Algorytmy i struktury danych (studia stacjonarne)

Dr Anna Muranova

Semestr zimowy 2025/2026, UWM w Olsztynie

Ćwiczenie 10 (Algorytm Dijkstry) 17.12.2025

Zadanie 10.1. Zrobić kolejką priorytetową `PriorityQueue` na podstawie słownika, gdzie kluczy mogą być jakimikolwiek, a wartości są liczbami.

```
class PriorityQueue:

    def __init__(self, dict):
        self.S = dict

    def __str__(self):
        return str(self.S)
```

Zaimplementować następujący metody:

(a) Metoda `insert(v, priority)` wstawi pod wartość `priority` pod (nowym) kluczem `v`.

(b) Metoda `minimum()` zwróci najmniejszą wartość i jej klucz.

(c) Metoda `extract_min()` zwróci najmniejszą wartość i jej klucz i usunie ten klucz (i wartość).

(d) Metoda `decrease_distance(self, v, k)` zamienia wartość pod kluczem `v` na `k`, tylko jeżeli ona większa od `k`. Zwroci **True** jeżeli była zamiana. **False** inaczej

(e) Testy:

```
queue = PriorityQueue({'a':0,'b':2,'c':3, 'd':0})
assert str(queue) == "{'a': 0, 'b': 2, 'c': 3, 'd': 0}"
queue.insert('e',0.5)

assert str(queue) == "{'a': 0, 'b': 2, 'c': 3, 'd': 0, 'e': 0.5}"
assert queue.minimum()== ('a', 0)
assert str(queue) == "{'a': 0, 'b': 2, 'c': 3, 'd': 0, 'e': 0.5}"
assert queue.extract_min()== ('a', 0)
assert str(queue) == "{'b': 2, 'c': 3, 'd': 0, 'e': 0.5}"
queue.decrease_distance('b',1)
assert str(queue) == "{'b': 1, 'c': 3, 'd': 0, 'e': 0.5}"
queue.decrease_distance('d',5)
assert str(queue) == "{'b': 1, 'c': 3, 'd': 0, 'e': 0.5}"
```

Zadanie 10.2. (a) Dodaj do klasy `Graf` metodą `weight(vertex1, vertex2)` która zwraca wagą krawędzi pomiędzy wierzchołkami lub $float('inf')$.

(b) Teraz masz wszystko żeby napisać w grafie metodą `dijkstra(vertex0)`.

(1) Przekształć graf na słownik sąsiadów.

(2) Utwórz słownik `d` w którym przechowujesz bieżące ścieżki i ich długość dla każdego wierzchołka. Na początku ścieżki są puste, długość nieskończoność do innych, dla wszystkich oprócz `v0`.

```
d[v0] = [[v0], 0]
```

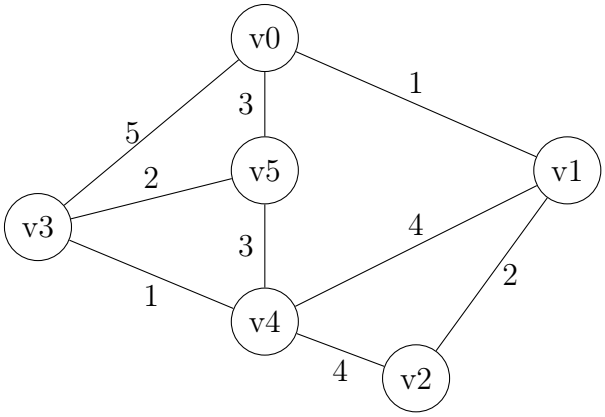
(3) Utwórz priorytetową kolejką `queue` ze wszystkich wierzchołków i odpowiednich długości ścieżek.

(4) Dopóki kolejka nie jest pusta:

- Pobierz wierzchołek `u` z kolejki.
- dla wszystkich jego sąsiadów, które są w kolejce: jeżeli ścieżka przez `u` (=ścieżka do u + weight(u,v)) jest krótsza, niż dotychczasowa, to zamień ścieżką i waga w `d` oraz priorytet w kolejce (użyj funkcji `queue.decrease_distance`)

(5) Zwróć `d`

(c) Przykład działania:



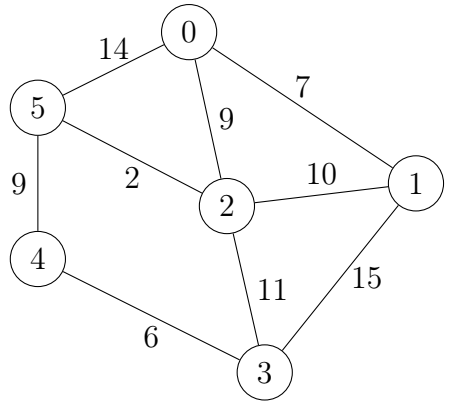

```
G = Graph()
for i in range(6):
    G.add_vertex(f"v{i}")
G.add_edge(G.vertices[0], G.vertices[1],1 )
G.add_edge(G.vertices[0], G.vertices[3],5 )
G.add_edge(G.vertices[0], G.vertices[5],3 )
G.add_edge(G.vertices[1], G.vertices[2],2 )
G.add_edge(G.vertices[1], G.vertices[4],4 )
G.add_edge(G.vertices[2], G.vertices[4],4 )
G.add_edge(G.vertices[3], G.vertices[5],2 )
G.add_edge(G.vertices[3], G.vertices[4],1 )
G.add_edge(G.vertices[4], G.vertices[5],3 )
print()
d = G.dijkstra(G.vertices[2])
for x in d:
    print(f"{x}:", end=" ")
```

```
    for y in d[x][0]:
        print(f"{y},", end=" ")
    print(d[x][1])
```

Wynik:

```
v0: v2, v1, 3
v1: v2, 2
v2: 0
v3: v2, v4, 5
v4: v2, 4
v5: v2, v1, v0, 6
```

(d) Jeszcze przykład działania:




```
G = Graph()
G.from_adjacency_matrix([[0,7,9,0,0,14],[7,0,10,15,0,0],
[9,10,0,11,0,2],[0,15,11,0,6,0],[0,0,0,6,0,9],[14,0,2,0,9,0]])
d = G.dijkstra(G.V[0])
for x in d:
    print(f"{x}:", end=" ")
    for y in d[x][0]:
        print(f"{y},", end=" ")
    print(d[x][1])
```

Wynik:

```
v0: v0, 0
v1: v0, v0, 7
v2: v0, v0, 9
v3: v0, v0, v2, 20
v4: v0, v0, v2, v5, 20
v5: v0, v0, v2, 11
```

Zadanie 10.3. * Zaimplementuj algorytm Dijkstry dla macierze sąsiedztwa pryz pomocy list (bez słowników, bez klasy `Graph`).