Interval Partitioning Report

Mark Dear, Stelios Papaoikonomou, Radek Niemczyk, Rasmus Løbner Christensen

October 2, 2015

Results

Our implementation produces the expected results on all input–output file pairs and right scores. Just in a different order.

Implementation details

Our implementation takes the input files and creates two hashmaps one for the proteins and one for the costs (CostsParser.java and ProteinParser.java). Then our OPT(int i, int j) method follows the pseudocode to create the table of steps. It will get called itself recursively on the memos table to fill it, but because of our if (result == null) method, it will only be called when it wasn't ever called for the element of memos we are currenly in OPT for. So we have max n*m calls for all the table given that the hashmap has constant complexity. Inside of the iterations we usually call twice Math.max so overall n*m*2*complexityOf(Math.max) and if we consider the last one constant we have O(m*n) time.

Our getMatching() method which returns the strings with the best alignment has a while loop with n + m -1 iterations. This happens because in order to reach the (0,0) element we have to reach the right element of it or the bottom one which will take us out of the loop. We also consider everything inside of constant time like append. Afterwards we are located on one boarder of the table either top row or left column so we iterate (max n or m times and minimum of 1) with the two if statements to finally reach the (0,0) element. Again we encounter append and some loops with append with max limit of iterations n-1 and because the append has a complexity of O(1) (we used StreamBuilder instead of stream so O(1) insteaad of O(n)) the final complexity of our computations is O(m*n + (n+m))= O(n^2). Lastly we print the output to the console.