R-7.5

In order to compute $D^k$, we only use the matrix $D^{k-1}$. Therefore, we can reuse "old" matrices (we don’t need them anymore). This allows us to only ever have two matrices for $D^k$, reducing the memory to $O(n^2)$.

R-7.7

Initial graph; colors represent different clouds
after processing \((a, b)\)

after processing \((a, f)\)
after processing \((b, c)\)

after processing \((b, g)\)

3
after processing \((c, d)\)

after processing \((c, h)\)
after processing \((a, d)\)

after processing \((d, e)\)
We are running Dijkstra’s algorithm on a graph $G$ with $\Omega(n^2)$ edges. Unsorted sequence implementation of the priority queue $Q$ allows for $O(1)$ time for key lookup and decrease (if, as previously, for each vertex we store a pointer that points to its position in the priority queue), and $O(n)$ for selection of an element with minimum key from a queue of $n$ items.

The algorithm starts with $n$ vertices in the queue, and each time a vertex is to be taken out, it takes time proportional to the size of the queue, and reduces size of the queue by 1. Therefore, the complexity of selection of the next vertex throughout the whole execution of the algorithm is:

$$n + (n - 1) + (n - 2) + \cdots + 2 + 1 = \Theta(n^2)$$

Since we have to check (and possibly decrease) key for every edge, and a single operation takes time $O(1)$, total time spent here is $\Theta(m) = \Omega(n^2)$.

Overall, time complexity is $\Theta(n^2) + \Omega(n^2) = \Theta(n^2)$.

Use the slightly modified DAG-based algorithm, that computes maximum (instead of minimum for the standard algorithm) distance of vertices from $s$. Return label of $t$.

Analysis: uses adjacency list to represent the graph and an additional list of size $O(n)$ to store topologically sorted vertices of the graph.

Time: $O(n + m)$.

Run a modified version of Dijkstra’s algorithm that label every vertex of the graph with the maximum bandwidth from $a$ to them. Modifications to the algorithm:

1. Initially set $d(a) \leftarrow +\infty$, and $d(v) \leftarrow 0$ for all $v \neq a$.
2. Always select the vertex with maximum label to add to the cloud.
3. Relaxing edge $e = (u, v)$ with bandwidth $b$:

   $$d(v) \leftarrow \max(d(v), \min(d(u), b))$$

At the end of the algorithm, $d(b)$ is the maximum bandwidth between $a$ and $b$. We can get the path that achieves this bandwidth, if for every vertex we stored the edge, which caused us to set its current label.

Time: same as Dijkstra’s algorithm: $O((n + m) \log n)$ for the standard implementation, $O(m + n \log n)$ when using Fibonacci heaps.