Mateusz Rus

Projekt zaliczeniowy język Python 2019

Temat: Implementacja grafu na macierzy sąsiedztwa (lista list) wraz z algorytmami BFS i DFS

Wprowadzenie:

Graf – podstawowy obiekt rozważań teorii grafów, struktura matematyczna służąca do przedstawiania i badania relacji między obiektami. W uproszczeniu graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków.

Przykładem reprezentacji grafu jest macierz sąsiedztwa. Jest to macierz kwadratowa, w której wiersze odwzorowują zawsze wierzchołki startowe, a kolumny zawsze wierzchołki końcowe. Jeśli dana krawędź istnieje na przecięciu danych wierzchołków wstawiamy 1 jeśli danej krawędzi nie ma wstawiamy 0. W grafach skierowanych dzięki temu znamy źródło i cel danej krawędzi natomiast w grafach nieskierowanych macierz sąsiedztwa jest symetryczna. Na potrzeby niniejszej implementacji w komórkach macierzy A[i][j] wstawiamy wagi krawędzi.

W projekcie znajdują się nastepujące pliki:

edge.py - zawiera klasę Edge reprezentującą krawędź w grafie

graph.py - zawiera klasę Graph reprezentującą graf

dfs.py - klasa z iteratorem DFS

bfs.py - klasa z iteratorem BFS

traverse.py - funkcje BFS i DFS

test.py - testy sprawdzające poprawność grafu

main.py - przykłady użycia.

Klasa Graph:

Konstruktor:

Graph(n, directed = False)

n- liczba wierzchołków ustalana raz podczas tworzenia grafu

directed – typ bool określa czy graf jest skierowany czy nie.

Graph.is_directed():

Typ bool, zwraca czy graf jest skierowany czy też nie

Graph.v()

zwraca liczbę wierzchołków

Graph.e()

zwraca liczbę krawędzi

Graph.add_node(node):

dodaje wierzchołek

node – wierzchołek, który dodajemy do grafu

Graph.has_node(node)

Sprawdza czy dany wierzchołek znajduje się w grafie.

node – wierzchołek

Zwracany typ: bool

Graph.del_node(node)

Usuwa wierzchołek

node - wierzchołek

Graph.add_edge(edge)

Wstawienie krawędzi

edge – krawędź w postaci obiektu Edge

Graph.has_edge(edge)

Sprawdza czy istnieje dana krawedz.

edge – krawedź w postaci obiektu Edge

Zwracany typ: bool

Graph.del_edge(edge)

Usunięcie krawędzi

Edge – krawedź w postaci obiektu Edge

Graph.weight(edge)

Zwraca wagę krawędzi

edge - krawedź w postaci obiektu Edge

Graph.iternodes()

Zwraca iterator po wierzchołkach

Graph.iteradjacent(node)

Zwraca iterator po wierzchołkach sąsiednich

node – wierzchołek

Graph.iteroutedges(node)

Zwraca iterator po krawędziach wychodzących

node - wierzchołek

Graph.iterinedges(node)

Zwraca iterator po krawędziach przychodzących

node - wierzchołek

Graph.iteredges()

Zwraca iterator po krawędziach

Graph.copy(self)

Zwraca kopię grafu

Graph.transpose()

Zwraca graf transponowany

Graph.complement()

Zwraca dopełnienie grafu

Graph.subgraph(nodes)

Zwraca podgraf indukowany

nodes – lista wierzchołków

Graph.iter_bfs(start_node)

Zwraca iterator bfs.

start_node – wierzchołek startowy

Graph. iter_dfs(self, start_node):

Zwraca iterator dfs.

start_node – wierzchołek startowy

Graph.traverse_dfs(self, start, visit=visit, visited=None):

Funkcja przchodząca graf w głąb.

start – wierzchołek startowy

visit – funkcja wywoływana dla każdego wierzchołka

visited – tablica wierzcholków odwiedzonych

Graph.traverse_bfs(self, start, visit=visit):

Funkcja przchodząca graf wszerz.

start – wierzchołek startowy

visit – funkcja wywoływana dla każdego wierzchołka

Klasa Edge:

Konstruktor:

edge(self, source, target, weight)

source - źródło - skąd wychodzi krawędź (z jakiego wierzchołka, przy grafie nieskierowanym bez znaczenia)

target – cel - dokąd wchodzi krawędź (do jakiego wierzchołka wchodzi krawędź)

weight – waga krawędzi.

```
dGraph = Graph(8, True)
uGraph = Graph(8)

print('Generowanie grafu: \n')

for i in range(7):
    dGraph.add_node(i)
    uGraph.add_node(i)

for i in range(random.randint(1, dGraph.v())):
    for j in range(random.randint(1, dGraph.v())):
        if i != j:
            dGraph.add_edge(Edge(i, j, random.randint(1,20)))
            uGraph.add_edge(Edge(i, j, random.randint(1,20)))

print("Reprezentacja  grafu na macierzy: \n")

dGraph.__repr__()
print('\n')
uGraph.__repr__()
```

```
Generowanie grafu:


Reprezentacja  grafu na macierzy:


0   8   7   3   8   5   8   0
12  0   6   10  18  12  16  0
12  6   0   9   15  17  0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0



0   10  14  20  15  3   9   0
10  0   6   8   16  3   7   0
14  6   0   8   19  20  0   0
20  8   8   0   0   0   0   0
15  16  19  0   0   0   0   0
3   3   20  0   0   0   0   0
9   7   0   0   0   0   0   0
0   0   0   0   0   0   0   0


Process finished with exit code 0
```

```
dGraph = Graph(8, True)
"""BFS i DFS"""
dGraph.traverse_dfs(2)
for i in dGraph.iter_dfs(2):
    print(i)

dGraph.traverse_bfs(2)
for i in dGraph.iter_bfs(2):
    print(i)

dGraph.__repr__()

for i in dGraph.iter_bfs(0):
    print(i)

for i in dGraph.iter_dfs(0):
    print(i)

dGraph.traverse_bfs(0)
dGraph.traverse_dfs(0)
```