Code Jam 2011 - World Finals

Analysis: Runs

On a contest made up of unconventional problems, this one stands out as being a little more "normal" than the rest. That doesn't make it easy though! The main techniques here are good old-fashioned dynamic programming and counting, but you need to be very good at them to stand a chance.

The most important observation is that you cannot afford to build strings from left to right. With 450,000 characters, there is just far too much state to keep track of. The correct approach turns out to be placing all of one character type, then all of the next, and so on. Towards that end, let's define N_c to be the number of characters of type c, and define $X_{c,r}$ to be the number of ways of arranging all characters of type 1, 2, ..., c so that there are exactly r runs. We will compute the X values iteratively using dynamic programming.

Consider a string S using only the first c - 1 character types, and having r_0 runs. Note that S has exactly $M = N_1 + N_2 + ... + N_{c-1}$ characters altogether. Additionally, it has a few different types of "character boundaries":

- There is the boundary before the first character and the boundary after the last character. If we add a run of type-c characters in either one of these locations, the total number of runs will increase by 1.
- There are $r_0 - 1$ boundaries between distinct characters. If we add a run of type-c characters in any one of these locations, the total number of runs will increase by 1.
- There are $M - r_0$ boundaries between identical characters. If we add a run of type-c characters in any one of these locations, the total number of runs will increase by 2.

So let's suppose we add x runs of type-c characters in any one of the $r_0 + 1$ boundaries from the first two groups, and we add y runs of type-c characters in any of the $M - r_0$ boundaries from the the third group. There are exactly ($r_0 + 1$ choose x) * ($M - r_0$ choose y) ways of choosing these locations, and we will end up with $r_0 + x + 2y$ runs this way. Finally, we need to divide up the N_c characters of type c into these x + y runs. This can be done in exactly ($N_c - 1$ choose x + y - 1) ways. To see why, imagine placing all the runs together. Then we need to choose x + y - 1 run boundaries from $N_c - 1$ possible locations. See here for more information.

Therefore, the number of ways of adding all N_c type-c characters to S so as to get a string with exactly r runs can be calculated as follows:

- Loop over all non-negative integers x, y such that $r_0 + x + 2y = r$.
- Add ($r_0 + 1$ choose x) * ($M - r_0$ choose y) * ($N_c - 1$ choose x + y - 1) to a running total.
- After looping over all x, y, this running total will contain the answer we want.

Note that the answer here depends only on r_0. Therefore, the total contribution from all strings with r_0 runs is exactly $X_{c-1,r0}$ times this quantity. Iterating over all r_0 gives us the recurrence we need for X!

This method is actually quite fast. We can use O(450,000 * 100) time to pre-compute all the choose values. Everything else runs in $O(26 * 100^3)$ time.

By the way, you probably need to calculate the choose values iteratively rather than recursively. 450,000 recursive calls will cause most programs to run out of stack space and crash! Here is a

pseudo-code with a sample implementation (modulo operations removed for clarity):

```
def CountTransitions(M, Nc, r0, r):
  # Special case: If adding the first batch of characters the
  # only possible result is to have one run, and there is only
  # one way to achieve that.
  if r0 == 0:
    return r == 1 ? 1 : 0
  result = 0
  dr = r - r0
  for (y = 0; r0 + 2 * y <= r; ++y):
    x = r - (r0 + 2 * y)
    nways_select_x = Choose(r0 + 1, x)
    nways_select_y = Choose(M - r0, y)
    nways_split = Choose(Nc - 1, x + y - 1)
    result += nways_select_x * nways_select_y * nways_split
  return result


def Solve(freq, runs_goal):
  runs_count = [0 for i in range(0, runs_goal + 1)]
  runs_count[0] = 1
  M = 0
  for i, Nc in enumerate(freq):
    if Nc > 0:
      old_runs_count = list(runs_count)
      runs_count = [0 for i in range(0, runs_goal + 1)]
      for (r0 = 0; r0 <= runs_goal; ++r0):
        for (int r = r0 + 1; r <= runs_goal; ++r):
          nways = CountTransitions(M, Nc, r0, r)
          runs_count[r] += nways * old_runs_count[r0]
      M += Nc
  return runs_count[runs_goal]
```