Algorithm 1 dijkstra GPU SSSP

Input: $G(V, E)$, source vertex s;
Output: $dist(v)$, $(v \in V)$, the weight of the shortest path from s to v;

```
1:
2: function initial(s, V)
3:     for each v ∈ V do
4:         dist(v) ← +∞;                          ▷ initialize dist to positive infinity;
5:     end for
6:     dist(s) ← 0;                               ▷ set the source distence to 0;
7: end function
8:
9: function dijkstraCudaFunc(G(V,E), dist, predist)   ▷ G(V,E), the initially distance array dist, a
   temporary distance array predist;
10:    u0 ← threadId;                             ▷ get the thread id;
11:    offset ← blockDim;                         ▷ get the number of threads in a block;
12:    flag ← (__shared__ memory) 1;              ▷ whether the dist has changed;
13:
14:    while true do
15:        if flag = 0 then
16:            break;
17:        end if
18:        flag ← 0;
19:
20:        u ← u0;
21:        while u<|V| do
22:            if dist(u) is not modified then
23:                continue;                      ▷ this IF can speedup;
24:            end if
25:            for each (u, v, w) ∈ E do
26:                atomicMin(&predist(v), dist(u) + w);   ▷ use the atomic opt to exclusive mutually;
27:            end for
28:            u ← (u + offset);                  ▷ reuse the thread;
29:        end while
30:
31:        __syncthreads();                       ▷ synchronize all threads in the same block;
32:
33:        u ← u0;
34:        while u<|V| do
35:            if predist(u)<dist(u) then
36:                dist(u) ← predist(u);
37:                flag ← 1;                      ▷ some vexter is updated;
38:            end if
39:            u ← (u + offset);
40:        end while
41:
42:        __syncthreads();                       ▷ synchronize all threads in the same block;
43:
44:    end while
45: end function
```

46:

47: $initial(s, V)$;

48:

49: $host\_to\_device(dist), host\_to\_device(G(V, E))$; ▷ copy the $dist$ and $G(V, E)$ from main memory to GPU memory;

50:

51: $dijkstraCudaFunc()$; ▷ call the CUDA kernal;

52: $device\_to\_host(dist)$; ▷ copy the $dist$ back;

53:

54: **return** $result$