>

This is a translation into Maple of (a previous version of) the Python program 21AAIC.py.

```
> get_row :=proc(i, w)
   local j;
   [ seq(max(0, 2·i − 1 − j), j = 0 ..w − 1) ]
   end proc
```

$get\_row := \textbf{proc}(i, w)$ **(1)**

```
    local j; [seq(max(0, 2*i − 1 − j), j = 0 ..w − 1)]
end proc
```

>

Warning: when r = 0, all_subfrequencies returns a singleton, which is not so good to iterate over in a succeding statement "for j in all_subfrequencies(c,0) ... "

```
> all_subfrequencies := proc(c, r)
     local s, g0, gs, j;
   if c = 1 then
     s := seq([g0], g0 = 0 ..r);
     return s;
   else
     s := NULL;
     for g0 from 0 to r − 1 do
      for gs in all_subfrequencies(c − 1, max(0, r − g0)) do
     s := s, [g0, op(gs)];
      od;
     od;
     s := s, [r, seq(0, j = 2 ..c)];
     return s;
   fi;
   end proc;
```

$all\_subfrequencies := \textbf{proc}(c, r)$ **(2)**

```
    local s, g0, gs, j;
    if c = 1 then
        s := seq([g0], g0 = 0 ..r); return s
    else
        s := NULL;
        for g0 from 0 to r − 1 do
            for gs in all_subfrequencies(c − 1, max(0, r − g0)) do
                s := s, [g0, op(gs)]
            end do
```

```
        end do;
        s := s, [r, seq(0, j = 2 ..c)];
        return s
    end if
end proc
```

```
>
> all_frequencies := proc(i, ks)
    local j, w, k, s, gs, tail_k;
    w := nops(ks);
    k := add(ks[j], j = 1 ..w);
   if i > w/2 then
    return all_subfrequencies(w, k)
   else
    tail_k := add(ks[j], j = w - 2·i + 2 ..w);
    if tail_k = 0 then
     s := [seq(0, j = 1 ..2·i - 1), seq(ks[w - 2·i + 2 - j], j = 1 ..w - 2·i + 1)];
     else
     s := NULL;
     for gs in all_subfrequencies(2·i - 1, tail_k) do
       s := s, [op(gs), seq(ks[w - 2·i + 2 - j], j = 1 ..w - 2·i + 1)]
    od;
    fi;
    return s;
   fi;
  end proc;
```

```
all_frequencies := proc(i, ks)                                                    (3)
    local j, w, k, s, gs, tail_k;
    w := nops(ks);
    k := add(ks[j], j = 1 ..w);
    if 1 / 2 * w < i then
        return all_subfrequencies(w, k)
    else
        tail_k := add(ks[j], j = w - 2 * i + 2 ..w);
        if tail_k = 0 then
            s := [seq(0, j = 1 ..2 * i - 1), seq(ks[w - 2 * i + 2 - j], j = 1 ..w - 2 * i
            + 1)]
        else
            s := NULL;
            for gs in all_subfrequencies(2 * i - 1, tail_k) do
                s := s, [op(gs), seq(ks[w - 2 * i + 2 - j], j = 1 ..w - 2 * i + 1)]
            end do
```

```
        end if;
        return s
    end if
end proc
```

```
> row_value := proc(row, fs)
    local w, j;
     w := nops(row);
    add(row[j]·fs[j], j = 1 ..w)
    end proc;
```

$row\_value := \mathbf{proc}(row, fs)$ **(4)**

```
    local  w, j;  w := nops(row);  add(row[j]*fs[j], j = 1 ..w)
end proc
```

```
>
```

Warning: Maple's list indices start at 1 while Python indices start at 0, so all indices in fs, ms1, ms are 1 unit larger in the Maple code than in the corresponding Python code.

In Maple we return "-1" in place of "None" in Python for failed potential frequencies fs.

```
> filter_frequencies := proc(fs, ms1, k)
     local j, w, ms, f, m, m0;
    w := nops(fs);
    ms := [seq(0, j = 1 ..w)];
    ms[1] := fs[1];
    for j from 2 to w do
      f := fs[j];
      m := ms1[j − 1];
     m0 := ms[j − 1];
     if m0 > m then
      m := m0;
     fi;
    m := m + f;
    if m > k then
     return -1;
    fi;
    ms[j] := m;
    od;
    return ms;
    end proc;
```

$filter\_frequencies := \mathbf{proc}(fs, ms1, k)$ **(5)**

```
    local  j, w, ms, f, m, m0;
```

```
    w := nops(fs);
    ms := [seq(0, j = 1..w)];
    ms[1] := fs[1];
    for j from 2 to w do
        f := fs[j];
        m := ms1[j - 1];
        m0 := ms[j - 1];
        if m < m0 then m := m0 end if;
        m := f + m;
        if k < m then return -1 end if;
        ms[j] := m
    end do;
    return ms
end proc
```

```
>
```

In rows0 and rows1 the components are lists of length 2 rather than ordered pairs as in the Python code. As remarked above, we use a return of -1 rather than "None" to signal a failed potential list of frequencies. We print the result as a power series in q rather than a list.

```
>
```

Input:
  highest weight is a list [k1,k2,...,kw] of length w,
  N is the highest degree that we count partitions of.

Output: the generating function of the counts of the number of admissible colored partitions.

```
> colored_partitions := proc(highest_weight, N)
   local w, k, i, j, a, rows0, rows1, values1, r2, min_next_row, afs, nv, value0, ms0,
       ms1, fs1, ms, value1, result;
  printf("%s%a \n", "highest_weight = ", highest_weight);
   w := nops(highest_weight);
   k := add(highest_weight[j], j = 1..w);
  printf("%s%a %s%a", "k = ", k, "w = ", w);
   i := 1;
   result := [seq(0, j = 1..N)];
   ms1 := [seq(add(highest_weight[w + 1 - a], a = 1..j), j = 1..w)];
   rows0 := [[0, ms1]];
   do
    rows1 := [];
    values1 := get_row(i, w);
```

```
  r2 := get_row(i + 1, w);
  min_next_row := r2[nops(r2)];
  afs := [all_frequencies(i, highest_weight)];
  nv := nops(rows0);
  for j from 1 to nv do
   value0 := rows0[j, 1];
   ms0 := rows0[j, 2];
   for fs1 in afs do
    ms := filter_frequencies(fs1, ms0, k);
    if ms ≠ -1 then
     value1 := row_value(values1, fs1) + value0;
     if value1 ≤ N then
      if value1 > value0 then
       result[value1] := 1 + result[value1];
      fi;
      if value1 ≤ N - min_next_row then
       rows1 := [op(rows1), [value1, ms]];
      fi;
     fi;
    fi;
   od;
  od;
 if values1[nops(values1)] ≥ N - 1 then
   return sort(1 + add(result[j]·q^j, j = 1 ..N), q, ascending);
 fi;
 i := i + 1;
 rows0 := rows1;
 od;
 end proc;
colored_partitions := proc(highest_weight, N)                                  (6)
  local w, k, i, j, a, rows0, rows1, values1, r2, min_next_row, afs, nv, value0,
  ms0, ms1, fs1, ms, value1, result;
  printf("%s%a \n", "highest_weight = ", highest_weight);
  w := nops(highest_weight);
  k := add(highest_weight[j], j = 1 ..w);
  printf("%s%a %s%a", "k = ", k, "w = ", w);
  i := 1;
  result := [seq(0, j = 1 ..N)];
  ms1 := [seq(add(highest_weight[w + 1 - a], a = 1 ..j), j = 1 ..w)];
  rows0 := [[0, ms1]];
  do
      rows1 := [ ];
      values1 := get_row(i, w);
      r2 := get_row(i + 1, w);
```

```
        min_next_row := r2[nops(r2)];
        afs := [all_frequencies(i, highest_weight)];
        nv := nops(rows0);
        for j to nv do
            value0 := rows0[j, 1];
            ms0 := rows0[j, 2];
            for fs1 in afs do
                ms := filter_frequencies(fs1, ms0, k);
                if ms<> -1 then
                    value1 := row_value(values1, fs1) + value0;
                    if value1 <= N then
                        if value0 < value1 then
                            result[value1] := 1 + result[value1]
                        end if;
                        if value1 <= N - min_next_row then
                            rows1 := [op(rows1), [value1, ms]]
                        end if
                    end if
                end if
            end do
        end do;
        if N - 1 <= values1[nops(values1)] then
            return sort(1 + add(result[j]*q^j, j = 1 ..N), q, ascending)
        end if;
        i := i + 1;
        rows0 := rows1
    end do
end proc
>
> colored_partitions([0, 0, 0, 1], 31)
highest_weight = [0, 0, 0, 1]
k = 1 w = 4
```

$$1+q+q^{2}+2\,q^{3}+3\,q^{4}+3\,q^{5}+5\,q^{6}+6\,q^{7}+8\,q^{8}+10\,q^{9}+13\,q^{10}+16\,q^{11}+21\,q^{12}+25\,q^{13}+31\,q^{14}+38\,q^{15}+47\,q^{16}+56\,q^{17}+69\,q^{18}+82\,q^{19}+99\,q^{20}+118\,q^{21}+141\,q^{22}+166\,q^{23}+199\,q^{24}+233\,q^{25}+275\,q^{26}+322\,q^{27}+379\,q^{28}+440\,q^{29}+516\,q^{30}+598\,q^{31} \tag{7}$$

```
>
> colored_partitions([0, 0, 1], 31)
```

highest_weight = [0, 0, 1]
k = 1 w = 3

$$1+q+q^{2}+2q^{3}+2q^{4}+3q^{5}+4q^{6}+5q^{7}+6q^{8}+8q^{9}+10q^{10}+12q^{11} +15q^{12}+18q^{13}+22q^{14}+27q^{15}+32q^{16}+38q^{17}+46q^{18}+54q^{19} +64q^{20}+76q^{21}+89q^{22}+104q^{23}+122q^{24}+142q^{25}+165q^{26}+192q^{27} +222q^{28}+256q^{29}+296q^{30}+340q^{31} \tag{8}$$

>

>