Algorytm Prima

Algorytm służący do obliczania minimalnego drzewa rozpinającego dla podanego grafu.

Opis algorytmu

Początkowo drzewo składa się z dowolnie wybranego wierzchołka (korzeń drzewa). W każdym kroku do drzea jest dodawana krawędź o najmniejszej wadze, wychodząca z wierzchołków znajdujących się już w minimalnym drzewie rozpinającym wraz z wierzchołkiem do którego prowadzi.

Jest to algorytm zachłanny, ponieważ w każdym kroku do drzewa jest dodawana krawędź, która w danym momencie wnosi najmniejszą wagę do drzewa.

Złożoność

Cały algorytm ma złożonowść $\mathcal{O}(n_4)$ Na złożoność algorytmu wpływa struktura w pętli while:

Pętla while wykonuje się tyle razy ile jest wierzchołków w grafie czyli $\mathcal{O}(n)$.

```
    while len(wierzcholki) != len(wierzcholkiOdzwiedzone):
        # ...
```

Pierwsza pętla for z każdym kolejnym wykonaniem się pętli while wykonuje się o jeden raz więcej, co też nam daje złożoność $\mathcal{O}(n)$.

```
    for item in wierzcholkiOdzwiedzone:
        # ...
```

Razem te pętle dają złozoność $\mathcal{O}(\sum_{i=1}^{n} i) = \mathcal{O}(n_2)$

Kolejna pętla for iteruje po wszystkich sąsiadach danego wyerzchołka. W najgorszym przypadku wierzchołek może sąsiadować ze wszytkimi wierzchołkami

(także z samym sobą) co daje nam złożoność $\mathcal{O}(n)$.

```
    for sasiad in listaSasiedztwa[item].items():
        # ...
```

Pierwsza instrukcja warunkowa if służy do przeszukiwania listy, które w pythonie ma złożoność $\mathcal{O}(n)$.

```
    if not sasiad[0] in wierzcholkiOdzwiedzone:
        # ...
```

Druga instrukcja warunkowa ma złożoność $\mathcal{O}(1)$.

```
    if temp < minWaga:
        # ...
```

Spis funkcji

Algorytm Prima

Funkcja ta przyjmuje listę sąsiedztwa w postaci (dict+dict), wykonuje algorytm Prima i zwraca obliczone minimalne drzewo dla podanego grafu w postaci (dict+dict)

```
def algorytmPrima(listaSasiedztwa):
    """Funkcja z algorytmem Prima do znajdywania minimalnego drzewa dla grafu"""
    # ...
    return minDrzewo
```

Wpisywanie do grafu

Funkcja przyjmuje jako argumenty graf (postać (dict+dict)), wierzchołek, który chcemy wpisać do grafu oraz jego sąsiada i wagę krawędzi ich łączących. Następnie wpisuje odpowiednio do grafu podaną relację

```
def wpisywanieDoGrafu(graf, wierzcholek, sasiad, waga):
    """Funkcja do tworzenia grafu w postaci (dict+dict)"""
    # ...
```

Wypisawnie stringa

Funkcja po otzrymaniu grafu w postaci (dict+dict) zwraca listę sąsiedztwa jako string

```
def wypisywanieStr(lista):
    """Funkcja do wypisywania minimalnego drzewa jako listy sąsiedztwa"""
    # ...
    return drzewo
```

Wczytah z pliku

Funkcja wczytuje z pliku listaSasiedztwa.txt listę sąsiedztwa grafu i zwraca reprezentacje (dict+dict) grafu.

```
def wczytajZPliku():
    """Funckja do wczytywania grafów z pliku w formie listy sasiedztwa"""
    # ...
    return listaSasiedztwa
```

Zapisz do Pliku

Funkcja przyjmuje graf w reprezentacji (dict+dict) i zapisuje go do pliku minDrzewo.txt w formie listy sąsiedztwa.

```
def zapiszDoPliku(minDrzewo):
    """Funkcja do zapisywania minimalnego drzewa dla podanego grafu w formie listy sąsiedz
    # ...
```

Generuj graf

Funkcja przyjmuje jako argomunty liczbę wierzchołków oraz maksymalną wagę do generowania grafu podane wcześniej przez użytkownika. Następnie generuje losowy graf i zwraca go w postaci (dict+dict).

```
def generujGraf(liczbaWierzcholkow, maxWaga):
    """Funkcja do generowania losowego grafu"""
    # ...
    return graf
```