Dokumentacja projektu zaliczeniowego

Autor: Szymon Fortuna, Informatyka stosowana, III rok
Przedmiot: Język Python, gr. 2 (środa, 10:00)
Rok akademicki: 2023/2024

Temat projektu

Tematem mojego projektu zaliczeniowego jest implementacja grafu ważonego metodą listy sąsiedztwa oraz algorytmu grafowego Floyda-Warshalla, znajdującego najkrótsze ścieżki pomiędzy wierzchołkami grafu.

Uruchomienie

Program (a konkretnie testy) można uruchomić:

a) W systemie Windows poleceniem: python test_graph.py
b) W systemie Linux poleceniem: python3 test_graph.py

Nie jest wymagane instalowanie żadnych dodatkowych modułów.

Graf

Graf to abstrakcyjny typ danych reprezentujący pewien zbiór V (wierzchołki) oraz zestaw powiązań pomiędzy jego elementami oznaczany jako E (krawędzie). Graf może być skierowany (zorientowany) lub nieskierowany. Graf nieskierowany różni się od skierowanego tym, że powiązanie między wierzchołkiem A i B oznacza takie samo powiązanie między B i A. Standardowo zbiór E definiujemy jako podzbiór iloczynu kartezjańskiego zbioru V (graf zorientowany) albo zbiór dwuelementowych podzbiorów zbioru V (graf niezorientowany), jednak każda krawędź może nieść ze sobą dodatkowe informacje, np. jej wagę (będącą liczbą).

Ścieżką pomiędzy wierzchołkami A i B nazywamy dowolny ciąg krawędzi (A, v_1), (v_1, v_2), (v_2, v_3), ….., (v_{n-2}, v_{n-1}), (v_{n-1}, v_n), (v_n, B). Wagą ścieżki jest suma wag każdej krawędzi na tej ścieżce.

Cykl to ścieżka, której oba końce są tym samym wierzchołkiem.

Algorytm Floyda-Warshalla

Naturalną potrzebą w grafie jest znalezienie najkrótszej ścieżki pomiędzy dwoma wierzchołkami. Istnieją różne algorytmy, które rozwiązują ten problem w różnych warunkach.

Algorytm Floyda-Warshalla znajduje najkrótszą ścieżkę pomiędzy każdą parą wierzchołków w grafie skierowanym, jak i nieskierowanym. Dopuszcza ujemne wagi krawędzi, jednak nie działa dla grafów z cyklami, w których suma wag krawędzi jest ujemna. Złożoność czasowa wynosi $O(|V|^3)$, a pamięciowa $O(|V|^2)$. Jest to algorytm dynamiczny.

Pseudokod algorytmu:

```
dla każdego wierzchołka i:
  dla każdego wierzchołka j:
    dystans[i][j] = +∞
    poprzednik[i][j] = niezdefiniowane
  dystans[i][i] = 0
  poprzednik[i][i] = i

dla każdej krawędzi e:
  dystans[e.start][e.koniec] = e.waga
  poprzednik[e.start][e.koniec] = e.start

dla każdego wierzchołka u:
  dla każdego wierzchołka v1:
    dla każdego wierzchołka v2:
      jeśli dystans[v1][v2] > dystans[v1][u] +
dystans[u][v2]:
        dystans[v1][v2] = dystans[v1][u] + dystans[u][v2]
        jeśli dystans[v2][v2] < 0:
            rzuć wyjątek - wykryto cykl ujemny, algorytm
```

```
nie działa dla tego grafu
        poprzednik[v1][v2] = poprzednik[u][v2]
```

Aby odczytać wagę najkrótszej ścieżki pomiędzy dwoma węzłami, wystarczy odnieść się do macierzy `dystans`. Natomiast odtworzenie ścieżki odbywa się za pomocą algorytmu:

```
jeśli poprzednik[v1][v2] == niezdefiniowane:
  rzuć wyjątek
ścieżka = []
dopóki v1 != v2:
  u = poprzednik[v1][v2]
  ścieżka.wstaw(początek, krawędź z u do v2)
  v2 = u
```

Implementacja grafu

Ten abstrakcyjny typ danych, jakim jest graf, zaimplementowałem w pliku *GraphImplementation.py*, używając obiektowego paradygmatu programowania. Jest on reprezentowany przez klasę `Graph`, która zawiera odpowiednie atrybuty oraz metody odpowiadające za podstawowe działania na grafie oraz algorytm Floyda-Warshalla.

Konstruktor tej klasy przyjmuje jeden parametr `directed` typu logicznego. Dla wartości `True` tworzony jest graf zorientowany, a w przeciwnym przypadku niezorientowany. Konstruktor inicjuje następujące atrybuty:

- `_is_directed` – zmienna logiczna, oznaczająca, czy graf jest skierowany, czy nie
- `_structure` – słownik przechowujący listę sąsiedztwa. Jest zaimplementowany według materiałów na stronie kursu (https://ufkapano.github.io/algorytmy/lekcja14/python2.html). Kluczami są etykiety wierzchołków, a elementami listy dwuelementowych krotek oznaczających krawędzie wychodzące z danego wierzchołka, gdzie pierwszy element jest etykietą wierzchołka końcowego danej krawędzi, a drugi element jej wagą
- `_is_floyded` – zmienna logiczna, która przechowuje informację, czy graf jest gotowy na pobranie danych opierających się o algorytm Floyda-

Warshalla (innymi słowy – czy dane w poniższych dwóch atrybutach są zgodne z aktualną strukturą grafu)

- `_floyd_distances` – słownik, który przechowuje długości najkrótszych ścieżek pomiędzy dwoma wierzchołkami; kluczami są krotki (wierzchołek startowy, wierzchołek końcowy), a wartościami liczby oznaczające długość ścieżki pomiędzy tymi wierzchołkami
- `_floyd_predecessors` – słownik przechowujący węzły pośrednie pomiędzy dwoma wierzchołkami. Klucze syntaktycznie i semantycznie są skonstruowane jak w słowniku powyżej, a wartościami są etykiety odpowiednich węzłów

Metody `add_node` i `add_edge` zostały zaimplementowane według materiałów na stronie kursu. Dodatkowo jednak ustawiają one flagę `_is_floyded` na `False`, ponieważ po dodaniu nowego węzła lub krawędzi macierze dystansów i poprzedników są nieaktualne.

Metody `list_nodes`, `list_edges` i `print_graph` są zaimplementowane według materiałów na stronie kursu. Pierwsze dwie zwracają listę wierzchołków / krawędzi, druga wypisuje na ekran cały graf.

Metoda `get_weight` zwraca wagę krawędzi między dwoma wierzchołkami podanymi jako argumenty. Jeśli nie ma pomiędzy nimi bezpośredniej krawędzi, zwraca `None`.

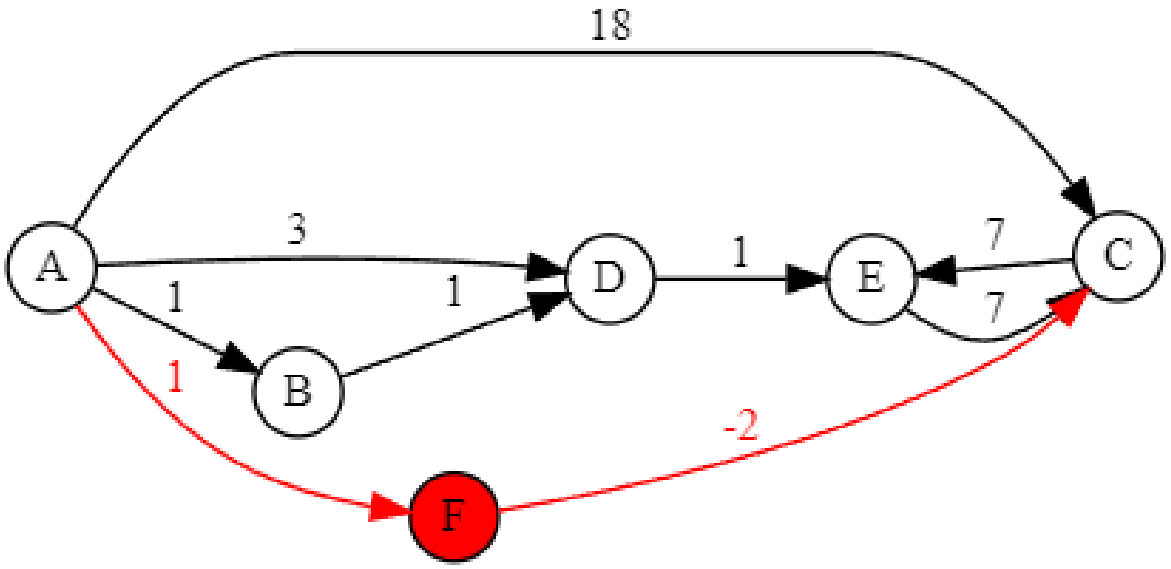
Metoda `_floyd` implementuje pierwszy pseudokod przedstawiony w poprzednim rozdziale. Na końcu ustawia flagę `_is_floyded` na `True`, bo macierze z algorytmu są już gotowe na odczyt z nich danych o ścieżkach.

Metoda `floyd_distance` zwraca długość najkrótszej ścieżki pomiędzy wierzchołkiem v1 a v2 przekazanymi jako argumenty. Najpierw jednak sprawdza, czy macierze są aktualne (czyli czy nie zostały dodane nowe węzły lub krawędzie, nieuwzględnione w macierzach) – jeśli nie, przed zwróceniem wywoływana jest metoda `_floyd`.

Metoda `floyd_path` zwraca informację o najkrótszej ścieżce pomiędzy wierzchołkiem v1 a v2 przekazanymi jako argumenty, zgodnie z drugim pseudokodem przedstawionym wyżej. Ścieżka jest reprezentowana jako lista napisów, gdzie zapisane są wierzchołki początkowy i końcowy krawędzi oraz jej waga. Podobnie, jak poprzednia metoda, ta również najpierw sprawdza, czy macierze `_floyd_distances` i `_floyd_predecessors` są aktualne.

Testy

W pliku *test_graph.py* przy pomocy modułu `unittest` zostały przygotowane dwa bardzo analogiczne zestawy testów – jeden dla grafów skierowanych, drugi dla nieskierowanych. Każdy zestaw zrealizowany został w oddzielnej klasie – `TestDirectedGraph` oraz `TestUndirectedGraph`.



W pierwszym przypadku analizuję graf jak powyżej (zarówno w wersji bez dodatkowej, czerwonej części, jak i z nią) oraz sześciowierzchołkowy graf bez krawędzi.

Przykładowo, testuję długość oraz przebieg najkrótszej ścieżki z „A” do „D”. Jej długość wynosi 2 i przechodzi przez węzeł „B”. Sprawdzam też, że nie istnieje ścieżka z „D” do „A”. Z kolei ścieżka z „A” do „C” wynosi początkowo 10 (przechodzi przez „B”, „D” i „E”), natomiast po dodaniu węzła „F” i dwóch krawędzi ma wagę -1 i przechodzi właśnie przez te nowododane elementy.

Testy dla grafów nieskierowanych opierają się o grafy podstawowe powyższych trzech grafów skierowanych.

Różnicą jest m. in. to, że w grafie czarnym istnieje ścieżka pomiędzy „D” i „A”, której nie było w grafie skierowanym, tak samo jest z „C” i „A”. Natomiast po dodaniu czerwonej części wykrywany jest ujemny cykl i rzucany wyjątek `ValueError`.