CMP-SCI 5130

Hao Lan

Pseudocode and Complexity analysis

11 March 2022

Pseudocode

```
1       Struct Node
2       Let parent, left, middle, right, val be member elements of Node.
3       Let CASE[0..n][0..n] and CTRL[0..n][0..n] be new vectors
/* Global 2D vectors, CASE, CTRL[0] are headers (first row: ID), first column are snp names such as CASE[1][0] and CTRL[2][0]  */

4       Let opt_pattern[0..n] be a new vector
        // char type stores the snp pattern when hit optimal solution //
5       RESULT= 0.0
        // double type stores optimal solution //

MakeTreeNode( c )  // character
1       Let newNode be new Node
2       newNode.val = c
3       return newNode

FindTwoAlleles( a ) // string vector a //
1       for i = 1 to a.size()
2               if a[i][0] not = a[i][1]
3               return a[i]
4       return "error"
        // see which two alleles in one snp //
```

```
Occur_difference( path ) // char vector path, length same to number of snp in file //
1       diff, count1, count2 = 0.0
2       found = true
3       for i = 1 to CASE[i].size()
4               for j = 0 to CASE.size() -1
5                       if 'N' in CASE[j+1][i]
6                               print CASE[0][i] + "has no alleles"
7                       else if path[j] == 'X'
8                               continue
9                       else if path[j] not in CASE[j+1][i]
10                              found = false
11              if found
12                      count1 = count1 +1
13              found = true
14      diff = count1 / CASE[1].size() -1      // occurrence percentage in CASE
15      for i = 1 to CTRL[i].size()
16              for j = 0 to CTRL.size -1
17                      if 'N' in CASE[j+1][i]
18                              print CASE[0][i] + "has no alleles"
19                      else if path[j] == 'X'
20                              continue
21                      else if path[j] not in CTRL[j+1][i]  // Path doesn't match the alleles
22                              found = false
23              if found        // bool value found = true
24                      count2 = count2 +1
25              found = true
26      diff = diff – count2 / CTRL[1].size()-1   // percentage in CASE - percentage in CTRL
27      return abs(diff)
```

```
BuildTree_and_travese_everything_whatever( node, allele, path )
// Function that finds the solution, where node is a Node struct, allele and path are char vector//
// allele has characters A T X C G X A C X…. Which are every SNPs' two pattern with additional 'X' //
// X in allele means when we pick it, we pick no allele in this SNP, which is equivalent to null //
1       if allele not empty
2               node.right = MakeTreeNode( allele.back() )
                //  .back() simply means the last element in it //
3               node.right.parent = node
4               delete last element in allele  //allele.pop_back()
5               node.middle = MakeTreeNode( allele.back() )
6               node.middle.parent = node
7               delete last element in allele
8               node.left = MakeTreeNode( allele.back() )
9               node.left.parent = node
10              delete last element in allele
11      else    // this means it's built to lowest level, now get one single path, and delete the node //
12              Let cur be new Node
13              cur = node
14              for i = 0 to CASE.szie() – 1p
15                      if cur.left, cur.right, cur.middle == NULL
16                              Let temp be new Node
17                              temp = cur
18                              path = path append cur.val   // path.push_back( cur.val )
19                              cur = cur.parent
20                              delete temp
21                      else
22                              path = path append cur.val // path.push_back( cur.val )
```

```
23                              cur = cur.parent
24          reverse(path)  // member function in vector to reverse vector path //
25          if RESULT < Occur_difference( path )
                    // get biggest occur diff here
26                  opt_pattern = path
27                  RESULT = Occur_difference( path )
28          return
29      BuildTree_and_travese_everything_whatever( node.right, allele, path )
30      BuildTree_and_travese_everything_whatever( node.middle, allele, path )
31      BuildTree_and_travese_everything_whatever( node.left, allele, path )
        // go over every pattern, path is deleted
```

```
Go_get_all_folks( C, a ) // 2D string Vector C and 1D char vector a
1       found = true
2       for i = 0 to C[1].size()
3           for j = 0 to C.size()-1
4               if a[j] == 'X'
5                   continue
6               else if a[j] not in CASE[j+1][i]
7                   found = false
8           if found
9               print C[0][i]
10          found = true
```

MAIN

```
1       caseGroup = casefileName
```

```
2       ctrlGroup = ctrlfileName
3       Let in1 = input from caseGroup as string // ifstream
4       Let in2 = input from ctrlGroup as string

5       for line, getline(in1, line)
6               Let ss1 be string converted from stream
7               Let row[0..n] be a new vector
8               for d, ss1 >> d
9                       row = row append d
10              CASE = CASE append row

11      for line, getline(in2, line)
12              Let ss2 be string converted from stream
13              Let row[0..n] be a new vector
14              for d, ss2 >> d
15                      row = row append d
16              CTRL = CTRL append row

17      types = “” ,   Let root be new Node
18      for i = 1 to CASE.size()
19              types = FindTwoAlleles( CASE[i] + ‘X’ ) + types
20      Let path[0..n] be a new vector, type[0..types.size] be a new vector

21      BuildTree_and_traverse_everything_whatever( root, type, path )

22      print "The optimal solution for " + CASE.size()-1 + " snp case/ctrl file is: " + RESULT
23      print “The pattern is: ”
24      for j=1 to CTRL.size()
25              print CTRL[j][0] + opt_pattern[j-1]
```

```
26      print "Note: The X means the program doesn't take any allele at the particular SNP, which doesn't counted in pattern size as well"
27      choice = 0
28      print "enter '1' to see all CASE ID that has this pattern, '2' for CTRL, '3' for both, other keys to end: "
29      if choice == 1
30              “CASE:”
31              print go_get_all_folks( CASE, opt_pattern )
32      else if choice == 2
33              print “CTRL: ”
34              go_get_all_folks( CTRL, opt_pattern )
35      else if choice == 3
36              print “CASE:”
37              go_get_all_folks( CASE, opt_pattern )
38              print “CTRL: ”
39              go_get_all_folks( CTRL, opt_pattern )
40      return 0
```

Complexity analysis

The space complexity is concluded in these aspects:

Global line 3: CASE, CTRL are 2D vectors which takes [0..n][0..n] space each, where n is number of id in the case and control text file, so is considered

O(# of ID X # of SNP + 1 (header))

Global line 4: opt_pattern is 1D array [0..n], where n is number of SNP in both case and control file.

O(# of SNP)

Main line 7 and 13: two vector called “row” [0..n] is 1D vector

O(# of ID)

Main line 20: path and type are 1D vector [0..n]:

O(# of SNP) + O(# of SNP X 3)

Since # of SNP X 3 is greater, the overall sub-section space complexity is:

O(# of SNP X 3)

Overall space complexity is: O(n^2) → O((# of ID X # of SNP + 1))

The time complexity concluded in these aspects:

Main line 5 and 11: Two for loop go over # of ID times:

O(# of ID) + O(# of ID) = O(# of ID) overall in this sub-section.

Main line 18 and 24: for loop go over # of SNP times:

O(n) = O(# of SNP)

Main 19: call function **FindTwoAlleles**: → Function definition line 1: a for-loop

O(# of ID) for worst case

O(# of SNP) for best case

Main 21 call function **BuildTree_and_travese_everything_whatever** → Function definition

Line 14: A for-loop O(# of ID)

Line 29, 30, 31: Recursion to make every branch O(3^n), where n is the # of SNP,

which is O(3^(# of SNP))

Line 25 and 27: call function **Occur_difference** → Function definition

Line 3,4 and 15,16: O(n^2) = O(# of ID X # of ID), since line 25 and 27 call it twice, it's total of 4 X O(# of ID X # of ID)

Overall time complexity is: O(3^(# of ID))