The following sequence counts the number of non-degenerate tetrahedral with a total edge length of n (sequence starts with n=6):

 1, 0, 0, 1, 1, 1, 3, 2, 3, 6, 6, 7, 12, 11, 18, 21, 25, 31, 38, 46, 56, 66, 76, 90, 117

This sequence does not appear to be in OEIS so I have submitted it (A208454).

I do not have an explicit formula for the terms of this sequence. I think there should be a simple formula (perhaps C-finite?) which describes the sequence. The sequence is generated by a Maple program (in Appendix) which may be somewhat opaque so here is a general description of the program:

1. Generate all lists of 6 natural numbers whose sum is n. Note: this is ordered partitions of n.
2. Interpret the elements of the list as the (potential) edges of a tetrahedron as follows. Let the tetrahedron by ABCD and the edge list is [AB,AC,AD,BC,BD,CD]
3. Check whether the list of edges generated is feasible for the tetrahedron. This involves two checks: (a) Check that all triangle inequalities are satisfied for surface triangles ABC, ABD, ACD, BCD and (b) Check that the volume of the tetrahedron is positive. Step (b) is necessary for two reasons: we want to eliminate any degenerate (zero volume) tetrahedral and it is not sufficient to simply check surface triangles for feasibility as the following example shows: [2,2,3,3,2,2] is infeasible!
4. Since there will be permutations of the edges that actually produce the same tetrahedron, we want to check for duplicates. This issue is discussed below.

That’s the gist of the program listing at the end of this note. The strategy is “brute force”. There might be speed ups that are obvious. I’d appreciate hearing about that.

**Checking for duplicate tetrahedra**

The automorphism group of the vertices of a tetrahedron is S4, the complete symmetric group on 4 elements. Each such automorphism induces an automorphism on the list of edges. In order to avoid counting a tetrahedron twice, the approach is:

1. Count the set of permutations of the edge list produced by all 24 vertex automorphisms. This is the weight of the original edge list.
2. When adding to the total count of tetrahedra, just add the reciprocal of the weight of the edge list (instead of adding 1).

Since every permutation of the edge list will eventually be generated, this approach guarantees that each unique tetrahedron will only be counted once.

This table shows some edge list values for small values of the total edge length

|  |  |
| --- | --- |
| Edge Sum | Edges |
| 6 | 1 – 1 – 1 – 1 – 1 – 1 |
| 7 | (none) |
| 8 | (none) |
| 9 | 1 – 1 – 1 – 2 – 2 – 2 |
| 10 | 1 – 2 – 2 – 2 – 2 – 1 |
| 11 | 1 – 2 – 2 – 2 – 2 – 2 |
| 12 | 1 – 1 – 3 – 1 – 3 – 3 |
|  | 1 – 2 – 2 – 2 – 2 – 3 |
|  | 2 – 2 – 2 – 2 – 2 – 2 |
| 13 | 1 – 2 – 3 – 2 – 3 – 2 |
|  | 2 – 2 – 2 – 2 – 2 – 3 |
| 14 | 1 – 3 – 3 – 3 – 3 – 1 |
|  | 1 – 2 – 3 – 2 – 3 – 3 |
|  | 2 – 2 – 2 – 3 – 3 – 2 |

**Maple code**

# Procs for tetrahedra enumeration

# Phil Benjamin

# Sample for counting inequivalent tetrahedra with given total edge length

# seq(CountTetra(n)[1],n=6..30)

# produces this output

# 1, 0, 0, 1, 1, 1, 3, 2, 3, 6, 6, 7, 12, 11, 18, 21, 25, 31, 38, 46, 56, 66, 76, 90, 117

# this sequence does not appear to be in OEIS ...

Help := proc()

 print(`V2x144(AB,AC,AD,BC,BD,CD)`);

 print(`IsTriangle(a,b,c)`);

 print(`CountTetra(len)`);

 print(`CountAuto(AB,AC,AD,BC,BD,CD)`);

end proc:

# Volume: squared and multiplied by 144 (to produce integer values)

# Edges are for tetrahedron ABCD

V2x144 := proc(AB,AC,AD,BC,BD,CD)

 local t;

 t := AB^2 \* CD^2 \* (AC^2 + AD^2 + BC^2 + BD^2 - AB^2 - CD^2);

 t := t + AC^2 \* BD^2 \* (AB^2 + AD^2 + BC^2 + CD^2 - AC^2 - BD^2);

 t := t + AD^2 \* BC^2 \* (AB^2 + AC^2 + BD^2 + CD^2 - AD^2 - BC^2);

 t := t - AB^2 \* AC^2 \* BC^2 - AB^2 \* AD^2 \* BD^2;

 t := t - AC^2 \* AD^2 \* CD^2 - BC^2 \* BD^2 \* CD^2;

 return t;

end proc:

# Check for feasible triangle

IsTriangle := proc(a,b,c)

 return evalb(a + b + c > 2\*max(a,b,c));

end proc:

# Find all inequivalent tetrahedrons with given total edge length

# Return triple

# (i) total count

# (ii) sequence of [[edges], automorphism count, volume]

CountTetra := proc(len)

 # Tetrahedron ABCD with edges AB (a) AC (b) AD (c) BC (f) BD (e) CD (d)

 # Note: edges at vertex A are {a,b,c}

 # Note: opposite pairs of edges are {a,d} {b,e} {c,f}

 local a, b, c, d, e, f;

 local count, nAuto, s, v;

 # Strategy: consider all assignments of {a .. f} with sum(a..f) = len

 # Prune according to constraints:

 # (i) all face triangles must be feasible

 # (ii) volume must be positive

 # Problem: account for duplicates

 # Brute force strategy: count distinct edge automorphisms

 # Weight count by inverse of automorphism count

 count := 0; s := [];

 for a from 1 to len - 5 do

 for b from 1 to len - a - 4 do

 for c from 1 to len - a - b - 3 do

 for d from 1 to len - a - b - c - 2 do

 # check triangle {b,c,d}

 if not IsTriangle(b,c,d) then next; end if;

 for e from 1 to len - a - b - c - d - 1 do

 # check triangle {a,c,e}

 if not IsTriangle(a,c,e) then next; end if;

 f := len - a - b - c - d - e;

 # check triangle {a,b,f}

 if not IsTriangle(a,b,f) then next; end if;

 # check triangle {d,e,f}

 if not IsTriangle(d,e,f) then next; end if;

 # check volume

 v := V2x144(a,b,c,f,e,d);

 if v <= 0 then next; end if;

 nAuto := CountAuto(a,b,c,f,e,d);

 count := count + 1/nAuto;

 s := [op(s),[[a,b,c,f,e,d],nAuto,v]];

 end do;

 end do;

 end do;

 end do;

 end do;

 return count,s;

end:

CountAuto := proc(AB,AC,AD,BC,BD,CD)

 local a, b, c, d, e, f;

 local s;

 a := AB; b := AC; c := AD; d := CD; e := BD; f := BC;

 s := {[a,b,c,f,e,d],[a,e,f,c,b,d],[a,c,b,e,f,d],[a,f,e,b,c,d]};

 s := s union {[d,b,f,c,e,a],[d,e,c,f,b,a],[d,c,e,b,f,a],[d,f,b,e,c,a]};

 s := s union {[b,a,c,f,d,e],[b,d,f,c,a,e],[b,c,a,d,f,e],[b,f,d,a,c,e]};

 s := s union {[e,a,f,c,d,b],[e,d,c,f,a,b],[e,c,d,a,f,b],[e,f,a,d,c,b]};

 s := s union {[c,a,b,e,d,f],[c,d,e,b,a,f],[c,b,a,d,e,f],[c,e,d,a,b,f]};

 s := s union {[f,a,e,b,d,c],[f,d,b,e,a,c],[f,b,d,a,e,c],[f,e,a,d,b,c]};

 return nops(s);

end proc: