8_2-representationGraph

1 Representasi Graph

Adjacency List, Adjacency Matrix

Terdapat dua cara untuk merepresentasikan suatu graph, yaitu : 1. Adjacency List 2. Adjacency Matrix

Representasi *graph* dan dan implementasi masing-masing representasi tersebut, adalah sebagai berikut

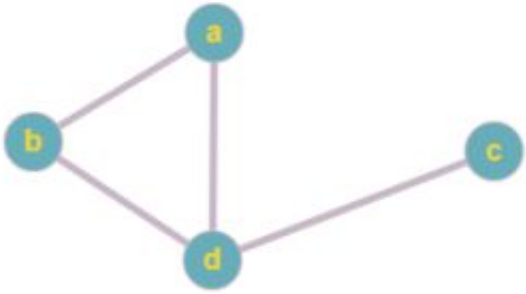
1. Section 1.1
    - Section 1.1.1
    - Section 1.1.3
    - Section 1.2.1
2. Section 1.3

1.1 Adjacency List

Representasi graph adjacency list di dalam python, dapat dilakukan dengan menggunakan dua cara : 1. list atau dictionary 2. linked list

Indeks dari *list/dictionary/linked list* merepresentasikan sebuah *vertex* atau *node* asal, dan setiap elemen dalam *list/dictionary/linked list* tersebut merepresentasikan *vertex* lain yang terhubung dari *vertex* asal. Dengan representasi *adjacency list* dapat memudahkan untuk menemukan semua tautan *vertex* yang terhubung secara langsung dengan *vertex* asal.

1.1.1 Representasi dengan List



Contoh *graph* ditunjukkan pada Gambar 1. **Gambar 1. *Graph*** Untuk setiap *vertex* (V) akan menyimpan *list* yang berisi tetangga dari V. $A : [b, d]$

artinya *vertex a* mempunyai tetangga *b* dan *d*.

Setiap *vertex* pada Gambar 1 tersebut terkoneksi dengan beberapa *vertex*, sebagai berikut : $a : [b, d]$ $b : [a, d]$ $c : [d]$ $d : [a, b, c]$

1.1.2 Code

berikut adalah implementasi representasi *graph* tersebut

```
In [1]: # list of lists
        adjLists = [ ['b','d'], ['a','d'], ['d'], ['a','b','c'] ]
        node= ['a','b','c','d']
        # testing
        print("Semua tetangga dari vertex a: ", adjLists[0])
        print("Semua tetangga dari vertex d: ", adjLists[3])
        print("\nMenampilkan semua vertex dan tetangganya masingmasing")
        n = len(adjLists)
        for n in range(0,n):
            print(node[n], ":", adjLists[n])
```

```
Semua tetangga dari vertex a:  ['b', 'd']
Semua tetangga dari vertex d:  ['a', 'b', 'c']

Menampilkan semua vertex dan tetangganya masingmasing
a : ['b', 'd']
b : ['a', 'd']
c : ['d']
d : ['a', 'b', 'c']
```

Section 1

1.1.3 Representasi dengan Dictionary

Alternatif lain untuk mengimplementasikan *adjacency list* menggunakan *dictionary*, yang terdiri dari pasangan $(key, record) = (node, adjList)$.

1.1.4 Code

Berikut implementasi *adjacency list* dengan *dictionary*

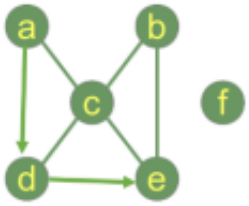
```
In [1]: adjLists_dict = {}
        # insert (vertex, list) pairs into dictionary
        adjLists_dict = {'a': ['b', 'd'],
         'b': ['a', 'd'],
         'c': ['d'],
         'd': ['a','b','c'],
         }
        # testing
        print("Semua tetangga dari vertex indeks ke a: ",
```

```
        adjLists_dict['a'])
        print("Semua tetangga dari vertex indeks ke d: ",
        adjLists_dict['d'])
        print("\nMenampilkan semua vertex dan tetangganya masing-masing")
        for node in adjLists_dict:
              print(node, ":", adjLists_dict[node])
```

```
Semua tetangga dari vertex indeks ke a:  ['b', 'd']
Semua tetangga dari vertex indeks ke d:  ['a', 'b', 'c']

Menampilkan semua vertex dan tetangganya masing-masing
a : ['b', 'd']
b : ['a', 'd']
c : ['d']
d : ['a', 'b', 'c']
```

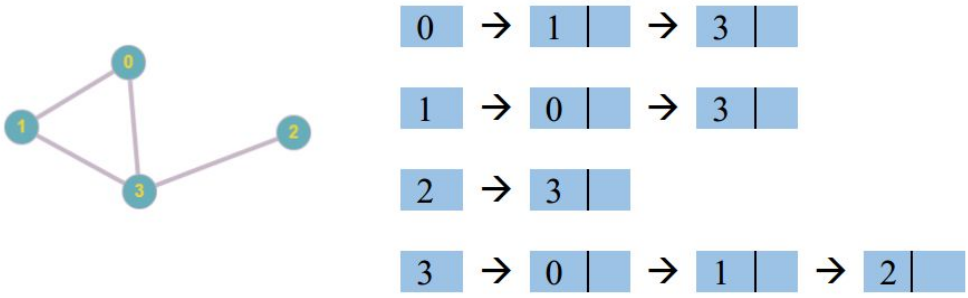
1.2 Latihan - 1



Dari graph berikut buatlah, struktur datanya menggunakan dictionary.
Section 1

1.2.1 Representasi dengan Linked List

Contoh graph dan ilustrasi representasi dengan linked list dapat dilihat pada Gambar 2.



Gambar 2. ***Graph* dan bentuk *Linked list***

1.2.2 Code

Berikut adalah implementasi representasi *graph* dengan menggunakan *linked list* :

```
In [6]: # Class untuk node dalam graph
        class AdjNode:
```

```
        def __init__(self, data): #constructor
            self.vertex = data
            self.next = None

    # Class Graph, ukuran array adalah jumlah dari vertex
    class Graph:
        def __init__(self, vertices): #constructor
            self.V = vertices
            self.graph = [None] * self.V

        def add_edge(self, src, dest): # Fungsi untuk menambah edge
        # Add node ke source node
            node = AdjNode(dest)
            node.next = self.graph[src]
            self.graph[src] = node
        # add source node ke destination
            node = AdjNode(src)
            node.next = self.graph[dest]
            self.graph[dest] = node

        def print_graph(self): # Fungsi untuk print graph
            for i in range(self.V):
                print("Adjacency list dari vertex {} \n head".format(i), end="")
                temp = self.graph[i]
                while temp:
                    print(" -> {}".format(temp.vertex), end="")
                    temp = temp.next
                    print(" \n")

    # Driver program to the above graph class
    V = 4
    graph = Graph(V)
    graph.add_edge(0, 1)
    graph.add_edge(0, 3)
    graph.add_edge(1, 3)
    graph.add_edge(2, 3)
    graph.print_graph()
```

```
Adjacency list dari vertex 0
 head -> 3

 -> 1

Adjacency list dari vertex 1
 head -> 3

 -> 0
```

```
Adjacency list dari vertex 2
 head -> 3

Adjacency list dari vertex 3
 head -> 2

 -> 1

 -> 0
```

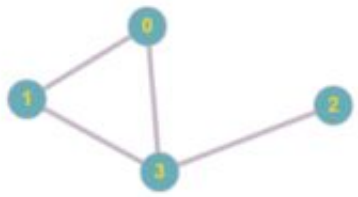
Section 1

1.3 Adjacency Matrix

Cara lain untuk merepresentasikan *graph* $G = \{V, E\}$ sebagai matriks boolean. Ukuran matriks adalah ordo $V \times V$ dimana V adalah jumlah *vertex* dalam *graph*. Baris direpresentasikan sebagai titik awal dan kolom sebagai titik tujuan. Untuk *Unweighted Graph* maka Nilai $A(i, j)$ adalah 1 atau 0 tergantung ada tidaknya garis (*edge*) yang menghubungkan dari titik i ke titik j. Bernilai 1 jika ada *edge*, dan 0 jika tidak ada *edge* antar titik tersebut. *Weighted Graph*, maka nilai matrik diisi dengan bobot dari *edge* $A(i, j) = nilaiBobot$.

1.3.1 Adjacency Matrix untuk Undirected Graph

Contoh *graph* dan representasi *adjacency matrix* untuk *undirected graph*, dapat dilihat pada Gambar

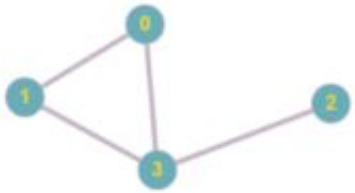


	0	1	2	3
0	0	1	0	1
1	1	0	0	1
2	0	0	0	1
3	1	1	1	0

3. **Gambar 3. *Undirected Graph* dan representasi *Adjacency Matrix***

1.3.2 Adjacency Matrix untuk Weighted Graph

Contoh *graph* dan representasi *adjacency matrix* untuk *weighted graph*, dapat dilihat pada Gambar 4.

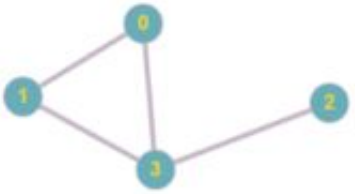


	0	1	2	3
0	0	1	0	1
1	1	0	0	1
2	0	0	0	1
3	1	1	1	0

Gambar 4. ***Weighted Graph*** **dan representasi** ***Adjacency Matrix***

1.3.3 Adjacency Matrix untuk Directed Graph

Contoh *graph* dan representasi *adjacency matrix* untuk *directed graph*, dapat dilihat pada Gambar



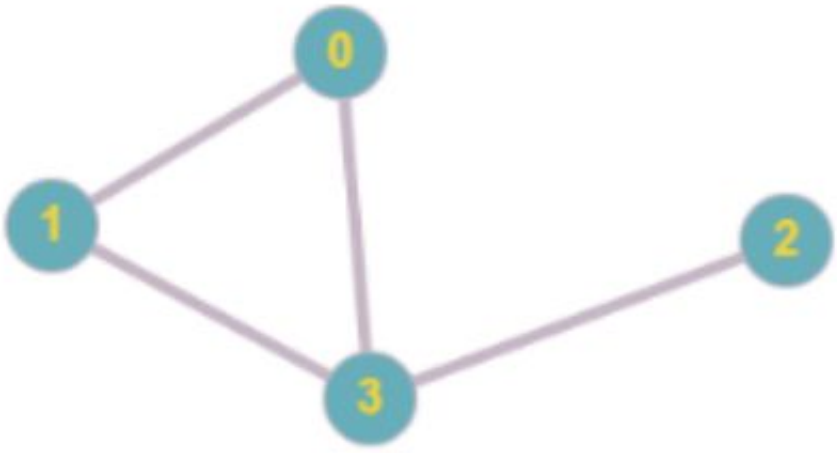
	0	1	2	3
0	0	1	0	1
1	1	0	0	1
2	0	0	0	1
3	1	1	1	0

5.

Gambar 5. ***Directed Graph*** **dan representasi** ***Adjacency Matrix***

1.3.4 Code

Terdapat contoh *undirected graph* seperti yang ditunjukkan pada Gambar 6.



Gambar 6. ***Undirected Graph*** Bentuk graph dibuatkan edgenya, yaitu $Edge(0,1)$; $Edge(0,3)$; $Edge(1,3)$; dan $Edge(2,3)$;

```
In [10]: class Graph(object):
             def __init__(self, size):
```

```
            self.adjMatrix = []
            for i in range(size):
                self.adjMatrix.append([0 for i in range(size)])
                self.size = size
        def addEdge(self, v1, v2):
            if v1 == v2:
                print("Same vertex %d and %d" % (v1, v2))
            self.adjMatrix[v1][v2] = 1
            self.adjMatrix[v2][v1] = 1
        def removeEdge(self, v1, v2):
            if self.adjMatrix[v1][v2] == 0:
                print("No edge between %d and %d" % (v1, v2))
                return
            self.adjMatrix[v1][v2] = 0
            self.adjMatrix[v2][v1] = 0

        def containsEdge(self, v1, v2):
            return True if self.adjMatrix[v1][v2] > 0 else False
        def __len__(self):
            return self.size
        def toString(self):
            for row in self.adjMatrix:
                for val in row:
                    print('{:4}'.format(val), end = ' ')
                print()
    def main():
        g = Graph(4)
        g.addEdge(0, 1)
        g.addEdge(0, 3)
        g.addEdge(1, 3)
        g.addEdge(2, 3)
        g.toString()

    main()
```

```
0    1    0    1
1    0    0    1
0    0    0    1
1    1    1    0
```

Section 1