CS 344: Dijkstra-Shortest-Path Algorithm

The following algorithm finds the shortest-path from a distinguished vertex s (source) to all other vertices.

DIJKSTRA-SHORTEST-PATH $(G = (V, E), w, s)$;
for all $v \in V$ **do** $d(v) := \infty$, $\pi(v) := nil$;
$d(s) := 0$;
$S := \emptyset$;
while $V \neq \emptyset$
  do $u :=$ Extract-Min;
    $S = S \cup \{u\}$;
    for all $v \in Adjacent(u)$ **do**
      if $d(v) > d(u) + w(u, v)$ **then** $d(v) := d(u) + w(u, v)$, $\pi(v) := u$;

Remarks. For a given vertex v, the number $d(v)$ will represent the cost of the shortest path from s to v. It is initialized to be infinity for all vertices different than s. The array π is the predecessor array. $\pi(v)$ gives the vertex that must be visited immediately before v, on the shortest-path from s to v. The procedure Extract-Min finds a vertex with smallest $d(v)$ value. It then removes this vertex from V. Adjacent(u) is the set of vertices adjacent to u. Each execution of Extract-Min takes $O(|V|)$ time. Since there are $|V|$ calls to Extract-Min, the total complexity of Dijkstra's shortest-path algorithm is $O(|V|^2)$.

If we use a binary heap as a priority queue to implement Dijkstra's algorithm then the Extract-Min operation finds the minimum element on a heap which can be done in constant time (if key values at children of a node are greater than or equal to the key value of the node itself). However, to remove the element and to update the heap again would take $O(\log |V|)$ operations (by swapping the element and the last element of the heap). Each time we adjust a d value of a node, by decreasing it, in order to update the queue it would take $O(\log |V|)$ time. This is because we would have to compare the key value of the node with that of its parent and if necessary to swap them. We can refer to this operation as Decrease-Key. Thus the total time complexity of $O((|V| + |E|) \log |V|)$.