CANDIDATE NUMBER:

IB255

EGT3

ENGINEERING TRIPOS PART IIB

Monday 2 May 2022 2 to 4.10

Module 4M26

ALGORITHMS AND DATA STRUCTURES

*Answer not more than **three** questions.*

All questions carry the same number of marks.

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

Write your candidate number not your name on the cover sheet.

STATIONERY REQUIREMENTS
Single-sided script paper

SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM
Computing equipment provided
CUED approved calculator allowed
Engineering Data Book

10 minutes reading time is allowed for this paper at the start of the exam. You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.

1.

Function **WeightedPath(strArr)** takes **strArr**, an array of strings, of the following structure as its input. The first element in the array is the number of nodes **N** in the graph. The next **N** elements are node names (e.g. A, B, C .. BS, MS .. etc.). The rest of the elements in the array are the connections between all of the nodes along with their weights separated by a pipe symbol (|): A|B|3, B|C|12 .. BS|MS|14 .. etc. The graph is undirected and node weights are always positive. Note, there may exist no connections at all.

Your program should implement Dijkstra's algorithm and return the shortest path (on the graph encoded by **strArr**) from the first node in the list to the last node in the list. There will only ever be one shortest path for any given graph. If no path between the first and the last node exists, your code should return -1. The array will have at least two nodes.

Note. File ./q1/utilities.py contains useful implementations of: (i) a priority queue based on **Heap** data structure, (ii) a **Node** class storing properties such as ***parent***, ***neighbours***, ****visited*** flag and shortest ***distance*** to this node as well as other useful properties, and (iii) a simple function **adj_mat_and_s_e_from_str** which convert string arrray (**strArr**) into a tuple containing the graph adjacency list **G** and start node **s** and end node **e** (see code below).

Examples

Input: ["4","A","B","C","D", "A|B|2", "B|C|11", "C|D|3", "B|D|2"]
Desired output: A-B-D

Input: ["4","x","y","z","w","x|y|2","y|z|14", "z|y|31"]
Desired output: -1

(a) Extend function *Dijkstra* in order to solve the aforementioned task. **[40%]**

In [2]:

```python
from utilities import Heap, Node, adj_mat_and_s_e_from_str
def Dijkstra(G, s, e): #G - adjacency list, s - start node, e - end node.
    # Returns a tuple (distance, path), where distace is the length of the
    # shortest path and **path** is a string containing node names in the
    # shortest path from s to e, separated by "-". If no path exists, it
    #returns None.

    nodes = {name: Node(name, neighbors) for name, neighbors in G.items()}
    nodes[s].distance = 0
    pq = Heap()
    for node in nodes.values():
        pq.push(node, node.distance)

    while len(pq):
        node = pq.pop()[0]
        node.visited = True
        if node.name == e:
            break
        for name in node.neighbors:
            neighbor = nodes[name]
            if ((not neighbor.visited) and
                        (node.distance + node.neighbors[name] < neighbor.distanc
                neighbor.distance = node.distance + node.neighbors[name]
                neighbor.parent = node
                pq.push(neighbor, neighbor.distance)

    path = []
    node = nodes[e]
    if not node.parent:
      return None
    while node:
        path.append(node)
        node = node.parent

    return (nodes[e].distance, '-'.join([node.name for node in path[::-1]]))

def ShortestPath(strArr):
  G, s, e = adj_mat_and_s_e_from_str(strArr)

  path_info = Dijkstra(G, s, e)
  return path_info[1] if path_info else "-1"

```

Example Test Case - 1

In [3]:

```python
input_value = ["4","A","B","C","D", "A|B|2", "B|C|11", "C|D|3", "B|D|2"]
print (ShortestPath(input_value))
```

```
A-B-D
```

Example Test Case - 2

In [4]:

```python
input_value =  ["4","x","y","z","w","x|y|2","y|z|14", "z|y|31"]
print (ShortestPath(input_value))
```

```
-1
```

Automatic Evaluation

Function **evaluate_solution** evaluates your function **ShortestPath** on **10** pre-set test cases. Note, when debugging you may want to run your code on a subset of tests.

Do not forget to run on all test cases when your final implementation is finished!

In [5]:

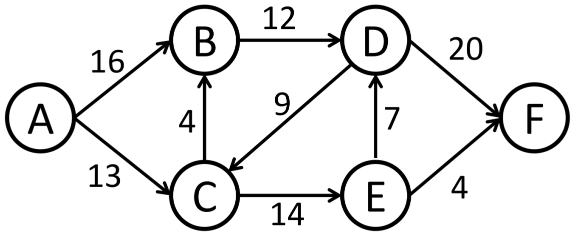
```
from evaluation_script import evaluate_solution
evaluate_solution(question_id=1,question_part_id=1,function=ShortestPath,test_ca
```

```
Running question 1, part 1, test id: 1
Input:
['3', 'A', 'B', 'C', 'B|C|13', 'A|B|2']
Obtained output:
A-B-C
<<CORRECT>>
Running question 1, part 1, test id: 2
Input:
['6', 'A', 'B', 'C', 'D', 'E', 'F', 'B|A|2', 'A|F|12', 'A|C|4', 'B|D|
1', 'D|E|1', 'C|D|4', 'F|E|1']
Obtained output:
A-B-D-E-F
<<CORRECT>>
Running question 1, part 1, test id: 3
Input:
['6', 'A', 'B', 'C', 'D', 'E', 'F', 'B|A|2', 'A|F|3', 'A|C|4', 'B|D|
1', 'D|E|1', 'C|D|4', 'F|E|1']
Obtained output:
A-F
<<CORRECT>>
Running question 1, part 1, test id: 4
Input:
['6', 'D', 'B', 'C', 'A', 'E', 'F', 'B|A|2', 'A|F|3', 'A|C|4', 'B|D|
1', 'D|E|12', 'C|D|4', 'F|E|1']
Obtained output:
D-B-A-F
<<CORRECT>>
Running question 1, part 1, test id: 5
Input:
['3', 'AA', 'BB', 'CC', 'CC|BB|102']
Obtained output:
-1
<<CORRECT>>
Running question 1, part 1, test id: 6
Input:
['8', 'C', 'B', 'A', 'D', 'E', 'F', 'G', 'H', 'C|D|1', 'D|F|2', 'G|F|
2', 'G|E|1', 'E|B|1', 'H|B|1', 'C|A|13', 'B|A|2', 'C|E|9']
Obtained output:
C-D-F-G-E-B-H
<<CORRECT>>
Running question 1, part 1, test id: 7
Input:
['7', 'D', 'A', 'N', 'I', 'E', 'L', 'B', 'D|A|1', 'A|N|2', 'L|B|22']
Obtained output:
-1
<<CORRECT>>
Running question 1, part 1, test id: 8
Input:
['3', 'GG', 'HH', 'JJ', 'GG|JJ|6', 'GG|HH|2', 'JJ|HH|2']
Obtained output:
GG-HH-JJ
<<CORRECT>>
Running question 1, part 1, test id: 9
Input:
['5', 'c', 'a', 'b', 'd', 'e', 'c|a|3', 'a|b|2', 'a|d|34', 'b|e|15',
'e|d|2']
Obtained output:
c-a-b-e
<<CORRECT>>
Running question 1, part 1, test id: 10
Input:
['8', 'C', 'B', 'A', 'D', 'E', 'F', 'G', 'H', 'C|D|1', 'D|F|2', 'G|F|
2', 'G|E|1', 'E|B|1', 'H|B|1', 'C|A|13', 'B|A|2', 'C|E|1']
Obtained output:
C-E-B-H
<<CORRECT>>

----------------------------------
Total correct outputs:
10 out of 10
```

(b) Answer questions about the graph illustrated below:

[15%]



(i) List all strongly connected components of this graph.

{A}, {B,C,E,D}, {F}

(ii) Calculate the maximum flow from node A to node F.

Augmented paths:
A-C-E-F -> 4
A-C-E-D-F -> 7
A-B-D-F ->12

Total flow: 23

(iii) Assume this graph is transformed into an undirected graph by replacing all directed edges with equivalent undirected edges of same corresponding weights. Write out the edges of the Minimum Spanning Tree discovered by Kruskal's algorithm in the same order as they would be found.

C-B, E-F, E-D, D-C, A-C

(c) Provide brief answers to questions below:

[20%]

(i) Assume that you are provided with an undirected graph which contains both positive and negative edges as well as an implementation of Prim's Minimum Spanning Tree (MST) algoritm, which only accepts graphs with non-negative edge weights. Can you still obtain the MST for this graph without modifying the Prim's algorithm implementation. Explain how.

Modify the graph by simply adding a positive constant C to all edges where C is the weight of the negative weight edge of largest magnitude in the original graph G. Then the total edge weight of the MST of the new graph will be equal to M+C(V-1). Here M is the total edge weight of the original graph and V is number of vertices (assuming all vertices are connected).

(ii) What are the advantages and disadvantages of the Bellman-Ford algorithm in comparison to Dijkstra's algorithm?

Advantages: (1) it can be applied to graphs with non-negative edges. (2) It can be also used to detect negative-weight cycles.

Disadvantage: Bellman-Ford algorithm has a higher runtime - O(VE). Runtime of Dijkstra's algorithm is O(V log V + E).

(iii) What is the order of growth of the memory used to represent a graph with V vertices and E edges using the adjacency-lists representation and the adjacency-matrix representation?

Adjacency list - O(V+E). Adjacency matrix - O(V^2).

(iv) Given a digraph where each edge is colored black or orange and two vertices s and t. Is it possible to find a path from s to t that uses the fewest number of black edges in time proportional to E + V in the worst case? Justify your answer.

This can be solved by a modified version of breadth-first search, where the vertices discovered by following black edges are put at the back of the queue (as usual) and the vertices discovered by following orange edges are put at the front of the queue.

(d) Extend your algorithm described in part (a) to find the second shortest path in the graph.

[25%]

Inputs and expected outputs have the same format as in part (a).

Examples

Input: ["4","A","B","C","D", "A|B|2", "B|C|11", "C|D|3", "B|D|2"]
Output: A-B-C-D

Input: ["4","x","y","z","w","x|y|2","y|z|14", "z|y|31"]
Output: -1

In [5]:

```python
def SecondShortestPath(strArr):
    G, s, e = adj_mat_and_s_e_from_str(strArr)

    LARGE=100000
    shortest_path = Dijkstra(G, s, e)
    if not (shortest_path):
        return "-1"
    p = shortest_path[1].split('-')
    shortest_2nd=None
    cost=0

    for i in range(0,len(p)-1):

        tmp_edge=G[p[i]][p[i+1]]
        G[p[i]][p[i+1]]=LARGE

        new_p = Dijkstra(G,p[i],e)

        G[p[i]][p[i+1]]=tmp_edge

        if new_p and (shortest_2nd is None or shortest_2nd[0]>cost+new_p[0]):
            shortest_2nd = (cost+new_p[0], '-'.join(p[:i]+new_p[1].split('-')))

        cost=cost+tmp_edge

    if(shortest_2nd is None)or shortest_2nd[0]>LARGE:
        return "-1"
    else:
        return shortest_2nd[1]
```

Example test case - 1

In [6]:

```python
input_value = ["4","A","B","C","D", "A|B|2", "B|C|11", "C|D|3", "B|D|2"]
print (SecondShortestPath(input_value))
```

```
A-B-C-D
```

Example test case - 2

In [7]:

```python
input_value =  ["4","x","y","z","w","x|y|2","y|z|14", "z|y|31"]
print (SecondShortestPath(input_value))
```

```
-1
```

Automatic evaluation

Function **evaluate_solution** evaluates your function **SecondShortestPath** on **10** pre-set test cases. Note, when debugging you may want to run your code on a subset of tests.

Do not forget to run on all test cases when your final implementation is finished!

In [8]:

```python
from evaluation_script import evaluate_solution
evaluate_solution(question_id=1,question_part_id=2,function=SecondShortestPath,t
```

```
Running question 1, part 2, test id: 1
Input:
['3', 'A', 'B', 'C', 'B|C|13', 'A|B|2']
Obtained output:
-1
<<CORRECT>>
Running question 1, part 2, test id: 2
Input:
['6', 'A', 'B', 'C', 'D', 'E', 'F', 'B|A|2', 'A|F|12', 'A|C|4', 'B|D|
1', 'D|E|1', 'C|D|4', 'F|E|1']
Obtained output:
A-C-D-E-F
<<CORRECT>>
Running question 1, part 2, test id: 3
Input:
['6', 'A', 'B', 'C', 'D', 'E', 'F', 'B|A|2', 'A|F|3', 'A|C|4', 'B|D|
1', 'D|E|1', 'C|D|4', 'F|E|1']
Obtained output:
A-B-D-E-F
<<CORRECT>>
Running question 1, part 2, test id: 4
Input:
['6', 'D', 'B', 'C', 'A', 'E', 'F', 'B|A|2', 'A|F|3', 'A|C|4', 'B|D|
1', 'D|E|12', 'C|D|4', 'F|E|1']
Obtained output:
D-C-A-F
<<CORRECT>>
Running question 1, part 2, test id: 5
Input:
['3', 'AA', 'BB', 'CC', 'CC|BB|102']
Obtained output:
-1
<<CORRECT>>
Running question 1, part 2, test id: 6
Input:
['8', 'C', 'B', 'A', 'D', 'E', 'F', 'G', 'H', 'C|D|1', 'D|F|2', 'G|F|
2', 'G|E|1', 'E|B|1', 'H|B|1', 'C|A|13', 'B|A|2', 'C|E|9']
Obtained output:
C-E-B-H
<<CORRECT>>
Running question 1, part 2, test id: 7
Input:
['7', 'D', 'A', 'N', 'I', 'E', 'L', 'B', 'D|A|1', 'A|N|2', 'L|B|22']
Obtained output:
-1
<<CORRECT>>
Running question 1, part 2, test id: 8
Input:
['3', 'GG', 'HH', 'JJ', 'GG|JJ|6', 'GG|HH|2', 'JJ|HH|2']
Obtained output:
GG-JJ
<<CORRECT>>
Running question 1, part 2, test id: 9
Input:
['5', 'c', 'a', 'b', 'd', 'e', 'c|a|3', 'a|b|2', 'a|d|34', 'b|e|15',
'e|d|2']
```

```
Obtained output:
c-a-d-e
<<CORRECT>>
Running question 1, part 2, test id: 10
Input:
['8', 'C', 'B', 'A', 'D', 'E', 'F', 'G', 'H', 'C|D|1', 'D|F|2', 'G|F|
2', 'G|E|1', 'E|B|1', 'H|B|1', 'C|A|13', 'B|A|2', 'C|E|1']
Obtained output:
C-D-F-G-E-B-H
<<CORRECT>>

----------------------------------
Total correct outputs:
10 out of 10
```