MINUT 2025 (7), s. 192–205
ISSN 2719-3063

Bartosz KUCHAREWICZ[1], Marta KUCHARCZYK[2], Bona WIECZOREK[3]

[1]Student, Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska, ul.Akademicka 16, 44-100 Gliwice

[2] Studentka, Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska, ul.Akademicka 16, 44-100 Gliwice

[3]Katedra Algorytmiki i Oprogramowania, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice

Zastosowanie struktury zbiorów rozłącznych w algorytmie Kruskala

Streszczenie. We współczesnym świecie wiele problemów związanych z optymalizacją połączeń w sieciach komputerowych, energetycznych czy drogowych znajduje matematyczne rozwiązanie w teorii grafów. Jednym z najistotniejszych zagadnień w tej dziedzinie jest wyznaczanie minimalnego drzewa rozpinającego (ang. *Minimum Spannig Tree*). W niniejszym artykule zaprezentowane zostaną dwa sposoby implementacji algorytmu Kruskala, różniące się podejściem do wykrywania cykli - metoda wykorzystująca strukturę zbiorów rozłącznych (ang. *Disjoint Set Union* lub *Union–Find data structure*) oraz rozwiązanie wykorzystujące własność spójnych składowych.

Słowa kluczowe: algorytm Kruskala, struktura zbiorów rozłącznych, spójne składowe, minimalne drzewo rozpinające, cykl, teoria grafów, algorytmy grafowe.

1. Wstęp

Wyznaczanie minimalnego drzewa rozpinającego (ang. *Minimum Spanning Tree*, MST) stanowi jedno z kluczowych zagadnień w optymalizacji sieci. Problemy tego typu pojawiają się m.in. przy projektowaniu sieci komputerowych, energetycznych czy telekomunikacyjnych, gdzie struktura sieci modelowana jest za pomocą nieskierowanego grafu ważonego. W takim modelu wierzchołki reprezentują węzły sieci, natomiast krawędzie odpowiadają połączeniom pomiędzy nimi, którym przypisane są wagi odzwierciedlające koszt realizacji danego połączenia, na przykład długość przewodu, koszt instalacji lub straty energii.

Problem wyznaczania minimalnego drzewa rozpinającego polega na znalezieniu podgrafu obejmującego wszystkie wierzchołki grafu wejściowego, który jest spójny i acykliczny, jednocześnie charakteryzuje się minimalną możliwą sumą wag krawędzi. Na wejściu algorytmu dany jest zatem spójny, nieskierowany graf ważony, natomiast na wyjściu otrzymujemy minimalne drzewo rozpinające tego grafu.

Autor korespondencyjny: B. Wieczorek (Bozena.Wieczorek@polsl.pl).
Data wpłynięcia: 10.12.2025.

Powszechnie stosowanym algorytmem do wyznaczania MST jest algorytm Kruskala. Jego działanie opiera się na zachłannym doborze krawędzi o najmniejszej wadze oraz stopniowym ich dołączaniu do budowanego drzewa, przy jednoczesnym unikaniu powstawania cykli [2].

W artykule zaprezentowane zostaną dwa sposoby implementacji algorytmu Kruskala, różniące się metodą wykrywania cykli: klasyczne podejście bazujące na analizie spójnych składowych oraz metoda wykorzystująca strukturę zbiorów rozłącznych.

W artykule stosowane będą następujące oznaczenia i pojęcia:

- n – liczba wierzchołków, m – liczba krawędzi,
- **graf nieskierowany** – rodzaj grafu, którego krawędzie łączące dwa wierzchołki nie mają określonego kierunku,
- **graf spójny** – graf, w którym między dowolną parą wierzchołków istnieje ścieżka,
- **graf ważony** – graf, w którym każdej krawędzi przypisano określoną wartość, zwaną wagą lub kosztem,
- **cykl** – ścieżka w grafie, która zaczyna się oraz kończy w tym samym wierzchołku i zawiera przynajmniej jedną krawędź,
- **minimalne drzewo rozpinające** – drzewo obejmujące wszystkie wierzchołki grafu oraz wybrane krawędzie, tak aby graf pozostał spójny oraz acykliczny, a suma wag wybranych krawędzi była najmniejsza spośród wszystkich możliwych drzew rozpinających tego grafu,
- **algorytm zachłanny** – algorytm dokonujący w każdym kroku lokalnie najlepszego wyboru, w celu uzyskania rozwiązania jak najbardziej zbliżonego do optymalnego.

2. Metoda spójnych składowych

2.1. Przedstawienie metody

Metoda spójnych składowych opiera się na identyfikacji i łączeniu wierzchołków grafu w grupy, w których każdy wierzchołek jest osiągalny z każdego innego. W algorytmie Kruskala metoda ta służy do wykrywania cykli podczas budowania drzewa rozpinającego. Jej celem jest połączenie wszystkich wierzchołków w jedną spójną składową, tworząc minimalne drzewo rozpinające. Dla spójnego grafu nieskierowanego zawierającego n wierzchołków, liczba krawędzi w drzewie rozpinającym wynosi $n - 1$.

2.2. Szczegółowy opis metody

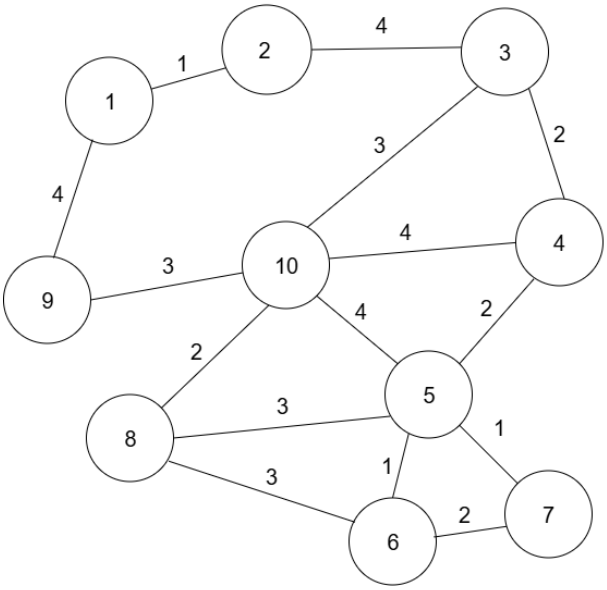
Kluczową koncepcją metody jest reprezentowanie grafu za pomocą numerów spójnych składowych. Wskazują one, do której grupy należy dany wierzchołek. Dzięki zastosowaniu tej metody możliwe jest szybkie sprawdzenie, czy dwa wierzchołki należą do tej samej składowej. Jest to bardzo istotne podczas budowy minimalnego drzewa rozpinającego. Ważnym założeniem tej metody jest również fakt, że każdy z wierzchołków tworzy własną spójną składową, a jej numer jednoznacznie identyfikuje przypisaną wierzchołkowi grupę. Podczas działania algorytmu przetwarzane są kolejne krawędzie grafu, posortowane niemalejąco według kosztu (wagi krawędzi). Dla każdej krawędzi $\{x, y\}$ następuje sprawdzenie, czy

wierzchołki x oraz y należą do różnych składowych - w takim przypadku następuje ich scalenie. Operacja ta sprawia, że wszystkie wierzchołki należące do tej samej składowej co wierzchołek y otrzymują numer składowej, do której należy wierzchołek x.

Dodanie krawędzi następuje tylko wtedy, gdy wierzchołki znajdują się w różnych składowych, co zapewnia, że w drzewie nie powstanie cykl. Proces powtarzany jest dla kolejnych krawędzi, aż wszystkie wierzchołki zostaną połączone w jedną spójną składową, co kończy budowę minimalnego drzewa rozpinającego.

2.3. Przedstawienie działania metody

Dany jest graf o liczbie wierzchołków $n = 10$ oraz liczbie krawędzi $m = 15$ (rysunek 1).



Rysunek 1. Przykładowy graf

Pierwszym krokiem działania algorytmu jest przypisanie każdemu wierzchołkowi numeru jego składowej. Na początku każdemu wierzchołkowi przypisuje się jego własny numer (tabela 1).

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	2	3	4	5	6	7	8	9	10

Tabela 1. Tablica początkowych składowych wierzchołków

Następnie należy posortować krawędzie w porządku niemalejącym według wag. Posortowane wagi krawędzi znajdują się w tabeli 2. Ostatnim etapem jest wyznaczenie minimalnego drzewa rozpinającego. Graf posiada dziesięć wierzchołków, a więc drzewo rozpinające powinno zawierać dziewięć krawędzi. Koszt MST jest sumą wag jego krawędzi.

Kolejne kroki wyznaczania drzewa przedstawione są poniżej.

- Przetwarzanie krawędzi 1 – 2 o wadze 1:

  Składowe wierzchołków 1 oraz 2 są różne - można dodać krawędź do MST. Po tej operacji zmienia się również składowa wierzchołka numer 2 – wynosi ona teraz 1.

Krawędź	*Waga*
{1, 2}	1
{5, 6}	1
{5, 7}	1
{3, 4}	2
{4, 5}	2
{6, 7}	2
{8, 10}	2
{3, 10}	3
{5, 8}	3
{6, 8}	3
{9, 10}	3
{1, 9}	4
{2, 3}	4
{4, 10}	4
{5, 10}	4

Tabela 2. Tablica krawędzi oraz wag

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	3	4	5	6	7	8	9	10

Koszt MST: 1

- Przetwarzanie krawędzi 5 – 6 o wadze 1:

  Składowe wierzchołków 5 oraz 6 są różne - można dodać krawędź do MST. Po tej operacji zmienia się również składowa wierzchołka numer 6 – wynosi ona teraz 5.

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	3	4	5	5	7	8	9	10

Koszt MST: 2

- Przetwarzanie krawędzi 5 – 7 o wadze 1:

  Składowe wierzchołków 5 oraz 7 są różne - można dodać krawędź do MST. Po tej operacji zmienia się również składowa wierzchołka numer 7 – wynosi ona teraz 5.

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	3	4	5	5	5	8	9	10

Koszt MST: 3

- Przetwarzanie krawędzi 3 – 4 o wadze 2:

  Składowe wierzchołków 3 oraz 4 są różne - można dodać krawędź do MST. Po tej operacji zmienia się również składowa wierzchołka numer 4 – wynosi ona teraz 3.

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	3	3	5	5	5	8	9	10

Koszt MST: 5

- Przetwarzanie krawędzi 4 – 5 o wadze 2:

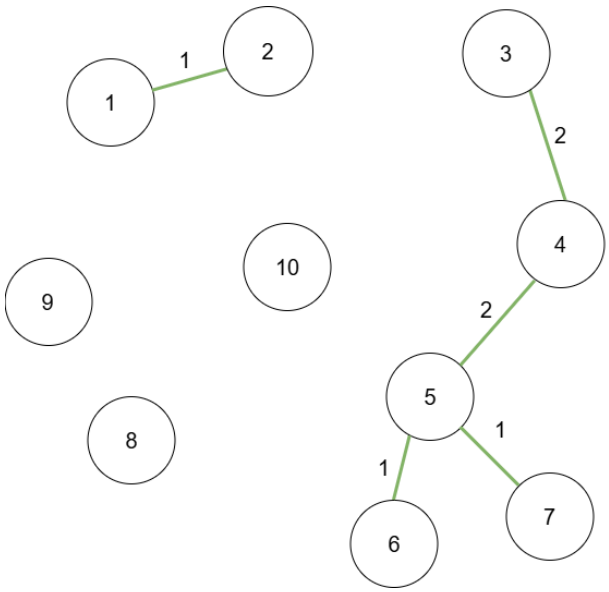
  Składowe wierzchołków 4 oraz 5 są różne - można dodać krawędź do MST. Po tej operacji zmieniają się również składowe wierzchołków numer 5, 6, 7 – wynoszą one teraz 3.

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	3	3	3	3	3	8	9	10

Koszt MST: 7

- Przetwarzanie krawędzi 6 – 7 o wadze 2:

  Składowe wierzchołków 6 oraz 7 są takie same - krawędź ta nie jest dodana do MST. Nie zmieniają się wartości składowych wierzchołków oraz suma MST. Stan grafu po tej operacji przedstawiony został na rysunku 2.

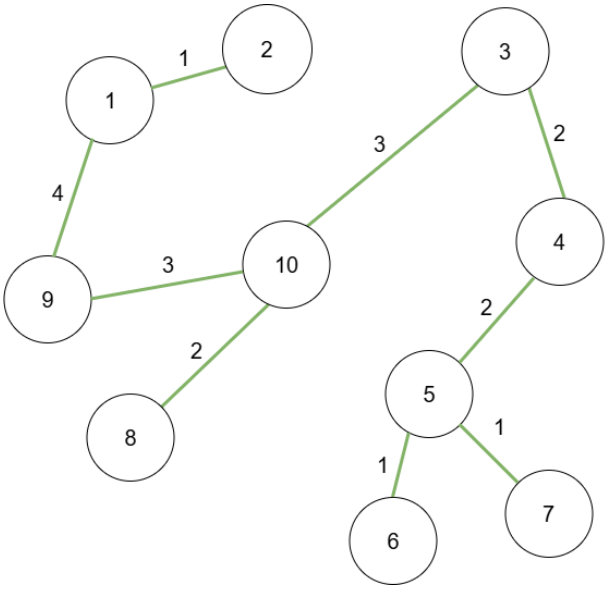


Rysunek 2. Częściowe drzewo rozpinające

- Kontynuujemy proces analogicznie aż do dołączenia $n-1$ krawędzi.

- Po zakończeniu działania algorytmu minimalne drzewo rozpinające zawiera krawędzie: {1, 2}, {5, 6}, {5, 7}, {3, 4}, {4, 5}, {8, 10}, {3, 10}, {9, 10}, {1, 9} a koszt całkowity wynosi 19. Końcowy graf został przedstawiony na rysunku 3.

wierzchołek	1	2	3	4	5	6	7	8	9	10
składowa	1	1	1	1	1	1	1	1	1	1



Rysunek 3. Minimalne drzewo rozpinające

2.4. Przykładowa implementacja

Przykładową implementację algorytmu Kruskala z metodą spójnych składowych przedstawiono na listingu 1.

```
#include <iostream>
#include <vector>
#include <algorithm>

struct krawedz {
  int x, y;
  long long waga;
};
void kruskal(int n, std::vector<std::pair<int, int>>& drzewo, std::vector<krawedz>&
    krawedzie)
{
  std::vector<int> skladowa(n + 1); // Numer sk adowych
  long long suma = 0;
  int stara_skladowa{}, nowa_skladowa{};

  for (int i = 1; i <= n; i++)
    skladowa[i] = i;

  sort(krawedzie.begin(), krawedzie.end(),[](auto& a, auto& b)
  {
    return a.waga < b.waga;
  });

  for (auto& k : krawedzie)
  {
    //Sprawdzanie czy istnieje cykl pomi dzy wierzcho kami
    if (skladowa[k.x] == skladowa[k.y])
      continue;
    //Dodanie kraw dzi do MST
    drzewo.push_back({ k.x, k.y });
```

```
30      suma += k.waga;
31      stara_skladowa = skladowa[k.y];
32      nowa_skladowa = skladowa[k.x];
33      //Przypisywanie nowych sk adowych
34      for (int i = 1; i <= n; i++)
35      {
36        if (skladowa[i] == stara_skladowa)
37          skladowa[i] = nowa_skladowa;
38      }
39      if (drzewo.size() == n - 1) break;
40    }
41    std::cout << "Koszt MST: " << suma << std::endl;
42  }
43  int main() {
44    int n, m, w1, w2;
45    long long koszt;
46    //Lista kraw dzi nale  cych do MST
47    std::vector<std::pair<int, int>> drzewo;
48    //Lista wszystkich kraw dzi
49    std::vector<krawedz> krawedzie;
50    std::cout << "Podaj liczbe wierzcholkow: " << std::endl;
51    std::cin >> n;
52    std::cout << "Podaj liczbe krawedzi: " << std::endl;
53    std::cin >> m;
54    for (int i = 1; i <= m; i++) {
55      std::cout << "Podaj krawedz - dwa wierzcholki oraz koszt (wage): " << std::endl;
56      std::cin >> w1 >> w2 >> koszt;
57      krawedzie.push_back({ w1, w2, koszt });
58    }
59    kruskal(n, drzewo, krawedzie);
60  }
```

Listing 1. Metoda spójnych składowych

2.5. Złożoność obliczeniowa metody

Algorytm Kruskala, w którym zastosowano metodę spójnych składowych, charakteryzuje się złożonością czasową równą $O(nm)$, gdzie n oznacza liczbę wierzchołków w grafie, a m — liczbę krawędzi łączących te wierzchołki. W fazie przygotowania, która obejmuje inicjalizację tablicy składowych złożoność obliczeniowa wynosi $O(n+m)$. Kolejną fazą działania algorytmu jest sortowanie krawędzi niemalejąco według ich wag. Ten etap posiada złożoność obliczeniową $O(m \log m)$. Ostatnim etapem jest budowanie minimalnego drzewa rozpinającego (MST). Podczas jego trwania każda krawędź jest analizowana, a w przypadku połączenia dwóch różnych składowych konieczne jest przejście po wszystkich wierzchołkach grafu w celu aktualizacji numerów składowych, co prowadzi do złożoności $O(n)$ dla każdej krawędzi. W rezultacie, całkowita złożoność etapu scalania składowych wynosi $O(nm)$.

3. Metoda z użyciem struktury zbiorów rozłącznych

3.1. Szczegółowy opis metody

Struktura zbiorów rozłącznych służy do zarządzania zbiorami. Główną ideą jest szybkie wykrywanie cykli bez przeszukiwania grafu. Struktura zbiorów rozłącznych przechowuje informację o tym, do jakiego

komponentu należy każdy wierzchołek [1]. Na początku każdy węzeł stanowi odrębny zbiór. Z każdym etapem dodawania krawędzi zbiory te są łączone, a operacje przeprowadzane na strukturze pozwalają zweryfikować, czy dwa wierzchołki znajdują się już w tym samym zbiorze. Jeżeli tak, oznacza to, że dodanie krawędzi spowodowałoby powstanie cyklu, więc należy ją pominąć. Dzięki temu algorytm Kruskala tworzy minimalne drzewo rozpinające bez konieczności dodatkowego przeszukiwania grafu. Struktura zbiorów rozłącznych składa się z dwóch podstawowych operacji: znajdowania (ang. *find*) oraz połączenia (ang. *union*). Pierwsza z nich odpowiada za znalezienie zbioru, do którego należy dany element, sprawdzając czy dwa elementy są w tym samym zbiorze, a druga polega na łączeniu, które scala dwa zbiory w jeden. W celu identyfikacji i odróżniania od siebie zbiorów używa się jednego wyróżnionego elementu zbioru, zwanego reprezentantem.
Idea działania algorytmu:

1. **Inicjalizacja struktury zbiorów rozłącznych.** Na początku każdy wierzchołek tworzy odrębny, jednoelementowy zbiór.

2. **Sortowanie krawędzi.** Wszystkie krawędzie grafu sortowane są niemalejące według wag (kosztów).

3. **Przetwarzanie ciągu posortowanych krawędzi:**

   - dla wierzchołka x wykonywana jest operacja znajdowania, aby ustalić jego reprezentanta,
   - dla wierzchołka y wykonywana jest operacja znajdowania, aby ustalić jego reprezentanta,
   - jeżeli reprezentanci zbiorów są różni, krawędź $\{x, y\}$ zostaje dodana do wynikowego drzewa rozpinającego, a oba zbiory są scalane operacją połączenia. W przeciwnym przypadku krawędź jest pomijana, ponieważ jej dodanie spowodowałoby powstanie cyklu.

4. **Kontynuacja procesu.** Proces trwa aż w drzewie rozpinającym znajdzie się $n - 1$ krawędzi.

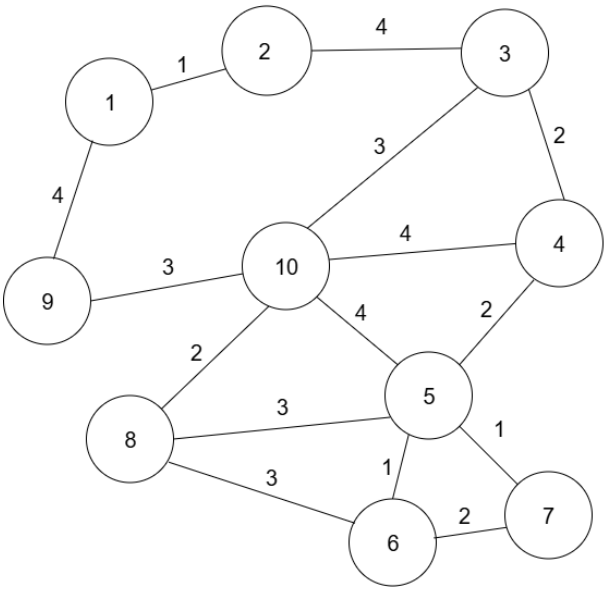
3.2. Metody znajdowania oraz łączenia

Metoda *znajdź(x)* służy do wyznaczenia reprezentanta zbioru, do którego należy element x. Każdy zbiór przedstawiany jest w postaci drzewa, w którym korzeń pełni rolę identyfikatora całej składowej. Działanie funkcji polega na rekurencyjnym przemierzaniu od wierzchołka x w górę drzewa aż do osiągnięcia jego korzenia. W celu minimalizacji złożoności kolejnych wywołań stosuje się technikę kompresji ścieżki (ang. *path compression*), polegającą na tym, że wszystkie wierzchołki napotkane w trakcie poszukiwania zostają bezpośrednio połączone z korzeniem.

Metoda *połącz(x, y)* pozwala na połączenie dwóch rozłącznych zbiorów, w których znajdują się elementy x oraz y. W pierwszym kroku wyznaczani są reprezentanci obu zbiorów: x = *znajdź(x)* oraz y = *znajdź(y)*. W przypadku gdy x jest różny od y, zbiory te są łączone poprzez przypisanie jednego z reprezentantów jako rodzica drugiego. W celu ograniczenia wzrostu wysokości drzew reprezentujących zbiory używa się techniki łączenia według rozmiaru (ang. *union by size*) lub łączenia według rangi (ang. *union by rank*). Zarówno w jednym, jak i drugim przypadku drzewo o mniejszej liczbie wierzchołków dołączane jest do większego z drzew.

3.3. Przedstawienie działania algorytmu

Rozważmy graf nieskierowany, ważony o liczbie wierzchołków $n = 10$ oraz liczbie krawędzi $m = 15$ (rysunek 4).



Rysunek 4. Przykładowy graf

- Na początku każdy wierzchołek tworzy osobny, jednoelementowy zbiór, tzn. jest swoim rodzicem i reprezentantem.

  $rodzic[x] = x$, $rozmiar[x] = 1$, gdzie x oznacza wierzchołek w grafie.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	2	3	4	5	6	7	8	9	10
$rozmiar[x]$	1	1	1	1	1	1	1	1	1	1

- Sortujemy krawędzie niemalejąco według wag (tabela 3).

Krawędź	$Waga$
$\{1,2\}$	1
$\{5,6\}$	1
$\{5,7\}$	1
$\{3,4\}$	2
$\{4,5\}$	2
$\{6,7\}$	2
$\{8,10\}$	2
$\{3,10\}$	3
$\{5,8\}$	3
$\{6,8\}$	3
$\{9,10\}$	3
$\{1,9\}$	4
$\{2,3\}$	4
$\{4,10\}$	4
$\{5,10\}$	4

Tabela 3. Tablica krawędzi oraz wag

Rozważamy po kolei krawędzie, wykonując dla każdej procedury: *znajdź(x)*, *znajdź(y)* oraz opcjonalnie *połącz(x, y)*.

- Rozpoczynamy pierwszą iterację. Rozważmy krawędź {1, 2} o wadze 1. *znajdź*(1) = 1, *znajdź*(2) = 2 → różni reprezentanci. Dodajemy krawędź do MST i wykonujemy *połącz*(1, 2). Po operacji uzyskujemy $rodzic[2] = 1$, $rozmiar[1] = 2$. Drzewo zawiera krawędź $\{1, 2\}$, a koszt częściowy wynosi 1.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	1	3	4	5	6	7	8	9	10
$rozmiar[x]$	2	1	1	1	1	1	1	1	1	1

- Kontynuujemy. Rozważamy krawędź {5, 6} o wadze 1. *znajdź*(5) = 5, *znajdź*(6) = 6 → różni reprezentanci. Dodajemy krawędź do MST i wykonujemy *połącz*(5, 6). Po operacji uzyskujemy $rodzic[6] = 5$, $rozmiar[5] = 2$. MST zawiera krawędzie {1, 2}, {5, 6}, a koszt częściowy wynosi 2.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	1	3	4	5	5	7	8	9	10
$rozmiar[x]$	2	1	1	1	2	1	1	1	1	1

- Rozważamy krawędź {5, 7} o wadze 1. *znajdź*(5) = 5, *znajdź*(7) = 7 → różni reprezentanci. Dodajemy krawędź do MST i wykonujemy *połącz*(5, 7). Po operacji uzyskujemy $rodzic[7] = 5$, $rozmiar[5] = 3$. Drzewo zawiera krawędzie {1, 2}, {5, 6}, {5, 7} a koszt częściowy wynosi 3.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	1	3	4	5	5	5	8	9	10
$rozmiar[x]$	2	1	1	1	3	1	1	1	1	1

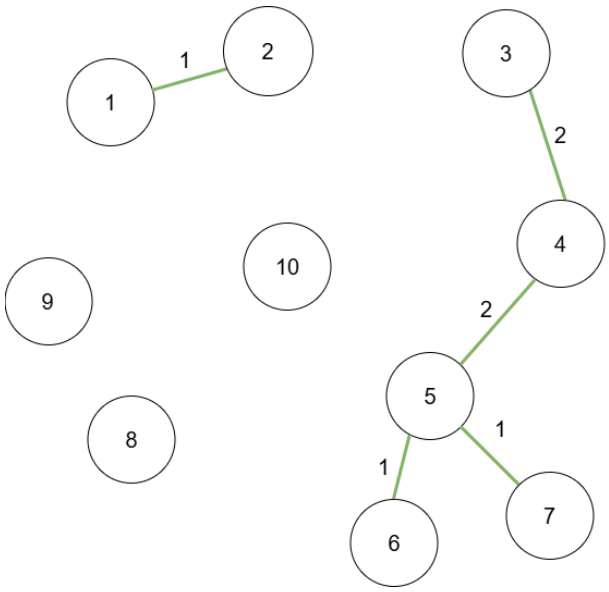
- Następnie rozważamy krawędź {3, 4} o wadze 2. *znajdź*(3) = 3, *znajdź*(4) = 4 → różni reprezentanci. Dodajemy krawędź do MST i wykonujemy *połącz*(3, 4). Po operacji uzyskujemy $rodzic[4] = 3$, $rozmiar[3] = 2$. MST zawiera krawędzie {1, 2}, {5, 6}, {5, 7}, {3, 4} a koszt częściowy wynosi 5.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	1	3	3	5	5	5	8	9	10
$rozmiar[x]$	2	1	2	1	3	1	1	1	1	1

- Rozważamy krawędź {4, 5} o wadze 2. *znajdź*(4) = 3, *znajdź*(5) = 5 → różni reprezentanci. Dodajemy krawędź do MST i wykonujemy *połącz*(4, 5). Po operacji uzyskujemy $rodzic[3] = 5$, $rozmiar[3] = 2$, $rozmiar[5] = 5$. MST zawiera krawędzie {1, 2}, {5, 6}, {5, 7}, {3, 4}, {4, 5} a koszt częściowy wynosi 7. Krawędzie dodane do drzewa przedstawiono na rysunku 5.

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	1	1	5	3	5	5	5	8	9	10
$rozmiar[x]$	2	1	2	1	5	1	1	1	1	1

- Krawędź $\{6, 7\}$ o wadze 2. *znajdź*(6) = 5, *znajdź*(7) = 5 → ten sam reprezentant. Krawędź tworzy cykl, więc pomijamy. MST pozostaje bez zmian.

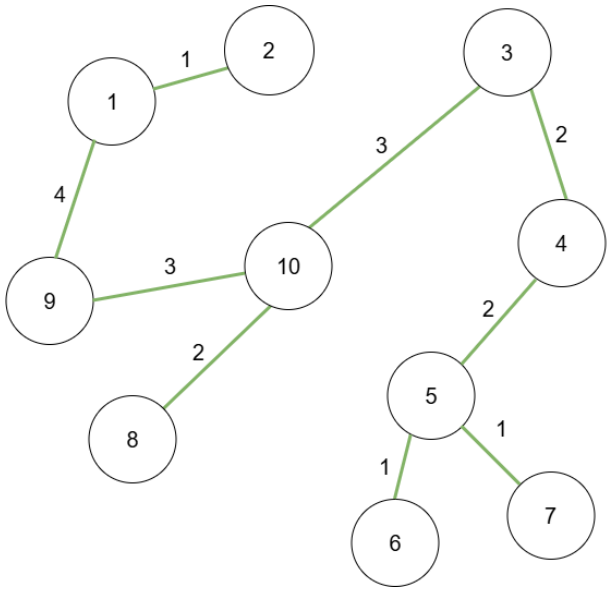


Rysunek 5. Częściowe drzewo rozpinające

- Kontynuujemy analogicznie aż do drzewa dołączonych zostanie $n-1$ krawędzi.

Minimalne drzewo rozpinające zawiera następujące krawędzie: {1, 2}, {5, 6}, {5, 7}, {3, 4}, {4, 5}, {8, 10}, {3, 10}, {9, 10}, {1, 9} a koszt całkowity wynosi 19 (rysunek 6).

x	1	2	3	4	5	6	7	8	9	10
$rodzic[x]$	5	1	5	3	5	5	5	5	5	5
$rozmiar[x]$	2	1	2	1	10	1	1	2	1	1



Rysunek 6. Minimalne drzewo rozpinające

3.4. Przykładowa implementacja

Przykładową implementację algorytmu Kruskala z wykorzystaniem struktury zbiorów rozłącznych przedstawiono w listingu 2. Rozwiązanie zostało podzielone na trzy główne funkcje: *znajdz*, *polacz*, *kruskal*. Główna funkcja *kruskal* realizuje właściwy algorytm znajdowania minimalnego drzewa rozpinającego.

```
#include<iostream>
#include<vector>
#include<algorithm>

//Struktura reprezentuj ca kraw d  w grafie
struct krawedz
{
  int x, y;
  long long koszt;
};
//Funkcja znajduj ca reprezentanta zbioru
int znajdz(int x, std::vector<int>& rodzic)
{
  if (x != rodzic[x])
    rodzic[x] = znajdz(rodzic[x], rodzic);
  return rodzic[x];
}
//Funkcja    czca    dwa zbiory
void polacz(int x, int y, std::vector<int>& rodzic, std::vector<long long>& rozmiar)
{
  x = znajdz(x, rodzic);
  y = znajdz(y, rodzic);
  if (x == y)
    return;
  //zawsze przy  czamy mniejszy zbi r do wi kszego
  if (rozmiar[x] < rozmiar[y])
    std::swap(x, y);
  rodzic[y] = x;
  rozmiar[x] += rozmiar[y];
}
//G  wna funkcja algorytmu Kruskala
void kruskal(int n, int m, std::vector<std::pair<int, int>>& drzewo, std::vector<
  krawedz>& krawedzie)
{
  //Inicjalizacja struktur
  std::vector<int> rodzic(n + 1);
  //ka dy zbi r ma pocz tkowo rozmiar r wny 1
  std::vector<long long> rozmiar(n + 1, 1);
  for (int i = 1; i <= n; i++)
  {
    rodzic[i] = i;
  }
  //Sortowanie kraw dzi niemalej co wed ug wag
  sort(krawedzie.begin(), krawedzie.end(), [](const krawedz& a, const krawedz& b)
  {
    return a.koszt < b.koszt;
  });
  long long suma = 0;
  int i = 0;
```

```
49    while (drzewo.size() < n - 1)
50    {
51      //je li kraw d     czy    r   ne zbiory, to dodajemy j   do MST
52      if (znajdz(krawedzie[i].x, rodzic) != znajdz(krawedzie[i].y, rodzic))
53      {
54        drzewo.push_back({ krawedzie[i].x, krawedzie[i].y });
55        polacz(krawedzie[i].x, krawedzie[i].y, rodzic, rozmiar);
56        suma += krawedzie[i].koszt;
57      }
58      i++;
59    }
60    std::cout << "Minimalne drzewo rozpinajace = " << suma;
61  }
62  int main()
63  {
64    int n, m, w1, w2;
65    long long koszt;
66    //Lista kraw dzi nale  cych do MST
67    std::vector<std::pair<int, int>> drzewo;
68    //Lista wszystkich kraw dzi
69    std::vector<krawedz> krawedzie;
70    std::cout << "Podaj liczbe wierzcholkow: " << std::endl;
71    std::cin >> n;
72    std::cout << "Podaj liczbe krawedzi: " << std::endl;
73    std::cin >> m;
74    for (int i = 1; i <= m; i++)
75    {
76      std::cout << "Podaj krawedz - dwa wiercholki oraz koszt (wage): " << std::endl;
77      std::cin >> w1 >> w2 >> koszt;
78      krawedzie.push_back({ w1, w2, koszt });
79    }
80    kruskal(n, m, drzewo, krawedzie);
81  }
```

Listing 2. Metoda z zastosowaniem struktury zbiorów rozłącznych

3.5. Złożoność obliczeniowa

Złożoność obliczeniowa algorytmu Kruskala z wykorzystaniem struktury zbiorów rozłącznych zależy od dwóch głównych etapów jego działania: sortowania krawędzi oraz procesu łączenia zbiorów w strukturze. W pierwszym etapie wszystkie krawędzie grafu są sortowane niemalejąco według wag, dla grafu o m krawędziach sortowanie wymaga czasu $O(m \log m)$. Drugi etap polega na iteracyjnym przetwarzaniu krawędzi i sprawdzaniu, czy oba wierzchołki należą do różnych zbiorów. Operacje te są realizowane za pomocą funkcji znajdowania, połączenia, które umożliwiają szybkie scalanie wierzchołków. Dzięki zastosowaniu optymalizacji, takich jak kompresja ścieżek oraz łączenie według rozmiaru lub rangi, każda z tych operacji jest wykonywana w czasie zbliżonym do stałego – $O(\alpha(n))$, gdzie $\alpha(n)$ to bardzo wolno rosnąca odwrotność funkcji Ackermana, uznawana praktycznie za stałą. Złożoność tego etapu jest więc $O(m\alpha(n))$. Łączna złożoność obliczeniowa algorytmu Kruskala wynosi $O(m \log m)$.

4. Podsumowanie

W niniejszym artykule przedstawiono dwa podejścia do realizacji algorytmu Kruskala: z wykorzystaniem struktury zbiorów rozłącznych oraz metodę opartą na spójnych składowych. Omówiono zasady ich działania, różnice implementacyjne oraz wpływ na złożoność obliczeniową. Analiza wykazała, że zastosowanie struktury zbiorów rozłącznych zapewnia wyższą wydajność, zwłaszcza w przypadku dużych grafów, natomiast podejście oparte na spójnych składowych, choć bardziej intuicyjne, okazuje się mniej efektywne. Poza omówionymi w artykule metodami wyznaczania minimalnego drzewa rozpinającego istnieją również inne algorytmy, takie jak Prima-Dijkstry oraz Borůvki.

Literatura

1. Robert Endre Tarjan. *Efficiency of a Good But Not Linear Set Union Algorithm.*, [w:] J. ACM Association for Computing Machinery, April 1975, New York, USA, 215–225. `https://doi.org/10.1145/321879.321884`
2. Haiming Li, Qiyang Xia, Yong Wang. *Research and Improvement of Kruskal Algorithm*, [w:] Journal of Computer and Communications, January 2017, 63-69. `https://www.researchgate.net/publication/320714849_Research_and_Improvement_of_Kruskal_Algorithm`