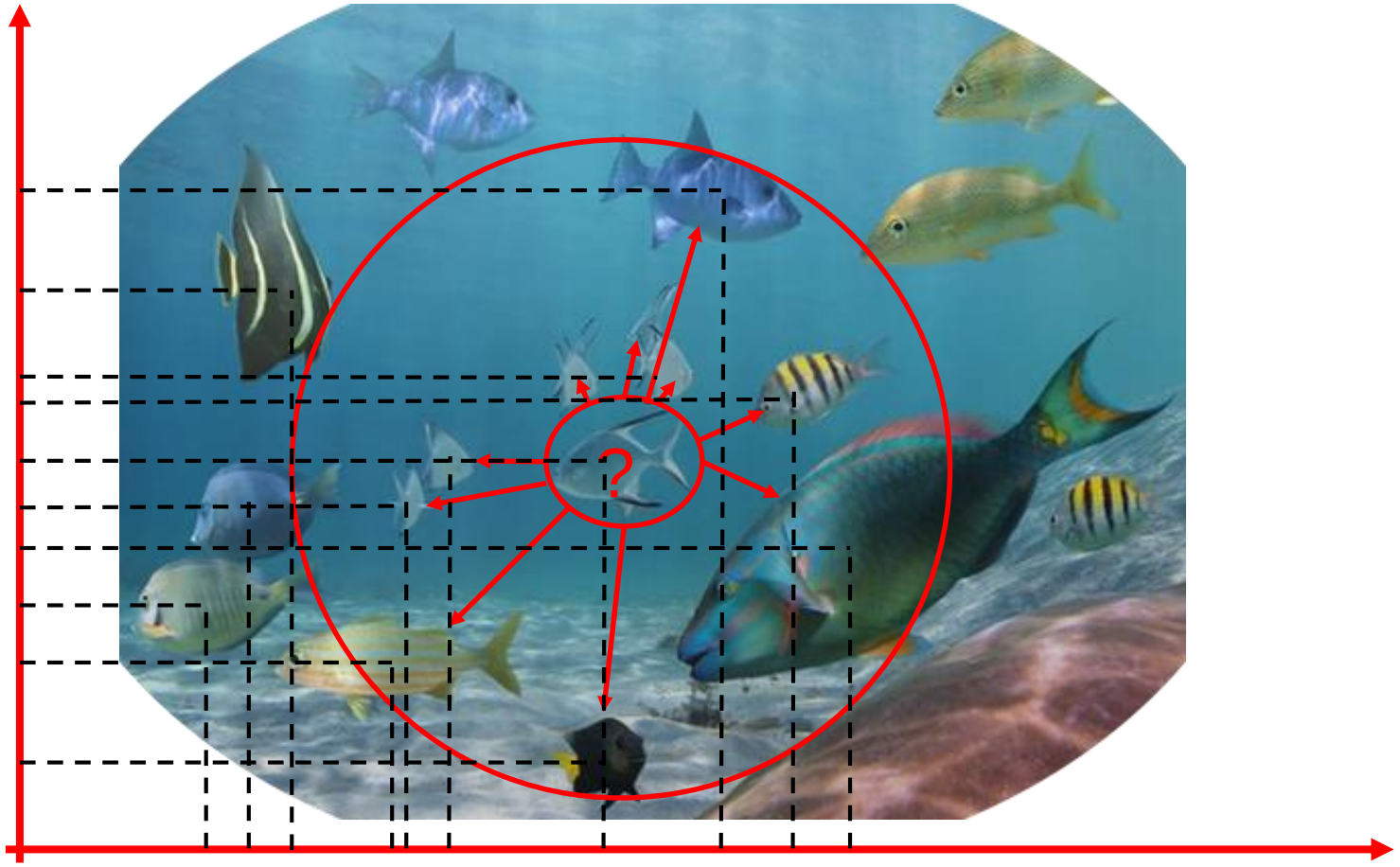


- **Zadaniem jest określenie klasy (grupy, rodzaju) obiektu na podstawie znanych klas (grup, rodzajów) jego k najbliższych sąsiadów.**
- **Operując na tabeli znanych obiektów, musimy je w pętli wszystkie przejrzeć i obliczyć odległość (np. Euklidesa) klasyfikowanego obiektu do nich, gdyż tabela nie udostępnia nam wiedzy o sąsiedztwie (relacji), więc musimy szukać w pętli!**

GRAFY AGDS JAKO MODEL PRZESTRZENI

Jeśli wykorzystamy asocjacyjną grafową strukturę danych AGDS, wtedy:

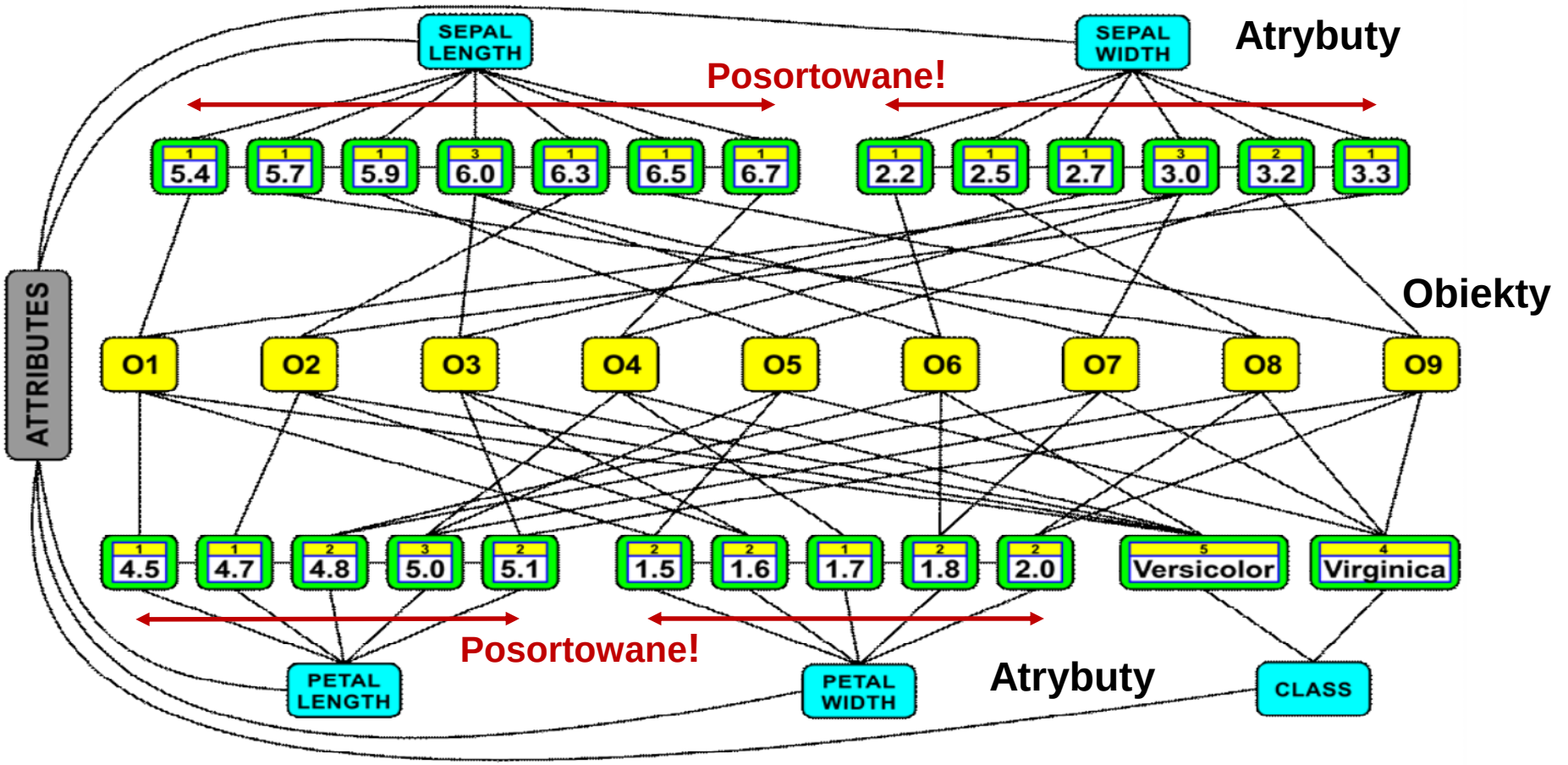
- **Grafowa uporządkowana organizacja obiektów względem ich położenia w dowolnej N-wymiarowej przestrzeni pozwoli nam na szybkie odnalezienie najbliższych sąsiadów niezależnie od wielkości zbioru przechowującego obiekty!**
- **W AGDSach przeszukujemy tylko sąsiednie obiekty w przestrzeni:**



ASOCJACYJNE GRAFOWE STRUKTURY DANYCH

W podstawowej wersji Asocjacyjne Grafowe Struktury Danych AGDS składają się z:

- **węzłów reprezentujących zagregowane, policzone i posortowane wartości poszczególnych atrybutów obiektów (rekordów, encji),**
- **węzłów reprezentujących zagregowane i policzone (jeśli w zbiorze danych występują duplikaty obiektów) obiekty (rekordy, encje).**



TRANSFORMACJA TABEL DO GRAFÓW AGDS

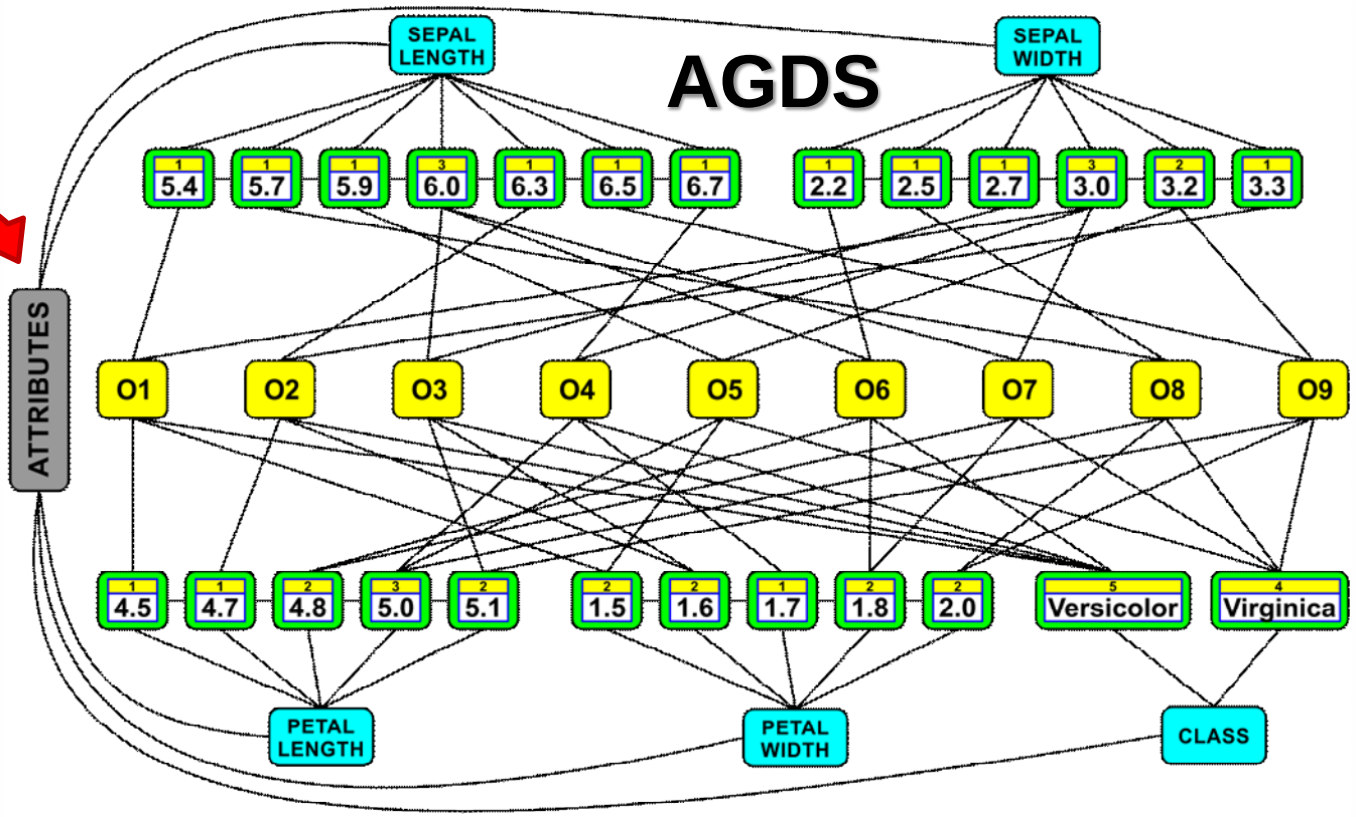
Chcąc uzyskać wyższą efektywność obliczeń na rzeczywistych danych, możemy dokonać asocjacyjnej transformacji tabel lub tablic do grafowej struktury danych AGDS, która umożliwia taką ich reprezentację, iż minima, maksima, wartości sąsiednie i obiekty podobne będą dostępne w czasie stałym, a wyznaczenie niektórych relacji wertykalnych, tj. sumy, średnie czy mediany, będzie szybsze niż dla tablic w przypadku, gdy dane zawierają duplikaty:

$$SUM = \sum_{ELEMENTS} KEY \cdot COUNTER$$

$$COUNT = \sum_{ELEMENTS} COUNTER$$

$$AVERAGE = SUM / COUNT$$

SAMPLE OBJECTS	ATTRIBUTES: SEPAL LENGTH	SEPAL WIDTH	PETAL LENGTH	PETAL WIDTH	CLASS LABEL
O1	5.4	3.0	4.5	1.5	Versicolor
O2	6.3	3.3	4.7	1.6	Versicolor
O3	6.0	2.7	5.1	1.6	Versicolor
O4	6.7	3.0	5.0	1.7	Versicolor
O5	6.0	2.2	5.0	1.5	Virginica
O6	5.9	3.2	4.8	1.8	Versicolor
O7	6.0	3.0	4.8	1.8	Virginica
O8	5.7	2.5	5.0	2.0	Virginica
O9	6.5	3.2	5.1	2.0	Virginica



PORÓWNANIE WYNIKÓW KNN dla TABEL i AGDS

Poniższa tabela prezentuje porównanie wyników klasyfikacji metodą K Najbliższych Sąsiadów operującą na różnych zbiorach danych przechowywanych w tabeli oraz w grafach AGDS. Dla tabel czas obliczeń rośnie wraz z ilością danych, zaś dla grafów jest prawie stały:

Dataset	Number of instances	Number of attributes	kNN classification time [ms]	kNN+AGDS classification time [ms]	kNN+AGDS construction time [ms]
Iris	150	4	0.10	0.08	1
Banknote	1372	4	0.29	0.09	5
HTRU2	17898	8	3.14	0.09	134
Shuttle	43500	9	7.67	1.06	278
Credit Card	30000	23	8.69	1.07	499
Skin	245057	3	26.87	1.10	683
Drive	58509	48	46.15	1.24	2224
HEPMASS	1048576	28	362.32	1.41	31214

SORTOWANIE TOPOLOGICZNE GRAFÓW

Sortowanie topologiczne (*topological sort*) polega na utworzeniu listy wierzchołków grafu skierowanego acyklicznego (*directed acyclic graph*) ułożonych w taki sposób, aby każdy wierzchołek znalazł się w liście przed swoimi sąsiadami.

Przykłady i algorytmy można znaleźć: http://eduinf.waw.pl/inf/alg/001_search/0137.php

Inne ciekawe, klasyczne i zalecane do przestudiowania zagadnienia z teorii grafów:

1. **Problem chińskiego listonosza: http://eduinf.waw.pl/inf/alg/001_search/0139.php**
2. **Problem komiwojażera: http://eduinf.waw.pl/inf/alg/001_search/0140.php**
3. **Algorytm Kruskala tworzenia minimalnego drzewa rozpinającego nad grafem: http://eduinf.waw.pl/inf/alg/001_search/0141.php**
4. **Problem optymalnego kolorowania grafu: http://eduinf.waw.pl/inf/alg/001_search/0142.php**
5. **Znajdowanie podgrafów pełnych (klik) w grafie: http://eduinf.waw.pl/inf/alg/001_search/0143.php**

BIBLIOGRAFIA I LITERATURA UZUPEŁNIAJĄCA

1. L. Banachowski, K. Diks, W. Rytter: „Algorytmy i struktury danych”, WNT, Warszawa, 2001.
2. Z. Fortuna, B. Macukow, J. Wąsowski: „Metody numeryczne”, WNT, Warszawa, 1993.
3. J. i M. Jankowscy: „Przegląd metod i algorytmów numerycznych”, WNT, Warszawa, 1988.
4. A. Kiełbasiński, H. Schwetlick: „Numeryczna algebra liniowa”, WNT, Warszawa 1992.
5. M. Sysło: „Elementy Informatyki”.
6. A. Szepietowski: „Podstawy Informatyki”.
7. R. Tadeusiewicz, P. Moszner, A. Szydełko: „Teoretyczne podstawy informatyki”.
8. W. M. Turski: „Propedeutyka informatyki”.
9. N. Wirth: „Wstęp do programowania systematycznego”.
10. N. Wirth: „ALGORYTMY + STRUKTURY DANYCH = PROGRAMY”.
11. Grafy: http://www.redblobgames.com/pathfinding/a-star/implementation.html
12. Złożoność obliczeniowa: http://xion.org.pl/files/texts/mgt/pdf/M_B.pdf
13. Sortowanie w Pythonie: https://docs.python.org/2/howto/sorting.html?highlight=complexity
14. Algorytmy i struktury danych: http://eduinf.waw.pl/inf/alg/001_search/0123.php
15. A. Horzyk, Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja, monografia, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2013.
16. A. Horzyk, Associative Graph Data Structures with an Efficient Access via AVB+trees, In: 2018 11th International Conference on Human System Interaction (HSI 2018), IEEE Xplore, pp. 169 - 175, 2018, DOI: 10.1109/HSI.2018.8430973.
17. A. Horzyk and K. Gołdon, Associative Graph Data Structures Used for Acceleration of K Nearest Neighbor Classifiers, In: 27th International Conference on Artificial Neural Networks (ICANN 2018), Springer-Verlag, LNCS 11139, pp. 648-658, 2018.