CS69001 Computing Laboratory – I

Assignment No: D2

Date: 03–September–2018

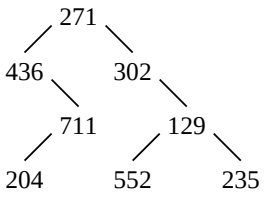
Write Python programs to solve the following parts.

Part 1

Solve Assignment B2 in Python. This part involves the following tasks.

- The input is provided in the file **`D21in.txt`**. The file stores a level-by-level and left-to-right-in-each-level listing of triples (k, l, r) with one triple per line. Read the file, and build your own tree in a list-of-lists format. Do **not** use any external library like **`NetworkX`** in this part. The following figure shows the representation format.

Binary tree



File content

```
271  1  1
436  0  1
302  0  1
711  1  0
129  1  1
204  0  0
552  0  0
235  0  0
```

List of lists

271	1	2
436	None	3
302	None	4
711	5	None
129	6	7
204	None	None
552	None	None
235	None	None

- Write functions to make preorder and inorder listing of keys in a binary tree stored in this format.
- Write a function to convert the tree to a max-heap (ordering only) in place.
- Write a function to convert the heap to a binary search tree in place. Use the algorithm described in Assignment B2.
- Output preorder and inorder listings of the initial tree, the intermediate heap, and the final BST.

Sample output

The output for the sample of Assignment B2 is given below.

```
+++ Tree created

[297, 1, 2]
[319, None, 3]
[124, 4, 5]
[282, None, None]
[530, 6, 7]
[424, None, None]
[287, 8, 9]
[214, None, 10]
[471, None, None]
[376, 11, 12]
[527, None, 13]
[149, None, 14]
[173, None, None]
[200, None, None]
[253, None, None]

+++ Preorder : 297 319 282 124 530 287 471 376 149 253 173 214 527 200 424
+++ Inorder  : 319 282 297 471 287 149 253 376 173 530 214 527 200 124 424

+++ Converted to heap

+++ Preorder : 530 319 282 527 471 376 287 297 253 149 173 214 200 124 424
+++ Inorder  : 319 282 530 287 376 253 149 297 173 471 214 200 124 527 424

+++ Converted to BST

+++ Preorder : 173 124 149 527 319 214 200 287 253 282 297 376 424 471 530
+++ Inorder  : 124 149 173 200 214 253 282 287 297 319 376 424 471 527 530
```

Part 2

Use the **NetworkX** library to solve the following problem. The file **D22in.txt** stores the data for a weighted undirected graph $G = (V, E)$. Assume that $V = \{0, 1, 2, \ldots, n-1\}$. The first two lines of the file store $n = |V|$ (the number of vertices in G) and $m = |E|$ (the number of edges). This is followed by m lines each storing a triple (u, v, w), where $u, v \in V$ with $u < v$, and w is the weight of the (undirected) edge (u, v). Prepare a **NetworkX Graph** by reading the input file. You may assume that the m edges stored in the input file are distinct from one another.

Generate one minimum spanning tree (MST) of G. Print the edges in the tree, and the cost of the MST.

Generate all the MSTs of G, and print their edges and costs. You may recursively construct all the $(n-1)$-element subsets of E. and print such a subset if and only if it is a tree and is an MST.

Sample output

```
FILE: D22in.txt

   10
   15
   0 1 4
   0 2 3
   0 3 4
   0 7 4
   0 9 4
   1 5 1
   1 9 3
   3 5 3
   3 8 1
   4 6 4
   4 9 1
   5 8 4
   6 7 1
   7 8 2
   8 9 3

PROGRAM OUTPUT:

   +++ MST found
   (0,1):4 (0,2):3 (1,5):1 (1,9):3 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 [19]

   +++ Generating all MSTs
   (0,1):4 (0,2):3 (1,5):1 (1,9):3 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 [19]
   (0,1):4 (0,2):3 (1,5):1 (1,9):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,1):4 (0,2):3 (1,5):1 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,3):4 (1,5):1 (1,9):3 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 [19]
   (0,2):3 (0,3):4 (1,5):1 (1,9):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,3):4 (1,5):1 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,7):4 (1,5):1 (1,9):3 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 [19]
   (0,2):3 (0,7):4 (1,5):1 (1,9):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,7):4 (1,5):1 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,9):4 (1,5):1 (1,9):3 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 [19]
   (0,2):3 (0,9):4 (1,5):1 (1,9):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
   (0,2):3 (0,9):4 (1,5):1 (3,5):3 (3,8):1 (4,9):1 (6,7):1 (7,8):2 (8,9):3 [19]
```

Part 3

Let $G = (V, E)$ be an unweighted undirected graph with $V = \{0, 1, 2, \ldots, n-1\}$. G may be unconnected. Your task is to connect G by adding the minimum number of edges. You should minimize the total cost of edge addition in the following sense. If $(u, v) \in E$, then u and v are already adjacent, so $\text{cost}(u, v) = 0$. Otherwise, a positive cost is attached to the (non-existing) edge (u, v) (here u and v are not adjacent, but may be connected by a path). The file **D23in.txt** stores data for the missing edges. The file starts with the number n of vertices in G. This is followed by a pair of triples (u, v, c), one in each line, where $(u, v) \notin E$ with $u < v$, and c is the cost of adding the edge (u, v). Notice that if $(u, v) \in E$, it does not appear in the input file. So G would consist of all the (undirected) edges (u, v) not listed in the input file.

Compute and print the connected components of G. If the number of components is 1, there is nothing to be done. Otherwise, proceed as follows.

Build the component graph $C(G)$ of G. The vertices of $C(G)$ are the components of G. For two vertices U, V of $C(G)$, the cost of the edge (U,V) is the minimum of all cost(u,v), where $u \in U$ and $v \in V$. $C(G)$ is thus a weighted complete undirected graph. Find a minimum spanning tree of $C(G)$. The edges of the MST prescribe which edges to add to G in order to make it connected at a minimum cost. Add these edges to G, and check whether the augmented G is connected.

In this part too, you may use the **`NetworkX`** library.

Sample output

The output for a graph on 32 nodes is given below. The input file is huge, and is not reproduced here. You can download the file from the lab page.

```
+++ Components in G
    [0, 3, 12, 13, 15, 17, 20, 22, 30]
    [1, 4, 6, 9, 11, 18, 19, 26]
    [2, 5, 8, 23, 25, 27, 28]
    [7]
    [10]
    [14]
    [16]
    [21]
    [24]
    [29]
    [31]

+++ Connecting G
    Adding edge (22,24) of cost 10
    Adding edge (12,18) of cost 10
    Adding edge (13,25) of cost 10
    Adding edge (20,21) of cost 14
    Adding edge (1,7) of cost 13
    Adding edge (1,10) of cost 11
    Adding edge (6,14) of cost 11
    Adding edge (2,31) of cost 13
    Adding edge (8,16) of cost 12
    Adding edge (21,29) of cost 10
--- Total connection cost = 114

+++ G is now connected
```

Submit three separate Python source files, one for each part.