Templates for the k-binomial complexity of the Tribonacci word,
M. Lejeune, M. Rigo, M. Rosenfeld
April 2019

Basics - Section 2 of the paper

Computation of a prefix of length 223 317 of the Tribonacci word.

```
In[•]:= τ[u_] := Flatten[u /. {0 → {0, 1}, 1 → {0, 2}, 2 → {0}}]
        tribo = Nest[τ, {0}, 20];
```

Computing factors of length i occurring in the prefix tribo.

```
In[•]:= factors[i_] := DeleteDuplicates[Partition[tribo, i, 1]];
        factors[10]
```

```
Out[•]= {{0, 1, 0, 2, 0, 1, 0, 0, 1, 0},
  {1, 0, 2, 0, 1, 0, 0, 1, 0, 2}, {0, 2, 0, 1, 0, 0, 1, 0, 2, 0},
  {2, 0, 1, 0, 0, 1, 0, 2, 0, 1}, {0, 1, 0, 0, 1, 0, 2, 0, 1, 0},
  {1, 0, 0, 1, 0, 2, 0, 1, 0, 1}, {0, 0, 1, 0, 2, 0, 1, 0, 1, 0},
  {0, 1, 0, 2, 0, 1, 0, 1, 0, 2}, {1, 0, 2, 0, 1, 0, 1, 0, 2, 0},
  {0, 2, 0, 1, 0, 1, 0, 2, 0, 1}, {2, 0, 1, 0, 1, 0, 2, 0, 1, 0},
  {0, 1, 0, 1, 0, 2, 0, 1, 0, 0}, {1, 0, 1, 0, 2, 0, 1, 0, 0, 1},
  {1, 0, 0, 1, 0, 2, 0, 1, 0, 2}, {0, 0, 1, 0, 2, 0, 1, 0, 2, 0},
  {0, 1, 0, 2, 0, 1, 0, 2, 0, 1}, {1, 0, 2, 0, 1, 0, 2, 0, 1, 0},
  {0, 2, 0, 1, 0, 2, 0, 1, 0, 0}, {2, 0, 1, 0, 2, 0, 1, 0, 0, 1},
  {1, 0, 0, 1, 0, 2, 0, 1, 0, 0}, {0, 0, 1, 0, 2, 0, 1, 0, 0, 1}}
```

To check if there are no missing factors (because we are only considering a prefix of the infinite word), we know the factor complexity hence, we simply compute:

```
In[•]:= Length[factors[10]] == 2 × 10 + 1
```

```
Out[•]= True
```

For instance, in the prefix of length 223 317 of the Tribonacci word, we still have all the factors of length up to 1000.

```
In[•]:= Length[factors[1000]] == 2001
```

```
Out[•]= True
```

Definition of functions

Here is a general definition of the binomial coefficients of words, and of the two functions ψ and ψ' introduced in the paper ; ϵ is a short-hand for the empty sequence.

In[•]:=
```
coeff[u_, v_] := coeff[u, v] = If[Length[v] == 0, 1,
   If[Length[u] < Length[v], 0, coeff[Drop[u, -1], v] +
     ((Last[u] == Last[v]) /. {True → 1, False → 0}) coeff[Drop[u, -1], Drop[v, -1]]]]

ψprim[u_] := Map[Count[u, #] &, {0, 1, 2}]

ψ[u_] := Join[ψprim[u], Map[coeff[u, #] &, Tuples[{0, 1, 2}, 2]]]

ϵ = {};
```

The Kronecker product is a built-in *Mathematica* function, we simply gives another notation for Kronecker product of two lists.

In[•]:=
```
c_List ⊗ d_List := Flatten[KroneckerProduct[c, d]]
```

Matrices associated with Tribonacci - equation (1) and Lemma 3

In[]:=
```
P3 = {{0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0},
    {1, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 1, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 1}};

M = {{1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0}, {0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0},
    {0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0}};

Mprim = {{1, 1, 1}, {1, 0, 0}, {0, 1, 0}};

invMprim = Inverse[Mprim];
invM = Inverse[M];
MatrixForm[Mprim]
MatrixForm[M]
```

Out[]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Out[]//MatrixForm=

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

About templates and parents - Section 3 of the paper

References to Definition 11 : computation of the sets pref(τ^n) and suff(τ^n)

For prefixes, it is enough to consider images of 0, this is not the case with suffixes; this is due to the special form of the morphism τ.

```
In[•]:= pref[n_] := Reverse[Table[Drop[#, -i], {i, 1, Length[#]}] &[Nest[τ, {0}, n]]];
suff[n_] := Sort[DeleteDuplicates[Flatten[
     Table[Table[Drop[#, i], {i, 1, Length[#]}] &[Nest[τ, {i}, n]], {i, 0, 2}], 1]]];
```

The following function takes the letters a_1 and a_2 of a template related to u and v, and returns the possible 4-tuples (p_u,s_u,p_v,s_v) as given in Definition 11.

```
In[•]:= findParam[a1_, a2_] := Block[{s1, s2},
  If[a1 == 0, s1 = {ϵ, {1}, {2}}, s1 = {ϵ}];
  If[a2 == 0, s2 = {ϵ, {1}, {2}}, s2 = {ϵ}];
  Tuples[{{ϵ, {0}}, s1, {ϵ, {0}}, s2}]
 ]
```

The next function takes a template t=(d,D_b,D_e,a_ 1,a_ 2) and a 4-tuple (p_u,s_u,p_v,s_v), and returns the corresponding parent template t', see Definition 12.

```
In[•]:= parent[{d_, Db_, De_, a1_, a2_}, {pu_, su_, pv_, sv_}] :=
 Block[{dprim, deprim, dbprim, aa1, aa2},
  dprim = invM.(d + ψ[Join[pu, su]] - ψ[Join[pv, sv]] +
       P3.(ψprim[pv] ⊗ d[[1 ;; 3]] + d[[1 ;; 3]] ⊗ ψprim[sv])
       - P3.((Db + ψprim[pu] - ψprim[pv]) ⊗ ψprim[Join[pu, su]] +
          ψprim[Join[pu, su]] ⊗ (De + ψprim[su] - ψprim[sv])));
  dbprim = invMprim.(Db + ψprim[pu] - ψprim[pv]);
  deprim = invMprim.(De + ψprim[su] - ψprim[sv]);

  aa1 = Which[
    su == ϵ, Which[a1 == 0, 2, a1 == 1, 0, a1 == 2, 1],
    su == {1}, 0,
    su == {2}, 1];
  aa2 = Which[
    sv == ϵ, Which[a2 == 0, 2, a2 == 1, 0, a2 == 2, 1],
    sv == {1}, 0,
    sv == {2}, 1];
  Return[{dprim, dbprim, deprim, aa1, aa2}]
 ]
```

Here is an example of application.

```
In[•]:= parent[{{1, 0, -1, -1, -1, 4, 1, 0, 1, -3, -1, 0}, {0, -1, 0}, {1, 1, -1}, 0, 0},
 {ϵ, {1}, {0}, {}}]
Out[•]= {{1, -1, 0, -1, 2, -1, -1, 0, -2, 1, 2, 0}, {-1, 0, 0}, {2, -1, 0}, 0, 2}
```

We provide a list of all possible parents of a given template (ranging over all the possible (p_u,s_u,p_v,s_v)).

```
In[•]:= listParents[template_] :=
 Map[parent[template, #] &, findParam[template[[4]], template[[5]]]]
```

Bounds - Section 4.1 of the paper

About eigenvalues and eigenvectors of M_τ

Computation of the (exact) eigenvalues and associated eigenvectors
duosVP is the list of pairs {eigenvalue, associated eigenvectors}.

```
In[•]:= vp = Eigensystem[Transpose[M]];
duosVP = Table[{vp[[1, i]], vp[[2, i]]}, {i, 1, Length[vp[[1]]]}];
θ = duosVP[[2, 1]] ;
(* This is the exact root close to 1.83929 - no approximation,
Mathematica keeps it as a root of polynomial. *)
α = 1 / θ;  (* Here we have the exact densities for 0, 1, 2. *)
β = 1 / (θ^2);
γ = 1 / (θ^3);
```

One of the eigenvalues (the third one in duosVP) can be approximated by -0.7718445063+1.1151425080 i. Its multiplicity equals 2. Let us denote by v1 and v2 the two associated eigenvectors.
To accelerate the computations, we virtually add to the vector duosVP a third occurrence of this eigenvalue, associated with the eigenvector v1+v2.

```
In[•]:= AppendTo[duosVP, {duosVP[[3, 1]], duosVP[[3, 2]] + duosVP[[4, 2]]}];
```

Note that from now, we will consider numerical approximations of these eigenvalues and eigenvectors, keeping 10 exact decimal numbers.

```
In[•]:= duosVP = N[duosVP, 10];
```

Constants c1, c2 and function f from Lemma 8

```
In[•]:= c1[r_, p_, s_] := Abs[r.(P3.(ψprim[p] ⊗ {α, β, γ} + {α, β, γ} ⊗ ψprim[s]))]
f[r_, p_, s_, δ0_, δ1_, δ2_] :=
 r.(ψ[Join[p, s]] + (P3.(ψprim[p] ⊗ {δ0, δ1, δ2} + {δ0, δ1, δ2} ⊗ ψprim[s])))
```

To compute c2(r,p,s), we need to compute l_0, l_1, l_2, u_0, u_1 and u_2, which depend on p and s. This is done by the two following functions.

In[•]:=
```
l[i_, p_, s_, dens_] := Max[Map[
    dens * Length[Join[ #[[1]], #[[2]] ]] - Count[ Join [#[[1]], #[[2]] ], i] &, Tuples[{
      Table[Drop[p, j], {j, 0, Length[p]}],
      Table[Drop[s, -j], {j, 0, Length[s]}]}]]] - 3 / 2

u[i_, p_, s_, dens_] := Min[Map[
    dens * Length[Join[ #[[1]], #[[2]] ]] - Count[ Join [#[[1]], #[[2]] ], i] &, Tuples[{
      Table[Drop[p, j], {j, 0, Length[p]}],
      Table[Drop[s, -j], {j, 0, Length[s]}]}]]] + 3 / 2
```

To compute the value of c2(r,p,s), we used the development carried on right after the proof of Lemma 23.

In[•]:=
```
c2[r_, p_, s_] := Module[{l0, l1, l2, u0, u1, u2},
  l0 = l[0, p, s, α];
  l1 = l[1, p, s, β];
  l2 = l[2, p, s, γ];
  u0 = u[0, p, s, α];
  u1 = u[1, p, s, β];
  u2 = u[2, p, s, γ];
  Return[Min[Max[Map[Abs[f[r, p, s, #[[1]], #[[2]], -#[[1]] - #[[2]]]] &,
      Tuples[{{l0, u0}, {l1, u1}}]]],
    Max[Map[Abs[f[r, p, s, #[[1]], -#[[1]] - #[[2]], #[[2]]]] &,
      Tuples[{{l0, u0}, {l2, u2}}]]],
    Max[Map[Abs[f[r, p, s, -#[[1]] - #[[2]], #[[1]], #[[2]]]] &,
      Tuples[{{l1, u1}, {l2, u2}}]]],
    Max[
     Map[Abs[f[r, p, s, #[[1]], #[[2]], #[[3]]]] &,
      Tuples[{{l0, u0}, {l1, u1}, {l2, u2}}]]]
  ]
 ]
```

Constant C(r) of Proposition 25, for eigenvalues λ such that $\theta > |\lambda| > 1$

Since all our computations are concerned with modulus of complex numbers and not the complex numbers theirselves, we don't need to consider all the eigenvalues. Indeed, if λ was already treated, we don't have to carry on the eigenvalue equal to the conjugate value of λ.
This is why we just have to look on components number 7,9,11,12 for eigenvalues of modulus less than 1, and on components number 3,4,13 for eigenvalues of modulus greater than 1.
This function computes the maximum value of $|r \cdot \psi(u)| / |u|$ over all factors u of Tribonacci of length at most l.

In[•]:=
```
smallW[r_, l_] := Max[Table[Max[Map[N[Abs[r.ψ[#]] / i, 10] &, factors[i]]], {i, 1, l}]]
```

The following function is an approximation of the ι that is used in the proof. The second one gives, for fixed λ and n, the minimum value of l which satisfies conditions (7).

```
In[•]:= ι[l_, n_] := N[l / (l + θ^n * (2 + (3 / 2) / (θ - 1))), 10]
lMin[n_, λ_] := Module[{l},
  l = Ceiling[θ^n * (2 + (3 / 2) / (θ - 1))];
  While[Abs[λ]^n / (ι[l, n] * θ^n) ≥ 1, l++];
  Return[l];
 ]
```

We compute the value of c3(r).

```
In[•]:= c3[r_, n_, l_] := Max[Map[ c1[r, #[[1]], #[[2]]] + 1 / l * c2[r, #[[1]], #[[2]]] &,
    Tuples[{pref[n], suff[n]}]]]
```

Finally, this function computes the value of the bound. Its depend on the chosen values of n and l, as explained in the proof. This function shouldn't be used if l is less than lMin[n,λ].
We compute the values of the bounds using n=6 and l=600. For information, lMin[6,λ] = 147 for all λ of modulus greater than 1. This second function returns pairs of elements of the form {n, bound}. This form will be useful later on. We add 10^(-7) to the bounds, since they are just approximations (which are exact up to 7 decimal digits).
Since the computations are quite long, the last command allows to stock these values in an out file.

```
In[•]:= Cgreat[λ_, r_, n_, l_] := Module[{temp = ι[l, n] * θ^n, pt = smallW[r, l]},
  Max[pt, Abs[λ]^n / temp * pt + c3[r, n, l] * temp / (temp - Abs[λ]^n)]
 ]
```

```
In[•]:= CgreatList =
 Table[{6, Cgreat[duosVP[[i, 1]], duosVP[[i, 2]], 6, 600] + 10^(-7)}, {i, {3, 4, 13}}]
Export["CgreatList_n6l600.m", CgreatList];
```

```
Out[•]= {{6, 3.8809965}, {6, 3.11975700}, {6, 2.46598485}}
```

If you don't want to do this computation every single execution, you can import the bounds.

```
In[•]:= CgreatList = Import["CgreatList_n6l600.m"];
```

Constant C(r) of Proposition 26, for eigenvalues λ such that $|\lambda| < 1$

As in Proposition 25, this bound depends on the value of the positive integer n that we choose. As the computations are quite fast, we chose better bound by allowing to take different values of n for different values of λ. The second function, for a given λ and its associated eigenvector, computes the best bound for different values of n, ranging from nMin to nMax. It returns the pair {n, bestBound}.

```
In[•]:= Csmall[λ_, r_, n_] :=
 Max[Map[c2[r, #[[1]], #[[2]]] &, Tuples[{pref[n], suff[n]}]]] / (1 - Abs[λ]^n)
```

In[•]:=
```
bestCsmall[λ_, r_, nMin_, nMax_] := Module[{listBounds, pos},
  listBounds = Table[Csmall[λ, r, n], {n, nMin, nMax}];
  pos = Ordering[listBounds, 1][[1]];
  Return[{pos + nMin - 1, listBounds[[pos]]}];
 ]
```

We now compute the values of the bounds, choosing the best n between 1 and 6 (these values can be modified).
We add 10^(-7) to the bounds, since they are just approximations (which are exact up to 7 decimal digits).
We then export them in an out file.

In[•]:=
```
CsmallList =
 Table[bestCsmall[duosVP[[i, 1]], duosVP[[i, 2]], 1, 6], {i, {7, 9, 11, 12}}]
CsmallList = Map[{#[[1]], #[[2]] + 10^(-7)} &, CsmallList];
Export["CsmallList.m", CsmallList];
```

Out[•]=
```
{{6, 2.19301322}, {6, 3.805281713}, {6, 3.18002470}, {6, 2.762848807}}
```

As before, if you don't want to do this computation every single execution, you can import the bounds.

```
CsmallList = Import["CsmallList.m"];
```

Bounds on templates - Section 4.2 of the paper

Since we work with approximations, we check in the following functions that we are still working with 7 exact decimal digits. Otherwise, the program will report an error (but still continue its execution).

Bounds associated with eigenvalues of modulus less than 1

We just define the following function that is will simplify the carried on computations.

In[•]:=
```
fbis[r_, t_, δ0_, δ1_, δ2_] :=
 r.(t[[1]] + (P3.(t[[2]] ⊗ {δ0, δ1, δ2} + {δ0, δ1, δ2} ⊗ t[[3]])))
```

The following function verifies that the template t is valid regarding the bound Csmall associated to the eigenvalue λ and the corresponding eigenvector r. We make use of the remark right after the proof of Lemma 27.

```
smallBound[t_, λ_, r_, n_, Csmall_] := Module[{aRe, bRe, aIm, bIm, minRe, minIm},
  {aRe, bRe, aIm, bIm} =
   Through[{Min[Re[#]] &, Max[Re[#]] &, Min[Im[#]] &, Max[Im[#]] &}[
     Map[fbis[r, t, #[[1]], #[[2]], -#[[1]] - #[[2]]] &,
      {{3/2, 0}, {3/2, -3/2}, {0, 3/2}, {0, -3/2}, {-3/2, 3/2}, {-3/2, 0}}]]];
  minRe = Which[aRe ≥ 0, aRe^2, bRe ≤ 0, bRe^2, True, 0];
  minIm = Which[aIm ≥ 0, aIm^2, bIm ≤ 0, bIm^2, True, 0];
  If[Accuracy[Sqrt[minRe + minIm]] < 7, Print["Accuracy problem"]];
  Return[Sqrt[minRe + minIm] ≤ 2 * Csmall];
 ]
```

Bounds associated with eigenvalues of modulus greater than 1

The following function verifies that the template t is valid regarding the bound Cgreat associated to the eigenvalue λ and the corresponding eigenvector r. We make use of the remark right after the proof of Lemma 28.

```
In[•]:= greatBound[t_, λ_, r_, n_, Cgreat_, L_] := Module[{lhs, rhs},
  lhs = Abs[r.(P3.(t[[2]] ⊗ {α, β, γ} + {α, β, γ} ⊗ t[[3]]))];
  rhs = (2 * L - Total[t[[1]][[1 ;; 3]]]) / L * Cgreat +
    Max[Map[Abs[fbis[r, t, #[[1]], #[[2]], -#[[1]] - #[[2]]]] &, {{3/2, 0},
       {3/2, -3/2}, {0, 3/2}, {0, -3/2}, {-3/2, 3/2}, {-3/2, 0}}]] / L;
  If[Accuracy[lhs] < 7 || Accuracy[rhs] < 7, Print["Accuracy problem"]];
  Return[lhs ≤ rhs];
 ]
```

Computing the bounded ancestors of templates [0,0,0,a1,a2] with a1 ≠ a2

Finite number of bounded ancestors

This section consists in computing all the ancestors of templates [0,0,0,a1,a2] with a1 ≠ a2 that are bounded by the constants we just computed, and to show that there are just a finite number of them. The first function takes as arguments one template and the list of all computed small and great bounds. It verifies that the template satisfies all the conditions from previous section. CsmallList and CgreatList contain pairs of elements, of the type {n, bound}.

```
In[•]:= validTemplate[template_, CsmallList_, CgreatList_, L_] :=
 Which[
  Not[smallBound[template, duosVP[[7, 1]], duosVP[[7, 2]],
    CsmallList[[1]][[1]], CsmallList[[1]][[2]]]], False,
  Not[smallBound[template, duosVP[[9, 1]], duosVP[[9, 2]],
    CsmallList[[2]][[1]], CsmallList[[2]][[2]]]], False,
  Not[smallBound[template, duosVP[[11, 1]], duosVP[[11, 2]],
    CsmallList[[3]][[1]], CsmallList[[3]][[2]]]], False,
  Not[smallBound[template, duosVP[[12, 1]], duosVP[[12, 2]],
    CsmallList[[4]][[1]], CsmallList[[4]][[2]]]], False,
  Not[greatBound[template, duosVP[[3, 1]], duosVP[[3, 2]],
    CgreatList[[1]][[1]], CgreatList[[1]][[2]], L]], False,
  Not[greatBound[template, duosVP[[4, 1]], duosVP[[4, 2]],
    CgreatList[[2]][[1]], CgreatList[[2]][[2]], L]], False,
  Not[greatBound[template, duosVP[[13, 1]], duosVP[[13, 2]],
    CgreatList[[3]][[1]], CgreatList[[3]][[2]], L]], False,
  True, True
 ]
```

This function takes a list of templates and only keeps the valid ones.

```
In[•]:= valid[templatesList_, CsmallList_, CgreatList_, L_] :=
 valid[templatesList, CsmallList, CgreatList, L] =
  Select[templatesList, validTemplate[#, CsmallList, CgreatList, L] &]
```

Finally, this function is the one computing all the ancestors of templates [0,0,0,a1,a2]. We assume that seenTemplates and toSeeTemplates are two global list variables. The first one is initially set at { } while the second one initially contains the six templates from which we start.

```
In[•]:= computingAncestors[CsmallList_, CgreatList_, L_] := Module[{currentTemplate},
  While[Length[toSeeTemplates] > 0 ,
   currentTemplate = toSeeTemplates[[1]];
   toSeeTemplates = Drop[toSeeTemplates, 1];
   toSeeTemplates = Union[toSeeTemplates, Complement[valid[
       listParents[currentTemplate], CsmallList, CgreatList, L], seenTemplates]];
   AppendTo[seenTemplates, currentTemplate]
  ]
 ]
```

This part of code is the one used to generate all the valid templates which will be stocked in seenTemplates. They will also be stocked in an out file.

```
In[•]:= toSeeTemplates =
          List[List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 0, 1],
           List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 0, 2],
           List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 1, 0],
           List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 1, 2],
           List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 2, 0],
           List[ConstantArray[0, 12], ConstantArray[0, 3], ConstantArray[0, 3], 2, 1]];
        seenTemplates = {};
        L = 15; (* This value can be adapted *)
        computingAncestors[CsmallList, CgreatList, L];
        Export["seenTemplates.m", seenTemplates];
        Print["Number of valid ancestors : ", Length[seenTemplates]]
```

```
Number of valid ancestors : 241544
```

If you don' t want to do this long computation every single execution, you can import the selected templates.

```
In[•]:= seenTemplates = Import["seenTemplates.m"];
```

Conclusion: the factor complexity equals the 2-binomial complexity

We just obtained a finite number of bounded ancestors. We want to conclude by Lemma 13. Then, making use of Proposition 17, if templates [0,0,0,a1,a2] are realizable, either there are realizable by a pair (u,v) of factors of Tribonacci with $\min(|u|,|v|) \leq L$, or one of their ancestors is realizable by a pair (u,v) of factors with $L \leq \min(|u|,|v|) \leq 2L$.
The first function deals with the first case. Since these templates are such that d = 0, D_b = 0, D_e = 0, (u,v) realizes one of them if and only if $\psi(u) = \psi(v)$.
This functions checks that all words of length at most L are non pairwise 2-binomially equivalent.

```
In[•]:= smallWordsNonEquiv[L_] := Module[{i, j, k, words},
          For[i = 1, i ≤ L, i++,
           words = factors[i];
           For[j = 1, j ≤ 2 * i + 1, j++,
            For[k = j + 1, k ≤ 2 * i + 1, k++,
             If[ψ[words[[j]]] == ψ[words[[k]]], Return[False]]]]];
          Return[True];
         ]
```

The following function takes as arguments two lists that are ψ-vectors of some words u and v, and a template. It checks that no pair (u,v) realizes the template. In this case, it returns True. Otherwise, it returns False.

```
In[•]:= verif[lu_, lv_, t_] := Not[MemberQ[
          Map[ #[[1]] - #[[2]] ==
             t[[1]] + P3. (t[[2]] ⊗ #[[1]][[1 ;; 3]] + #[[1]][[1 ;; 3]] ⊗ t[[3]]) &,
           Tuples[{lu, lv}]], False]]
```

Finally, this function verifies that no template from the list seenTemplates is realizable by a pair (u,v) of factors of Tribonacci with L ≤ min(|u|,|v|) ≤ 2 L.
To accelerate the verification, we compute the maximal length of |u| and |v| and we stock all the possible values of ψ[u] and ψ[v].

```
In[•]:= F[L_, seenTemplates_] :=
 Module[{listlistpsi, j, i, fac, currentTemplate, k, lgV, lu, lv, total, Dd},
  (* Maximal length of |u| and |v| *)
  Dd = Max[Map[Abs[Total[#[[1]][[1 ;; 3]]]] &, seenTemplates]];

  (* Generation of lists containing ψ[u] and ψ[v]. *)
  listlistpsi = {};
  For[j = L, j ≤ 2 L + Dd, j++,
    fac = factors[j];
    AppendTo[listlistpsi, List[ Map[ψ[#] &, Select[fac, #[[-1]] == 0 &]], Map[ψ[#] &,
       Select[fac, #[[-1]] == 1 &]], Map[ψ[#] &, Select[fac, #[[-1]] == 2 &]] ]
    ]
   ]

   (* Looking at every template separately *)
   For[j = 1, j ≤ Length[seenTemplates], j++,
    currentTemplate = seenTemplates[[j]];
    total = Total[currentTemplate[[1]][[1 ;; 3]]];
    (* Two cases depending on if |u|<|v| or not. *)
    If[total ≥ 0,
     For[i = 1, i ≤ L + 1, i++,
      If[verif[listlistpsi[[i + total, currentTemplate[[4]] + 1]], listlistpsi[[i,
          currentTemplate[[5]] + 1]], currentTemplate] == False, Return[False]]
     ],
     For[i = 1, i ≤ L + 1, i++,
      If[verif[listlistpsi[[i, currentTemplate[[4]] + 1]], listlistpsi[[i - total,
          currentTemplate[[5]] + 1]], currentTemplate] == False, Return[False]]
     ]
    ]
   ];
  Return[True];
 ]
```

Finally, these instructions check that Proposition 17 can be applied, as explained earlier. The value of L can change, but has to be equal to the one chosen in the verification of the greater bounds.

In[•]:=
```
L = 15;
smallWordsNonEquiv[L]
F[L, seenTemplates]
```

Out[•]= True

Out[•]= True