Algorithm 1 Insertion tache

```
1: function ADD_TASK(du_t, cpu_t)                          ▷ duration, number of cpu
2:     start_key ← [0, du_t, cpu_t]
3:     end_key ← [+∞, +∞, +∞]
4:     profile_tree → node_loop(start_key, end_key, {
5:         starting_time_min ← (freespace → {starting_time})
6:         processor_range_t ← (freespace → {cpu})
7:         return 1
8:     })
9:                                          ▷ Remove the number of cpu for the range
10:    processor_range_t ← (processor_range_t → reduce_to_basic(cpu_t))
11:    start_key ← [0, 0, 0]
12:    end_key ← [starting_time_min + du_t, +∞, +∞]
13:    profile_tree → node_loop(start_key, end_key, {          ▷ Research with contraints
14:        push freespace_impacted, freespace
15:        return 0
16:    })
17:    for freespace ← freespace_impacted do
18:        cut_freespace(freespace, starting_time_min, du_t, processor_range_t)
19:    end for
20: end function
```

Algorithm 2 Decoupage Freespace

```
1: function CUT_FREESPACE(freespace, start_time, duration, processor_range)
2:     if Intersection(freespace → {cpu}, processor_range) > 0 then
3:         profile_tree → remove(freespace)
4:         if freespace → {starting_time} < start_time then
5:             new freespaceright_freespace ← (freespace → {starting_time}, start_time − freespace → {starting_time}, freespace → {cpu})
6:             profile_tree → add(right_freespace)
7:         end if
8:         if freespace → {starting_time} < (start_time + duration) then
9:             new freespaceleft_freespace ← (freespace → {starting_time}, start_time − freespace → {starting_time}, freespace → {cpu})
10:            profile_tree → add(left_freespace)
11:        end if
12:        range_test ← (freespace → {cpu} → copy())
13:        range_test → remove(processor_range)
14:        if range_test → size() > 0 then
15:            new freespacenew_freespace ← (freespace → {starting_time}, freespace → {duration}, range_test)
16:            profile_tree → add(new_freespace)
17:        end if
18:    end if
19: end function
```

Algorithm 3 Suppression tache

```
1: function REMOVE_TASK(st_t, d_t, cpu_t)          ▷ Starting time, duration, cpu range
2:     et_t ← (st_t + d_t)                          ▷ End time of job
3:     start_key ← [0, 0, 0]
4:     end_key ← [st_t + d_t, +∞, +∞]
5:     profile_tree → node_loop(start_key, end_key, {
6:     push freespace_impacted, freespace
7:     st_t2 ← (freespace → {starting_time})
8:     et_t2 ← (freespace → {starting_time} + freespace → {duration})
9:     if (st_t ≤ st_t2 and st_t2 ≤ et_t) then
10:        direction{freespace} ← −1
11:    else if (st_t2 ≤ st_t and st_t ≤ et_t2) then
12:        direction{freespace} ← 1
13:    end if
14:    if exist direction{freespace} then
15:        for t ← st_t2 to et_t2 do
16:            if exist tab{t} then
17:                add tab{t} ← (freespace → {cpu} inter tab{t})
18:            else
19:                add tab{t} ← (freespace → {cpu})
20:            end if
21:        end for
22:    end if
23:    }
24:    for freespace ← freespace_impacted do
25:        extend_freespace(freespace)
26:    end for
27: end function
```

Algorithm 4 Augmentation Freespace

```
1: function EXTEND_FREESPACE(freespace)
2:     if direction{freespace} = 1 then
3:         t_next ← (freespace → {starting_time} + freespace → {duration})
4:     else
5:         t_next ← freespace → {starting_time}
6:     end if
7:     new_t ← t_next
8:     while (exist tab{new_t} and (tab{new_t} inter freespace → {cpu}) ≥ freespace → {cpu} → size()) do
9:         t_max ← new_t
10:        new_t ← (new_t + direction{freespace})
11:    end while
12:    if direction{freespace} = 1 then
13:        freespace → {duration} ← (t_next − freespace → {starting_time})
14:    else
15:        freespace → {duration} ← (freespace → {starting_time} − new_t + freespace → {duration})
16:        freespace → {starting_time} ← new_t
17:    end if
18:    TODO :VERIFIER SI L'ESPACE N'EST PAS COMPRIS DANS UN AUTRE
19: end function
```