Preliminary for the next topic: Priority queue

(*This part should not take more than 45 minutes, preferably 15 minutes*)
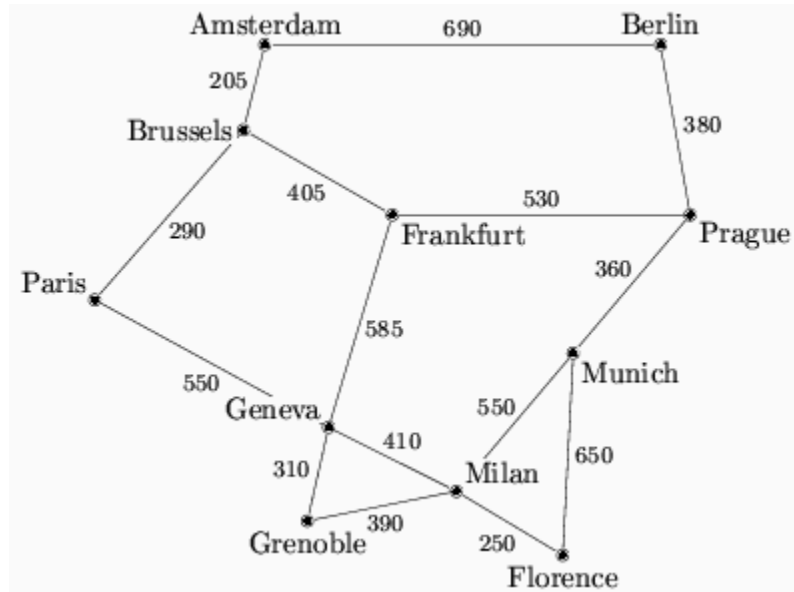
1. Download the simplePriorityQueue.py code from Class Materials

2. Learn how to use the Simple_Priority_Queue class. Examples are already given in the code (commented). To import Simple_Priority_Queue to your program, place simplePrioirtyQueue.py in the same folder as your program, the add the following line.

   from simplePriorityQueue import Simple_Priority_Queue

NOTE:

   This priority queue is simply the *heap* data structure in which you are supposed to have studied from Data Structure and Algorithms course. Therefore, you are assumed to have adequate familiarity with it.

PROBLEM A: SHORTEST ROUTE ON MAP

Find the shortest path from a city to a destination e.g. from Amsterdam to Florence.

INPUT:
- The first line is the number of cities, n.
- Given that cities are represented by numbers, starting from 0, each following line of the input file gives information of an edge. For example, a line containing 5 3 305 means that there is an edge from city 5 to city 3 (and vice versa) with the distance 305 km.

OUTPUT: One number, the shortest total route from city 0 to city n-1.

Test cases: Example_maps.zip (already in the Class Materials)

1) Use the following code to read the input map file, which will store the map as a collection of adjacency lists.

```python
from sys import stdin

INF = 1000000000
V = int(input())
adj = [[] for i in range(V)]

for line in stdin:
    x = line.split()
    u = int(x[0])
    v = int(x[1])
    w = int(x[2])
    adj[u].append((v,w))
    adj[v].append((u,w))
```
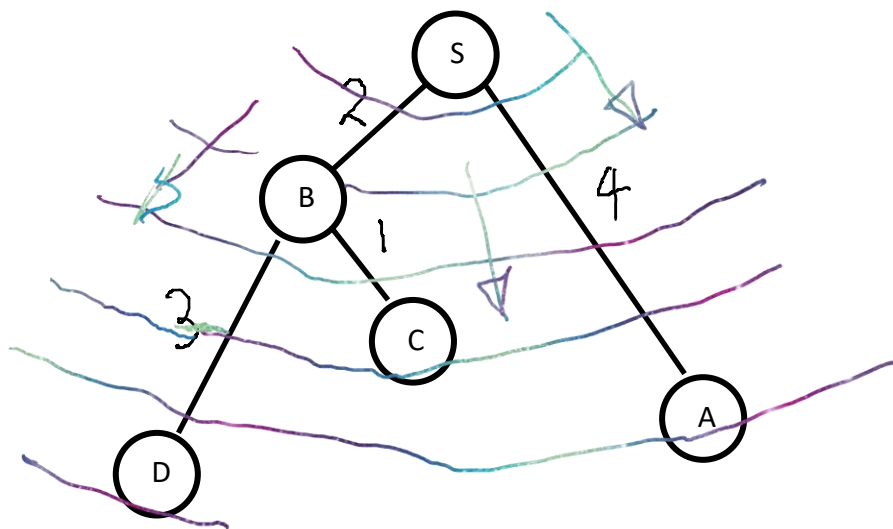
KEEP IN MIND that a map is a *weighted* graph. Therefore, the information stored in the adjacency lists is not only adjacencies, but also the weight of each adjacency (in this case, the distance).

2) The technique to solve map problem is a generalization of Breadth-First Search (BFS), called ***Uniform Cost Search (UCS).***
BFS is applicable, for finding solution in minimal number of steps, only when every search to reach successor state is of <u>equal cost</u>. For example, in maze, every search to the adjacent position takes 1 step, equally.
When each search does not necessary cost equally, UCS is an optimized form of continuous BFS. UCS expands the search outward from the state with smaller *total* cost to the state with larger *total* cost.

In this example, if the search expands uniformly from S, state B will be reached first (cost = 2), then C (cost = 3, from S), A (cost = 4), D (cost = 5, from S), respectively.

There is no need to sweepingly scan the search outward, because the cost of reaching states A, B, C, and D, can establish the order of states to be searched.

Essentially, UCS is simply BFS that operates on priority-queue instead of First-In-First-Out queue. The key of each state is the total cost of reaching that state, from the initial state. Consequently, the priority-queue will always dequeue the state whose total cost is the smallest.

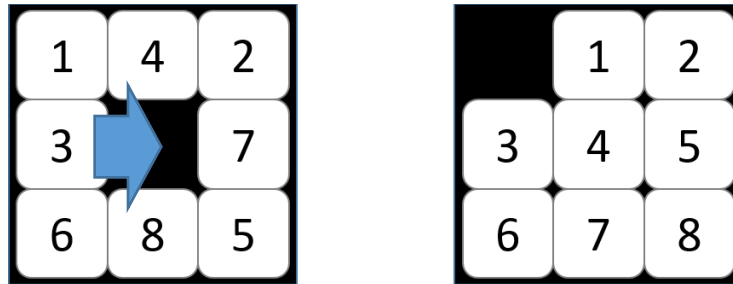Write a program that solves the "shortest route on map" problem using UCS technique.

Hint The successor states of state u are those indicated in adj[u].

Alternative explanation of UCS (a lecturing video):
https://www.youtube.com/watch?v=dRMvK76xQJI

PROBLEM B: 8-PUZZLE

Targeted algorithmic technique: **Iterative Deepening A-star (IDA*)**

INPUT: A permutation of {0, .. , 8} arranged in 3x3 format. 0 represents the hole.

OUTPUT: The minimum number of moves required to reach the goal state.

Supplemental materials:

- Iterative Deepening Search.pptx (lecture note)
- Class lecture

1) Study the provided Depth-First-Search code (a brute-force recursive technique) for 8-puzzle in 8puzzle_IDS_template.py. It is critical to understand the role of the ***depth limit*** for DFS function.

2) Complete the IDS function. Test your complete program against the provided test cases.

3) Study the provided Iterative Deepening A-star code on the following issues:
   - What is the heuristic function ? (A-star requires it for a determining lowerbound)
   - How the goal state test is implicitly determined.
   - How the maximum searching depth is set, compared to the way that IDS uses.
   - Why is a boolean variable "found" required for recursion termination, compared to the way IDS uses (which is testing if the return value == -1) ?

4) Complete the IDAstar function. Test your complete program against the provided test cases.