Grocery Store

by Peng Cao

codility
WE TEST CODERS

It's time to show you how the Codility Challenge codenamed (Hydrogenium) can be solved. You can still give it a try, but no certificate will be granted. The problem asks for the shortest path in a weighted graph.

Golden solution $O(n^2)$

We need to reach a grocery store as soon as possible. We start from square 0, so it is a standard single-source shortest path problem. We can use the famous Dijkstra's algorithm to find the shortest path from square 0 to every other square. Because the Dijkstra's algorithm finds the shortest path in terms of length, we can finish our solution as soon as we reach the first open store. Note that the graph is undirected.

1: Golden solution — $O((m + n) \log n)$.

```
def grocery_store(A, B, C, D):
    M = len(A)
    N = len(D)
    # Build the graph
    G = [[]] * N
    for i in xrange(M):
        G[A[i]] = G[A[i]] + [(B[i], C[i])]
        G[B[i]] = G[B[i]] + [(A[i], C[i])]
    # Initialize the queue and distance table
    dist = [-1] * N
    Q = PriorityQueue()
    Q.put((0, 0))
    # Search
    while not Q.empty():
        (s, i) = Q.get()
        if dist[i] == -1:
            dist[i] = s
            if s <= D[i]:
                return s
            for (j, t) in G[i]:
                Q.put((s + t, j))
    return -1
```

The total time complexity is $O((m+n) \log n)$ due to the time required for the priority queue. We can improve above solution to $O(m + n \log n)$ if we use Fibonacci heap. The maximal value of m equals n^2 so the time complexity is $O(n^2 \log n)$ or $O(n^2)$ in case of Fibonacci heap.

We can also n times look for the closest square, calculating the shortest distance to it:

2: Golden solution — $O(n^2)$.

```
def grocery_store(A, B, C, D):
    MAX_INT = 10**9
    M = len(A)
    N = len(D)
    # Build the graph
    G = [[]] * N
    for i in xrange(M):
        G[A[i]] = G[A[i]] + [(B[i], C[i])]
        G[B[i]] = G[B[i]] + [(A[i], C[i])]
    # Initialize the distance table
    dist = [MAX_INT] * N
    visit = [False] * N
    dist[0] = 0
    # Look for minimum value
    for k in xrange(N):
        # Find the minimum
        s = MAX_INT
        for j in xrange(N):
            if dist[j] < s and visit[j] == False:
                s = dist[j]
                i = j
        visit[i] = True
        if s <= D[i]:
            return s
        for (j, t) in G[i]:
            dist[j] = min(dist[j], s + t)
    return -1
```

The time complexity of above solution is $O(n^2)$.