1 function Rumburak(Graph, source):

2 create vertex priority queue Q

3

4 dist[source] ← 0 // Initialization

5 Q.add\_with\_priority(source, 0) // associated priority equals dist[·]

6

7 for each vertex v in Graph.Vertices:

8 if v ≠ source

9 prev[v] ← UNDEFINED // Predecessor of v

10 dist[v] ← INFINITY // Unknown distance from source to v

11 Q.add\_with\_priority(v, INFINITY)

12

13

14 while Q is not empty: // The main loop

15 u ← Q.extract\_min() // Remove and return best vertex

16 for each neighbor v of u: // Go through all v neighbors of u

17 alt ← dist[u] + Graph.Edges(u, v)

18 if alt < dist[v]:

19 prev[v] ← u

20 dist[v] ← alt

21 Q.decrease\_priority(v, alt)

22

23 return dist, prev

1 – funkce – O(1)

2 – vytvoreni priority queue – O(1)

3 –

4 – hodnota – O(1)

5 – zarazeni do priority queue O(log n) (z wikipedie)

6 –

7 – pro každý … - O(n) (iterujeme přes n)

8 – funkce if O(1)

9 – nastaveni hodnoty O(1)

10 – nastaveni hodnoty O(1)

11 – pridani do priority queue O(log n)

12 –

13 –

14 – while loop O(n)

15 – odebrani prvku O(log n) (pracujeme s priority queue )

16 – pro sousedy – O(n^2) (nejhorsi pripad je každý soused)

17 – nova hodnota O(1)

18 - funkce if O(1)

19 – nova honota O(1)

20 – nova hodnota O(1)

21 – snizeni v priority queue O(log n)

22 –

23 – konec funkce

V první smycce (do 11) mame casovou slozitost O(n log n) v druhé O(n log n) pro body 14, 15 a potom O(n^2 log n) pro zbytek druhé smycky, celkove O(n log n + n^2 log n ) tedy O(n^2 log n) coz je vyssi nez ta první, proto celkove to bude O(n^2 log n)