CS201c: Solutions to Programming Evaluation 5

Finding the maximum success probability path (without any constraint on number of red edges).

First, consider the problem of finding the path from A to B with maximum success probability. (We want to find such a path without giving any consideration to the number of red edges on the path.)

For an edge e of graph G, define its ``length" l(e) as

$$l(e) = - \log (1-p(e))$$

Note that $0 < 1-p(e) < 1$, and hence l(e) is always a positive number.

Suppose a path P in G consists of the sequence of edges e_1, e_2, …, e_z.
The length of path l(P) is defined as the real number l(e_1)+l(e_2)+...+l(e_z).

Note the following relation between success probability and length of path P:

$$- \log (s(P)) = l(P)$$

Thus, a path from A to B having maximum success probability is the same as the minimum length path from A to B.

Thus, one can find maximum success probability path from A to B by running Dijsktra's algorithm from A, with respect to the lengths l(e) of edges e in E. The running time is O(|E| log(|V|)), if we use a binary heap.

Note. You can use "math.h" to compute logarithm of a real number.

Incorporating the constraint on red edges used.

We will use the wave propagation analogy.

Each wave arrival event has four parameters *[t, nr, w, v]*, where

(i) *t* is the time of wave arrival at the other node (this parameter is the same as that for Dijkstra's algorithm),
(ii) *nr* is the number of red edges used by the wave packet at the arrival time (this is the new parameter added to incorporate red edges), and

(iii) w is the sending node and v is the receiving node.

We maintain a binary heap H consisting of all current wave arrival events. The heap is ordered by the wave arrival times (i.e., the first parameter). If two wave arrival events have the same wave arrival time, ties can be broken arbitrarily.

For each node v, we maintain an array B_v[0...k] of k+1 floats. For 0<=i<=k, B[i] is the earliest time at which a wave that passes through *exactly* i red edges ends up arriving at node v.

Initially, all entries in arrays B_v[0...k], where v is not equal to A, are set to infinity.
For v=s, B_v[0]=0 and B_v[z]=infinity for 1<=z<=k.

The receiving and transmitting algorithm for wave packets is as follows:

```
Let W=(t, nr, w, v) be the wave event obtained by calling deleteMin() on heap H.
If B_v[nr] is not equal to infinity:
        Do nothing.
else (if B_v[nr] is equal to infinity):
        Set B_v[nr]=t
        For each edge (v, y) in E:
                if (v,y) is blue:
                        Insert a wave arrival event [t+l(v,y), nr, v, y] into heap H.
                else (if (v,y) is red):
                        If (nr > 0):
                                Insert a wave arrival event [t+l(v,y), nr-1, v, y] into heap H.
```

Output.

```
Let t be the destination node.
For i in [0...k]:
        If B_t[i] <= -log (g):
                Output `Yes' and halt.
Output 'No' and halt.
```

Running time.

Total number of insert operations in heap H:

```
        At most (k+1)*|Adj(v)| insert operation for each node v.
        Thus, the total number of insert operations is at most: 2*(k+1)*|E|.
        Total time taken by insert operations is O ( (k+1)*|E|*log(|V|) )
```

Total number of deleteMin operations in heap H <= number of insert operations.

```
        So, the total time taken by deleteMin() operations is O ( (k+1)*|E|*log(|V|) )
```

Finally, you go through the adjacency list Adj(v) of every vertex v at most (k+1) times.
Therefore, the time taken for this is O((k+1)*|E|)

Thus, total running time of the above algorithm is O ((k+1)*|E|*log(|V|)).

Proof of correctness.

Let 0<=q<=k.

Let L*(v,q) be the minimum length of a path from A to v with exactly q red edges. Then, we have that, when the heap H becomes empty:

Lemma 1. B_v[q] <= L*(v,q).

Lemma 2. L*(v,q) <= B_v[q]

Prove these two lemmas yourself.