Project 1: Scientific Computing

Gianmaria Lucca
MAT: 241440

In this brief report we analyze and explain the functions used in **project1.py**.

- **number2base_rep(n, b)**: this function takes as input the integer n and the base b and returns the representation $(n)_b$. The function computes iteratively the remainder and the integer division between the value of n and the base b, changing the value of n as $n \,//\, b$.

- **admissible(n, b)**: the function determines if the number n is *b–admissible*, so if $(n)_b$, seen as a string, has no neighbouring equivalent substrings. The implementation is based on this procedure:
  - We know that the maximum length of 2 neighbouring substrings is $upper = len\left((n)_b\right) \,//\, 2$, so we create, for $i \in \{1, 2..., upper\}$, a list of all the possible substrings of length i.
  - Given the list, we check if 2 contiguous substrings are equivalent: if so n is **not** b–admissible.
  - Otherwise, if $\forall i \in \{1, 2..., upper\}$ the algorithm hasn't found any pair of equivalent and contiguous substrings, then the number n is b–admissible.

- **count_admissible(b, start, end)**: function that takes the base b and returns the number of b–admissible integers between *start* and *end*. This function relies on **admissible(n, b)**.

- **count_admissible_width(b, width)**: returns the number of b–admissible numbers whose representation in base b has exactly length *width*. The implementation is based on this procedure:
  - We know that, for the first digit of $(n)_b$, we have $b-1$ possible choices (since 0 isn't acceptable): so, there are $(b-1)\, b^{width-1}$ possible combinations of digits.
  - Using the recursive function **All_Width_Length_Rec**, we append to a list all the possible strings with length *width*.
  - Given the list, we check iteratively if a given entry is b–admissible or not.

- **largest_multi_admissible(L, start, end)**: given a list L of bases b, this function determines the biggest b–admissible number n such that $start \leq n < end$, $\forall b \in L$.
  The implementation is based on two nested for loops (one for the iteration of the numbers and the other for the bases in the list L).

We report now the results obtained for the tests, changing the integer k, of the following functions:

- **count_admissible(5, 10**k, 10**(k+1))**:
  k = 1, The value is 60, Elapsed time = 3.0780e-04
  k = 2, The value is 370, Elapsed time = 4.9226e-03
  k = 3, The value is 2715, Elapsed time = 6.0163e-02
  k = 4, The value is 17238, Elapsed time = 7.0859e-01
  k = 5, The value is 111465, Elapsed time = 4.0186e+00
  k = 6, The value is 776004, Elapsed time = 4.3917e+01
  k = 7, The value is 4829962, Elapsed time = 4.7495e+02
  We can observe that the CPU time depends like powers of 10, in particular it depends like $C * 10^k$, for some constant C.

- **count_admissible_width(3, k)**:
  k = 1, The value is 2, Elapsed time = 5.0068e-06
  k = 2, The value is 4, Elapsed time = 1.0729e-05
  k = 3, The value is 8, Elapsed time = 2.5988e-05
  k = 4, The value is 12, Elapsed time = 9.5844e-05
  k = 5, The value is 20, Elapsed time = 3.5238e-04
  k = 6, The value is 28, Elapsed time = 1.1303e-03
  k = 7, The value is 40, Elapsed time = 5.6028e-03
  k = 8, The value is 52, Elapsed time = 1.5219e-02
  k = 9, The value is 72, Elapsed time = 4.8984e-02
  k = 10, The value is 96, Elapsed time = 1.5705e-01
  k = 11, The value is 136, Elapsed time = 4.4520e-01

  Here we can observe that it seems that the CPU time depends like powers of 10, but the time is increased by a factor 10 **for every two iterations** of k.

- **largest_multi_admissible([3, 5, 7, 10], 1, 10**k)**:
  k = 1, The value is 7, Elapsed time = 1.5950e-04
  k = 2, The value is 96, Elapsed time = 1.6518e-03
  k = 3, The value is 923, Elapsed time = 5.6443e-03
  k = 4, The value is 8165, Elapsed time = 6.6020e-02
  k = 5, The value is 70921, Elapsed time = 4.3481e-01
  k = 6, The value is 657984, Elapsed time = 4.7145e+00
  k = 7, The value is 8428271, Elapsed time = 5.4321e+01

  As in the first test, here it seems the CPU time depends like $C * 10^k$.