**Algorithm 1:** `fill_runs_slice(T, t, from, to, v)`

**Input**: Tree, `T`, supporting `WeinerLink` and `Parent`; Query string `t`; Interval over `t`, [`from`, `to`); Starting node `v`.

**Output**: Tripple (`first`, `last`, `v`): `first` index of the first failing `WeinerLink` call; `last` index last failing `WeinerLink` call; `v` the node corresponding to `last`;

```
1  currnode ← T.WeinerLink(T.root(), t[to − 1]);
2  idx_set ← FALSE;
3  k ← to − 2;
4  repeat
5  |  c ← t[k − 1];
6  |  if T.IsRoot(T.WeinerLink(currnode, t[to − 1])) then
7  |  |  if idx_set = FALSE then
8  |  |  |  first ← k;
9  |  |  |  idx_set ← TRUE;
10 |  |  end
11 |  |  runs[k] ← 0;
12 |  |  repeat
13 |  |  |  currnode ← T.parent(v);
14 |  |  until T.IsRoot(T.WeinerLink(currnode, c));
15 |  |  v ← T.WeinerLink(currnode, c);
16 |  |  last ← k;
17 |  else
18 |  |  runs[k] ← 1
19 |  end
20 |  currnode ← T.WeinerLink(currnode, c);
21 |  k ← k − 1;
22 until k > from;
```

**Algorithm 2:** Build `RUNS` in parallel.

**Input**: Tree, `T`, supporting `WeinerLink` and `Parent`; Query string `t`; Number of threads `nthreads` dividing |`t`|.

```
1 S ← |t|/nthreads;
2 R ← [1..nthreads];
3 for i ← 1 to nthreads do
4 |  R[i] ← fill_runs_slice(T, t, (i − 1) ∗ S, i ∗ S);
5 end
6 for i ← 1 to nthreads − 1 do
7 |  fill_runs_slice(T, t, T[i − 1].first, R[i].second, R[i].third);
8 end
```

**Algorithm 3:** `fill_ms_slice`

**Input**: Cst of `s`, `T`, supporting `WeinerLink` and `Parent`; Query string `t`; Vector `ms` initialized at 0; Vector `runs`; Interval over `t`, [`from`, `to`)

**Output**: Last index `j` of `ms`

```
1  k ← from, i ← 0, h* ← k + 1;
2  c ← t[k];
3  v ← T.root();
4  repeat
5  |  h ← h*;
6  |  u ← v;
7  |  repeat
8  |  |  c ← t[h*];
9  |  |  u ← T.WeinerLink(v, c);
10 |  |  if !T.IsRoot(u) then
11 |  |  |  v ← u;
12 |  |  |  h* ← h* + 1;
13 |  |  end
14 |  until !T.IsRoot(u) and h* < |t|;
15 |  // resize if needed;
16 |  i ← i − h* + 1;
17 |  ms[i++] ← 1;
18 |  if h* < |t| then
19 |  |  repeat
20 |  |  |  v ← T.Parent(v);
21 |  |  |  u ← T.WeinerLink(v, c);
22 |  |  until !T.IsRoot(u);
23 |  |  h* ← h* + 1;
24 |  end
25 |  k' ← k + 1;
26 |  repeat
27 |  |  k' ← k' + 1;
28 |  until k' < |t| and runs[k'] ≠ 0;
29 |  // resize if needed;
30 |  ms[k + 1..k' − 1] ← 1;
31 |  j ← max(k' − 1, to);
32 until k < to;
```

**Algorithm 4:** Build `RUNS` in parallel.

**Input**: Tree, `T`, supporting `WeinerLink` and `Parent`; Query string `t`; Number of threads `nthreads` dividing $|\mathtt{t}|$.

```
1 for i ← 1 to nthreads do
2 |  ms_i ← [1..(2 * |t|)/nthreads];
3 end
4 for i ← 1 to nthreads do
5 |  J_i ← fill_ms_slice(T, t, ms_i, runs, (i − 1) * S, i * S);
6 end
7 for i ← 1 to nthreads − 1 do
8 |  ms_i ← resize_to(J_i);
9 end
```