Algorithm 1: Computing earliest-arrival time

Input : A temporal graph $G = (V, E)$ in its edge stream representation, source vertex x, time interval $[t_\alpha, t_\omega]$
Output : The earliest-arrival time from x to every vertex $v \in V$ within $[t_\alpha, t_\omega]$

1 Initialize $t[x] = t_\alpha$, and $t[v] = \infty$ for all $v \in V \setminus \{x\}$;
2 **foreach** *incoming edge* $e = (u, v, t, \lambda)$ *in the edge stream* **do**
3   **if** $t + \lambda \le t_\omega$ *and* $t \ge t[u]$ **then**
4   **if** $t + \lambda < t[v]$ **then**
5             $t[v] \leftarrow t + \lambda$;
6   **else if** $t \ge t_\omega$ **then**
7   Break the for-loop and go to Line 8;
8 **return** $t[v]$ for each $v \in V$;

The classic Dijkstra's algorithm for computing single-source shortest paths is based on the fact that the prefix-subpath of a shortest path is also a shortest path. However, according to Lemma 1, the prefix-subpath of an earliest-arrival path may not be an earliest-arrival path. This seems to imply that the greedy strategy to grow the shortest paths that is applied in Dijkstra's algorithm cannot be applied to compute earliest-arrival paths, though the following observation shows otherwise.

LEMMA 6. *Let $\mathbf{P}$ be the set of earliest-arrival paths from x to a vertex v_k within the time interval $[t_\alpha, t_\omega]$. If $\mathbf{P} \neq \emptyset$, then there exists $P = \langle x, v_1, v_2, \ldots, v_k \rangle \in \mathbf{P}$ such that every prefix-subpath, $P_i = \langle x, v_1, v_2, \ldots, v_i \rangle$, is an earliest-arrival path from x to v_i within $[t_\alpha, t_\omega]$, for $1 \le i \le k$.*

PROOF. Given any earliest-arrival path $P \in \mathbf{P}$, if not every prefix-subpath in it is an earliest-arrival path, we can always construct a path $\hat{P}$ as follows. We traverse P in reverse order and find the first vertex v_i such that the corresponding prefix-subpath P_i is not an earliest-arrival path from x to v_i. Thus, there exists another path $\hat{P}_i$ that is an earliest-arrival path from x to v_i. We replace P_i in P by $\hat{P}_i$. The new path $\hat{P}$ is still a valid temporal path because $end(\hat{P}_i) < end(P_i)$. In addition, $\hat{P}$ is an earliest-arrival path from x to v_k (i.e., $\hat{P} \in \mathbf{P}$) because $end(\hat{P}) = end(P)$. This process continues until every prefix-subpath is an earliest-arrival path and the resulting $\hat{P}$ is in $\mathbf{P}$, which proves the lemma. □

Based on Lemma 6, we can apply the greedy strategy to grow the earliest-arrival paths in a similar way to Dijkstra's algorithm. However, this approach needs to use a minimum priority queue, resulting in an algorithm with $O(m \log \pi + m \log n)$ time and $O(M+n)$ space complexity [19], which is too expensive for processing temporal graphs with a large number of temporal edges.

Dijkstra's greedy strategy requires the entire graph to be present as random access to vertices and edges are needed. However, for temporal graphs, Lemma 5 implies that the input graph can be in the natural edge stream representation, and it is possible to compute the earliest-arrival paths with only one scan of the graph. We present our one-pass algorithm in Algorithm 1 and elaborate as follows.

We use an array $t[v]$ to keep the current earliest-arrival time from x to every vertex $v \in V$ that has been seen in the stream. According to Lemma 5, if there is a temporal path P from x to v so that all edges on P have been seen in the stream, then $t[v] = end(P) = t + \lambda$ as updated in Line 5. The condition "$t + \lambda < t[v]$" in Line 4 ensures that $t[v]$ will be updated with the smallest $end(P)$ for any P from x to v within the time interval $[t_\alpha, t_\omega]$.

We linearly scan G and for each incoming edge $e = (u, v, t, \lambda)$ in the stream, we check whether e meets the time constraint of a temporal path within $[t_\alpha, t_\omega]$, i.e., whether $t + \lambda \le t_\omega$ and $t \ge t[u]$. If yes, we grow the temporal path by extending to v via the edge e. During the process, we update $t[v]$ when necessary as discussed earlier. The process terminates when we meet the first edge in the stream that has starting time greater than or equal to t_ω (Lines 6-7).

EXAMPLE 3. *Consider the temporal graph G in Figure 1(a), where we assume that the traversal time λ is 1 for all edges. Let a be the source vertex. We compute the earliest-arrival time from a to every vertex in G within the time interval $[1, 4]$.*

Initially, $t[a] = 1$, and $t[v] = \infty$ for all $v \in V \setminus \{a\}$. The first incoming edge is $(a, b, 1, 1)$, since it satisfies the conditions in Lines 3-4, we update $t[b] = 1 + 1 = 2$ in Line 5. The second edge is $(a, b, 2, 1)$, the condition in Line 4 is not satisfied. The next edge is $(g, j, 2, 1)$, since $t[g] = \infty$, the condition "$t \ge t[u] = t[g]$" in Line 3 is not met. Then, the edges $(b, g, 3, 1)$, $(b, h, 3, 1)$, and $(a, f, 3, 1)$ are followed, for which we update $t[g] = 4$, $t[h] = 4$, and $t[f] = 4$. After that the edge $(a, c, 4, 1)$ comes, which satisfies the condition in Line 6 and the process is terminated. It can be easily verified that we have obtained the correct earliest-arrival time from a to every vertex in G within the time interval $[1, 4]$. ■

The following lemma shows that when Algorithm 1 terminates, $t[v]$ correctly reports the earliest-arrival time from x to v.

LEMMA 7. *For any vertex $v \in V$, if the earliest-arrival path from x to v within the time interval $[t_\alpha, t_\omega]$ exists, then $t[v]$ returned by Algorithm 1 is the corresponding earliest-arrival time; otherwise, $t[v] = \infty$.*

PROOF. Suppose that the earliest-arrival path from x to v within $[t_\alpha, t_\omega]$ exists. Then, according to Lemma 6, there exists an earliest-arrival path from x to v, $P = \langle x = v_1, v_2, \ldots, v_k, v_{k+1} = v \rangle$, such that every prefix-subpath of P is an earliest-arrival path from x to some vertex v_i on P. Let $t_e[v_i]$ be the earliest-arrival time from x to v_i, for $1 \le i \le k+1$. Let $e_1, e_2, \ldots, e_k$ be the edges on P, where $e_i = (v_i, v_{i+1}, t_i, \lambda_i)$ for $1 \le i \le k$. Then, we have $t_i \ge t_e[v_i]$ and $t_i + \lambda_i = t_e[v_{i+1}]$ for $1 \le i \le k$.

We prove that Algorithm 1 computes $t[v_i] = t_e[v_i]$, for $1 \le i \le k+1$, by induction on i. When $i = 1$, $x = v_1$, $t[x] = t_e[x] = t_\alpha$ is initialized in Line 1 of Algorithm 1, and $t[x]$ will not be updated any more. When $i = 2$, obviously we have $t[v_2] = t_e[v_2] = t_1 + \lambda_1$ when we process e_1, and $t[v_2]$ will not be updated again due to the condition in Line 4. Now assume that for $i = j$, where $j < k+1$, $t[v_j] = t_e[v_j] = t_{j-1} + \lambda_{j-1}$ when we process e_{j-1}. Consider $i = j+1$ and we want to prove $t[v_{j+1}] = t_e[v_{j+1}]$. According to Lemma 5, e_j comes after e_{j-1} in the stream. Thus, when the algorithm scans e_j, we have the following two cases regarding the value of $t[v_{j+1}]$. (1) $t[v_{j+1}] = t_e[v_{j+1}]$. In this case, Line 5 will not be processed due to the condition in Line 4 and $t[v_{j+1}]$ gives the correct earliest-arrival time from x to v_{j+1}. (2) $t[v_{j+1}] > t_e[v_{j+1}]$. In this case, $t[v_{j+1}]$ is updated to $t_e[v_{j+1}] = t_j + \lambda_j$ in Line 5, and it will not be updated again due to the condition in Line 4. In both cases, we have $t[v_{j+1}] = t_e[v_{j+1}]$ and by induction, $t[v_i] = t_e[v_i]$ for $1 \le i \le k+1$.

Finally, if the earliest-arrival path from x to v does not exist, then there is no temporal path from x to v and $t[v]$ remains to be ∞. □

The following theorem states our main result for earliest-arrival path computation.

THEOREM 1. *Algorithm 1 correctly computes the earliest-arrival time from a source vertex x to every vertex $v \in V$ within the time interval $[t_\alpha, t_\omega]$ using only one linear scan of the graph, $O(n+M)$ time and $O(n)$ space.*

PROOF. The correctness is proved in Lemma 7. The initialization in Line 1 takes $O(n)$ time. Every temporal edge in G is