

Chapitre 1

Retrait de sommets d'un DST (suite des travaux)

1.1 Présentation de différents cas de figure

1.1.1 Retraits avec fusion

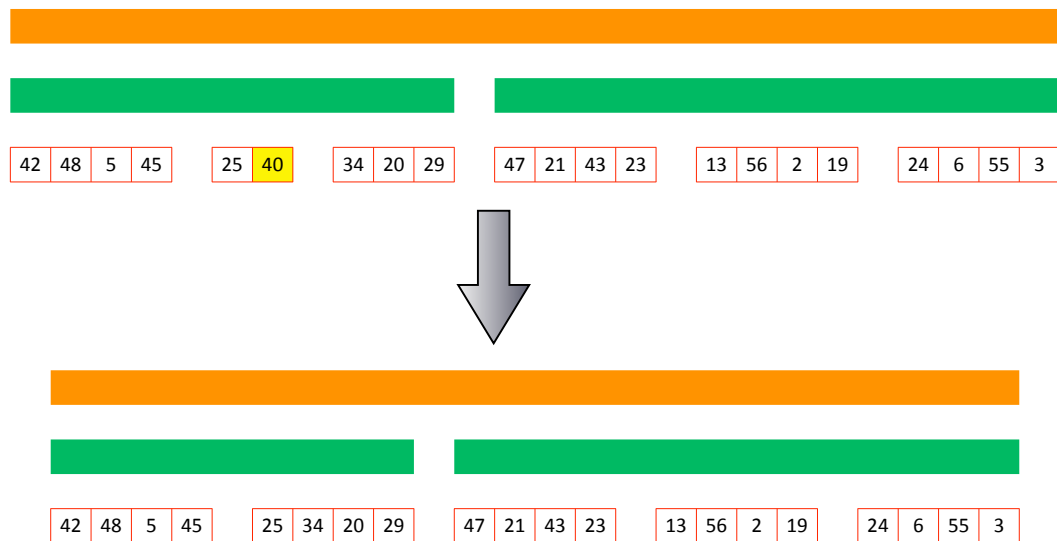


FIGURE 1.1 – Exemple de retrait : le sommet 40 quitte un DST [2,4]

Ici, le départ de 40 laisse un nœud orphelin, le 25, que le groupe à sa droite peut accueillir.

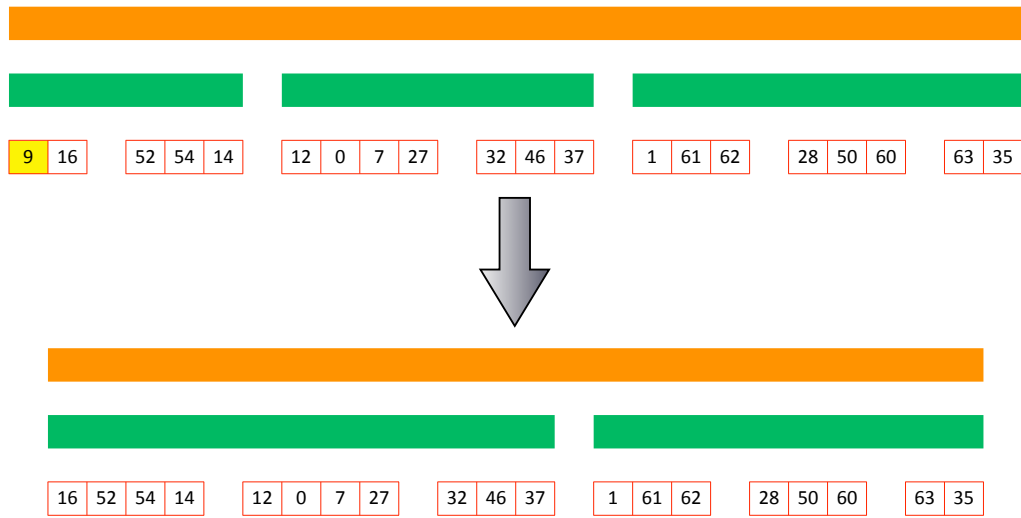


FIGURE 1.2 – Exemple de retrait : le sommet 9 quitte un DST $[2,4]$

Cette fois, la fusion de 16 avec le groupe $[52 \dots 14]$ provoque la fusion de l'étage supérieur (vert).

1.2 Retraits avec transfert

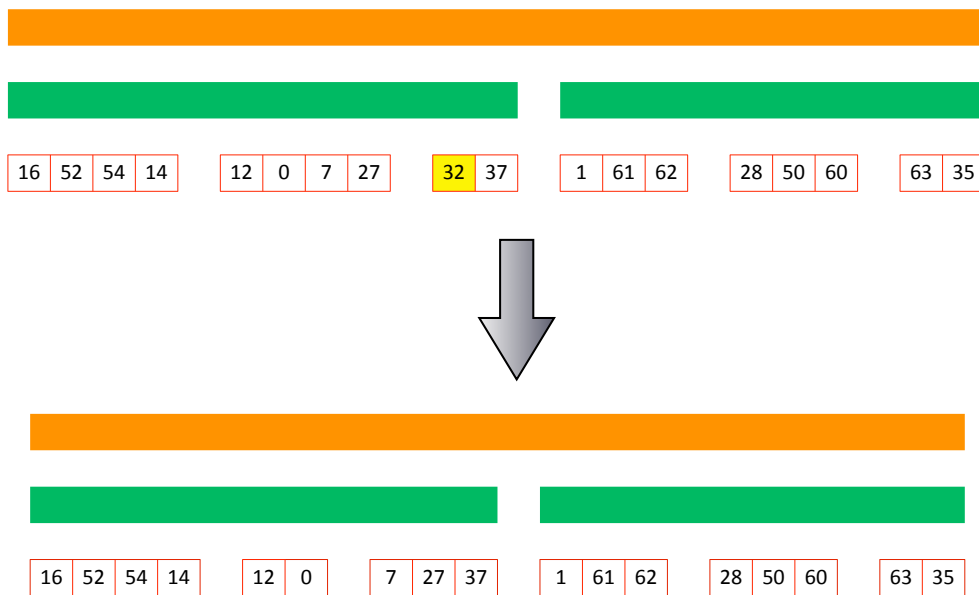


FIGURE 1.3 – Exemple de retrait : le sommet 32 quitte un DST $[2,4]$

Les groupes $[16 \dots 14]$ et $[12 \dots 27]$ n'ont pas de place pour accueillir 37. Il va donc y avoir un transfert de $[7, 27]$ vers 37.

Algorithme 1 : Procédure exécutée par un nœud qui quitte le DST

```
1: procedure LEAVE( )
    ▷ travaille sur des copies des tables de routage
2:   cpy_brothers  $\leftarrow$  me.brothers
3:   cpy_preds  $\leftarrow$  me.preds
4:   for stage  $\leftarrow$  0 to height(me.brothers) - 1 do
    ▷ INFORME MES PRÉDÉCESSEURS DE MON DÉPART
5:     for pred  $\leftarrow$  0 to size(cpy_preds[stage]) - 1 do
6:       if (cpy_preds[stage][pred].id  $\neq$  me.id) then
7:         if (stage = 0) then
8:           SEND_MSG_ASYNC(cpy_preds[stage][pred].id, del_bro(stage, me.id))
9:         else
10:            new_rep_id  $\leftarrow$  un de mes frères de l'étage 0 choisi aléatoirement
            ▷ en informe le prédécesseur
11:            if (new_rep_id  $\neq$  me.id) then
12:              SEND_MSG_ASYNC(cpy_preds[stage][pred].id,
13:                             repl_bro(stage, new_rep_id))
14:            end if
15:          end if
16:        end for
        ▷ prédécesseur suivant
    ▷ INFORME MES FRÈRES DE MON DÉPART
17:    for brother  $\leftarrow$  0 to size(cpy_brothers[stage]) - 1 do
18:      if (cpy_brothers[stage][brother].id  $\neq$  me.id) then
19:        SEND_MSG_ASYNC(cpy_brothers[stage][brother].id,
20:                       del_pred(stage, me.id))
21:      end if
22:    end for
    ▷ frère suivant
    ▷ étage suivant
    ▷ CHARGE UN DE MES FRÈRES DE TRAITER LES FUSIONS OU TRANSFERTS
23:    if (size(me.brothers[0])  $\leq$  a) then
24:      idx  $\leftarrow$  index d'un de mes frères de l'étage 0
25:      SEND_MSG_SYNC(me.brothers[0][idx].id, merge_req())
26:    end if
27: end procedure
```

Algorithme 2 : Transfert de nœuds du groupe courant vers un groupe appelant

```
1: procedure TRANSFERT(st, right, cut_pos)
2:   if (right = 1) then
3:     start  $\leftarrow$  cut_pos
4:     end  $\leftarrow$  size(me.brothers[stage]) - 1
5:     answer.stay_id  $\leftarrow$  me.brothers[stage][cut_pos - 1].id
6:   else
7:     start  $\leftarrow$  0
8:     end  $\leftarrow$  cut_pos
9:     answer.stay_id  $\leftarrow$  me.brothers[st][cut_pos + 1].id
10:  end if

11:  for i  $\leftarrow$  start to end do
12:    answer.rep_array[i - start] = me.brothers[st][i]
13:  end for

14:  BROADCAST(me, st, cut_node(st, right, cut_pos))

15:  return answer
16: end procedure
```

Algorithme 3 : Scinde un nœud lors d'un transfert

```
1: procedure CUT_NODE(stage, right, cut_pos)
2:   pos_me  $\leftarrow$  index(me.brothers[stage], me.id)

3:   if (right = 0) then
4:     start  $\leftarrow$  0
5:     end  $\leftarrow$  cut_pos
6:     new_node  $\leftarrow$  me.brothers[stage][cut_pos + 1]
7:   else
8:     start  $\leftarrow$  cut_pos
9:     end  $\leftarrow$  size(me.brothers[stage]) - 1
10:    new_node  $\leftarrow$  me.brothers[stage][0]
11:   end if

12:   if (pos_me  $\geq$  start and pos_me  $\leq$  end) then
13:     SHIFT_BRO(stage + 1, new_node, right)
14:   end if

15:   if (right = 0) then
16:     if (pos_me  $\leq$  cut_pos) then
17:       start  $\leftarrow$  cut_pos + 1
18:       end  $\leftarrow$  size(me.brothers[stage] - 1)
19:     else
20:       start  $\leftarrow$  0
21:       end  $\leftarrow$  cut_pos
22:     end if
23:   else
24:     if (pos_me < cut_pos) then
25:       start  $\leftarrow$  cut_pos
26:       end  $\leftarrow$  size(me.brothers[stage] - 1)
27:     else
28:       start  $\leftarrow$  0
29:       end  $\leftarrow$  cut_pos - 1
30:     end if
31:   end if

32:   DEL_MEMBER(stage, start, end)
33: end procedure
```

Algorithme 4 : Supprime une partie du groupe courant à un étage donné

```
1: procedure DEL_MEMBER(stage, start, end)
2:   nb_del  $\leftarrow$  end - start + 1
3:   if (nb_del = 0) then
4:     return
5:   end if                                     ▷ mémorise les nœuds à effacer avant de commencer

6:   for i  $\leftarrow$  0 to nb_del - 1 do
7:     id_del[i]  $\leftarrow$  me.brothers[stage][start + i].id
8:   end for                                     ▷ boucle d'effacement

9:   for i  $\leftarrow$  0 to nb_del - 1 do
10:    if (id_del[i]  $\neq$  me.id) then             ▷ ne pas effacer 'moi'
11:      pos2del  $\leftarrow$  index(me.brothers[stage], id_del[i])

12:      if (pos2del < size(me.brothers[stage] - 1)) then
13:        for j  $\leftarrow$  pos2del to size(me.brothers[stage] - 1) do
14:          me.brothers[stage][j]  $\leftarrow$  me.brothers[stage][j + 1]
15:        end for
16:      else
17:        j  $\leftarrow$  pos2del
18:      end if

19:      me.brothers[stage][j].id  $\leftarrow$  -1
20:      SEND_MSG_ASYNC(id_del[i], del_pred(stage, me.id))
21:    end if                                     ▷ 'moi' n'est plus prédécesseur de id_del[i]
22:  end for
23: end procedure
```

Algorithme 5 : Décale les membres du groupe pour en accueillir un nouveau à la position de 'moi'. Le membre en trop est détruit.

```

1: procedure SHIFT_BRO(stage, new_node, right)
    ▷ s'assure de n'exécuter cette fonction qu'une fois (elle est diffusée par cut_node)
2:   pos_new_node  $\leftarrow$  index(me.brothers[stage], new_node.id)
3:   if (pos_new_node > -1) then
4:     return
5:   end if

6:   pos_me  $\leftarrow$  index(me.brothers[stage], me.id)

7:   if (right = 1) then
8:     if (pos_me < b) then
9:       lost_id  $\leftarrow$  me.brothers[stage][pos_me + 1].id
10:      me.brothers[stage][pos_me + 1]  $\leftarrow$  me.brothers[stage][pos_me]
11:    else
12:      Affiche un message d'erreur et stoppe la procédure
13:    end if
14:  else
15:    if (pos_me > 0) then
16:      lost_id  $\leftarrow$  me.brothers[stage][pos_me - 1].id
17:      me.brothers[stage][pos_me - 1]  $\leftarrow$  me.brothers[stage][pos_me]
18:    else
19:      Affiche un message d'erreur et stoppe la procédure
20:    end if
21:  end if

22:  me.brothers[stage][pos_me]  $\leftarrow$  new_node
    ▷ 'moi' doit être un prédécesseur de new_node

23:  SEND_MSG_ASYNC(new_node.id, add_pred(stage, me.id))
    ▷ 'moi' ne doit plus être un prédécesseur de lost_id

24:  SEND_MSG_ASYNC(lost_id, del_pred(stage, me.id))
25: end procedure

```

Algorithme 6 : Traite les fusions ou transferts consécutifs à un départ

```
1: procedure MERGE_REQUEST( )

2:   stage  $\leftarrow$  0
3:   size_last_stage  $\leftarrow$  0
4:   while (size(me.brothers[stage]) < a) and stage < height(me.brothers) - 1 do
5:     pos_contact  $\leftarrow$  merge_or_transfer(me, stage)
6:     if (pos_contact > -1) then
7:       transfer  $\leftarrow$  0
8:       pos_me  $\leftarrow$  index(stage + 1, me.id)
9:       if (pos_me > pos_contact) then
10:        right  $\leftarrow$  11
11:      else
12:        right  $\leftarrow$  10
13:      end if
14:      SEND_MSG_SYNC(me.brothers[stage + 1][pos_contact].id,
        merge(stage, pos_me, pos_contact, right, me.brothers[stage]))
15:      if (pos_me > pos_contact) then
16:        right  $\leftarrow$  1
17:      else
18:        right  $\leftarrow$  0
19:      end if
20:      SEND_MSG_SYNC(me.brothers[stage + 1][pos_contact].id,
        broadcast_merge(stage, pos_me, pos_contact, right, me.brothers[stage]))
21:
22:      CLEAN_UPPER_STAGE(stage, pos_me, pos_contact)
23:      BROADCAST(me, stage+1, clean_upper_stage(stage, pos_me, pos_contact))
24:    else
```

Algorithme 6 : MERGE_REQUEST (partie 2)

```
25:                                     ▷ FUSION IMPOSSIBLE - IL FAUT FAIRE UN TRANSFERT

26:      $pos\_me\_up \leftarrow \text{index}(stage + 1, me.id)$ 

27:     if ( $pos\_me\_up = 0$ ) then

28:          $pos\_contact \leftarrow 1$ 
29:          $right \leftarrow 0$ 
30:          $cut\_pos \leftarrow b - a - 1$ 
31:     else

32:          $pos\_contact \leftarrow pos\_me\_up - 1$ 
33:          $right \leftarrow 1$ 
34:          $cut\_pos \leftarrow a$ 
35:     end if

36:      $contact\_id \leftarrow me.brothers[stage + 1][pos\_contact].id$ 

37:      $answer \leftarrow \text{SEND\_MSG\_SYNC}(contact\_id,$ 
         $\quad \text{transfer}(stage, right, cut\_pos, me.id))$ 

38:                                     ▷ AJOUTE LES NŒUDS REÇUS

39:      $current\_bro \leftarrow me.brothers[stage]$                                      ▷ sauvegarde l'étage courant

40:                                     ▷ chaque membre de la branche courante ajoute les nœuds reçus
41:      $\text{BR\_ADD\_BRO\_ARRAY}(stage, answer.rep\_array, \text{mod}((right + 1), 2))$ 

42:                                     ▷ chaque membre de la branche voisine ajoute les nœuds courants
43:      $\text{SEND\_MSG\_SYNC}(answer.rep\_array[0].id,$ 
         $\quad \text{br\_add\_bro\_array}(stage, current\_bro, right))$ 

44:                                     ▷ nettoyage de l'étage supérieur
45:                                     ▷ exécution locale
46:      $\text{UPDATE\_UPPER\_STAGE}(stage, pos\_contact, answer.stay\_id)$ 

47:                                     ▷ diffusion
48:      $\text{BROADCAST}(me, stage + 1,$ 
         $\quad \text{update\_upper\_stage}(stage, pos\_contact, answer.stay\_id))$ 
49:     end if

50:      $stage \leftarrow stage + 1$                                      ▷ étage suivant
51: end while
```

Algorithme 6 : MERGE_REQUEST (partie 3)

52: ▷ TRAITEMENT DE LA RACINE

53: $i \leftarrow 0$

54: **while** ($me.brothers[0][i].id = me.id$) **do**

55: $i \leftarrow i + 1$

56: **end while**

57: $size_last_stage \leftarrow SEND_MSG_SYNC(me.brothers[0][i].id,$
 $get_size(height(me.brothers) - 1))$

58: **if** ($size_last_stage = 1$) **then** ▷ détruit la racine

59: BROADCAST($me, height(me.brothers) - 1, del_root(height(me.brothers))$)

60: **end if**

61: **end procedure**

Algorithme 7 : regarde s'il faut faire une fusion ou un transfert. Retourne la position d'un contact pour la fusion ou -1 si tranfert

1: **procedure** MERGE_OR_TRANSFERT($stage$)

2: $idx_bro \leftarrow 0$

3: $merge \leftarrow 0$

4: **while** ($merge = 0$ **and** $idx_bro < size(me.brothers[stage + 1])$) **do**

5: **if** ($me.brothers[stage + 1][idx_bro].id \neq me.id$) **then**

6: $size \leftarrow SEND_MSG_SYNC(me.brothers[stage + 1][idx_bro].id,$
 $get_size(stage))$

7: **if** ($size \leq b - size(me.brothers[stage])$) **then**

8: $merge \leftarrow 1$

9: **end if**

10: **end if**

11: $idx_bro \leftarrow idx_bro + 1$

12: **end while**

13: **if** ($merge = 1$) **then**

14: **return** $idx_bro - 1$

15: **else**

16: **return** -1

17: **end if**

18: **end procedure**

Algorithme 8 : fusionne des nœuds “orphelins” (*source*) au groupe courant (*cible*)

```
1: procedure MERGE(nodes_array, nodes_array_size, stage, pos_me, pos_contact, right)

2:   if (size(nodes_array) = size(me.brothers[stage])) then                                ▷ déjà fait
3:     return
4:   end if

5:   if (size(me.brothers[stage]) < a) then      ▷ change le sens d'arrivée si 'moi' est dans la source
6:     right ← mod((right + 1), 2)
7:   end if

8:   loc_right ← 0

9:   switch (right)
10:    case (0) :                                ▷ les nouveaux nœuds viennent de la gauche - diffusion
11:      if (index(stage + 1, me.id) = 0) then
12:        loc_right ← 1
13:      else
14:        loc_right ← 0
15:      end if
16:    break

17:    case (1) :                                ▷ les nouveaux nœuds viennent de la droite - diffusion
18:      if (index(stage + 1, me.id) = size(me.brothers[stage + 1] - 1)) then
19:        loc_right ← 0
20:      else
21:        loc_right ← 1
22:      end if
23:    break

24:    case (10) :                               ▷ les nouveaux nœuds viennent de la gauche - 1ère fusion
25:      loc_right ← 0
26:    break

27:    case (11) :                               ▷ les nouveaux nœuds viennent de la droite - 1ère fusion
28:      loc_right ← 1
29:    break
30:  end switch

31:                                          ▷ nombre de nœuds à incorporer
32:  if (nodes_array_size ≥ size(me.brothers[stage])) then
33:    loc_nodes_array_size ← nodes_array_size - size(me.brothers[stage])
34:  else
35:    loc_nodes_array_size ← nodes_array_size
36:  end if
```

Algorithme 8 : MERGE (partie 2)

```
37:   if (loc_right = 0) then                                ▷ prend la partie gauche de la liste des nœuds fournie
38:       if (loc_nodes_array_size > 0) then
39:           for i ← 0 to loc_nodes_array_size − 1 do
40:               loc_nodes_array[i] ← nodes_array[i]
41:           end for
42:       end if
43:   else                                                    ▷ prend la partie droite de la liste des nœuds fournie
44:       if (loc_nodes_array_size > 0) then
45:           for i ← (nodes_array_size − loc_nodes_array_size) to (nodes_array_size − 1) do
46:               loc_nodes_array[i − (nodes_array_size − loc_nodes_array_size)] ← nodes_array[i]
47:           end for
48:       end if
49:   end if

50:   if (loc_nodes_array_size > 0) then
51:       if (loc_right = 0) then                            ▷ insère les nouveaux frères au début (gauche)

52:           for i ← loc_nodes_array_size − 1 to 0 do
53:               INSERT_BRO(stage, loc_nodes_array[i])
54:               SEND_MSG_ASYNC(loc_nodes_array[i], add_pred(stage, me.id))
55:           end for
56:       else                                                ▷ ajoute les nouveaux frères à la fin (droite)

57:           for i ← 0 to loc_nodes_array_size − 1 do
58:               ADD_BROTHER(stage, loc_nodes_array[i])
59:               SEND_MSG_ASYNC(loc_nodes_array[i], add_pred(stage, me.id))
60:           end for
61:       end if
62:   end if
63: end procedure
```
