**Algorithm 1** Insertion tache

**function** ADD_TASK($du\_t, cpu\_t$) ▷ duration, number of cpu

On recherche le premier emplacement possible pour la tache avec au minimum la durée $du\_t$ et le nombre de cpu $cpu\_t$.

On récupère le temps de départ $starting\_time\_min$ et la range de cpu $processor\_range\_t$ de l'emplacement.
On réduit la range de cpu $processor\_range\_t$ du nombre de cpu $cpu\_t$.
On récupère la liste de tous les emplacements potentiellement impactés : ceux dont le temps de départ est inférieur ou égale au temps de départ min $starting\_time\_min$ + la durée $du\_t$ .

On ne garde que ceux pour qui :
    l'intersection de la range de cpu avec $processor\_range\_t$ est non vide
**and**
    dont le temps de départ + la durée $\geq starting\_time\_min$

$tab\_resultat \leftarrow$ vide

**for** tous les emplacements restant **do**
    On supprime l'emplacement de l'arbre des emplacements
    $tab\_resultat = tab\_resultat \cup$ **cut_freespace**($emplacement, starting\_time\_min, processor\_range\_t$)
**end for**

$tab\_final \leftarrow$ vide

**for** chaque élément de $tab\_resultat$ **do**
    **if is_necessary_freespace**($element, tab\_resultat$) **then**
        $tab\_final = tab\_final \cup$ élément
    **end if**
**end for**

**for** chaque élément de $tab\_final$ **do**
    Ajouter à l'arbre des emplacements $element$
**end for**

**end function**

**Algorithm 2** Decoupage Freespace

1: **function** CUT_FREESPACE($freespace, start\_time, duration, processor\_range$)
2: $new\_emplacement \leftarrow$ vide
3:
4: **if** temps de départ de $freespace < start\_time$ **then**
5: On crée un nouvel emplacement $left\_freespace$ avec :
6: Temps de départ : celui de $freespace$
7: Durée : $start\_time$ - temps de départ de $freespace$
8: Cpu : Cpu de $freespace$
9: $new\_emplacement \leftarrow new\_emplacement \cup left\_freespace$
10: **end if**
11:
12: **if** l'intersection de la range de cpu de $freespace$ et $processor\_range <$ au nombre de processeurs de $freespace$ **then**
13: On crée un nouvel emplacement $new\_freespace$ avec :
14: Temps de départ : celui de $freespace$
15: Durée : durée de $freespace$
16: Cpu : différence entre la range cpu de $freespace$ et $processor\_range$
17: $new\_emplacement \leftarrow new\_emplacement \cup new\_freespace$
18: **end if**
19:
20: On crée un nouvel emplacement $right\_freespace$ avec :
21: Temps de départ : $(start\_time + duration)$
22: Durée : durée de $freespace$ - (temps de départ de $freespace$ - $start\_time + (start\_time + duration)$)
23: Cpu : Cpu de $freespace$
24: $new\_emplacement \leftarrow new\_emplacement \cup right\_freespace$
25:
26: **return** $new\_emplacement$
27: **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)
3:
4:     On récupère la liste de tous les emplacements qui ont un temps de départ inférieur à (st_t + d_t)
5:
6:     On ne garde que ceux qui :
7:         existent dans l'intervalle de temps st_t à et_t
8:
9:     for tous les emplacements restant do
10:        On supprime l'emplacement de l'arbre des emplacements
11:    end for
12:
13:    tab_resultat ← vide
14:
15:    tab_resultat =extend_freespace(liste des emplacements + task)
16:
17:    tab_final ← vide
18:
19:    for chaque élément de tab_resultat do
20:        if is_necessary_freespace(element, tab_resultat) then
21:            tab_final = tab_final ∪ élément
22:        end if
23:    end for
24:
25:    for chaque élément de tab_final do
26:        Ajouter à l'arbre des emplacements element
27:    end for
28:
29: end function
```

**Algorithm 4** Augmentation Freespace

```
1: function EXTEND_FREESPACE(freespaces)
2:     new_emplacements ← vide
3:     events ← vide
4:
5:     ▷ events est une table de hachage de temps
6:     ▷ chaque temps a deux listes de freespace, 'start' et 'end'
7:
8:     for chaque éléments f de freespaces do
9:         Ajoute à events au temps de départ de f, f dans la liste "start"
10:        Ajoute à events au temps final de f, f dans la liste "end"
11:    end for
12:
13:    for chaque éléments de events do
14:        Ajoute élément à liste_events
15:    end for
16:
17:    On trie par ordre croissant liste_events
18:    temps_init ← premier temps de liste_events
19:    cpu_init ← premier range cpu de liste_events
20:
21:    for chaque éléments temps de liste_events do
22:        cpu ← cpu_init
23:
24:        for chaque éléments f de ev.temps.start do
25:            cpu ← cpu ∪ cpu de f
26:        end for
27:
28:        for chaque éléments f de ev.temps.end do
29:            cpu ← cpu − cpu de f
30:        end for
31:
32:        if cpu_init ≠ cpu and temps_init ≠ temps then
33:            On crée un nouvel emplacement new_freespace avec :
34:                Temps de départ : temps_init
35:                Durée : temps − temps_init
36:                Cpu : cpu_init
37:            new_emplacement ← new_emplacement ∪ new_freespace
38:            temps_init ← temps
39:        end if
40:        cpu_init ← cpu
41:    end for
42:
43:    return new_emplacements
44: end function
```

**Algorithm 5** Suppression des Freespaces inutiles

```
1: function IS_NECESSARY_FREESPACE(freespace, freespace_list)
2:
3:     for tous les éléments de freespace_list do
4:         if éléments != freespace then
5:             if temps de départ de freespace ≥ temps de départ de space and temps final de freespace ≤ temps
   final de space then
6:                 if (range cpu de freespace ∩ range cpu de space) = nombre de cpu de freespace then
7:                     return 0
8:                 end if
9:             end if
10:        end if
11:    end for
12:    return 1
13: end function
```