Rapport OCaml

GONZALEZ Mathieu & HUBERT Gustav

Mai 2019

Table des matières

1 Tris de listes : résultats

1.1 Tri par partition fusion

```
#partitionne [1];;
   int list = ([1], [])

#partitionne [1;2];;
   int list = ([1], [2])

#partitionne [1;2;3;4;5];;
   int list = ([5; 3; 1], [4; 2])

#fusionne (<=) [1;3;5;7;9] [2;4;6;8];;
   int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]

#fusionne (>=) [99;55;33;11] [666;222;44;0];;
   int list = [666; 222; 99; 55; 44; 33; 11; 0]

#fusionne (>=) [99;55;33;11;-9;-22;-55] [666;222;44;0;-44;-222];;
   int list = [666; 222; 99; 55; 44; 33; 11; 0; -9; -22;
               -44; -55; -222]

#tri_partition_fusion (<=) [55;33;99;77;12;0;48;96;3;2];;
   int list = [0; 2; 3; 12; 33; 48; 55; 77; 96; 99]

#tri_partition_fusion (>=) (random_list 100 10);;
   int list = [85; 79; 58; 58; 49; 39; 22; 17; 11; 6]
```

```
#tri_partition_fusion (>=) (random_list 10000 100);;
   int list = [9937; 9892; 9665; 9663; 9579; 9565; 9548;
               9523; 9250; 9083; 8927; 8904; 8834; 8772;
               8570; 8557; 8546; 8515; 8418; 8392; 8138;
               8069; 8050; 7940; 7910; 7894; 7809; 7598;
               7578; 7433; 7241; 7120; 7085; 7071; 7054;
               7042; 7014; 7001; 6992; 6778; 6722; 6721;
               6646; 6553; 6515; 6290; 6192; 6177; 6149;
               5875; 5818; 5581; 5560; 5515; 5484; 5415;
               5396; 5257; 5054; 4818; 4811; 4673; 4454;
               4364; 4131; 3987; 3883; 3873; 3782; 3659;
               3608; 3589; 3586; 3498; 3474; 3414; 3133;
               2995; 2812; 2696; 2603; 2493; 2379; 2044;
               1841; 1839; 1808; 1716; 1619; 1489; 1373;
               1344; 809; 738; 336; 302; 259; 115; 113;
               106]

#tri_partition_fusion (>=) (random_list 100 50);;
   int list = [97; 97; 96; 94; 90; 88; 85; 81; 78; 78; 76;
               74; 74; 70; 68; 62; 58; 58; 55; 54; 46; 42;
               42; 40; 36; 35; 34; 31; 29; 28; 27; 27; 24;
               20; 19; 19; 17; 17; 16; 14; 13; 12; 11; 10;
               9; 6; 5; 3; 3; 0]

#tri_partition_fusion (>=) (random_list 10 50);;
   int list = [9; 9; 9; 9; 9; 9; 8; 8; 8; 8; 8; 8; 7; 7; 7;
               7; 7; 6; 5; 5; 5; 5; 5; 5; 4; 4; 4; 3; 3; 3;
               3; 3; 3; 3; 3; 3; 2; 2; 2; 2; 2; 2; 1; 1; 1;
               0; 0; 0; 0; 0]
```

1.2 Tri pivot

```
#partitionne_pivot (>=) [8; 0; 3; 1; 7; 34; 87; 11; 4; 199] 8;;
   int list = ([199; 11; 87; 34; 8], [4; 7; 1; 3; 0])

#tri_pivot (<=) (random_list 50 10);;
   int list = [0; 13; 16; 29; 29; 35; 37; 37; 41; 45]

#tri_pivot (<=) (random_list 1000 100);;
   int list = [15; 33; 46; 48; 53; 63; 81; 85; 108; 130;
               144; 159; 162; 177; 223; 228; 228; 238; 254;
               258; 262; 264; 265; 274; 278; 307; 307; 319;
               323; 328; 336; 337; 337; 339; 350; 352; 435;
               449; 466; 471; 472; 474; 476; 484; 505; 510;
               531; 535; 565; 571; 580; 580; 582; 600; 601;
               619; 620; 621; 630; 631; 634; 637; 638; 639;
               644; 662; 669; 675; 684; 694; 694; 705; 711;
               727; 727; 740; 741; 759; 794; 817; 835; 843;
               874; 876; 887; 888; 913; 919; 927; 934; 936;
               939; 955; 963; 972; 973; 987; 987; 996; 998]

#tri_pivot (<=) (random_list 100 50);;
   int list = [0; 2; 3; 4; 4; 5; 6; 10; 10; 12; 13; 14; 15;
               17; 19; 20; 23; 27; 33; 34; 37; 40; 40; 41;
               49; 52; 52; 56; 57; 60; 61; 63; 63; 64; 67;
               67; 73; 75; 76; 80; 82; 86; 87; 87; 88; 89;
               91; 94; 95; 99]

#tri_pivot (<=) (random_list 10 50);;
   int list = [0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 2; 2; 2; 3; 3;
               4; 4; 4; 4; 4; 4; 4; 5; 5; 5; 6; 6; 6; 6; 6;
               6; 6; 7; 7; 7; 7; 7; 8; 8; 8; 8; 8; 9; 9; 9;
               9; 9; 9; 9; 9]
```

1.3 Tri à bulle

```
#tri_bulle_bis (<=) [9;2;3;7;6;4;1;2;8;0];;
   int list = [2; 3; 7; 6; 4; 1; 2; 8; 0; 9]

#tri_bulle (>=) (random_list 50 10);;
   int list = [43; 38; 36; 32; 26; 25; 25; 20; 8; 4]

#tri_bulle (>=) (random_list 1000 100);;
   int list = [994; 990; 965; 942; 925; 925; 922; 909; 894;
               893; 869; 863; 859; 857; 850; 846; 844; 810;
               807; 788; 779; 741; 719; 695; 692; 674; 672;
               672; 667; 652; 628; 615; 605; 598; 594; 593;
               588; 582; 573; 557; 546; 540; 530; 527; 504;
               488; 481; 481; 467; 461; 460; 458; 450; 448;
               435; 418; 396; 396; 382; 380; 377; 374; 372;
               372; 356; 339; 338; 326; 323; 307; 294; 279;
               269; 267; 247; 239; 232; 213; 204; 190; 179;
               151; 148; 139; 136; 128; 119; 113; 109; 106;
               106; 71; 62; 54; 54; 13; 10; 2; 2; 0]

#tri_bulle (>=) (random_list 100 50);;
   int list = [99; 96; 95; 92; 91; 88; 88; 86; 86; 83; 82;
               80; 77; 74; 73; 72; 71; 69; 67; 67; 58; 58;
               57; 55; 54; 52; 52; 46; 46; 39; 37; 36; 36;
               34; 33; 25; 25; 24; 20; 20; 19; 16; 14; 12;
               11; 11; 6; 5; 3; 1]

#tri_bulle (>=) (random_list 10 50);;
   int list = [9; 9; 9; 9; 9; 9; 8; 8; 8; 8; 8; 7; 7; 7; 6;
               6; 6; 6; 6; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 4;
               4; 4; 4; 4; 4; 3; 3; 2; 2; 2; 2; 2; 1; 1; 1;
               1; 0; 0; 0; 0]
```

1.4 Fonctions utilitaires

```
#min_list (<) [28; 6; 1; -8; 22; 33];;
   int = -8

#suppr_doublons [1; 2; 3; 2; 4; 3; 5; 1];;
   int list = [2; 4; 3; 5; 1]
```

2 Choix d'une liste : comparaison des temps de calculs

Calcul du temps moyen de chaque algorithme sur 1000 itérations avec le comparateur (>) :

Type et taille de la liste utilisé	Tri par partition-fusion	Tri pivot	Tri à bulle
Liste courte sans doublons (50)	0.0113 ms	0.00954 ms	0.0639 ms
Liste courte avec doublons (50)	0.0114 ms	0.0115 ms	0.058 ms
Liste longue sans doublons (10,000)	7.4 ms	6.8 ms	4600 ms
Liste longues avec doublon (10,000)	6.9 ms	18 ms	4500 ms
Liste trié par (>) (100)	0.0238 ms	0.15 ms	0.0045 ms
Liste trié par (<) (100)	0.0293 ms	0.17 ms	0.284 ms

Cela nous permet de faire les conclusions suivantes :

2.1 Tri par partition-fusion

1. Listes courtes :
   — Avec doublons : Très rapide
   — Sans doublons : Rapide
2. Listes longues :
   — Avec doublons : Très rapide
   — Sans doublons : Passable
3. Liste trié : Rapide
4. Liste trié inverse : Rapide

Dans un premier temps nous allons voir comment ce comporte le tri partition-fusion en fonction de différents paramètres. L'on peut voir que le tri partition-fusion reste efficace et assez rapide peu importe le format de liste en entrée. Cela est dû au fait qu'il ne se soucis pas des valeurs contenu dans la liste qu'il tri dans un premier temps, mais qu'il n'utilise que le rang des éléments de la liste.

2.2 Tri par pivot

1. Listes courtes :
   — Avec doublons : Très rapide
   — Sans doublons : Très rapide
2. Listes longues :
   — Avec doublons : Lent
   — Sans doublons : Très rapide
3. Liste trié : Très lent
4. Liste trié inverse : Très lent

Ensuite, le tri pivot. Cette technique de tri reste, selon les resultats, une technique assez fiable même si l'on remarque clairement un temps de plus en plus long pour des demandes plus complexes avec des listes plus longues ou des listes trié par ordre croissant ou décroissant.

L'on constate aussi un temps plus long sur les listes avec des doublons. L'utilisation d'un pivot explique ce comportement. Plus il y a de doublons, plus la liste droite recoit des éléments au détriment de la liste gauche. Cela a pour effet une progression lente ou la liste de droite décroit lentement en taille. Lorsque tout les éléments de la liste sont pareils, par exemple, la liste de gauche est toujours vide et le seule changement que subit la liste de droite est la suppression du pivot. Afin de mitiger ce problème il faudrait calculer une troisième partition contenant les éléments égal au pivot. Cela n'aiderait par contre pas dans le cas de listes triées, dans lequel l'algorithme se comporte de façon similaire.

2.3 Tri à bulle

1. Listes courtes :
   — Avec doublons : Très lent
   — Sans doublons : Très lent
2. Listes longues :
   — Avec doublons : Extrèmement lent
   — Sans doublons : Extrèmement lent
3. Liste trié : Extrèmement rapide
4. Liste trié inverse : Lent

Pour finir, le tri à bulle, le tri le moins efficace selon les résultats. Les temps très lent enregistré lors des tests sont dû au fait que le tri a bulles est un tri qui dois refaire plusieur fois le tour de la liste tout en permutant les chiffres au fur et a mesure ce qui donne une méthode de tri sûr mais très lente et très peu efficace quelque soit la taille de la liste, la présence ou non de doublons. Seule exception lorsque la liste est trié le tri à bulle est extrèmement rapide.

3 Conclusion

Pour conclure, en comparant les temps des différentes techniques de tri programmé et testé, l'on peut voir que le tri par partition-fusion a un certain avantages face aux autres technique de tri car les temps mesuré lors des test reste relativement proche et rapide selon les variantes de liste longues ou courtes, trié ou non, mais aussi des doublons qui ne lui posent visiblement pas de problème. N'ayant pas l'intention de vérifier de quelle type de liste il s'agit, le tri par partition-fusion semble le meilleur choix puisqu'il n'y a que peu de disparités de temps dans les différents cas.