
Algorithm 1 Insertion tache

```
1: function ADD_TASK(du_t, cpu_t) ▷ duration, number of cpu
2:
3:   On recherche le premier emplacement possible pour la tache avec au minimum la durée du_t et le nombre
   de cpu cpu_t.
4:
5:   On récupère le temps de départ starting_time_min et la range de cpu processor_range_t de l'emplace-
   ment.
6:   On réduit la range de cpu processor_range_t du nombre de cpu cpu_t.
7:   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 .
8:
9:   On ne garde que ceux pour qui :
10:     l'intersection de la range de cpu avec processor_range_t est non vide
11:   and
12:     dont le temps de départ + la durée  $\geq$  starting_time_min
13:
14:   tab_resultat  $\leftarrow$  vide
15:
16:   for tous les emplacements restant do
17:     On supprime l'emplacement de l'arbre des emplacements
18:     tab_resultat = tab_resultat  $\cup$  cut_freespace(emplacement, starting_time_min, processor_range_t)
19:   end for
20:
21:   tab_final  $\leftarrow$  vide
22:
23:   for chaque élément de tab_resultat do
24:     if is_necessary_freespace(element, tab_resultat) then
25:       tab_final = tab_final  $\cup$  élément
26:     end if
27:   end for
28:
29:   for chaque élément de tab_final do
30:     Ajouter à l'arbre des emplacements element
31:   end for
32:
33: 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:    $et\_t \leftarrow (st\_t + d\_t)$ 
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:    $tab\_resultat \leftarrow$  vide
10:
11:   for tous les emplacements restant do
12:      $tab\_resultat = tab\_resultat \cup \text{extend\_freespace}(emplacement)$ 
13:   end for
14:
15:    $tab\_final \leftarrow$  vide
16:
17:   for chaque élément de  $tab\_resultat$  do
18:     if is_necessary_freespace( $element, tab\_resultat$ ) then
19:        $tab\_final = tab\_final \cup$  élément
20:     end if
21:   end for
22:
23:   for chaque élément de  $tab\_final$  do
24:     Ajouter à l'arbre des emplacements  $element$ 
25:   end for
26:
27: end function
```

Algorithm 4 Augmentation Freespace

```
1: function EXTEND_FREESPACE(freespace, task)
2:   new_emplacements  $\leftarrow$  vide
3:
4:   if (temps de départ de freespace = temps final de task) or (temps de départ de task = temps final de
   freespace) then
5:     if les cpu de freespace et task sont les mêmes then
6:       On modifie l'emplacement freespace avec :
7:       Temps de départ : temps de départ min(task, freespace)
8:       Durée : Durée de task + durée de freespace
9:       Cpu : Cpu de freespace
10:      new_emplacements  $\leftarrow$  new_emplacements  $\cup$  freespace
11:    else
12:      On crée un nouvel emplacement new_freespace avec :
13:      Temps de départ : temps de départ min(task, freespace)
14:      Durée : Durée de task + durée de freespace
15:      Cpu : Cpu de freespace  $\cap$  cpu de task
16:      new_emplacements  $\leftarrow$  new_emplacements  $\cup$  new_freespace  $\cup$  freespace
17:    end if
18:  else
19:    On crée un nouvel emplacement new_freespace avec :
20:    Temps de départ : temps de départ max(freespace, task)
21:    Durée : temps de départ max(freespace, task) - temps final min(freespace, task)
22:    Cpu : Cpu de freespace  $\cup$  cpu de task
23:    new_emplacements  $\leftarrow$  new_emplacements  $\cup$  new_freespace  $\cup$  freespace
24:  end if
25:
26:  return new_emplacements
27: 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  $\neq$  freespace then
5:       if temps de départ de freespace  $\geq$  temps de départ de space and temps final de freespace  $\leq$  temps
       final de space then
6:         if (range cpu de freespace  $\cap$  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
```
