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

function CUT_FREESPACE($freespace, start\_time, duration, processor\_range$)
  $new\_emplacement \leftarrow$ vide

  if temps de départ de $freespace < start\_time$ **then**
    On crée un nouvel emplacement $left\_freespace$ avec :
      Temps de départ : celui de $freespace$
      Durée : $start\_time$ - temps de départ de $freespace$
      Cpu : Cpu de $freespace$
    $new\_emplacement \leftarrow new\_emplacement \cup left\_freespace$
  end if

  if l'intersection de la range de cpu de $freespace$ et $processor\_range <$ au nombre de processeurs de $freespace$ **then**
    On crée un nouvel emplacement $new\_freespace$ avec :
      Temps de départ : celui de $freespace$
      Durée : durée de $freespace$
      Cpu : différence entre la range cpu de $freespace$ et $processor\_range$
    $new\_emplacement \leftarrow new\_emplacement \cup new\_freespace$
  end if

  On crée un nouvel emplacement $right\_freespace$ avec :
    Temps de départ : $(start\_time + duration)$
    Durée : durée de $freespace$ - (temps de départ de $freespace$ - $start\_time + (start\_time + duration)$)
    Cpu : Cpu de $freespace$
  $new\_emplacement \leftarrow new\_emplacement \cup right\_freespace$

  return $new\_emplacement$
end function

Algorithm 3 Suppression tache

function REMOVE_TASK(st_t, d_t, cpu_t) ▷ Starting time, duration, cpu range
  $et\_t \leftarrow (st\_t + d\_t)$

  On récupère la liste de tous les emplacements qui ont un temps de départ inférieur à $(st\_t + d\_t)$

  On ne garde que ceux qui :
    existent dans l'intervalle de temps st_t à et_t

  $tab\_resultat \leftarrow$ vide

  $tab\_resultat = tab\_resultat \cup$ **extend_freespace**(liste des emplacements + task)

  $tab\_final \leftarrow$ 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 4 Augmentation Freespace

```
1: function EXTEND_FREESPACE(freespaces)
2:     new_emplacements ← vide
3:     events ← vide
4:
5:     for chaque éléments de freespaces do
6:         Ajoute à events au temps de départ de éléments dans "start" l'élément en question.
7:         Ajoute à events au temps final de éléments dans "end" l'élément en question.
8:     end for
9:
10:    for chaque éléments de events do
11:        Ajoute élément à liste_events
12:    end for
13:
14:    On trie par ordre croissant liste_events
15:    temps_init ← premier temps de liste_events
16:    cpu_init ← vide
17:
18:    for chaque éléments de liste_events do
19:        temps ← temps de élément
20:        if éléments est de type "start" then
21:            cpu ← cpu ∪ cpu de élément
22:        else if éléments est de type "end" then
23:            cpu ← cpu − cpu de élément
24:        end if
25:        if cpu_init ≠ cpu and temps_init ≠ temps then
26:            On crée un nouvel emplacement new_freespace avec :
27:                Temps de départ : temps_init
28:                Durée : temps − temps_init
29:                Cpu : cpu_init
30:            new_emplacement ← new_emplacement ∪ new_freespace
31:            temps_init ← temps
32:        end if
33:        cpu_init ← cpu
34:    end for
35:
36:    return new_emplacements
37: 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
```