Algorytmy 2
Laboratorium: algorytm Kruskala i struktura Union-Find

Przemysław Klęsk

1 grudnia 2019

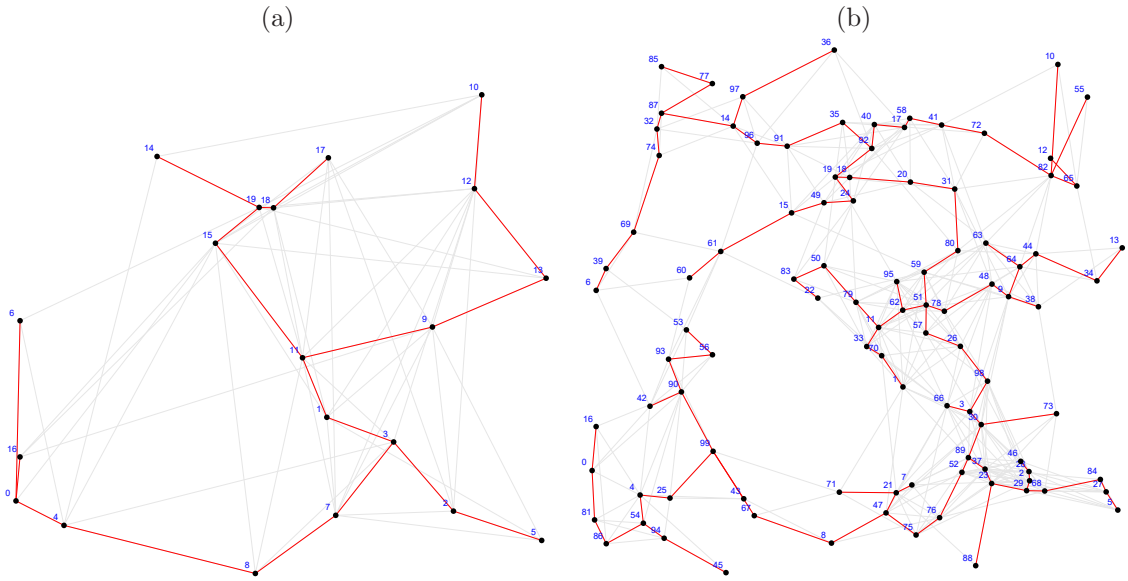
1 Cel

Celem zadania jest wykonanie implementacji *algorytmu Kruskala*, który znajduje *minimalne drzewo rozpinające* (MST — ang. *minimum spanning tree*) dla danego grafu nieskierowanego z wagami. Wagi są przypisane do krawędzi grafu i reprezentują koszty przejścia. Minimalne drzewo rozpinające to taki podzbiór krawędzi grafu, którego suma wag jest najmniejsza i jednocześnie w ramach tego podzbioru istnieje droga pomiędzy każdą parą węzłów grafu. Przykłady pokazano na Rys. 1.

Algorytm Kruskala w trakcie pracy buduje wiele cząstkowych drzew rozpinających i stopniowo łączy je w jedno wynikowe drzewo. Taki sposób działania powoduje, że algorytm ten idealnie współpracuje ze strukturą danych o nazwie *Union-Find*, zwanej także strukturą zbiorów rozłącznych. Struktura ta „śledzi losy" pewnych obiektów, pogrupowanych w rozłączne zbiory, i pozwala na wykonywanie dwóch typów operacji: łączenia (union) — podane dwa zbiory zostają połączone w jeden zbiór, oraz znajdowania (find) — dla podanego numeru obiektu wskazany zostaje zbiór, do którego ten obiekt należy (odbywa się to poprzez wskazanie reprezenta lub tzw. elementu kanonicznego zbioru).

Na wysokim poziomie algorytm Kruskala można wypowiedzieć następująco:

1. Zainicjalizuj strukturę Union-Find (na początku każdy węzeł stanowi odrębny jednoelementowy zbiór).
2. Posortuj krawędzie grafu według wag rosnąco.
3. Dla każdej krawędzi (i, j) w posortowanej kolekcji:
   - 3.1 Znajdź zbiór, do którego należy węzeł i (operacja find).
   - 3.2 Znajdź zbiór, do którego należy węzeł j (operacja find).
   - 3.3 Jeżeli znalezione zbiory dla i oraz j są różne to: dodaj krawędź (i, j) do wynikowego drzewa rozpinającego oraz połącz te zbiory (operacja union).
4. Zwróć wynikowe drzewo rozpinające.



Rysunek 1: Przykładowe minimalne drzewa rozpinające dla grafów, w których liczby węzłów i krawędzi wynoszą: (a) $n = 20$, $e = 65$, (b) $n = 100$, $e = 378$. Wagi odpowiadają odległościom pomiędzy węzłami w linii prostej.

Każdy ze zbiorów rozłącznych jest w ramach struktury Union-Find przechowywany w postaci drzewa, przy czym wykorzystuje się prostą implementację indeksową. Tzn. cała kolekcja takich drzew jest reprezentowana przez jedną tablica z indeksami rodziców (do celów pomocniczych można prowadzić także drugą tablicę tzw. rang[1]).

Jeżeli n i e oznaczają odpowiednio liczbę węzłów i krawędzi w grafie, to złożoność algorytmu Kruskala jest rzędu $O(e \log e)$, co gdy graf jest gęsty, tzn. dla e bliskiego $n(n-1)/2$, jest równoważne złożoności $O(n^2 \log n)$.

2 Instrukcje, wskazówki, podpowiedzi

1. Na potrzeby tego ćwiczenia w implementacji należy przewidzieć pewne pomocnicze typy (struktury lub klasy) reprezentujące: **węzeł w grafie** (ze współrzędnymi x, y), **krawędź w grafie** (para indeksów wraz z kosztem), oraz sam **graf**. Obiekty te będą w ramach eksperymentów wczytywane z plików tekstowych (patrz zawartość pliku `pklesk_kruskal_union_find.zip`, który towarzyszy tej instrukcji).

2. Implementacja może wykorzystywać wcześniej wykonane własne kontenery np. listy lub tablicy dynamicznej (np. poprzez dołączenie odpowiednich plików nagłówkowych `.h`) w celu: przechowania węzłów lub krawędzi grafu, reprezentacji wynikowego MST, itp.

[1] ranga to górne ograniczenie na wysokość drzewa

3. **Struktura Union-Find** może zostać zaimplementowana w formie klasy (lub struktury) zawierającej dwie tablice liczb całkowitych. Jedna z nich będzie przechowywała wskazania na indeksy obiektów-rodziców w ramach zbiorów rozłącznych — i tym samym będzie stanowiła reprezentację kolekcji drzew. Druga tablica będzie przechowywała rangi (czyli górne ograniczenia na wysokości drzew). Interfejs tej stuktury powinien zawierać następujące funkcje / metody:

   (a) konstruktor (argument: n — liczba węzłów w grafie),

   (b) łączenie dwóch zbiorów (argumenty: dwa indeksy węzłów, których zbiory chcemy połączyć),

   (c) znajdowanie reprezentanta zbioru (argument: indeks węzła; wynik: indeks reprezentanta zbioru, do którego należy podany węzeł).

   Uwaga: zgodnie z informacjami z wykładu, należy przygotować dwa warianty metody łączenia (zwykłe łączenie oraz „union-by-rank"), a także dwa warianty metody znajdowania (bez i z kompresją ścieżki — „path compression").

4. **Algorytm Kruskala** może być zrealizowany jako funkcja otrzymująca jako argument graf (lub listę wszystkich krawędzi) i zwracająca jako wynik minimalne drzewo rozpinające (wynik może być także listą z podzbiorem krawędzi). Wewnątrz funkcji należy wykorzystać dowolny własny algorytm sortujący (nie gorszy niż klasy $e \log e$).

3 Eksperymenty

Wewnątrz spakowanego pliku `pklesk_kruskal_union_find.zip` znajdują się trzy pliki tekstowe `g1.txt`, `g2.txt`, `g3.txt` reprezentujące grafy. Format: n — liczba węzłów (wiersz pierwszy), współrzędne (x, y) węzłów oddzielone spacją (kolejne n wierszy), e — liczba krawędzi (wiersz nr $n + 2$), krawędzie podane w formie pary indeksów i wagi oddzielonych spacjami (kolejne e wierszy). Należy przygotować funkcję, która potrafi odczytywać pliki tekstowe w tym formacie i zamieniać je na grafy. Uwaga: informacje o współrzędnych (x, y) węzłów nie będą bezpośrednio potrzebne, ponieważ koszty wag (równe odległościom pomiędzy węzłami) zostały już obliczone. Współrzędne można ewentualnie wykorzystać do weryfikacji wizualnej i wyrysowania grafu (poza oceną).

Dla wspomnianych trzech grafów należy **wykonać algorytm Kruskala raportując** na ekran:

- liczbę krawędzi i sumę wag w wynikowym drzewie rozpinającym,
- czas obliczeń kroku sortującego krawędzie,
- czas obliczeń głównej pętli w algorytmie Kruskala,
- liczbę wykonań operacji find (należy wprowadzić odpowiedni licznik).

Eksperymenty należy powtórzyć włączając lub wyłączając warianty „union-by-rank" oraz „path compression".

4 Sprawdzenie antyplagiatowe — przygotowanie wiadomości e-mail do wysłania

1. Kod źródłowy programu po sprawdzeniu przez prowadzącego zajęcia laboratoryjne musi zostać przesłany na adres *algo2@zut.edu.pl*.

2. Plik z kodem źródłowym musi mieć nazwę wg schematu: `nr_albumu.algo2.nr_lab.main.c` (plik może mieć rozszerzenie `.c` lub `.cpp`). Przykład: `123456.algo2.lab06.main.c` (szóste zadanie laboratoryjne studenta o numerze albumu 123456). Jeżeli kod źródłowy programu składa się z wielu plików, to należy stworzyć jeden plik, umieszczając w nim kody wszystkich plików składowych.

3. Plik musi zostać wysłany z poczty ZUT (zut.edu.pl).

4. Temat maila musi mieć postać: `ALGO2 IS1 XXXY LAB06`, gdzie `XXXY` to numer grupy (np. `ALGO2 IS1 210C LAB06`).

5. W pierwszych trzech liniach pliku z kodem źródłowym w komentarzach muszą znaleźć się:

   - informacja identyczna z zamieszczoną w temacie maila (linia 1),
   - imię i nazwisko autora (linia 2),
   - adres e-mail (linia 3).

6. Mail nie może zawierać żadnej treści (tylko załącznik).

7. W razie wykrycia plagiatu, wszytkie uwikłane osoby otrzymają za dane zadanie ocenę 0 punktów (co jest gorsze niż ocena 2 w skali $\{2, 3, 3.5, 4, 4.5, 5\}$).